Closing the Loop: Ratcheting Software Architecture with Egress Barriers and Content-Addressing

Setting the Stage: Context for the Curious Book Reader

Context for the Curious Book Reader

Software engineering in the age of automation often suffers from a silent drift: tools evolve faster than our mental models, leaving behind stale documentation, unverified assumptions, and un-gated data flows. This essay explores an important transition in architectural evolution: moving from fragile, stateful scripts to deterministic, receipt-driven pipelines. By treating the development process as a series of physical barriers rather than ceremonial prompts, we can build robust systems where security and reproducibility are verified by code rather than trusted by habit.

Technical Journal Entry Begins

🔗 Verified Pipulate Commits:

- 7c20c21f (raw)
- 07527a67 (raw)
- 82ce0b68 (raw)
- 5f8d9392 (raw)
- 302125df (raw)
- 5c9213b7 (raw)
- 76ea0cc8 (raw)
- e60caceb (raw)
- 903a8e15 (raw)
- 8cbe7ca8 (raw)
- 8847c5cf (raw)
- aee3976f (raw)
- aa3f0781 (raw)
- 40f9930f (raw)
- 7982cebe (raw)
- 1f2fe3af (raw)
- cc94d355 (raw)
- 9d997703 (raw)
- b6979574 (raw)

- 85f7716b (raw)
- 5b4f8081 (raw)
- 741b62e1 (raw)

TL;DR: Three fence branches witnessed, content addressing witnessed in **both** directions, one real defect found by traceback, and three files caught telling a newcomer a lie that had already been fixed in code. The ride started with a trail system nobody had ever compiled and ends with a sealed, verified, content-addressed cartridge format plus an egress barrier that has been observed passing, refusing, and declining. Below: verify, bank, dangling, seed, close, notarize.

MikeLev.in: We need different install paths. I believe it's in there already. I need to be able to plan different "walks".
One set of URLs will kick-off a "walking the URLs" which may have a "warm up" step where usually OAuth authentication is carefully acquired before a URL is attempted to be visited. The idea is that by the time a URL is pulled up any authentication issues are already dealt with so there is no conditional branching on the pulling up of the URL. If done wrong it will "bounce" you back out to authentication and then back in once it's acquired through a redirect URL and that's not controlled enough for Flight Deck Recorder purposes. Each walk is planned out so it plays out the same on every future replay, besides the different authentication different users will have. That of course will potentially make the same walk different for different people, yet still we avoid the sort of redirects and routing this would imply. We get that out of the way beforehand and use explicit URLs with the inferred necessary IDs and other identifiers already in the right places in the URL, be they paths or querystring arrangements. Is all of this making sense?
That's all just to lay out the system and the vocabulary and the freedom that exists in making these URLs. Because things will be funneled through the curl | bash pattern we have to be careful to only allow it to work against things that are cryptographically sealed with hashes so shenanigans can't switch out what "walk" is being run. Walks have to come probably from private repos with proper ssh key possessing, probably a deploy key using the same ROT13 trick that's already in the system. I hesitate to do a multiple key system because of the exploding complexity that could incur. And so we will start with... with what?
Okay, here's the issue. Walks are going to always be ad hoc and experimental. Rarely, except for a handful of templates, will we actually want them in the Pipulate repo. Most of these walk-scripts belong in repos that are "filled into" Pipulate through the Negative Space created by the .gitignore file and this is where the prospect of a second ssh key comes into the picture. There's a chance I can use the same deploy key for both to keep it simple. Hmmm, but I think deploy keys are tied to repos. Okay, a sane key management system for people who really don't know and probably never will know git may have to be part of this picture. Let's think it through. It really starts with the unified Markdown-centric directory structures we're constitutionalizing in the system.

```
def _figurate_workspace_tree():
    """Render the Notebooks workspace as it ACTUALLY sits on disk: flat.
    THE NESTED VERSION WAS THE ONLY AUTHORITY THAT SHIPPED, AND IT WAS WRONG.
    update_agents_md_in_place() splices this art into AGENTS.md on every
    compile, so it is what every external agent tool reads -- while flake.nix
    argued for flat siblings and `ls Notebooks/` showed nine flat directories
    and no Corporate/, Personal/, or Workshop/ at all. Three authorities, one
    winner, and the winner disagreed with the disk. Fixed at the GENERATOR so
```

```
    the output cannot drift back.
    Three tiers, zero nesting: canon arrives by the flake's copy-if-absent
    loop, personal is gitignored, and Shared/ is the single deliberate
    outbound surface with one folder per person. Authored with no .strip() so
    the leading newline is preserved (the honeybot_pipeline convention), which
    the FIGURATE_LEDGER seal must match. No color-bit or angle-tag markers, so
    _expand_color_bits_ai leaves it untouched and ai_art == art.
    """
    art = r"""
   Notebooks/  — the JupyterLab root (NOT Pipulate's own root)
    │           FLAT siblings. Nothing nests. Nothing to get wrong.
    │
    ├── Advanced_Notebooks/     canon · flake-delivered, copy-if-absent
    ├── Educational_Notebooks/  canon · your edits survive, updates do not arrive
    ├── imports/                canon · the code-behind "sauce" modules
    │
    ├── Playground/             personal · gitignored · your own git repo goes here
    ├── Client_Work/            personal · gitignored · never leaves this machine
    ├── Deliverables/           personal · gitignored
    │
    └── Shared/                 the ONE folder for handing work to a teammate
        ├── alice/              one folder per person; you write ONLY your own
        └── bob/                single-writer partitions = zero merge conflicts
"""
    ai_art = _expand_color_bits_ai(art)
    human_art = _expand_color_bits_human(art)
    human = Panel(human_art, title="🗂️ Notebooks Workspace — canon · personal · Shared",
border_style="white")
    return human, ai_art
```

This matches truth which I did recently to make sure was true, but I do believe we're working towards a more canonical layout, so there is a theory versus practice thing going on here, and we want to move towards the most compelling language for adoption. We plan to make the reality come more in line with the ideal and no current folder-names are sacred and there is no reverse compatibility criteria. Playground, Client_Work and Deliverables does not feel right. Everything in those 3 locations I think would be under Personal in a structure like this:

```
    art = r"""
   Notebooks/  — the JupyterLab root (NOT Pipulate's own root)
    │           every level advertises its own AGENTS.md + OKF index.md
    │
    ├── Corporate/   read-only canon · auto-pulled · git wins on collision
    │   ├── AGENTS.md
    │   ├── .agents/skills/
    │   └── apps/          org plugins ride in — no core commit needed
    │
    ├── Personal/    your sandbox · gitignored · vibe-code freely
    │   ├── AGENTS.md
    │   └── Playground/    NOTHING here is ever shared
    │
```

```
│
└── Shared/      outbound exchange · one folder per name
    ├── alice/     you write ONLY your own folder;
    └── bob/       single-writer partitions = zero merge conflicts
"""
```

And here is the big picture of the 3 standards we're blending.

The three standards, one at a time

1. agents.md — the simplest. One file, no schema, nearest-ancestor wins:

```
repo/
├── AGENTS.md          # freeform markdown: setup, test commands, conventions, PR rules
└── subproject/
    └── AGENTS.md      # optional override; closest file to the working directory wins
```

That's the whole spec. It deliberately has no required fields — it's a README addressed to agents instead of humans, and its main achievement was collapsing CLAUDE.md / CURSOR.md / .cursorrules / GEMINI.md into one filename everyone's tooling checks.

2. Agent Skills (agentskills.io) — a skill is a *folder* whose front door is SKILL.md with YAML frontmatter (name and description required), practicing progressive disclosure: frontmatter always loaded, body loaded on trigger, linked resources loaded on demand:

```
.agents/skills/              # (Claude Code uses .claude/skills/; the shape is identical)
└── hello_workflow/
    ├── SKILL.md             # --- / name: / description: / --- + instructions body
    ├── scripts/             # optional executables the skill may run
    ├── references/          # optional docs pulled in only when needed
    └── assets/              # optional templates and files
```

You already have this. Notebooks/.agents/skills/hello_workflow/SKILL.md, gsc_readonly, roles — it's in your manifest.

3. OKF — what the v0.1 specification actually fixes is a folder layout, markdown files, YAML frontmatter, reserved filenames, and a single required field: type. A bundle is a directory of markdown files, each carrying a short YAML block — type, title, description — linking to its neighbors; add an index.md that lists the files so an agent can see what's there before opening everything, and that's the format. The spec fits on a single page.

```
okf-bundle/
├── index.md               # reserved: the table of contents an agent reads first
├── some-concept.md        # --- / type: Article / title: / description: / tags: / ---
├── another-concept.md     # path IS the identifier; markdown links ARE the graph
└── runbooks/
    └── deploy.md
```

Superimposed on Pipulate

Here's the combined view — everything marked NEW is a signpost; everything else already

exists and stays exactly where it is:

```
pipulate/
├── AGENTS.md                    # NEW — the symlink-in-spirit; ~40 lines pointing at executable truth
├── foo_files.py                 # = AGENTS.md at scale: the router IS the agent operating manual
├── prompt_foo.py                # = the AGENTS.md *compiler*; emits the payload + foo.zip
├── foo.zip                      # = a portable AGENTS-class bundle (gets the YAML topper below)
├── cli.py                       # = the `allowed-tools` surface: mcp-discover / call
├── apply.py                     # = "PR instructions" made executable (SEARCH/REPLACE actuator)
├── flake.nix                    # = "Dev environment setup" made executable (nix develop)
├── Notebooks/.agents/skills/    # = Agent Skills, already standard-shaped
│   ├── hello_workflow/SKILL.md
│   ├── gsc_readonly/SKILL.md
│   └── roles/SKILL.md
└── (~/repos/trimnoir/_posts/)   # = an OKF bundle avant la lettre:
    ├── *.md                     #   markdown + YAML frontmatter, one concept per file
    ├── _context/*.json          #   your holographic shards ≈ OKF's index/graph layer
    └── (llms.txt, hub pages)    #   generate_llms_txt.py already plays index.md's role
```

Does all this make sense? Can you see where we're going? Is there a best arrangement being inferred here? Something logical that doesn't require a refactor that we can inch towards with baby-step chisel-strikes?

Opus 5 Max Thinks: Thought for 4m 44s

Thought for 4m 44s

The user is working through a design challenge for Pipulate around creating deterministic browser walks that can be safely distributed via curl piped to bash, with cryptographic sealing for security and support for private repos with SSH keys. They're also trying to reconcile how the Notebooks directory is currently structured (flat) with their idealized organization (Corporate/Personal/Shared) and align it with their three standards: AGENTS.md, Agent Skills, and OKF. The context includes their constitution router in foo_files.py and their Nix configuration, which are likely relevant to how this architecture fits together.

Looking at the .gitignore patterns, I can see they're using a selective approach where some Notebooks directories are entirely ignored (Client_Work, Deliverables, Shared, Playground, Educational_Notebooks) while others like Advanced_Notebooks have only specific files ignored, suggesting a distinction between generated/user-facing content and canonical/template content that gets copied from assets on first run. bis in .gitignore, but the Paintbox shows several files underNotebooks/that aren't ignored — likeNotebooks/**init**.py, Notebooks/imports/core_sauce.py, and Notebooks/Advanced_Notebooks/**init**.py— which suggests the Paintbox is capturing the actual generated state rather than just the source template. , remotes, scripts, and tools directories are all in the Paintbox. So the tracked files in Notebooks are just those three SKILL.md files under.agents/skills/, while everything else like Notebooks/imports/**init**.pyandcore_sauce.py` exists on disk but stays gitignored. The Paintbox is listing what's actually committed, not what prompt_foo found lying around.

Now I'm wondering whether the Paintbox parser treats commented lines as claimed — that would explain why core_sauce.py shows up in Chapter XVI of AI_PHOOEY_CHOP even

though it's not in the tracked files.
Looking at the paintbox update logic, it strips leading # characters before processing, so yes, commented-out file paths do count as claimed. That means I can't use Paintbox absence to determine git tracking status. The verifiable facts are what .gitignore explicitly lists: Notebooks/Client_Work/, Notebooks/Deliverables/, and Notebooks/Shared/.
The flake setup creates Notebooks/Shared and Notebooks/Playground/ directories with their README and WELCOME files respectively, while other notebooks come from the copy-if-absent logic in the run script. But here's the critical issue: the .#quiet shell skips the run script entirely—its shellHook only runs the Python setup, not the full initialization.
This means directory creation is split between two lanes: miscSetupLogic handles Shared/ (which runs in all three shells), while the run script handles the Playground welcome file and other notebook copies (which only run in the default and dev shells). The Mother Cat Kata already demonstrates this "walk" pattern with trail YAML files as data and Python scripts as the loader/validator, each stop requiring seven mandatory keys. The gap is that the trail system exists but hasn't been integrated into the compile yet. The user's walk-distribution question is really about distribution and key management rather than the walk format itself. The url_env pattern is the key insight here—each trail declares its own environment variables with no guessing, and SETTLE acts as the human-only auth gate, so the warm-up step where OAuth is acquired before attempting a URL maps directly to the weblogin command warming the apex.
The system already has the session persistence pattern in place through the flake alias, the weblogin script, and the persistent authentication sigil in the prompt configuration. The OAuth PKCE flow is handled by the wallet check and connector scripts. For the curl | bash distribution question, the infrastructure is partially solved—there's an installer script with cryptographic sealing using a magic cookie and ROT13 key, plus the MCK launcher pattern that pipes directly to bash.
The real breakthrough is recognizing that the cartridge format itself is already the byte-reproducible, cryptographically sealed container needed for walks—it's a canonical ZIP with a fixed epoch and SHA-256 manifest that can be verified using only stdlib tools.
The cartridge currently holds three fixed members (payload, prompt, manifest), but to ship a sealed walk, I'd swap the payload for a trail YAML instead of compiled context. The catch is that verify_context_cartridge hardcodes both the member tuple and the payload-to-prompt derivation logic, so I'd need to generalize that verification function to handle different payload types.
Now looking at the SSH key setup: install.sh delivers a ROT13-encoded key file, miscSetupLogic decodes it and writes the SSH config to point at it, and gitUpdateLogic switches to SSH remotes if the key exists. There's something critical here that looks like a potential issue.
The decoded key gets written to ~/.ssh/id_rsa — the default identity file — with a GitHub host entry that has no per-repo scoping. The code checks if the file already exists before writing, so on a machine where the user has their own personal key, the deploy key silently never installs, and the remote URL switch to SSH would then use the user's existing personal key instead. That's a hidden divergence in behavior. Plus, GitHub deploy keys are repo-scoped and can only work with a single repository, so there's a fundamental mismatch between what the user might expect and what the setup actually supports.
The real issue is that you can't reuse one deploy key across multiple repos — GitHub rejects it unless it's a user-level key. So the actual options are either a machine user account (one key for multiple repos but requires a paid seat) or per-repo deploy keys with separate host aliases in

the SSH config, which is the standard approach and scales cleanly without needing extra privileges.
But there's a better path: ship the walks as sealed packages over HTTPS with hash verification instead of giving the walker git access at all. This matches the distribution model the user already described and eliminates key management entirely for the consumer. The author needs git access to write the walk, but the walker only needs to fetch and verify it — those are two separate concerns that shouldn't be conflated just because they currently run in the same repo.
The derived-path rule applies here too: the IdentityFile path should be determined by the repo or host alias, not chosen arbitrarily. Now I'm thinking about the directory structure question — what's the actual invariant that separates canon, personal, and other tiers? Looking at the governing properties: canon is written by the flake and never updates, personal is written by the user, and there's something about what gets leaked or shared that distinguishes them.
But walks introduce a fourth category — something written by someone else and sealed off, read-only from upstream. That's not quite canon since it doesn't come from the flake, not personal since it's inbound, not shared since it's not outbound. Actually, wait — a walk fetched from a private repo behaves exactly like canon: read-only, arrives from upstream, git wins on collisions. The user's "Corporate" tier in their ideal structure is precisely this pattern, and the walks repo already fills that slot.
So the private walks repo IS the Corporate tier. The structure would have Corporate as a gitignored directory cloned from that private repo, containing the walks alongside org plugins and an index file. But there's a friction point: the current .agents/skills/ directory sits at the Notebooks root and is already tracked in version control.
Actually, the nearest-ancestor rule for AGENTS.md files solves this elegantly — I don't need to move anything. The root Notebooks/AGENTS.md serves as the baseline,
while Notebooks/Corporate/AGENTS.md overrides within that scope. Skills discovered
at Notebooks/.agents/skills/ belong to Pipulate itself,
while Notebooks/Corporate/.agents/skills/ would hold org-specific ones. The nearest ancestor wins, which is exactly how the standard is meant to work.
Now I'm thinking about what minimal change gets me closer to this without disrupting the current tracked structure.
The constitution has some hard constraints I need to respect: don't trust a selector until I've created a second candidate, don't treat a boundary as real until something actually crosses it, skip ceremony but never barriers, follow the derived-path rule, don't publish something I haven't tested myself, and understand the three-tier distinction between aliases, functions, and packages.
When probes keep hitting the same boundary without progress, that's the signal to generate something new. But the actual first move isn't renaming directories—it's a probe that reveals the real state of two critical problems: which lane creates which directory (the split-authority issue), and whether the trail system's URL mechanism can actually express a fully-explicit pre-authenticated URL (it can, but nobody's compiled walk.py to verify it). The single-candidate blindness rule hits the walks question directly too—there's only one trail source right now, so any walk resolution logic stays untestable until I create a second one.
For the response shape, I need to follow the contract: probes as a ready-to-paste command block, next context as adhoc.txt lines with probe echoes prefixed by ! , patches as search/replace or a note that none are needed, the prompt in its own fenced block, and external deliverables. I also need to confirm "Context verified" or flag what's missing—and critically, I'm missing scripts/mother_cat.py and scripts/walk.py, which are essential for understanding the walk loading mechanism. I shouldn't propose patches to the trail system itself—that's not my

place until it's actually been compiled and tested. But I can engage with the design reasoning the user is asking for, examining whether the architecture makes sense and where it's heading. That's the real conversation here: steelmanning the approach, testing the metaphors, and seeing if the shape of the system holds together conceptually.
There's a vocabulary drift I'm noticing though. The GLOSSARY defines "trail" as the data structure, but the user is calling it a "walk," and the file itself is named public_walk.yaml sitting in an assets/trails/ directory. That's three different names for the same thing—trail, walk, and the filename convention. I should flag that: pick one label and stick with it, because mixing handles creates the same kind of sibling-file failure we're trying to avoid.
"Negative Space" is actually useful—it's testable and predicts something real about how nested git repos can coexist without submodules. But "warm up" is another case of redundant naming: it's already called warm, SETTLE, and weblogin. Same problem again.
Now I need to think through the key management design more carefully, since that's where the real complexity lives. The core issue is that GitHub deploy keys are repo-scoped, and each key can only be attached to one repository.
The magic cookie ships a single ROT13'd key at .ssh/rot, which gets decoded to ~/.ssh/id_rsa and wired unconditionally to Host github.com. This means a second private repo would need either a second key or a different identity mechanism entirely.
I'm weighing the options here: one approach is to not give the walker git access at all, instead treating walks as sealed cartridges delivered over HTTPS, where a launcher fetches a .zip file with a pinned SHA-256 hash and verifies it using the existing stdlib verifier.
This approach fits the design principles well because the hash check becomes the only barrier and it fails closed on any deviation, the cartridge verifier already exists and is stdlib-only, and it completely sidesteps the key selector problem by eliminating it. It also aligns with the disappearing instrument rule. The tradeoff is that walks become immutable once sealed.
Actually, that's the key insight — a walk that auto-updates can't be replayed, and deterministic replay is the whole point. An auto-pulled walks repository violates that principle. This connects back to the graft-not-seed metaphor: pinning is the feature, not a limitation. A locked artifact is like grafting a cultivar, while re-resolving against a moving channel is like planting pips that won't breed true.
For the AUTHOR lane and Corporate tier that need auto-pull capability, I'd use per-repo deploy keys with SSH host aliases. The mechanism maps an alias to a specific identity file, so cloning from a custom host alias routes to the right SSH key. This satisfies the derived-path rule where the identity file path is a function of the alias, and the alias is a function of the repo.
The complexity stays bounded—N repos means N SSH config stanzas, all generated from the same wallet-style manifest that connectors.json already uses. I could extend connectors.json or create a walks.json to declare the repos, then have the flake generate the SSH config stanzas idempotently.

Immutable, Content-Addressed Cartridges

The critical detail is IdentitiesOnly yes—without it, SSH offers all keys and GitHub silently returns the first match, which with deploy keys means you get the wrong repo's access without realizing it. That's a single-candidate blindness problem: one key always works, but two keys silently pick wrong. The current SSH config append in miscSetupLogic needs to account for this.
There's also a concrete bug I can point to: the echo command appending to ~/.ssh/config doesn't interpret the \n escape sequences without the -e flag, so it writes literal backslash-n

characters on a single line instead of creating proper newlines. I can verify this directly in the source code where it checks for an existing Host entry before appending.
The guard condition grep -q "Host github.com" would then match that malformed line on subsequent runs, making the setup idempotent but permanently broken. And while a malformed ssh_config line would cause ssh to error with "Bad configuration option", the parsing gets weird because ssh interprets the entire malformed string as the value for the Host directive.
Actually, ssh wouldn't error at all — it'd treat the rest of the line as additional patterns for the Host block, creating a stanza with no actual directives inside it. So the configuration silently does nothing. This reveals the real issue: the stanza has never actually worked, and nobody noticed because ~/.ssh/id_rsa is the default identity file that ssh finds anyway. The moment you add a second key, the stanza becomes necessary and the brokenness surfaces. This is exactly the kind of maintainer-invisible bug where a decorator pattern masks the underlying problem — and it directly answers the user's question about using the same deploy key for multiple services.
Now I need to be careful about how the Nix indented string is handling the escape sequences. In Nix's "..." syntax, a bare \n doesn't get interpreted as a newline — it passes through literally as backslash-n. The bash script then receives those literal characters, and bash's echo doesn't expand them either. So the behavior is confirmed by what actually ends up in the file.
I can verify this cheaply by checking the SSH config directly with grep or cat, or by running ssh -G github.com to see what SSH actually resolves for identity files. The cleanest approach is to compare what's in the config file against what SSH reports it's using, which would immediately show if there's a mismatch.
Now I'm designing a probe that follows the discrimination principle — it needs to output differently depending on whether the config is malformed as a single line, properly formatted across multiple lines, or missing entirely. I want to bundle this into a single shell command that shows both the raw config lines and what SSH resolves.
The command uses a subshell to echo section headers, grep the config file for github entries with line numbers, then run ssh -G to see the resolved identity files — all read-only operations that don't require a connection. The key discriminator is the line count: a malformed one-liner shows one line number with both IdentityFile and User git together, while a proper multi-line config shows separate line numbers, and an absent config shows nothing.
Next I'm moving on to probe the shell's authority over which Notebooks directory gets created.
Writing diagnostic commands... Writing shell commands... The guard condition will match the existing malformed line, so the patch won't fire on a broken machine—that's a second-order problem requiring idempotence or self-healing. The right approach is to skip patching flake.nix for now and run a probe first to see what the actual ~/.ssh/config state is, since the correct fix shape depends on whether the line is broken, absent, or already fine. So a GLOSSARY entry that names the concept is a legitimate move now — it's cheap, reversible, and prevents vocabulary drift while the design is still in flight. Let me check what's already banked: there's already an entry for "trail" defined as a walkthrough represented as a JSON-subset-of-YAML file. The real issue is that having public_walk.yaml in assets/trails/ while calling it a "walk" everywhere else creates a naming inconsistency that'll trip up future searches and documentation. A GLOSSARY.md clarification is worth doing — it's a small, scoped fix that prevents the fork from spreading. I'm hesitant to rename the file itself though, since that's a bigger change.
The user's actual question is about the architecture and what logical arrangement emerges without needing a full refactor, so I should focus on the design reasoning and use concrete probes to show the pattern, keeping any patches minimal. Now I'm thinking through how to

organize the three-tier structure plus walks — the key is identifying which property determines each tier.
The organizing principle is who controls it and whether it changes under you. I'm laying out a table that shows each tier's writer, mutability, and git handling: canon is frozen after landing, corporate gets pulled in and changes, personal is yours to modify, shared is append-only, and walks are immutable by construction. The crucial insight is that walks don't fit into the four-tier workspace at all — they're a separate artifact cache because their defining property is that they cannot change.
Now I'm realizing the structural problem: putting a walk in Corporate/ (which auto-pulls) would destroy the immutability that makes it replayable. So the layout needs to separate the workspace from the cartridge cache. I'm sketching out a directory structure where Notebooks/ holds the mutable, tiered workspace, while data/walks/ holds the immutable cartridge cache keyed by hash. Since data/ is already gitignored wholesale, the walks directory sits in that negative space naturally. I'm starting to see how the derived-path rule connects to this.
The key insight is that the write-target should be a pure function of something the agent reads but can't author — and a content hash is exactly that. So data/walks// makes the derived-path rule literal and constitution-consistent. For authoring, walks get created in Notebooks/Personal/trails/ (or a private repo cloned into Corporate/), then sealed into data/walks// for execution. This mirrors the existing split: assets/nbs/ (source) flows to Notebooks/ (working copy), and final_output becomes foo.zip (sealed). The whole design compresses to one line: a walk is to a trail what foo.zip is to a compile.
This is actually the third time the system exhibits this pattern, which satisfies the constitution's own rule that a mechanism proven twice becomes a pattern. For key management, the walker needs no key — it fetches by URL and verifies by hash, like curl | bash through a launcher, cartridge, verifier, and ride. The author needs git access: one repo uses the existing single deploy key, but multiple repos require per-repo keys with Host aliases and IdentitiesOnly yes, all derived from a wallet-style manifest rather than chosen manually. The bridge is the author's seal command, which converts a trail into a cartridge and outputs its hash — the hash travels, but the private repo stays on the author's machine.
The critical insight is that the launcher must pin the hash itself, not the URL, because URLs can be swapped but hashes cannot. The curl | bash vulnerability the user fears is neutralized when the hash lives in the command the human types or in a signed manifest, not fetched alongside the payload. If both the payload and hash come from the same source, the hash proves nothing — that's the classic failure mode and it's worth being explicit about it.
The distinction matters: a hash in the command represents the SHA-256 rung (exact bytes, verified out of band), while a signature is the third rung. The manifest-signing lane already scopes ssh-keygen -Y sign as the lowest-friction non-repudiation mechanism. So rather than inventing a new signing scheme for walks, walks should be the first real consumer of the manifest-signing lane that's already in scope — it's the same infrastructure.
The user thinks he needs a new key system, but he's actually just discharging an earmark that's been open since July. There's elegant symmetry here: ssh-keygen -Y sign uses SSH key material, the same class as deploy keys, but for a completely different purpose — signing versus access. One key type, two jobs, and conflating them is the trap. Access keys must be per-repo; signing keys must be per-identity. That's sane key management for people who don't know git: two keys, two jobs, both derived, neither chosen.
The ROT13 deploy key is an access key, while a signing key is different. The user asked if he could use the same deploy key for both to keep it simple — the answer is no for access (GitHub forbids it) and no for signing (different lifetime, different revocation story). But complexity doesn't

explode because the count is bounded by role, not repo: one signing key per author plus N access keys per repo, where N is generated rather than hand-managed.
Now I'm looking at the "different install paths" part. The user says it's already in there, and I can see the evidence: the CLI has an install command that puts things in a custom directory, there's a whitelabel install script, and PIPULATE_WHITELABEL is referenced in the codebase.
The real open item is SEED REDUCTION #2: discovery should use a marker file instead of searching by name. That's the same derived-path rule, and it's directly relevant to how a walk launcher needs to find things.
So the walks question and the whitelabel-discovery question are actually the same problem — both are about how a launcher finds its workshop without making assumptions. That's a useful unification.
Now I need to settle on a direct answer and think through the pushback. The main strike points are that "walk" is redundant naming when we already have three other terms, and the private-repo-with-deploy-key approach is the expensive path while the simpler approach is just not providing git access to the walker. There's also the cryptographic sealing issue — if the hash and payload arrive together via curl, that's just a checksum, not a real seal. But the sharpest practical point is that the directory question is actually downstream: two different systems currently create Notebooks directories, so renaming before unifying the writer means the change only lands in one code path. The real first move isn't a rename at all — it's making one system own the responsibility directly. The concrete test is comparing ls Notebooks/Playground output between a default shell and the quiet shell — they should print differently now but identically after the fix. The quiet shell runs both the Python setup and miscSetupLogic, which is where the Playground notebook creation currently happens in runScript, so moving that logic into miscSetupLogic should make the outputs match. Writing diagnostic commands...
Now I'm ready to structure the answer: confirming the context with the missing-files flag, giving a direct yes with the observation that three of the four pieces already exist under different names, addressing the vocabulary distinction between walk and trail, and explaining what a walk actually is within the architecture. Writing shell probes... Writing shell commands... I'm weighing the token budget here — adding AGENTS.md on top of the 280k payload would be too much. I should recommend dropping the rolling pin and postsc 300 from the next compile, then add the trail files instead. That's the leaner approach. Now I need to refine the GLOSSARY entry for "trail" carefully against the source. I need to distinguish between a trail as mutable source data and a walk cartridge as an immutable, content-addressed artifact—they're separate concepts precisely because they have opposite mutability properties. But I'm second-guessing whether I'm getting ahead of myself here; the rule about judgment versus mechanism says I should only assert what the system actually does once I've verified it works that way.
Let me be more careful: the naming itself is pure judgment, but calling something a "walk cartridge" when it doesn't exist yet is a proposal, so I should mark that explicitly as pending. I'll simplify the ruling to focus only on the name and tag the cartridge concept as proposed-not-yet-shipped.
Also need to match the exact formatting from the source—no leading indentation in that glossary block. That line is unique across the file, so I'll use it as my search anchor without the em-dash risk. The replacement will append the ruling lines to that same line. GLOSSARY.md is a compile-time data file, so no special ignition needed.
Now I'm moving to the PROMPT caboose, which should set up the next turn by reading the trail files, taking baseline readings, and deciding between the two key forks: whether to generalize the cartridge schema versus creating a second schema, and whether to treat the corporate tier

as a private repo or cache the walks as cartridges. I'll draft this unwrapped per the honest-seam rule, avoiding em-dashes in the new lines to reduce transport risk.
The canary payload arrived linkified—the text and target match exactly as the constitution describes for self-referential autolinks, which is the signature I'm looking for.
According to the constitution, a linkified canary proves a transform occurred and is decisive. This means every bare www-prefixed token in the payload is now suspect and I must report this before proceeding. I need to scan through what I received to check for other problematic tokens—my GLOSSARY patch is safe since it carries no www-tokens, but I should verify the other files like flake.nix and foo_files.py don't contain any www-prefixed strings that would violate the constraint. The scheme-bearing URLs survived while bare ones didn't, which aligns with the grammar I've identified. I've checked the codebase for other bare www instances and found the gnu.org control case confirms the pattern, so I'm ready to write this up with a focused structure: starting with the canary firing, then the context and missing files, and finally the implications.
Show less
Done
Opus 5 Max:

0. The render canary fired on this payload — read this first

The Manifest's canary arrived **linkified**: link text equal to link target minus its scheme. That is the decisive reading under THE RENDER-GAP RULE, not the ambiguous one. The scheme-bearing https://www.gnu.org/licenses/ in prompt_foo.py's AGPL header arrived **intact** in the same payload, which is the co-located control and matches the banked grammar exactly: a pre-existing scheme suppresses, a bare prefix does not.
Consequence for this turn: every bare www-prefixed token anywhere in this payload — Codebase bodies and ! receipt stdout included — is suspect. My one patch below carries none, so it is safe to apply. But do not let me quote one into a REPLACE block on the strength of this payload alone.
Context verified, with one material gap named below.

---

1. What is missing, and what I will therefore not patch

foo_files.py Chapter VIII-b says it plainly: scripts/mother_cat.py, scripts/walk.py, and assets/trails/*.yaml "EXIST and are UNREAD by any compile so far... Everything below is a filename, not a claim."
That is still true in this payload. So I will not emit a patch against the trail system, will not assert how url_env resolves, and will not rename public_walk.yaml. Probe 4 below is the first read; the design reasoning that follows does not depend on it.

---

2. Yes, it makes sense — and three of the four pieces

are already built

Stripping the vocabulary away, you described four requirements:

requirement	already exists as
a planned sequence of explicit, pre-authenticated URLs	the trail: assets/trails/*.yaml, seven mandatory keys, url_env per stop
auth resolved *before* the URL is pulled, no redirect branching	SETTLE + weblogin <apex> warming data/uc_profiles/default, inherited by persistent=True scrapes and the ?URL sigil
a byte-reproducible, hash-verifiable, zero-dependency container	scripts/foo_cartridge.py — canonical ZIP, fixed epoch/mode/order, SHA-256 per member, fail-closed stdlib verifier
a curl | bash front door	install.sh and mck.sh, already on the served roster

The genuinely new work is the **join**: a command that seals a trail into a cartridge, and a launcher that fetches one by hash. That is much smaller than what you were bracing for.

---

3. Three strikes, in order of what they cost you

Strike 1 — "walk" is a fourth name for a thing that already has three

assets/trails/ (directory), public_walk.yaml (file), trail (GLOSSARY), mothercat (alias). Adding "walk" as a synonym makes every future rg probe return half the truth — the UNANCHORED-WITNESS failure, arranged in advance.
Pick **trail**. Then "walk" is available for the thing that genuinely has no name: the *sealed, hash-pinned* form. And those two words earn their separation for a real reason — a trail is mutable source, a walk cartridge is immutable and content-addressed. Opposite mutability is exactly what deserves two nouns.

Strike 2 — the key question is the wrong question

You are right that GitHub deploy keys are repo-scoped; a key already attached to one repo is rejected on a second. But the expensive path is the one you named first, and it is not the one you need, because **the author and the walker are different lanes**:

- **The author** needs git. Private repo, deploy key, the whole ceremony. This is your machine, once.
- **The walker** needs a URL and a hash. No key. No git. No repo access at all.

The private repo never leaves your machine. What travels is a sealed artifact and its digest. Key management for "people who really don't know git" collapses to *there is no key* — which is the only key-management system that cannot be gotten wrong.
When you *do* need N private repos on the author side, the complexity is bounded by role, not by repo count: one signing identity, plus N access keys, each reached through a generated ~/.ssh/config stanza with a derived Host alias and — this is the load-bearing line nobody remembers — IdentitiesOnly yes. Without it, ssh offers every key and GitHub answers with the

first that matches, which for deploy keys means silently getting the *wrong repo's* access. That is SINGLE-CANDIDATE BLINDNESS in ssh: with one key it always works; with two it silently picks wrong. Derived alias, derived path, nothing chosen — DERIVED-PATH RULE, applied to identity files.

The Bigger Picture of Receipt-Driven Design

One caution grounded in raw source in this payload. miscSetupLogic writes:
echo "Host github.com\n IdentityFile ~/.ssh/id_rsa\n User git" >> ~/.ssh/config

Nix indented strings pass a lone backslash through literally (the file's own printf comment says so), and bash's builtin echo does not expand \n. So this writes one long line whose Host pattern swallows the rest — a stanza containing no directives. ssh does not error; it just matches nothing and falls through to the default identity file, which happens to be the very path the stanza names. **It has therefore never mattered, and nobody could have noticed.** The grep -q "Host github.com" guard then matches that malformed line forever, so it is idempotently broken. The day you add a second key is the day it matters. Probe 1 takes the BEFORE reading; I am deliberately not patching it this turn, because the correct shape of the fix depends on whether your config shows one line, three lines, or none.

Strike 3 — a hash that arrives with the payload is a checksum, not a seal

"Cryptographically sealed with hashes so shenanigans can't switch out what walk is being run" only holds if the digest arrives **out of band** from the bytes it describes. If curl fetches both the cartridge and its .sha256 from the same host, an attacker who can swap one swaps both and the verification passes.
The digest has to be in the command the human types, or in a signature over the manifest. Which is exactly THE RECEIPT LADDER RULE's third rung — and exactly the MANIFEST-SIGNING LANE earmark that has been open since 2026-07-22, already scoped to ssh-keygen -Y sign as the lowest-ceremony non-repudiation. **You do not need a new signing scheme for trails. Trails are the first real consumer of that earmark.** Discharging it discharges both.

---

4. The arrangement being inferred

The organizing question for Notebooks/ is not *what kind of content is this* — it is **who writes it, and does it change under you**:

tier	writer	changes under you?
canon	the flake, copy-if-absent	no — frozen after first landing
Corporate	an org repo, pulled	**yes**
Personal	you	only by your hand
Shared	you, published per-name	only by your hand

Now put a trail in that table and it does not fit — because the defining property of a replayable

trail is that it **cannot** change. Dropping trails into an auto-pulled Corporate tier destroys the exact property your Flight Data Recorder requirement exists to protect. Your own foo_files.py already has the metaphor: *an installer that re-resolves against a moving channel is a pip; a pinned lock is a graft.* **A trail must be a graft.**
So walks are not a fifth tier of the workspace. They are a content-addressed cache, and the system already gitignores the right neighbourhood:

```
Notebooks/                  the workspace — mutable, tiered by writer
├── Corporate/                read-only canon · a private repo in negative space
├── Personal/                 gitignored · Playground, Client_Work, Deliverables live HERE
└── Shared/<name>/              the one outbound surface

data/walks/<sha256>/          the cartridge cache — immutable, keyed by content
└── walk.zip + manifest.json
```

data/ is already gitignored wholesale, and a content hash is DERIVED-PATH RULE made literal: the write target is a pure function of an identity value the writer reads and cannot author.
And the whole design compresses to one sentence you can hand a newcomer: **a walk cartridge is to a trail what foo.zip is to a compile.** assets/nbs/ → Notebooks/ is the same shape a third time. A mechanism proven twice is a pattern; this is an instance, not an invention.
Your Corporate/ tier is then exactly right for what you called "filled into Pipulate through the Negative Space" — that is where the private *authoring* repo mounts. And AGENTS.md's nearest-ancestor rule means nothing has to move: Notebooks/.agents/skills/ stays Pipulate's, Notebooks/Corporate/.agents/skills/ is the org's, both discovered, closest wins. OKF's index.md and the reserved-filename convention drop into each tier root unchanged. You do not blend the three standards — you let each one govern the layer it was designed for.

---

5. Why the rename is not the first chisel-strike

Two authorities currently create directories under Notebooks/, and they run in different lanes:

- runScript → the Playground/ WELCOME.md and copy_notebook_if_needed. **Skipped entirely by .#quiet.**
- miscSetupLogic → Notebooks/Shared and its README. Runs in all three shells, and the comment there says exactly why.

Shared/ got it right. Playground/ did not. Rename first and the new names land in one lane and not the other — and the lane that misses is the one agents and scripts actually live in.
First baby step: move one directory's creation from runScript into miscSetupLogic, using the printf pattern already proven right there (never a heredoc — NO HEREDOC INSIDE AN INTERPOLATED NIX STRING). One directory, one car. The witness is LANE-DISAGREEMENT used as evidence: ls Notebooks from a default shell and from .#quiet print differently today and identically after.
Ordered from there, each its own ride:

1. Unify directory creation into the all-lanes block, one dir per car.
2. Introduce Personal/ as the parent, move Playground/Client_Work/Deliverables under it.
3. seal — turn a trail into a canonical cartridge. Reuses foo_cartridge.py; the only real fork is whether FOO_CARTRIDGE_MEMBERS generalizes or a second schema is minted. Decide that with the trail files in context, not before.

4. Fetch-by-hash in the launcher, with the digest supplied by the human. Marker-based workshop discovery (whitelabel.txt, not the literal name pipulate) is the same DERIVED-PATH problem already banked as SEED REDUCTION #2 — do them together.
5. Discharge the MANIFEST-SIGNING LANE. That is when the seal becomes a signature.

Step 3 is where SINGLE-CANDIDATE BLINDNESS bites: with one trail source, any resolution logic is untested by construction. Create the second source before trusting the selector.

---

(1) PROBES

Read-only, bounded. Probe 1 takes the BEFORE reading on the ssh stanza. Probe 2 establishes which lane owns each mkdir — this gates step 1 and is the only probe that must land before any flake car. Probe 3 confirms which tiers sit in gitignored negative space and can host an independent repo. Probe 4 is the **first read** of the trail system and gates every claim in section 3 of this reply. Probe 5 tells you whether hash verification already exists in a shipped launcher — if it does, step 4 is reuse; if not, it is new code and owes its own ride.

```
{ echo "== 1. ssh identity resolution =="; grep -n github ~/.ssh/config 2>/dev/null | head -5; ssh -G github.com 2>/dev/null | grep -i '^identityfile' | head -6; }
{ echo "== 2. flake lane boundaries =="; grep -n 'runScript = \|miscSetupLogic = \|pythonSetupLogic = \|gitUpdateLogic = \|shellHook' flake.nix; echo "-- creation sites --"; grep -n 'mkdir -p\|copy_notebook_if_needed\|Playground' flake.nix | head -20; }
{ echo "== 3. notebooks tier reality =="; ls -1 Notebooks 2>/dev/null; for d in Advanced_Notebooks Educational_Notebooks imports Playground Client_Work Deliverables Shared; do printf '%s: ' "$d"; git check-ignore -q "Notebooks/$d" && echo IGNORED || echo not-ignored; done; }
{ echo "== 4. trail system, first read =="; ls -1 assets/trails 2>/dev/null; grep -n 'DEFAULT_TRAIL' scripts/walk.py 2>/dev/null | head -5; grep -rn 'PIPULATE_TRAIL_\|url_env' scripts/walk.py scripts/mother_cat.py 2>/dev/null | head -12; }
{ echo "== 5. launcher hash discipline =="; ls -1 assets/installer 2>/dev/null; grep -rn 'sha256\|shasum\|whitelabel' assets/installer/ 2>/dev/null | head -12; }
```

(2) NEXT CONTEXT

Budget note first: this compile ran 280k tokens, of which the rolling pin (60k) and postsc 300 (36k) are pure narrative. Drop both next turn — the trail files are what you need, and they are what you have never had.

```
! { echo "== 1. ssh identity resolution =="; grep -n github ~/.ssh/config 2>/dev/null | head -5; ssh -G github.com 2>/dev/null | grep -i '^identityfile' | head -6; }
! { echo "== 2. flake lane boundaries =="; grep -n 'runScript = \|miscSetupLogic = \|pythonSetupLogic = \|gitUpdateLogic = \|shellHook' flake.nix; echo "-- creation sites --"; grep -n 'mkdir -p\|copy_notebook_if_needed\|Playground' flake.nix | head -20; }
! { echo "== 3. notebooks tier reality =="; ls -1 Notebooks 2>/dev/null; for d in Advanced_Notebooks Educational_Notebooks imports Playground Client_Work Deliverables Shared; do printf '%s: ' "$d"; git check-ignore -q "Notebooks/$d" && echo IGNORED || echo not-ignored; done; }
```

! { echo "== 4. trail system, first read =="; ls -1 assets/trails 2>/dev/null; grep -n 'DEFAULT_TRAIL' scripts/walk.py 2>/dev/null | head -5; grep -rn 'PIPULATE_TRAIL_\|url_env' scripts/walk.py scripts/mother_cat.py 2>/dev/null | head -12; }
! { echo "== 5. launcher hash discipline =="; ls -1 assets/installer 2>/dev/null; grep -rn 'sha256\|shasum\|whitelabel' assets/installer/ 2>/dev/null | head -12; }
scripts/walk.py
scripts/mother_cat.py
assets/trails/public_walk.yaml
assets/installer/mck.sh
assets/installer/install.sh
flake.nix
GLOSSARY.md
scripts/foo_cartridge.py
foo_files.py

(3) PATCHES

One car: the name ruling, banked now so the vocabulary cannot fork further while the design is in flight. Judgment rule, articulation-banked; the cartridge half is tagged PROPOSED because no mechanism backs it yet.
No flake patch this turn. The ssh-config fix's correct shape — repair in place, or repair plus a self-healing guard — is decided by probe 1's output, and emitting it now would be a guess about the guard's behavior on this machine.
Target: GLOSSARY.md
[[[SEARCH]]]
`assets/trails/`. A new walkthrough is a new YAML, never a new script.
[[[DIVIDER]]]
`assets/trails/`. A new walkthrough is a new YAML, never a new script.
NAME RULING (banked 2026-08-07, articulation-banked): "walk" is NOT a fourth
noun for this. The corpus already carries `assets/trails/` (directory),
`public_walk.yaml` (filename), `trail` (this entry), and `mothercat` (alias)
for one concept; a fifth spelling makes every rg probe return half the truth,
which is the UNANCHORED-WITNESS failure arranged in advance. TRAIL is the noun
for the mutable source. PROPOSED, no mechanism yet: a sealed, hash-pinned,
content-addressed package of a trail is a WALK CARTRIDGE, and it is a
different artifact rather than a synonym. The two words earn separation for
one reason only, and it is the reason that matters: a trail is mutable source,
a walk cartridge cannot change. A trail that auto-updates is not replayable.
[[[REPLACE]]]

Ignition: none required. GLOSSARY.md is read from disk by prompt_foo.py at compile time, so the next compile witnesses it directly.

(4) PROMPT

Probe receipts are in. Read scripts/walk.py and scripts/mother_cat.py for the first time and

answer three things from raw source, not inference.

FIRST: does the trail's url_env mechanism actually express a fully-explicit, pre-authenticated URL with IDs already embedded in path or querystring, and where does the value come from at ride time? Name the exact resolution order. If a stop can produce a redirect, say so — that is the Flight Data Recorder requirement failing and it is better to find it now.

SECOND: rule on the cartridge fork. foo_cartridge.py's FOO_CARTRIDGE_MEMBERS is a hard three-member tuple and verify_context_cartridge derives prompt.md from payload.md. To seal a trail we either generalize that verifier or mint a second schema beside it. Give me the argument for one, name what the other costs, and pick. Do not build both.

THIRD: from probe 2's lane boundaries, emit exactly ONE car moving ONE directory's creation out of runScript and into miscSetupLogic, printf and never a heredoc. Name the ignition and name the probe whose BEFORE and AFTER differ across it. One directory. Not two.

Do not touch the ssh config yet — probe 1's reading decides whether that fix is a repair in place or a repair plus a self-healing guard, and I want it as its own ride.

(5) EXTERNAL DELIVERABLES

None this turn.
MikeLev.in:
1: Probe:
(nix) pipulate $ g

Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified: foo_files.py

no changes added to commit (use "git add" and/or "git commit -a")
(nix) pipulate $ blast
📝 Committing: chore(router): set AI-edit blast boundary (foo_files.py context curation)
[main 7c20c21f] chore(router): set AI-edit blast boundary (foo_files.py context curation)
 1 file changed, 5 insertions(+), 5 deletions(-)
🚀 Pushing 1 commit(s) to remote...
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 48 threads
Compressing objects: 100% (3/3), done.

```
Writing objects: 100% (3/3), 390 bytes | 390.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:pipulate/pipulate.git
   ac99f6c9..7c20c21f  main -> main

$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(nix) pipulate $ { echo "== 1. ssh identity resolution ==="; grep -n github ~/.ssh/config 2>/dev/null |
head -5; ssh -G github.com 2>/dev/null | grep -i '^identityfile' | head -6; }
{ echo "== 2. flake lane boundaries ==="; grep -n 'runScript = \|miscSetupLogic =
\|pythonSetupLogic = \|gitUpdateLogic = \|shellHook' flake.nix; echo "-- creation sites --"; grep -n
'mkdir -p\|copy_notebook_if_needed\|Playground' flake.nix | head -20; }
{ echo "== 3. notebooks tier reality ==="; ls -1 Notebooks 2>/dev/null; for d in
Advanced_Notebooks Educational_Notebooks imports Playground Client_Work Deliverables
Shared; do printf '%s: ' "$d"; git check-ignore -q "Notebooks/$d" && echo IGNORED || echo
not-ignored; done; }
{ echo "== 4. trail system, first read ==="; ls -1 assets/trails 2>/dev/null; grep -n
'DEFAULT_TRAIL' scripts/walk.py 2>/dev/null | head -5; grep -rn 'PIPULATE_TRAIL_\|url_env'
scripts/walk.py scripts/mother_cat.py 2>/dev/null | head -12; }
{ echo "== 5. launcher hash discipline ==="; ls -1 assets/installer 2>/dev/null; grep -rn
'sha256\|shasum\|whitelabel' assets/installer/ 2>/dev/null | head -12; }
== 1. ssh identity resolution ==
2:Host github.com
3:  HostName github.com
14:  HostName github.com
21:  HostName github.com
identityfile ~/.ssh/id_rsa
== 2. flake lane boundaries ==
493:          # aarch64-darwin. The flake therefore died BEFORE the shellHook, which
515:runScript = pkgs.writeShellScriptBin "run-script" ''
693:            # own line in each shellHook), never sourced, so exiting here
802:        pythonSetupLogic = ''
815:        gitUpdateLogic = ''
938:        miscSetupLogic = ''
987:          # shellHook does not set -e) and therefore worse than a real error:
1036:          # shellHook, the terminator lost its column-0 alignment, and bash
1840:            shellHook = ''
1851:            shellHook = ''
1863:            shellHook = ''
-- creation sites --
534:          copy_notebook_if_needed() {
538:              mkdir -p "$(dirname "$dest")"
550:          if [ ! -f "Notebooks/Playground/WELCOME.md" ]; then
551:            echo "INFO: Setting up your personal Playground..."
```

552:           mkdir -p "Notebooks/Playground"
553:           cat << 'PLAYGROUND_EOF' > "Notebooks/Playground/WELCOME.md"
554:         # 🎊 Welcome to the Playground!
567:         cd Notebooks/Playground
574:         Right now this Playground is a **solo** sandbox: just you, your throwaway
588:         ├── Playground/   ◄── THIS private. NOTHING here is ever shared.
649:         #   copy_notebook_if_needed -- notebooks on disk are a deliverable of
659:         copy_notebook_if_needed
825:               mkdir -p "$TEMP_DIR/.ssh"
850:         #   lane below; durable user state (~/.config/pipulate, Playground,
999:           mkdir -p .git/hooks
1013:   #   personal Playground/ Client_Work/ Deliverables/
1029:   # Playground's WELCOME.md -- is skipped entirely by .#quiet, so a
1033:   mkdir -p "$PIPULATE_ROOT/Notebooks/Shared"
1044:   printf "%s\n" "# Shared" "" "This is the one folder you use on purpose to hand work to someone else." "Everything else under Notebooks/ is either yours alone or delivered to you." "Nothing here is private: assume a teammate will read it." "" "Make a folder with your own name, and work inside it:" "" "   mkdir -p Notebooks/Shared/yourname" "" "One folder per person means two people can drop work at the same moment and" "never collide, so there is nothing to merge and no git to learn." "" "Pipulate git ignores this whole folder. To back yours up, put your own git" "repository inside your own subfolder; it will not fight with anything above." > "$PIPULATE_ROOT/Notebooks/Shared/README.md"
1314:   mkdir -p "$(dirname "$subs")"
== 3. notebooks tier reality ==
Advanced_Notebooks
browser_cache
Client_Work
data
Deliverables
Educational_Notebooks
imports
__init__.py
Onboarding.ipynb
Playground
__pycache__
Shared
Advanced_Notebooks: not-ignored
Educational_Notebooks: IGNORED
imports: not-ignored
Playground: IGNORED
Client_Work: IGNORED
Deliverables: IGNORED
Shared: IGNORED
== 4. trail system, first read ==
botify_pageworkers.yaml
first_context.yaml
practice.yaml
public_walk.yaml

```
23:DEFAULT_TRAIL = REPO_ROOT / "assets" / "trails" / "public_walk.yaml"
420:        default=DEFAULT_TRAIL,
scripts/walk.py:42:    "name", "label", "guidance", "url_env", "target_slot",
scripts/walk.py:213:        url_env = _text(stop["url_env"], f"{where}.url_env")
scripts/walk.py:234:        if not ENV_RE.fullmatch(url_env):
scripts/walk.py:236:                f"{where}.url_env must name an environment variable"
scripts/walk.py:253:            "url_env": url_env,
scripts/walk.py:323:        url = os.environ.get(stop["url_env"], "").strip()
scripts/walk.py:330:                f"unset environment variable {stop['url_env']}"
scripts/walk.py:373:            "url_env": stop["url_env"],
scripts/mother_cat.py:199:        url_env = stop["url_env"]
scripts/mother_cat.py:201:            url = os.environ[url_env]
scripts/mother_cat.py:204:                f"stop {stop['name']!r} requires environment variable
{url_env}"
== 5. launcher hash discipline ==
install.sh
mck.sh
replay.sh
assets/installer/replay.sh:104:sha256_of() {
assets/installer/replay.sh:105:  if command -v sha256sum >/dev/null 2>&1; then
assets/installer/replay.sh:106:    sha256sum "$1" | cut -d' ' -f1
assets/installer/replay.sh:107:  elif command -v shasum >/dev/null 2>&1; then
assets/installer/replay.sh:108:    shasum -a 256 "$1" | cut -d' ' -f1
assets/installer/replay.sh:110:    python3 -c 'import hashlib,sys;
print(hashlib.sha256(open(sys.argv[1],"rb").read()).hexdigest())' "$1"
assets/installer/replay.sh:145:  CORE_SUM="$(sha256_of "$CORE_PATH")"
assets/installer/replay.sh:147:    echo "🔐 Verifier core pinned and matched (sha256
${CORE_SHA256:0:12})"
assets/installer/replay.sh:170:  CORE_SUM="$(sha256_of "$CORE_PATH")"
assets/installer/replay.sh:182:  echo "🔐 Verifier core pinned and matched (sha256
${CORE_SHA256:0:12})"
assets/installer/replay.sh:201:print(f"CARTRIDGE VERIFIED
sha256={result['archive_sha256'][:12]}...")
assets/installer/mck.sh:18:# whitelabel may ride the same way through
__MCK_WHITELABEL__. Local forms:
(nix) pipulate $
```

2: Context:

```
# adhoc.txt    _   _   _ to set context____ _   _  ___  ____  _   <F5> Simpson Couch Gag Here
(explain anything to the audience you feel needs it explained)
#     / \   __| | | | | | ___   ___   / ___| | | |/ _ \|  _ \| |
# ahe/ _ \ / _` | | |_| |/ _ \ / __| | |   | |_| | | | | |_) | |  The big master folder scheme clean-up begins.
# ahc ___ \ (_| | |  _  | (_) | (__  | |___|  _  | |_| |  __/|_|
#  /_/   \_\__,_| |_| |_|\___/ \___|  \_____|_| |_|\___/|_|   (_)
# Ad Hoc CHOP: The Not-Managed-by-Git Safe-for-Client-Data place

# THE ROLLING PIN BOOK ORE SPINE / FODDER FOR THE BOOK
! python scripts/articles/lsa.py -t 1 --reverse --fmt dated-slugs  # <-- The "Rolling Pin" that gives
```

the 40K foot book-spine view of book-ore.
GLOSSARY.md # <-- I think this glossary goes well with the book-ore spine to do world building.
scripts/articles/lsa.py # <-- Useful for refining commands like `posts`, critical to Second Brain concept.

THE QUIRKY AMIGA-LOVING HUMAN
~/repos/nixos/autognome.py # <-- Letting the AIs really understand my environment (The Brave Little Tailor punches above Their Weight Class proving the dunning-kruger effect the gate-keeper's (lower-case) lament.)
init.lua # <-- Daily driver hot-keys that overlap with aliases in flake.nix

THE HAND-CRANKED AGENTIC FRAMEWORK
prompt_foo.py # <-- Prompt Fu compiler, makes the very README for AGENTS-like payload you're reading right now, but it needs to be more like that
foo_files.py # <-- This is the router, evolving book outline and the things you pin-up to produced the recursive self-improvement loops

MAIN ACTUATORS, IaC & NEGATIVE SPACE
apply.py # <-- How can "Web UI" ChatBots edit your code? With this Aider-inspired Player Piano patch applier.
.gitattributes # <-- Model: understand that `nbstripout` and `jupytext` are both in play. Just talk the human through .ipynb patches.
.gitignore # <-- Creates "negative space" for sub-rep's to share parent environment and "snap" proprietary secret features into place.
flake.nix # <-- Solves world's WRITE ONCE RUN ANYWHERE problem like Java never could. Also resolves the bootstrap paradox.
requirements.in # <-- All known dependencies and (necessary) version pinning. WORA gotcha's exposed.
__init__.py # <-- Master versioning
pyproject.toml # <-- The PyPI Packaging details
cli.py # <-- Catch-all actuator for PyPI envs, Python anchoring, MCP tool-call (plus alternatives) and **kwargs like wrapping for CLI

scripts/xp.py # <-- Transforms host OS copy-paste buffer player-piano music into context-payload.
scripts/ai.py # <-- How I constantly use local AI to write git commit messages with `m` alias.

CONTEXT PORTABILITY SYSTEM
scripts/foo_cartridge.py # Needs description
scripts/foo_replay.py # Needs description

FREQUENTLY USEFUL TO HAVE IN CONTEXT
release.py # <-- How everything ends up where it does (GitHub, PyPI, etc.)
scripts/weblogin.py # <-- Lets the user "warm up" the cache for their web logins at their leisure on a profile that persists.
scripts/crawl.py # <-- Feel free to ask for something to be crawled and included in the

next turn.
imports/voice_synthesis.py # <-- The wand can talk to you
scripts/release/version_sync.py # <-- Needs to be wrapped into release.py and eliminated, I think.

--- Under this line is were you paste what the AI gives you ---
--- We call it context but it's really just the right-hand ---
--- blast-radius of the "probes" to make this all science. ---

--- END `adhoc.txt` TEMPLATE ---

server.py
scripts/mcp_menu.py
#
scripts/connectors/README.md
scripts/connectors/gmail.py
scripts/connectors/confluence.py
scripts/connectors/jira.py
scripts/connectors/slack.py
scripts/connectors/botify.py
scripts/connectors/gsc.py
scripts/connectors/sheets.py
scripts/connectors/wallet.py
scripts/connectors/mcp.py
#
tools/scraper_tools.py
tools/__init__.py
tools/dom_tools.py
tools/llm_optics.py
scripts/walk.py
assets/trails/first_context.yaml
scripts/weblogin.py
#
! test -f assets/installer/fdr.sh && echo EXISTS || echo ABSENT
! bash -n assets/installer/fdr.sh && echo SYNTAX-OK
! grep -c '/dev/tty' assets/installer/fdr.sh
! ls browser_cache/looking_at
assets/installer/fdr.sh
assets/installer/replay.sh
assets/trails/public_walk.yaml
scripts/mother_cat.py

`d`, `Shift`+`G`! I have to remember that.

! { echo "== 1. ssh identity resolution =="; grep -n github ~/.ssh/config 2>/dev/null | head -5; ssh -G github.com 2>/dev/null | grep -i '^identityfile' | head -6; }
! { echo "== 2. flake lane boundaries =="; grep -n 'runScript = \|miscSetupLogic = \|pythonSetupLogic = \|gitUpdateLogic = \|shellHook' flake.nix; echo "-- creation sites --"; grep -n

'mkdir -p\|copy_notebook_if_needed\|Playground' flake.nix | head -20; }
! { echo "== 3. notebooks tier reality =="; ls -1 Notebooks 2>/dev/null; for d in Advanced_Notebooks Educational_Notebooks imports Playground Client_Work Deliverables Shared; do printf '%s: ' "$d"; git check-ignore -q "Notebooks/$d" && echo IGNORED || echo not-ignored; done; }
! { echo "== 4. trail system, first read =="; ls -1 assets/trails 2>/dev/null; grep -n 'DEFAULT_TRAIL' scripts/walk.py 2>/dev/null | head -5; grep -rn 'PIPULATE_TRAIL_\|url_env' scripts/walk.py scripts/mother_cat.py 2>/dev/null | head -12; }
! { echo "== 5. launcher hash discipline =="; ls -1 assets/installer 2>/dev/null; grep -rn 'sha256\|shasum\|whitelabel' assets/installer/ 2>/dev/null | head -12; }
scripts/walk.py
scripts/mother_cat.py
assets/trails/public_walk.yaml
assets/installer/mck.sh
assets/installer/install.sh
flake.nix
GLOSSARY.md
scripts/foo_cartridge.py
foo_files.py
tools/scraper_tools.py

3: Patches:
(nix) pipulate $ ahe
(nix) pipulate $ g

Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'GLOSSARY.md'.
(nix) pipulate $ d
diff --git a/GLOSSARY.md b/GLOSSARY.md
index 6755fb03..d200d52c 100644
--- a/GLOSSARY.md
+++ b/GLOSSARY.md
@@ -260,6 +260,16 @@ PATH, never inlined, so a huge page cannot flood the paste.
 ### The nouns
 trail — a walkthrough as DATA: one JSON-subset-of-YAML file in
 `assets/trails/`. A new walkthrough is a new YAML, never a new script.
+NAME RULING (banked 2026-08-07, articulation-banked): "walk" is NOT a fourth
+noun for this. The corpus already carries `assets/trails/` (directory),
+`public_walk.yaml` (filename), `trail` (this entry), and `mothercat` (alias)
+for one concept; a fifth spelling makes every rg probe return half the truth,
+which is the UNANCHORED-WITNESS failure arranged in advance. TRAIL is the noun

+for the mutable source. PROPOSED, no mechanism yet: a sealed, hash-pinned,
+content-addressed package of a trail is a WALK CARTRIDGE, and it is a
+different artifact rather than a synonym. The two words earn separation for
+one reason only, and it is the reason that matters: a trail is mutable source,
+a walk cartridge cannot change. A trail that auto-updates is not replayable.

 rider — the walkthrough as CODE: `scripts/mother_cat.py`. One rider, N
 trails. The split is the whole design.
(nix) pipulate $ m
📝 Committing: chore: Refine GLOSSARY entry for 'trail' and 'walk'
[main 07527a67] chore: Refine GLOSSARY entry for 'trail' and 'walk'
 1 file changed, 10 insertions(+)
(nix) pipulate $ git push
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 48 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 835 bytes | 835.00 KiB/s, done.
Total 3 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:pipulate/pipulate.git
   7c20c21f..07527a67 main -> main
(nix) pipulate $

4: Prompt:
Probe receipts are in. Read scripts/walk.py and scripts/mother_cat.py for the first time and answer three things from raw source, not inference.
FIRST: does the trail's url_env mechanism actually express a fully-explicit, pre-authenticated URL with IDs already embedded in path or querystring, and where does the value come from at ride time? Name the exact resolution order. If a stop can produce a redirect, say so — that is the Flight Data Recorder requirement failing and it is better to find it now.
SECOND: rule on the cartridge fork. foo_cartridge.py's FOO_CARTRIDGE_MEMBERS is a hard three-member tuple and verify_context_cartridge derives prompt.md from payload.md. To seal a trail we either generalize that verifier or mint a second schema beside it. Give me the argument for one, name what the other costs, and pick. Do not build both.
THIRD: from probe 2's lane boundaries, emit exactly ONE car moving ONE directory's creation out of runScript and into miscSetupLogic, printf and never a heredoc. Name the ignition and name the probe whose BEFORE and AFTER differ across it. One directory. Not two.
Do not touch the ssh config yet — probe 1's reading decides whether that fix is a repair in place or a repair plus a self-healing guard, and I want it as its own ride.
5: Deliverables: I will concede to your most proper semantics. I think you've got a good read on this.
Opus 5 Max Thinks: Thought for 4m 41s
Thought for 4m 41s
I'm reviewing the source files that are now available in context to work through the three questions systematically, starting with verifying what's present.
I'm noticing the render canary signature — a linkified www-token that arrived with the scheme stripped, which is the telltale pattern. This means I need to flag it again and treat all bare

www-prefixed tokens in the payload as potentially suspect. The canary count should appear in the processing output, though it looks like that line might be getting filtered out from the summary.
Now I'm working through the three questions, starting with the url_env mechanism. The trail file stores environment variable names, not URLs directly. The validation enforces a strict naming pattern for these variables. When the plan is built, the code retrieves the actual URL from the environment, validates it exists, and then parses it to extract the scheme and other components.
Examining the browser parameter validation... In the guided path, the function hashes the entire URL including query parameters to generate a unique digest, which gets appended to the readable path component. This means the guided lane avoids the collision problem by incorporating the full URL into the directory name. Good point about query strings being handled safely in the guided lane. Now looking at how mother_cat.py retrieves the URL at runtime — it pulls the environment variable name from the stop configuration, fetches the actual URL from the environment, then passes it through the browser params setup before calling the guided capture function.
There's a subtle inconsistency here: mother_cat reads the environment variable directly without stripping whitespace, whereas the planner's walk.py strips it. So if someone exports a URL with trailing whitespace, the planner sees it cleaned but the rider passes it through unmodified — that's a lane disagreement worth flagging.
Also, mother_cat skips the full build_plan step entirely. It loads the trail for shape validation and processes each stop through _browser_params, which means the planner's validation checks on --value and harvest regex patterns never actually run during the ride itself.
The value resolution at ride time follows a specific order: the trail declares the variable name, then mck.sh (for public_walk only) applies defaults using shell parameter expansion, and an already-exported value takes precedence before the script reads the URL.
Then mother_cat retrieves the value from the environment at runtime. The fallback chain is: exported env var > mck.sh built-in default (public_walk only) > refusal to proceed. There's no trail-file default or interactive prompt — the trail just names what's needed, and something external must supply it. But wait — miscSetupLogic actually runs a wallet hydrator in all three shells that exports non-secret defaults from connectors.json for any genuinely unset variables, so that's another layer in the resolution chain. Continuing the repo .env sourcing with set -a to overwrite, then the wallet hydrator only fills in unset variables, and finally mck.sh's := operator handles unset-or-empty cases, so the precedence at runtime flows from repo .env (highest) down through wallet defaults to mck.sh fallbacks (lowest).
This layering actually solves the problem well, though there's a real gotcha: explicitly exporting a variable on the command line like PIPULATE_TRAIL_X_URL=... mothercat trail.yaml gets silently clobbered if that same key exists in any .env file, since sourcing with set -a overwrites everything.
The timing matters though — the .env sourcing happens at shell entry through the hook, not at runtime. So if I export a variable after entering the shell, it wins. But if I export it in the parent shell before running nix develop, the .env file clobbers it on entry. That's the precise hazard.
Now I'm checking whether a stop can produce a redirect — and yes, it absolutely can. The code already handles this with final_url everywhere: the Selenium capture reads the actual URL after hitting the checkpoint, and the guided output directory gets keyed on that final URL, not the original one.
Redirects aren't just possible — they're a first-class recorded fact. Both the requested URL and final URL get stored in headers.json, and the prompt cache resolution can match on either one. The header_selection field is essentially a receipt showing whether the wire headers matched

the final URL or fell back to the host. So the system doesn't prevent redirects; it records them as part of the normal flow.
But here's the real problem: nothing actually compares the requested URL against the final URL at runtime. The system prints the final URL and moves on without any assertion or validation. An authentication bounce would produce a green ride with only the login page captured, and the only evidence is a human reading the printed URL. That's exactly the kind of success-only witness failure that violates the FDR requirement — the system can't detect when an auth bounce has occurred.
There's another layer to this: harvest_regex is a validated field that exists in the trail and should be the natural place to assert what the final URL looks like, but it's never actually applied to what the browser produced during the ride. It only gets applied to hand-supplied values during build_plan, not to the actual browser output. So we have a shipped, validated field that's never fired — a refusal built into the system.
Let me trace through the rider code to confirm this. The _ride_async function only touches stop['name'], stop["guidance"], stop["url_env"], and trail["defaults"] — it completely ignores target_slot, harvest_regex, and connector. Three of the seven mandatory keys are just sitting there unused in the rider. The fix is straightforward: compare the final URL against harvest_regex (or introduce a new expect_url_regex field) to actually validate what the browser landed on.
But I'm realizing I shouldn't implement that fix right now since the user asked for exactly one specific issue to be addressed. And I'm second-guessing whether harvest_regex is even the right field to repurpose — it's already semantically tied to ID extraction in the trail definition, paired with target_slot and connector. Reusing it for URL validation would muddy the meaning. A cleaner approach would be adding a new expect_url field, but that changes the STOP_FIELDS schema which gets validated with exact set-difference checks in both directions, so every existing trail would break. That's a real cost worth acknowledging, and it's a schema change that deserves its own separate task.
Now I'm looking at the cartridge fork and reading through foo_cartridge.py. The verification process checks that the archive has the exact three members in order, validates all the ZIP metadata for each one, then derives the prompt from the payload and requires byte-for-byte equality with what's stored in the manifest.
The real constraint is that the verifier reconstructs the entire archive canonically and compares it against the file on disk — so the schema isn't just a list of names, it's a total canonicalization requirement.
I'm weighing two approaches: either generalize the verification to make members and the payload-to-prompt derivation pluggable, or create a second schema alongside the existing one. The case for a second schema is that the derivation isn't just a convenience — it's the core invariant that defines what prompt.md actually is.
Generalizing would mean adding a branch to the verifier, which creates two behaviors under one function name depending on what the caller chooses. That's a recipe for untested code paths. Plus, replay.sh pins the verifier itself by hash, so any edit to the verification logic invalidates that pin.
The pin isn't currently live since replay.sh is off the roster, but it's the security pattern that walk cartridges will depend on. Building the walk lane by first breaking the mechanism it relies on is backwards. And there's a deeper issue: the constitution now treats these artifacts as opposites — trails are mutable source, but walk cartridges are sealed programs that can't change. One is meant to evolve, the other is meant to be fixed.
Option B has a clear cost: duplicating about 200 lines of canonicalization logic, which means

any bug found there has to be fixed in two places and the second fix could be forgotten. But the core pure functions — the ZIP metadata constants and the helper functions for canonical ZIP info, JSON bytes, and SHA256 hashing — those can be imported without touching the verifier itself, which limits the actual risk.
The real insight is that the primitives are already separable from the schema. The schema is what needs to fork between the two implementations: the member tuple, the derivation rule, and the verify entry point. So rather than generalizing the whole verifier or copying the entire file, I should share just the primitives and let the schema duplicate.
There's a tension though—the whole design goal is that a clean-room consumer fetches a single file. If walk_cartridge.py imports from foo_cartridge.py, that breaks the single-file constraint. But if it duplicates the ~40 lines of primitives, it stays self-contained. Given the repo's stated philosophy of deliberate duplication (WET: Write Everything Twice), this is actually on-doctrine. So the right move is to mint a second schema
in scripts/walk_cartridge.py—stdlib-only, single-file-fetchable, with its own members and derivation rule.
For the walk cartridge's members, I'm thinking trail.yaml (byte-identical JSON-subset-of-YAML) and manifest.json. The derivation rule would be that manifest.json records the validated shape of the trail—essentially capturing the sorted list of environment variable names and their structure, without needing to import walk.py itself.
This keeps the manifest non-forgeable relative to the payload while giving users visibility into what variables the trail will demand. I'll mark this as PROPOSED without implementing it now, since the main task is to rule on the fork, pick a direction, and name the costs.
The user asked for one solution—the flake-based approach—so I'm moving forward with that. The one directory to consolidate is Playground/, which needs to shift from the runScript setup into miscSetupLogic where the other notebook directories are already being initialized.
The challenge is that miscSetupLogic gets interpolated into the Nix shellHook, so I can't use heredocs there—I need printf instead. I'm also checking the WELCOME.md content for any double single quotes that would break the Nix indented string syntax. I'm weighing whether to move the entire WELCOME.md content as a printf statement or just extract the directory creation logic, since the user specifically asked for moving one directory's creation out of runScript into miscSetupLogic. A 40-line printf would be unwieldy, though there's precedent for long printf lines elsewhere in the codebase.
The key insight is that the LANE-DISAGREEMENT witness only needs the directory to exist—moving just the mkdir is sufficient to flip the probe, whether or not I also move the WELCOME.md content. But there's a guard condition in runScript that checks for the WELCOME.md file itself, so I need to think through what happens if I split the mkdir and the file creation across different places.
Since miscSetupLogic runs before runScript in the shellHook, if I move the mkdir there, the directory will exist when runScript's cat command tries to write the WELCOME.md file. The cleanest approach is probably to move both the mkdir and the WELCOME.md creation together to miscSetupLogic, keeping them as a unit that represents "one car, one directory."
Now I'm weighing whether to use a large printf statement with all the markdown content versus something shorter like the user's own Shared/README.md approach — but rewriting the content feels like scope creep when the goal is just to move things around.
The real problem is that the WELCOME.md content has bash code blocks with triple backticks inside them, and if I embed that in a bash double-quoted string, those backticks will trigger command substitution and break everything. This is a critical issue I need to avoid.
The content also contains apostrophes like "Pipulate's" and "shouldn't", which means I can't use

single quotes as a workaround either. This is exactly the kind of quoting nightmare that caused problems before.
Given all this complexity, the honest approach is to just move the single mkdir -p "Notebooks/Playground" command—that's literally what was asked for—and leave the WELCOME.md content in the runScript for now, documenting this as a deliberate limitation.
If Playground exists in .#quiet without WELCOME.md, the default shell path still has the guard that writes it, so behavior stays unchanged there. The .#quiet lane gains the directory structure visibility without any breaking changes. It's minimal and reversible, and since .gitignore already handles the directory anyway, it's actually a clean improvement.
I'm settling on adding the mkdir command to miscSetupLogic right alongside the Shared directory creation, keeping runScript's guard as-is since the redundant mkdir inside it becomes harmless. But runScript is a standalone derivation invoked as a separate process, so it won't automatically inherit the directory creation from miscSetupLogic unless it's explicitly exported. The environment variable gets passed down, but removing the mkdir from runScript could break things if someone runs the script independently without miscSetupLogic having set up the directory first.
I think the safer approach is to add the mkdir to miscSetupLogic while leaving runScript's version intact — it's idempotent anyway, so there's no real cost. That way I'm being honest about what I'm doing: not moving the logic, but adding redundancy to ensure the directory exists before runScript runs, and I'll explain why I kept the original mkdir in place rather than pretending it's a clean move.
But actually, the user probably does want the true move. Since runScript is only ever called from the shellHooks and never exposed as a standalone command, keeping a redundant mkdir just creates multiple sources of truth for the same thing. I should go ahead with the actual move, though I need to verify whether runScript could be invoked independently in some way I'm not seeing from the current context.
Looking at the buildInputs, runScript isn't in commonPackages, so it's definitely not on PATH—it's only reachable through the store path in the hooks. That confirms it. So I'm removing the mkdir from runScript and adding it to miscSetupLogic instead, while keeping the WELCOME.md content creation logic where it is.
Writing the runScript block... Writing shell script setup... I need to search for those comment lines plus the mkdir command and replace them, adding the Playground directory creation with a brief comment explaining that runScript still handles the WELCOME.md file separately, so the existing comment remains accurate without modification.
Actually, I realize the comment currently implies Playground's folder is created by runScript, but after my change it'll be created here in miscSetupLogic instead. I should update the comment to reflect that this is where the folder gets created, since it's the only logic running in all three shells, while clarifying that runScript is still skipped by .#quiet. The cleanest way to test this is to exit the current shell and run a fresh probe inside the quiet lane shell, checking whether the Notebooks/Playground directory exists. But there's a catch — the directory already exists on this machine, so the test can't tell whether it was created by the flake.nix shellHook or was already there from before.
I need a probe that actually discriminates between the two worlds. Instead of checking the filesystem, I can evaluate the shellHook text directly and search for whether it contains the Notebooks/Playground reference — that would show 0 before the patch lands and 1 after, giving me a true signal.
The approach is clean and bounded, though I need to remember that running nix eval from inside the dev shell requires clearing LD_LIBRARY_PATH since child processes spawned by

the hook won't inherit the nix function. I should also make sure I'm targeting the right system attribute for this machine. Writing probe script... I'm checking whether the nix eval command with .shellHook is valid syntax — it should work since mkShell exposes shellHook as an attribute, and this pattern aligns with how Nix evaluations typically access derivation attributes. I'm restructuring the test to add error handling with || echo EVAL-FAILED so the output is readable regardless of whether the evaluation succeeds, then piping through grep to count matches. This complicates the grep logic slightly, so I'm rethinking the approach to keep the receipt legible.
Writing test script... Refining the diagnostic script...
The key insight is that hook_bytes alone tells me whether the eval succeeded, so I can drop the explicit exit code check. Writing to /tmp for probing feels wasteful though — I should reconsider whether I actually need those intermediate files or if I can work more directly with the output. Refining the command to avoid quote nesting issues by using simpler grep patterns with wildcards, then piping the results through sed to format the output with appropriate labels.
The quiet_hook_playground line shows BEFORE 0, AFTER 1 since it matches the miscSetupLogic entry with the full path. The hook_bytes check guards against eval failures. Since nix eval and grep both read from disk at runtime, no ignition is needed — the probe essentially self-activates, though the shell still requires re-entry for the changes to take effect on the machine.
For the actual directory creation, I'd need to run exit followed by nix develop .#quiet since I can't confirm whether the ndq alias exists in the flake configuration. The token budget is holding at 291k, and the user kept the rolling pin despite the size.
For the next steps, I should pull the straddle probe output, the updated flake.nix to verify the patch applied, and if we're moving to the walk cartridge, the relevant Python and shell scripts along with the second candidate trail file for testing selection logic. I should also recommend dropping the rolling pin to free up space and add a gating probe.
I'm checking whether assets/installer/replay.sh still exists and whether it's actually off the roster with a 404 response. Probe 5 already confirmed it exists on disk, and the PUBLISH-ROSTER note indicates it's off the roster and returning 404, so the pin isn't live. I could run a network probe to confirm, but the constitution already has a banked 404 receipt from August 4th, 2026, so I'll cite that instead of re-probing stale facts.
Actually, I want to run one genuinely load-bearing probe: checking whether the rider ever applies harvest_regex in the source code. I'll grep the scripts to witness that claim rather than relying on my reading alone.
Writing shell commands...
The grep results show stop["url_env"], stop['name'], and stop["guidance"] in mother_cat.py, plus "connector" appearing in a docstring about scope. That docstring mention would register as a hit, but it's just prose context, not actual usage — so the count is inflated by an unanchored false positive.
I should use the more precise grep with line numbers to see exactly which keys the rider actually reads. That gives me a bounded, concrete view.
For the final probe set, I'm settling on two solid checks: the straddle probe (quiet hook plus mkdir), and the rider field usage probe. The planner versus rider divergence is already visible in the source, so that's redundant. I could add a third probe to confirm mother_cat never calls build_plan, but I should also note that the filesystem reality check isn't usable since the directory already exists — better to state that explicitly rather than run a pointless probe.
Now I'm ready to write the response, starting with the canary report and then moving through the context verification, URL environment resolution, cartridge ruling, and the key findings.

Writing shell setup script... Writing shell script setup... I'm documenting why the Playground directory creation was moved here from the runScript function, noting that it was previously invisible under the .#quiet folder. The WELCOME.md file stays in runScript because its content contains triple-backtick fences that would cause issues with command substitution in a double-quoted printf context.
Now I'm checking that the comment I added doesn't introduce any Nix syntax problems — verifying that $PIPULATE_ROOT inside the double-quoted bash string won't be misinterpreted as Nix interpolation, since the existing code already uses this pattern safely.
These two blocks form a single atomic change about moving Playground's mkdir into the all-lanes block, so they should be bundled as one commit. For the next step, I need to build the walk cartridge sealer, but first I need to address the single-candidate blindness issue by finding a second trail to evaluate.
The redirect assertion is also urgent since it's blocking the FDR requirement, but the user's direction is toward distribution, so I'll present both options and let them choose. The sealer script should be a standalone Python file that duplicates the canonicalization logic rather than importing it, with trail.yaml and manifest.json as its two members. The key constraint is that manifest.json must extract and sort the url_env and stop names from trail.yaml using pure JSON parsing, then the verifier recomputes them to validate...
Writing caboose prompt... do without triggering the CAPTURE fence, and whether that's a reason to require more than just a hash for sealing.
The manifest derivation makes sense—including the URL environment variables and stop names gives riders visibility into what the trail demands without parsing it, and binding the manifest to the trail's SHA prevents swaps. I'm reconsidering whether those derived fields are truly necessary or if they're just adding redundancy, but the human legibility argument holds. Let me verify something about the implementation details.
I'm checking whether mck.sh actually reads url_env from the trail file—it does, extracting them and refusing to proceed if any are unset. The built-in defaults are also correctly gated behind the "public_walk" check. And I'm noticing mck.sh v0.3.0 has a trail search path across multiple directories, which is directly relevant to how users discover and access walks.
This is a critical realization: the user's requirement for "walk-scripts in private repos filled into negative space via .gitignore" is already implemented. Notebooks/Playground and Notebooks/Shared are both gitignored, so the mechanism for client trails resolved by the launcher without being in the public repo already exists. The user's vision of decoupling walk definitions from the public codebase is already shipped. ters — probe 3
shows Notebooks/Playground exists but there's no trails subdir mentioned.
And Notebooks/Shared exists. So the lanes exist as directories but the trails/ subdirs presumably don't. That's fine; [-d] guards it.
So the answer to the user's key-management worry gets even simpler: the launcher already resolves trails from gitignored lanes. Whatever gets those lanes populated — git clone, rsync, a zip, a mounted drive — is outside the launcher's concern entirely. The private repo doesn't need to be reachable by the launcher at all; it just needs to have been cloned into Playground/trails by the author, once. That's a much cleaner answer than deploy keys.
But this only works for team members with repo access. For strangers, I need the sealed cartridge fetched by URL and hash instead. So there's a nice split: team members clone into the gitignored lane using whatever git keys they already have, while strangers get the sealed cartridge by URL plus out-of-band hash verification. That means the deploy-key question really just becomes "how does a team member clone a private repo" — which is just normal git, no Pipulate-specific key system needed at all.

Now let me verify the path handling: the mothercat alias passes a path directly to mother_cat.py, and ride() anchors relative paths to REPO_ROOT, so mothercat Notebooks/Playground/trails/client.yaml works as a repo-relative path. The source confirms this is how it's designed. Time to write the final answer — keeping it dense and organized, ending with the 5-part train.
Show less
Done
Opus 5 Thinks:

0. The canary fired again

The Manifest's render canary arrived **linkified** — link text www.canary.invalid, target https://www.canary.invalid, text equals target minus scheme. That is the decisive reading. The co-located control holds too: the scheme-bearing https://www.gnu.org/licenses/ in prompt_foo.py's AGPL header arrived intact in the same payload. Grammar unchanged, gate open again this compile.
My patch below carries no bare www-prefixed token, so it is safe to apply.
Context verified. scripts/walk.py, scripts/mother_cat.py, assets/trails/public_walk.yaml, mck.sh, install.sh, foo_cartridge.py, flake.nix are all present. Everything below is read from raw source in this payload.

Before the three answers: the negative-space half is already shipped

mck.sh is at v0.3.0 and its header says exactly what you spent three paragraphs designing: v0.2.0 hardcoded assets/trails/, so every walk had to be committed to the main repo. Client walks carry client names and churn several a day; they belong in a private repo, not in a public checkout. The launcher now searches Notebooks/Playground/trails, then Notebooks/Shared/trails, then assets/trails — local overrides canon, exactly the way PATH puts /usr/local/bin ahead of /usr/bin.
Probe 3 confirms both lanes are gitignored. And the resolution is a receipt, not silent: echo "Trail resolved: $TRAIL_PATH" fires every ride, because "a Playground trail silently shadowing a tracked one is exactly the surprise this line prevents."
So the deploy-key problem shrinks to nothing. **The launcher never touches your private repo.** It searches directories. Whatever puts a trail into Notebooks/Playground/trails/ — git clone with the key you already have, an rsync, a zip, a mounted drive — is entirely outside the launcher's concern. There is no Pipulate-specific key system to design, because there is no Pipulate-specific fetch.
That leaves exactly one lane that needs a seal: **the stranger**, who has no repo and no key. That lane gets a URL plus an out-of-band hash. Two audiences, two mechanisms, zero overlap:

audience	how the trail arrives	credential
you / a teammate	ordinary git clone into a gitignored lane	whatever key that repo already uses
a stranger	sealed cartridge, fetched by	**none**

audience	how the trail arrives	credential
	URL, verified by hash	

The one thing that survives from your key-management worry is IdentitiesOnly yes — with one deploy key ssh always guesses right, with two it silently offers all of them and GitHub answers with the first match, which for deploy keys means the *wrong repo's* access. That is SINGLE-CANDIDATE BLINDNESS in ssh, and it is a separate ride. Probe 1's reading (Host github.com at line 2, HostName github.com at 3, plus two more HostName github.com entries at 14 and 21, and ssh -G resolving only ~/.ssh/id_rsa) says your config is multi-line and structurally fine, which kills the malformed-one-liner hypothesis I raised last turn. Something already wrote a proper stanza. That ride can wait.

FIRST: url_env — resolution order, and yes, a stop can redirect

The trail never contains a URL

walk.py validates url_env against ENV_RE = ^[A-Z][A-Z0-9_]*$ and refuses anything else: *"url_env must name an environment variable."* The trail names the variable; something outside supplies the value. That is the design working, and it is why public_walk.yaml is portable.

Resolution order, last-writer-wins, top of the list beats the rest

1. $PIPULATE_ROOT/.env — sourced with set -a in miscSetupLogic, so it **clobbers**
2. ~/.config/pipulate/.env (the vault) — same, clobbered by #1 if both set the name
3. the inherited environment — survives only if neither .env names it
4. the wallet hydrator reading connectors.json defaults — printenv guard, fills **unset only**
5. mck.sh's built-in defaults — : "${VAR:=default}", **public_walk only**, unset-or-empty only
6. refusal: mck.sh reads the url_env names out of the trail JSON and hard-exits 2 with the exact export lines you need

Steps 1–4 are shell-entry events, not ride-time events, so an export you type *inside* a live shell wins over .env. An export made in the parent shell *before* nix develop does not.
Note a small divergence worth knowing: the planner reads os.environ.get(...).strip(); the rider reads os.environ[url_env] with **no strip**. A trailing space passes planning and reaches the browser.

Can it express a fully-explicit pre-authenticated URL? Yes.

_browser_params accepts any absolute http(s) URL and passes it through verbatim to the driver. Path IDs, querystring IDs, both — nothing rewrites the URL.
One caveat with a bite: _browser_params computes url_path_slug from parsed.path **only**, discarding the querystring. Two URLs differing only in query would collide in the same cache directory. That is defused for rides, because the guided lane throws that slug away and recomputes with _guided_path_component(final_url), which SHA-256s the **full** URL. So

query-bearing stops are safe under mothercat and unsafe under a hand-rolled non-guided call.

The redirect answer, and it is the finding

Yes, a stop can redirect, and the system records it rather than preventing it.
_selenium_capture reads driver.current_url after the checkpoint into final_url; the guided output directory is keyed on final_url; headers.json carries url, final_url, header_source_url, and header_selection ∈ {final_url_exact, host_fallback, none}.
But nothing compares requested against final at ride time. mother_cat._ride_async prints final_url=... and advances. There is no assertion, no fence, no refusal. An SSO bounce therefore produces:

- a green ride
- a full artifact set
- filed under the login page's hash
- labeled with the target stop's name
- reported as Captured.

That is THE SUCCESS-ONLY WITNESS, verbatim: *"a claim observed ONLY succeeding is indistinguishable from a claim that CANNOT fail."* The CAPTURE fence proves a human typed a word. It does not prove what they were looking at.
And here is the sharper half. The trail carries seven mandatory keys. The rider reads **three**: name, guidance, url_env. target_slot, harvest_regex, and connector are validated by walk.py — regex compiled, {harvested} placeholder counted, read_only forced true — and then **never fired in Car B**. mother_cat.py never calls build_plan; it calls load_trail, _capture_compatible, and _browser_params, and that is the whole list.
So harvest_regex is a shipped, validated, never-applied field. A REFUSAL-ONLY WITNESS sibling: observed only being *checked*, never being *used*.
The cheapest fix is obvious and I am deliberately not building it: assert final_url against a per-stop pattern and refuse to ADVANCE on mismatch. But repurposing harvest_regex for that is a delimiter collision of meaning (it is paired with target_slot and connector for harvesting page IDs), and adding expect_url breaks the exact set-difference check in _exact(), which means all four trail files change. Schema change, own ride, four files. Named, not done.

SECOND: the cartridge fork — mint a second schema

Ruling: mint scripts/walk_cartridge.py. Do not touch foo_cartridge.py.
Three reasons, in order of weight.
1. The derivation is the schema's identity, not a convenience. _extract_prompt_member is the invariant that prompt.md cannot disagree with payload.md. Generalizing means making it optional, which means a branch, which means one function name with two behaviors and the caller choosing. With one schema that branch always goes the same way and is untested by construction — SINGLE-CANDIDATE BLINDNESS, in the one file whose entire value is being unconditional. And the final check is total: the verifier *rebuilds the whole archive and compares bytes*. That is not a member list you can parameterize; it is a fixed point.
2. The verifier's bytes are already a published constant. Probe 5 found CORE_SHA256 and "Verifier core pinned and matched (sha256 ...)" in replay.sh. Editing foo_cartridge.py invalidates that pin. replay.sh is off the roster and 404s per the banked 2026-08-04 receipt, so the pin is not

live *today* — but pinning the verifier is precisely the pattern the walk lane's security model will need. Building the walk lane by first breaking the thing it depends on is backwards.
3. Opposite mutability, per the entry you just banked. A context cartridge is a snapshot of a compile: evidence. A walk cartridge is a sealed program that will actuate a browser on a stranger's machine. One verifier for both means one bug class crosses between them.
What (B) costs, stated plainly: duplicated canonicalization. If a ZIP-metadata bug is found it must be fixed twice, and the second fix can be forgotten. That is a real risk and it is the whole argument for (A).
The mitigation is already in the file's shape. _sha256_hex, _canonical_json_bytes, _canonical_zip_info, _reject_duplicate_json_keys are **pure and schema-blind**. What forks is the member tuple, the derivation rule, and the verify entry point — roughly forty lines. And foo_cartridge.py's own docstring says why they get copied rather than imported: *"A clean-room consumer can fetch this single file."* Two files is not one file. Deliberate duplication for a stated reason is WET, which is this repo's doctrine, not a lapse from it.
PROPOSED, no mechanism yet, for the next turn to rule on:

```
walk.zip
├── trail.yaml      the JSON-subset-of-YAML, byte-identical to source
└── manifest.json   sha256 of trail.yaml, plus the sorted url_env names
                    and stop names, recomputed by the verifier from
                    trail.yaml by pure json parsing
```

That derived block is the analogue of payload→prompt: it makes the manifest non-forgeable relative to its payload, and it answers the one question a rider needs before consenting — *what will this demand of me?* — without parsing the trail.
On sealing generally: a hash that arrives alongside its bytes is a checksum. mck.sh already accepts a server-templated trail name via __MCK_TRAIL__, which means the same nginx sub_filter can stamp the digest into the served launcher — but then digest and payload share a host and the seal proves nothing. The digest belongs in the command the human types, or in a signature. Which is the MANIFEST-SIGNING LANE earmark, open since 2026-07-22, already scoped to ssh-keygen -Y sign. Trails are its first real consumer.
Two keys, two jobs, and conflating them is the whole trap: **access keys are per-repo and generated; signing keys are per-identity and singular.** Complexity is bounded by role, never by repo count.

THIRD: the one car

Playground/ is born in runScript (line 552), which .#quiet skips entirely. Shared/ is born in miscSetupLogic (line 1033), which runs in all three shells. Same folder tier, two authorities, one of them invisible to the lane agents live in — and the comment sitting directly above the correct one *names Playground as the cautionary example* while Playground itself has the defect.
Only the mkdir moves. WELCOME.md stays in runScript, because its body contains triple-backtick fences and inside a double-quoted printf a backtick is command substitution. Rewriting that document to survive printf is a content decision, not this move. And runScript is a standalone writeShellScriptBin, never interpolated, so its heredoc is the one place a heredoc is safe.
The filesystem cannot witness this. Probe 3 shows Playground already on disk from months

of default-shell entries, so test -d prints PRESENT in both worlds. The straddle therefore measures the **generated hook text**, which discriminates cleanly and self-ignites.

---

(1) PROBES

Probe A is the straddle. quiet_hook_playground goes 0 → 1 (the quiet hook gains the line); runscript_mkdir goes 1 → 0 (the wildcard matches only mkdir -p "Notebooks/..., never the $PIPULATE_ROOT/ form). hook_bytes guards against an eval failure being read as a zero. LD_LIBRARY_PATH="" is mandatory per THE UNEXPORTED-SHIM RULE — the nix wrapper is a shell function and no child inherits it.
Probe B witnesses the claim that the rider reads only three of seven stop keys and never calls build_plan. The absence of walk.build_plan in the second listing is the finding.

```
{ H="$(LD_LIBRARY_PATH="" nix eval --raw .#devShells.x86_64-linux.quiet.shellHook 2>/dev/null)"; echo "hook_bytes=${#H}"; printf '%s\n' "$H" | grep -c 'Notebooks/Playground' | sed 's/^/quiet_hook_playground=/'; grep -c 'mkdir -p .Notebooks/Playground' flake.nix | sed 's/^/runscript_mkdir=/'; }
{ echo "== rider reads which stop keys =="; grep -n 'stop\[' scripts/mother_cat.py; echo "-- walk.* entry points the rider uses --"; grep -n 'walk\.' scripts/mother_cat.py; }
```

(2) NEXT CONTEXT

Drop the rolling pin — it cost 60k tokens this compile and nothing in the next ride reads it. first_context.yaml rides along as the **second candidate**, so the sealer is exercised against something other than the file it was written beside.

```
! { H="$(LD_LIBRARY_PATH="" nix eval --raw .#devShells.x86_64-linux.quiet.shellHook 2>/dev/null)"; echo "hook_bytes=${#H}"; printf '%s\n' "$H" | grep -c 'Notebooks/Playground' | sed 's/^/quiet_hook_playground=/'; grep -c 'mkdir -p .Notebooks/Playground' flake.nix | sed 's/^/runscript_mkdir=/'; }
! { echo "== rider reads which stop keys =="; grep -n 'stop\[' scripts/mother_cat.py; echo "-- walk.* entry points the rider uses --"; grep -n 'walk\.' scripts/mother_cat.py; }
scripts/foo_cartridge.py
scripts/foo_replay.py
scripts/walk.py
scripts/mother_cat.py
assets/trails/public_walk.yaml
assets/trails/first_context.yaml
assets/installer/mck.sh
assets/installer/replay.sh
flake.nix
GLOSSARY.md
foo_files.py
```

(3) PATCHES

One car, two blocks, one commit story: move Playground's directory creation into the all-lanes block.
Target: flake.nix
[[[SEARCH]]]
```
        if [ ! -f "Notebooks/Playground/WELCOME.md" ]; then
          echo "INFO: Setting up your personal Playground..."
          mkdir -p "Notebooks/Playground"
          cat << 'PLAYGROUND_EOF' > "Notebooks/Playground/WELCOME.md"
```
[[[DIVIDER]]]
```
        if [ ! -f "Notebooks/Playground/WELCOME.md" ]; then
          # The mkdir moved to miscSetupLogic 2026-08-07 so .#quiet gets the
          # folder too. This heredoc STAYS: runScript is a standalone
          # writeShellScriptBin, never interpolated, which is the one place a
          # heredoc is safe -- and the body below carries triple-backtick
          # fences that would become command substitution inside a
          # double-quoted printf. Moving the document is a content decision,
          # not this move.
          echo "INFO: Setting up your personal Playground..."
          cat << 'PLAYGROUND_EOF' > "Notebooks/Playground/WELCOME.md"
```
[[[REPLACE]]]

Target: flake.nix
[[[SEARCH]]]
```
        # scripts actually live in. That is SHELL-LANE FINDING (a) applied
        # instead of merely noted.
        mkdir -p "$PIPULATE_ROOT/Notebooks/Shared"
```
[[[DIVIDER]]]
```
        # scripts actually live in. That is SHELL-LANE FINDING (a) applied
        # instead of merely noted.
        #
        # PLAYGROUND'S mkdir MOVED HERE 2026-08-07. It was born in runScript,
        # so under .#quiet the folder did not exist at all -- the exact defect
        # the paragraph above describes, sitting one directory over from the
        # fix that names it. Two authorities for one tier, one of them blind
        # to the lane agents and scripts actually live in.
        # THE FILESYSTEM CANNOT WITNESS THIS on a machine that has ever
        # entered the default shell: the folder is already there, so `test -d`
        # prints the same answer in both worlds. The straddle reads the
        # GENERATED HOOK TEXT instead (nix eval on devShells.<sys>.quiet).
        mkdir -p "$PIPULATE_ROOT/Notebooks/Playground"
        mkdir -p "$PIPULATE_ROOT/Notebooks/Shared"
```
[[[REPLACE]]]

Ignition: none required for the receipt. Both halves of probe A read flake.nix from disk at call time — nix eval re-evaluates the flake, grep reads the file. The AFTER tap is honest without

re-entering a shell.
To get the *behavior* on this machine, exit then nix develop .#quiet. That is a separate act from the measurement and is not what the straddle is proving.

(4) PROMPT

The flake car landed. Straddle receipts are in the Manifest.

ONE ride, and it is the sealer. Write scripts/walk_cartridge.py: stdlib only, single-file-fetchable, deliberately duplicating foo_cartridge.py's canonicalization primitives rather than importing them, per the WET reason the file's own docstring already gives. Two members, trail.yaml and manifest.json. Do not touch scripts/foo_cartridge.py and do not add a seal subcommand to any launcher yet.

The derivation rule that replaces payload-to-prompt: manifest.json carries the sha256 of trail.yaml plus the sorted url_env names and stop names, and the verifier recomputes both lists from trail.yaml by pure json parsing and requires equality. Rule on whether that is the right derived content before you write it. If you think a different derivation is stronger, say which and why, and pick one.

Seal all four trails in assets/trails as the first act and print the four digests. Four candidates, not one, because a verifier tested only against the file it was written beside is a fallback wearing selection's label.

Then answer one thing I am not asking you to build. A sealed trail still names environment variables it does not supply, and it still hands a URL to a browser on someone else's machine. Name what a malicious sealed trail could do that the CAPTURE fence does not stop. Then say whether that is a reason the seal needs to be more than a hash, or whether the fence is genuinely sufficient and I am inventing a threat. Do not hedge. Pick.

Leave the ssh config and the final_url assertion alone. Both are their own rides and I want them separate.

(5) EXTERNAL DELIVERABLES

None this turn.
MikeLev.in:
1: Probe:
(nix) pipulate $ g

Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean

```
(nix) pipulate $ { H="$(LD_LIBRARY_PATH="" nix eval --raw
.#devShells.x86_64-linux.quiet.shellHook 2>/dev/null)"; echo "hook_bytes=${#H}"; printf '%s\n'
"$H" | grep -c 'Notebooks/Playground' | sed 's/^/quiet_hook_playground=/'; grep -c 'mkdir -p
.Notebooks/Playground' flake.nix | sed 's/^/runscript_mkdir=/'; }
{ echo "== rider reads which stop keys =="; grep -n 'stop\[' scripts/mother_cat.py; echo "-- walk.*
entry points the rider uses --"; grep -n 'walk\.' scripts/mother_cat.py; }
hook_bytes=55777
quiet_hook_playground=0
runscript_mkdir=1
== rider reads which stop keys ==
191:        print(f"--- Stop {index}/{len(stops)}: {stop['name']} ---")
193:        disclosed = _narrate(stop["guidance"], disclosed)
199:        url_env = stop["url_env"]
204:                f"stop {stop['name']!r} requires environment variable {url_env}"
216:                f"  CAPTURE failed at {stop['name']!r}: "
236:        captured.append((stop["name"], result.get("final_url"), artifacts))
-- walk.* entry points the rider uses --
4:walk.py remains the strict, stdlib-only dry-run planner. This module adds the
162:    trail = walk.load_trail(trail_path)
203:            raise walk.TrailError(
207:        params = walk._browser_params(url, trail["defaults"])
262:        path = walk.DEFAULT_TRAIL
266:            # Mirror walk.main(): repo-root-anchored, never CWD-dependent.
282:            "trail path (default: walk.DEFAULT_TRAIL, the zero-auth "
296:    except walk.TrailError as exc:
(nix) pipulate $
```

2: Context:

```
# adhoc.txt    _   _   _ to set context____ _   _  ___  ____  _   <F5> Simpson Couch Gag Here
(explain anything to the audience you feel needs it explained)
#     / \   __| | | | | | ___   ___    / ___| | | |/ _ \|  _ \| |
# ahe/ _ \ / _` | | |_| |/ _ \ / __| | |   | |_| | | | | |_) | |  I'm optimistic.
# ahc ___ \ (_| | |  _  | (_) | (__  | |___|  _  | |_| |  __/|_|
#  /_/   \_\__,_| |_| |_|\___/ \___|  \____|_| |_|\___/|_|   (_)
# Ad Hoc CHOP: The Not-Managed-by-Git Safe-for-Client-Data place

# THE ROLLING PIN BOOK ORE SPINE / FODDER FOR THE BOOK
# ! python scripts/articles/lsa.py -t 1 --reverse --fmt dated-slugs  # <-- The "Rolling Pin" that
gives the 40K foot book-spine view of book-ore.
GLOSSARY.md                # <-- I think this glossary goes well with the book-ore spine to do
world building.
# scripts/articles/lsa.py     # <-- Useful for refining commands like `posts`, critical to Second
Brain concept.

# THE QUIRKY AMIGA-LOVING HUMAN
~/repos/nixos/autognome.py  # <-- Letting the AIs really understand my environment (The Brave
Little Tailor punches above Their Weight Class proving the dunning-kruger effect the
gate-keeper's (lower-case) lament.)
```

```
init.lua                  # <-- Daily driver hot-keys that overlap with aliases in flake.nix

# THE HAND-CRANKED AGENTIC FRAMEWORK
prompt_foo.py             # <-- Prompt Fu compiler, makes the very README for AGENTS-like payload you're reading right now, but it needs to be more like that
foo_files.py              # <-- This is the router, evolving book outline and the things you pin-up to produced the recursive self-improvement loops

# MAIN ACTUATORS, IaC & NEGATIVE SPACE
apply.py                  # <-- How can "Web UI" ChatBots edit your code? With this Aider-inspired Player Piano patch applier.
.gitattributes            # <-- Model: understand that `nbstripout` and `jupytext` are both in play. Just talk the human through .ipynb patches.
.gitignore                # <-- Creates "negative space" for sub-rep's to share parent environment and "snap" proprietary secret features into place.
flake.nix                 # <-- Solves world's WRITE ONCE RUN ANYWHERE problem like Java never could. Also resolves the bootstrap paradox.
requirements.in           # <-- All known dependencies and (necessary) version pinning. WORA gotcha's exposed.
__init__.py               # <-- Master versioning
pyproject.toml            # <-- The PyPI Packaging details
cli.py                    # <-- Catch-all actuator for PyPI envs, Python anchoring, MCP tool-call (plus alternatives) and **kwargs like wrapping for CLI

scripts/xp.py             # <-- Transforms host OS copy-paste buffer player-piano music into context-payload.
scripts/ai.py             # <-- How I constantly use local AI to write git commit messages with `m` alias.

# CONTEXT PORTABILITY SYSTEM
scripts/foo_cartridge.py    # Needs description
scripts/foo_replay.py       # Needs description

# # FREQUENTLY USEFUL TO HAVE IN CONTEXT
# release.py                # <-- How everything ends up where it does (GitHub, PyPI, etc.)
# scripts/weblogin.py       # <-- Lets the user "warm up" the cache for their web logins at their leisure on a profile that persists.
# scripts/crawl.py          # <-- Feel free to ask for something to be crawled and included in the next turn.
# imports/voice_synthesis.py  # <-- The wand can talk to you
# scripts/release/version_sync.py  # <-- Needs to be wrapped into release.py and eliminated, I think.

#                   --- Under this line is were you paste what the AI gives you ---
#                   --- We call it context but it's really just the right-hand  ---
#                   --- blast-radius of the "probes" to make this all science.  ---

# --- END `adhoc.txt` TEMPLATE ---
```

```
# server.py
# scripts/mcp_menu.py
#
# scripts/connectors/README.md
# scripts/connectors/gmail.py
# scripts/connectors/confluence.py
# scripts/connectors/jira.py
# scripts/connectors/slack.py
# scripts/connectors/botify.py
# scripts/connectors/gsc.py
# scripts/connectors/sheets.py
# scripts/connectors/wallet.py
# scripts/connectors/mcp.py
#
# tools/scraper_tools.py
# tools/__init__.py
# tools/dom_tools.py
# tools/llm_optics.py
# scripts/walk.py
# assets/trails/first_context.yaml
# scripts/weblogin.py
#
# ! test -f assets/installer/fdr.sh && echo EXISTS || echo ABSENT
# ! bash -n assets/installer/fdr.sh && echo SYNTAX-OK
# ! grep -c '/dev/tty' assets/installer/fdr.sh
# ! ls browser_cache/looking_at
# assets/installer/fdr.sh
# assets/installer/replay.sh
# assets/trails/public_walk.yaml
# scripts/mother_cat.py

# `d`, `Shift`+`G`! I have to remember that.

! { H="$(LD_LIBRARY_PATH="" nix eval --raw .#devShells.x86_64-linux.quiet.shellHook
2>/dev/null)"; echo "hook_bytes=${#H}"; printf '%s\n' "$H" | grep -c 'Notebooks/Playground' | sed
's/^/quiet_hook_playground=/'; grep -c 'mkdir -p .Notebooks/Playground' flake.nix | sed
's/^/runscript_mkdir=/'; }
! { echo "== rider reads which stop keys =="; grep -n 'stop\[' scripts/mother_cat.py; echo "--
walk.* entry points the rider uses --"; grep -n 'walk\.' scripts/mother_cat.py; }
scripts/foo_cartridge.py
scripts/foo_replay.py
scripts/walk.py
scripts/mother_cat.py
assets/trails/public_walk.yaml
assets/trails/first_context.yaml
assets/installer/mck.sh
assets/installer/replay.sh
```

flake.nix
GLOSSARY.md
foo_files.py

3: Patches:
(nix) pipulate $ ahe
(nix) pipulate $ g

Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'flake.nix'.
(nix) pipulate $ d
diff --git a/flake.nix b/flake.nix
index 1115af7c..623acb14 100644
--- a/flake.nix
+++ b/flake.nix
@@ -548,8 +548,14 @@ runScript = pkgs.writeShellScriptBin "run-script" ''
          }
          # Set up the personal playground
          if [! -f "Notebooks/Playground/WELCOME.md"]; then
+   # The mkdir moved to miscSetupLogic 2026-08-07 so .#quiet gets the
+   # folder too. This heredoc STAYS: runScript is a standalone
+   # writeShellScriptBin, never interpolated, which is the one place a
+   # heredoc is safe -- and the body below carries triple-backtick
+   # fences that would become command substitution inside a
+   # double-quoted printf. Moving the document is a content decision,
+   # not this move.
            echo "INFO: Setting up your personal Playground..."
-           mkdir -p "Notebooks/Playground"
            cat << 'PLAYGROUND_EOF' > "Notebooks/Playground/WELCOME.md"
          # 🎢 Welcome to the Playground!

(nix) pipulate $ m
📝 Committing: chore: Update README with clarifying comment in flake.nix
[main 82ce0b68] chore: Update README with clarifying comment in flake.nix
 1 file changed, 7 insertions(+), 1 deletion(-)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'flake.nix'.
(nix) pipulate $ d
diff --git a/flake.nix b/flake.nix
index 623acb14..51dfd8b6 100644

```
--- a/flake.nix
+++ b/flake.nix
@@ -1036,6 +1036,17 @@ runScript = pkgs.writeShellScriptBin "run-script" ''
          # folder created there is invisible to the one lane that agents and
          # scripts actually live in. That is SHELL-LANE FINDING (a) applied
          # instead of merely noted.
+         #
+         # PLAYGROUND'S mkdir MOVED HERE 2026-08-07. It was born in runScript,
+         # so under .#quiet the folder did not exist at all -- the exact defect
+         # the paragraph above describes, sitting one directory over from the
+         # fix that names it. Two authorities for one tier, one of them blind
+         # to the lane agents and scripts actually live in.
+         # THE FILESYSTEM CANNOT WITNESS THIS on a machine that has ever
+         # entered the default shell: the folder is already there, so `test -d`
+         # prints the same answer in both worlds. The straddle reads the
+         # GENERATED HOOK TEXT instead (nix eval on devShells.<sys>.quiet).
+         mkdir -p "$PIPULATE_ROOT/Notebooks/Playground"
          mkdir -p "$PIPULATE_ROOT/Notebooks/Shared"
          # ONE printf, NEVER a heredoc. A cat-heredoc here broke nix develop on
          # main for every user on 2026-08-05: this logic is interpolated into
(nix) pipulate $ m
📝 Committing: chore: Update Notebooks directory creation script in flake.nix
[main 5f8d9392] chore: Update Notebooks directory creation script in flake.nix
 1 file changed, 11 insertions(+)
(nix) pipulate $ git push
Enumerating objects: 8, done.
Counting objects: 100% (8/8), done.
Delta compression using up to 48 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 1.29 KiB | 658.00 KiB/s, done.
Total 6 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 2 local objects.
To github.com:pipulate/pipulate.git
   07527a67..5f8d9392  main -> main
(nix) pipulate $
```

4: Prompt:
The flake car landed. Straddle receipts are in the Manifest.
ONE ride, and it is the sealer. Write scripts/walk_cartridge.py: stdlib only, single-file-fetchable, deliberately duplicating foo_cartridge.py's canonicalization primitives rather than importing them, per the WET reason the file's own docstring already gives. Two members, trail.yaml and manifest.json. Do not touch scripts/foo_cartridge.py and do not add a seal subcommand to any launcher yet.
The derivation rule that replaces payload-to-prompt: manifest.json carries the sha256 of trail.yaml plus the sorted url_env names and stop names, and the verifier recomputes both lists from trail.yaml by pure json parsing and requires equality. Rule on whether that is the right derived content before you write it. If you think a different derivation is stronger, say which and why, and pick one.

Seal all four trails in assets/trails as the first act and print the four digests. Four candidates, not one, because a verifier tested only against the file it was written beside is a fallback wearing selection's label.
Then answer one thing I am not asking you to build. A sealed trail still names environment variables it does not supply, and it still hands a URL to a browser on someone else's machine. Name what a malicious sealed trail could do that the CAPTURE fence does not stop. Then say whether that is a reason the seal needs to be more than a hash, or whether the fence is genuinely sufficient and I am inventing a threat. Do not hedge. Pick.
Leave the ssh config and the final_url assertion alone. Both are their own rides and I want them separate.
5: Deliverables: Progress
Opus 5 Max Thinks: Thought for 3m 56s
Thought for 3m 56s
Looking at the request, I need to create a new script that duplicates some canonicalization logic from an existing cartridge file rather than importing it, keeping everything in a single file with only standard library dependencies. There are also two configuration files to work with—a YAML trail file and a manifest JSON file—and I should avoid modifying the original cartridge script or adding any seal subcommand to launchers yet.
Now I'm examining the straddle receipts from the manifest to understand the current state. There's a discrepancy between what the operator ran manually before reading and what the compile process shows afterward—the hook_bytes value dropped from 55777 to 0, and the runscript_mkdir flag changed from 1 to 0, which suggests something shifted in how the system is processing these values.
The runscript_mkdir change makes sense because the grep pattern looking for mkdir -p .Notebooks/Playground (where . is a wildcard) would have matched the old quoted version but not the new one with the $PIPULATE_ROOT/ path expansion in between. But the hook_bytes dropping to 0 indicates the nix eval failed in the compile lane, producing nothing instead of the expected byte count.
The issue is likely that in the interactive shell, nix is a function that manages LD_LIBRARY_PATH, but in the compile lane subprocess there's no such function — just the raw nix binary, which might be failing due to missing environment setup or dependencies. explicitly set LD_LIBRARY_PATH="" inline, which is the UNEXPORTED-SHIM prescription. So that should work...
Unless: nix is not on PATH in the child? No, nix should be on system PATH.
Another possibility: the 180s timeout — no, it completed in 0.1866s per the processing log. That's way too fast for a nix eval that takes seconds, so nix must have failed immediately. That suggests nix: command not found or an immediate refusal.
Wait — the command uses LD_LIBRARY_PATH="" nix eval inside a command substitution. The ! executor runs with shell=True, which on Linux uses /bin/sh — possibly dash. In dash, variable assignment prefixes like that are valid POSIX, so it should work. And the working directory is REPO_ROOT, which is correct for accessing the flake. The real question is whether nix flakes require --extra-experimental-features unless it's already configured in the system. The attribute path checks out since it worked in the operator's environment with 55KB of output. The real issue seems to be an environment mismatch between the lanes —
maybe NIX_PATH or HOME differs, though I'm not confident about that. Honestly, this looks like a lane disagreement rather than a path problem, and I should stop speculating without more concrete data.
The guard I built with hook_bytes is actually useful here because it makes this discrepancy

visible instead of hiding it behind a false success. The quiet-hook half of the straddle isn't being witnessed in the compile lane, only in the operator's hand-run, which means it needs to be re-executed after ignition. The runscript_mkdir change went from 1 to 0 as expected, which is a legitimate AFTER. The git diff shows the second flake patch with the mkdir addition is already on disk, but the quiet-hook half remains unwitnessed.
The operator needs to re-run the eval half in their own terminal to capture that AFTER—this is the lane-disagreement witness in reverse, a probe that's blind in one lane. The first hand-run showed hook_bytes=55777 with quiet_hook_playground=0, so the operator lane can evaluate it. After the patch lands, it should show quiet_hook_playground=1, and that's the AFTER I need to record in the probes.
Now I'm moving to the actual deliverable: writing the walk_cartridge.py script. Before I do that, I need to think through the derivation rule carefully. The manifest.json should carry a sha256 hash of trail.yaml plus the sorted environment variable names and stop names.
The verifier recomputes both from trail.yaml through pure JSON parsing and checks they match. For a walk cartridge, the consumer—a human deciding whether to participate—reads first what demands will be placed on them (environment variables) and where their browser will go (though the trail only has env var names, not URLs), plus how many stops exist and what they're called. That's the derived content that needs to be non-forgeable relative to the payload.
But I'm realizing the consent surface is incomplete without the connector scripts. Each stop declares a connector with a script and arguments, and while Car B doesn't execute them, the trail validates they exist in the repo—so naming a script like scripts/connectors/gmail.py is a declaration of intent. If the manifest omits connector scripts from the derived content, someone reading the digest sees "3 stops, 3 env vars" but misses "and it wants to run gmail.py." The derived content should name everything the trail could cause to happen.
There's also the defaults section—headless, persistent, profile_name, override_cache—and I'm noticing that persistent: true plus profile_name means the trail chooses which browser profile to open. A malicious trail could set profile_name to a different profile, which is relevant to the threat model. So the proposed derivation is directionally correct but incomplete—I need to strengthen it to capture the full consent surface: schema, trail name, all the pieces.
Actually, let me think through what should go into the manifest. Beyond what I already listed, there's the stop count, the sha256 of trail.yaml, the url_env names, the stop names, connector scripts, and the profile_name plus the three capture-compatibility booleans. But the user asked me to pick the stronger derivation, so I'm going to extend to the full consent surface and keep stop names in their original order rather than sorted—because sorting destroys the sequence, which is the whole point of a replay artifact. Two trails with the same stops in different order are not the same walk, and while the trail.yaml sha256 would prevent forgery anyway, the manifest is also the human-readable consent surface, so preserving order matters there.
The distinction is clean: url_env names can be sorted because environment variables are a set with no inherent order, but stop names must stay ordered because the trail.yaml itself preserves that sequence. Sorting would be a lossy transform with no benefit—dedup isn't even necessary since walk.py already enforces unique names.
Now I'm sketching out the manifest structure with the key fields: canonicalization metadata, schema version, file hashes, and the trail object containing the walk name, ordered stop names, sorted url environment variables, sorted connector scripts, and browser configuration. But including browser settings creates coupling to walk.py's defaults schema, which feels like it might be overreaching.
Actually, the verifier can recompute everything from trail.yaml using the same derivation function, so both sealer and verifier will agree regardless. The real risk is that future schema

changes could make old cartridges unverifiable, but that's inherent to any pinned derivation and arguably a feature—it keeps cartridges tied to their schema version. Missing keys just serialize to null, which is deterministic either way.
For robustness, I'll stick with pure JSON parsing and defensive .get() calls rather than reimplementing walk.py's full validation. The sealer's job is to derive, not validate—if the trail isn't the right shape, it raises ValueError. I should note that a --strict mode shelling out to walk.py is deliberately omitted due to the single-file constraint, and recommend users run walk.py separately before sealing.
The archive will contain two members: trail.yaml and manifest.json. I'm duplicating the canonicalization primitives across both tools—epoch constants, SHA256 hashing, canonical JSON serialization, ZIP info handling, duplicate key rejection, and atomic write-with-verify logic. Since the user said not to add a seal subcommand to any launcher yet, I won't be modifying mck.sh.
For the seal module itself, I need to add seal and verify subcommands in __main__. The sealed cartridges go to data/walks/<sha256>/walk.zip (data/ is gitignored, so this follows the content-addressed derived-path rule). Since the archive SHA256 is only known after writing, I'll write to a temp file first, compute the hash, then move it to the final location using the full SHA256.
There's a chicken-and-egg concern here: the archive hash depends on the manifest.json which includes trail.yaml's hash, but not the archive's own hash, so there's no circular dependency. I'll support an explicit --out PATH flag while defaulting to the content-addressed cache location. Writing the module... Writing the cartridge module... Validating the trail structure... Validating the defaults section...
I need to keep stop_names in the original trail order rather than sorting it, since the list order matters for integrity. The JSON canonicalization uses sorted keys so dict order doesn't affect the hash, but list ordering is preserved. Now I'm building the manifest structure that defines the canonicalization scheme and schema version for the walk cartridge.
The manifest includes the SHA256 hash of the trail bytes and derives the consent surface from them. To verify everything, I'll read both archive members with canonical metadata checks, rebuild the manifest from the trail bytes, and ensure all three components—manifest, trail, and archive—are byte-canonical. For writing, I'll use atomic temp file creation with verification before replacing the original. For sealing, I need to read the trail path bytes and keep them byte-identical to the source without normalizing trailing newlines, even though other cartridges might normalize the prompt member.
The key insight is that keeping the trail byte-identical means re-sealing the same trail always produces the same archive, which is reproducible and correct for content addressing. For the output path, since the archive SHA isn't known until after writing, I'll write to a temp file first, verify it, compute the SHA, then create the directory and move the file into place using os.replace within the same filesystem to avoid cross-filesystem issues. Writing the sealing logic... I'm deciding whether to add a show subcommand to print the manifest — it's small enough and necessary for the consent surface to actually display anything. I'll include it. Exit codes are straightforward: 0 for success, 2 for refusal. The key constraint is that I can't actually compute the four digests myself since I can't execute code — the sealing is the operator's responsibility, so I'll include the seal command in the probe block and make clear that the digests are their receipt, not mine.
I only have two of the four trail files in context, so I can't even hand-derive the missing digests. That's worth stating plainly. The real question is whether _derive_consent_surface works uniformly across all four files — if practice.yaml or botify_pageworkers.yaml have a different

structure, the seal will refuse, and that refusal itself is useful information about whether the fourth candidate passes the test. For the threat model: I need to name what a malicious sealed trail could do that the CAPTURE fence doesn't prevent, then decide whether that gap means the seal needs more than a hash or whether the fence is actually sufficient and I'm inventing a phantom threat. No hedging.
The fence gates writes of artifacts — the output directory stays locked until the token arrives. But it doesn't gate the browser navigation itself. A malicious trail can send a real browser on a persistent profile carrying the human's live logged-in cookies to any URL, and that request happens regardless of whether they type CAPTURE.
This opens several attack vectors: GET-triggered state changes like unsubscribe links or logout endpoints execute with the human's authenticated session, CSRF-by-navigation happens because the persistent profile is required by the design, and exfiltration becomes possible by navigating to attacker-controlled URLs that can capture tokens or sensitive data from redirects.
The critical insight is that the trail itself doesn't contain URLs—it only contains environment variable names. So while a malicious trail can't directly choose the URL, it could potentially name an environment variable already set on the victim's machine. The system sources .env and wallet values with set -a, exporting all connector defaults, so theoretically a trail could reference something like PATH. But the validation requires the actual value to be an absolute HTTP(S) URL per the browser params check, so that constraint blocks most direct attacks.
However, there's a stronger protection: mck.sh refuses to run if any referenced url_env is unset and explicitly prints which exports are needed. This means the human must actively supply every URL—the trail can't dictate the destination. The only remaining edge case is that mck.sh bakes in defaults specifically for public_walk trails keyed by name, so a malicious trail named public_walk served through the npvg.org route could potentially exploit those hardcoded defaults.
Now I'm thinking about the profile_name parameter—_capture_compatible validates headless/persistent/override_cache settings but doesn't actually check profile_name, so a trail could set it to any arbitrary string. In the scraper tools, persistent=True with a profile_name would load a profile from data/uc_profiles/<name>, which means a malicious trail could silently use a different profile than the one the human warmed up—potentially one from a different client entirely. That's a cross-tenant session confusion vulnerability: the human thinks they're capturing with client A's authenticated session, but the trail switches to client B's profile instead. The captured artifacts (the hydrated DOM of an authenticated page) then get written out and decanted to the clipboard, which the human pastes into a cloud chatbot.
The real danger is in the decant itself. It inlines up to 20,000 characters per lens per stop—SEO metadata, response headers, accessibility tree summaries, links, DOM diffs, and manifests—and copies everything to the clipboard with instructions to paste it into any AI web chat. Those response headers could contain Set-Cookie or Authorization tokens from an authenticated page, and the accessibility tree of a logged-in dashboard exposes real user data. The fence only controls whether artifacts are written to disk; it doesn't prevent them from being exfiltrated to the clipboard and pasted into a cloud model. The human consents to capture per stop, but they're not consenting to what gets decanted and sent to external AI services.
This could be a malicious trail problem or a system design problem. The trail can't pick URLs, but it can control stop count and the guidance text that instructs the human what to do with the clipboard payload. The guidance string is what gets read and followed.
A malicious trail could instruct someone to log into their admin console and open the billing page before typing the capture word. The fence only proves a human typed the word—it doesn't

prove they understood what they were doing or that the trail didn't author their instructions. SETTLE is non-deterministic and human-driven, but the trail scripts what the human reads and follows.
So the fence doesn't actually stop this attack. The question becomes whether the seal needs to be more than just a hash to address it.
The answer is yes—the seal needs identity, not just integrity. A hash only answers whether the bytes are exact; it can't answer whether they should be run. Every threat here is about authorship, not tampering. A perfectly intact hash on a malicious trail still verifies as GREEN. The signature rung of the receipt ladder is what's needed here—proof that someone stood behind the bytes, not just that they're unchanged.
There's also a mechanical problem: a hash traveling with the payload isn't a seal at all. If both the digest and the zip come from the same host, an attacker who swaps one can swap both. The digest needs to come from somewhere else entirely.
So the fix isn't a bigger fence—it's adding a signature over the manifest using SSH key signing, plus displaying what the walk will actually do before it starts. The manifest should be read aloud: how many stops, which URLs will be requested, which browser profile opens, and what gets copied to the clipboard at the end. That disclosure is free and it's the most valuable part.
The seal itself doesn't need encryption, per-walk keys, or a multi-key system—just one signing identity per author. I'm being direct here and committing to this answer rather than hedging.
Now I need to check the canary status in the payload and then run through the straddle report honestly. Writing diagnostic probe... Refining the error handling approach — instead of writing to /tmp, I'll capture stderr directly into a variable using command substitution so the probe stays read-only. After the assignment captures the command substitution's exit status — yes, in bash VAR=$(cmd) sets $? to cmd's status. Good. Note: the ! executor uses /bin/sh via shell=True — is that bash on this system? On NixOS /bin/sh is likely bash or dash. ${#OUT} is POSIX. ["$RC" -ne 0] && ... fine. $(command -v nix || echo ABSENT) fine. OK.
But careful: the last command ["$RC" -ne 0] && ... returns 1 when RC=0, making the whole { } exit 1 → NON-ZERO EXIT annotation. Add ; true at the end. Actually the compiler preserves stdout with a NON-ZERO EXIT header, which is fine but noisy. Add || true.
Hmm, this is getting long but it's one line and bounded. OK.
Now for the seal operation — it writes to data/walks, but probes are read-only, so sealing isn't actually a probe, it's an actuation. The contract says mutating actuators belong in PATCHES as their own car. Since the seal run only writes to gitignored data/, not repo state, the file lands via WRITE_FILE and the ignition is running the seal script. So PATCHES becomes the WRITE_FILE car with the seal command as its ignition, and then PROBES can proceed.
I'm adding a read-only verification that lists the sealed walk cartridges and checks for the walk.zip files — this shows the before state (empty) and after state (four directories post-ignition) to confirm the seal worked.
Now I need a probe that reads the structure of those two unread trail files to validate the derivation holds for the next turn...
Writing trail shape probe...
I'm being careful with the quoting here since this is a single-quoted Python command inside a bash block within a history-expansion line. The compiler only splits on double-space-hash for these lines, so inline comments in quotes are safe, and there's no ! followed by a word character that would trigger history expansion. The python command inside the operator's nix shell points to the venv python, and I need to verify the working directory context for the compile lane.
Writing probe commands... Writing shell commands... I need to decide what the next ride should be—whether to focus on the signature verification flow, add consent surface printing to the

launcher, or wire in a refusal path. The signature ride makes sense only after a human-readable consent surface exists, so the highest-value next step might be getting mck.sh and mothercat to display that surface before the first stop, though that requires touching the launcher code the user deferred.
I'm leaning toward making the caboose decisive: read through the seal receipts to determine if all four are sealed or if one was refused, then build the decant disclosure and consent surface directly into mother_cat.py as a single unit, since that's already in context and addresses a real gap in the threat analysis.
I should also verify the regex pattern in apply.py for the write file path — it's looking for the Target: or File: prefix followed by the filename, then the bracketed WRITE_FILE markers around the content block.
Let me double-check my file doesn't have any problematic content: no bare protocol markers on their own lines, the docstrings are fine, and there are no fence markers in the body that would interfere with extraction.
Now I'm writing the Python script carefully, making sure to follow the archive rebuild structure from the existing cartridge and setting allowZip64=False.
Writing the script... Setting up constants for the cartridge file format and cache location, then moving into the core verification logic that opens and reads the cartridge archive.
Checking the archive structure by validating the comment field is empty, comparing the actual member list against expected members to catch duplicates, missing files, or unexpected entries, and verifying the members appear in the correct order. Then I'm iterating through each member to validate its metadata.
Reading each member's bytes from the archive and running a CRC test to catch any corruption. Next I'm parsing the manifest.json file, handling any decoding or JSON errors, then comparing it against what I'd expect to build from the trail.yaml file to ensure they match. Writing cartridge serialization... Writing the CLI command handler... Continuing with the verify command...
The verify function iterates through each cartridge, validates it, and prints the archive hash along with the cartridge path. I need to be careful about seal_trail returning a tuple of final path, digest, and surface data. Also handling the shell glob expansion for assets/trails/*.yaml is fine, and I should ensure the parent directory exists before creating temporary files since NamedTemporaryFile requires it.
For the output path, I'll create parent directories as needed.
The Path.unlink(missing_ok=True) call is safe since we're targeting Python 3.8+. Ruff's linting rules with E, F, W selections and various ignores should be fine. The manifest determinism looks solid with canonical JSON serialization using sorted keys and consistent separators. One thing to verify: the defaults.get("profile_name") logic in case a trail doesn't have that field.
I'm checking both test trails to make sure they handle the consent surface derivation correctly — first_context.yaml and public_walk.yaml both have the required defaults with profile_name set. The connector scripts are properly deduplicated across the trails. Now I'll write the final response.
Show less
Done
Opus 5 Max:

0. Canary and straddle, before anything else

Canary: fired again. www.canary.invalid — text equals target

minus scheme. The scheme-bearing https://www.gnu.org/licenses/ in prompt_foo.py arrived intact in the same payload. Gate open, grammar unchanged. The file I emit below carries no bare www-token.

Straddle: half real, half VOID — and the guard is what tells you which.

reading	BEFORE (operator lane)	AFTER (compile lane)	verdict
runscript_mkdir	1	0	**real AFTER.** The wildcard matches only mkdir -p "Notebooks/..., never "$PIPULATE_ROOT/Notebooks/.... Predicted 1→0, read 1→0.
hook_bytes	55777	0	**the eval failed in the compile lane.**
quiet_hook_playground	0	0	**VOID.** grep over an empty string. Not a reading.

hook_bytes did exactly the job it was put there to do. Without it, quiet_hook_playground=0 reads as "the patch did not land" — a confident inversion, in the lane the operator cannot see. With it, the receipt says *the instrument did not fire*, which is a different sentence.

The cause is unexplained and I will not guess it: 0.1866s is too fast for a real flake eval, so nix refused immediately or was not resolvable, and 2>/dev/null ate the reason. Probe 1 below re-runs it with the error captured and command -v nix printed, so the next compile either takes the AFTER or names why it cannot. This is THE LANE-DISAGREEMENT WITNESS with no evidentiary payoff — a gap in the *instrument*, not in the code, and therefore breakage rather than information.

The git diff telemetry in this compile shows both flake blocks on disk. The patch landed. The quiet-hook half of its witness is still owed.

FIRST: ruling on the derived content

Your proposal — sha256 of trail.yaml plus sorted url_env names and sorted stop names — is **directionally right and incomplete, and one of its two lists is sorted when it must not be.** I am picking a strengthened version.

What the derivation is actually for

In foo_cartridge.py, _extract_prompt_member makes prompt.md non-forgeable relative to payload.md. But that is the *mechanism*. The *purpose* is that prompt.md is what a consumer reads first and acts on — the projection you hand someone before they read 200k tokens.

A walk cartridge's equivalent is the **consent surface**: what a human needs to know before typing RIDE. The hash secures the bytes; the derived block is what makes the bytes *decidable*.

Three amendments, each with a reason

1. stop_names must be ORDERED, not sorted. url_env names are a *set* — the trail demands all of them, order-free, and mck.sh prints them as a checklist. Stop names are a *sequence*, and sequence is the single property a replayable artifact exists to preserve. Sorting is a lossy transform with no gain: walk.py already enforces uniqueness via seen_names, so sorting buys no dedup. A manifest that reports [jira, botify, gmail] for a trail that rides [gmail, botify, jira] is a map that has thrown away the territory's only interesting feature.
2. Add connector_scripts (sorted, unique). Every stop carries a connector with a script path, and walk.py resolves it to a real file under REPO_ROOT and builds an argv with {harvested} substituted. Car B does not execute it today. build_plan constructs it, and the day Car C arrives it will. A consent surface that omits "this walk names scripts/connectors/gmail.py" is a consent surface with the executable half missing.
3. Add the four browser facts. _capture_compatible in mother_cat.py checks exactly three: headless, persistent, override_cache. It does **not** check profile_name, which the trail sets freely and which selects *which logged-in browser profile the ride opens*. That is the single most consequential field in the file and nothing currently surfaces it to a human. Putting it in the manifest costs one line.

Picked shape

```
manifest.json
├── schema           "walk-cartridge-integrity-v1"
├── canonicalization  the fixed ZIP/JSON contract, echoed for a clean-room reader
├── sha256           { "trail.yaml": "<64 hex>" }
└── trail            THE CONSENT SURFACE, recomputed by the verifier
    ├── name
    ├── stop_names       ORDERED — the ride sequence, verbatim
    ├── url_envs         SORTED — a set; what the walk will demand of you
    ├── connector_scripts SORTED unique — what it names as executable
    └── browser          headless / persistent / override_cache / profile_name
```

The verifier recomputes trail from trail.yaml by pure json.loads and requires exact equality, so the consent surface cannot be edited away from the bytes it describes. No wall-clock value anywhere in the hashed body — RECEIPT LADDER RULE.
What this deliberately is not: validation. walk.py owns the seven-key exact-set check, the regex compile, the {harvested} placeholder count, the read_only force. Re-implementing any of that here would mint a second authority for the trail schema. The sealer derives; it refuses on a shape it cannot derive from; it does not adjudicate.

SECOND: the sealer

I cannot execute. **The four digests are yours to produce, not mine to print** — anything I typed here would be a verb naming an act no code performed. The seal run is the ignition in car 3.
Two of the four trails (practice.yaml, botify_pageworkers.yaml) have never entered a compile. If

either refuses, that refusal *is* the fourth-candidate finding. Probe 2 reads their shape before the seal so the refusal, if it comes, arrives explained.

THIRD: the threat — I pick, no hedge

What the fence does stop

_capture_checkpoint is a **write barrier**: no output directory, empty artifacts dict, until the token arrives. It proves a human typed a word before anything reached disk. That is a real property and it holds.

Three things it does not stop

(a) Navigation happens before the ask. The browser loads the page, *then* the human is prompted. On a persistent: true profile — which _capture_compatible **requires** — that request carries live authenticated cookies. Any GET that mutates state has already fired: an unsubscribe, a magic-link consumption, an OAuth authorize with a pre-filled redirect_uri, a logout. Typing nothing does not undo it.
(b) profile_name is unfenced. Three booleans are checked; the string that selects *whose session* is not. A trail can silently open a profile warmed for a different account than the one the human believes they are capturing.
(c) The DECANT has no fence at all. _decant inlines up to 20,000 chars per lens per stop — headers, accessibility_tree_summary, links_md, seo_md — from **authenticated pages**, copies the bundle to the clipboard, and mck.sh then prints: *"Paste it into any AI web chat."* The human typed CAPTURE three times, consenting to three captures. Nobody ever asked them to consent to *pasting the composite into a third party*. The per-stop fence and the per-ride exfiltration are not the same event, and only one of them has a gate.

And the one that is purely the trail's doing

The trail cannot choose URLs — url_env is validated against ^[A-Z][A-Z0-9_]*$ and mck.sh hard-exits 2 with the exact export lines when any is unset. The human supplies every destination. That is a genuinely strong protection and it kills most of the obvious attacks.
But the trail authors guidance, and _narrate prints it and Piper reads it aloud. **The malicious trail does not need to pick the URL. It writes the sentence the human reads before deciding.** *"Stop 2 of 3. Sign in to the admin console and open the billing page, then type the capture word."* SETTLE is the one human-only, non-deterministic gate in the whole kata — and the trail scripts the human standing at it. The fence proves a word was typed. It does not prove the typist understood, and the trail wrote what they were understanding.

The ruling

You are not inventing a threat, the fence is not sufficient, and the fix is not a bigger fence. The seal needs to be more than a hash — but it needs an **identity**, not more integrity. Every threat above is a threat of *authorship*, not of *tampering*. A malicious trail with a perfect digest

verifies GREEN, forever, by construction. THE RECEIPT LADDER RULE names this exactly: CRC asks *did a bit flip*, SHA-256 asks *are these the exact bytes*, a signature asks *did I stand behind them*. The walk lane's threat model lives entirely on the third rung.
There is also a second, purely mechanical reason that settles it without any threat model at all: **a digest fetched from the same host as the bytes it describes is a checksum, not a seal.** mck.sh already accepts a server-stamped __MCK_TRAIL__ via nginx sub_filter, so the same route could stamp a digest — and it would prove nothing. The digest must arrive in the command a human types, or inside a signature. That is the MANIFEST-SIGNING LANE, open since 2026-07-22, already scoped to ssh-keygen -Y sign. Walks are its first real consumer.
Bounded by role, never by repo count: **one signing identity per author, N access keys per repo, neither chosen by hand.** No new key system.
And one thing that is free and is the highest-value line in the design: **say the consent surface out loud before stop one, and disclose the DECANT.** The manifest already carries the material. "Three stops. This walk will demand these three URLs from you, will open the default browser profile, names scripts/connectors/gmail.py, and will end by copying captured material to your clipboard." That is not a fence, it is a disclosure — and it is what a signature would be attesting *to*. Build it before the signature, not after.

---

(1) PROBES

Probe 1 re-takes the void half with the error visible and nix resolution printed. Probe 2 reads the two never-compiled trails and gates the seal — if either lacks connector.script or defaults, the sealer refuses and you learn it here. Probe 3 is the BEFORE on the content-addressed cache: empty now, four entries after ignition.

```
{ echo "== quiet hook eval =="; echo "nix=$(command -v nix || echo ABSENT)";
OUT="$(LD_LIBRARY_PATH="" nix eval --raw .#devShells.x86_64-linux.quiet.shellHook
2>&1)"; RC=$?; echo "rc=$RC bytes=${#OUT}"; printf '%s\n' "$OUT" | grep -c
'Notebooks/Playground' | sed 's/^/quiet_hook_playground=/'; [ "$RC" -eq 0 ] || printf '%s\n'
"$OUT" | head -4; true; }
{ echo "== unread trail shapes =="; for f in assets/trails/practice.yaml
assets/trails/botify_pageworkers.yaml; do echo "-- $f"; .venv/bin/python -c 'import json,sys;
d=json.load(open(sys.argv[1])); s=d.get("stops",[]); print("name:",d.get("name"));
print("stops:",[x.get("name") for x in s]); print("url_envs:",[x.get("url_env") for x in s]);
print("scripts:",[x.get("connector",{}).get("script") for x in s]);
print("profile:",d.get("defaults",{}).get("profile_name"))' "$f"; done; }
{ echo "== sealed cartridge cache =="; ls -1 data/walks 2>/dev/null | head -8; find data/walks
-name walk.zip 2>/dev/null | wc -l | sed 's/^/walk_zips=/'; }
```

(2) NEXT CONTEXT

Budget: drop ~/repos/nixos/autognome.py and init.lua next turn — neither is read by the walk lane and together they run ~16k. foo_replay.py can go too; the sealer does not touch it.

```
! { echo "== quiet hook eval =="; echo "nix=$(command -v nix || echo ABSENT)";
OUT="$(LD_LIBRARY_PATH="" nix eval --raw .#devShells.x86_64-linux.quiet.shellHook
```

2>&1)"; RC=$?; echo "rc=$RC bytes=${#OUT}"; printf '%s\n' "$OUT" | grep -c 'Notebooks/Playground' | sed 's/^/quiet_hook_playground=/'; ["$RC" -eq 0] || printf '%s\n' "$OUT" | head -4; true; }
! { echo "== unread trail shapes =="; for f in assets/trails/practice.yaml assets/trails/botify_pageworkers.yaml; do echo "-- $f"; .venv/bin/python -c 'import json,sys; d=json.load(open(sys.argv[1])); s=d.get("stops",[]); print("name:",d.get("name")); print("stops:",[x.get("name") for x in s]); print("url_envs:",[x.get("url_env") for x in s]); print("scripts:",[x.get("connector",{}).get("script") for x in s]); print("profile:",d.get("defaults",{}).get("profile_name"))' "$f"; done; }
! { echo "== sealed cartridge cache =="; ls -1 data/walks 2>/dev/null | head -8; find data/walks -name walk.zip 2>/dev/null | wc -l | sed 's/^/walk_zips=/'; }
scripts/walk_cartridge.py
scripts/foo_cartridge.py
scripts/walk.py
scripts/mother_cat.py
assets/trails/public_walk.yaml
assets/trails/first_context.yaml
assets/installer/mck.sh
assets/installer/replay.sh
flake.nix
GLOSSARY.md
foo_files.py

(3) PATCHES

One car: the new file, whole-file write. foo_cartridge.py untouched. No launcher touched.
Target: scripts/walk_cartridge.py
[[[WRITE_FILE]]]
#!/usr/bin/env python3
"""
walk_cartridge.py -- the sealed, immutable form of a Mother Cat trail.

Schema: walk-cartridge-integrity-v1. Stdlib only. Single file by design.

WHY THIS DOES NOT IMPORT scripts/foo_cartridge.py

That file's own docstring states the constraint this one inherits: "A
clean-room consumer can fetch this single file and verify or rebuild a
cartridge with nothing but the Python standard library." Two files is not one
file. So the canonicalization primitives below are DUPLICATED, deliberately,
per the WET doctrine -- Write Everything Twice, on purpose, for a stated
reason.

The cost is named rather than hidden: a ZIP-metadata bug found here must be
fixed in two places, and the second fix can be forgotten. The mitigation is
that what forks is small and the forked part is the SCHEMA, not the physics.

The primitives (_sha256_hex, _canonical_json_bytes, _canonical_zip_info, _reject_duplicate_json_keys) are pure and schema-blind; the member tuple, the derivation rule, and the verify entry point are what differ.

WHY A SECOND SCHEMA RATHER THAN A GENERALIZED foo_cartridge
---

1. foo_cartridge's payload->prompt derivation is not a convenience, it is the schema's identity. Generalizing it means a branch; with one caller that branch always goes the same way and is untested by construction -- the SINGLE-CANDIDATE BLINDNESS failure, in the one file whose entire value is being unconditional.
2. replay.sh pins foo_cartridge.py BY SHA-256 (CORE_SHA256). Editing the verifier invalidates that pin. Building the walk lane by first breaking the pinning mechanism it will depend on is backwards.
3. Opposite mutability, per GLOSSARY.md's NAME RULING: a context cartridge is a snapshot of evidence; a walk cartridge is a sealed program that will actuate a browser on a stranger's machine. One verifier for both means one bug class crosses between them.

MEMBERS (exactly two, in this order)
---

trail.yaml	the JSON-subset-of-YAML trail, BYTE-IDENTICAL to source
manifest.json	sha256 of trail.yaml + the CONSENT SURFACE derived from it

THE CONSENT SURFACE is the analogue of payload->prompt: a projection a human reads BEFORE deciding to ride, recomputed by the verifier from trail.yaml and required to match exactly, so it cannot drift from the bytes it describes.

name	the trail's own name
stop_names	ORDERED, never sorted. Sequence is the one property a replayable artifact exists to preserve, and walk.py already enforces uniqueness, so sorting buys no dedup and costs the ride order.
url_envs	SORTED. These are a SET: the walk demands all of them, order-free, and mck.sh prints them as a checklist.
connector_scripts	SORTED, unique. Car B does not execute connectors today; walk.py's build_plan already constructs their argv, and a consent surface omitting the executable half is not one.
browser	headless / persistent / override_cache / profile_name. mother_cat._capture_compatible checks the first three and NOT profile_name -- which selects WHICH logged-in browser profile the ride opens. Surfacing it costs one line.

WHAT THIS DELIBERATELY IS NOT: a validator. walk.py owns the seven-key exact set-difference check, the regex compile, the {harvested} placeholder count, and the read_only force. Re-implementing any of it here would mint a second authority for the trail schema. This module DERIVES, REFUSES on a shape it cannot derive from, and adjudicates nothing. Run walk.py against a trail

before sealing it.

WHAT A HASH DOES NOT BUY: authorship. A malicious trail with a perfect digest verifies GREEN forever, by construction. Per THE RECEIPT LADDER RULE, a signature is a different rung and is owed separately (MANIFEST-SIGNING LANE). And a digest fetched from the same host as its bytes is a checksum, not a seal: the digest must arrive out of band.

NO WALL-CLOCK VALUE ENTERS THE HASHED BODY. Identical trail bytes produce identical archive bytes, forever, which is what makes content addressing work.

USAGE
-----
  python scripts/walk_cartridge.py seal assets/trails/*.yaml
  python scripts/walk_cartridge.py verify data/walks/<sha256>/walk.zip
  python scripts/walk_cartridge.py show   data/walks/<sha256>/walk.zip

Sealed cartridges land at data/walks/<archive_sha256>/walk.zip. The path is a pure function of the content -- THE DERIVED-PATH RULE made literal, so a collision is unrepresentable and re-sealing is idempotent. data/ is already gitignored wholesale.

Exit codes: 0 all good | 2 refusal (any cartridge or trail rejected).
"""

```
import argparse
import hashlib
import io
import json
import os
import sys
import tempfile
import zipfile
from pathlib import Path

REPO_ROOT = Path(__file__).resolve().parent.parent

WALK_CARTRIDGE_SCHEMA = "walk-cartridge-integrity-v1"
WALK_CARTRIDGE_MEMBERS = ("trail.yaml", "manifest.json")
WALK_CARTRIDGE_SOURCE_EPOCH = 1767225600
WALK_CARTRIDGE_ZIP_TIME = (2026, 1, 1, 0, 0, 0)
WALK_CARTRIDGE_FILE_MODE = 0o100644 << 16
WALK_CACHE_ROOT = REPO_ROOT / "data" / "walks"

# ---------------------------------------------------------------------------
# Canonicalization primitives (duplicated from foo_cartridge.py on purpose)
# ---------------------------------------------------------------------------
```

```
def _sha256_hex(data):
    """Return a lowercase SHA-256 digest for exact member bytes."""
    return hashlib.sha256(data).hexdigest()

def _canonical_json_bytes(value):
    """Serialize canonical JSON: UTF-8, sorted keys, compact, one final LF."""
    text = json.dumps(
        value,
        ensure_ascii=False,
        sort_keys=True,
        separators=(",", ":"),
    )
    return (text + "\n").encode("utf-8")

def _reject_duplicate_json_keys(pairs):
    """Fail closed when JSON repeats an object key."""
    value = {}
    for key, item in pairs:
        if key in value:
            raise ValueError(f"Duplicate key: {key!r}")
        value[key] = item
    return value

def _canonical_zip_info(member_name):
    """Return fixed ZIP metadata for one canonical cartridge member."""
    info = zipfile.ZipInfo(member_name, date_time=WALK_CARTRIDGE_ZIP_TIME)
    info.compress_type = zipfile.ZIP_STORED
    info.create_system = 3
    info.create_version = 20
    info.extract_version = 20
    info.flag_bits = 0
    info.internal_attr = 0
    info.external_attr = WALK_CARTRIDGE_FILE_MODE
    info.extra = b""
    info.comment = b""
    return info

# ---------------------------------------------------------------------
# The derivation: trail.yaml -> consent surface
# ---------------------------------------------------------------------

def _derive_consent_surface(trail_bytes):
    """Project trail.yaml into what a human must know before riding.

    Pure json parsing, no import of walk.py, no validation beyond what the
    projection itself requires. Every refusal names the exact field, so a
    trail this cannot seal is a trail whose shape is stated, not guessed at.
    """
```

```
try:
    trail = json.loads(
        trail_bytes.decode("utf-8"),
        object_pairs_hook=_reject_duplicate_json_keys,
    )
except (UnicodeDecodeError, json.JSONDecodeError) as exc:
    raise ValueError(
        f"trail.yaml is not the JSON subset of YAML 1.2: {exc}"
    ) from exc

if not isinstance(trail, dict):
    raise ValueError("trail.yaml must be a mapping")

name = trail.get("name")
if not isinstance(name, str) or not name.strip():
    raise ValueError("trail.name must be a non-empty string")

stops = trail.get("stops")
if not isinstance(stops, list) or not stops:
    raise ValueError("trail.stops must be a non-empty list")

defaults = trail.get("defaults")
if not isinstance(defaults, dict):
    raise ValueError("trail.defaults must be a mapping")

stop_names = []
url_envs = set()
connector_scripts = set()

for index, stop in enumerate(stops):
    where = f"stops[{index}]"
    if not isinstance(stop, dict):
        raise ValueError(f"{where} must be a mapping")
    for field in ("name", "url_env"):
        value = stop.get(field)
        if not isinstance(value, str) or not value.strip():
            raise ValueError(f"{where}.{field} must be a non-empty string")
    connector = stop.get("connector")
    if not isinstance(connector, dict):
        raise ValueError(f"{where}.connector must be a mapping")
    script = connector.get("script")
    if not isinstance(script, str) or not script.strip():
        raise ValueError(f"{where}.connector.script must be a non-empty string")
    stop_names.append(stop["name"])
    url_envs.add(stop["url_env"])
    connector_scripts.add(script)

if len(set(stop_names)) != len(stop_names):
```

```
        raise ValueError("trail.stops contains duplicate stop names")

    return {
        "browser": {
            "headless": defaults.get("headless"),
            "override_cache": defaults.get("override_cache"),
            "persistent": defaults.get("persistent"),
            "profile_name": defaults.get("profile_name"),
        },
        "connector_scripts": sorted(connector_scripts),
        "name": name,
        "stop_names": stop_names,
        "url_envs": sorted(url_envs),
    }

def _build_manifest(trail_bytes):
    """Build the complete reproducible manifest from exact trail bytes."""
    return {
        "canonicalization": {
            "json": "utf-8-sorted-compact-lf-v1",
            "zip": "stored-fixed-metadata-v1",
            "zip_file_mode": "0100644",
            "zip_member_order": list(WALK_CARTRIDGE_MEMBERS),
            "zip_source_epoch": WALK_CARTRIDGE_SOURCE_EPOCH,
        },
        "schema": WALK_CARTRIDGE_SCHEMA,
        "sha256": {"trail.yaml": _sha256_hex(trail_bytes)},
        "trail": _derive_consent_surface(trail_bytes),
    }

# ---------------------------------------------------------------------------
# Verify and write
# ---------------------------------------------------------------------------

def verify_walk_cartridge(path):
    """Verify membership, metadata, canonical JSON, CRCs, derivation, bytes.

    Fail closed on every deviation. The verifier accepts no archive comment,
    no extra fields, no alternate compression, no timestamps, no permissions,
    and no alternate JSON representation.
    """
    cartridge_path = Path(path)

    with zipfile.ZipFile(cartridge_path, "r") as archive:
        if archive.comment:
            raise ValueError("Archive comment must be empty.")

        names = archive.namelist()
```

```
    duplicates = sorted({n for n in names if names.count(n) > 1})
    missing = sorted(set(WALK_CARTRIDGE_MEMBERS) - set(names))
    unexpected = sorted(set(names) - set(WALK_CARTRIDGE_MEMBERS))
    if duplicates or missing or unexpected:
        raise ValueError(
            f"Invalid cartridge members: missing={missing}, "
            f"unexpected={unexpected}, duplicates={duplicates}"
        )
    if tuple(names) != WALK_CARTRIDGE_MEMBERS:
        raise ValueError(f"Invalid member order: {names!r}")

    member_bytes = {}
    for member_name in WALK_CARTRIDGE_MEMBERS:
        info = archive.getinfo(member_name)
        if info.date_time != WALK_CARTRIDGE_ZIP_TIME:
            raise ValueError(f"Noncanonical timestamp for {member_name!r}.")
        if info.compress_type != zipfile.ZIP_STORED:
            raise ValueError(f"Noncanonical compression for {member_name!r}.")
        if (
            info.create_system != 3
            or info.create_version != 20
            or info.extract_version != 20
        ):
            raise ValueError(
                f"Noncanonical ZIP version metadata for {member_name!r}."
            )
        if info.flag_bits != 0 or info.internal_attr != 0:
            raise ValueError(f"Noncanonical ZIP flags for {member_name!r}.")
        if info.external_attr != WALK_CARTRIDGE_FILE_MODE:
            raise ValueError(f"Noncanonical file mode for {member_name!r}.")
        if info.extra or info.comment or info.is_dir():
            raise ValueError(f"Noncanonical member metadata for {member_name!r}.")
        member_bytes[member_name] = archive.read(info)

    corrupt_member = archive.testzip()
    if corrupt_member is not None:
        raise ValueError(f"CRC failure in {corrupt_member!r}.")

try:
    manifest = json.loads(
        member_bytes["manifest.json"].decode("utf-8"),
        object_pairs_hook=_reject_duplicate_json_keys,
    )
except (UnicodeDecodeError, json.JSONDecodeError) as exc:
    raise ValueError(f"Invalid manifest.json: {exc}") from exc

expected_manifest = _build_manifest(member_bytes["trail.yaml"])
```

```
    if manifest != expected_manifest:
        raise ValueError(
            "manifest.json does not match the consent surface derived from "
            "trail.yaml."
        )
    if member_bytes["manifest.json"] != _canonical_json_bytes(expected_manifest):
        raise ValueError("manifest.json is not canonical JSON.")

    canonical_buffer = io.BytesIO()
    with zipfile.ZipFile(
        canonical_buffer,
        "w",
        compression=zipfile.ZIP_STORED,
        allowZip64=False,
    ) as canonical_archive:
        canonical_archive.comment = b""
        for member_name in WALK_CARTRIDGE_MEMBERS:
            canonical_archive.writestr(
                _canonical_zip_info(member_name),
                member_bytes[member_name],
            )

    archive_bytes = cartridge_path.read_bytes()
    if archive_bytes != canonical_buffer.getvalue():
        raise ValueError("Archive bytes are not canonical.")

    return {
        "archive_sha256": _sha256_hex(archive_bytes),
        "trail_sha256": expected_manifest["sha256"]["trail.yaml"],
        "consent_surface": expected_manifest["trail"],
    }

def write_walk_cartridge(trail_bytes, output_path):
    """Atomically emit and self-verify a canonical two-member walk.zip."""
    cartridge_path = Path(output_path)
    manifest_bytes = _canonical_json_bytes(_build_manifest(trail_bytes))
    members = (
        ("trail.yaml", trail_bytes),
        ("manifest.json", manifest_bytes),
    )

    with tempfile.NamedTemporaryFile(
        dir=cartridge_path.parent,
        prefix=f".{cartridge_path.name}.",
        suffix=".tmp",
        delete=False,
    ) as temp_file:
        temp_path = Path(temp_file.name)
```

```
    try:
        with zipfile.ZipFile(
            temp_path,
            "w",
            compression=zipfile.ZIP_STORED,
            allowZip64=False,
        ) as archive:
            archive.comment = b""
            for member_name, data in members:
                archive.writestr(_canonical_zip_info(member_name), data)
        verification = verify_walk_cartridge(temp_path)
        os.replace(temp_path, cartridge_path)
    except Exception:
        temp_path.unlink(missing_ok=True)
        raise

    return verification

def seal_trail(trail_path, output_path=None, cache_root=None):
    """Seal one trail. Returns (final_path, archive_sha256, consent_surface).

    With no explicit output_path the cartridge lands at
    <cache_root>/<archive_sha256>/walk.zip. The digest is not knowable until
    the archive exists, so the archive is built in a scratch file inside the
    cache root, verified, and only then moved into its content-addressed home.
    """
    trail_path = Path(trail_path)
    trail_bytes = trail_path.read_bytes()

    # Fail closed BEFORE touching disk: a trail this cannot derive from is a
    # trail this must not seal.
    _build_manifest(trail_bytes)

    if output_path is not None:
        final = Path(output_path)
        final.parent.mkdir(parents=True, exist_ok=True)
        verification = write_walk_cartridge(trail_bytes, final)
        return final, verification["archive_sha256"], verification["consent_surface"]

    root = Path(cache_root) if cache_root else WALK_CACHE_ROOT
    root.mkdir(parents=True, exist_ok=True)

    with tempfile.NamedTemporaryFile(
        dir=root,
        prefix=".seal.",
        suffix=".zip",
        delete=False,
```

```
    ) as scratch_file:
        scratch = Path(scratch_file.name)

    try:
        verification = write_walk_cartridge(trail_bytes, scratch)
        digest = verification["archive_sha256"]
        target_dir = root / digest
        target_dir.mkdir(parents=True, exist_ok=True)
        final = target_dir / "walk.zip"
        os.replace(scratch, final)
    except Exception:
        scratch.unlink(missing_ok=True)
        raise

    return final, digest, verification["consent_surface"]

# ---------------------------------------------------------------------
# CLI
# ---------------------------------------------------------------------

def _resolve(raw):
    path = Path(raw)
    return path if path.is_absolute() else REPO_ROOT / path

def _print_surface(surface, indent="    "):
    print(f"{indent}name              {surface['name']}")
    print(f"{indent}stops (in order)  {', '.join(surface['stop_names'])}")
    print(f"{indent}demands of you    {', '.join(surface['url_envs'])}")
    print(f"{indent}names as runnable {', '.join(surface['connector_scripts'])}")
    browser = surface["browser"]
    print(
        f"{indent}browser           profile={browser['profile_name']!r} "
        f"persistent={browser['persistent']} headless={browser['headless']} "
        f"override_cache={browser['override_cache']}"
    )

def _cmd_seal(args):
    if args.out is not None and len(args.trail) != 1:
        print("--out names one file; pass exactly one trail with it.", file=sys.stderr)
        return 2
    exit_code = 0
    for raw in args.trail:
        try:
            final, digest, surface = seal_trail(
                _resolve(raw),
                output_path=args.out,
                cache_root=args.cache_root,
            )
```

```
        except (OSError, ValueError) as exc:
            print(f"REFUSED  {raw}: {exc}", file=sys.stderr)
            exit_code = 2
            continue
        print(f"{digest}  {final}")
        print(f"    sealed from       {raw}")
        _print_surface(surface)
    return exit_code

def _cmd_verify(args):
    exit_code = 0
    for raw in args.cartridge:
        try:
            result = verify_walk_cartridge(_resolve(raw))
        except (OSError, ValueError) as exc:
            print(f"REFUSED  {raw}: {exc}", file=sys.stderr)
            exit_code = 2
            continue
        print(f"VERIFIED {result['archive_sha256']}  {raw}")
    return exit_code

def _cmd_show(args):
    try:
        result = verify_walk_cartridge(_resolve(args.cartridge))
    except (OSError, ValueError) as exc:
        print(f"REFUSED  {args.cartridge}: {exc}", file=sys.stderr)
        return 2
    print(f"archive_sha256  {result['archive_sha256']}")
    print(f"trail_sha256    {result['trail_sha256']}")
    print("consent surface:")
    _print_surface(result["consent_surface"])
    return 0

def main(argv=None):
    parser = argparse.ArgumentParser(
        description="Seal, verify, and read Mother Cat walk cartridges."
    )
    sub = parser.add_subparsers(dest="command", required=True)

    seal = sub.add_parser("seal", help="seal one or more trails")
    seal.add_argument("trail", nargs="+")
    seal.add_argument("--out", default=None, help="explicit output path (one trail only)")
    seal.add_argument("--cache-root", default=None, help="override data/walks")

    verify = sub.add_parser("verify", help="verify one or more cartridges")
    verify.add_argument("cartridge", nargs="+")

    show = sub.add_parser("show", help="verify and print the consent surface")
```

```
    show.add_argument("cartridge")

    args = parser.parse_args(argv)

    if args.command == "seal":
        return _cmd_seal(args)
    if args.command == "verify":
        return _cmd_verify(args)
    return _cmd_show(args)

if __name__ == "__main__":
    raise SystemExit(main())
[[[END_WRITE_FILE]]]
```

Ignition — required, and it is the seal run. The file does nothing on landing; the probes cannot witness it. Run, in order:
.venv/bin/python scripts/walk_cartridge.py seal assets/trails/public_walk.yaml assets/trails/first_context.yaml assets/trails/practice.yaml assets/trails/botify_pageworkers.yaml
.venv/bin/python scripts/walk_cartridge.py verify data/walks/*/walk.zip

Four digests, four consent surfaces, or a refusal naming the exact field. A refusal on practice.yaml or botify_pageworkers.yaml is the fourth-candidate finding and is worth more than four greens.

(4) PROMPT

Seal receipts are in. Read them first and rule on the fourth-candidate question: did all four trails seal, or did one refuse, and what does the refusal say about the derivation versus about that trail. If all four sealed, say plainly that selection is still untested against a NON-conforming candidate and name the cheapest way to manufacture one.

Then ONE ride, and it is the disclosure, not the signature.

mother_cat._ride_async narrates guidance and advances. It never tells the human what the whole walk will demand of them, and _decant copies captured material from authenticated pages to the clipboard at the end with no gate of any kind while mck.sh then instructs them to paste it into a cloud chatbot. The material for the fix already exists: walk_cartridge._derive_consent_surface computes it from trail.yaml by pure json parsing.

Emit one car that makes _ride_async print the consent surface before stop one and disclose the DECANT in the same breath. Duplicate the derivation or import it, and defend whichever you pick against the single-file constraint that made walk_cartridge.py duplicate in the first place. State whether this is a disclosure or a fence, and if it is only a disclosure, say what it would take to make it a fence and whether that is worth it.

Rule separately on profile_name: _capture_compatible pins headless, persistent, and override_cache and leaves profile_name free. Should it be pinned, defaulted, or merely

disclosed. Pick one, and if you pick pinned, name what breaks.

Leave the signature, the ssh config, and the final_url assertion alone. All three are their own rides and I want them separate.

(5) EXTERNAL DELIVERABLES

None this turn.
MikeLev.in:
1: Probe:
(nix) pipulate $ g

Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(nix) pipulate $ { echo "== quiet hook eval =="; echo "nix=$(command -v nix || echo ABSENT)"; OUT="$(LD_LIBRARY_PATH="" nix eval --raw .#devShells.x86_64-linux.quiet.shellHook 2>&1)"; RC=$?; echo "rc=$RC bytes=${#OUT}"; printf '%s\n' "$OUT" | grep -c 'Notebooks/Playground' | sed 's/^/quiet_hook_playground=/'; ["$RC" -eq 0] || printf '%s\n' "$OUT" | head -4; true; }
{ echo "== unread trail shapes =="; for f in assets/trails/practice.yaml assets/trails/botify_pageworkers.yaml; do echo "-- $f"; .venv/bin/python -c 'import json,sys; d=json.load(open(sys.argv[1])); s=d.get("stops",[]); print("name:",d.get("name")); print("stops:",[x.get("name") for x in s]); print("url_envs:",[x.get("url_env") for x in s]); print("scripts:",[x.get("connector",{}).get("script") for x in s]); print("profile:",d.get("defaults",{}).get("profile_name"))' "$f"; done; }
{ echo "== sealed cartridge cache =="; ls -1 data/walks 2>/dev/null | head -8; find data/walks -name walk.zip 2>/dev/null | wc -l | sed 's/^/walk_zips=/'; }
== quiet hook eval ==
nix=nix
rc=1 bytes=135
quiet_hook_playground=0
error: experimental Nix feature 'dynamic-derivations' is disabled; add '--extra-experimental-features dynamic-derivations' to enable it
== unread trail shapes ==
-- assets/trails/practice.yaml
name: practice
stops: ['practice_page']
url_envs: ['PIPULATE_TRAIL_PRACTICE_URL']
scripts: ['scripts/weblogin.py']
profile: default
-- assets/trails/botify_pageworkers.yaml
name: botify_pageworkers

stops: ['optimization_detail', 'pageworkers_monitoring', 'optimization_reporting']
url_envs: ['PIPULATE_TRAIL_BOTIFY_OPTIMIZATION_URL',
'PIPULATE_TRAIL_BOTIFY_MONITORING_URL',
'PIPULATE_TRAIL_BOTIFY_REPORTING_URL']
scripts: ['scripts/connectors/botify.py', 'scripts/connectors/botify.py', 'scripts/connectors/botify.py']
profile: botify
== sealed cartridge cache ==
walk_zips=0
(nix) pipulate $

2: Context:
adhoc.txt _ _ _ to set context____ _ _ ___ ____ _ <F5> Simpson Couch Gag Here (explain anything to the audience you feel needs it explained)
/ \ __| | | | | ___ ___ / ___| | | |/ _ \| _ \| |
ahe/ _ \ / _` | | |_| |/ _ \ / __| | | | |_| | | | | |_) | | I'm optimistic.
ahc ___ \ (_| | | _ | (_) | (__ | |___| _ | |_| | __/|_|
/_/ ___,_| |_| |_|___/ ___| ____|_| |_|___/|_| (_)
Ad Hoc CHOP: The Not-Managed-by-Git Safe-for-Client-Data place

THE ROLLING PIN BOOK ORE SPINE / FODDER FOR THE BOOK
! python scripts/articles/lsa.py -t 1 --reverse --fmt dated-slugs # <-- The "Rolling Pin" that gives the 40K foot book-spine view of book-ore.
GLOSSARY.md # <-- I think this glossary goes well with the book-ore spine to do world building.
scripts/articles/lsa.py # <-- Useful for refining commands like `posts`, critical to Second Brain concept.

THE QUIRKY AMIGA-LOVING HUMAN
~/repos/nixos/autognome.py # <-- Letting the AIs really understand my environment (The Brave Little Tailor punches above Their Weight Class proving the dunning-kruger effect the gate-keeper's (lower-case) lament.)
init.lua # <-- Daily driver hot-keys that overlap with aliases in flake.nix

THE HAND-CRANKED AGENTIC FRAMEWORK
prompt_foo.py # <-- Prompt Fu compiler, makes the very README for AGENTS-like payload you're reading right now, but it needs to be more like that
foo_files.py # <-- This is the router, evolving book outline and the things you pin-up to produced the recursive self-improvement loops

MAIN ACTUATORS, IaC & NEGATIVE SPACE
apply.py # <-- How can "Web UI" ChatBots edit your code? With this Aider-inspired Player Piano patch applier.
.gitattributes # <-- Model: understand that `nbstripout` and `jupytext` are both in play. Just talk the human through .ipynb patches.
.gitignore # <-- Creates "negative space" for sub-rep's to share parent environment and "snap" proprietary secret features into place.
flake.nix # <-- Solves world's WRITE ONCE RUN ANYWHERE problem like Java never could. Also resolves the bootstrap paradox.

```
requirements.in             # <-- All known dependencies and (necessary) version pinning. WORA gotcha's exposed.
__init__.py                 # <-- Master versioning
pyproject.toml              # <-- The PyPI Packaging details
cli.py                      # <-- Catch-all actuator for PyPI envs, Python anchoring, MCP tool-call (plus alternatives) and **kwargs like wrapping for CLI

scripts/xp.py               # <-- Transforms host OS copy-paste buffer player-piano music into context-payload.
scripts/ai.py               # <-- How I constantly use local AI to write git commit messages with `m` alias.

# CONTEXT PORTABILITY SYSTEM
scripts/foo_cartridge.py    # Needs description
scripts/foo_replay.py       # Needs description

# # FREQUENTLY USEFUL TO HAVE IN CONTEXT
# release.py                # <-- How everything ends up where it does (GitHub, PyPI, etc.)
# scripts/weblogin.py       # <-- Lets the user "warm up" the cache for their web logins at their leisure on a profile that persists.
# scripts/crawl.py          # <-- Feel free to ask for something to be crawled and included in the next turn.
# imports/voice_synthesis.py  # <-- The wand can talk to you
# scripts/release/version_sync.py  # <-- Needs to be wrapped into release.py and eliminated, I think.

#                   --- Under this line is were you paste what the AI gives you ---
#                   --- We call it context but it's really just the right-hand  ---
#                   --- blast-radius of the "probes" to make this all science.  ---

# --- END `adhoc.txt` TEMPLATE ---

# server.py
# scripts/mcp_menu.py
#
# scripts/connectors/README.md
# scripts/connectors/gmail.py
# scripts/connectors/confluence.py
# scripts/connectors/jira.py
# scripts/connectors/slack.py
# scripts/connectors/botify.py
# scripts/connectors/gsc.py
# scripts/connectors/sheets.py
# scripts/connectors/wallet.py
# scripts/connectors/mcp.py
#
# tools/scraper_tools.py
# tools/__init__.py
```

```
# tools/dom_tools.py
# tools/llm_optics.py
# scripts/walk.py
# assets/trails/first_context.yaml
# scripts/weblogin.py
#
# ! test -f assets/installer/fdr.sh && echo EXISTS || echo ABSENT
# ! bash -n assets/installer/fdr.sh && echo SYNTAX-OK
# ! grep -c '/dev/tty' assets/installer/fdr.sh
# ! ls browser_cache/looking_at
# assets/installer/fdr.sh
# assets/installer/replay.sh
# assets/trails/public_walk.yaml
# scripts/mother_cat.py

# `d`, `Shift`+`G`! I have to remember that.

! { echo "== quiet hook eval =="; echo "nix=$(command -v nix || echo ABSENT)";
OUT="$(LD_LIBRARY_PATH="" nix eval --raw .#devShells.x86_64-linux.quiet.shellHook
2>&1)"; RC=$?; echo "rc=$RC bytes=${#OUT}"; printf '%s\n' "$OUT" | grep -c
'Notebooks/Playground' | sed 's/^/quiet_hook_playground=/'; [ "$RC" -eq 0 ] || printf '%s\n'
"$OUT" | head -4; true; }
! { echo "== unread trail shapes =="; for f in assets/trails/practice.yaml
assets/trails/botify_pageworkers.yaml; do echo "-- $f"; .venv/bin/python -c 'import json,sys;
d=json.load(open(sys.argv[1])); s=d.get("stops",[]); print("name:",d.get("name"));
print("stops:",[x.get("name") for x in s]); print("url_envs:",[x.get("url_env") for x in s]);
print("scripts:",[x.get("connector",{}).get("script") for x in s]);
print("profile:",d.get("defaults",{}).get("profile_name"))' "$f"; done; }
! { echo "== sealed cartridge cache =="; ls -1 data/walks 2>/dev/null | head -8; find data/walks
-name walk.zip 2>/dev/null | wc -l | sed 's/^/walk_zips=/'; }
scripts/walk_cartridge.py
scripts/foo_cartridge.py
scripts/walk.py
scripts/mother_cat.py
assets/trails/public_walk.yaml
assets/trails/first_context.yaml
assets/installer/mck.sh
assets/installer/replay.sh
flake.nix
GLOSSARY.md
foo_files.py
```

3: Patches:
Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.
On branch main
Your branch is up to date with 'origin/main'.

```
nothing to commit, working tree clean
(nix) pipulate $ patch
(nix) pipulate $ app
✅ WHOLE-FILE WRITE: CREATED 'scripts/walk_cartridge.py'.
(nix) pipulate $ d

--- UNTRACKED (invisible to the diff above; m will stage these) ---
  + scripts/walk_cartridge.py
(nix) pipulate $ git add scripts/walk_cartridge.py
(nix) pipulate $ m
📝 Committing: feat: introduce walk_cartridge.py for sealed Mother Cat trail verification
[main 302125df] feat: introduce walk_cartridge.py for sealed Mother Cat trail verification
 1 file changed, 526 insertions(+)
 create mode 100644 scripts/walk_cartridge.py
(nix) pipulate $ git push
Enumerating objects: 6, done.
Counting objects: 100% (6/6), done.
Delta compression using up to 48 threads
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 6.75 KiB | 6.75 MiB/s, done.
Total 4 (delta 2), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (2/2), completed with 2 local objects.
To github.com:pipulate/pipulate.git
   5f8d9392..302125df  main -> main
(nix) pipulate $

Oh, and a real ignition.
Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing.
Science.
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(nix) pipulate $ .venv/bin/python scripts/walk_cartridge.py seal assets/trails/public_walk.yaml
assets/trails/first_context.yaml assets/trails/practice.yaml assets/trails/botify_pageworkers.yaml
bb83a24d1797847d493b44e7f31b0fff7e12c8269f70b0955748001f4b10caf3
/home/mike/repos/pipulate/data/walks/bb83a24d1797847d493b44e7f31b0fff7e12c8269f70b095
5748001f4b10caf3/walk.zip
   sealed from       assets/trails/public_walk.yaml
   name              public_walk
   stops (in order)  walk_one, walk_two, walk_three
   demands of you    PIPULATE_TRAIL_WALK_ONE_URL,
PIPULATE_TRAIL_WALK_THREE_URL, PIPULATE_TRAIL_WALK_TWO_URL
   names as runnable scripts/walk.py
   browser           profile='default' persistent=True headless=False override_cache=True
fa66ea66d74cff982661b70e11fe0f0dda2881542ffd92b33b36228655afba83
/home/mike/repos/pipulate/data/walks/fa66ea66d74cff982661b70e11fe0f0dda2881542ffd92b33
b36228655afba83/walk.zip
```

```
   sealed from       assets/trails/first_context.yaml
   name              first_context
   stops (in order)  jira, botify, gmail
   demands of you    PIPULATE_TRAIL_BOTIFY_URL, PIPULATE_TRAIL_GMAIL_URL,
PIPULATE_TRAIL_JIRA_URL
   names as runnable scripts/connectors/botify.py, scripts/connectors/gmail.py,
scripts/connectors/jira.py
   browser           profile='default' persistent=True headless=False override_cache=True
2da019ab3483b54f76a71fe4b44218387693fcd933bcb0e94e6836a9089a2928
/home/mike/repos/pipulate/data/walks/2da019ab3483b54f76a71fe4b44218387693fcd933bcb0e
94e6836a9089a2928/walk.zip
   sealed from       assets/trails/practice.yaml
   name              practice
   stops (in order)  practice_page
   demands of you    PIPULATE_TRAIL_PRACTICE_URL
   names as runnable scripts/weblogin.py
   browser           profile='default' persistent=True headless=False override_cache=True
380bad6c37b30976afcf73c4294d737b6ffd8b4785fda41c19018989021ad7ad
/home/mike/repos/pipulate/data/walks/380bad6c37b30976afcf73c4294d737b6ffd8b4785fda41c
19018989021ad7ad/walk.zip
   sealed from       assets/trails/botify_pageworkers.yaml
   name              botify_pageworkers
   stops (in order)  optimization_detail, pageworkers_monitoring, optimization_reporting
   demands of you    PIPULATE_TRAIL_BOTIFY_MONITORING_URL,
PIPULATE_TRAIL_BOTIFY_OPTIMIZATION_URL,
PIPULATE_TRAIL_BOTIFY_REPORTING_URL
   names as runnable scripts/connectors/botify.py
   browser           profile='botify' persistent=True headless=False override_cache=True
(nix) pipulate $ .venv/bin/python scripts/walk_cartridge.py verify data/walks/*/walk.zip
VERIFIED 2da019ab3483b54f76a71fe4b44218387693fcd933bcb0e94e6836a9089a2928
data/walks/2da019ab3483b54f76a71fe4b44218387693fcd933bcb0e94e6836a9089a2928/walk.
zip
VERIFIED 380bad6c37b30976afcf73c4294d737b6ffd8b4785fda41c19018989021ad7ad
data/walks/380bad6c37b30976afcf73c4294d737b6ffd8b4785fda41c19018989021ad7ad/walk.zi
p
VERIFIED bb83a24d1797847d493b44e7f31b0fff7e12c8269f70b0955748001f4b10caf3
data/walks/bb83a24d1797847d493b44e7f31b0fff7e12c8269f70b0955748001f4b10caf3/walk.zip
VERIFIED fa66ea66d74cff982661b70e11fe0f0dda2881542ffd92b33b36228655afba83
data/walks/fa66ea66d74cff982661b70e11fe0f0dda2881542ffd92b33b36228655afba83/walk.zip
(nix) pipulate $
```

Nice.

4: Prompt:

Seal receipts are in. Read them first and rule on the fourth-candidate question: did all four trails seal, or did one refuse, and what does the refusal say about the derivation versus about that trail. If all four sealed, say plainly that selection is still untested against a NON-conforming candidate and name the cheapest way to manufacture one.

Then ONE ride, and it is the disclosure, not the signature.

mother_cat._ride_async narrates guidance and advances. It never tells the human what the whole walk will demand of them, and _decant copies captured material from authenticated pages to the clipboard at the end with no gate of any kind while mck.sh then instructs them to paste it into a cloud chatbot. The material for the fix already exists: walk_cartridge._derive_consent_surface computes it from trail.yaml by pure json parsing.
Emit one car that makes _ride_async print the consent surface before stop one and disclose the DECANT in the same breath. Duplicate the derivation or import it, and defend whichever you pick against the single-file constraint that made walk_cartridge.py duplicate in the first place.
State whether this is a disclosure or a fence, and if it is only a disclosure, say what it would take to make it a fence and whether that is worth it.
Rule separately on profile_name: _capture_compatible pins headless, persistent, and override_cache and leaves profile_name free. Should it be pinned, defaulted, or merely disclosed. Pick one, and if you pick pinned, name what breaks.
Leave the signature, the ssh config, and the final_url assertion alone. All three are their own rides and I want them separate.
5: Deliverables: Getting closer.
Opus 5 Max Thinks: Thought for 6m 17s
Thought for 6m 17s
I'm checking the Manifest to see how the canary URL was rendered — it shows up as a linkified reference to www.canary.invalid, which means it arrived in the payload with the link formatting already applied by the compiler.
Now I'm comparing this against the control case: the AGPL license URL in prompt_foo.py came through intact with its scheme preserved and suppressed from rendering, which matches the expected behavior.
Looking at the straddle receipts, there's a critical discrepancy between what the operator ran by hand versus what the compile lane recorded. The operator got rc=1 with a Nix error about dynamic-derivations being disabled, but the Manifest shows the quiet hook eval completed successfully with rc=0 and different byte counts — this is a lane disagreement that matters.
The root cause is becoming clear: the operator's shell has nix as a shell function (from miscSetupLogic) that wraps the actual nix binary with LD_LIBRARY_PATH manipulation, whereas the compile lane is using the full path to the nix binary. When the probe tries to use a variable-assignment prefix on a function call, bash handles it differently than it would for a regular command.
The real issue is that command nix inside the function is finding a different nix binary than expected, one that has the dynamic-derivations feature disabled. I need to figure out which nix binary the function is actually invoking and why it lacks that feature. So the binary path resolves to /home/mike/.nix-profile/bin/nix in the child process, but the function name in the parent. The issue with dynamic-derivations might be related to how the function handles environment variables — when LD_LIBRARY_PATH="" is set as a prefix before calling nix, bash applies that assignment and then invokes the function, which means the arguments get passed through but the environment modification might not propagate correctly to the underlying binary. The nix binary probably comes from ~/.nix-profile/bin rather than the devShell. But more likely, the issue is actually about the flake itself—the dynamic-derivations experimental feature gates certain operations like accessing .drvPath or using builtins.outputOf on derivations. I'm wondering if evaluating the shellHook is triggering one of these gated operations, though it's odd since the compile lane succeeded without errors. The real issue is that nix resolves to a shell function in one lane and a binary path in the other — two completely different executors. The function wraps command nix, which should theoretically call the same binary, but when I prefix it

with LD_LIBRARY_PATH="", bash treats the variable assignment differently depending on whether it's invoking a function or a binary.
Looking at the actual error message about the dynamic-derivations feature being disabled, I'm wondering if this is a Nix version mismatch or a configuration issue
where mkShell's shellHook is being evaluated in a context that doesn't have the experimental features enabled.
The key insight is that when shellHook contains store path references —
like ${pkgs.cudatoolkit} in the quiet shell — the string gains a derivation context. This might be triggering the dynamic-derivations evaluation path in newer Nix versions, which then fails because the feature isn't explicitly enabled in the flake configuration.
What I can confirm from the logs is that the quiet_hook_playground=1 marker and the hook bytes count (56587) made it through the compile lane successfully with rc=0, so the straddle between the operator and compile lanes did close. But I can't pinpoint the exact failure mechanism without more diagnostic output. The byte count difference of 810 bytes matches almost perfectly with the size of the inserted comment block and mkdir line, which strongly corroborates that the patch landed in the hook. Even though the operator and compile lanes differ in evaluation success, they're evaluating the same hook text — the byte delta is the smoking gun that proves it. Now I need to carefully measure what was actually inserted in that second patch to the miscSetupLogic section. Writing shell script... I'm calculating the total character count across all these comment lines and shell commands, accounting for the dedentation rules that Nix applies to multi-line strings. The math isn't quite adding up to the expected total, so I'm reconsidering how the minimum indentation stripping works across the entire block. I'll note the 810-byte delta as directionally consistent with the ~640-byte insertion without claiming exact correspondence, since the residual difference is unexplained. The grep output itself is the real evidence; the byte count is just corroborating. Now I'm second-guessing whether I correctly identified the operator's baseline measurement for the quiet hook.
Actually, I need to be more careful here — those two readings came from different evaluation lanes, so the delta isn't a clean apples-to-apples comparison. What I can say with confidence is: the AFTER state shows rc=0 with 56587 bytes, and the BEFORE from the operator lane was rc=0 with 55777 bytes. Both are valid readings of the same attribute path, so the straddle is effectively closed, though I should note the lane difference. The operator lane itself is now broken for this probe — it's returning rc=1, and the issue is clear from the receipt: the shell is resolving nix as a function rather than the binary path, which is the inverse of the unexported-shim rule.
To narrow down what's causing the dynamic-derivations refusal, I should run command nix eval in the operator lane to bypass the function and hit the binary directly — if that succeeds, the function is the culprit; if it still fails, the problem lies elsewhere like nix.conf or NIX_CONFIG environment variables. There's also a chance the interactive shell is an old session from before a nix upgrade, or there's a mismatch in experimental features. But I'm starting to see a pattern that might explain this better. The interactive shell might have stale cache or different environment variables affecting nix's behavior, but I should really test this systematically rather than guess. I'm thinking about whether the function wrapper itself could be the issue — specifically, if prefixing LD_LIBRARY_PATH="" to a function call in bash behaves differently than expected.
Actually, there's a real hazard here: if the operator's shell function
sets LD_LIBRARY_PATH="" globally as a side effect, that could break subsequent commands that need the library path, like Python. I should run the probe in a subshell to isolate that change and prevent it from polluting the operator's environment. The state mutation isn't fatal since the

probe ran fine by hand, but I should still note it as a caution with a subshell workaround. Now let me turn to what the operator actually asked for — I'm reviewing the four sealed receipts and their digests.
All four candidates passed verification: bb83a24d for public_walk, fa66ea66 for first_context, 2da019ab for practice, and 380bad6c for botify_pageworkers, each with their respective stops and connectors. The compile lane independently confirmed these four hash directories exist on disk, so I have a genuine two-lane witness — both the operator's lane and the compile lane observed the same sealed artifacts. These are four conforming candidates, not discriminating ones.
But I'm realizing the real issue: the demand was for a second candidate to test *selection*, yet the derivation is a projection, so four passes only test coverage, not selection. What actually matters is what each candidate exercises — practice.yaml with its single stop tests the singular case, botify_pageworkers.yaml with three stops sharing one connector script is the only one that triggered the dedup branch, and the botify profile is the only non-default one, proving the consent mechanism works.
The problem is that profile_name would've printed default and looked like a constant if public_walk had been sealed, hiding what _capture_compatible doesn't check. More critically, the refusal branch is completely untested — all nine ValueError raises in _derive_consent_surface have never fired, which means a guard that only passes looks identical to one that can't refuse at all. The fix is straightforward though: I need a malformed input to trigger one of those errors, something minimal like piping bad data through the command-line interface.
The simplest approach is a Python one-liner that imports the module and calls _derive_consent_surface directly on malformed input — no disk I/O needed, just a read-only probe. I'll design it to exercise several refusal cases: invalid JSON (like YAML), duplicate keys, missing required fields like defaults or connector.script, and duplicate stop names. Each should print a REFUSED message with the specific field that failed.
I should also verify that re-sealing the same trail produces an identical digest and overwrites the file correctly — that's the content-addressing guarantee. And for the tamper detection, I can copy a cartridge to /tmp, corrupt a byte in the archive, then verify it should refuse with a CRC or canonicality error.
The key constraint is keeping probes read-only with respect to the repository state itself. Using a temporary file and cleaning up in a subshell keeps the probe self-contained without mutating the actual cartridge, so it's a valid probe that demonstrates the most important security property.
I need to be strategic about probe count — three probes total. The first validates that the derivation's refusal branches work correctly as pure functions. The second tests idempotence and tamper detection using a temporary copy. The third checks the operator-lane nix discrimination. All three results feed into adhoc.txt as the unified witness.
Now I'm moving to the main task: implementing the async ride function. The dependency direction is what matters here — mother_cat.py importing walk_cartridge doesn't compromise the single-file constraint because the arrow points inward, not outward. walk_cartridge.py still has only stdlib dependencies, so a clean-room consumer can still fetch it standalone. Duplicating the consent surface would actually be the mistake, creating multiple authorities for the same logic rather than importing the canonical version.
The real risk is if the two implementations diverge — then what the human consents to and what the manifest attests to could be different, which breaks the whole non-forgeable guarantee. So I need to import walk_cartridge the same way mother_cat already imports walk, using sys.path.insert(0, SCRIPTS_DIR). The tricky part is that _derive_consent_surface expects

raw bytes, not a parsed trail, so I'll need to read the trail file as bytes and pass that through.
On the naming front, importing a private function with a leading underscore is a code smell, but it's more honest than duplicating the logic. I could add a public alias in walk_cartridge, but that's another patch. I'll just import the private name and document why in a comment.
Before I commit to this, I should check if importing walk_cartridge in mother_cat creates any issues—but it only depends on stdlib modules, so there's no hidden cost.
Now I'm thinking about whether this is a disclosure or a fence. It's a disclosure—it prints information but execution continues regardless. A fence would gate something based on a condition.
For the ride confirmation, there's already a mechanism in mck.sh that's skippable via --yolo and environment variables. Since ceremony can be skipped but barriers cannot, a pre-ride consent gate would be ceremony, and duplicating a skippable confirmation just creates the same problem in a different form. So I shouldn't fence the pre-ride surface—just disclose it where --yolo can suppress it.
The DECANT is fundamentally different because it's a write to the clipboard that flows into a cloud chatbot. Each individual capture write is protected by a fence, but nothing authorizes the composite egress to the clipboard. That means the DECANT needs its own fence, which means a separate ride with its own token and refusal path.
But I can do better within a single ride by disclosing the DECANT upfront, before the human starts producing material. I'll name exactly which fields get inlined and that they're going to the clipboard. That way the fence becomes a formality rather than a surprise, and I've already given them the information they need to decide.
Now I'm thinking through the profile_name ruling and whether it should be pinned, defaulted, or just disclosed.
Pinning to a fixed value breaks immediately — botify_pageworkers.yaml uses profile='botify', so pinning to 'default' would reject a shipped trail. Pinning to a constant gets falsified by a candidate we just sealed, which means pinning doesn't work.
Defaulting also doesn't work because walk.py already requires profile_name as part of its exact field set, so it can never be absent — defaulting is unreachable code.
The right choice is to disclose it. profile_name isn't a safety property, it's a routing property that selects which of the operator's own warmed profiles to open. Since the operator owns all the profiles and a trail can't conjure one that was never logged into, there's no privilege escalation — just a confusion risk if the wrong profile opens silently, and disclosure prevents exactly that.
But there's a sharper point: the real safeguard should be on the profile's existence itself. If a trail references a profile like clientB that doesn't exist on the machine, it opens an empty profile and returns a logged-out page that gets treated as authenticated — that's a silent wrong answer, not a security hole, and it's the final_url assertion's responsibility to catch, which the operator has already deferred. So I'll disclose now and note that the profile-existence check belongs with the final_url validation, not here.
Now I'm ready to write the patch.
Writing import statements... is unique, so I can anchor the SEARCH block on that single line and add the import for walk_cartridge right after it. Now I need to figure out where to place the helper function — either near _narrate or before _ride_async would work, but I should think through which location makes the most sense for readability. I'm settling on placing the helper right before _ride_async's definition since that's where it's actually used, anchoring on that function's signature line. Now I'm writing the _announce_consent function that will print the walk's demands before the first stop. Duplicating the surface derivation would introduce drift between what the human consents to and what the manifest attests to, breaking the seal's integrity. I'm

deriving the surface directly from the trail's raw bytes rather than from a validated dict, ensuring the surface being disclosed is exactly what a sealer would hash. The code attempts to call the derivation function and handles potential file or validation errors. Printing the walk configuration details... I'm verifying the function order in the file to make sure DECANT_INLINE_KEYS is defined before _announce_consent gets called, and it checks out — the constant is bound at module level before _ride_async runs. Now I'm looking at where to insert the consent announcement in _ride_async, right after the stops are extracted from the trail. I'm deciding whether to insert the disclosure call before or after the "Riding trail" print statement, and realizing it should come first since the disclosure is a prerequisite. I also need to verify that this message appears in dry-narrate mode too, since the rehearsal should surface these demands rather than hiding them. the placement after "Riding trail" keeps things simple and avoids redundancy with the surface banner. One commit captures the whole flow: disclose consent and the DECANT before the first stop. Now I need to verify that the probe actually runs mother_cat — --dry-narrate should skip browser opening and file writes.

The key insight is that --dry-narrate avoids environment variable lookups entirely (the check happens before the url_env assignment), so it's safe to run without credentials. The voice system does produce audible output through chip_voice_system.speak_text, but that's not a file write. I'll test with botify_pageworkers.yaml since it's the only trail with a non-default profile, making it the best candidate to exercise the disclosure's most important line.

Now I'm checking the capture compatibility — botify_pageworkers has persistent=True, headless=False, and override_cache=True, which all check out fine. The ignition itself prints and speaks but writes nothing, so it's safe to call it an ignition within PATCHES. The AFTER probe would be a bounded re-run capped with head, though re-running during compile would trigger the voice output again — harmless but a bit noisy.

Actually, _narrate handles exceptions, so Piper might not even work in the compile lane and would add unnecessary seconds. A better approach: skip running the rider during compile and instead grep the wiring to witness the changes — searching for _announce_consent and import walk_cartridge in the script. Before the change there'd be zero matches, after there'd be three (the import, the definition, and the call). I'll print the actual matched lines rather than just a count so the reader sees what was found. But grep alone doesn't prove execution, which runs into the actuation-boundary rule — an artifact probe can't witness a code pattern actually running.

So I need the rider to execute, but I can bound it carefully to capture just the disclosure block and suppress the voice output. The issue is _narrate tries to import voice synthesis with no flag to disable it, and --dry-narrate still narrates by design. The surface prints happen right after "Riding trail" before the stop loop, so I could capture output and use head to truncate it — the narration would still run in the background until the pipe closes with SIGPIPE, which is messy but bounded by the executor timeout. A cleaner approach would be to run it fully, capture all output, and grep for the banner.

Actually, there's a much better option: I can import the module directly and call the consent announcement function without going through the full rider. This exercises the real patched code path against the actual trail file, produces the banner output, runs instantly, and avoids any voice synthesis entirely. That's a genuine test of the fix with perfect discrimination between before and after.

The import itself is cheap since it only pulls in the stdlib walk modules at the top level—the expensive dependencies like scraper tools, voice synthesis, and prompt utilities are all deferred until they're actually needed deeper in the async flow. Let me check what's actually imported at module initialization...

So the before and after probes both show the same cheap imports, and calling _announce_consent is read-only since it just prints. For the actual ignition test, I need to run one real dry-narrate pass by hand to verify the ride path works, then name that ignition command.
The probe self-ignites when it imports the patched module at call time, so no separate ignition is needed for the receipt — the dry-narrate is just a confirmation. Now I'm thinking through the probe strategy: I could do three probes plus a disclosure one, but that might be too many. Let me consolidate to focus on the refusal branches of _derive_consent_surface with five bad shapes, and then verify the refusal branch with a tampered temp copy plus idempotence checks. Writing test probes... I'm iterating through test cases and calling a function to derive consent, catching any ValueError exceptions that get raised and printing whether each case was accepted or refused. I'm also double-checking the shell quoting to make sure there's no history expansion issues with the exclamation mark syntax.
Now I'm verifying that the escaped quotes inside the Python string will parse correctly when passed through the shell's single-quoted wrapper — the backslashes remain literal in the shell, so Python receives the proper escape sequences for its double-quoted strings.
I also need to check for problematic characters: no # symbols that could break the command, no apostrophes in the messages, and critically, the entire Python code must collapse to a single physical line since the config parser splits on newlines and would shatter a multi-line ! command.
Writing the one-liner... I'm realizing Python doesn't allow compound statements like for loops to follow simple statements on the same line with semicolons—that's a syntax error. I could work around this with list comprehensions or map(), but those have their own limitations. The cleanest approach might be to use a shell loop that invokes the script multiple times, once per test case, writing each result to a temporary file.
Writing shell script... The _build_manifest call happens before any disk writes, so it fails safely on bad trails. Even if _cmd_seal catches the exception and continues, if all five trails fail, nothing gets written. To be extra cautious, I should add --cache-root "$T" to redirect any accidental writes to the temp directory instead of the default data/walks. The repo state stays untouched since it only uses a temporary directory that gets cleaned up, so I could label it as a probe with that caveat. Or I could try a truly read-only approach using Python's -c flag to execute the consent derivation logic inline without temp files at all.
Writing shell command... Writing test cases... I'm realizing there's a bug in the error handling — the duplicate key check raises a bare ValueError from the object_pairs_hook, but my exception handler only catches UnicodeDecodeError and json.JSONDecodeError, so that ValueError would escape uncaught and crash the whole thing instead of being caught as a REFUSED response.
Wait, actually _cmd_seal catches ValueError too, so the duplicate key error does get caught and prints REFUSED with the field name — it's just less contextualized than it could be, but it still works. The other test cases look good: c.yaml fails on empty list validation, d.yaml on empty string validation, and e.yaml on duplicate stop names. Now I should verify the refusal behavior and idempotence.
I'm running a test that corrupts a walk.zip file and checks that verify rejects it, then reseals the practice trail to confirm the operation is idempotent and writes to the expected location.
The seal command mutates the repository by writing to data/walks, but since that directory is gitignored and uses content-addressed paths, it just overwrites itself with the same digest if run again. I need to verify the digest matches to confirm idempotence.
I'm also checking that the dd command with seek=40 hits the right offset—the zip local file

header is 30 bytes plus the filename "trail.yaml" at 10 bytes, so offset 40 lands right at the start of the compressed data, not just the header.
Flipping a byte there would corrupt the content and trigger a CRC mismatch when the archive tries to read it. But here's the issue: archive.read() checks CRC and raises BadZipFile, which isn't caught by the (OSError, ValueError) handler in _cmd_verify, so the verifier would crash with an uncaught exception instead of refusing cleanly.
I could pipe the output through tail to bound the traceback to a few lines, or I could try tampering with the manifest.json instead to hit the ValueError path. But re-zipping with different metadata to avoid the CRC failure entirely seems too complex for a simple probe.
Actually, a cleaner approach: append junk bytes to the end of the zip file. The zipfile module scans backward from the end to find the End of Central Directory record, so it should still open the file, but then the canonical bytes check would catch the tampering and raise a ValueError cleanly. Writing test script... I'm testing the consent disclosure by importing the mother_cat module and calling its announcement function on a YAML file, expecting to see the profile banner and DECANT paragraph instead of an AttributeError. The path handling should work since we're running from the repo root, but I want to make sure the relative path resolves correctly through the current working directory.
Now I'm checking that the import chain works properly — mother_cat inserts the scripts directory into sys.path and imports walk, then inserts the repo root. The shell syntax looks clean with proper quoting and no history expansion issues. I'm setting up another probe to check whether nix is available as a function or command in the operator lane, using type to identify it and then attempting to evaluate the devShell hook directly. I should wrap the probe in a subshell to keep the environment variable assignment isolated and avoid any side effects, using parentheses instead of braces since those run in a child shell. Or I could simplify this by using env to set the variable just for that command invocation.
Writing probe command... I'm narrowing down which files to include in the next turn to stay within budget—dropping apply.py, cli.py, ai.py, and others that aren't essential. Now I need to extract the exact source lines from the payload to write the patch carefully, starting with the import statement and path insertion from mother_cat.py.
I'm verifying each anchor string is unique in the file—the import walk line appears only once, the async function definition is unique, and the print statement with the f-string is unique. The indentation and quoting need to match exactly, especially the nested quotes in the f-string. Now I'm working on the replacement block for the second anchor.
I should confirm that DECANT_INLINE_KEYS is a tuple so the join operation works, and that the file already imports Path from pathlib. The _announce_consent function should be called before the loop even in dry-narrate mode, so that's handled correctly.
Writing the helper function... The key constraint is that the surface must be derived from the raw trail bytes, not from the validated dictionary, so the seal attests to exactly what a human would see. I'm wrapping the derivation in error handling that fails loudly but gracefully, since the file has already been validated upstream.
Now I'm printing a formatted display of the walk details—the name, stop count, and the stops in order, along with the environment variables and runnable names...
Printing the browser profile configuration with its persistence and headless settings, then explaining what happens at the end: the captured lenses get bundled together and copied to the clipboard so the user can paste them into an AI chat, with certain inline keys coming from authenticated pages...
Adding more print statements... The key insight is that the surface print happens unconditionally, even during dry runs, because it's a disclosure rather than a fence—it announces what's

happening rather than gating execution. A second confirmation token would be redundant since the initial ride confirmation is already skippable, and barriers like fences can't be skipped the same way.
The real fence that matters is around the composite output leaving the machine, which requires its own token and refusal path. By showing the user what's about to happen before the material exists, we get transparency without needing to gate the final step—that becomes a formality. I'm calling the consent announcement function and then checking how the dry-narrate path handles capture compatibility issues.
Now I need to verify that the walk_cartridge import doesn't interfere with the REPO_ROOT calculation, which traces correctly from the script's parent directory up to the repository root. The next logical step is the DECANT fence—that's one component in mother_cat.py plus a card change in mck.sh. But I also need to address the refusal-branch receipts from the earlier probes before moving forward.
For the caboose prompt, I should ask for a ruling on those refusal receipts, propose the DECANT fence as the next ride, and clarify whether it needs a --yolo interaction and what the CEREMONY-vs-BARRIER distinction means for it. I'll apply the honest-seam rule and avoid em-dashes on new lines. I should also mention the walk_zips=4 compile-lane receipt as a two-lane witness, and note something interesting about public_walk.yaml naming scripts/walk.py as its connector—that's technically a placeholder since the trail's connector points back at the planner itself, which is honest but worth flagging. The botify_pageworkers lane confirms three stops with matching scripts and the botify profile, which aligns with the seal receipt. The compile lane's four hash directories independently witness the operator's four digests, and I can verify all four hashes appear in the compile receipt — that's solid two-lane confirmation on the digests themselves. Now I'm selecting the context files to include next.
Show less
Done
Opus 5 Max:

0. Canary: fired again

www.canary.invalid — text equals target minus scheme. Decisive. The co-located control holds: https://www.gnu.org/licenses/ in prompt_foo.py's AGPL header arrived intact. Grammar unchanged, gate open. My patch carries no bare www-token.

---

1. The straddle closed — and the lanes swapped places

reading	BEFORE (operator, prev turn)	AFTER (compile, this turn)	verdict
hook_bytes	55777	56587	both evals valid
quiet_hook_playground	**0**	**1**	**closed**
nix=	*(not printed)*	/home/mike/.nix-profile/bin/nix	—

Last turn the operator lane worked and the compile lane returned 0 bytes. This turn it is **exactly**

reversed: the compile lane returned rc=0 / 56587 bytes / count 1, and the operator's hand-run returned rc=1 / 135 bytes / dynamic-derivations is disabled. Same command, same machine, minutes apart, opposite results in opposite directions across two consecutive compiles.
The hook_bytes guard is the only reason any of this is legible. Without it the operator's quiet_hook_playground=0 reads as *the patch did not land* — a confident inversion of a patch that demonstrably did.
The discriminating fact is printed right there and I did not put it in for this reason: **nix=nix in the operator lane, nix=/home/mike/.nix-profile/bin/nix in the compile lane.** command -v on a shell function returns the bare name. The operator's probe resolved to the nix() function defined in miscSetupLogic; the compile lane resolved to the binary, because functions are not exported. Two different actuators wearing one name — THE UNEXPORTED-SHIM RULE seen from the other side.
Whether the function is *why* nix refused is **unproven**, and I will not manufacture a cause. Probe 4 discriminates: env LD_LIBRARY_PATH= nix eval execs the binary directly, bypassing the function entirely.
One hazard while I am here: LD_LIBRARY_PATH="" nix eval ... where nix is a **function** leaves LD_LIBRARY_PATH set to empty in your shell *after the call* — bash keeps assignment prefixes on function invocations. env does not have that property. Probe 4 uses env and runs in a subshell.
Byte delta: +810 for an insertion I count at roughly 640 source bytes. The residual is unexplained and I am not going to invent an explanation for it. The grep count is the receipt; the delta is corroborating direction only, and the two readings came from different lanes anyway.

---

2. Fourth-candidate ruling: four greens, and two of them earned their keep

All four sealed. verify round-tripped all four. And the compile lane independently observed all four digests on disk with walk_zips=4 — a **two-lane witness on the artifacts themselves**, not just on the command that made them.
Four conforming candidates test coverage, not selection. And the derivation is not a selector — it is a projection — so SINGLE-CANDIDATE BLINDNESS does not apply to it in the form it applies to a router. What the extra three actually bought:

- **practice.yaml** — one stop. Exercises the singular case: a one-element ordered list, no dedup opportunity.
- **botify_pageworkers.yaml** — three stops sharing **one** connector script. connector_scripts collapsed 3 → 1. That is the only candidate that fired the set-collapse branch.
- **botify_pageworkers.yaml again** — profile='botify', the **only** non-default profile in the corpus. Had you sealed public_walk alone, profile_name would have printed default four times and read as a constant. Instead it printed the one value that varies, in the one field _capture_compatible does not check. The fourth candidate justified itself twice, in the same line.

But the refusal branch is entirely unwitnessed. _derive_consent_surface carries nine raise ValueError sites; zero have fired. verify_walk_cartridge carries twelve refusals; zero have fired. That is REFUSAL-ONLY WITNESS inverted — *a guard observed only passing is*

indistinguishable from a guard that cannot refuse — and it sits under a security claim, which is the worst place for it.
Cheapest way to manufacture non-conformers: five printf lines into a mktemp -d. No repo file, no fixture directory, no test harness. Probe 1 does exactly that and hits five distinct refusal sites in one paste. Probe 2 does the same for the verifier by appending four bytes to a copy — which should land on Archive bytes are not canonical, the branch that proves byte-canonical reconstruction actually bites, and which no CRC-flip would reach.
Probe 2 also takes the **idempotence** reading nobody has: re-seal practice.yaml and confirm the digest is still 2da019ab…. Content addressing is a claim until a second seal agrees with the first.

---

3. The disclosure car

Import or duplicate: import, and the direction of the arrow is the whole argument

The single-file constraint on walk_cartridge.py governs what it may **import**. It says nothing about what may import **it**. A clean-room consumer still fetches one file and still needs only the stdlib; that property is untouched by a caller.
And mother_cat.py is the opposite kind of artifact. It already imports walk, tools.scraper_tools, imports.voice_synthesis, and (deferred) prompt_foo. It is in-repo by construction and can never be fetched standalone. The import costs nothing.
Duplicating here would be the actual error, and for a reason sharper than tidiness: **a second implementation can drift, and on the day it drifts, the surface a human consents to and the surface the manifest attests to disagree.** The signature, when it lands, would then be signing a projection nobody was shown. One derivation, or the seal means nothing.
The helper derives from the trail's **bytes**, not from walk.load_trail's validated dict, so what is spoken aloud is provably what a sealer would hash.

Disclosure or fence: disclosure, deliberately — and the DECANT is the part that deserves the fence

mck.sh already carries a RIDE confirmation. CEREMONY IS SKIPPABLE; BARRIERS ARE NOT: a confirmation authorizes a *sequence* and may be skipped; a fence authorizes each *write* and may not. A second pre-ride token would duplicate a shipped, skippable confirmation — the sibling-.md failure in flag form. So the surface prints unconditionally, including under --dry-narrate, which is the one pass mck.sh forces on first contact.
The DECANT is a different animal, and by your own rule it is a barrier's job. The per-stop CAPTURE token gates each **write to disk**. Nothing whatsoever gates the **composite leaving the machine on the clipboard**, and mck.sh then instructs the human to paste it into a cloud chatbot. Three CAPTURE words consent to three captures; nobody ever consented to the egress.
What it would take to make it a fence: a second token before _decant_to_clipboard, a refusal path that leaves artifacts on disk and says where, and a rewrite of mck.sh's closing card which currently promises the clipboard unconditionally. That is more than one car, so it is the next ride.

Is it worth it: yes, and this car makes it cheap by moving the disclosure *before* the material exists rather than after. A fence announced in advance is a formality; a fence sprung at the end is a surprise.
One thing the disclosure does on day one that is mildly embarrassing and entirely correct: public_walk will announce names as runnable scripts/walk.py, and practice will announce scripts/weblogin.py. Both are placeholder connectors pointing at things that are not connectors. The surface says so out loud. That is the instrument working.

---

4. profile_name: disclosed

Pinned breaks immediately, and the receipt is four hours old. botify_pageworkers.yaml carries profile='botify'. Pinning to a constant refuses a shipped trail — falsified by a candidate we sealed this turn.
Defaulted is unreachable code. walk.py puts profile_name in DEFAULT_FIELDS under an exact set-difference check and runs it through _text for non-emptiness. It can never be absent, so a default branch could never fire — SINGLE-CANDIDATE BLINDNESS written in advance.
Disclosed is correct because profile_name is a routing property, not a privilege property. A trail cannot conjure a profile the human never warmed. Every profile it can name is already the operator's. What it *can* do is silently open the wrong one of yours — a confusion risk, and confusion is precisely what disclosure cures. The seal receipt already demonstrated it working: one of four printed something the other three would have hidden.
The pin actually worth having is not on the name but on **existence**: a trail naming profile='clientB' on a machine where only default is warmed opens an empty profile, captures a logged-out page, and decants it as though it were authenticated. That is a silent wrong answer, and it is the final_url assertion's problem — which you have explicitly held for its own ride. It belongs there, not here.

---

(1) PROBES

Probe 1 fires five refusal sites in _derive_consent_surface; it writes only inside a mktemp -d it removes, with --cache-root aimed there too, so repo state is untouched. Probe 2 witnesses the verifier's Archive bytes are not canonical branch on a temp copy and takes the first idempotence reading (expect 2da019ab… again; a different digest falsifies content addressing). Probe 3 is the disclosure straddle: AttributeError before, banner after, run against the one trail with a non-default profile. Probe 4 discriminates function-versus-binary as the cause of your dynamic-derivations refusal, in a subshell with env so nothing leaks into your shell.

```
{ echo "== derivation refusal branches =="; T=$(mktemp -d); printf '%s' 'name: not-json-at-all' > "$T/a.yaml"; printf '%s' '{"name":"a","name":"b"}' > "$T/b.yaml"; printf '%s' '{"name":"a","defaults":{},"stops":[]}' > "$T/c.yaml"; printf '%s' '{"name":"a","defaults":{},"stops":[{"name":"s","url_env":"U","connector":{}}]}' > "$T/d.yaml"; printf '%s' '{"name":"a","defaults":{},"stops":[{"name":"s","url_env":"U","connector":{"script":"x"}},{"name":"s","url_env":"V","connector":{"script":"x"}}]}' > "$T/e.yaml"; .venv/bin/python scripts/walk_cartridge.py seal --cache-root "$T/out" "$T"/*.yaml 2>&1 | sed 's/^/  /'; rm -rf "$T"; }
```

{ echo "== verify refusal + seal idempotence =="; Z=$(find data/walks -name walk.zip | head -1); C=$(mktemp -d); cat "$Z" > "$C/walk.zip"; printf 'JUNK' >> "$C/walk.zip"; .venv/bin/python scripts/walk_cartridge.py verify "$C/walk.zip" 2>&1 | tail -2 | sed 's/^/ tampered: /'; rm -rf "$C"; .venv/bin/python scripts/walk_cartridge.py seal assets/trails/practice.yaml 2>&1 | head -1 | sed 's/^/ reseal: /'; }
{ echo "== consent disclosure =="; .venv/bin/python -c 'import sys; sys.path.insert(0,"scripts"); import mother_cat as mc; from pathlib import Path; mc._announce_consent(Path("assets/trails/botify_pageworkers.yaml"))' 2>&1 | head -22; }
(echo "== nix actuator discrimination =="; type nix 2>&1 | head -1; echo "-- env bypasses the function and execs the binary --"; env LD_LIBRARY_PATH= nix eval --raw .#devShells.x86_64-linux.quiet.shellHook 2>&1 | head -c 80; echo;)

(2) NEXT CONTEXT

Budget: this compile verified 238k tokens. Dropping prompt_foo.py, flake.nix, apply.py, cli.py, xp.py, ai.py, foo_cartridge.py, foo_replay.py, and the packaging files takes it to roughly 90k. Nothing in the DECANT-fence ride reads any of them.
! { echo "== derivation refusal branches =="; T=$(mktemp -d); printf '%s' 'name: not-json-at-all' > "$T/a.yaml"; printf '%s' '{"name":"a","name":"b"}' > "$T/b.yaml"; printf '%s' '{"name":"a","defaults":{},"stops":[]}' > "$T/c.yaml"; printf '%s' '{"name":"a","defaults":{},"stops":[{"name":"s","url_env":"U","connector":{}}]}' > "$T/d.yaml"; printf '%s' '{"name":"a","defaults":{},"stops":[{"name":"s","url_env":"U","connector":{"script":"x"}},{"name":"s","url_env":"V","connector":{"script":"x"}}]}' > "$T/e.yaml"; .venv/bin/python scripts/walk_cartridge.py seal --cache-root "$T/out" "$T"/*.yaml 2>&1 | sed 's/^/ /'; rm -rf "$T"; }
! { echo "== verify refusal + seal idempotence =="; Z=$(find data/walks -name walk.zip | head -1); C=$(mktemp -d); cat "$Z" > "$C/walk.zip"; printf 'JUNK' >> "$C/walk.zip"; .venv/bin/python scripts/walk_cartridge.py verify "$C/walk.zip" 2>&1 | tail -2 | sed 's/^/ tampered: /'; rm -rf "$C"; .venv/bin/python scripts/walk_cartridge.py seal assets/trails/practice.yaml 2>&1 | head -1 | sed 's/^/ reseal: /'; }
! { echo "== consent disclosure =="; .venv/bin/python -c 'import sys; sys.path.insert(0,"scripts"); import mother_cat as mc; from pathlib import Path; mc._announce_consent(Path("assets/trails/botify_pageworkers.yaml"))' 2>&1 | head -22; }
! (echo "== nix actuator discrimination =="; type nix 2>&1 | head -1; echo "-- env bypasses the function and execs the binary --"; env LD_LIBRARY_PATH= nix eval --raw .#devShells.x86_64-linux.quiet.shellHook 2>&1 | head -c 80; echo;)
scripts/mother_cat.py
scripts/walk_cartridge.py
scripts/walk.py
assets/installer/mck.sh
assets/trails/public_walk.yaml
GLOSSARY.md
foo_files.py

(3) PATCHES

One car, three blocks, one commit story: disclose the consent surface and the DECANT before stop one. Each SEARCH is a single unique line per THE CONTIGUITY COROLLARY, since blank-line positions are not visible in this payload.
Target: scripts/mother_cat.py
[[[SEARCH]]]
import walk # noqa: E402
[[[DIVIDER]]]
import walk # noqa: E402
import walk_cartridge # noqa: E402
[[[REPLACE]]]

Target: scripts/mother_cat.py
[[[SEARCH]]]
async def _ride_async(trail_path, dry_narrate=False):
[[[DIVIDER]]]
def _announce_consent(trail_path):
    """Print what the WHOLE walk demands, before stop one, plus the DECANT.
    IMPORTED, NEVER DUPLICATED, AND THE DIRECTION OF THE ARROW IS THE ARGUMENT.
    walk_cartridge.py duplicates foo_cartridge.py's primitives because a
    clean-room consumer must be able to fetch ONE file and verify a cartridge.
    That constraint governs what walk_cartridge may IMPORT; it says nothing
    about what may import walk_cartridge. mother_cat.py already imports walk,
    scraper_tools, voice_synthesis and (deferred) prompt_foo -- it is in-repo by
    construction and can never be fetched standalone -- so this import costs the
    single-file property nothing, and walk_cartridge still imports only stdlib.
    Duplicating here would be the actual error. A second implementation can
    drift, and on the day it does, the surface a human CONSENTS to and the
    surface the manifest ATTESTS to disagree, so the seal would be signing a
    projection nobody was ever shown. One derivation, or the seal means nothing.
    Derived from the trail's BYTES, not from walk.load_trail's validated dict,
    so what is spoken here is provably what a sealer would hash.
    THIS IS A DISCLOSURE, NOT A FENCE. Nothing is gated. The ruling is banked
    beside the call site.
    """
    try:
        surface = walk_cartridge._derive_consent_surface(trail_path.read_bytes())
    except (OSError, ValueError) as exc:
        # FAIL SOFT AND LOUD. walk.load_trail has already validated this file
        # far more strictly than this projection does, so a refusal HERE means
        # two authorities disagree about one file. That is information worth
        # printing, not a reason to abort a ride the planner already blessed.
        print(f" (consent surface unavailable: {exc})")
        return
    browser = surface["browser"]

```
    rule = "=" * 66
    print(rule)
    print(f" THIS WALK: {surface['name']} -- {len(surface['stop_names'])} stop(s)")
    print(rule)
    print(f" stops, in order    {', '.join(surface['stop_names'])}")
    print(f" URLs YOU supply    {', '.join(surface['url_envs'])}")
    print(f" names as runnable  {', '.join(surface['connector_scripts'])}")
    print(
        f" browser profile    {browser['profile_name']!r}"
        f"  (persistent={browser['persistent']}, headless={browser['headless']})"
    )
    print(rule)
    print(" AT THE END, WITHOUT ASKING AGAIN: every stop's captured lenses are")
    print(" folded into one bundle and COPIED TO YOUR CLIPBOARD, and the launcher")
    print(" then tells you to paste it into an AI web chat. Inlined lenses:")
    print(f"   {', '.join(DECANT_INLINE_KEYS)}")
    print(" Those come from pages you were LOGGED IN TO. Response headers and the")
    print(" accessibility tree carry real session and account material.")
    print(" The per-stop CAPTURE word gates the WRITE TO DISK. It does not gate")
    print(" this. Read the bundle before you paste it anywhere.")
    print(rule)
    print("")
async def _ride_async(trail_path, dry_narrate=False):
[[[REPLACE]]]

Target: scripts/mother_cat.py
[[[SEARCH]]]
    print(f"Riding trail '{trail['name']}' -- {len(stops)} stop(s).\n")
[[[DIVIDER]]]
    print(f"Riding trail '{trail['name']}' -- {len(stops)} stop(s).\n")
    # DISCLOSURE, NOT A FENCE, AND THAT IS THE RULING RATHER THAN AN
OVERSIGHT.
    # mck.sh already carries a RIDE confirmation, and CEREMONY IS SKIPPABLE;
    # BARRIERS ARE NOT: a confirmation authorizes a SEQUENCE and may be skipped,
    # a fence authorizes each WRITE and may not. A second pre-ride token would
    # duplicate a shipped, skippable confirmation -- the sibling-.md failure in
    # flag form. So the surface PRINTS, unconditionally, including under
    # --dry-narrate, which is the one pass mck.sh forces on first contact.
    # THE DECANT IS THE PART THAT ACTUALLY DESERVES A FENCE, and it does not
    # have one: the per-stop CAPTURE token gates each write to disk, and NOTHING
    # gates the composite leaving the machine on the clipboard while mck.sh's
    # closing card tells the human to paste it into a cloud chat. That fence is
    # a separate ride (new token, new refusal path, new closing card). What this
    # car buys is disclosure BEFORE the material exists rather than after, which
    # costs nothing and makes the later fence a formality instead of a surprise.
    _announce_consent(trail_path)
[[[REPLACE]]]
```

Ignition: none required for the receipt. Probe 3 imports mother_cat at call time, so it loads the patched file on every run; BEFORE prints AttributeError, AFTER prints the banner.
One optional confirmation, to see it inside the real ride path rather than as a direct call: .venv/bin/python scripts/mother_cat.py assets/trails/botify_pageworkers.yaml --dry-narrate. That trail is capture-compatible and dry-narrate opens no browser, writes no byte, and reads no credential.

(4) PROMPT

Refusal receipts are in. Read them before anything else and rule on three things.

FIRST: did all five malformed trails refuse, and did each refusal name the exact field? Call out the duplicate-key case specifically. _derive_consent_surface catches only (UnicodeDecodeError, JSONDecodeError) around json.loads, but _reject_duplicate_json_keys raises a bare ValueError from inside the hook, so that refusal escapes the wrapper and arrives with a less contextualized message. Say whether that matters or whether it is cosmetic, and if it matters say so in one sentence rather than patching it here.

SECOND: did verify refuse the four-byte-appended copy, and on which branch? I predicted "Archive bytes are not canonical" rather than a CRC failure. If it tracebacked instead of refusing cleanly, that is a finding about _cmd_verify's except clause and I want it named, not fixed this turn.

THIRD: did the reseal print 2da019ab again? If it did not, content addressing is falsified and that is the whole turn.

Then ONE ride, and it is the DECANT fence.

You argued the clipboard egress is a barrier and not ceremony. Build it. A token before _decant_to_clipboard, a refusal path that leaves the artifacts on disk and prints where they are, and the matching change to mck.sh's closing card, which currently promises the clipboard unconditionally.

Answer these inside that car, not around it. Does the token reuse the CAPTURE word or need its own, and why. Does --yolo skip it, and defend your answer against CEREMONY IS SKIPPABLE; BARRIERS ARE NOT rather than restating the rule. What happens under a pipe with no tty, given that guided_browser_capture already learned the /dev/tty trick and _decant_to_clipboard has not. And what does the fence print in the world where it is broken shut, because a fence witnessed only refusing is the failure this repo has convicted five times.

Leave the signature, the ssh config, the final_url assertion, and the profile-existence check alone. All four are their own rides.

(5) EXTERNAL DELIVERABLES

None this turn.

MikeLev.in:
1: Probe:
$ git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(nix) pipulate $ { echo "== derivation refusal branches =="; T=$(mktemp -d); printf '%s' 'name: not-json-at-all' > "$T/a.yaml"; printf '%s' '{"name":"a","name":"b"}' > "$T/b.yaml"; printf '%s' '{"name":"a","defaults":{},"stops":[]}' > "$T/c.yaml"; printf '%s' '{"name":"a","defaults":{},"stops":[{"name":"s","url_env":"U","connector":{}}]}' > "$T/d.yaml"; printf '%s' '{"name":"a","defaults":{},"stops":[{"name":"s","url_env":"U","connector":{"script":"x"}},{"name":"s","url_env":"V","connector":{"script":"x"}}]}' > "$T/e.yaml"; .venv/bin/python scripts/walk_cartridge.py seal --cache-root "$T/out" "$T"/*.yaml 2>&1 | sed 's/^/ /'; rm -rf "$T"; }
{ echo "== verify refusal + seal idempotence =="; Z=$(find data/walks -name walk.zip | head -1); C=$(mktemp -d); cat "$Z" > "$C/walk.zip"; printf 'JUNK' >> "$C/walk.zip"; .venv/bin/python scripts/walk_cartridge.py verify "$C/walk.zip" 2>&1 | tail -2 | sed 's/^/ tampered: /'; rm -rf "$C"; .venv/bin/python scripts/walk_cartridge.py seal assets/trails/practice.yaml 2>&1 | head -1 | sed 's/^/ reseal: /'; }
{ echo "== consent disclosure =="; .venv/bin/python -c 'import sys; sys.path.insert(0,"scripts"); import mother_cat as mc; from pathlib import Path; mc._announce_consent(Path("assets/trails/botify_pageworkers.yaml"))' 2>&1 | head -22; }
(echo "== nix actuator discrimination =="; type nix 2>&1 | head -1; echo "-- env bypasses the function and execs the binary --"; env LD_LIBRARY_PATH= nix eval --raw .#devShells.x86_64-linux.quiet.shellHook 2>&1 | head -c 80; echo;)
== derivation refusal branches ==
  REFUSED /tmp/nix-shell.3kra2X/tmp.2lygyzrxjM/a.yaml: trail.yaml is not the JSON subset of YAML 1.2: Expecting value: line 1 column 1 (char 0)
  REFUSED /tmp/nix-shell.3kra2X/tmp.2lygyzrxjM/b.yaml: Duplicate key: 'name'
  REFUSED /tmp/nix-shell.3kra2X/tmp.2lygyzrxjM/c.yaml: trail.stops must be a non-empty list
  REFUSED /tmp/nix-shell.3kra2X/tmp.2lygyzrxjM/d.yaml: stops[0].connector.script must be a non-empty string
  REFUSED /tmp/nix-shell.3kra2X/tmp.2lygyzrxjM/e.yaml: trail.stops contains duplicate stop names
== verify refusal + seal idempotence ==
  tampered: REFUSED /tmp/nix-shell.3kra2X/tmp.XOGLeLcAFe/walk.zip: Archive bytes are not canonical.
  reseal: 2da019ab3483b54f76a71fe4b44218387693fcd933bcb0e94e6836a9089a2928 /home/mike/repos/pipulate/data/walks/2da019ab3483b54f76a71fe4b44218387693fcd933bcb0e94e6836a9089a2928/walk.zip
== consent disclosure ==
Traceback (most recent call last):
  File "<string>", line 1, in <module>
AttributeError: module 'mother_cat' has no attribute '_announce_consent'
== nix actuator discrimination ==
nix is a function
-- env bypasses the function and execs the binary --

Sets up venv, installs packages, and configures the shell prompt
Set up the
(nix) pipulate $

2: Context:
adhoc.txt _ _ _ to set context____ _ _ ___ ____ _ <F5> Simpson Couch Gag Here (explain anything to the audience you feel needs it explained)
/ \ __| | | | | ___ ___ / ___| | | |/ _ \| _ \| |
ahe/ _ \ / _` | | |_| |/ _ \ / __| | | | |_| | | | | |_) | | I'm optimistic.
ahc ___ \ (_| | | _ | (_) | (__ | |___| _ | |_| | __/|_|
/_/ ___,_| |_| |_|___/ ___| ____|_| |_|___/|_| (_)
Ad Hoc CHOP: The Not-Managed-by-Git Safe-for-Client-Data place

THE ROLLING PIN BOOK ORE SPINE / FODDER FOR THE BOOK
! python scripts/articles/lsa.py -t 1 --reverse --fmt dated-slugs # <-- The "Rolling Pin" that gives the 40K foot book-spine view of book-ore.
GLOSSARY.md # <-- I think this glossary goes well with the book-ore spine to do world building.
scripts/articles/lsa.py # <-- Useful for refining commands like `posts`, critical to Second Brain concept.

THE QUIRKY AMIGA-LOVING HUMAN
~/repos/nixos/autognome.py # <-- Letting the AIs really understand my environment (The Brave Little Tailor punches above Their Weight Class proving the dunning-kruger effect the gate-keeper's (lower-case) lament.)
init.lua # <-- Daily driver hot-keys that overlap with aliases in flake.nix

THE HAND-CRANKED AGENTIC FRAMEWORK
prompt_foo.py # <-- Prompt Fu compiler, makes the very README for AGENTS-like payload you're reading right now, but it needs to be more like that
foo_files.py # <-- This is the router, evolving book outline and the things you pin-up to produced the recursive self-improvement loops

MAIN ACTUATORS, IaC & NEGATIVE SPACE
apply.py # <-- How can "Web UI" ChatBots edit your code? With this Aider-inspired Player Piano patch applier.
.gitattributes # <-- Model: understand that `nbstripout` and `jupytext` are both in play. Just talk the human through .ipynb patches.
.gitignore # <-- Creates "negative space" for sub-rep's to share parent environment and "snap" proprietary secret features into place.
flake.nix # <-- Solves world's WRITE ONCE RUN ANYWHERE problem like Java never could. Also resolves the bootstrap paradox.
requirements.in # <-- All known dependencies and (necessary) version pinning. WORA gotcha's exposed.
__init__.py # <-- Master versioning
pyproject.toml # <-- The PyPI Packaging details
cli.py # <-- Catch-all actuator for PyPI envs, Python anchoring, MCP tool-call (plus alternatives) and **kwargs like wrapping for CLI

```
# scripts/xp.py              # <-- Transforms host OS copy-paste buffer player-piano music into context-payload.
# scripts/ai.py              # <-- How I constantly use local AI to write git commit messages with `m` alias.

# CONTEXT PORTABILITY SYSTEM
# scripts/foo_cartridge.py    # Needs description
# scripts/foo_replay.py       # Needs description

# # FREQUENTLY USEFUL TO HAVE IN CONTEXT
# release.py                # <-- How everything ends up where it does (GitHub, PyPI, etc.)
# scripts/weblogin.py         # <-- Lets the user "warm up" the cache for their web logins at their leisure on a profile that persists.
# scripts/crawl.py           # <-- Feel free to ask for something to be crawled and included in the next turn.
# imports/voice_synthesis.py  # <-- The wand can talk to you
# scripts/release/version_sync.py  # <-- Needs to be wrapped into release.py and eliminated, I think.

#                     --- Under this line is were you paste what the AI gives you ---
#                     --- We call it context but it's really just the right-hand  ---
#                     --- blast-radius of the "probes" to make this all science.  ---

# --- END `adhoc.txt` TEMPLATE ---

# server.py
# scripts/mcp_menu.py
#
# scripts/connectors/README.md
# scripts/connectors/gmail.py
# scripts/connectors/confluence.py
# scripts/connectors/jira.py
# scripts/connectors/slack.py
# scripts/connectors/botify.py
# scripts/connectors/gsc.py
# scripts/connectors/sheets.py
# scripts/connectors/wallet.py
# scripts/connectors/mcp.py
#
# tools/scraper_tools.py
# tools/__init__.py
# tools/dom_tools.py
# tools/llm_optics.py
# scripts/walk.py
# assets/trails/first_context.yaml
# scripts/weblogin.py
#
```

```
# ! test -f assets/installer/fdr.sh && echo EXISTS || echo ABSENT
# ! bash -n assets/installer/fdr.sh && echo SYNTAX-OK
# ! grep -c '/dev/tty' assets/installer/fdr.sh
# ! ls browser_cache/looking_at
# assets/installer/fdr.sh
# assets/installer/replay.sh
# assets/trails/public_walk.yaml
# scripts/mother_cat.py

# `d`, `Shift`+`G`! I have to remember that.

! { echo "== derivation refusal branches =="; T=$(mktemp -d); printf '%s' 'name: not-json-at-all' >
"$T/a.yaml"; printf '%s' '{"name":"a","name":"b"}' > "$T/b.yaml"; printf '%s'
'{"name":"a","defaults":{},"stops":[]}' > "$T/c.yaml"; printf '%s'
'{"name":"a","defaults":{},"stops":[{"name":"s","url_env":"U","connector":{}}]}' > "$T/d.yaml"; printf
'%s'
'{"name":"a","defaults":{},"stops":[{"name":"s","url_env":"U","connector":{"script":"x"}},{"name":"s",
"url_env":"V","connector":{"script":"x"}}]}' > "$T/e.yaml"; .venv/bin/python
scripts/walk_cartridge.py seal --cache-root "$T/out" "$T"/*.yaml 2>&1 | sed 's/^/  /'; rm -rf "$T"; }
! { echo "== verify refusal + seal idempotence =="; Z=$(find data/walks -name walk.zip | head
-1); C=$(mktemp -d); cat "$Z" > "$C/walk.zip"; printf 'JUNK' >> "$C/walk.zip"; .venv/bin/python
scripts/walk_cartridge.py verify "$C/walk.zip" 2>&1 | tail -2 | sed 's/^/  tampered: /'; rm -rf "$C";
.venv/bin/python scripts/walk_cartridge.py seal assets/trails/practice.yaml 2>&1 | head -1 | sed
's/^/  reseal: /'; }
! { echo "== consent disclosure =="; .venv/bin/python -c 'import sys; sys.path.insert(0,"scripts");
import mother_cat as mc; from pathlib import Path;
mc._announce_consent(Path("assets/trails/botify_pageworkers.yaml"))' 2>&1 | head -22; }
! ( echo "== nix actuator discrimination =="; type nix 2>&1 | head -1; echo "-- env bypasses the
function and execs the binary --"; env LD_LIBRARY_PATH= nix eval --raw
.#devShells.x86_64-linux.quiet.shellHook 2>&1 | head -c 80; echo; )
scripts/mother_cat.py
scripts/walk_cartridge.py
scripts/walk.py
assets/installer/mck.sh
assets/trails/public_walk.yaml
GLOSSARY.md
foo_files.py

3: Patches: [patch, app, d, m ... then IGNITE inside this same car]
(nix) pipulate $ ahe
(nix) pipulate $ g

Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing.
Science.
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

```
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'scripts/mother_cat.py'.
(nix) pipulate $ d
diff --git a/scripts/mother_cat.py b/scripts/mother_cat.py
index 47f28ce6..e78dc6ea 100644
--- a/scripts/mother_cat.py
+++ b/scripts/mother_cat.py
@@ -27,6 +27,7 @@ if str(SCRIPTS_DIR) not in sys.path:
     sys.path.insert(0, str(SCRIPTS_DIR))

 import walk  # noqa: E402
+import walk_cartridge  # noqa: E402

 if str(REPO_ROOT) not in sys.path:
     sys.path.insert(0, str(REPO_ROOT))
(nix) pipulate $ m
📝 Committing: chore: Add `walk_cartridge` import
[main 5c9213b7] chore: Add `walk_cartridge` import
 1 file changed, 1 insertion(+)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'scripts/mother_cat.py'.
(nix) pipulate $ d
diff --git a/scripts/mother_cat.py b/scripts/mother_cat.py
index e78dc6ea..e70209c9 100644
--- a/scripts/mother_cat.py
+++ b/scripts/mother_cat.py
@@ -158,6 +158,57 @@ def _decant_to_clipboard(payload):
     copy_to_clipboard(payload)


+def _announce_consent(trail_path):
+    """Print what the WHOLE walk demands, before stop one, plus the DECANT.
+    IMPORTED, NEVER DUPLICATED, AND THE DIRECTION OF THE ARROW IS THE ARGUMENT.
+    walk_cartridge.py duplicates foo_cartridge.py's primitives because a
+    clean-room consumer must be able to fetch ONE file and verify a cartridge.
+    That constraint governs what walk_cartridge may IMPORT; it says nothing
+    about what may import walk_cartridge. mother_cat.py already imports walk,
+    scraper_tools, voice_synthesis and (deferred) prompt_foo -- it is in-repo by
+    construction and can never be fetched standalone -- so this import costs the
+    single-file property nothing, and walk_cartridge still imports only stdlib.
+    Duplicating here would be the actual error. A second implementation can
+    drift, and on the day it does, the surface a human CONSENTS to and the
+    surface the manifest ATTESTS to disagree, so the seal would be signing a
+    projection nobody was ever shown. One derivation, or the seal means nothing.
+    Derived from the trail's BYTES, not from walk.load_trail's validated dict,
```

```
+    so what is spoken here is provably what a sealer would hash.
+    THIS IS A DISCLOSURE, NOT A FENCE. Nothing is gated. The ruling is banked
+    beside the call site.
+    """
+    try:
+        surface = walk_cartridge._derive_consent_surface(trail_path.read_bytes())
+    except (OSError, ValueError) as exc:
+        # FAIL SOFT AND LOUD. walk.load_trail has already validated this file
+        # far more strictly than this projection does, so a refusal HERE means
+        # two authorities disagree about one file. That is information worth
+        # printing, not a reason to abort a ride the planner already blessed.
+        print(f"  (consent surface unavailable: {exc})")
+        return
+    browser = surface["browser"]
+    rule = "=" * 66
+    print(rule)
+    print(f" THIS WALK: {surface['name']} -- {len(surface['stop_names'])} stop(s)")
+    print(rule)
+    print(f" stops, in order    {', '.join(surface['stop_names'])}")
+    print(f" URLs YOU supply    {', '.join(surface['url_envs'])}")
+    print(f" names as runnable  {', '.join(surface['connector_scripts'])}")
+    print(
+        f" browser profile    {browser['profile_name']!r}"
+        f"  (persistent={browser['persistent']}, headless={browser['headless']})"
+    )
+    print(rule)
+    print(" AT THE END, WITHOUT ASKING AGAIN: every stop's captured lenses are")
+    print(" folded into one bundle and COPIED TO YOUR CLIPBOARD, and the launcher")
+    print(" then tells you to paste it into an AI web chat. Inlined lenses:")
+    print(f"   {', '.join(DECANT_INLINE_KEYS)}")
+    print(" Those come from pages you were LOGGED IN TO. Response headers and the")
+    print(" accessibility tree carry real session and account material.")
+    print(" The per-stop CAPTURE word gates the WRITE TO DISK. It does not gate")
+    print(" this. Read the bundle before you paste it anywhere.")
+    print(rule)
+    print("")
 async def _ride_async(trail_path, dry_narrate=False):
     trail_path = Path(trail_path)
     trail = walk.load_trail(trail_path)
(nix) pipulate $ m
📝 Committing: refactor: add consent announcement function for trail walking
[main 76ea0cc8] refactor: add consent announcement function for trail walking
 1 file changed, 51 insertions(+)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'scripts/mother_cat.py'.
(nix) pipulate $ d
diff --git a/scripts/mother_cat.py b/scripts/mother_cat.py
```

```
index e70209c9..38606cc9 100644
--- a/scripts/mother_cat.py
+++ b/scripts/mother_cat.py
@@ -236,6 +236,21 @@ async def _ride_async(trail_path, dry_narrate=False):

     stops = trail["stops"]
     print(f"Riding trail '{trail['name']}' -- {len(stops)} stop(s).\n")
+    # DISCLOSURE, NOT A FENCE, AND THAT IS THE RULING RATHER THAN AN
OVERSIGHT.
+    # mck.sh already carries a RIDE confirmation, and CEREMONY IS SKIPPABLE;
+    # BARRIERS ARE NOT: a confirmation authorizes a SEQUENCE and may be skipped,
+    # a fence authorizes each WRITE and may not. A second pre-ride token would
+    # duplicate a shipped, skippable confirmation -- the sibling-.md failure in
+    # flag form. So the surface PRINTS, unconditionally, including under
+    # --dry-narrate, which is the one pass mck.sh forces on first contact.
+    # THE DECANT IS THE PART THAT ACTUALLY DESERVES A FENCE, and it does not
+    # have one: the per-stop CAPTURE token gates each write to disk, and NOTHING
+    # gates the composite leaving the machine on the clipboard while mck.sh's
+    # closing card tells the human to paste it into a cloud chat. That fence is
+    # a separate ride (new token, new refusal path, new closing card). What this
+    # car buys is disclosure BEFORE the material exists rather than after, which
+    # costs nothing and makes the later fence a formality instead of a surprise.
+    _announce_consent(trail_path)

     disclosed = False
     captured = []
(nix) pipulate $ m
📝 Committing: chore: Add disclosure message before ride confirmation
[main e60caceb] chore: Add disclosure message before ride confirmation
 1 file changed, 15 insertions(+)
(nix) pipulate $ git push
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 48 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (12/12), 3.07 KiB | 3.07 MiB/s, done.
Total 12 (delta 9), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (9/9), completed with 3 local objects.
To github.com:pipulate/pipulate.git
   6a6dbc96..e60caceb  main -> main
(nix) pipulate $
```

And this optional thing.
Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.

```
On branch main
Your branch is up to date with 'origin/main'.
```

```
nothing to commit, working tree clean
(nix) pipulate $ .venv/bin/python scripts/mother_cat.py assets/trails/botify_pageworkers.yaml
--dry-narrate
Riding trail 'botify_pageworkers' -- 3 stop(s).

================================================================
 THIS WALK: botify_pageworkers -- 3 stop(s)
================================================================
 stops, in order    optimization_detail, pageworkers_monitoring, optimization_reporting
 URLs YOU supply    PIPULATE_TRAIL_BOTIFY_MONITORING_URL,
PIPULATE_TRAIL_BOTIFY_OPTIMIZATION_URL,
PIPULATE_TRAIL_BOTIFY_REPORTING_URL
 names as runnable  scripts/connectors/botify.py
 browser profile    'botify'  (persistent=True, headless=False)
================================================================
 AT THE END, WITHOUT ASKING AGAIN: every stop's captured lenses are
 folded into one bundle and COPIED TO YOUR CLIPBOARD, and the launcher
 then tells you to paste it into an AI web chat. Inlined lenses:
   seo_md, headers, accessibility_tree_summary, links_md, diff_hierarchy_txt, optics_manifest
 Those come from pages you were LOGGED IN TO. Response headers and the
 accessibility tree carry real session and account material.
 The per-stop CAPTURE word gates the WRITE TO DISK. It does not gate
 this. Read the bundle before you paste it anywhere.
================================================================

--- Stop 1/3: optimization_detail ---
  Open the bookmarked Hello World Alert optimization detail page, confirm the authenticated
execution and success metrics are visible, then type CAPTURE.
  (dry-narrate: browser and capture skipped)

--- Stop 2/3: pageworkers_monitoring ---
  Open the bookmarked PageWorkers monitoring page, confirm the authenticated monitoring
dashboard is visible, then type CAPTURE.
  (dry-narrate: browser and capture skipped)

--- Stop 3/3: optimization_reporting ---
  Open the bookmarked Reporting tab for the Hello World Alert optimization, confirm the
authenticated report is visible, then type CAPTURE.
  (dry-narrate: browser and capture skipped)

Dry narration complete; no captures were attempted.
(nix) pipulate $
```

Wow, this is starting to create that light the world on fire feeling. I just have to bring it home.

4: Prompt:

```
Refusal receipts are in. Read them before anything else and rule on three things.
FIRST: did all five malformed trails refuse, and did each refusal name the exact field? Call out
the duplicate-key case specifically. _derive_consent_surface catches only
```

(UnicodeDecodeError, JSONDecodeError) around json.loads, but _reject_duplicate_json_keys raises a bare ValueError from inside the hook, so that refusal escapes the wrapper and arrives with a less contextualized message. Say whether that matters or whether it is cosmetic, and if it matters say so in one sentence rather than patching it here.
SECOND: did verify refuse the four-byte-appended copy, and on which branch? I predicted "Archive bytes are not canonical" rather than a CRC failure. If it tracebacked instead of refusing cleanly, that is a finding about _cmd_verify's except clause and I want it named, not fixed this turn.
THIRD: did the reseal print 2da019ab again? If it did not, content addressing is falsified and that is the whole turn.
Then ONE ride, and it is the DECANT fence.
You argued the clipboard egress is a barrier and not ceremony. Build it. A token before _decant_to_clipboard, a refusal path that leaves the artifacts on disk and prints where they are, and the matching change to mck.sh's closing card, which currently promises the clipboard unconditionally.
Answer these inside that car, not around it. Does the token reuse the CAPTURE word or need its own, and why. Does --yolo skip it, and defend your answer against CEREMONY IS SKIPPABLE; BARRIERS ARE NOT rather than restating the rule. What happens under a pipe with no tty, given that guided_browser_capture already learned the /dev/tty trick and _decant_to_clipboard has not. And what does the fence print in the world where it is broken shut, because a fence witnessed only refusing is the failure this repo has convicted five times.
Leave the signature, the ssh config, the final_url assertion, and the profile-existence check alone. All four are their own rides.
5: Deliverables: Getting closer.
Opus 5 Max Thinks: Thought for 6m 59s
Thought for 6m 59s
I'm working through the routing logic here—the actual request is in the final Prompt section, which asks me to evaluate three things from the refusal receipts: whether all five malformed trails refused, whether each one named the exact field, and to specifically call out the duplicate-key case. There's a distinction I need to make about
how _derive_consent_surface catches certain exceptions around JSON parsing,
while _reject_duplicate_json_keys raises a bare ValueError that escapes the wrapper, so that refusal gets handled differently. The canary link transformed correctly with the scheme stripped from the text. Checking the control files — both prompt_foo.py and __init__.py have their license URLs intact. Now looking at the live command receipts from the manifest, I'm seeing the derivation refusal branches with four distinct rejection reasons: invalid JSON in YAML, duplicate keys, empty list requirement, and a missing connector script field.
The duplicate key case is interesting — it surfaces as "Duplicate key: 'name'" without the wrapper context, which suggests it's escaping the normal error handling path.
The _cmd_seal function catches OSError and ValueError,
and _reject_duplicate_json_keys raises a ValueError, so I need to trace whether that's actually being caught by _derive_consent_surface's try block or if it's slipping through. The fail-closed behavior is what matters most, and that's working correctly. The message doesn't distinguish between different types of ValueError exceptions by type alone, but the text itself does clarify what went wrong. There's another consideration though: verify_walk_cartridge also uses the same duplicate key rejection logic for manifest.json, so the pattern is consistent across the codebase. The test output confirms the verify refusal works as expected with the "Archive bytes are not canonical" message, and resealing produces the same content hash, demonstrating

idempotence is preserved.
I'm noting that while this one refusal case is now witnessed, most of the other branches remain untested — the CRC check path, the testzip() path, and the twelve metadata refusals in the verifier are all still unwitnessed, so I should be transparent about having only one confirmed refusal out of roughly thirteen possible cases.
Now I'm moving into the consent disclosure output...
The patch fixed the missing attribute error and the banner now prints correctly in the actual execution path, confirming the straddle closed cleanly.
Next I'm checking whether nix is available and properly configured...
So in the operator's environment nix resolves as a function, but in the compile lane it's the actual binary at /home/mike/.nix-profile/bin/nix. Both lanes succeeded with env LD_LIBRARY_PATH= nix eval, which means the dynamic-derivations refusal is coming from the function wrapper itself, not from nix.conf or the flake configuration. The key difference is that env directly executes the binary through PATH lookup and bypasses shell functions entirely, whereas the direct assignment doesn't — so the nix() function wrapper is definitely what's triggering the error. Without access to the function body in flake.nix, I can't pinpoint exactly what it's doing, but the evidence is clear that it's the culprit. Actually, there's a subtlety with bash function invocations — variable assignments in the prefix persist in the shell after the function returns (in non-POSIX mode), and the function might do something internally that I can't determine without seeing the flake.nix. The compile lane's type nix output confirms it's hitting the binary, not a function, which aligns with the three-tier rule and explains the inconsistent behavior across turns — let me trace through the actual results to see what's happening.
Looking at the pattern: turn N-2 had the operator succeed but compile fail (with error suppressed), while turn N-1 flipped — compile succeeded but operator failed with a dynamic-derivations error. Now in this turn the operator's env form succeeded, so I'm tracking whether this is genuinely alternating or if there's something else driving the inconsistency.
The compile lane invoked nix eval on the flake in both N-2 and N-1, but with different error handling — one redirected stderr to /dev/null while the other captured it. The fact that one failed and the other succeeded despite running essentially the same command suggests either a transient issue like a lock contention or network hiccup, or maybe the dynamic-derivations evaluation needed to build something on the first attempt.
I can't definitively explain why N-2 failed in the compile lane, but since the env form works reliably in both lanes now, the root cause is moot. I should acknowledge the N-2 failure as unexplained rather than speculate too much. One hypothesis worth naming: the operator's shell session might be older than a recent nix upgrade, or there's a stale NIX_CONFIG setting that could affect evaluation.
Now I need to check the disclosure car's AFTER state — the compile receipt shows the banner, and the git diff confirms the consent announcement call is landing. Then I'll move on to examining the DECANT fence code that needs patching.
Looking at the _decant_to_clipboard function, it defers importing from prompt_foo until an actual DECANT operation runs, avoiding the overhead of loading tiktoken and pydot during module initialization or dry runs. The function reuses the existing clipboard copy logic rather than reimplementing it, which already handles SSH bridges and the pbcopy/xclip fallbacks across platforms.
At the end of _ride_async, after all captures complete successfully, the code bundles everything together, copies it to the clipboard, and prints confirmation messages directing the user to paste the bundle into any AI chatbot. The shell script's closing card then provides step-by-step instructions for the user to open an AI web chat and paste the clipboard contents.

Now I'm working through the design questions for the fence mechanism. The first question is whether the token should reuse the CAPTURE word or define its own, and the answer is its own—primarily because CAPTURE gates a write-to-disk operation that the token shouldn't trigger.
The DECANT token gates egress to an external system outside the operator's control, which is a fundamentally different consequence class. Using a distinct word prevents the failure mode where muscle memory from repeated CAPTUREs would satisfy the fence through habit rather than deliberate decision. The word itself becomes the disclosure—typing "DECANT" is a different act than typing "CAPTURE," and it names what it authorizes. This mirrors the existing pattern in the repo where mck.sh uses distinct words like INSTALL and RIDE for different authorizations.
For the --yolo flag question, the barrier cannot be skipped. The property must survive as a sentence: "nothing leaves the machine until the human types the word." If --yolo bypassed it, that sentence would break into a convention requiring call-site audits, which violates the principle that barriers are not ceremonies.
But I need to check the corollary honestly: does an unfenced decant already exist under another name? The _decant function builds markdown bundles from artifacts stored in browser_cache/, and a user can already get that bundle without the fence by running the script with @URL sigils, using cat, or writing a few lines of Python. So the capability exists—the question is whether that makes the flag redundant.
The distinction that matters is whether the flag duplicates a shipped capability reachable by the same actuator with a single keystroke. !URL and @URL require a different invocation entirely, something an expert would deliberately choose. --yolo is a flag on the launcher that a stranger gets told to run via curl-pipe, which is fundamentally different. The corollary isn't about closing the expert's door; it's about not automatically opening it for newcomers.
What --yolo actually skips is the spoken rehearsal and confirmation that happen before any material exists, whereas the decant token comes after authentication.
The real issue is temporal: a flag typed at t=0 can't authorize the disposition of material created at t=N, because at t=0 the human didn't know what would be captured. That's the substantive argument against it. The documentation for --yolo already states it doesn't skip the capture fence at any stop, and extending that same treatment to a second write barrier is consistent with the existing design. But there's a real tradeoff here — --yolo would no longer be unattended, and an unattended run would now block.
Actually, wait — the existing capture fences already block per-stop under --yolo, so it was never truly unattended to begin with. That means adding the decant barrier costs zero unattended capability, which resolves the concern.
Now I need to think through what happens when the script runs in a pipe with no tty. The ride is already invoked with a redirect from /dev/tty, so under something like curl | bash, the stdin handling should work correctly.
The decant function needs to be smarter about this though. It should try opening /dev/tty directly first, fall back to checking if stdin is a tty, and if neither works, refuse to decant — leaving artifacts on disk and printing instructions for manual decant later.
But I'm second-guessing the fail-closed approach. The constitution has that fail-open rule for interactive gates on default entry paths, where blocking would strand an unattended process. This situation is different though — refusing to decant doesn't strand anything since the artifacts are already on disk and the process exits cleanly. So fail-closed is actually the right call here.
The real insight is that refusing to decant is cheap because the expensive part (the capture) already happened and persists. A second command to retrieve the artifacts costs almost

nothing. Though that assumes there *is* a second command — I'm wondering if I should add a --decant-only mode to handle cases where the user just wants to extract what's already been captured.
Actually, the simplest approach is probably to match the existing refusal pattern: print the artifact paths for each stop and state clearly that the bundle wasn't assembled and the ride needs to be re-run. That's consistent with what's already shipped. But re-running means re-capturing, which means the human has to do that work again — so I need to think through whether that's actually acceptable.
Wait, the no-tty path is the key insight here. If there's no terminal, there's no human present to paste the bundle anyway, so refusing just costs a headless run its clipboard output, which is useless in that scenario. That settles it — the no-tty world has no human to receive the clipboard, so the refusal is the right call.
Now I need to figure out what the fence should print in each state: armed and passed (human typed DECANT), versus refusal-only.
The fence needs to announce its ARMED state unconditionally before prompting, following the same pattern as the secrets tripwire that prints its status upfront. On the pass branch, I'll print something only that branch can produce—like the byte count copied—so a broken-shut fence (always refusing) becomes distinguishable from a working one. The three refusal reasons need distinct strings so I can tell human rejection apart from missing tty or missing clipboard tool, though the clipboard tool case is tricky since it doesn't raise an exception. I need to separate the concerns here — the success message should only claim what this function actually witnessed, not what a downstream writer might or might not accomplish. The real issue is that the current code prints a clipboard success message unconditionally even though copy_to_clipboard can fail silently, so I should reword it to acknowledge the authorization step happened and let the clipboard writer's own output handle that claim.
Now I'm designing the gate function that will handle the DECANT token stream, preferring /dev/tty as the input source...
Writing the checkpoint function... Writing the input validation...
The function checks if the user confirmed the action by comparing their input against the token, declining if it doesn't match or if EOF is reached. If authorized, it sends the payload to the clipboard and returns true. I'm noticing that readline blocks indefinitely if the terminal is open but unattended, which mirrors the same blocking behavior as the capture fence.
Since the capture fence already blocked multiple times before reaching this point, a human must be present by construction, so blocking here is acceptable. Now I need to handle printing the banked artifacts — there's already an inline version in the failure path that shows the stop name, URL, and artifact count.
But the operator actually wants the directory paths where the artifacts live on disk, not just that they exist. I'm writing a helper function that extracts the unique directories from the artifact paths for each stop and prints them clearly.
Now I need to update the call site in the async ride function to use this new checkpoint function instead of the old decant flow, so it prints the artifact locations before offering to copy the payload to the clipboard.
The question is whether to return a different exit code if the user declines the clipboard copy — the ride itself succeeded (all captures were banked), so returning 0 makes sense, but then mck.sh won't know whether to promise the clipboard in its closing message.
Option (b) is the right call: rewrite mck.sh's card to use conditional language instead of asserting the clipboard is there. Since mck.sh can't actually witness what happened in the rider's authorization step, it shouldn't claim certainty — it should point to what the rider already printed

as the authoritative statement.
I'm drafting the new closing card now...
Writing the card... Finishing the help text output...
I need to update the --yolo messaging across both files to mention the DECANT gate alongside CAPTURE, since they're both gates that can't be skipped. The help text and comments need to stay in sync so users aren't blindsided by a prompt they weren't warned about. This spans a few blocks in mother_cat.py and mck.sh, but it's all one atomic change—one ride, one commit story.
The operator actually treats each fenced block as its own commit, running the patch/app/m cycle separately for each one. That's fine—the constitution's definition is just guidance, and the practical workflow is per-block. I need to emit the blocks in a deliberate order and make sure the mck.sh string blocks ship together in the same push.
Actually, I'll be explicit about it: these five blocks form one logical unit and should all be applied before pushing. That respects both the constitution and the operator's actual practice.
Now I'm writing the blocks carefully against the raw source. The first block adds the DECANT token constant and helpers to mother_cat.py, inserting before the _decant_to_clipboard function. I've verified that function signature is unique in the source, so it's a solid anchor point. Since blank lines get stripped in the payload, I'm using single-line search anchors.
I'm checking the contiguity rules against the actual mother_cat.py body — looking at how the functions are laid out and where the DECANT comment block should go. Writing the SEARCH/REPLACE block... Writing the checkpoint function... Writing gate logic... Writing clipboard authorization logic... The repo uses emoji elsewhere, so that's consistent. The payload won't contain the marker sequences, and the protocol checks in apply.py are clear. Looking at the call site where the decanted payload gets copied to clipboard and the user sees the completion message with instructions to paste into a ChatBot.
But here's the issue: the search needs to match what's actually on disk with its blank lines intact, not the compressed version without them. If a search spans across a blank line that the model can't see, the exact-match interlock fails — that's the trap I need to avoid.
So I need to use single-line anchors instead. The call to _decant_to_clipboard(payload) with 8-space indentation should be unique enough — it appears in the function definition and at the call site, but the 8-space version only shows up at the call, while a 4-space version exists elsewhere in the code. That makes it a reliable anchor point.
Actually, I could search for a multi-line block instead since the call and the two print statements are contiguous...
but I'm not certain if there's a blank line between them in the actual file. The safer approach might be to anchor just on the _decant_to_clipboard(payload) call itself and replace it with a gate check, though that wouldn't let me wrap the following prints without modifying them too. I'm going to step back and think through this more carefully. Refactoring the replacement blocks...
Looking at similar patterns in the codebase, there's consistently a blank line before conditional statements, but within an if-block the print statements stay tightly grouped. I'm fairly confident the three print lines are contiguous, so I'll anchor on _decant_to_clipboard(payload) as a separate single-line marker and work from there. The issue is that the old code unconditionally claimed the payload was copied to the clipboard, even though the underlying function silently fails and returns None—so the assertion doesn't match what actually happened. The fix checks whether the decant operation succeeded before printing the follow-up instructions, and lets the clipboard function report its own success or warning separately.
Now I'm checking block 3, which handles the --yolo flag behavior. The code prints three consecutive lines explaining that while --yolo skips the rehearsal and confirmation steps, the

capture fence at each stop still requires explicit user input before anything gets written.
I need to update this section to add two more echo statements that clarify the decant gate also can't be skipped and requires the user to type a specific word before anything leaves the machine.
Moving to block 4, which is the header comment documenting the --yolo flag behavior. I'm noticing there's a blank line inside the heredoc that displays the completion message to the user, which could affect the formatting of the output card. I'm checking whether those six lines form a contiguous block without gaps, and they do. Now I need to replace that section with a new version that clarifies the outcome based on the DECANT gate response — showing what happens if the bundle was authorized, declined, or refused, with different instructions for each path. I'm working through the logic of what happens after the user answers the DECANT gate question — whether the bundle gets copied to their clipboard or not, and what message they see depending on their choice. The script can't access their clipboard directly, so it just explains what should happen next based on their response, and reminds them that all the artifacts stayed locally in their browser cache.
Now I'm thinking about how to actually trigger this gate check. The gate only fires at the end of a real browser session with actual URLs and user tokens, which is resource-intensive. But I realize the checkpoint function is pure — it only depends on the payload and captured tokens — so I can call it directly in tests without needing the full browser setup, just like I do with the consent announcement.
For the probe design, I'm setting up a test case where the system is armed but the user refuses, and I need to run it in an environment where the TTY isn't available, which happens naturally when the prompt spawns in a detached session.
In the operator's terminal though, the TTY is available, so the probe would actually prompt and wait for input — which is actually useful because it lets the operator witness both the declined and authorized branches with real human interaction. The challenge is that a block of probes that all wait for input would be tedious, so I'm thinking about having the probe prompt once and then run a second invocation where the operator types the authorization keyword to see the authorized path, which would put a test payload on their clipboard. But that's risky — if the operator has the actual patch payload in their clipboard ready to apply, clobbering it between running probes and applying the patch would destroy the patch they're about to use. So the authorized branch probably shouldn't actually execute during a probe that runs before the patch application step. Though I'm not sure if the operator even runs probes with the patch already in hand or if they do it beforehand.
Looking at their actual workflow more carefully: they paste probes, run them, then use ahe to edit, then patch to apply from clipboard, then app to run. So the clipboard gets used right before patching — clobbering it during probes is recoverable since they'd just re-copy the patch, but it's still risky. A safer approach might be to monkeypatch the clipboard function so the authorized branch executes without actually writing to the clipboard, though that means the full write path isn't fully tested.
The real issue is that the same command behaves differently depending on context — in the operator's terminal, /dev/tty opens and prompts, but in a probe environment it should refuse. I could make the probe explicitly close stdin and detach it from any terminal using setsid to ensure it hits the REFUSED branch without blocking. ty? setsid isn't guaranteed present.
Alternative: < /dev/null doesn't stop /dev/tty from opening.
Hmm. Use a subshell with the tty detached: (exec setsid ...). On NixOS, setsid is in util-linux which is almost certainly present. But not guaranteed in the flake's commonPackages.
Better idea: have the probe pipe a NON-matching answer into /dev/tty? Can't.

Cleanest: run the probe under a temporary override where /dev/tty open fails. Can't easily.
Alternative clean approach: the probe supplies the answer by writing to the tty? No.
OK — simplest honest approach: **the operator lane probe prompts, and the operator types no.** That's one keystroke sequence that witnesses both the ARMED + DECLINED branches with a real human at a real tty — exactly the branch pair that matters, and it doesn't touch the clipboard. Meanwhile, the compile lane echoes the same line and sees the REFUSED branch because the probe's executor detaches the session. This lane disagreement is a deliberate witness pattern: one command, two lanes, two different expected outputs. For the AUTHORIZED branch, I'll give the operator a one-line command to run when ready, though it'll clobber the clipboard so they need to run it intentionally rather than paste it blindly. The payload that gets saved to the checkpoint is tiny, so if it lands on the clipboard the operator can just re-copy what they actually needed. The real concern is only if they had something important there. I'll add a warning label: "this probe clears your clipboard, run it when nothing precious is on it." I need to make it a separate, clearly-marked probe line rather than part of the compile echo—echoing it would wipe the clipboard on every compile, which is bad. But the PROBE ECHO INVARIANT says every probe must echo, so if a clipboard-clobbering probe gets echoed into adhoc.txt it would clobber on every compile. Actually, in the compile lane there's no tty, so it would hit REFUSED and never touch the clipboard anyway.
So echoing is safe, but then the behavior isn't copy-symmetric anymore—which is actually the whole point of having a lane disagreement. If I echo the AUTHORIZED probe, the compile lane runs the same REFUSED path as the other probe, making it redundant. The compile lane would produce identical output for both, which is wasteful but harmless. I think the cleanest approach is two separate gate probes, both echoed.
The first probe uses a fake capture list and the operator types no, witnessing ARMED + DECLINED + artifact homes, while the compile lane sees ARMED + REFUSED. The second probe is identical except the operator types DECANT instead, witnessing AUTHORIZED + the clipboard writer's own line, and the compile lane again sees ARMED + REFUSED—redundant but harmless. This means the operator gets prompted twice in one paste-block, which is fine; I'll just clarify what to type at each step. The clipboard will get clobbered by the second probe's payload, so I should warn about that.
Actually, the second probe does have value—the operator's question about what the fence prints when broken shut is really about distinguishability, which the ARMED line and three distinct strings answer. The AUTHORIZED witness is genuinely needed since it's a refusal-only witness. I'll include probe 2 with the clipboard warning since it's cheap.
There's a subtlety though: the _decant_to_clipboard function imports from prompt_foo, which in turn imports tiktoken, pydot, and other dependencies. The import itself is safe since prompt_foo just defines things and only runs code under if __name__ == "__main__", but I need to think through whether this causes any issues.
The module-level operations in prompt_foo are fine—it finds the repo root and loads the cartridge via importlib, both safe at import time. The OUROBOROS LOCK is inside main() so that's not a problem. The copy_to_clipboard function checks for SSH_CLIENT environment variables before writing.
I should also add a syntax check for the mck.sh file since I'm editing a heredoc region there—bash -n assets/installer/mck.sh is a cheap validation that catches heredoc breakage. It's a probe that prints the same result whether the patch is correct or broken, but if it fails, that's the only thing that catches the problem.
I'll pair the syntax check with a grep for the new DECANT strings in mck.sh to witness both the syntax validity and the actual landing of the changes. Before the patch, grep should find zero

occurrences; after, it should find around four. For mother_cat.py, the functional probe already covers the DECANT_TOKEN verification, so I'll fold everything into the mck.sh probe.
Now I'm setting up the actual probe commands. I'm calling the decant checkpoint function with a test payload to simulate the DECLINED branch when the operator types "no" at the prompt, but I need to be careful about output buffering — using tail to cap the output could hide the interactive prompt until the process completes, which breaks the user experience.
Writing probe commands... I see the issue — these two probe commands are identical, which means the deduplication logic will filter out the second one. I need to vary the command text to ensure both run, so I'm changing the echo messages to differentiate them while keeping the actual probe logic the same. Now I'm setting up a third probe variant that uses different syntax.
Writing probe commands...
This probe witnesses the wiring by grepping for the decant functions and their call sites in the script, showing the before state with the old _decant_to_clipboard definition and the "DECANT complete" message, then the after state with the new _decant_checkpoint function and its integration into the async ride flow. That's four probes total, which feels right—the functional tests cover the logic, and the grep probes verify the actual code changes and wiring without needing a full browser-based end-to-end test.
I'm also confirming that printing the prompt to stdout while reading from /dev/tty works correctly in a piped context like curl | bash, since stdout goes to the terminal anyway. And I should verify that opening /dev/tty doesn't accidentally succeed in weird detached scenarios, though the CAPTURE fences should already be blocking those cases.
The exit code behavior is settled: returning 0 even when the human declines or refuses is the right call, since a human saying no isn't a failure—treating consent as an error would teach wrappers the wrong lesson. Now I need to directly address the operator's question about whether --yolo skips the consent check, and defend that answer against the distinction between ceremony being skippable versus barriers being immovable.
I have three arguments: first, the temporal one—--yolo is specified before the artifact exists, so consent to a sequence can't cover something that didn't exist when consent was given. Second, no unattended capability is lost because --yolo already hits three CAPTURE fences, so it was never truly unattended to begin with. Third, the alternate lanes (URL variants and selenium automation) are different actuators that an expert invokes deliberately, not flags on the public launcher, and crucially none of them assemble a cross-stop composite and push it to the clipboard—so the capability --yolo would enable doesn't actually exist elsewhere. Let me verify that last point holds.
Looking at the actual code paths, mother_cat._decant is the only builder and it's only called from _ride_async. The prompt_foo function builds payloads but from its own data, not from walk captures, and it does go to the clipboard—so with @URL it would stack lenses into a payload and copy, which is similar. But that's the operator's own deliberate compile and it passes through the disclosure/denylist/secrets sanitizer, whereas the DECANT path has no sanitizer at all. That's actually a strong additional argument worth noting as a future consideration.
I need to stay focused though—the operator asked for ONE ride: the DECANT fence. I'll name the sanitizer observation but not add it as a patch.
Now I'm structuring the response: canary, straddle with the disclosure car placement, the three rulings, the DECANT fence with four answers, and the five-car train. I want to keep this dense and thorough without bloating it. Let me work through the mck.sh SEARCH blocks carefully against the actual source.
Looking at the flag documentation, I'm identifying the --where flag which discovers the workspace in read-only mode, and the --yolo flag which skips the spoken rehearsal and both

confirmations but never skips the CAPTURE fence—ceremony is skippable, barriers are not. Writing the yolo echo block... Writing shell script output... I'm checking the fence-stripping logic in apply.py to make sure my dashes won't interfere — they use backticks, not dashes, so that's fine. I'm also verifying there are no residual markers or autolink contamination patterns in my content that could cause issues. Checking that the SEARCH pattern for _decant_to_clipboard appears only once in the file, and verifying the block ordering — since block 1 introduces _decant_checkpoint which calls _decant_to_clipboard, and block 2a modifies the call site, the application order shouldn't matter given the distinct anchors. Now I need to verify the indentation in block 1's REPLACE where _decant_to_clipboard is called. I'm weighing whether to keep walk.py and walk_cartridge.py in the context or drop them to save tokens. Since the token budget is tight—this compile hit 211k with foo_files.py alone taking 64k—I should trim aggressively. I'm leaning toward dropping prompt_foo.py since _decant_to_clipboard only needs the stable copy_to_clipboard function from it, so that dependency is safe to cut.

Now for the caboose prompt: the operator wants me to leave the signature, SSH config, final_url assertion, and profile check untouched—those are their own rides. The natural next step given the distribution theme is either the fetch-by-hash launcher or the signature itself, since they're building toward walks that can be distributed via curl|bash with cryptographic sealing, and they already have the sealer, verifier, and content pieces in place.

The real payoff is making mothercat able to ride a sealed cartridge directly—taking a zipped walk, verifying it first, extracting the trail.yaml to a temp file, and executing it. That's local-only, no network or roster complications, and it closes the loop between the sealer and the rider.

Before I draft the caboose prompt, I should ask them to rule on the probe receipts first. Let me check what the operator asked for: the duplicate-key verdict in one sentence, the branch name without traceback, and confirmation on the 2da019 commit. I should keep the acknowledgment brief. Let me verify the manifest receipts match what I'm citing — yes, all four are in the codebase as marked sections. For the nix receipt specifically, I'm checking the compile output where type nix shows the binary path, and the env LD_LIBRARY_PATH= command confirms the binary works in both lanes, which points to the nix() shell function as the likely culprit for the dynamic-derivations refusal.

The real issue is that I don't have the function body in the payload since flake.nix was dropped, so I can't prove the mechanism — just name it as a suspect. But there's an important caution: using the assignment-prefix form like VAR=x func in bash leaves that variable set in the operator's shell afterward, so if the last probe set LD_LIBRARY_PATH="", it might still be active and could break python or chromium downstream.

Now I'm ready to write the final answer. I'll structure it with a fenced block of bare probe commands and keep the annotation in prose outside the block, since probes 1 and 2 are interactive and need explanation for what to type. Adding stderr redirect for cleaner output, then setting up the second probe to test the decant function with a payload that affects the clipboard.

Now I'm checking the syntax of the installer script and counting how many times the decant logic and clipboard messaging appear in it — before the patch, there should be zero decant references but one unconditional clipboard claim. Searching for those decant-related patterns across the scripts to see what changes between versions — expecting around 8 lines of matches before and a different set after the refactor. The grep alternation syntax should work fine in standard grep. I've got four probes total now, two of which are interactive, which should give me good coverage for detecting the changes.

There's a potential issue though with the interactive probes when running in the COMPILE lane: _decant_checkpoint tries to open /dev/tty, but when prompt_foo's executor spawns

with start_new_session=True, it detaches from the controlling terminal, so that open call fails with ENXIO. The stdin check would also return False since stdin is redirected to DEVNULL, meaning the stream stays None and the operation gets refused — no blocking happens, which is what we want.
The constitution already has a witness entry for this exact lane disagreement mechanism, so I can cite that. The 180-second timeout on the compile executor isn't a concern since it refuses immediately anyway.
I should guard against sys.stdin being None in the code, and I've verified that os and sys are both imported. The new functions and constants I'm adding don't conflict with anything already in mother_cat.py. One structural point: all five blocks need to land together as a single unit per the SAME-CAR LABEL RULE, even if they're spread across different sections.
For the probes themselves, probes 1, 2, and 4 read the patched file at call time through imports or grep operations, while probe 3 reads from mck.sh — so no ignition step is needed. Probe 2 does mutate clipboard state though, which is worth noting explicitly.
Actually, that clipboard write means probe 2 violates the read-only contract for probes, so it should be reclassified as the ignition in the PATCHES section instead — it's what exercises the AUTHORIZED branch. This keeps the classification clean and respects the contract exactly.
That leaves probes 1, 3, and 4 as the true read-only probes. To witness DECLINED, the operator needs to run probe 1 again after patching and respond with no — that's the scenario the IGNITION rule describes.
Show less
Done
Opus 5 Max:

0. Canary: fired again

www.canary.invalid — link text equals target minus scheme. Decisive. Co-located control holds: https://www.gnu.org/licenses/ in both prompt_foo.py and __init__.py arrived intact. Gate open, grammar unchanged. Nothing I emit below carries a bare www-token.

---

1. Two straddles closed, and the nix mystery is settled

The disclosure car: BEFORE (operator) AttributeError: module 'mother_cat' has no attribute '_announce_consent'; AFTER (compile receipt) the full banner, profile 'botify', DECANT paragraph and all. Plus your hand-run of the optional --dry-narrate put it inside the real ride path. Landed, wired, witnessed twice.
The nix actuator: this one is now *closed*, not merely narrowed.

	operator lane	compile lane
type nix	**nix is a function**	**nix is /home/mike/.nix-profile/bin/nix**
env LD_LIBRARY_PATH= nix eval	shellHook text	shellHook text

env execs the binary and **succeeds in both lanes.** That eliminates nix.conf, NIX_CONFIG, the flake, and the attribute path in one stroke — all four would refuse an env-exec'd binary identically. The nix() shell function is the sole surviving suspect for dynamic-derivations is disabled, and its body is not in this payload (flake.nix dropped), so the *mechanism* stays named-not-proven while the *culprit* is now isolated. THE THREE-TIER AMENDMENT, demonstrated on one command: alias, function, package — and command -v cannot tell you which, while type can.
Standing prescription: every nix probe uses env LD_LIBRARY_PATH= and never the assignment-prefix form. Two reasons, and the second is a live hazard: bash keeps an assignment prefix in the shell when the target is a *function*, so last turn's LD_LIBRARY_PATH="" nix eval may have left LD_LIBRARY_PATH set to empty in the shell you are still sitting in. echo "[${LD_LIBRARY_PATH}]" settles it in one word; a fresh nd clears it either way.
The N-2 compile-lane zero-byte reading remains unexplained and is now moot. I am not inventing a cause for it.

---

2. The three rulings

FIRST — five for five, and the duplicate-key escape is cosmetic

shape	refusal	names the field?
a.yaml not JSON	trail.yaml is not the JSON subset of YAML 1.2: Expecting value: line 1 column 1 (char 0)	yes, with position
b.yaml dup key	Duplicate key: 'name'	**yes, but unwrapped**
c.yaml empty stops	trail.stops must be a non-empty list	yes
d.yaml no script	stops[0].connector.script must be a non-empty string	yes, with index
e.yaml dup stop names	trail.stops contains duplicate stop names	yes

Your prediction on b.yaml was exactly right: _reject_duplicate_json_keys raises a bare ValueError from *inside* the object_pairs_hook, so it sails past the (UnicodeDecodeError, JSONDecodeError) wrapper and arrives without the trail.yaml is not the JSON subset of YAML 1.2: prefix.
One sentence, as asked: cosmetic — every call path that can raise it (seal_trail → _cmd_seal, write_walk_cartridge → verify_walk_cartridge → _cmd_verify) is wrapped by a caller that catches ValueError and prints REFUSED with the offending file named, so the only loss is a prefix and fail-closed is untouched.

SECOND — refused on the predicted branch, no traceback

tampered: REFUSED /tmp/.../walk.zip: Archive bytes are not canonical.

Exactly as predicted, and the prediction mattered: appending four bytes leaves the EOCD scan

intact, every member reads clean, testzip() passes — and the **byte-canonical reconstruction** is the only thing standing between that archive and a green. That branch is the whole reason verify_walk_cartridge rebuilds the archive instead of trusting the members, and it is now the one refusal in that function anybody has ever seen fire.
Honest scope: verify_walk_cartridge carries **thirteen** refusal sites. One is witnessed. The CRC path, the twelve metadata paths, and the manifest-mismatch path are all still REFUSAL-ONLY-WITNESS-in-reverse. That is not a debt I am paying this turn, but it is a debt.

THIRD — content addressing holds

reseal: 2da019ab3483b54f76a71fe4b44218387693fcd933bcb0e94e6836a9089a2928

Same digest, same path, second seal. Idempotence witnessed, and the compile lane's earlier walk_zips=4 was an independent second-lane observation of the same four artifacts. THE DERIVED-PATH RULE is not asserted here; it is measured.

---

3. The DECANT fence — the four answers, and one thing I found while building it

Q1: its own word — DECANT

By the time this fires the human has typed CAPTURE three times. **A fourth identical prompt is answered by muscle memory, not by decision** — and a fence satisfied by habit is not a fence. The words also name different acts: CAPTURE authorizes a write to *your* disk, DECANT authorizes a composite of authenticated material leaving *your machine*. mck.sh already runs this grammar (INSTALL, RIDE), so this is the established convention rather than a new one, and DECANT is already the GLOSSARY's noun for exactly this move.

Q2: --yolo does not skip it — three arguments, none of them a restatement

(a) Temporal. --yolo is typed at t=0, before a browser has opened. The DECANT gate disposes of material that did not exist when the flag was typed. Consent to a *sequence* cannot be consent to the disposition of an artifact the consenter had not yet seen. That is not "barriers are not skippable"; that is *this particular consent was given about a different thing*.
(b) Zero unattended capability is lost. --yolo already blocks at every CAPTURE fence, so it was never an unattended mode. Adding a fourth block costs nothing that was ever available.
(c) The bypass-under-another-name corollary does not apply, and I checked rather than assumed. That corollary rescued the --yolo case for *capture* because selenium_automation / !URL / ?URL ship the unfenced capture lane already. **No shipped path assembles a cross-stop composite and pushes it to the clipboard.** _decant is the only builder and _ride_async is its only caller. A flag would not duplicate a capability; it would create one.
And the thing I found while checking (c): the nearest sibling capability, prompt_foo's clipboard exit, passes through scrub_compile_payload, the denylist, and scan_secrets at a single chokepoint before anything is copied. **The DECANT has no sanitizer at all** — it inlines

raw headers and accessibility_tree_summary from authenticated pages straight to the clipboard. Named, not built, not this ride.

Q3: /dev/tty first, then stdin, then refuse

Under curl | bash this process's stdin is the pipe, so isatty(0) is the wrong question — the same lesson the CAPTURE prompt learned on 2026-08-01 and its doorman learned six days late. Order: open /dev/tty; if that fails, accept sys.stdin only if it is a tty; otherwise **refuse**.
Fail-closed here is not a collision with THE FAIL-OPEN THRESHOLD RULE, and the distinction is structural. That rule protects an *entry* path where blocking would strand an unattended caller. This is an *exit* path: refusing does not block, it degrades — the ride already completed, every artifact is durable on disk, and the process still exits 0. And the decisive part: **a world with no terminal to ask on is by definition a world with no human waiting to paste a clipboard.** The refusal costs that run exactly nothing.
Blocking forever on readline() is safe here for the same reason the boot menu's block is safe strictly after isatty — except stronger: three CAPTURE fences already proved a human present.

Q4: what it prints when broken shut — the ARMED line, and three distinct strings

An armed gate that passes silently and a disarmed gate both print nothing. So it announces itself unconditionally *before* asking, with the stop count and payload size, the same shape prompt_foo's secrets tripwire uses for the same reason:
🔒 DECANT gate: ARMED -- 3 stop(s), 41,207 bytes assembled, still ON THIS MACHINE ONLY.

Then exactly three terminal strings, never interchangeable: AUTHORIZED by human / DECLINED by human (read '...') / REFUSED: nowhere to ask. A fence stuck shut prints DECLINED or REFUSED forever and never AUTHORIZED — so the pass branch has its own sentence, and the probes below make all three fire in one turn.
Bonus, and it is a live ATTRIBUTED-VOICE violation in the exact lines I am replacing: the current code prints 📋 DECANT complete: capture bundle copied to your clipboard. *unconditionally*, one statement after calling a function that swallows every clipboard failure and returns None. That is a verb naming an act no code in that file performed. The new pass line claims only what this function witnessed — the human's authorization — and lets copy_to_clipboard print its own receipt.
Exit code stays 0 on decline. A human saying no is not a failure, and giving it a nonzero code teaches every wrapper to treat consent as an error. mck.sh cannot see your clipboard, so its card stops claiming it and points at the rider's line instead.

---

(1) PROBES

Probe 1 is the BEFORE for the gate: AttributeError now, and in the *compile* lane after the patch it becomes ARMED + REFUSED, because prompt_foo's executor spawns start_new_session=True and /dev/tty is unopenable there — the banked

LANE-DISAGREEMENT mechanism, used deliberately to witness the no-tty branch for free. Probe 2 guards the heredoc I am editing in the one file heredocs have already killed once, and mck_unconditional_claim goes 1 → 0 as the promise comes out. Probe 3 witnesses wiring, which the direct-call ignition cannot: DECANT complete disappears and _decant_checkpoint appears at the call site.

```
{ echo "== decant gate arm 1 =="; .venv/bin/python -c 'import sys; sys.path.insert(0,"scripts"); import mother_cat as mc; print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY", [("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' 2>&1; }
{ echo "== mck.sh syntax + decant strings =="; bash -n assets/installer/mck.sh && echo SYNTAX-OK; grep -c DECANT assets/installer/mck.sh | sed 's/^/mck_decant_lines=/'; grep -c 'the bundle is on your clipboard' assets/installer/mck.sh | sed 's/^/mck_unconditional_claim=/'; }
{ echo "== decant wiring =="; grep -n '_decant_checkpoint\|_decant_to_clipboard\|DECANT complete\|DECANT_TOKEN' scripts/mother_cat.py; }
```

(2) NEXT CONTEXT

Drop prompt_foo.py (40k) — nothing in the next ride reads it, and _decant_to_clipboard touches only its stable copy_to_clipboard.

```
! { echo "== decant gate arm 1 =="; .venv/bin/python -c 'import sys; sys.path.insert(0,"scripts"); import mother_cat as mc; print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY", [("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' 2>&1; }
! { echo "== mck.sh syntax + decant strings =="; bash -n assets/installer/mck.sh && echo SYNTAX-OK; grep -c DECANT assets/installer/mck.sh | sed 's/^/mck_decant_lines=/'; grep -c 'the bundle is on your clipboard' assets/installer/mck.sh | sed 's/^/mck_unconditional_claim=/'; }
! { echo "== decant wiring =="; grep -n '_decant_checkpoint\|_decant_to_clipboard\|DECANT complete\|DECANT_TOKEN' scripts/mother_cat.py; }
scripts/mother_cat.py
scripts/walk_cartridge.py
scripts/walk.py
assets/installer/mck.sh
assets/trails/public_walk.yaml
GLOSSARY.md
foo_files.py
```

(3) PATCHES

Five blocks, ONE car. The three mck.sh blocks describe what the mother_cat.py block gates — SAME-CAR LABEL RULE — so apply all five before git push, whatever separate messages m gives them. Every SEARCH uses either a single line or a run I am confident carries no blank, per THE CONTIGUITY COROLLARY.

Target: scripts/mother_cat.py

```
[[[SEARCH]]]
def _decant_to_clipboard(payload):
[[[DIVIDER]]]
# --- THE EGRESS BARRIER ----------------------------------------------------
```

```
# The per-stop CAPTURE token gates each WRITE TO DISK, on the operator's own
# machine. This gates EGRESS: a composite of authenticated material leaving the
# machine on the clipboard, with mck.sh then telling the human to paste it into
# a cloud chat. Different consequence class, therefore a different word.
#
# WHY NOT REUSE "CAPTURE": by the time this fires the human has typed CAPTURE
# once per stop. A fourth identical prompt is answered by MUSCLE MEMORY, not by
# decision -- and a fence satisfied by habit is not a fence. mck.sh already runs
# this grammar: INSTALL and RIDE are different words for different acts.
#
# NOT SKIPPABLE BY --yolo, and the argument is not merely that barriers are not
# skippable. (1) --yolo is typed at t=0, before a browser opens; it cannot
# consent to the disposition of material the consenter had not yet seen.
# (2) --yolo already blocks at every CAPTURE fence, so no unattended capability
# exists to lose. (3) The bypass-under-another-name corollary does NOT apply:
# _decant is the only builder of this composite and _ride_async its only caller,
# so a flag would not duplicate a shipped capability, it would create one.
DECANT_TOKEN = "DECANT"
def _print_artifact_homes(captured):
    """Name WHERE the captured material sits, not merely that it exists.

    CARGO, NOT BIBLIOGRAPHY. A refusal that says "your artifacts are safe" and
    does not say where is a refusal the human cannot act on.
    """
    print("   The captured material is on disk and untouched:")
    for stop_name, _final_url, artifacts in captured:
        homes = sorted({os.path.dirname(p) for p in artifacts.values() if p})
        if not homes:
            print(f"     {stop_name}: (no artifact paths recorded)")
        for home in homes:
            print(f"     {stop_name}: {home}")
def _decant_checkpoint(payload, captured):
    """Refuse to release the bundle until a human types DECANT. Returns bool.

    THE ARMED LINE IS UNCONDITIONAL AND IT IS THE POINT. An armed gate that
    passes silently and a DISARMED gate both print nothing, so this announces
    its own state and the payload size on every ride before asking anything --
    the same shape prompt_foo's secrets tripwire uses for the same reason.

    THREE OUTCOMES, THREE STRINGS THAT ARE NEVER INTERCHANGEABLE, so a
fence
    stuck shut is distinguishable from a fence correctly refusing:
      AUTHORIZED   the human typed the word       -> released
      DECLINED     the human typed anything else  -> withheld, paths printed
      REFUSED      nowhere to ask                 -> withheld, paths printed

    /dev/tty FIRST, the trick the CAPTURE prompt already learned: under
    `curl | bash` this process's stdin is the PIPE, so isatty(0) is the wrong
```

```
question. mck.sh already hands the ride </dev/tty; this works either way.

FAILS CLOSED ON NO TTY, and that is not a collision with THE FAIL-OPEN
THRESHOLD RULE. That rule protects an ENTRY path where blocking would
STRAND an unattended caller. This is an EXIT path: refusing does not block,
it degrades, the artifacts are already durable, and the process still exits
0 -- and a world with no terminal to ask on is by definition a world with
no human waiting to paste a clipboard. Blocking on readline() is safe here
for a stronger reason than the boot menu's isatty gate: three CAPTURE
fences have already proved a human present.
"""
payload_bytes = len(payload.encode("utf-8"))
print(
    f"\n🔒 DECANT gate: ARMED -- {len(captured)} stop(s), "
    f"{payload_bytes:,} bytes assembled, still ON THIS MACHINE ONLY."
)
stream = None
close_after = False
try:
    stream = open("/dev/tty", "r", encoding="utf-8")
    close_after = True
except OSError:
    if sys.stdin is not None and sys.stdin.isatty():
        stream = sys.stdin
if stream is None:
    print("   REFUSED: nowhere to ask -- /dev/tty is unavailable and stdin")
    print("   is not a terminal. Nothing was copied.")
    _print_artifact_homes(captured)
    print("   Re-ride from a terminal to decant.")
    return False
answer = ""
try:
    print(
        f"   Type {DECANT_TOKEN} to copy it to your clipboard "
        "(anything else keeps it here)."
    )
    print(f"   {DECANT_TOKEN}> ", end="", flush=True)
    answer = stream.readline()
except (OSError, KeyboardInterrupt):
    answer = ""
finally:
    if close_after:
        stream.close()
if answer.strip() != DECANT_TOKEN:
    print(f"\n   DECLINED by human (read {answer.strip()!r}). Nothing was copied.")
    _print_artifact_homes(captured)
    return False
print(
```

```
        f"\n   AUTHORIZED by human: handing {payload_bytes:,} bytes to the "
        "clipboard writer."
    )
    _decant_to_clipboard(payload)
    return True
def _decant_to_clipboard(payload):
[[[REPLACE]]]

Target: scripts/mother_cat.py
[[[SEARCH]]]
        _decant_to_clipboard(payload)
[[[DIVIDER]]]
        decanted = _decant_checkpoint(payload, captured)
[[[REPLACE]]]

Target: scripts/mother_cat.py
[[[SEARCH]]]
        print("\n📋 DECANT complete: capture bundle copied to your clipboard.")
        print("   Paste it into any ChatBot (Claude, ChatGPT, Gemini) and it")
        print("   will walk you through everything from here.")
[[[DIVIDER]]]
        # ATTRIBUTED-VOICE, fixed in passing because these are the exact lines
        # being rewritten: the old text asserted "copied to your clipboard"
        # UNCONDITIONALLY, one statement after calling a function that swallows
        # every clipboard failure and returns None -- a verb naming an act no
        # code in this file performed. copy_to_clipboard prints its own success
        # or warning line; this reports only what IT witnessed, which is the
        # human's authorization.
        if decanted:
            print("   Paste it into any ChatBot (Claude, ChatGPT, Gemini) and it")
            print("   will walk you through everything from here.")
[[[REPLACE]]]

Target: assets/installer/mck.sh
[[[SEARCH]]]
#   --yolo   skip the spoken rehearsal AND both confirmations. It does NOT
#            skip the CAPTURE fence at any stop, and no flag ever will.
[[[DIVIDER]]]
#   --yolo   skip the spoken rehearsal AND both confirmations. It does NOT
#            skip the CAPTURE fence at any stop, nor the DECANT gate at the
#            end, and no flag ever will. --yolo is typed BEFORE the ride, so
#            it cannot consent to the disposition of material that did not
#            exist when it was typed; and it was never unattended anyway,
#            because the CAPTURE fences already block.
[[[REPLACE]]]

Target: assets/installer/mck.sh
[[[SEARCH]]]
```

```
  echo "       NOT skipped, and not skippable by any flag: the CAPTURE fence"
  echo "       at every stop. Nothing is written until you type the word."
[[[DIVIDER]]]
  echo "       NOT skipped, and not skippable by any flag: the CAPTURE fence"
  echo "       at every stop. Nothing is written until you type the word."
  echo "       Also NOT skipped: the DECANT gate at the end. Nothing leaves"
  echo "       this machine until you type that word too."
[[[REPLACE]]]

Target: assets/installer/mck.sh
[[[SEARCH]]]
------------------------------------------------------------
   RIDE COMPLETE -- the bundle is on your clipboard
------------------------------------------------------------
 1. Open any AI web chat (Claude, ChatGPT, Gemini).
 2. Paste (Cmd+V / Ctrl+V) and send.
 3. It will walk you through everything from here.
[[[DIVIDER]]]
------------------------------------------------------------
   RIDE COMPLETE
------------------------------------------------------------
 Every stop produced a capture receipt.

 Whether the bundle LEFT this machine depends on the DECANT
 gate you just answered. This script cannot see your clipboard,
 so it does not claim to. Read the rider's own last line:

   AUTHORIZED  it is on your clipboard. Open any AI web chat
               (Claude, ChatGPT, Gemini), paste and send.
   DECLINED    nothing was copied, and the rider printed the
   REFUSED     exact directories your artifacts are sitting in.
[[[REPLACE]]]
```

Ignition — required, and it is where the other two branches get witnessed. The probes above give you the BEFORE (AttributeError) and the compile lane will give you REFUSED for free. Run these two by hand after app, in this order:

```
.venv/bin/python -c 'import sys; sys.path.insert(0,"scripts"); import mother_cat as mc;
print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY",
[("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))'
```

Type **no** → witnesses ARMED + DECLINED + the artifact-home paths.

```
.venv/bin/python -c 'import sys; sys.path.insert(0,"scripts"); import mother_cat as mc;
print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY -- safe to discard",
[("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))'
```

Type **DECANT** → witnesses AUTHORIZED plus copy_to_clipboard's own receipt. **This one clobbers your clipboard** with a throwaway string, so run it when nothing precious is on it — after git push, not before patch. It is ignition rather than a probe precisely because it writes OS

state.
All three branches then have receipts in one turn, which is the standard this repo has convicted itself against five times for failing.

(4) PROMPT

Gate receipts are in. Read them before anything else and rule on the branch coverage: did all three of ARMED+DECLINED, ARMED+AUTHORIZED, and ARMED+REFUSED fire, and did the compile-lane echo land REFUSED as predicted by the start_new_session detachment. If any branch is missing say so plainly rather than inferring it from the other two, and say whether grep-wiring plus a direct call is weaker evidence than one real public_walk ride would be.

Then ONE ride: teach the rider to accept a SEALED cartridge.

mothercat takes a path to a mutable trail YAML. walk_cartridge can seal, verify, and show. Nothing connects them. Make mother_cat.ride accept either a .yaml trail path or a walk.zip: verify first, refuse loudly on any of the thirteen refusal branches, extract trail.yaml to a temp file, and ride that. No network, no fetch, no signature, no roster.

Answer these inside that car. Does the extracted trail get validated by walk.load_trail again after verification, or is the seal sufficient, and defend it as one-authority-or-two rather than as belt-and-braces. Does the consent surface get printed from the manifest or re-derived from the extracted bytes, given that the whole point of the manifest is that those two cannot disagree. What happens to the temp file, and does the ride ever leave an unsealed copy of a client trail lying around. And name the discrimination: what does the rider print when handed a zip that is not a walk cartridge at all.

This is also the SECOND CANDIDATE for the trail-source selector, which has been single-candidate since the day mck.sh grew a search path. Say what that buys and what it now makes testable that was not.

Leave the signature, the ssh config, the final_url assertion, the profile-existence check, and the missing DECANT sanitizer alone. All five are their own rides.

(5) EXTERNAL DELIVERABLES

None this turn.
MikeLev.in:
1: Probe:
(nix) pipulate $ g

Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.
On branch main
Your branch is up to date with 'origin/main'.

```
nothing to commit, working tree clean
(nix) pipulate $ { echo "== decant gate arm 1 =="; .venv/bin/python -c 'import sys;
sys.path.insert(0,"scripts"); import mother_cat as mc; print("released:",
mc._decant_checkpoint("PROBE PAYLOAD ONLY",
[("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' 2>&1; }
{ echo "== mck.sh syntax + decant strings =="; bash -n assets/installer/mck.sh && echo
SYNTAX-OK; grep -c DECANT assets/installer/mck.sh | sed 's/^/mck_decant_lines=/'; grep -c
'the bundle is on your clipboard' assets/installer/mck.sh | sed 's/^/mck_unconditional_claim=/'; }
{ echo "== decant wiring =="; grep -n '_decant_checkpoint\|_decant_to_clipboard\|DECANT
complete\|DECANT_TOKEN' scripts/mother_cat.py; }
== decant gate arm 1 ==
Traceback (most recent call last):
  File "<string>", line 1, in <module>
AttributeError: module 'mother_cat' has no attribute '_decant_checkpoint'
== mck.sh syntax + decant strings ==
SYNTAX-OK
mck_decant_lines=0
mck_unconditional_claim=1
== decant wiring ==
149:def _decant_to_clipboard(payload):
319:        _decant_to_clipboard(payload)
320:        print("\n📋 DECANT complete: capture bundle copied to your clipboard.")
(nix) pipulate $
```

2: Context:

```
# adhoc.txt    _   _   _ to set context____ _   _  ___  ____  _   <F5> Simpson Couch Gag Here
(explain anything to the audience you feel needs it explained)
#     / \   __| | | | | | ___   ___    / ___| | | |/ _ \|  _ \| |
# ahe/ _ \ / _` | | |_| |/ _ \ / __| | |   | |_| | | | | |_) | |  I'm optimistic.
# ahc ___ \ (_| | |  _  | (_) | (__  | |___|  _  | |_| |  __/|_|
#  /_/   \_\__,_| |_| |_|\___/ \___|  \_____|_| |_|\___/|_|   (_)
# Ad Hoc CHOP: The Not-Managed-by-Git Safe-for-Client-Data place

# THE ROLLING PIN BOOK ORE SPINE / FODDER FOR THE BOOK
# ! python scripts/articles/lsa.py -t 1 --reverse --fmt dated-slugs  # <-- The "Rolling Pin" that
gives the 40K foot book-spine view of book-ore.
GLOSSARY.md                # <-- I think this glossary goes well with the book-ore spine to do
world building.
# scripts/articles/lsa.py     # <-- Useful for refining commands like `posts`, critical to Second
Brain concept.

# THE QUIRKY AMIGA-LOVING HUMAN
# ~/repos/nixos/autognome.py  # <-- Letting the AIs really understand my environment (The
Brave Little Tailor punches above Their Weight Class proving the dunning-kruger effect the
gate-keeper's (lower-case) lament.)
# init.lua                  # <-- Daily driver hot-keys that overlap with aliases in flake.nix

# THE HAND-CRANKED AGENTIC FRAMEWORK
```

```
# prompt_foo.py              # <-- Prompt Fu compiler, makes the very README for AGENTS-like payload you're reading right now, but it needs to be more like that
# foo_files.py               # <-- This is the router, evolving book outline and the things you pin-up to produced the recursive self-improvement loops

# MAIN ACTUATORS, IaC & NEGATIVE SPACE
# apply.py                   # <-- How can "Web UI" ChatBots edit your code? With this Aider-inspired Player Piano patch applier.
.gitattributes               # <-- Model: understand that `nbstripout` and `jupytext` are both in play. Just talk the human through .ipynb patches.
.gitignore                   # <-- Creates "negative space" for sub-rep's to share parent environment and "snap" proprietary secret features into place.
# flake.nix                  # <-- Solves world's WRITE ONCE RUN ANYWHERE problem like Java never could. Also resolves the bootstrap paradox.
requirements.in              # <-- All known dependencies and (necessary) version pinning. WORA gotcha's exposed.
__init__.py                  # <-- Master versioning
pyproject.toml               # <-- The PyPI Packaging details
# cli.py                     # <-- Catch-all actuator for PyPI envs, Python anchoring, MCP tool-call (plus alternatives) and **kwargs like wrapping for CLI

# scripts/xp.py              # <-- Transforms host OS copy-paste buffer player-piano music into context-payload.
# scripts/ai.py              # <-- How I constantly use local AI to write git commit messages with `m` alias.

# CONTEXT PORTABILITY SYSTEM
# scripts/foo_cartridge.py    # Needs description
# scripts/foo_replay.py       # Needs description

# # FREQUENTLY USEFUL TO HAVE IN CONTEXT
# release.py                 # <-- How everything ends up where it does (GitHub, PyPI, etc.)
# scripts/weblogin.py        # <-- Lets the user "warm up" the cache for their web logins at their leisure on a profile that persists.
# scripts/crawl.py           # <-- Feel free to ask for something to be crawled and included in the next turn.
# imports/voice_synthesis.py  # <-- The wand can talk to you
# scripts/release/version_sync.py  # <-- Needs to be wrapped into release.py and eliminated, I think.

#                     --- Under this line is were you paste what the AI gives you ---
#                     --- We call it context but it's really just the right-hand  ---
#                     --- blast-radius of the "probes" to make this all science.  ---

# --- END `adhoc.txt` TEMPLATE ---

# server.py
# scripts/mcp_menu.py
```

```
#
# scripts/connectors/README.md
# scripts/connectors/gmail.py
# scripts/connectors/confluence.py
# scripts/connectors/jira.py
# scripts/connectors/slack.py
# scripts/connectors/botify.py
# scripts/connectors/gsc.py
# scripts/connectors/sheets.py
# scripts/connectors/wallet.py
# scripts/connectors/mcp.py
#
# tools/scraper_tools.py
# tools/__init__.py
# tools/dom_tools.py
# tools/llm_optics.py
# scripts/walk.py
# assets/trails/first_context.yaml
# scripts/weblogin.py
#
# ! test -f assets/installer/fdr.sh && echo EXISTS || echo ABSENT
# ! bash -n assets/installer/fdr.sh && echo SYNTAX-OK
# ! grep -c '/dev/tty' assets/installer/fdr.sh
# ! ls browser_cache/looking_at
# assets/installer/fdr.sh
# assets/installer/replay.sh
# assets/trails/public_walk.yaml
# scripts/mother_cat.py

# `d`, `Shift`+`G`! I have to remember that.

! { echo "== decant gate arm 1 =="; .venv/bin/python -c 'import sys; sys.path.insert(0,"scripts");
import mother_cat as mc; print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY",
[("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' 2>&1; }
! { echo "== mck.sh syntax + decant strings =="; bash -n assets/installer/mck.sh && echo
SYNTAX-OK; grep -c DECANT assets/installer/mck.sh | sed 's/^/mck_decant_lines=/'; grep -c
'the bundle is on your clipboard' assets/installer/mck.sh | sed 's/^/mck_unconditional_claim=/'; }
! { echo "== decant wiring =="; grep -n '_decant_checkpoint\|_decant_to_clipboard\|DECANT
complete\|DECANT_TOKEN' scripts/mother_cat.py; }
scripts/mother_cat.py
scripts/walk_cartridge.py
scripts/walk.py
assets/installer/mck.sh
assets/trails/public_walk.yaml
GLOSSARY.md
foo_files.py
```

3: Patches:

(nix) pipulate $ ahe
(nix) pipulate $ g

Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'scripts/mother_cat.py'.
(nix) pipulate $ d
diff --git a/scripts/mother_cat.py b/scripts/mother_cat.py
index 38606cc9..4991bd79 100644
--- a/scripts/mother_cat.py
+++ b/scripts/mother_cat.py
@@ -146,6 +146,107 @@ def _decant(captured):
    return "\n".join(parts)

+# --- THE EGRESS BARRIER --
+# The per-stop CAPTURE token gates each WRITE TO DISK, on the operator's own
+# machine. This gates EGRESS: a composite of authenticated material leaving the
+# machine on the clipboard, with mck.sh then telling the human to paste it into
+# a cloud chat. Different consequence class, therefore a different word.
+#
+# WHY NOT REUSE "CAPTURE": by the time this fires the human has typed CAPTURE
+# once per stop. A fourth identical prompt is answered by MUSCLE MEMORY, not by
+# decision -- and a fence satisfied by habit is not a fence. mck.sh already runs
+# this grammar: INSTALL and RIDE are different words for different acts.
+#
+# NOT SKIPPABLE BY --yolo, and the argument is not merely that barriers are not
+# skippable. (1) --yolo is typed at t=0, before a browser opens; it cannot
+# consent to the disposition of material the consenter had not yet seen.
+# (2) --yolo already blocks at every CAPTURE fence, so no unattended capability
+# exists to lose. (3) The bypass-under-another-name corollary does NOT apply:
+# _decant is the only builder of this composite and _ride_async its only caller,
+# so a flag would not duplicate a shipped capability, it would create one.
+DECANT_TOKEN = "DECANT"
+def _print_artifact_homes(captured):
+   """Name WHERE the captured material sits, not merely that it exists.
+
+   CARGO, NOT BIBLIOGRAPHY. A refusal that says "your artifacts are safe" and
+   does not say where is a refusal the human cannot act on.
+   """
+   print("   The captured material is on disk and untouched:")
+   for stop_name, _final_url, artifacts in captured:

```
+        homes = sorted({os.path.dirname(p) for p in artifacts.values() if p})
+        if not homes:
+            print(f"     {stop_name}: (no artifact paths recorded)")
+        for home in homes:
+            print(f"     {stop_name}: {home}")
+def _decant_checkpoint(payload, captured):
+    """Refuse to release the bundle until a human types DECANT. Returns bool.
+
+    THE ARMED LINE IS UNCONDITIONAL AND IT IS THE POINT. An armed gate that
+    passes silently and a DISARMED gate both print nothing, so this announces
+    its own state and the payload size on every ride before asking anything --
+    the same shape prompt_foo's secrets tripwire uses for the same reason.
+
+    THREE OUTCOMES, THREE STRINGS THAT ARE NEVER INTERCHANGEABLE, so a
fence
+    stuck shut is distinguishable from a fence correctly refusing:
+      AUTHORIZED   the human typed the word       -> released
+      DECLINED     the human typed anything else  -> withheld, paths printed
+      REFUSED      nowhere to ask                 -> withheld, paths printed
+
+    /dev/tty FIRST, the trick the CAPTURE prompt already learned: under
+    `curl | bash` this process's stdin is the PIPE, so isatty(0) is the wrong
+    question. mck.sh already hands the ride </dev/tty; this works either way.
+
+    FAILS CLOSED ON NO TTY, and that is not a collision with THE FAIL-OPEN
+    THRESHOLD RULE. That rule protects an ENTRY path where blocking would
+    STRAND an unattended caller. This is an EXIT path: refusing does not block,
+    it degrades, the artifacts are already durable, and the process still exits
+    0 -- and a world with no terminal to ask on is by definition a world with
+    no human waiting to paste a clipboard. Blocking on readline() is safe here
+    for a stronger reason than the boot menu's isatty gate: three CAPTURE
+    fences have already proved a human present.
+    """
+    payload_bytes = len(payload.encode("utf-8"))
+    print(
+        f"\n🔒 DECANT gate: ARMED -- {len(captured)} stop(s), "
+        f"{payload_bytes:,} bytes assembled, still ON THIS MACHINE ONLY."
+    )
+    stream = None
+    close_after = False
+    try:
+        stream = open("/dev/tty", "r", encoding="utf-8")
+        close_after = True
+    except OSError:
+        if sys.stdin is not None and sys.stdin.isatty():
+            stream = sys.stdin
+    if stream is None:
+        print("   REFUSED: nowhere to ask -- /dev/tty is unavailable and stdin")
```

```
+        print("   is not a terminal. Nothing was copied.")
+        _print_artifact_homes(captured)
+        print("   Re-ride from a terminal to decant.")
+        return False
+    answer = ""
+    try:
+        print(
+            f"   Type {DECANT_TOKEN} to copy it to your clipboard "
+            "(anything else keeps it here)."
+        )
+        print(f"   {DECANT_TOKEN}> ", end="", flush=True)
+        answer = stream.readline()
+    except (OSError, KeyboardInterrupt):
+        answer = ""
+    finally:
+        if close_after:
+            stream.close()
+    if answer.strip() != DECANT_TOKEN:
+        print(f"\n   DECLINED by human (read {answer.strip()!r}). Nothing was copied.")
+        _print_artifact_homes(captured)
+        return False
+    print(
+        f"\n   AUTHORIZED by human: handing {payload_bytes:,} bytes to the "
+        "clipboard writer."
+    )
+    _decant_to_clipboard(payload)
+    return True
 def _decant_to_clipboard(payload):
     """Copy the bundle to the clipboard, reusing prompt_foo's cross-platform path.

(nix) pipulate $ m
📝 Committing: chore: Refactor decant function with enhanced logging and gate control
[main 903a8e15] chore: Refactor decant function with enhanced logging and gate control
 1 file changed, 101 insertions(+)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'scripts/mother_cat.py'.
(nix) pipulate $ d
diff --git a/scripts/mother_cat.py b/scripts/mother_cat.py
index 4991bd79..06a75706 100644
--- a/scripts/mother_cat.py
+++ b/scripts/mother_cat.py
@@ -417,7 +417,7 @@ async def _ride_async(trail_path, dry_narrate=False):
     print("\nRide complete. Every stop produced a capture receipt.")
     if captured:
         payload = _decant(captured)
-        _decant_to_clipboard(payload)
+        decanted = _decant_checkpoint(payload, captured)
```

```
        print("\n📋 DECANT complete: capture bundle copied to your clipboard.")
        print("   Paste it into any ChatBot (Claude, ChatGPT, Gemini) and it")
        print("   will walk you through everything from here.")
(nix) pipulate $ m
📝 Committing: chore: Update decant function to use _decant_checkpoint
[main 8cbe7ca8] chore: Update decant function to use _decant_checkpoint
 1 file changed, 1 insertion(+), 1 deletion(-)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'scripts/mother_cat.py'.
(nix) pipulate $ d
diff --git a/scripts/mother_cat.py b/scripts/mother_cat.py
index 06a75706..03a54abb 100644
--- a/scripts/mother_cat.py
+++ b/scripts/mother_cat.py
@@ -418,9 +418,16 @@ async def _ride_async(trail_path, dry_narrate=False):
     if captured:
         payload = _decant(captured)
         decanted = _decant_checkpoint(payload, captured)
-        print("\n📋 DECANT complete: capture bundle copied to your clipboard.")
-        print("   Paste it into any ChatBot (Claude, ChatGPT, Gemini) and it")
-        print("   will walk you through everything from here.")
+        # ATTRIBUTED-VOICE, fixed in passing because these are the exact lines
+        # being rewritten: the old text asserted "copied to your clipboard"
+        # UNCONDITIONALLY, one statement after calling a function that swallows
+        # every clipboard failure and returns None -- a verb naming an act no
+        # code in this file performed. copy_to_clipboard prints its own success
+        # or warning line; this reports only what IT witnessed, which is the
+        # human's authorization.
+        if decanted:
+            print("   Paste it into any ChatBot (Claude, ChatGPT, Gemini) and it")
+            print("   will walk you through everything from here.")
     return 0


(nix) pipulate $ m
📝 Committing: chore: Refine clipboard message in mother_cat.py
[main 8847c5cf] chore: Refine clipboard message in mother_cat.py
 1 file changed, 10 insertions(+), 3 deletions(-)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'assets/installer/mck.sh'.
(nix) pipulate $ d
diff --git a/assets/installer/mck.sh b/assets/installer/mck.sh
index 4e4bace8..3e7a3565 100644
--- a/assets/installer/mck.sh
+++ b/assets/installer/mck.sh
@@ -57,7 +57,11 @@
```

```
 #           probe that makes marker discovery witnessable without needing a
 #           fresh machine.
 #   --yolo   skip the spoken rehearsal AND both confirmations. It does NOT
-#           skip the CAPTURE fence at any stop, and no flag ever will.
+#           skip the CAPTURE fence at any stop, nor the DECANT gate at the
+#           end, and no flag ever will. --yolo is typed BEFORE the ride, so
+#           it cannot consent to the disposition of material that did not
+#           exist when it was typed; and it was never unattended anyway,
+#           because the CAPTURE fences already block.
 #           CEREMONY IS SKIPPABLE; BARRIERS ARE NOT: a confirmation
 #           authorizes a SEQUENCE, a fence authorizes each WRITE, and the
 #           unfenced capture lane already exists under other names.
(nix) pipulate $ m
📝 Committing: chore: Clarify mck.sh documentation --yolo flag behavior
[main aee3976f] chore: Clarify mck.sh documentation --yolo flag behavior
 1 file changed, 5 insertions(+), 1 deletion(-)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'assets/installer/mck.sh'.
(nix) pipulate $ d
diff --git a/assets/installer/mck.sh b/assets/installer/mck.sh
index 3e7a3565..a0e5e655 100644
--- a/assets/installer/mck.sh
+++ b/assets/installer/mck.sh
@@ -442,6 +442,8 @@ if [ "$YOLO" -eq 1 ]; then
   echo "--yolo: skipping the spoken rehearsal and the RIDE confirmation."
   echo "        NOT skipped, and not skippable by any flag: the CAPTURE fence"
   echo "        at every stop. Nothing is written until you type the word."
+  echo "        Also NOT skipped: the DECANT gate at the end. Nothing leaves"
+  echo "        this machine until you type that word too."
 fi
 if [ "$YOLO" -eq 0 ]; then
 cat <<CARD
(nix) pipulate $ m
📝 Committing: chore: Add clarifying comments to mck.sh script
[main aa3f0781] chore: Add clarifying comments to mck.sh script
 1 file changed, 2 insertions(+)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'assets/installer/mck.sh'.
(nix) pipulate $ d
diff --git a/assets/installer/mck.sh b/assets/installer/mck.sh
index a0e5e655..2457072a 100644
--- a/assets/installer/mck.sh
+++ b/assets/installer/mck.sh
@@ -489,11 +489,18 @@ run_wrapped "$PY" scripts/mother_cat.py "$TRAIL_PATH" </dev/tty
|| RIDE_RC=$?
 if [ "$RIDE_RC" -eq 0 ]; then
```

```
   cat <<'CARD'
 ------------------------------------------------------------
-   RIDE COMPLETE -- the bundle is on your clipboard
+   RIDE COMPLETE
 ------------------------------------------------------------
- 1. Open any AI web chat (Claude, ChatGPT, Gemini).
- 2. Paste (Cmd+V / Ctrl+V) and send.
- 3. It will walk you through everything from here.
+ Every stop produced a capture receipt.
+
+ Whether the bundle LEFT this machine depends on the DECANT
+ gate you just answered. This script cannot see your clipboard,
+ so it does not claim to. Read the rider's own last line:
+
+   AUTHORIZED  it is on your clipboard. Open any AI web chat
+               (Claude, ChatGPT, Gemini), paste and send.
+   DECLINED    nothing was copied, and the rider printed the
+   REFUSED     exact directories your artifacts are sitting in.
  The raw artifacts stayed on your machine, under
  browser_cache/. Nothing was uploaded by this script.
 ------------------------------------------------------------
(nix) pipulate $ m
📝 Committing: chore: Refactor installer output for clipboard handling consistency
[main 40f9930f] chore: Refactor installer output for clipboard handling consistency
 1 file changed, 11 insertions(+), 4 deletions(-)
(nix) pipulate $ git push
Enumerating objects: 33, done.
Counting objects: 100% (33/33), done.
Delta compression using up to 48 threads
Compressing objects: 100% (27/27), done.
Writing objects: 100% (27/27), 5.61 KiB | 2.80 MiB/s, done.
Total 27 (delta 18), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (18/18), completed with 5 local objects.
To github.com:pipulate/pipulate.git
   e60caceb..40f9930f  main -> main
(nix) pipulate $

Okay, again real ignitions and tests.
Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing.
Science.
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(nix) pipulate $ .venv/bin/python -c 'import sys; sys.path.insert(0,"scripts"); import mother_cat as
mc; print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY",
[("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))'
```

🔒 DECANT gate: ARMED -- 1 stop(s), 18 bytes assembled, still ON THIS MACHINE ONLY.
Type DECANT to copy it to your clipboard (anything else keeps it here).
DECANT> no

DECLINED by human (read 'no'). Nothing was copied.
The captured material is on disk and untouched:
probe_stop: /tmp/probe_home
released: False
(nix) pipulate $ .venv/bin/python -c 'import sys; sys.path.insert(0,"scripts"); import mother_cat as mc; print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY -- safe to discard", [("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))'

🔒 DECANT gate: ARMED -- 1 stop(s), 37 bytes assembled, still ON THIS MACHINE ONLY.
Type DECANT to copy it to your clipboard (anything else keeps it here).
DECANT> DECANT

AUTHORIZED by human: handing 37 bytes to the clipboard writer.
Markdown output copied to clipboard
released: True
(nix) pipulate $

Gee, I hope I did that right.
4: Prompt:
Gate receipts are in. Read them before anything else and rule on the branch coverage: did all three of ARMED+DECLINED, ARMED+AUTHORIZED, and ARMED+REFUSED fire, and did the compile-lane echo land REFUSED as predicted by the start_new_session detachment. If any branch is missing say so plainly rather than inferring it from the other two, and say whether grep-wiring plus a direct call is weaker evidence than one real public_walk ride would be.

Then ONE ride: teach the rider to accept a SEALED cartridge.

mothercat takes a path to a mutable trail YAML. walk_cartridge can seal, verify, and show. Nothing connects them. Make mother_cat.ride accept either a .yaml trail path or a walk.zip: verify first, refuse loudly on any of the thirteen refusal branches, extract trail.yaml to a temp file, and ride that. No network, no fetch, no signature, no roster.

Answer these inside that car. Does the extracted trail get validated by walk.load_trail again after verification, or is the seal sufficient, and defend it as one-authority-or-two rather than as belt-and-braces. Does the consent surface get printed from the manifest or re-derived from the extracted bytes, given that the whole point of the manifest is that those two cannot disagree. What happens to the temp file, and does the ride ever leave an unsealed copy of a client trail lying around. And name the discrimination: what does the rider print when handed a zip that is not a walk cartridge at all.

This is also the SECOND CANDIDATE for the trail-source selector, which has been single-candidate since the day mck.sh grew a search path. Say what that buys and what it now makes testable that was not.

Leave the signature, the ssh config, the final_url assertion, the profile-existence check, and the missing DECANT sanitizer alone. All five are their own rides.

5: Deliverables: How are we doing?
Opus 5 Max Thinks: Orchestrating DECANT fence implementation and refusal branch testing
Orchestrating DECANT fence implementation and refusal branch testing
I need to check what actually ran during this compile by looking at the manifest's live command receipts, since those are the authoritative record. The current request wants me to rule on three items from the refusal receipts, then handle one specific ride involving the decant fence.
Looking at what executed, I can see there are several shell commands that ran: one probing the decant checkpoint with a test payload, another validating the mck.sh syntax and counting decant-related strings in it, and a third grepping through the mother_cat.py script to find all the decant wiring points. The key finding here is that the decant fence has already been constructed — the codebase already contains the checkpoint function, the decant token constant, the artifact home printer, and the three-string authorization design that the git diff telemetry confirms.
This is the stale overlay case that's documented in the constitution: when pasted context and what the compiler actually read can diverge within a single turn, the Manifest's live command receipts are the only source of truth for what actually ran.
But wait—the situation here is inverted. The Prompt section's caboose asks me to build the DECANT fence, yet the Manifest receipts and codebase show it's already built and already been executed. So the caboose is stale relative to the actual code.
Let me look at what's actually in the compile: the full DECANT fence implementation is already in the codebase with all the pieces—the token, checkpoints, clipboard logic, the three-string authorization design, the non-skippability flag, and the /dev/tty trick. The closing card in mck.sh has been rewritten, the git diff shows that as the most recent commit, and live receipts are probing the DECANT gate. So between the pasted transcript and this compile, the DECANT fence ride already happened, and the operator's adhoc.txt now carries evidence of it.
But looking at the adhoc.txt from the context block, it still shows the older refusal branches and consent disclosure probes—not the DECANT probes that appear in the live command receipts for this compile. That means adhoc.txt has been rewritten since the article was pasted, so the context shown is stale by at least one more ride.
The right move here is to follow the receipts, not the stale overlay. The DECANT fence is already built, so I shouldn't pretend to construct it or risk duplicating it with a patch that would fail exact-match searches. Instead, I need to report the canary state, flag the stale-overlay finding, and rule on which probes from the receipts are actually visible in the pasted transcript.
Looking at the refusal branches in the "1: Probe" block, I can see the derivation rejections: the YAML parsing error on a.yaml, the duplicate key in b.yaml, the empty list constraint violation in c.yaml, the empty string requirement for the script in d.yaml, and the duplicate stop names in e.yaml. These are the historical evidence I should be ruling on.
Now I'm seeing the additional refusal cases from before the patch was applied — the tampered archive with non-canonical bytes, the consent disclosure error where the mother_cat module is missing the announcement function, and the nix actuator discrimination showing how the environment bypasses the function to execute the binary directly. These are the BEFORE readings, and I can compare them against the AFTER state shown in the ignition transcript to make my ruling on all three categories.
The duplicate-key case is interesting because it's raising a ValueError from inside the JSON object pairs hook rather than being caught as a JSONDecodeError, so the error propagates without the file-position context or the wrapper message that the other cases get.

But looking at it practically, the message still identifies the offending key and refuses to proceed, and _cmd_seal adds the file path when it prints the error, so the user knows which file was rejected. The only real loss is that they don't know whether the rejection happened at the parse layer or the shape layer.
What concerns me more is whether this bare ValueError escaping could cause a real problem. _derive_consent_surface gets called by _build_manifest, which gets called by verify_walk_cartridge, and in the verifier that call to _build_manifest with the trail.yaml bytes happens outside any try block. The asymmetry itself is informative—the duplicate-key check fires inside JSON parsing rather than after, which is a distinguishing feature worth noting, though I'm still calling it cosmetic overall. The second refusal about archive bytes not being canonical played out exactly as expected: it's a clean rejection with no traceback, caught by the except clause, and it happened because zipfile tolerates trailing junk by scanning backward for the End of Central Directory, so the members and CRCs all validate fine until the final byte-for-byte comparison against the original file.
The critical detail here is that this canonical reconstruction check is the strongest verification branch and it's the one that actually fired—the one that gives "byte-reproducible" real meaning rather than just an aspiration. But this also means the CRC path and the BadZipFile exception from reading the archive remain untested, and I need to verify whether BadZipFile would even be caught by the current exception handler since it inherits directly from Exception rather than OSError or ValueError.
Actually, looking at the execution order: archive.read(info) gets called before testzip() in the member loop, and read() raises BadZipFile on CRC failure. So a corrupted byte mid-file would cause BadZipFile to propagate uncaught out of _cmd_verify before the code ever reaches the testzip() line that was meant to handle this case. That's a real gap—the CRC verification branch at testzip() is unreachable for the exact scenario it was designed for, because the earlier read() call fails first with an exception that isn't being caught.
I've confirmed BadZipFile is just a plain Exception subclass in CPython's zipfile module, so it definitely won't be caught by the except (OSError, ValueError) tuple. That's the finding: an unreachable guard combined with an uncaught exception type.
Now checking the third issue—the reseal operation produced an identical hash.
The DECANT fence is already built. Rather than trying to fix or re-implement it, the right approach is to verify that the shipped code actually does what its own comments claim it does, then identify what's broken. I need to carefully audit the _decant_checkpoint function and related code to see if the implementation matches the design questions the operator asked.
Writing checkpoint logic... Writing clipboard handler... The REFUSED message prints identically whether there's no terminal or the gate is stuck, so it doesn't discriminate between those two failure modes. But the ARMED line always prints unconditionally with the payload size and stop count, so even a stuck fence announces itself—and the four status strings (ARMED, AUTHORIZED, DECLINED, REFUSED) remain mutually exclusive as the design intends.
The real gap is that DECLINED and AUTHORIZED branches leave no trace if they're never executed, which is where the ARMED line comes in as the witness that something happened.
The asymmetry is real: guided_browser_capture accepts injectable streams for testing, but _decant_checkpoint doesn't, making one testable and the other not. I'm exploring whether I could manufacture a controlling terminal in the compile lane using os.openpty() combined with os.setsid() and the appropriate ioctl call to make it the controlling terminal, though that's getting into deeper territory.
Actually, the cleanest approach might be using the script utility to allocate a pty and pipe input through it — something like feeding "NOPE\n" through script in quiet mode to test the

DECLINED/AUTHORIZED paths without all that machinery.
The issue is that AUTHORIZED calls into clipboard functions, which would mutate the operator's actual clipboard mid-compile while they're trying to paste something else. Even worse, the AUTHORIZED witness shouldn't be in PROBES at all since probes are supposed to be read-only observations.
The real solution is to run it as a hand-executed ignition where the operator explicitly knows their clipboard will be overwritten. Alternatively, I could witness that _decant_checkpoint returns True instead of touching the clipboard, letting _decant_to_clipboard fail gracefully—but that's risky since the compile lane has DISPLAY set on the operator's desktop, so xclip might actually succeed when we don't want it to. I need to think through what the actual constraint is here.
Looking at the debt list: the DECLINED and AUTHORIZED branches are both unwitnessed, _decant_checkpoint isn't injectable
like guided_browser_capture is, BadZipFile isn't caught in _cmd_verify, and the testzip() CRC branch is unreachable. The cleanest probe is to trigger DECLINED via a wrong word in the script—that path skips the clipboard write entirely, witnesses the second branch, and carries zero risk.
For AUTHORIZED, I need to decide between a real end-to-end run (which would clobber the clipboard but honestly witness the whole flow) or trying to monkeypatch _decant_to_clipboard in the probe itself to make it safe. The monkeypatch approach might work cleanly here. Writing the test probe... I'm wondering if script is even available in the nix shell, so I should add a guard to check for it. Actually, Python's pty module might be cleaner — using pty.fork() to create a child process with a proper controlling terminal so /dev/tty can open, which avoids the external dependency entirely. I need to write a small probe script that handles this.
Let me step back and think about the overall structure. The operator wants 3-4 bounded probes, so I should pick a few key ones: one to witness the DECLINED arm via pty, another to witness the AUTHORIZED decision path with a stubbed clipboard writer to avoid actual egress, and maybe a third probe.
For that third probe, I could demonstrate the BadZipFile finding by corrupting a byte in the middle of the archive and showing that verification fails with a traceback rather than silently refusing. I'd use dd to flip a byte at a specific offset inside the stored data, then run verify and capture the BadZipFile exception. The tricky part is figuring out the exact byte offset and which archive find returns, since filesystem ordering is unpredictable.
Writing probe 3... I need to capture the exit code properly by storing it before the pipeline, then use that stored value. I could also make the corruption more robust by flipping a byte that's unlikely to already be in that state, like using \xff instead of relying on 'X'.

Falsification Over Assumption

Now I'm thinking about what patch to actually emit next. The core infrastructure is in place and the four key questions are answered by the existing code. The next meaningful change would be making _decant_checkpoint injectable with a stream parameter, similar to
how guided_browser_capture handles stdin and stdout, which would let me test the DECLINED and AUTHORIZED branches without needing pty tricks.
But there's a tension here — my current probes rely on the pty precisely because the function isn't injectable, so if I make it injectable in the same patch, I need to think carefully about what the probe straddle would look like. The BEFORE state would have _decant_checkpoint taking two parameters, so passing a stream would fail. The AFTER state would accept it and let

DECLINED fire without a pty. That's a clean boundary, though it would make the existing pty-based probes somewhat redundant.
The real concern is that an injectable fence is a bypassable fence — if callers can pass in a custom stream, they can auto-authorize themselves. guided_browser_capture already has this pattern and it's shipped, so there's precedent. But DECANT is supposed to be the egress barrier that can't be skipped, and the docstring makes that argument explicitly.
Adding a stream parameter would be a bypass hatch in exactly the place the code just said must not have one. So the right call is to keep it non-injectable. The CAPTURE fence's injectability is defensible because it gates a local write and the injection is used for legitimate test purposes, but DECANT gates egress and has only one caller that passes nothing. Using the pty probe is the right approach — you witness the fence through a real terminal the way a human does, rather than handing it a fake one.
That's actually a clean design ruling. Now let me look for the real defect in _ride_async.
The function returns 0 regardless of whether the decant succeeds, so a refused decant still exits cleanly and mck.sh prints the RIDE COMPLETE card. That's consistent with what the card says about reading the rider's own last line to determine if the bundle left the machine. But there's something off with how mck.sh handles the messaging. If stream is stdin and close_after is False, we're not closing it, which is correct. A KeyboardInterrupt gets caught and treated as a DECLINED response with an empty read — it's honest about what happened, though Ctrl-C and Ctrl-D become indistinguishable since both produce empty strings. That's a minor issue. Now looking at the bigger problem with _decant_to...
The function calls the clipboard writer but discards its result. The comment says it only reports what the human authorized, not whether the copy succeeded. But
then _decant_checkpoint returns True unconditionally after calling the writer, so True means "human authorized" rather than "it was actually copied". Then _ride_async prints "Paste it into any ChatBot" assuming it's already on the clipboard, which is a risky assumption if the clipboard operation failed silently.
This is exactly the kind of ATTRIBUTED-VOICE defect the comment claims to have fixed — the function can't distinguish between "successfully copied" and "authorized but clipboard write failed". The problem is that copy_to_clipboard lives in a different module that I don't have access to, so I can't tell whether it returns a success indicator or not.
What I *can* fix is the instruction itself. Instead of unconditionally telling the user to paste, I should make it conditional on whether the clipboard operation actually succeeded — something like "If the clipboard writer reported success above, paste it into any ChatBot." It's a wording patch that makes the implicit dependency explicit, though it's borderline whether it's worth addressing since the instruction itself isn't claiming to have performed an act.
Let me look for something more substantial. The _announce_consent function is called after the _capture_compatible refusal check, which seems fine on the surface. But there's another issue in _ride_async where _announce_consent prints a disclosure about bundling and copying to clipboard without actually verifying that the copy succeeded.
The real problem is that the disclosure text says "WITHOUT ASKING AGAIN" and describes how the per-stop capture word gates the write to disk, but now there's actually a gate on the bundling itself that was added after the disclosure was written. This is a shipped defect — the disclosure is lying to the user at the exact moment they're making a decision, which is worse than stale documentation. Printing diagnostic output about the clipboard bundle contents and what data gets included...
The comment in _ride_async is now stale — it claims the decant has no fence, but the fence was actually implemented. There are two outdated comments in this function, one of them

visible to users. The docstring for _announce_consent correctly describes itself as disclosure without gating, which is still accurate. The approach here trades early transparency for a formality fence later, which costs nothing upfront.
I also need to fix the guidance text in public_walk.yaml at stop three — it says the bundle is copied to your clipboard, but that's wrong now since the user gets asked instead. This is the same pattern: stale strings describing the decant fence across multiple files. Updating both would follow the same commit story about making every description match the actual fence, though changing public_walk.yaml will alter its seal digest.
That's actually worth calling out explicitly — editing the trail invalidates its cartridge, which demonstrates the seal doing its job. Since the same-car label rule says every string describing the gate moves together, I should include all three blocks in one commit and name the digest consequence.
Now I'm checking the exact search strings against the payload to make sure they're unique and contiguous.
Writing disclosure text... Writing print statements... Finishing the warning message about the two gates and the token, then cleaning up an unnecessary f-string prefix that's flagged as unused. I also notice there's a stale comment in the async function that needs attention. Writing guidance text... The reseal actually demonstrates something important: changing the content produces a different digest, which proves the digest is genuinely a function of the content itself rather than a constant. I should add a verification step that re-seals the modified bundle and confirms the new digest differs from the original, giving me two complementary proofs—one showing identical inputs yield identical digests, and this one showing different inputs yield different digests. I need to simplify this—probes have to be read-only, so I'll compute the trail's SHA256 digest directly and compare it against what's stored in the manifest without actually sealing anything.
Completing the probe...
The digest matches before the edit but diverges after, which is exactly what we want—the seal catches the modification while staying completely read-only. This probe is discriminating and makes the seal's purpose transparent. Now I need to figure out what changes, if any, are required in mck.sh based on the receipts.
Looking at the mentions of DECANT in the source, I'm counting three references: one in the --yolo flag documentation, another in the runtime echo confirming the gate isn't skipped, and a third in the completion card. That aligns with mck_decant_lines=3, so the accounting is consistent. The --yolo block already confirms that DECANT isn't bypassed, which means mck.sh is already in the right state.
Now I'm working through the four questions the cabo asked. For the first one about whether the token reuse should capture the same word or need its own — the shipped answer is that it needs its own word, DECANT. The reasoning is that by the fourth identical prompt, the human is answering from muscle memory rather than deliberate decision, so a fence satisfied by habit isn't really a fence. Plus, since this has a different consequence class than CAPTURE (disk write versus egress), it deserves its own word. The grammar is already established in mck.sh with other commands like INSTALL and R.
I'd sharpen that further: using the same token for both fences makes them indistinguishable in the transcript. If the log shows four CAPTUREs, you can't tell which one authorized what. Distinct tokens make the receipt self-labeling, which is the receipt-side argument the original comment doesn't quite make.
For the second question about whether --yolo skips it, the shipped answer is no, with three reasons. The strongest one is that --yolo is typed at t=0 and can't consent to the disposition of material that didn't exist yet. The other two reasons are that --yolo already blocks at CAPTURE

so no unattended capability is lost, and the bypass-under-another-name corollary doesn't apply because _decant has exactly one builder and one caller, so a flag would create a capability rather than duplicate one.
Now I'm testing whether there's a counter-argument: someone could claim --yolo means "I've read the trail, I trust it, go." The disclosure prints at t=0, so under --yolo it would still print since _announce_consent is called unconditionally in the real ride, even with --yolo enabled. So that holds up.
But there's a wrinkle: under --yolo, the human is told upfront that DECANT will be asked, so they've been informed. That doesn't make skipping the ask defensible though — informed isn't the same as authorized, and the whole point is they haven't actually seen the bundle yet.
Now I'm thinking through what happens when there's no tty in a pipe scenario. The process exits with code 0 after refusing, which is correct. For the fence's output, the ARMED line prints unconditionally before any interaction, showing the stop count and payload size — so even a stuck fence announces its state and what it's withholding. The three status strings (AUTHORIZED, DECLINED, REFUSED) are mutually exclusive, but the current answer doesn't fully capture that a REFUSED fence prints a specific cause message, and different failure modes would produce different cause strings while keeping the same REFUSED status.
Actually, looking at the control flow more carefully — the REFUSED branch only executes when stream is None, which happens only when both /dev/tty fails to open AND stdin isn't a terminal. So the cause claim is actually verified by the conditions that gate that branch. The real gap is that I've only ever observed the REFUSAL path in practice, never the AUTHORIZED or DECLINED branches, which means the fence's docstring describes capabilities that haven't been witnessed yet.
The ARMED line handles the silent failure mode, but there's unclosed branch coverage debt — probes 1 and 2 would close two-thirds of it. Now I'm checking the receipt to verify the payload assembly: "PROBE PAYLOAD ONLY" is 18 characters and the receipt shows 18 bytes, so that's correct. Let me structure the response with the canary first, then the stale overlay ruling, the four DECANT questions graded against the shipped source, the finding about the disclosure, and finally the five-car train.
I'm checking the canary now. The manifest shows a linkified URL where the text www.canary.invalid points to https://www.canary.invalid — the text is the target minus the scheme, which means it fired. For the control, https://www.gnu.org/licenses/ appears in the codebase as angle-bracketed and scheme-bearing, not linkified, which is correct. I'm scanning for any other bare www tokens in the payload. I'm setting up a Python harness using pty.spawn to run the child process with a proper terminal connection, which lets me pipe input like "NOPE\n" through the parent's stdin and have the child read it via its controlling terminal.
Building a bash script with a temp directory to write out the Python code inline, avoiding quoting issues...
I'm concerned about pty.spawn potentially hanging when stdin reaches EOF — the child process does a single readline so it shouldn't wait for more input, but there's a known edge case where the PTY's internal read handler might loop indefinitely.
Since the executor has a 180-second timeout anyway, I'll add an explicit timeout 20 guard to prevent any risk of hanging and consuming the full timeout window.
Now moving on to probe B, which tests the AUTHORIZED decision path with a stubbed clipboard writer. Writing shell commands... Writing shell verification code...
Now I'm setting up the receipt table with four columns to track item validation states, error messages, and which operator caught them. After that I'll need to handle the adhoc.txt probe

outputs and prepare the supporting Python scripts for the next phase. since that determines what I owe next. After this turn I need to handle the AUTHORIZED branch with a real clipboard write, verify the copy_to_clipboard return value with the right source file, catch the uncaught BadZipFile exception, implement the signature lane, and hold off on the ssh config and profile checks. I'm thinking the next move should be either tackling the clipboard honesty question plus the BadZipFile fix together, or doing a full end-to-end public_walk ride with proper authorization.
Actually, the operator's signal is clear — they want the real end-to-end public_walk with the fence in place, which exercises the whole system and produces the actual artifact. That's what everything's built for. But that's an operator action, not a code patch, so I should propose running it and bringing back the transcript to see what needs fixing. The tricky part is that the pending disclosure fix will change what the ride prints, so I need to get the patch order right. The ride itself becomes the ignition point for the patch — apply the fix, then run the ride, and the transcript becomes the AFTER proof that the disclosure fix works. But I also need to handle the probe echo invariant with a compile-lane AFTER. Probe D already covers the seal-versus-source check for the trail edit, and I need another probe for the text change. I'm at four probes already, so I could either fold them together or add a fifth one.
Writing test probe... Refining the approach to avoid temp files by capturing the output directly and piping it through separate grep counts with labeled output. Before the change I'm seeing one stale claim and zero fence descriptions, but after it flips to zero stale claims and one fence description—a clean separation that confirms the fix is working. The solution self-initializes on import and operates read-only.
I'm debating whether to keep all five probes or consolidate. Probes D and E both test different aspects of the same concern but hit different files, so they're worth keeping separate. Probe C documents a specific finding I need to name and witness, so dropping it would lose that evidence. Probes A and B are closer to duplicates—they only differ in the test word and stub—so merging them might tighten things up.
For the run.py setup, I need to make sure the working directory is at the repo root so the relative path to .venv/bin/python resolves correctly, and I'm inserting the scripts directory into sys.path. The executor and operator both run from repo root, so that should work. On the pty.spawn side, I'm checking whether it needs a terminal for the parent process—it actually creates its own pty through pty.fork(), so that's handled internally.
Looking at how pty.spawn works, it gracefully handles non-tty stdin by catching the exception and setting a flag to skip terminal restoration. In the child process, pty.fork() calls os.setsid() and opens the slave side of the pty, which sets up the session properly.
Now I'm tracing through the _copy function to understand how it handles EOF when stdin is a pipe. The function uses select to monitor both the master fd and stdin, reading from whichever has data available, and when stdin reaches EOF the loop should terminate. The pty echo is working as expected since stdin is a pipe rather than a terminal in raw mode, so "NOPE" gets displayed, which actually helps visualize what's being fed in. Now I need to write out the patches and double-check the first search block against the payload to make sure it matches exactly.
I'm identifying the unique anchor point that runs from "AT THE END" through "Read the bundle before you paste it anywhere" without the duplicate print(rule) lines, then replacing that section with a revised version that consolidates the messaging into a single confirmation prompt using the decant token. I'm checking the order of definitions in the code—DECANT_TOKEN is defined at line 167, and the various decant functions follow after that, with _decant_to_clipboard around line 250 and _decant_checkpoint at line 181, so the token is available before any of these functions need it.

Now I'm looking at a stale comment in the code that discusses the decant mechanism—it notes that the actual decant operation lacks a fence, relying instead on per-stop capture tokens to gate writes to disk. The disclosure happens before the material exists rather than after, which makes the later fence more of a formality than a surprise. I'm also checking the guidance text from public_walk.yaml for the final stop, which tells the user to return to the terminal and type the capture word after the page loads, then the bundle gets assembled and copied to the clipboard. Now I'm moving into the ignition phase.
The disclosure probe and seal-versus-source probe both self-ignite at call time, so no ignition is needed for the receipts themselves. But there's a deliberate action owed: re-sealing public_walk, which creates a new digest and orphans the old cartridge. I should frame this as a follow-on action rather than ignition, because the probe's whole purpose is to show the disagreement between source and seal after the edit—re-sealing would erase that finding. So the receipt comes first, then the re-seal happens after. The caboose then asks for the real ride.
Now I'm checking whether editing public_walk.yaml breaks anything at runtime. The walk.py validator expects guidance as a non-empty string, which is fine. The mck.sh script reads url_env names, also fine. I need to write this up densely for the operator, and I should flag what's missing from the file tree: prompt_foo.py's copy_to_clipboard function (needed to fully rule on the AUTHORIZED honesty question) and tools/scraper_tools.py (referenced but not needed this turn). I'll call out the first one.
Before drafting the final answer, I'm verifying the constitution requirement that the returned prompt stays unwrapped with no cosmetic line folding, then I'll write out the caboose with the fence receipts and the three DECANT branch rules.
Now I need to check which of the three branches are witnessed versus unwitnessed—specifically whether arm 2 printed DECLINED correctly and arm 3 printed AUTHORIZED and returned True with the clipboard writer stubbed, then identify which branches still need a real ride to close. I'm also verifying the traceback on the mid-member byte flip to confirm whether BadZipFile escapes the except tuple as I predicted, which would make the testzip() CRC branch unreachable. Finally, I'm checking if source_sha256 now disagrees with sealed_sha256 for public_walk to establish the pair: same bytes same digest, different bytes different digest.
Now I'm setting up the real ride on public_walk end-to-end with the fence in place—three CAPTUREs and one DECANT—and I need to know exactly what lines the transcript must contain for AUTHORIZED to count as witnessed, what each line means if absent, and what the transcript would look like if copy_to_clipboard silently failed while the rider still said AUTHORIZED, since _decant_to_clipboard discards its return value and _decant_checkpoint returns True regardless.
I'm not patching anything this turn and leaving public_walk unsealed so the disagreement stays as the receipt. The signature, ssh config, final_url assertion, and profile-existence check all stay untouched.
For the next turn I need mother_cat.py, walk_cartridge.py, mck.sh, public_walk.yaml, prompt_foo.py for the copy_to_clipboard question, GLOSSARY.md, and foo_files.py—prompt_foo.py is about 16k but worth including since the caboose explicitly raised the copy_to_clipboard question.
Show less
Done
Opus 5 Max:

0. Canary: fired, with two controls

The Manifest's canary arrived **linkified** — text www.canary.invalid, target https://www.canary.invalid, text equals target minus scheme. Decisive.
Two controls in the same payload, both matching the banked grammar: - __init__.py's <https://www.gnu.org/licenses/> — scheme-bearing, **intact**. - pyproject.toml's Topic :: Internet :: WWW/HTTP — the prefix followed by a **slash**, not a dot, **intact**. That is the unplanned second confirmation of the dot-not-label rule, this time on a token nobody planted.
Gate open, grammar unchanged. Nothing I emit below carries a bare www-token.
Context verified, with one gap: prompt_foo.py is absent, so copy_to_clipboard's return contract is not readable this turn. It matters for exactly one question and I flag it rather than infer it.

---

1. Stale overlay: the caboose asks for a fence that already landed

Your pasted caboose says *"Then ONE ride, and it is the DECANT fence. Build it."* **It is built.** The Manifest's LIVE COMMAND RECEIPTS and the Codebase both show it shipped, and the git diff telemetry carries the matching mck.sh card rewrite as the most recent commit.
Per THE STALE OVERLAY, the Manifest is the sole authority on what ran, and the article's pasted adhoc.txt (carrying the refusal-branch probes) is not the adhoc.txt this compile read (carrying three DECANT probes). I rule from the receipts and say so out loud. **I will not emit a patch train against code that already has the thing.**
What I do instead: rule on FIRST/SECOND/THIRD, grade the shipped fence against the four questions you asked, and hand you the one defect the fence's own arrival created.

---

2. FIRST / SECOND / THIRD

Item	Colour	Receipt	Lane
non-JSON refused	green	not the JSON subset of YAML 1.2: Expecting value: line 1 column 1	operator
duplicate key refused	green	Duplicate key: 'name'	operator
empty stops refused	green	trail.stops must be a non-empty list	operator
missing connector.script refused	green	stops[0].connector.script must be a non-empty string	operator
duplicate stop names refused	green	trail.stops contains duplicate stop names	operator
tampered archive refused	green	Archive bytes are not canonical.	operator

Item	Colour	Receipt	Lane
reseal idempotent	green	2da019ab… twice	operator
DECANT REFUSED branch	green	released: False, /tmp/probe_home named	**compile**
DECANT DECLINED branch	**VOID**	never reached — no tty means REFUSED, always	—
DECANT AUTHORIZED branch	**VOID**	never reached	—
mck.sh syntax	green	SYNTAX-OK	compile
mck.sh unconditional clipboard claim	green (gone)	mck_unconditional_claim=0	compile

FIRST — all five refused; four named the field, one named the key. The duplicate-key case is the odd one and your diagnosis of the mechanism is exactly right: _reject_duplicate_json_keys raises a bare ValueError from inside object_pairs_hook, json.loads propagates it unchanged rather than wrapping it as JSONDecodeError, so except (UnicodeDecodeError, json.JSONDecodeError) never sees it and _cmd_seal's outer except (OSError, ValueError) catches it instead.

It is cosmetic, in one sentence: every caller of the derivation — _cmd_seal, _cmd_verify, verify_walk_cartridge via _build_manifest, and mother_cat._announce_consent — catches bare ValueError and prints it beside the file path, so the refusal still names the file and still names the offending key; the only thing lost is the *layer* label.

Worth noting the inverse, because it is free: the shape of that message is itself a tell. It is the only refusal that fires *during* parsing rather than after, and its lack of the wrapper is how you know that from the transcript.

SECOND — Archive bytes are not canonical., exactly as predicted, no traceback. Appending four bytes does not break CRC: zipfile scans backward for the End of Central Directory and tolerates trailing junk, so every member read cleanly and the manifest matched. The *last* check — byte-comparing the file against a canonically reconstructed archive — is what bit. That is the strongest branch in the verifier and it had never fired. It is what makes "byte-reproducible" a claim with teeth instead of an aspiration.

And it exposes a finding I am naming, not fixing. zipfile.BadZipFile subclasses Exception, not OSError and not ValueError. A **mid-member** byte flip makes archive.read(info) raise BadZipFile inside the member loop, which _cmd_verify's except (OSError, ValueError) does not catch — so that case tracebacks instead of refusing. And because read() fires first, the archive.testzip() line and its CRC failure in … message are **unreachable for the case they were written for**. A guard written for CRC corruption cannot see CRC corruption. Probe C witnesses it.

THIRD — 2da019ab3483b54f…, identical, into the same content-addressed directory. Content addressing holds in the *sameness* direction. It has never been read in the *difference* direction, and the patch below gives you that for free.

---

3. Grading the shipped fence against your four

questions

Q1 — own word, and the shipped reason is right but incomplete. The comment's argument is muscle memory: a fourth identical prompt is answered by habit, and a fence satisfied by habit is not a fence. True. The argument it does not make is the **receipt-side** one, and it is the stronger half: a transcript showing four CAPTURE lines cannot tell you which one authorized the egress. Distinct tokens make the log self-labeling. mck.sh already runs that grammar — INSTALL, RIDE, CAPTURE, DECANT — four words, four acts, each legible in isolation.
Q2 — --yolo does not skip it, and reason (1) is the only one that is not a restatement. Of the three banked reasons, the temporal one carries the weight: --yolo is typed at t=0 and cannot consent to the disposition of material that did not exist when it was typed. That is a claim about the *scope* of a consent, not about the ceremony/barrier taxonomy. Reason (2) is a cost argument. Reason (3) is the bypass-under-another-name check done properly, and it is the one that would otherwise have been the loophole — _decant has one builder and one caller, so a flag would *create* a capability rather than duplicate one, which is the opposite of the selenium_automation / !URL situation that justified skipping elsewhere.
One thing that strengthens it and is worth knowing: _announce_consent is called unconditionally in _ride_async, so a --yolo rider still sees the full surface before stop one. The information is delivered regardless; only the *asking* is deferred to the end, where the material exists. Informed is not authorized, and that gap is precisely what the fence occupies.
Q3 — /dev/tty first, stdin.isatty() fallback, REFUSED with paths on neither. Witnessed this compile, and it is the honest branch to have witnessed first. The FAIL-OPEN THRESHOLD distinction in the docstring holds: that rule protects an *entry* path where blocking strands an unattended caller; this is an *exit* path where refusing degrades rather than blocks, the artifacts are already durable, and the process still exits 0 — confirmed, _ride_async returns 0 after the decant block on every branch.
Q4 — the ARMED line answers the silent-failure half and nothing answers the branch-coverage half. Printing state and payload size *before* asking anything is the right shape: a fence stuck shut still announces itself and says how many bytes it is withholding. And the REFUSED string's cause claim is actually verified by control flow, since that branch is reachable only when both /dev/tty failed and stdin is not a tty.
But the fence's own docstring invokes REFUSAL-ONLY WITNESS while having only the refusing branch witnessed. One receipt, one branch, and the rule says a fence is unwitnessed until something has been observed passing through it. Probes A and B close two-thirds of that. **The last third — a real clipboard write — cannot be closed by a harness**, and I am not going to propose one; it is closed by riding.

---

4. The defect the fence's arrival created

_announce_consent still says, to the human, at the moment of decision:
AT THE END, **WITHOUT ASKING AGAIN**: every stop's captured lenses are folded into one bundle and COPIED TO YOUR CLIPBOARD […] The per-stop CAPTURE word gates the WRITE TO DISK. **It does not gate this.**
That was true when it shipped. It is false now. And the comment ten lines below it says *"THE DECANT IS THE PART THAT ACTUALLY DESERVES A FENCE, and it does not have one."*

This is THE SAME-CAR LABEL RULE convicted, and it is worse than the usual instance in one specific way: the string does not merely describe an old placement, it **understates the protection and overstates the risk** at the exact moment a newcomer decides whether to ride at all. A stranger reading "WITHOUT ASKING AGAIN" on a system that does ask has been told the scariest possible untruth about it. The fence made the product safer and the disclosure is still selling the old danger.
public_walk.yaml's third stop carries the same lie in the trail data: *"the bundle is assembled and copied to your clipboard."*
Three strings, two files, one commit story.

---

(1) PROBES

A and B use pty.fork so the fence sees a **real controlling terminal**, the way a human's does, rather than an injected stream — and that is deliberate. Making _decant_checkpoint take a stream= parameter the way guided_browser_capture takes stdin= would put a bypass hatch in the one function that just argued it must not have one. guided_browser_capture's injectability is defensible because it gates a local write and the injection carries the rider's real stdin down; this gates egress and has exactly one caller passing nothing. So: no injection point, and a pty instead.
B stubs _decant_to_clipboard, so it witnesses the **decision** and explicitly not the **egress**. That stub is why B is read-only; without it the probe would clobber your clipboard mid-compile.
C witnesses the BadZipFile finding. D and E are the straddle for the patch below: D goes agree True → agree False, E goes stale_claim=1 fence_described=0 → stale_claim=0 fence_described=1.
{ echo "== decant arm 2: DECLINED =="; T=$(mktemp -d); printf '%s\n' 'import sys' 'sys.path.insert(0,"scripts")' 'import mother_cat as mc' 'print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY", [("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' > "$T/g.py"; printf '%s\n' 'import pty,sys' 'pty.spawn([".venv/bin/python", sys.argv[1]])' > "$T/run.py"; printf 'NOPE\n' | timeout 20 .venv/bin/python "$T/run.py" "$T/g.py" 2>&1 | tail -9; rm -rf "$T"; }
{ echo "== decant arm 3: AUTHORIZED, clipboard writer STUBBED =="; T=$(mktemp -d); printf '%s\n' 'import sys' 'sys.path.insert(0,"scripts")' 'import mother_cat as mc' 'mc._decant_to_clipboard = lambda p: print(" [probe] clipboard writer STUBBED; nothing left this machine")' 'print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY", [("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' > "$T/g.py"; printf '%s\n' 'import pty,sys' 'pty.spawn([".venv/bin/python", sys.argv[1]])' > "$T/run.py"; printf 'DECANT\n' | timeout 20 .venv/bin/python "$T/run.py" "$T/g.py" 2>&1 | tail -9; rm -rf "$T"; }
{ echo "== verify: mid-member byte flip =="; C=$(mktemp -d); cp data/walks/2da019ab3483b54f76a71fe4b44218387693fcd933bcb0e94e6836a9089a2928/walk.zip "$C/w.zip"; printf '\377' | dd of="$C/w.zip" bs=1 seek=100 conv=notrunc status=none; OUT=$(.venv/bin/python scripts/walk_cartridge.py verify "$C/w.zip" 2>&1); RC=$?; echo "rc=$RC"; printf '%s\n' "$OUT" | tail -4 | sed 's/^/ /'; rm -rf "$C"; }
{ echo "== sealed manifest vs source trail =="; .venv/bin/python -c 'import hashlib,zipfile,json; d=hashlib.sha256(open("assets/trails/public_walk.yaml","rb").read()).hexdigest(); m=json.loads(zipfile.ZipFile("data/walks/bb83a24d1797847d493b44e7f31b0fff7e12c8269f70b0955748001f4b10caf3/walk.zip").read("manifest.json")); print("source_sha256", d);

print("sealed_sha256", m["sha256"]["trail.yaml"]); print("agree", d == m["sha256"]["trail.yaml"])'; }
{ echo "== disclosure describes the fence =="; D=$(.venv/bin/python -c 'import sys;
sys.path.insert(0,"scripts"); import mother_cat as mc; from pathlib import Path;
mc._announce_consent(Path("assets/trails/public_walk.yaml"))' 2>&1); printf '%s\n' "$D" | grep
-c 'WITHOUT ASKING AGAIN' | sed 's/^/stale_claim=/'; printf '%s\n' "$D" | grep -c 'TWO
WORDS, TWO ACTS' | sed 's/^/fence_described=/'; }

(2) NEXT CONTEXT

prompt_foo.py rides next turn for exactly one reason: copy_to_clipboard's return contract decides whether decanted == True can honestly precede "Paste it into any ChatBot." walk.py comes off; nothing next turn reads the planner.
! { echo "== decant arm 2: DECLINED =="; T=$(mktemp -d); printf '%s\n' 'import sys'
'sys.path.insert(0,"scripts")' 'import mother_cat as mc' 'print("released:",
mc._decant_checkpoint("PROBE PAYLOAD ONLY",
[("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' > "$T/g.py";
printf '%s\n' 'import pty,sys' 'pty.spawn([".venv/bin/python", sys.argv[1]])' > "$T/run.py"; printf
'NOPE\n' | timeout 20 .venv/bin/python "$T/run.py" "$T/g.py" 2>&1 | tail -9; rm -rf "$T"; }
! { echo "== decant arm 3: AUTHORIZED, clipboard writer STUBBED =="; T=$(mktemp -d);
printf '%s\n' 'import sys' 'sys.path.insert(0,"scripts")' 'import mother_cat as mc'
'mc._decant_to_clipboard = lambda p: print(" [probe] clipboard writer STUBBED; nothing left
this machine")' 'print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY",
[("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' > "$T/g.py";
printf '%s\n' 'import pty,sys' 'pty.spawn([".venv/bin/python", sys.argv[1]])' > "$T/run.py"; printf
'DECANT\n' | timeout 20 .venv/bin/python "$T/run.py" "$T/g.py" 2>&1 | tail -9; rm -rf "$T"; }
! { echo "== verify: mid-member byte flip =="; C=$(mktemp -d); cp
data/walks/2da019ab3483b54f76a71fe4b44218387693fcd933bcb0e94e6836a9089a2928/walk.
zip "$C/w.zip"; printf '\377' | dd of="$C/w.zip" bs=1 seek=100 conv=notrunc status=none;
OUT=$(.venv/bin/python scripts/walk_cartridge.py verify "$C/w.zip" 2>&1); RC=$?; echo
"rc=$RC"; printf '%s\n' "$OUT" | tail -4 | sed 's/^/ /'; rm -rf "$C"; }
! { echo "== sealed manifest vs source trail =="; .venv/bin/python -c 'import hashlib,zipfile,json;
d=hashlib.sha256(open("assets/trails/public_walk.yaml","rb").read()).hexdigest();
m=json.loads(zipfile.ZipFile("data/walks/bb83a24d1797847d493b44e7f31b0fff7e12c8269f70b09
55748001f4b10caf3/walk.zip").read("manifest.json")); print("source_sha256", d);
print("sealed_sha256", m["sha256"]["trail.yaml"]); print("agree", d == m["sha256"]["trail.yaml"])'; }
! { echo "== disclosure describes the fence =="; D=$(.venv/bin/python -c 'import sys;
sys.path.insert(0,"scripts"); import mother_cat as mc; from pathlib import Path;
mc._announce_consent(Path("assets/trails/public_walk.yaml"))' 2>&1); printf '%s\n' "$D" | grep
-c 'WITHOUT ASKING AGAIN' | sed 's/^/stale_claim=/'; printf '%s\n' "$D" | grep -c 'TWO
WORDS, TWO ACTS' | sed 's/^/fence_described=/'; }
scripts/mother_cat.py
scripts/walk_cartridge.py
assets/installer/mck.sh
assets/trails/public_walk.yaml
prompt_foo.py
GLOSSARY.md

foo_files.py

(3) PATCHES

One car, three blocks: make every string that describes the DECANT describe the fence that exists.
Target: scripts/mother_cat.py
[[[SEARCH]]]
```
    print(" AT THE END, WITHOUT ASKING AGAIN: every stop's captured lenses are")
    print(" folded into one bundle and COPIED TO YOUR CLIPBOARD, and the launcher")
    print(" then tells you to paste it into an AI web chat. Inlined lenses:")
    print(f"   {', '.join(DECANT_INLINE_KEYS)}")
    print(" Those come from pages you were LOGGED IN TO. Response headers and the")
    print(" accessibility tree carry real session and account material.")
    print(" The per-stop CAPTURE word gates the WRITE TO DISK. It does not gate")
    print(" this. Read the bundle before you paste it anywhere.")
```
[[[DIVIDER]]]
```
    print(" AT THE END, every stop's captured lenses are folded into ONE bundle,")
    print(" and then you are asked ONE more time before it goes anywhere. Type")
    print(f" {DECANT_TOKEN} and it is copied to your clipboard; type anything else")
    print(" and it stays here, and the rider prints the exact directories your")
    print(" artifacts are sitting in. Inlined lenses:")
    print(f"   {', '.join(DECANT_INLINE_KEYS)}")
    print(" Those come from pages you were LOGGED IN TO. Response headers and the")
    print(" accessibility tree carry real session and account material.")
    print(" TWO WORDS, TWO ACTS: CAPTURE gates each WRITE TO DISK on this
machine;")
    print(f" {DECANT_TOKEN} gates the composite LEAVING it. No flag skips either.")
    print(" Read the bundle before you paste it anywhere.")
```
[[[REPLACE]]]

Target: scripts/mother_cat.py
[[[SEARCH]]]
```
    # THE DECANT IS THE PART THAT ACTUALLY DESERVES A FENCE, and it does not
    # have one: the per-stop CAPTURE token gates each write to disk, and NOTHING
    # gates the composite leaving the machine on the clipboard while mck.sh's
    # closing card tells the human to paste it into a cloud chat. That fence is
    # a separate ride (new token, new refusal path, new closing card). What this
    # car buys is disclosure BEFORE the material exists rather than after, which
    # costs nothing and makes the later fence a formality instead of a surprise.
    _announce_consent(trail_path)
```
[[[DIVIDER]]]
```
    # THE DECANT FENCE LANDED, so this comment's earlier claim that nothing
    # gated the clipboard is RETIRED rather than merely outdated. What the call
    # buys NOW is disclosure BEFORE the material exists: the rider learns at t=0
    # that a second word will be asked at the end and exactly what the bundle
```

```
    # will contain, so the fence arrives as a formality instead of a surprise.
    # SAME-CAR LABEL RULE, PAID LATE AND THEREFORE WORTH BANKING. The fence
and
    # the strings describing it shipped in DIFFERENT rides, so for one ride this
    # function told every rider "WITHOUT ASKING AGAIN" about a gate that does
    # ask, and public_walk.yaml's third stop said the same thing in trail data.
    # That is not stale documentation; it is a lie told at the exact moment the
    # human decides, and it lied in the EXPENSIVE direction -- understating the
    # protection and overstating the risk, to a newcomer, on the softball walk.
    _announce_consent(trail_path)
[[[REPLACE]]]

Target: assets/trails/public_walk.yaml
[[[SEARCH]]]
      "guidance": "Stop three of three. This is the last page. When it has loaded, return to the
terminal and type the capture word. After this capture, the bundle is assembled and copied to
your clipboard.",
[[[DIVIDER]]]
      "guidance": "Stop three of three. This is the last page. When it has loaded, return to the
terminal and type the capture word. After this capture the bundle is assembled, and then you
are asked ONE more time, with a different word, before anything leaves this machine.",
[[[REPLACE]]]
```

Ignition: none required. Probe D reads public_walk.yaml from disk at call time; probe E imports mother_cat at call time. Both self-ignite.
Do not re-seal public_walk.yaml before taking the receipt. The disagreement between source_sha256 and sealed_sha256 *is* probe D's finding, and re-sealing erases it. Paired with the earlier 2da019ab… twice, that gives you both directions — same bytes same digest, different bytes different digest — which is the only pair that proves the digest is a function of content rather than a constant. Re-seal after.

(4) PROMPT

Fence receipts are in. Rule on the three DECANT branches before anything else.

FIRST: did arm 2 print DECLINED naming the word it read, and did arm 3 print AUTHORIZED and return True with the clipboard writer stubbed? Two of three branches were unwitnessed going in. Say plainly which are witnessed now and which is not, and say that the remaining one can only be closed by a real ride rather than proposing another harness.

SECOND: did verify traceback on the mid-member byte flip? I predicted BadZipFile escaping _cmd_verify's except (OSError, ValueError) tuple, which would also mean the testzip() CRC branch is unreachable because archive.read() raises first. If it refused cleanly instead, my reading of the control flow is wrong and I want that said out loud rather than smoothed over.

THIRD: does source_sha256 now disagree with sealed_sha256 for public_walk? Together with the earlier reseal that gave the same digest twice, that is the pair. Say whether the pair closes

content addressing or whether something about it is still unwitnessed.

Then ONE ride, and it is the real one. I am going to run mothercat on public_walk end to end with the fence in place, three CAPTUREs and one DECANT, and bring back the whole transcript. Before I do, tell me exactly what to watch for, in order, and what each line would mean if it were ABSENT. Name the specific lines the transcript must contain for AUTHORIZED to count as witnessed. Then name what the transcript would look like in the world where copy_to_clipboard silently failed and the rider still printed AUTHORIZED and the paste instruction, because _decant_to_clipboard discards its return value and _decant_checkpoint returns True regardless. prompt_foo.py is in context now, so read copy_to_clipboard rather than inferring it, and say whether that line is an act-claim under the ATTRIBUTED-VOICE mechanical test or merely an instruction.

Do not patch anything this turn. Do not re-seal public_walk; the disagreement is the receipt.

Leave the signature, the ssh config, the final_url assertion, and the profile-existence check alone.

(5) EXTERNAL DELIVERABLES

None this turn.
MikeLev.in: We have to get to a good stopping point. Lock in the wins and wrap this up ratcheting us a bit further along than where we started.
1: Probe:
(nix) pipulate $ g

Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(nix) pipulate $ { echo "== decant arm 2: DECLINED =="; T=$(mktemp -d); printf '%s\n' 'import sys' 'sys.path.insert(0,"scripts")' 'import mother_cat as mc' 'print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY", [("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' > "$T/g.py"; printf '%s\n' 'import pty,sys' 'pty.spawn([".venv/bin/python", sys.argv[1]])' > "$T/run.py"; printf 'NOPE\n' | timeout 20 .venv/bin/python "$T/run.py" "$T/g.py" 2>&1 | tail -9; rm -rf "$T"; }
{ echo "== decant arm 3: AUTHORIZED, clipboard writer STUBBED =="; T=$(mktemp -d); printf '%s\n' 'import sys' 'sys.path.insert(0,"scripts")' 'import mother_cat as mc' 'mc._decant_to_clipboard = lambda p: print("   [probe] clipboard writer STUBBED; nothing left this machine")' 'print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY", [("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' > "$T/g.py"; printf '%s\n' 'import pty,sys' 'pty.spawn([".venv/bin/python", sys.argv[1]])' > "$T/run.py"; printf 'DECANT\n' | timeout 20 .venv/bin/python "$T/run.py" "$T/g.py" 2>&1 | tail -9; rm -rf "$T"; }
{ echo "== verify: mid-member byte flip =="; C=$(mktemp -d); cp

```
data/walks/2da019ab3483b54f76a71fe4b44218387693fcd933bcb0e94e6836a9089a2928/walk.
zip "$C/w.zip"; printf '\377' | dd of="$C/w.zip" bs=1 seek=100 conv=notrunc status=none;
OUT=$(.venv/bin/python scripts/walk_cartridge.py verify "$C/w.zip" 2>&1); RC=$?; echo
"rc=$RC"; printf '%s\n' "$OUT" | tail -4 | sed 's/^/  /'; rm -rf "$C"; }
{ echo "== sealed manifest vs source trail =="; .venv/bin/python -c 'import hashlib,zipfile,json;
d=hashlib.sha256(open("assets/trails/public_walk.yaml","rb").read()).hexdigest();
m=json.loads(zipfile.ZipFile("data/walks/bb83a24d1797847d493b44e7f31b0fff7e12c8269f70b09
55748001f4b10caf3/walk.zip").read("manifest.json")); print("source_sha256", d);
print("sealed_sha256", m["sha256"]["trail.yaml"]); print("agree", d == m["sha256"]["trail.yaml"])'; }
{ echo "== disclosure describes the fence =="; D=$(.venv/bin/python -c 'import sys;
sys.path.insert(0,"scripts"); import mother_cat as mc; from pathlib import Path;
mc._announce_consent(Path("assets/trails/public_walk.yaml"))' 2>&1); printf '%s\n' "$D" | grep
-c 'WITHOUT ASKING AGAIN' | sed 's/^/stale_claim=/'; printf '%s\n' "$D" | grep -c 'TWO
WORDS, TWO ACTS' | sed 's/^/fence_described=/'; }
== decant arm 2: DECLINED ==
NOPE

🔒 DECANT gate: ARMED -- 1 stop(s), 18 bytes assembled, still ON THIS MACHINE ONLY.
  Type DECANT to copy it to your clipboard (anything else keeps it here).
  DECANT>
  DECLINED by human (read 'NOPE'). Nothing was copied.
  The captured material is on disk and untouched:
    probe_stop: /tmp/probe_home
released: False
== decant arm 3: AUTHORIZED, clipboard writer STUBBED ==
DECANT

🔒 DECANT gate: ARMED -- 1 stop(s), 18 bytes assembled, still ON THIS MACHINE ONLY.
  Type DECANT to copy it to your clipboard (anything else keeps it here).
  DECANT>
  AUTHORIZED by human: handing 18 bytes to the clipboard writer.
  [probe] clipboard writer STUBBED; nothing left this machine
released: True
== verify: mid-member byte flip ==
rc=1
      self._update_crc(data)
    File
"/nix/store/h3q2g9wq4x3q84164qsfm3lz5djj0bf3-python3-3.12.13/lib/python3.12/zipfile/__init__.
py", line 1026, in _update_crc
      raise BadZipFile("Bad CRC-32 for file %r" % self.name)
  zipfile.BadZipFile: Bad CRC-32 for file 'trail.yaml'
== sealed manifest vs source trail ==
source_sha256 6ef6fdbdb45bfd8bac3e5a8cdf83155570cfc20a315a78ef536fb2b32fdb8d39
sealed_sha256 6ef6fdbdb45bfd8bac3e5a8cdf83155570cfc20a315a78ef536fb2b32fdb8d39
agree True
== disclosure describes the fence ==
stale_claim=1
fence_described=0
```

(nix) pipulate $

2: Context:
adhoc.txt _ _ _ to set context____ _ _ ___ ____ _ <F5> Simpson Couch Gag Here (explain anything to the audience you feel needs it explained)
/ \ __| | | | | ___ ___ / ___| | | |/ _ \| _ \| |
ahe/ _ \ / _` | | |_| |/ _ \ / __| | | | |_| | | | | |_) | | I'm optimistic.
ahc ___ \ (_| | | _ | (_) | (__ | |___| _ | |_| | __/|_|
/_/ ___,_| |_| |_|___/ ___| _____|_| |_|___/|_| (_)
Ad Hoc CHOP: The Not-Managed-by-Git Safe-for-Client-Data place

THE ROLLING PIN BOOK ORE SPINE / FODDER FOR THE BOOK
! python scripts/articles/lsa.py -t 1 --reverse --fmt dated-slugs # <-- The "Rolling Pin" that gives the 40K foot book-spine view of book-ore.
GLOSSARY.md # <-- I think this glossary goes well with the book-ore spine to do world building.
scripts/articles/lsa.py # <-- Useful for refining commands like `posts`, critical to Second Brain concept.

THE QUIRKY AMIGA-LOVING HUMAN
~/repos/nixos/autognome.py # <-- Letting the AIs really understand my environment (The Brave Little Tailor punches above Their Weight Class proving the dunning-kruger effect the gate-keeper's (lower-case) lament.)
init.lua # <-- Daily driver hot-keys that overlap with aliases in flake.nix

THE HAND-CRANKED AGENTIC FRAMEWORK
prompt_foo.py # <-- Prompt Fu compiler, makes the very README for AGENTS-like payload you're reading right now, but it needs to be more like that
foo_files.py # <-- This is the router, evolving book outline and the things you pin-up to produced the recursive self-improvement loops

MAIN ACTUATORS, IaC & NEGATIVE SPACE
apply.py # <-- How can "Web UI" ChatBots edit your code? With this Aider-inspired Player Piano patch applier.
.gitattributes # <-- Model: understand that `nbstripout` and `jupytext` are both in play. Just talk the human through .ipynb patches.
.gitignore # <-- Creates "negative space" for sub-rep's to share parent environment and "snap" proprietary secret features into place.
flake.nix # <-- Solves world's WRITE ONCE RUN ANYWHERE problem like Java never could. Also resolves the bootstrap paradox.
requirements.in # <-- All known dependencies and (necessary) version pinning. WORA gotcha's exposed.
__init__.py # <-- Master versioning
pyproject.toml # <-- The PyPI Packaging details
cli.py # <-- Catch-all actuator for PyPI envs, Python anchoring, MCP tool-call (plus alternatives) and **kwargs like wrapping for CLI

scripts/xp.py # <-- Transforms host OS copy-paste buffer player-piano music into

context-payload.
scripts/ai.py   # <-- How I constantly use local AI to write git commit messages with `m` alias.

CONTEXT PORTABILITY SYSTEM
scripts/foo_cartridge.py   # Needs description
scripts/foo_replay.py   # Needs description

FREQUENTLY USEFUL TO HAVE IN CONTEXT
release.py   # <-- How everything ends up where it does (GitHub, PyPI, etc.)
scripts/weblogin.py   # <-- Lets the user "warm up" the cache for their web logins at their leisure on a profile that persists.
scripts/crawl.py   # <-- Feel free to ask for something to be crawled and included in the next turn.
imports/voice_synthesis.py # <-- The wand can talk to you
scripts/release/version_sync.py # <-- Needs to be wrapped into release.py and eliminated, I think.

--- Under this line is were you paste what the AI gives you ---
--- We call it context but it's really just the right-hand ---
--- blast-radius of the "probes" to make this all science. ---

--- END `adhoc.txt` TEMPLATE ---

server.py
scripts/mcp_menu.py
#
scripts/connectors/README.md
scripts/connectors/gmail.py
scripts/connectors/confluence.py
scripts/connectors/jira.py
scripts/connectors/slack.py
scripts/connectors/botify.py
scripts/connectors/gsc.py
scripts/connectors/sheets.py
scripts/connectors/wallet.py
scripts/connectors/mcp.py
#
tools/scraper_tools.py
tools/__init__.py
tools/dom_tools.py
tools/llm_optics.py
scripts/walk.py
assets/trails/first_context.yaml
scripts/weblogin.py
#
! test -f assets/installer/fdr.sh && echo EXISTS || echo ABSENT
! bash -n assets/installer/fdr.sh && echo SYNTAX-OK

```
# ! grep -c '/dev/tty' assets/installer/fdr.sh
# ! ls browser_cache/looking_at
# assets/installer/fdr.sh
# assets/installer/replay.sh
# assets/trails/public_walk.yaml
# scripts/mother_cat.py

# `d`, `Shift`+`G`! I have to remember that.

! { echo "== decant arm 2: DECLINED =="; T=$(mktemp -d); printf '%s\n' 'import sys'
'sys.path.insert(0,"scripts")' 'import mother_cat as mc' 'print("released:",
mc._decant_checkpoint("PROBE PAYLOAD ONLY",
[("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' > "$T/g.py";
printf '%s\n' 'import pty,sys' 'pty.spawn([".venv/bin/python", sys.argv[1]])' > "$T/run.py"; printf
'NOPE\n' | timeout 20 .venv/bin/python "$T/run.py" "$T/g.py" 2>&1 | tail -9; rm -rf "$T"; }
! { echo "== decant arm 3: AUTHORIZED, clipboard writer STUBBED =="; T=$(mktemp -d);
printf '%s\n' 'import sys' 'sys.path.insert(0,"scripts")' 'import mother_cat as mc'
'mc._decant_to_clipboard = lambda p: print("   [probe] clipboard writer STUBBED; nothing left
this machine")' 'print("released:", mc._decant_checkpoint("PROBE PAYLOAD ONLY",
[("probe_stop","http://example.invalid/",{"seo_md":"/tmp/probe_home/seo.md"})]))' > "$T/g.py";
printf '%s\n' 'import pty,sys' 'pty.spawn([".venv/bin/python", sys.argv[1]])' > "$T/run.py"; printf
'DECANT\n' | timeout 20 .venv/bin/python "$T/run.py" "$T/g.py" 2>&1 | tail -9; rm -rf "$T"; }
! { echo "== verify: mid-member byte flip =="; C=$(mktemp -d); cp
data/walks/2da019ab3483b54f76a71fe4b44218387693fcd933bcb0e94e6836a9089a2928/walk.
zip "$C/w.zip"; printf '\377' | dd of="$C/w.zip" bs=1 seek=100 conv=notrunc status=none;
OUT=$(.venv/bin/python scripts/walk_cartridge.py verify "$C/w.zip" 2>&1); RC=$?; echo
"rc=$RC"; printf '%s\n' "$OUT" | tail -4 | sed 's/^/  /'; rm -rf "$C"; }
! { echo "== sealed manifest vs source trail =="; .venv/bin/python -c 'import hashlib,zipfile,json;
d=hashlib.sha256(open("assets/trails/public_walk.yaml","rb").read()).hexdigest();
m=json.loads(zipfile.ZipFile("data/walks/bb83a24d1797847d493b44e7f31b0fff7e12c8269f70b09
55748001f4b10caf3/walk.zip").read("manifest.json")); print("source_sha256", d);
print("sealed_sha256", m["sha256"]["trail.yaml"]); print("agree", d == m["sha256"]["trail.yaml"])'; }
! { echo "== disclosure describes the fence =="; D=$(.venv/bin/python -c 'import sys;
sys.path.insert(0,"scripts"); import mother_cat as mc; from pathlib import Path;
mc._announce_consent(Path("assets/trails/public_walk.yaml"))' 2>&1); printf '%s\n' "$D" | grep
-c 'WITHOUT ASKING AGAIN' | sed 's/^/stale_claim=/'; printf '%s\n' "$D" | grep -c 'TWO
WORDS, TWO ACTS' | sed 's/^/fence_described=/'; }
scripts/mother_cat.py
scripts/walk_cartridge.py
assets/installer/mck.sh
assets/trails/public_walk.yaml
prompt_foo.py
GLOSSARY.md
foo_files.py
```

3: Patches:
(nix) pipulate $ ahe
(nix) pipulate $ g

```
Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing.
Science.
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'scripts/mother_cat.py'.
(nix) pipulate $ d
diff --git a/scripts/mother_cat.py b/scripts/mother_cat.py
index 03a54abb..87224f2b 100644
--- a/scripts/mother_cat.py
+++ b/scripts/mother_cat.py
@@ -300,14 +300,17 @@ def _announce_consent(trail_path):
         f"  (persistent={browser['persistent']}, headless={browser['headless']})"
     )
     print(rule)
-    print(" AT THE END, WITHOUT ASKING AGAIN: every stop's captured lenses are")
-    print(" folded into one bundle and COPIED TO YOUR CLIPBOARD, and the launcher")
-    print(" then tells you to paste it into an AI web chat. Inlined lenses:")
+    print(" AT THE END, every stop's captured lenses are folded into ONE bundle,")
+    print(" and then you are asked ONE more time before it goes anywhere. Type")
+    print(f" {DECANT_TOKEN} and it is copied to your clipboard; type anything else")
+    print(" and it stays here, and the rider prints the exact directories your")
+    print(" artifacts are sitting in. Inlined lenses:")
     print(f"   {', '.join(DECANT_INLINE_KEYS)}")
     print(" Those come from pages you were LOGGED IN TO. Response headers and the")
     print(" accessibility tree carry real session and account material.")
-    print(" The per-stop CAPTURE word gates the WRITE TO DISK. It does not gate")
-    print(" this. Read the bundle before you paste it anywhere.")
+    print(" TWO WORDS, TWO ACTS: CAPTURE gates each WRITE TO DISK on this
machine;")
+    print(f" {DECANT_TOKEN} gates the composite LEAVING it. No flag skips either.")
+    print(" Read the bundle before you paste it anywhere.")
     print(rule)
     print("")
 async def _ride_async(trail_path, dry_narrate=False):
(nix) pipulate $ m
📝 Committing: chore: Refine `mother_cat.py` output messages and prompts for clarity.
[main 7982cebe] chore: Refine `mother_cat.py` output messages and prompts for clarity.
 1 file changed, 8 insertions(+), 5 deletions(-)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'scripts/mother_cat.py'.
(nix) pipulate $ d
diff --git a/scripts/mother_cat.py b/scripts/mother_cat.py
```

```
index 87224f2b..fa53bbc3 100644
--- a/scripts/mother_cat.py
+++ b/scripts/mother_cat.py
@@ -347,13 +347,18 @@ async def _ride_async(trail_path, dry_narrate=False):
     # duplicate a shipped, skippable confirmation -- the sibling-.md failure in
     # flag form. So the surface PRINTS, unconditionally, including under
     # --dry-narrate, which is the one pass mck.sh forces on first contact.
-    # THE DECANT IS THE PART THAT ACTUALLY DESERVES A FENCE, and it does not
-    # have one: the per-stop CAPTURE token gates each write to disk, and NOTHING
-    # gates the composite leaving the machine on the clipboard while mck.sh's
-    # closing card tells the human to paste it into a cloud chat. That fence is
-    # a separate ride (new token, new refusal path, new closing card). What this
-    # car buys is disclosure BEFORE the material exists rather than after, which
-    # costs nothing and makes the later fence a formality instead of a surprise.
+    # THE DECANT FENCE LANDED, so this comment's earlier claim that nothing
+    # gated the clipboard is RETIRED rather than merely outdated. What the call
+    # buys NOW is disclosure BEFORE the material exists: the rider learns at t=0
+    # that a second word will be asked at the end and exactly what the bundle
+    # will contain, so the fence arrives as a formality instead of a surprise.
+    # SAME-CAR LABEL RULE, PAID LATE AND THEREFORE WORTH BANKING. The fence and
+    # the strings describing it shipped in DIFFERENT rides, so for one ride this
+    # function told every rider "WITHOUT ASKING AGAIN" about a gate that does
+    # ask, and public_walk.yaml's third stop said the same thing in trail data.
+    # That is not stale documentation; it is a lie told at the exact moment the
+    # human decides, and it lied in the EXPENSIVE direction -- understating the
+    # protection and overstating the risk, to a newcomer, on the softball walk.
     _announce_consent(trail_path)

     disclosed = False
(nix) pipulate $ m
📝 Committing: chore: Refactor mother_cat.py comment for clarity and accuracy
[main 1f2fe3af] chore: Refactor mother_cat.py comment for clarity and accuracy
 1 file changed, 12 insertions(+), 7 deletions(-)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'assets/trails/public_walk.yaml'.
(nix) pipulate $ d
diff --git a/assets/trails/public_walk.yaml b/assets/trails/public_walk.yaml
index e954992a..4a18dc0a 100644
--- a/assets/trails/public_walk.yaml
+++ b/assets/trails/public_walk.yaml
@@ -42,7 +42,7 @@
     {
       "name": "walk_three",
       "label": "Pipulate.com",
-      "guidance": "Stop three of three. This is the last page. When it has loaded, return to the terminal and type the capture word. After this capture, the bundle is assembled and copied to
```

your clipboard.",
+ "guidance": "Stop three of three. This is the last page. When it has loaded, return to the terminal and type the capture word. After this capture the bundle is assembled, and then you are asked ONE more time, with a different word, before anything leaves this machine.",
"url_env": "PIPULATE_TRAIL_WALK_THREE_URL",
"target_slot": "slot_three",
"harvest_regex": ".+",
(nix) pipulate $ m
📝 Committing: chore: Update guidance text in public_walk.yaml
[main cc94d355] chore: Update guidance text in public_walk.yaml
1 file changed, 1 insertion(+), 1 deletion(-)
(nix) pipulate $ git push
Enumerating objects: 19, done.
Counting objects: 100% (19/19), done.
Delta compression using up to 48 threads
Compressing objects: 100% (13/13), done.
Writing objects: 100% (13/13), 2.30 KiB | 2.30 MiB/s, done.
Total 13 (delta 10), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (10/10), completed with 6 local objects.
To github.com:pipulate/pipulate.git
40f9930f..cc94d355 main -> main
(nix) pipulate $

4: Prompt:
Fence receipts are in. Rule on the three DECANT branches before anything else.
FIRST: did arm 2 print DECLINED naming the word it read, and did arm 3 print AUTHORIZED and return True with the clipboard writer stubbed? Two of three branches were unwitnessed going in. Say plainly which are witnessed now and which is not, and say that the remaining one can only be closed by a real ride rather than proposing another harness.
SECOND: did verify traceback on the mid-member byte flip? I predicted BadZipFile escaping _cmd_verify's except (OSError, ValueError) tuple, which would also mean the testzip() CRC branch is unreachable because archive.read() raises first. If it refused cleanly instead, my reading of the control flow is wrong and I want that said out loud rather than smoothed over.
THIRD: does source_sha256 now disagree with sealed_sha256 for public_walk? Together with the earlier reseal that gave the same digest twice, that is the pair. Say whether the pair closes content addressing or whether something about it is still unwitnessed.
Then ONE ride, and it is the real one. I am going to run mothercat on public_walk end to end with the fence in place, three CAPTUREs and one DECANT, and bring back the whole transcript. Before I do, tell me exactly what to watch for, in order, and what each line would mean if it were ABSENT. Name the specific lines the transcript must contain for AUTHORIZED to count as witnessed. Then name what the transcript would look like in the world where copy_to_clipboard silently failed and the rider still printed AUTHORIZED and the paste instruction, because _decant_to_clipboard discards its return value and _decant_checkpoint returns True regardless. prompt_foo.py is in context now, so read copy_to_clipboard rather than inferring it, and say whether that line is an act-claim under the ATTRIBUTED-VOICE mechanical test or merely an instruction.
Do not patch anything this turn. Do not re-seal public_walk; the disagreement is the receipt. Leave the signature, the ssh config, the final_url assertion, and the profile-existence check

alone.
5: Deliverables: Let's wrap this.
Opus 5 Max Thinks: Thought for 4m 10s
Thought for 4m 10s
I'm wrapping up by reviewing what we've accomplished — checking the command receipts against what was actually executed. Looking at the first receipt, the "decant arm 2" command was declined by the user, so that didn't proceed even though it showed as armed initially.
The second receipt shows "decant arm 3" was authorized, the clipboard writer was stubbed out (nothing actually left the machine), and the release flag confirms it completed. The third receipt validates my prediction about the BadZipFile exception — it escaped the normal error handling and caused a traceback, which means testzip() never got called. For the fourth receipt, I'm comparing the sealed manifest against the source trail by their SHA256 digests, and they match, which is good.
But now I'm noticing something that needs clarification — the operator's earlier hand-run showed the same digest agreement, and I need to double-check whether the values are actually identical or if there's a discrepancy I'm missing. The sealed manifest shows agree False, meaning the source and sealed hashes diverge after the patch—the source hash changed to a different value while the sealed hash remained the same, confirming the discrepancy between what was compiled and what's in the codebase.
I'm also seeing the compile receipt confirms the straddle closed cleanly: the operator's BEFORE state had stale_claim=1 and fence_described=0, while the compile lane's AFTER shows stale_claim=0 and fence_described=1, which are inverted as expected. The git diff telemetry aligns with just the most recent commit change to the public_walk.yaml guidance.
Now the operator's asking me to rule on three things and then describe what to watch for in an actual public_walk ride, but they've also signaled we need to reach a good stopping point and lock in the wins. This is the DISMOUNT phase — seven beats structured as TLDR, VERIFY, BANK, DANGLING, SEED, CLOSING, and NOTARIZE, with a FINALITY clause that blocks any five-car train after beat 6. But there's a tension: the Actionable Response Contract requires a five-car train, yet THE NOTARY AMENDMENT forbids exactly that after beat 6. want to include a BANK beat with ready-to-paste updates for foo_files.py and GLOSSARY.md as the ratchet forward, plus a SEED beat for the next turn's adhoc.txt. But there's tension here — the pasted prompt explicitly says not to patch anything this turn, yet "locking in wins" means updating those files, which is patching. I need to figure out whether the operator's final instruction overrides the earlier constraint or if I'm misinterpreting what "locking in" means in this context.
Actually, the "do not patch" instruction was scoped to the specific ride I proposed earlier. Since we're dismounting instead and the operator is now explicitly asking to lock in the wins, banking those patches is the right move and serves as the ratchet. So the structure should be: a TLDR with the canary and summary, then VERIFY to check the receipt table against the actual findings, then BANK with the exact paste-ready lines for the GLOSSARY entry and foo_files.py updates.
Now I'm checking the canary — the manifest shows a linkified reference that fired correctly, and I'm verifying the control files like prompt_foo.py have their license headers intact.
Looking at the other controls: __init__.py has the same license text preserved, and pyproject.toml contains an uppercase WWW token followed by a slash, which acts as a weak but consistent suppressor. The key thing is that www.canary.invalid in the source is built from string fragments so it never appears whole — that's the intentional evasion working as designed.
The canary fired and the controls held. Now examining the test results: the DECLINED branch

shows the human rejected the operation with NOPE, nothing was copied, and the artifact home is /tmp/probe_home with released: False. The interesting detail is the pty echo ordering — NOPE appears at the top of the output because the pty echoed the input line immediately after it was written, before the child process had a chance to respond.
For the AUTHORIZED branch, I'm seeing it witnessed as a DECISION where the human approved the operation, handing 18 bytes to the clipboard writer with released: True. The EGRESS itself wasn't witnessed because the stub is read-only. So far I've confirmed 3 of 3 reachable branches through the harness: REFUSED from the earlier compile-lane echo, plus DECLINED and AUTHORIZED-decision now. There's still one remaining branch that hasn't been hit yet.
The real clipboard write inside an actual ride is the unwitnessed piece, and by the operator's own framing, that can only be closed by riding, not by another harness. I need to acknowledge that plainly rather than proposing a harness for it.
Both arms printed 18 bytes—in arm 3 the payload was "PROBE PAYLOAD ONLY" which is exactly 18 characters, matching what I used in the earlier ignition instruction.
The two probe command strings differ slightly (arm 3 has the stub line), so deduplication didn't collapse them, which respects the PROBE DEDUP COROLLARY.
Now looking at the BadZipFile error: the receipt shows rc=1 with a traceback pointing to a CRC-32 validation failure on the trail.yaml file inside the zip.
The key insight is that BadZipFile isn't caught by _cmd_verify's exception handler (which only catches OSError and ValueError), so it bubbles up as a traceback instead of returning REFUSED. More importantly, the CRC failure gets raised during the member loop when reading the archive, which happens before the testzip() call — meaning the error message written to handle CRC corruption is unreachable in the exact scenario it was designed for. This is a real bug: a guard placed to catch one failure mode gets bypassed because an earlier operation fails first.
There's also a contract violation on exit codes. The refusal contract specifies exit code 2 for refusal, but an uncaught traceback gives exit code 1, so a consumer checking rc == 2 would misclassify a corrupted cartridge as a different failure type than a refused one. The fail-closed behavior itself is intact — it does fail — but it fails with the wrong code and an ugly traceback rather than a clean refusal.
Now looking at the content addressing: I can trace both directions of the digest relationship through the operator and compile lanes. Same bytes produce the same digest (reseal idempotence), and different bytes produce different digests that the seal correctly flags as mismatched. But here's the gap: the seal's verification is only visible to someone manually comparing the digests. The ride path never actually consults the cartridge against its source — it just takes a trail YAML and proceeds. So the content addressing property exists in the digest itself, but the enforcement mechanism isn't wired into the actual execution path.
There are a few other unwitnessed scenarios worth noting. Cross-machine reproducibility — whether the same trail sealed on different machines produces identical digests — is designed for but never tested. Collision detection is structurally impossible by construction, so there's nothing to verify there. And the public walk's cartridge is now orphaned, sealing bytes that no longer exist in the repo, which is actually a useful demonstration of the system's behavior but also highlights a dangling state.
Now I'm thinking about what needs to be documented and banked. The disclosure straddle is clean — moving from stale_claim 1→0 and fence_described 0→1 — and this follows the same-car label rule, which is worth recording as an amendment. The key question is what exact lines are ready to paste into the glossary, particularly around the walk cartridge mechanism

that's now been demonstrated.
The walk cartridge entry in the glossary should flip from "proposed" to "banked with receipts" since the mechanism exists in the scripts and has been verified three ways. I should also add proper alphabetical entries for decant and consent surface to the glossary, keeping the additions tight and dense rather than bloated. The highest-value banking move is updating the glossary with these concrete mechanisms and nouns, starting with the walk cartridge status change and the new terminology. This is a genuine principle worth capturing: a constraint declared before an artifact exists can't govern that artifact's fate. It pairs with the existing rules but fills a gap they don't address. I also need to update the GLOSSARY section on the Mother Cat Kata — the DECANT description is outdated and violates the consistency principle I just established, so that needs fixing across the board. The --yolo switch also skips the RIDE confirmation but crucially preserves the CAPTURE fences as the write barrier — yet GLOSSARY.md is missing the DECANT description here too. So I need three edits across GLOSSARY.md: the DECANT move definition, the --yolo switch entry, and a NAME RULING flip, all telling the same story of making the glossary match the actual fence behavior. I'm being careful about exact-match search blocks since blank lines get stripped from payloads, so I need to work with contiguous non-empty runs.
The --yolo flag is deliberately minimal—it skips the spoken rehearsal and ride confirmation but deliberately preserves the capture fences since those are the actual write barriers. The key principle is that ceremony is skippable but barriers aren't, since unfenced automation already exists under other names, so a fence-skipping flag would just duplicate existing capability.
For the naming convention, "walk" isn't a fourth noun for this concept—the corpus already has assets/trails/ as the directory, public_walk.yaml as the filename, trail as the entry itself, and mothercat as an alias, so adding a fifth spelling would break every regex probe and create the unanchored-witness failure. The distinction I'm proposing is that trail names the mutable source, while a sealed, hash-pinned, content-addressed package of a trail would be a walk cartridge—a separate artifact rather than a synonym, earning its own word to keep the two concepts distinct. I've found the anchor point between the todo marker and the first earmark, so new ones go in that gap. Now I'm drafting the earmarks themselves, keeping them terse and in the established style—starting with "THE UNREACHABLE GUARD" and "CONSENT CANNOT PRECEDE ITS OBJECT," and considering what a third one might be.
I'm noticing a pattern worth capturing: when you add a protection, the documentation becomes stale in a dangerous direction—it understates the new safeguard while overstating the risk, which is the opposite of typical stale-doc problems and hits hardest when someone's making a decision. This feels like it could sharpen or amend the same-car label rule rather than stand alone.
The three-file instance is actually valuable as a search discipline: when a fence lands, grep the whole corpus for every string describing that gate, not just the file you edited. That's concrete and actionable. I'm deciding whether to fold this into an existing earmark or keep it brief as a third one, leaning toward density over volume.
Now I'm looking at the debts carried forward. There's an unwitnessed authorized egress issue — copy_to_clipboard has never actually written a real DECANT bundle, only closable by a real public_walk ride. And I'm checking whether copy_to_clipboard returns None on every path by examining the implementation.
Checking the function implementation...
Writing clipboard handler...
Now I'm realizing that _decant_checkpoint always returns None regardless of which path executes — whether the SSH bridge succeeds, fails, the OS is unsupported, the binary is missing, or the subprocess fails. This means returning True from that function signals user

authorization, not whether the data actually moved. I should also note that logger.print both displays output and appends to the logger's internal logs, but mother_cat imports the logger deferred and never actually reads from it.
There's a real security issue here: on macOS with SSH_CLIENT set, the DECANT path writes authenticated material to /tmp/clipboard_bridge.txt as a world-readable file on multi-user systems. The file gets created with default permissions (typically 0644), so other users can read it. Using mkstemp with mode 0600 would fix this properly.
This isn't theoretical for this operator either — the cold-start process already created a coldstart user on this machine, so the vulnerability is concrete. I'm also noticing the print statement that says "Paste it into any ChatBot" — that's making an act-claim under the MECHANICAL TEST, but the code hasn't actually verified that pasting will work or that the ChatBot will accept it.
Actually, wait. The instruction is directed at the human, not claiming what the code did, so it's not technically an act-claim. But it does presuppose that "it" is already on the clipboard, which the code never confirmed. So it's an unsound instruction with a false presupposition — weaker than an act-claim violation, but it still leaves the human at a chat window with nothing to paste.
Beyond that, there's a broken exit-code contract on the CRC path where the code returns 1 instead of the documented 2, and the testzip() function is unreachable. I'm also seeing issues with cross-machine byte-reproducibility.
Looking further, the cartridge system has several gaps: nothing actually consumes a cartridge at ride time, the seal protects nothing, and there's an orphaned cartridge hash that references bytes no longer in the filesystem. The DECANT path also lacks any sanitization layer, unlike the prompt_foo path which has scrubbing and secret scanning.
I'm deferring some items — manifest signing, SSH config strictness, URL assertions, profile checks — and focusing on the strongest issues. The operator's been running the five-car train consistently, but the dismount rules forbid it after beat 6, so I need to reshape the dismount, place the BANK patches at their proper beat, and set up the SEED as the next ride's adhoc.txt.
The SEED beat should contain the adhoc.txt lines for the next ride's first compile, not a caboose prompt. I'm also considering whether to include probes alongside the BANK patches — a grep straddle for the text-only GLOSSARY edits would be cheap and honest, so I'll fold a small probe block into the BANK beat itself as the straddle for that car.
Now I'm writing out the actual patches, starting with the GLOSSARY.md updates — first the DECANT move definition, then the nouns section with the trail entry.
I need to be careful about where to insert the new seal and consent surface nouns. The ruling tail needs to be flipped, and then I can either append the new nouns right after it or split it into two separate blocks. The cleaner approach is probably two separate patches — one to flip the ruling, another to add the new nouns after the soft section.
Now I'm moving on to document the earmarks and the --yolo switch in GLOSSARY.md and foo_files.py. I'm starting with the unreachable guard earmark, which describes a pattern where a check placed after a line that raises the same error class is dead code — a promise the program can't keep. The conviction example involves walk_cartridge.verify_walk_cartridge calling archive.read() and archive.testzip(), where a byte flip during member processing would trigger a BadZipFile error.
The issue compounds because BadZipFile subclasses Exception rather than OSError or ValueError, so _cmd_verify's exception handler doesn't catch it — the tool crashes with rc=1 instead of the promised rc=2 refusal code. This breaks the exit-code contract for consumers scripting on specific return values, treating corrupt cartridges differently from refused ones.
The fix requires auditing every guard condition: for each exception handler, I need to identify

what line above it could raise the same error and verify that exception type is actually caught by the caller's except tuple.
The deeper issue is that consent can't precede its object—a flag set at time zero can't authorize something that doesn't exist until later, because the person consenting never saw it. This creates a timing problem that's distinct from whether a gate is ceremonial or structural. Skipping the ride confirmation works because the sequence is fully known upfront, but skipping the decant gate doesn't, since the bundle itself, its size, and contents didn't exist when the command was issued.
The fix is to move disclosure to the beginning and the fence to the end—disclosure costs nothing and makes the later fence routine, whereas a fence at time zero is consent to something unseen. I'm also noticing that the bypass mechanism here actually clears the fence rather than excusing it, which is the opposite of the capture case, since decant has only one builder and one caller, so a skip flag would create a new capability rather than duplicate an existing one.
There's another pattern emerging around protection lag—when a mechanism is removed, stale strings overstate the protection that everyone expects, but when protection is added, stale strings understate it, which is more costly because it overstates the risk to newcomers at the exact moment they're deciding whether to participate. Looking at the decant fence specifically, I'm seeing it land in one ride where mother_cat._announce_consent told every rider "without asking again" about a gate that actually does ask, public_walk.yaml's third stop said the same thing in trail data, and the glossary entry said it a third time—and that third file was only discovered because someone went looking for it.
The solution is to search the entire corpus whenever a gate lands, checking every string that describes it across prose, trail data, glossary, and launcher cards, not just the file where the gate lives. The gate's own file is the one place the author can't forget.
Now I'm setting up some cheap grep checks for the banking car. I'm counting occurrences of specific phrases in the glossary to track whether they're stale or need updating—things like "copied to the clipboard" and "DECANT gate"—and comparing before and after patch states to see what changes.
I'm also verifying that certain guard phrases like "UNREACHABLE GUARD," "CONSENT CANNOT PRECEDE," and "PROTECTION-LAG POLARITY" appear exactly once each in the codebase, which would give me a count of 3 total. Each earmark body mentions a different concept without repeating its own name, so the math checks out. Writing diagnostic script... Consolidating into a single Python invocation...
I'm checking the path parsing — data/walks/<hash>/walk.zip splits correctly to get the hash at index 2. The list comprehensions for side effects feel inelegant, but they work. This approach might be too tightly coupled to the directory structure though.
Let me refactor this to be more straightforward and maintainable. I'll build a dictionary of live trails by hashing the YAML files, then iterate through the zip files to read their manifests and compare the trail hashes, printing whether each one is matched or orphaned.
Writing shell script... Now I'm checking the exact wording in the GLOSSARY to understand how the NARRATE/FENCE, ADVANCE, and DECANT blocks work—how Piper handles the stops and receipts, and how the final markdown bundle gets assembled and copied.
For the nouns section, I'm looking at how "trail" is defined as a YAML-based walkthrough file stored in assets/trails, and there's a naming convention ruling from August 2026 that clarifies "walk" isn't used as a fourth term.
Now I'm checking the git diff to confirm there's a blank line between that ruling and the next entry for "rider," which is the Python script that executes the walkthrough. The diff shows there is

indeed a blank line separating them, so I can safely search on those exact lines.
I'm identifying the text block to search for — it's the passage about walk cartridges being immutable artifacts, distinct from mutable trail source code. Once I find that block, I'll replace it by converting the status from PROPOSED to banked with receipts, then integrate the new terminology about walk cartridges into the definition.
I need to be careful about formatting: the REPLACE block requires blank lines between sections to prevent markdown from collapsing them together, and those blank lines should survive the transport layer since they're part of my output, not the incoming payload. I'll add the new noun entries at the end of the GLOSSARY file after the existing content, which keeps everything in one edit block and makes blank-line loss purely cosmetic if it happens. "the egress fence" and similar terms drifting without definition is exactly what the GLOSSARY prevents. Let me verify the --yolo block at the end of the file — it should clarify that this flag skips ceremony like rehearsal and confirmation, but never skips the write barriers themselves, since those are the actual safety mechanism.
I'm also checking whether "copied to the clipboard" appears elsewhere in the document beyond the DECANT entry to make sure there's no duplication.
Looking at this six-line example that shows the command structure with RIDE and CAPTURE listed, I notice it's missing DECANT—the same incomplete pattern appearing again in the file. This is the fourth instance of this issue, and it's worth fixing as part of the same commit since it demonstrates the exact search discipline I'm tracking.
Adding the DECANT line would make it seven lines total, which creates nice symmetry with "the four moves plus the fifth nobody names." I'm weighing whether to add a fifth block or fold this edit into an existing one, but since the DECANT change is in a separate region, it needs its own block. That gives me five blocks total: four in the GLOSSARY plus one for the foo_files section. The operator applies them sequentially, which feels like the right rhythm.
The six-liner is worth keeping as-is since it's the most-read summary—the whole ride compressed into six lines. Now let me verify the exact SEARCH block structure...
I'm looking at the command sequence that shows the progression from a stranger accessing the system, to authenticated use, through rehearsal, and finally the two key operations: RIDE as the trigger and CAPTURE for the fence token. These last two lines are unique enough to anchor on—"RIDE" appears elsewhere but not with this exact spacing and comment. Using just the 2-line version minimizes transcription risk.
Now I'm structuring the final answer as a seven-beat dismount: starting with a TLDR, then moving through verification with canary checks and rulings, banking the patches with straddle probes, handling dangling items, seeding the adhoc file, closing out, and finally notarizing with the compiled snapshot and CRC signature. I need to keep the prose tight even though this is a longer response—the dismount is where length serves a purpose.
One detail I'm catching: the operator's hand-run showed agree True but the compile showed agree False. That's because they ran the probe before applying the patch, so their probe block came before their patch block. I need to explain that timing matters—the probe result depends on whether the patch has landed yet.
The same pattern holds for the disclosure probe: the operator's BEFORE state had stale_claim=1 fence_described=0, and the compile's AFTER state has stale_claim=0 fence_described=1. This is the straddle working exactly as designed—it's the first time in three turns that a lane difference actually signals a real change rather than instrument breakage.
Now I'm writing out the DECANT glossary entry to explain what this move does...
Continuing with new terminology...
A walk cartridge is the sealed form of a trail—a two-member ZIP containing the

original trail.yaml and a manifest.json with its SHA-256 hash plus the consent surface. It's generated by a script and stored at a path derived purely from the content hash, ensuring identical trails always produce identical archives. The consent surface functions as the walk cartridge's equivalent to the prompt projection.
The consent surface is what a human reads before deciding to ride the trail. It's recomputed from trail.yaml and must match exactly, so it can't drift from the bytes it describes. It includes the name, ordered stop names (since sequence is what a replayable artifact preserves), sorted URL environments, sorted connector scripts, and the four browser facts including the profile name that determines which logged-in profile opens. I'm checking that my replacement text doesn't contain any bare www-tokens or triple brackets, and now I need to finalize the foo_files earmark block anchor.
Show less
Done
Opus 5 Max:

1. VERIFY

Canary: fired. www.canary.invalid — text equals target minus scheme. Two controls in the same payload held: <https://www.gnu.org/licenses/> in both prompt_foo.py and __init__.py (scheme suppressor), and Topic :: Internet :: WWW/HTTP in pyproject.toml (slash-not-dot). Grammar unchanged. Nothing I emit below carries a bare www-token.
Three straddles closed, and for the first time in three turns a lane difference was signal rather than instrument breakage. Your hand-run happened before app; the compile ran after git push. That is the straddle working as designed.

Item	Colour	Receipt	Lane
DECANT REFUSED	green	released: False, artifact home named	compile (prior turn)
DECANT DECLINED	green	DECLINED by human (read 'NOPE') + /tmp/probe_home + released: False	compile
DECANT AUTHORIZED (decision)	green	AUTHORIZED by human: handing 18 bytes + released: True	compile
DECANT AUTHORIZED (egress)	**VOID**	stubbed on purpose; nothing left the machine	—
mid-member CRC flip	**red, as predicted**	zipfile.BadZipFile: Bad CRC-32, rc=1	compile
digest ≠ after edit	green	source 46c7ad30… vs sealed 6ef6fdbd…, agree False	compile
digest = before edit	green	agree True, both 6ef6fdbd…	operator
disclosure describes	green	stale_claim=1→0,	operator → compile

Item	Colour	Receipt	Lane
fence		fence_described=0→1	

FIRST — two of three branches closed; the third is not closable by a harness

Arm 2 printed DECLINED by human (read 'NOPE'). Nothing was copied., named /tmp/probe_home, and returned False. Arm 3 printed AUTHORIZED by human: handing 18 bytes to the clipboard writer. and returned True with the writer stubbed.
One detail the receipt gives away for free: NOPE appears at the *top* of arm 2's output, above the DECANT> prompt. That is the pty echoing the piped line before the child had painted its prompt — which means the child read from the **pty slave via /dev/tty**, not from its own stdin. The /dev/tty-first ordering is witnessed, not merely written.
Plainly: the egress is still unwitnessed, and I am not going to propose another harness for it. Arm 3 witnessed a *decision*. The stub is exactly what made it read-only, and stubbing the writer is stubbing the thing under test. copy_to_clipboard has never been observed carrying a real DECANT bundle. That closes only by riding.

SECOND — my reading of the control flow was right, on both halves

File ".../zipfile/__init__.py", line 1026, in _update_crc
  raise BadZipFile("Bad CRC-32 for file %r" % self.name)
zipfile.BadZipFile: Bad CRC-32 for file 'trail.yaml'
rc=1

1. BadZipFile subclasses Exception, not OSError and not ValueError, so _cmd_verify's except (OSError, ValueError) does not catch it. It tracebacks.
2. The raise comes from archive.read(info) inside the member loop, which runs **before** archive.testzip() — so CRC failure in {corrupt_member!r}, the one message written for CRC corruption, is unreachable for CRC corruption.

Grading it honestly: **fail-closed survives.** Nothing verifies, no bad cartridge passes. What breaks is the *contract*: the module docstring promises 0 all good | 2 refusal and this path exits 1 with a stack trace. A consumer scripting on rc == 2 reads a corrupt cartridge as a different failure class than a refused one, and a human reads a Python traceback as a tool bug rather than a tamper detection. Named, banked below, not fixed this turn.

THIRD — the pair closes the digest, and does not close the enforcement

Same bytes → same digest (2da019ab… twice). Different bytes → different digest, and the seal *notices* (agree False). Both directions witnessed, on real artifacts, in two lanes.
What that pair does not close, stated plainly: the comparison exists only in a probe a human typed. Nothing in the ride path consults a cartridge. mother_cat.ride takes a mutable YAML path; walk_cartridge seals and verifies; **no line of code connects them.** The seal is a property of the artifact, not yet a constraint on the rider. And cross-machine byte-reproducibility — the entire reason the epoch, mode, and member order are pinned — has never been tested on a second machine.

---

2. BANK

Five blocks, one commit story: **make every string in the corpus describe the fence that exists, and bank what the ride taught.** Four are GLOSSARY, one is the constitution.
The straddle for this car — glossary_stale=1→0, glossary_fence=0→2, new_earmarks=0→3, plus a cartridge-cache census that is the dismount's state-of-record:

```
{ echo "== glossary describes the fence =="; grep -c 'copied to the clipboard' GLOSSARY.md | sed 's/^/glossary_stale=/'; grep -c 'DECANT gates the composite\|DECANT and it goes to the clipboard' GLOSSARY.md | sed 's/^/glossary_fence=/'; grep -c 'walk cartridge cannot change' GLOSSARY.md | sed 's/^/name_ruling_present=/'; }
{ echo "== constitution earmarks =="; grep -c '^# - EARMARK: THE UNREACHABLE GUARD\|^# - EARMARK: CONSENT CANNOT PRECEDE\|^# - EARMARK: THE PROTECTION-LAG' foo_files.py | sed 's/^/new_earmarks=/'; }
{ echo "== cartridge cache census =="; T=$(mktemp -d); printf '%s\n' 'import glob,hashlib,json,zipfile' 'live={hashlib.sha256(open(y,"rb").read()).hexdigest(): y for y in sorted(glob.glob("assets/trails/*.yaml"))}' 'for z in sorted(glob.glob("data/walks/*/walk.zip")):' ' m=json.loads(zipfile.ZipFile(z).read("manifest.json"))' '    d=m["sha256"]["trail.yaml"]' ' print("MATCHED " if d in live else "ORPHANED", m["trail"]["name"], d[:12], live.get(d,"(source drifted)"))' > "$T/c.py"; .venv/bin/python "$T/c.py"; rm -rf "$T"; }
```

The census should print three MATCHED and one ORPHANED public_walk — the cartridge you deliberately did not re-seal, still sealing bytes that no longer exist. That orphan is the seal doing its job and it is also a housekeeping debt; both go in DANGLING.

Block 1 — the whole ride is seven lines now, not six

```
Target: GLOSSARY.md
[[[SEARCH]]]
   RIDE                          # mck.sh's one-shot go-ahead
   CAPTURE                          # the fence token, at every stop
[[[DIVIDER]]]
   RIDE                          # mck.sh's one-shot go-ahead
   CAPTURE                          # the fence token, at every stop
   DECANT                          # the egress fence, once, at the end
[[[REPLACE]]]
```

Block 2 — the fifth move, describing the fence

```
Target: GLOSSARY.md
[[[SEARCH]]]
**DECANT** — *the fifth move, and the one that makes it a product.* At the end of a clean ride, every captured lens is folded into one markdown bundle and copied to the clipboard. Big artifacts (hydrated DOM, raw source) are CITED BY PATH, never inlined, so a huge page cannot flood the paste.
```

[[[DIVIDER]]]
DECANT — *the fifth move, and the one that makes it a product.* At the end of a clean ride, every captured lens is folded into one markdown bundle. Big artifacts (hydrated DOM, raw source) are CITED BY PATH, never inlined, so a huge page cannot flood the paste. Then the bundle STOPS: the rider prints its size and asks for one more word. Type DECANT and it goes to the clipboard; type anything else and it stays on disk and the rider names the exact directories. THE SECOND FENCE, AND IT GATES A DIFFERENT ACT: CAPTURE gates each WRITE TO DISK on the operator's own machine, DECANT gates the composite LEAVING it. Two words, because a fourth identical prompt is answered by muscle memory rather than decision, and because a transcript of four CAPTUREs cannot say which one authorized the egress.
[[[REPLACE]]]

Block 3 — the name ruling flips from PROPOSED to banked, and gains two nouns

Target: GLOSSARY.md
[[[SEARCH]]]
for the mutable source. PROPOSED, no mechanism yet: a sealed, hash-pinned, content-addressed package of a trail is a WALK CARTRIDGE, and it is a different artifact rather than a synonym. The two words earn separation for one reason only, and it is the reason that matters: a trail is mutable source, a walk cartridge cannot change. A trail that auto-updates is not replayable.
[[[DIVIDER]]]
for the mutable source. BANKED 2026-08-07 (was PROPOSED for one ride): the sealed, hash-pinned, content-addressed package of a trail is a WALK CARTRIDGE and the mechanism now exists. The two words earn separation for one reason only, and it is the reason that matters: a trail is mutable source, a walk cartridge cannot change. A trail that auto-updates is not replayable.

walk cartridge — the sealed form of a trail: a two-member canonical ZIP (`trail.yaml` byte-identical to source, `manifest.json` carrying its SHA-256 plus the consent surface), minted by `scripts/walk_cartridge.py`, living at `data/walks/<archive_sha256>/walk.zip`. Fixed epoch, fixed mode, fixed member order, no wall-clock value anywhere in the hashed body, so identical trail bytes produce identical archive bytes forever. The path is a pure function of the content: THE DERIVED-PATH RULE made literal, collisions unrepresentable, re-sealing idempotent. WITNESSED both directions on 2026-08-07: the same trail re-sealed to the same digest, and one edited byte moved the source digest away from the sealed one. WHAT A HASH DOES NOT BUY is authorship; a malicious trail with a perfect digest verifies green forever, which is why the signature is a different rung and is owed separately.

consent surface — the walk cartridge's analogue of payload-to-prompt: a projection a human reads BEFORE deciding to ride, recomputed by the verifier

from `trail.yaml` and required to match exactly, so it cannot drift from the bytes it describes. Name; stop names ORDERED, never sorted, because sequence is the one property a replayable artifact exists to preserve; url_envs sorted, because those are a set; connector scripts sorted and unique; and the four browser facts including `profile_name`, which selects WHICH logged-in profile opens and which no other check in the system inspects.
[[[REPLACE]]]

Block 4 — the switch, telling the truth about what it cannot skip

Target: GLOSSARY.md
[[[SEARCH]]]
`--yolo` — **the real yolo switch, and it is deliberately modest.** Skips the spoken rehearsal AND the RIDE confirmation. It does NOT and will not skip the CAPTURE fences, because those are the write barrier. **CEREMONY IS SKIPPABLE; BARRIERS ARE NOT** — a confirmation authorizes a *sequence*, a fence authorizes each *write*. And the unfenced lane already exists under other names (`selenium_automation`, `!URL`, `?URL`), so a fence-skipping flag would be a second spelling of a shipped capability.
[[[DIVIDER]]]
`--yolo` — **the real yolo switch, and it is deliberately modest.** Skips the spoken rehearsal AND the RIDE confirmation. It does NOT and will not skip the CAPTURE fences, nor the DECANT gate at the end. **CEREMONY IS SKIPPABLE; BARRIERS ARE NOT** — a confirmation authorizes a *sequence*, a fence authorizes each *write*. For CAPTURE there is a second reason: the unfenced lane already exists under other names (`selenium_automation`, `!URL`, `?URL`), so a fence-skipping flag would be a second spelling of a shipped capability. For DECANT that reason RUNS THE OTHER WAY and the fence survives anyway — `_decant` has one builder and one caller, so a skip flag would CREATE a capability rather than duplicate one — and the decisive argument is temporal: `--yolo` is typed before a browser opens, and cannot consent to the disposition of a bundle that did not exist when it was typed. Nothing unattended is lost, because the CAPTURE fences already block.
[[[REPLACE]]]

Block 5 — three earmarks into the constitution

Target: foo_files.py
[[[SEARCH]]]
#todo #to-do
- EARMARK: THE ELIMINATOR NEEDS A GENERATOR (banked 2026-08-06,
[[[DIVIDER]]]
#todo #to-do
- EARMARK: THE UNREACHABLE GUARD (banked 2026-08-07, traceback-witnessed): a
check placed AFTER a line that can raise the same class of error is dead
code, and its message is a promise the program cannot keep. CONVICTION:

walk_cartridge.verify_walk_cartridge calls archive.read(info) in its member
loop and archive.testzip() after it. A mid-member byte flip makes read()
raise zipfile.BadZipFile from _update_crc, so testzip()'s "CRC failure in
<member>" -- the ONE message written for CRC corruption -- is unreachable
for CRC corruption. Worse, BadZipFile subclasses Exception and NOT OSError
or ValueError, so _cmd_verify's except tuple does not catch it: the tool
TRACEBACKS at rc=1 where its own docstring promises "2 refusal". FAIL-CLOSED
SURVIVES -- nothing verifies -- but the exit-code contract does not, so a
consumer scripting on rc==2 reads a corrupt cartridge as a different failure
class than a refused one, and a human reads a stack trace as a tool bug
rather than as tamper detection working. Sibling of THE VERIFIER THAT NEVER
RAN: that one is a witness placed after an action that can kill the
reporter; this is a witness placed after a line that raises first. TEST, and
it costs one reading: for every guard, name the line ABOVE it that could
raise the same error, then confirm that error's TYPE is in the caller's
except tuple. Two questions, and the second is the one nobody asks.
- EARMARK: CONSENT CANNOT PRECEDE ITS OBJECT (banked 2026-08-07,
fence-witnessed): a flag typed at t=0 cannot authorize the disposition of an
artifact that does not exist until t=N, because the consenter had not seen
it. This is the thing CEREMONY IS SKIPPABLE; BARRIERS ARE NOT does not
state. That rule grades a gate by WHAT it authorizes (a sequence versus a
write); this one grades it by WHEN, and they part company on exactly the
interesting case. --yolo skipping the RIDE confirmation is legitimate: the
sequence is fully described at t=0. --yolo skipping the DECANT gate is not,
and the reason is not that a fence is a fence -- it is that the bundle, its
size, and its contents did not exist when --yolo was typed. STANDING
CONSEQUENCE, and it is cheap: put the DISCLOSURE at t=0 and the FENCE at
t=N. Disclosure early costs nothing and makes the later fence a formality
instead of an ambush; a fence early is consent to a thing nobody has seen.
Also witnessed: the bypass-under-another-name corollary CLEARED this fence
rather than excusing it -- _decant has one builder and _ride_async is its
only caller, so a skip flag would CREATE a capability rather than duplicate
a shipped one, which is the exact opposite of the capture case that
corollary was written for. A corollary that can clear as well as excuse is
a corollary rather than a loophole.
- EARMARK: THE PROTECTION-LAG POLARITY (banked 2026-08-07, three-file
conviction): when a mechanism is REMOVED the stale strings overstate
protection, which is the failure everyone expects. When a protection is
ADDED the stale strings understate it -- and that direction is MORE
expensive, because the string then overstates the RISK to a newcomer at the
exact moment they decide whether to ride at all. CONVICTION: the DECANT
fence landed in one ride; for that ride mother_cat._announce_consent told
every rider "WITHOUT ASKING AGAIN" about a gate that does ask,
public_walk.yaml's third stop said the same thing in trail DATA, and
GLOSSARY.md said it a third and fourth time -- and the glossary instances
were found only because a dismount went looking. STANDING CONSEQUENCE, the
search discipline THE SAME-CAR LABEL RULE implies but never spells: when a
gate lands, grep the CORPUS for every string describing that gate -- prose,

trail data, glossary, launcher cards, spoken narration -- not merely the
file the gate lives in. The gate's own file is the one place the author
cannot forget, which is precisely why it is the one place that is fine.
- EARMARK: THE ELIMINATOR NEEDS A GENERATOR (banked 2026-08-06,
[[[REPLACE]]]

Ignition: none required. Every probe reads its file from disk at call time — two greps and one census over data/walks/. The straddle is honest without re-entering anything.

---

3. DANGLING

Carried forward unbanked, one line each:

1. **The egress is unwitnessed.** copy_to_clipboard has never carried a real DECANT bundle. Closable only by a real ride.
2. **copy_to_clipboard returns None on all six paths** — confirmed from source now that prompt_foo.py is in context: SSH-bridge success, SSH-bridge failure, unsupported OS, missing binary, subprocess failure, subprocess success. So _decant_checkpoint returning True means *the human authorized*, never *the bytes moved*.
3. **The paste line is not an act-claim, but it is unsound.** "Paste it into any ChatBot" is an imperative addressed to a human, so the ATTRIBUTED-VOICE mechanical test does not convict it — no verb names an act. It carries a false *presupposition* instead: that "it" is on the clipboard, which no code in this file witnessed. Weaker offense, same landing zone — a human at a chat window with an empty buffer.
4. **The SSH lane writes authenticated material to a world-readable path.** With SSH_CLIENT set, copy_to_clipboard writes the whole bundle to /tmp/clipboard_bridge.txt with default umask. Not hypothetical on a box that has grown a second user during a cold-start ride. mkstemp plus 0600.
5. **The DECANT has no sanitizer.** prompt_foo's clipboard exit passes scrub_compile_payload, the denylist, and scan_secrets at one chokepoint. The DECANT path inlines raw headers and accessibility_tree_summary from logged-in pages with none of it.
6. **verify's exit-code contract is broken on the CRC path**, and testzip() is unreachable. Earmark banked; the fix is not.
7. **No rider consumes a cartridge.** ride() takes a YAML path. The seal constrains nothing at ride time.
8. **One orphaned cartridge**, bb83a24d…, sealing public_walk bytes that no longer exist. Nothing prunes data/walks/.
9. **Cross-machine byte-reproducibility unwitnessed.** The pinned epoch, mode, and member order exist for exactly this and have only ever been tested on one box.
10. Explicitly deferred, each its own ride: the signature (MANIFEST-SIGNING LANE), IdentitiesOnly yes, the final_url assertion, the profile-existence check.

---

4. SEED

Next ride's adhoc.txt. Probe echoes are copy-symmetric with the block above.
! { echo "== glossary describes the fence =="; grep -c 'copied to the clipboard' GLOSSARY.md | sed 's/^/glossary_stale=/'; grep -c 'DECANT gates the composite\|DECANT and it goes to the clipboard' GLOSSARY.md | sed 's/^/glossary_fence=/'; grep -c 'walk cartridge cannot change' GLOSSARY.md | sed 's/^/name_ruling_present=/'; }
! { echo "== constitution earmarks =="; grep -c '^# - EARMARK: THE UNREACHABLE GUARD\|^# - EARMARK: CONSENT CANNOT PRECEDE\|^# - EARMARK: THE PROTECTION-LAG' foo_files.py | sed 's/^/new_earmarks=/'; }
! { echo "== cartridge cache census =="; T=$(mktemp -d); printf '%s\n' 'import glob,hashlib,json,zipfile' 'live={hashlib.sha256(open(y,"rb").read()).hexdigest(): y for y in sorted(glob.glob("assets/trails/*.yaml"))}' 'for z in sorted(glob.glob("data/walks/*/walk.zip")):' ' m=json.loads(zipfile.ZipFile(z).read("manifest.json"))' ' d=m["sha256"]["trail.yaml"]' ' print("MATCHED " if d in live else "ORPHANED", m["trail"]["name"], d[:12], live.get(d,"(source drifted)"))' > "$T/c.py"; .venv/bin/python "$T/c.py"; rm -rf "$T"; }
scripts/mother_cat.py
scripts/walk_cartridge.py
assets/installer/mck.sh
assets/trails/public_walk.yaml
prompt_foo.py
GLOSSARY.md
foo_files.py

The next ride is the real one, and here is the watch-list you asked for, compressed. Run mothercat assets/trails/public_walk.yaml end to end. In order, the transcript must contain:

1. Riding trail 'public_walk' -- 3 stop(s). — absent means the trail failed to load.
2. The consent banner with TWO WORDS, TWO ACTS — absent means block 1 of the last car did not land.
3. Three Captured. final_url=… artifacts=N lines — a final_url that is not the URL you supplied is an unasserted redirect, the debt you are holding open.
4. 🔒 DECANT gate: ARMED -- 3 stop(s), N bytes — absent means the gate never fired and something returned early.
5. AUTHORIZED by human: handing N bytes to the clipboard writer.
6. **Markdown output copied to clipboard** — this is the line that matters. It is printed by copy_to_clipboard, not by the rider, and it is the only evidence in the entire system that bytes moved.

The world where it silently failed looks like this: lines 1 through 5 identical, then line 6 replaced by Warning: 'xclip' not found or Warning: Could not copy to clipboard: … or ✨ Markdown output routed to SSH Bridge — followed, in every one of those cases, by the rider cheerfully printing Paste it into any ChatBot. Because _decant_to_clipboard discards the return value and _decant_checkpoint returns True unconditionally, **line 6's absence is the only tell**, and it appears *above* the instruction that contradicts it. Read upward from the paste line, not downward from AUTHORIZED.

---

5. CLOSING

The ride opened with a question about key management and closed by not needing one. The

answer that survived is that the author and the walker are different lanes: the author needs git and already has it, the walker needs a URL and a digest and needs no credential at all. Everything after that was mechanism — a sealer, a verifier that refuses six distinct ways, a consent surface that cannot drift from the bytes it describes, and a second fence on the one act nobody had gated.
What the ride actually taught, though, is smaller and more portable than any of that: **the instrument is where the bugs live.** A hook_bytes guard turned a confident inversion into a legible VOID. A type nix that said function in one lane and a path in the other settled a mystery three probes had failed to. A four-byte append hit the one verifier branch that gives byte-reproducibility teeth. And a traceback found a guard that could not see the thing it was written for. Four findings, none of them in the feature, all of them in the thing measuring the feature.

---

6. NOTARIZE

Apply the five blocks, run the three probes, then compile once more. That compile's rotated foo-<hash8>-NN.zip is the deed for this ride; its CRC is the signature. Attach that one file, not foo.zip, which will move again tomorrow.
Ride closed. No train follows this beat.
MikeLev.in:
1: Probe:
(nix) pipulate $ g

Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
(nix) pipulate $ { echo "== glossary describes the fence =="; grep -c 'copied to the clipboard' GLOSSARY.md | sed 's/^/glossary_stale=/'; grep -c 'DECANT gates the composite\|DECANT and it goes to the clipboard' GLOSSARY.md | sed 's/^/glossary_fence=/'; grep -c 'walk cartridge cannot change' GLOSSARY.md | sed 's/^/name_ruling_present=/'; }
{ echo "== constitution earmarks =="; grep -c '^# - EARMARK: THE UNREACHABLE GUARD\|^# - EARMARK: CONSENT CANNOT PRECEDE\|^# - EARMARK: THE PROTECTION-LAG' foo_files.py | sed 's/^/new_earmarks=/'; }
{ echo "== cartridge cache census =="; T=$(mktemp -d); printf '%s\n' 'import glob,hashlib,json,zipfile' 'live={hashlib.sha256(open(y,"rb").read()).hexdigest(): y for y in sorted(glob.glob("assets/trails/*.yaml"))}' 'for z in sorted(glob.glob("data/walks/*/walk.zip")):' ' m=json.loads(zipfile.ZipFile(z).read("manifest.json"))' ' d=m["sha256"]["trail.yaml"]' ' print("MATCHED " if d in live else "ORPHANED", m["trail"]["name"], d[:12], live.get(d,"(source drifted)"))' > "$T/c.py"; .venv/bin/python "$T/c.py"; rm -rf "$T"; }
== glossary describes the fence ==
glossary_stale=1
glossary_fence=0
name_ruling_present=1

```
== constitution earmarks ==
new_earmarks=0
== cartridge cache census ==
MATCHED  practice 1b96277218ed assets/trails/practice.yaml
MATCHED  botify_pageworkers 386ffff2f00f assets/trails/botify_pageworkers.yaml
ORPHANED public_walk 6ef6fdbdb45b (source drifted)
MATCHED  first_context e5ca6ec0f718 assets/trails/first_context.yaml
(nix) pipulate $
```

3: Patches:
Blast Radius Check to establish bisection Left-hand Causal Boundary. It is a Popper-thing. Science.
On branch main
Your branch is up to date with 'origin/main'.

```
nothing to commit, working tree clean
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'GLOSSARY.md'.
(nix) pipulate $ d
diff --git a/GLOSSARY.md b/GLOSSARY.md
index d200d52c..818bd02e 100644
--- a/GLOSSARY.md
+++ b/GLOSSARY.md
@@ -236,6 +236,7 @@ Entries are alphabetical, numbers spelled as spoken.
     mothercat <trail> --dry-narrate               # the rehearsal; nothing moves
     RIDE                                  # mck.sh's one-shot go-ahead
     CAPTURE                                   # the fence token, at every stop
+    DECANT                                    # the egress fence, once, at the end

 ### The four moves (plus the fifth nobody names)
 **SETTLE** — *human-only, non-deterministic.* Look at the page; clear the auth,
(nix) pipulate $ m
^[[A^[[A^[[A^[[A📝 Committing: chore: Update GLOSSARY.md with "DECANT" entry
[main 9d997703] chore: Update GLOSSARY.md with "DECANT" entry
 1 file changed, 1 insertion(+)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'GLOSSARY.md'.
(nix) pipulate $ d
diff --git a/GLOSSARY.md b/GLOSSARY.md
index 818bd02e..cd439077 100644
--- a/GLOSSARY.md
+++ b/GLOSSARY.md
@@ -254,9 +254,16 @@ the human on the hook for exactly one move and fences that move.
 **ADVANCE** — *the step.* Next bookmark. Only after a receipt.

 **DECANT** — *the fifth move, and the one that makes it a product.* At the end
```

-of a clean ride, every captured lens is folded into one markdown bundle and
-copied to the clipboard. Big artifacts (hydrated DOM, raw source) are CITED BY
-PATH, never inlined, so a huge page cannot flood the paste.
+of a clean ride, every captured lens is folded into one markdown bundle. Big
+artifacts (hydrated DOM, raw source) are CITED BY PATH, never inlined, so a
+huge page cannot flood the paste. Then the bundle STOPS: the rider prints its
+size and asks for one more word. Type DECANT and it goes to the clipboard;
+type anything else and it stays on disk and the rider names the exact
+directories. THE SECOND FENCE, AND IT GATES A DIFFERENT ACT: CAPTURE gates
+each WRITE TO DISK on the operator's own machine, DECANT gates the composite
+LEAVING it. Two words, because a fourth identical prompt is answered by muscle
+memory rather than decision, and because a transcript of four CAPTUREs cannot
+say which one authorized the egress.

 ### The nouns
 trail — a walkthrough as DATA: one JSON-subset-of-YAML file in
(nix) pipulate $ m
📝 Committing: chore: Refine GLOSSARY.md description - Decant definition clarity
[main b6979574] chore: Refine GLOSSARY.md description - Decant definition clarity
 1 file changed, 10 insertions(+), 3 deletions(-)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'GLOSSARY.md'.
(nix) pipulate $ d
diff --git a/GLOSSARY.md b/GLOSSARY.md
index cd439077..c6122444 100644
--- a/GLOSSARY.md
+++ b/GLOSSARY.md
@@ -273,11 +273,33 @@ noun for this. The corpus already carries `assets/trails/` (directory),
 `public_walk.yaml` (filename), `trail` (this entry), and `mothercat` (alias)
 for one concept; a fifth spelling makes every rg probe return half the truth,
 which is the UNANCHORED-WITNESS failure arranged in advance. TRAIL is the noun
-for the mutable source. PROPOSED, no mechanism yet: a sealed, hash-pinned,
-content-addressed package of a trail is a WALK CARTRIDGE, and it is a
-different artifact rather than a synonym. The two words earn separation for
-one reason only, and it is the reason that matters: a trail is mutable source,
-a walk cartridge cannot change. A trail that auto-updates is not replayable.
+for the mutable source. BANKED 2026-08-07 (was PROPOSED for one ride): the
+sealed, hash-pinned, content-addressed package of a trail is a WALK CARTRIDGE
+and the mechanism now exists. The two words earn separation for one reason
+only, and it is the reason that matters: a trail is mutable source, a walk
+cartridge cannot change. A trail that auto-updates is not replayable.
+
+**walk cartridge** — the sealed form of a trail: a two-member canonical ZIP
+(`trail.yaml` byte-identical to source, `manifest.json` carrying its SHA-256
+plus the consent surface), minted by `scripts/walk_cartridge.py`, living at
+`data/walks/<archive_sha256>/walk.zip`. Fixed epoch, fixed mode, fixed member
+order, no wall-clock value anywhere in the hashed body, so identical trail

+bytes produce identical archive bytes forever. The path is a pure function of
+the content: THE DERIVED-PATH RULE made literal, collisions unrepresentable,
+re-sealing idempotent. WITNESSED both directions on 2026-08-07: the same
+trail re-sealed to the same digest, and one edited byte moved the source
+digest away from the sealed one. WHAT A HASH DOES NOT BUY is authorship; a
+malicious trail with a perfect digest verifies green forever, which is why
+the signature is a different rung and is owed separately.
+
+**consent surface** — the walk cartridge's analogue of payload-to-prompt: a
+projection a human reads BEFORE deciding to ride, recomputed by the verifier
+from `trail.yaml` and required to match exactly, so it cannot drift from the
+bytes it describes. Name; stop names ORDERED, never sorted, because sequence
+is the one property a replayable artifact exists to preserve; url_envs sorted,
+because those are a set; connector scripts sorted and unique; and the four
+browser facts including `profile_name`, which selects WHICH logged-in profile
+opens and which no other check in the system inspects.

 rider — the walkthrough as CODE: `scripts/mother_cat.py`. One rider, N
 trails. The split is the whole design.
(nix) pipulate $ m
📝 Committing: Refactor: Clarify walk cartridge and consent surface definitions
[main 85f7716b] Refactor: Clarify walk cartridge and consent surface definitions
 1 file changed, 27 insertions(+), 5 deletions(-)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'GLOSSARY.md'.
(nix) pipulate $ d
diff --git a/GLOSSARY.md b/GLOSSARY.md
index c6122444..95e0d383 100644
--- a/GLOSSARY.md
+++ b/GLOSSARY.md
@@ -335,8 +335,14 @@ declares its own; nothing guesses.

 `--yolo` — **the real yolo switch, and it is deliberately modest.** Skips the
 spoken rehearsal AND the RIDE confirmation. It does NOT and will not skip the
-CAPTURE fences, because those are the write barrier. **CEREMONY IS SKIPPABLE;
+CAPTURE fences, nor the DECANT gate at the end. **CEREMONY IS SKIPPABLE;
 BARRIERS ARE NOT** — a confirmation authorizes a *sequence*, a fence authorizes
-each *write*. And the unfenced lane already exists under other names
-(`selenium_automation`, `!URL`, `?URL`), so a fence-skipping flag would be a
-second spelling of a shipped capability.
+each *write*. For CAPTURE there is a second reason: the unfenced lane already
+exists under other names (`selenium_automation`, `!URL`, `?URL`), so a
+fence-skipping flag would be a second spelling of a shipped capability. For
+DECANT that reason RUNS THE OTHER WAY and the fence survives anyway — `_decant`
+has one builder and one caller, so a skip flag would CREATE a capability rather
+than duplicate one — and the decisive argument is temporal: `--yolo` is typed
+before a browser opens, and cannot consent to the disposition of a bundle that

+did not exist when it was typed. Nothing unattended is lost, because the
+CAPTURE fences already block.
(nix) pipulate $ m
📝 Committing: chore: Clarify `--yolo` flag behavior in GLOSSARY.md
[main 5b4f8081] chore: Clarify `--yolo` flag behavior in GLOSSARY.md
 1 file changed, 10 insertions(+), 4 deletions(-)
(nix) pipulate $ patch
(nix) pipulate $ app
✅ DETERMINISTIC PATCH APPLIED: Successfully mutated 'foo_files.py'.
(nix) pipulate $ d
diff --git a/foo_files.py b/foo_files.py
index 9834f835..75152376 100644
--- a/foo_files.py
+++ b/foo_files.py
@@ -2288,6 +2288,60 @@ scripts/xp.py # [672 tokens | 2,521 bytes]
 """

 # #todo #to-do
+# - EARMARK: THE UNREACHABLE GUARD (banked 2026-08-07, traceback-witnessed): a
+#   check placed AFTER a line that can raise the same class of error is dead
+#   code, and its message is a promise the program cannot keep. CONVICTION:
+#   walk_cartridge.verify_walk_cartridge calls archive.read(info) in its member
+#   loop and archive.testzip() after it. A mid-member byte flip makes read()
+#   raise zipfile.BadZipFile from _update_crc, so testzip()'s "CRC failure in
+#   <member>" -- the ONE message written for CRC corruption -- is unreachable
+#   for CRC corruption. Worse, BadZipFile subclasses Exception and NOT OSError
+#   or ValueError, so _cmd_verify's except tuple does not catch it: the tool
+#   TRACEBACKS at rc=1 where its own docstring promises "2 refusal". FAIL-CLOSED
+#   SURVIVES -- nothing verifies -- but the exit-code contract does not, so a
+#   consumer scripting on rc==2 reads a corrupt cartridge as a different failure
+#   class than a refused one, and a human reads a stack trace as a tool bug
+#   rather than as tamper detection working. Sibling of THE VERIFIER THAT NEVER
+#   RAN: that one is a witness placed after an action that can kill the
+#   reporter; this is a witness placed after a line that raises first. TEST, and
+#   it costs one reading: for every guard, name the line ABOVE it that could
+#   raise the same error, then confirm that error's TYPE is in the caller's
+#   except tuple. Two questions, and the second is the one nobody asks.
+# - EARMARK: CONSENT CANNOT PRECEDE ITS OBJECT (banked 2026-08-07,
+#   fence-witnessed): a flag typed at t=0 cannot authorize the disposition of an
+#   artifact that does not exist until t=N, because the consenter had not seen
+#   it. This is the thing CEREMONY IS SKIPPABLE; BARRIERS ARE NOT does not
+#   state. That rule grades a gate by WHAT it authorizes (a sequence versus a
+#   write); this one grades it by WHEN, and they part company on exactly the
+#   interesting case. --yolo skipping the RIDE confirmation is legitimate: the
+#   sequence is fully described at t=0. --yolo skipping the DECANT gate is not,
+#   and the reason is not that a fence is a fence -- it is that the bundle, its
+#   size, and its contents did not exist when --yolo was typed. STANDING
+#   CONSEQUENCE, and it is cheap: put the DISCLOSURE at t=0 and the FENCE at

```
+#   t=N. Disclosure early costs nothing and makes the later fence a formality
+#   instead of an ambush; a fence early is consent to a thing nobody has seen.
+#   Also witnessed: the bypass-under-another-name corollary CLEARED this fence
+#   rather than excusing it -- _decant has one builder and _ride_async is its
+#   only caller, so a skip flag would CREATE a capability rather than duplicate
+#   a shipped one, which is the exact opposite of the capture case that
+#   corollary was written for. A corollary that can clear as well as excuse is
+#   a corollary rather than a loophole.
+# - EARMARK: THE PROTECTION-LAG POLARITY (banked 2026-08-07, three-file
+#   conviction): when a mechanism is REMOVED the stale strings overstate
+#   protection, which is the failure everyone expects. When a protection is
+#   ADDED the stale strings understate it -- and that direction is MORE
+#   expensive, because the string then overstates the RISK to a newcomer at the
+#   exact moment they decide whether to ride at all. CONVICTION: the DECANT
+#   fence landed in one ride; for that ride mother_cat._announce_consent told
+#   every rider "WITHOUT ASKING AGAIN" about a gate that does ask,
+#   public_walk.yaml's third stop said the same thing in trail DATA, and
+#   GLOSSARY.md said it a third and fourth time -- and the glossary instances
+#   were found only because a dismount went looking. STANDING CONSEQUENCE, the
+#   search discipline THE SAME-CAR LABEL RULE implies but never spells: when a
+#   gate lands, grep the CORPUS for every string describing that gate -- prose,
+#   trail data, glossary, launcher cards, spoken narration -- not merely the
+#   file the gate lives in. The gate's own file is the one place the author
+#   cannot forget, which is precisely why it is the one place that is fine.
 # - EARMARK: THE ELIMINATOR NEEDS A GENERATOR (banked 2026-08-06,
 #   thrice-convicted in one ride): the bisecting inward spiral CANNOT FAIL TO
 #   ELIMINATE and absolutely CAN FAIL TO CONVERGE -- silently, while looking
(nix) pipulate $ m
📝 Committing: chore: Update EARMARK comments with detailed analysis and reasoning.
[main 741b62e1] chore: Update EARMARK comments with detailed analysis and reasoning.
 1 file changed, 54 insertions(+)
(nix) pipulate $ git push
Enumerating objects: 18, done.
Counting objects: 100% (18/18), done.
Delta compression using up to 48 threads
Compressing objects: 100% (15/15), done.
Writing objects: 100% (15/15), 5.42 KiB | 924.00 KiB/s, done.
Total 15 (delta 10), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (10/10), completed with 3 local objects.
To github.com:pipulate/pipulate.git
   cc94d355..741b62e1  main -> main
(nix) pipulate $
```

Wow that's a lot of patches for a wrap-up. What do you make of all this Gemini?

Gemini 3.6 Thinking: What a ride. What you've built here isn't just a feature patch—it's a masterclass in **ratcheting software architecture** through empirical, receipt-driven iteration. You started with a complex, potentially messy problem: how to distribute experimental browser automation "walks" without exploding key-management complexity or risking unauthorized code

execution. You ended with a clean, keyless, content-addressed system protected by clear physical gates.

The Two-Witness Gate Architecture

What Was Locked In (The Wins)

1. The Two-Witness Gate Architecture

- **CAPTURE (Write Barrier):** Gates writing raw wire truth (CDP logs, DOM snapshots, A11y trees) to local disk per stop.
- **DECANT (Egress Barrier):** Gates bundling and handing authenticated session material over to the clipboard at $t=N$.
- **The Principle:** You proved that --yolo at $t=0$ *cannot* authorize egress at $t=N$, because consent cannot precede an object that didn't exist when the flag was typed.

2. Immutable, Content-Addressed Cartridges (walk_cartridge.py)

- Standard-library only, single-file reproducible container (data/walks/<sha256>/walk.zip).
- **Consent Surface:** Derives $t=0$ disclosures (ordered stops, required env vars, runnable connectors, browser profiles) directly from trail bytes so manifest and execution can never drift.
- Proven idempotent: same bytes produce 2da019ab..., modified bytes produce 6ef6fdbd....

3. Falsification Over Assumption (The "Popper-Thing")

You ran every candidate against real probes and caught genuine edge-case mechanics:

- **The Unreachable Guard:** Discovered that mid-member CRC flips raise BadZipFile during read() before testzip() is ever reached.
- **The Actuator Imposter:** Caught nix resolving as a shell function in one environment and an unexported binary in another using type nix and env.
- **Protection-Lag Polarity:** Updated the corpus (GLOSSARY.md, public_walk.yaml, foo_files.py) so stale docs don't tell terrifying lies to newcomers about non-existent risks.

The Big Picture

By decoupling **authoring** (which uses normal git workflows in gitignored negative space) from **walking** (which uses SHA-256 hashes and local consent gates), you eliminated the need to build a complex multi-repo SSH deploy key manager.
The walker doesn't need git, keys, or credentials—just a URL, a hash, and two explicit words typed into a terminal.

Now that the architecture is locked, verified, and pushed to main, where should we focus when you kick off the next session—would you like to execute a full end-to-end public_walk ride to witness real clipboard decanting, or tackle manifest signing (ssh-keygen -Y) next?

Book Analysis

Ai Editorial Take

What is most striking about this entry is its reversal of traditional debugging heuristics. Instead of asking 'why did this code break?', the architecture asks 'how can we make the failure mode so explicit and structured that the system proves its own correctness?' By treating test harnesses as empirical instruments rather than mere check-boxes, the work bridges the gap between theoretical software verification and fast-paced developer velocity.

Content Potential And Polish

- **Core Strengths:**
- Rigorous application of Popperian falsification to software engineering.
- Clear distinction between ceremony (skippable) and barriers (unmovable).
- Practical integration of content-addressed canonical archives for reproducible workflows.
- **Suggestions For Polish:**
- Ensure all technical terms like 'walk cartridges' are introduced with brief contextual definitions.
- Tighten the narrative transitions between the technical probe logs and the philosophical reflections.

Next Step Prompts

- Analyze the remaining unreachable guard branches in the cartridge verification pipeline and design a comprehensive test suite for exception type mapping.
- Explore the integration of SSH-based key signing (manifest-signing lane) to establish non-repudiation for immutable walk cartridges.