# Revisión funcional externa de rlm-agent

- **Proyecto:** https://github.com/nicolasramos/rlm-agent
- **Commit revisado:** `5178d62ffece0679746ad3f7a14cd3aa4c30bf59`, obtenido el 2026-09-11. Todos los números de línea de este informe se refieren a este commit.
- **Qué es:** una revisión funcional hecha por un único revisor externo, preparada como aporte técnico para los mantenedores. Ningún segundo revisor independiente la ha comprobado (ver §9).
- **Idioma:** traducción al español del informe original en inglés, versión v2 (2026-09-11). Las salidas de los scripts del Apéndice B se reproducen tal cual, en inglés.

**Cómo leer los hallazgos.** Cada hallazgo indica las versiones exactas, el archivo y la función, una reproducción mínima, lo que se esperaba, lo que se observó y lo que la observación *no* demuestra. «Observado» significa observado en nuestra reproducción, con el commit, las versiones de runtime y la plataforma indicados. No se probó nada dentro de OpenCode, Pi ni Hermes; el §5 explica hasta dónde llega cada hallazgo.

## Resumen

### Hallazgos principales

| ID | Prioridad | Nivel de evidencia | Hallazgo | Sección |
|---|---|---|---|---|
| F-2 | P1 · Alta | 1 · código/runtime | `rlm_lake.forget` reescribe el archivo completo del lake; se perdieron entradas añadidas en paralelo por otros procesos del kernel (24, 27 y 35 de 200; control sin `forget`: 0, 0 y 0) | §6.1 |
| F-3 | P1 · Alta | 2 · solo adaptador | El `ContextLake` de Hermes nunca recarga su caché; un `forget` posterior eliminó una entrada escrita por otro proceso | §6.2 |
| F-1 | P1 · Alta | runtime 1 · adaptador 2 · OpenCode 3 | Bun 1.4.0: `Bun.write()` con la opción que usa el adaptador de OpenCode no conservó las entradas anteriores; solo quedó la última entrada del lake. El impacto en OpenCode depende de la versión de Bun que realmente ejecute | §6.3 |

### Observaciones secundarias

| ID | Prioridad | Nivel de evidencia | Observación | Sección |
|---|---|---|---|---|
| B-1 | P2 · Media | 3 · verificación en host | La salida de `print()` del kernel se emite con `"id": null` y después de `result`; esto localiza una causa de la nota documentada de que `ipython` no captura `print()` | §7.1 |
| B-2 | P2 · Media | 3 · verificación en host | El cliente de kernel del adaptador de Hermes ignora los eventos `stdout`: `print` devuelve `→ None` y `%%bash` solo `→ exit code 0` | §7.2 |
| D-1 | P2 · Media | 2 · solo adaptador | El README y el docstring del adaptador de Hermes describen la compactación «via `register_context_engine`»; `register()` no registra ningún callback de compactación | §7.3 |
| F-4 | P3 · Baja | 1 · código/runtime | Tras un `store` que falla con error reportado, la misma sesión del kernel sigue sirviendo el valor no escrito | §7.4 |
| F-5 | P3 · Baja | 3 · verificación en host | El archivo del lake se elige a partir de la cadena de ruta del proyecto; una barra final o un enlace simbólico seleccionan otro lake | §7.5 |
| P-3 | P3 · Baja | 1 · código/runtime | `search` del kernel ignora las etiquetas; el kernel guarda timestamps en segundos y los adaptadores en milisegundos | §7.6 |

## 1. Alcance

**Dentro del alcance:** las partes de rlm-agent que pueden ejercitarse sin editor y sin LLM:

- el kernel persistente de Python (`kernel/kernel.py`), a través de su protocolo stdio;
- las implementaciones del context lake: el puente `rlm_lake` del kernel y el código `ContextLake` de los adaptadores de OpenCode, Pi y Hermes;
- el cliente de kernel y el registro del plugin del adaptador de Hermes.

**Fuera del alcance:** los editores. Los adaptadores se cargaron **fuera** de OpenCode, Pi y Hermes, así que ningún resultado de este informe es una observación a nivel de editor (ver §5, §11 y §13).

## 2. Versión / commit revisado

| archivo | sha256 |
|---|---|
| `kernel/kernel.py` | `c4830814df5d79b3906456f5d98369fdc4768c14e33eb766409c88e125e32100` |
| `adapters/opencode/rlm.ts` | `55490fee09b9928b3eee264b4ec0a68def4374145380ea120994c3e8453ea36a` |
| `adapters/pi/rlm.ts` | `cf5800e91d158a54894b54c6456a5f8be0582cfb6db55b02d2d6a5d005988a66` |
| `adapters/hermes/__init__.py` | `67a3914f6e9b4e5cdd681ea3b33315d26c5f42f37829f538da5b994ee25c6cfa` |
| `tests/test_kernel.py` | `d59820155fdffedc7998e62b7a9de435bb5f2202c875665fa74a02476cd143b0` |
| `README.md` | `9668ba662b239dbd880bc87b1a08bfb395791e5f839eae3a973b6ed0cd5029ec` |

El repositorio no se modificó. El adaptador de OpenCode, que importa un paquete del host, se ejecutó desde una copia idéntica byte a byte junto a un stub mínimo. El módulo del adaptador de Hermes se importó directamente desde el checkout.

## 3. Entorno de prueba

| elemento | valor |
|---|---|
| plataforma | macOS 26.6.2, arm64 |
| Python | 3.13.15 |
| Bun | 1.4.0, revisión `1381054db427439f5d669f28aa1984faa26d8e94` (scripts de F-1) |
| Node.js | 24.20.0 (solo en las ejecuciones del arnés con el adaptador de Pi; ningún script de `REPRO/` usa Node) |
| editores | ninguno instalado: **no** se usaron OpenCode, Pi ni Hermes |
| llamadas a LLM | ninguna |
| entorno | mínimo (`env -i`), con `HOME` y `TMPDIR` en un directorio temporal de trabajo; no se leyó ni escribió ninguna configuración real de editores |
| red | denegada a nivel del sistema operativo para los procesos de Python y Node. El mismo método no restringió el proceso de Bun; ninguna prueba necesitaba red |

**Stubs del host.** Solo satisfacen importaciones y llamadas de registro; no sustituyen ninguna lógica del lake ni del kernel.

- **Adaptador de OpenCode** (arnés y `REPRO/f1b`): `@opencode-ai/plugin` sustituido por un stub mínimo cuyo `tool(def)` devuelve `def`.
- **Adaptador de Pi** (solo arnés): `typebox` sustituido por un stub mínimo, más un objeto que registra `registerTool`.
- **Adaptador de Hermes** (arnés y cinco scripts): el módulo se importa directamente; `REPRO/d1` pasa un objeto de contexto que registra cada llamada a método.

## 4. Qué se probó

**Protocolo del kernel:**

- arranque y evento `ready`;
- evaluación de expresiones;
- salida de `print()`;
- persistencia del espacio de nombres dentro de una sesión;
- estado tras un reinicio;
- stdout, stderr y código de salida de `%%bash`;
- el `tests/test_kernel.py` del propio proyecto (pasan 25/25 comprobaciones).

**Context lake.** La misma batería se ejecutó contra el puente `rlm_lake` del kernel y el código `ContextLake` de los tres adaptadores:

- `store` y `get` exacto, incluido Unicode multilínea;
- `search`: mayúsculas y minúsculas, límite de resultados, `max_results`, etiquetas, regex y regex inválida;
- `find` y `stats`;
- persistencia entre sesiones;
- aislamiento entre proyectos y entre distintas escrituras de una misma ruta de proyecto;
- modificación externa del archivo del lake vista por una sesión viva y por una nueva;
- `forget`;
- una prueba determinista de vista obsoleta;
- una pequeña prueba de concurrencia con control;
- una prueba de fallo de escritura.

**Dos caminos de código.**

1. **Un arnés de pruebas** ejecutó la batería anterior sobre las cuatro implementaciones del lake. No se incluye. Cuando este informe usa un resultado del arnés, lo marca como observación de apoyo.
2. **10 scripts de reproducción en `REPRO/`, con caminos de control** (§9). Se escribieron por separado del arnés, crean en línea sus propios stubs mínimos y se ejecutaron sobre un checkout limpio; sus salidas están en `artifacts/`.

Ambos caminos los escribió el mismo revisor: son implementaciones separadas, no reproducciones independientes hechas por otra parte. Cada hallazgo de §6–§7 tiene un script en `REPRO/` y su salida en `artifacts/`.

## Lo que funcionó según lo documentado

Observado en las ejecuciones del arnés; la mayoría de estos puntos no tiene un script en `REPRO/`.

**Kernel**

- Arranca en unos 0,2 s y mantiene limpio su canal stdout, que solo transporta tramas del protocolo.
- Evalúa expresiones, conserva el estado entre peticiones e informa de los errores (por ejemplo `NameError`) como eventos `error`.
- Atribuye la salida de `%%bash` a la petición, antes de `result`, con el código de salida como valor.
- No conserva el espacio de nombres tras un reinicio salvo que se tome un snapshot, lo que coincide con el diseño.

**Context lake**

- `store` y `get` devuelven exactamente lo guardado, incluido Unicode multilínea, en las implementaciones del kernel, Hermes y Pi.
- Las entradas persisten entre sesiones, y cada cadena de ruta de proyecto tiene su propio lake.
- `search` no distingue mayúsculas, respeta el límite de 10 resultados y `max_results`, admite regex y recurre a la coincidencia literal ante un patrón inválido.
- `find` busca subcadenas sin distinguir mayúsculas y sin regex.

- Las implementaciones del kernel, Pi y OpenCode recargan el lake cuando cambia el mtime del archivo, así que una sesión viva ve las altas y modificaciones externas.
- `forget` elimina las entradas coincidentes y compacta el archivo a una línea por clave.

## 5. Cómo se clasifican los hallazgos

El **nivel de evidencia** indica hasta dónde llega una observación. Se asigna según dónde se observó el propio resultado dañino.

| nivel | significado |
|---|---|
| **1 · Confirmado en el código/runtime probado** | El resultado se observó ejecutando código sin modificar (un proceso del kernel, una función o una clase) o el propio runtime, en las versiones indicadas. Ningún comportamiento del editor interviene en producirlo. |
| **2 · Confirmado solo a nivel de adaptador** | El resultado se observó en código del adaptador ejecutado fuera de su editor, con las entradas o el ciclo de vida del editor simulados. El mecanismo está confirmado; su efecto dentro del editor no se observó. |
| **3 · Requiere verificación en la integración con el host** | El comportamiento se reprodujo a nivel de kernel o de adaptador, pero que afecte o no a los usuarios depende de un factor del host que no se observó y que podría cambiar la conclusión. Cada hallazgo nombra ese factor. |

**Prioridad**

| prioridad | significado |
|---|---|
| **P1 · Alta** | En nuestra reproducción se perdieron datos guardados en el lake sin ningún aviso. |
| **P2 · Media** | Se pierde o se atribuye mal una salida, o no ocurre un comportamiento documentado. |
| **P3 · Baja** | Una inconsistencia de bajo impacto, una limitación o una diferencia de implementación. |

**Tipo**

| tipo | significado |
|---|---|
| **defecto** | Comportamiento que contradice lo documentado, o lo que suponemos que es el comportamiento previsto (en ese caso, el hallazgo indica que es una suposición). |
| **causa de una limitación documentada** | El proyecto ya documenta el síntoma; el hallazgo localiza de dónde viene. |
| **discrepancia** | La documentación y la implementación no coinciden. |
| **limitación** | Se deriva del diseño; se incluye porque puede sorprender a los usuarios. |
| **diferencia** | Dos implementaciones se comportan de forma distinta; no es necesariamente un defecto. |

**Campos.** Cada hallazgo indica prioridad y tipo, nivel de evidencia, versiones, ubicación en el código, reproducción mínima y archivo de evidencia. Después vienen *Esperado*, *Observado*, *No demostrado*, *Impacto* y *Dirección sugerida*. Los resultados del arnés aparecen solo como observaciones de apoyo etiquetadas.

---

## 6. Hallazgos principales

### 6.1 F-2 — `forget` reescribe el archivo del lake; se pueden perder entradas añadidas en paralelo por otros procesos

- **Prioridad · tipo:** P1 · Alta · defecto
- **Nivel de evidencia:** 1 · confirmado en el código/runtime probado (procesos del kernel; sin editor)
- **Versiones:** rlm-agent `5178d62` · Python 3.13.15 · macOS 26.6.2 arm64
- **Ubicación:** `kernel/kernel.py`, `_LakeBridge.forget` (líneas 429–442).

    - Recarga el archivo si cambió su mtime (`_ensure`, líneas 339–365) y elimina de memoria las claves que coinciden.
    - Después trunca y reescribe el archivo completo desde memoria (`open(self._file, "w")`, línea 439).
    - Nada serializa esta operación con otros escritores.

- **Reproducción mínima:** `python3 REPRO/f2_kernel_forget_race.py <checkout> 3`
- **Evidencia:** `artifacts/f2_kernel_forget_race.txt`

**Esperado.** Solo se eliminan las entradas que coinciden con el patrón. La herramienta `rlm_forget` se describe como la que elimina entradas "whose key or content matches a regex pattern" (`adapters/hermes/__init__.py`, línea 576). El `rlm_lake.forget` del kernel no tiene una descripción propia.

**Observado.** Varios procesos del kernel compartieron un mismo `RLM_LAKE_FILE` y empezaron sus bucles en el mismo instante:

- A y B guardaron 100 entradas cada uno;
- C ejecutó 10 ciclos de guardar su propia clave y luego aplicarle `forget`.

| escenario, 3 ejecuciones de cada uno | entradas de A/B que faltaban al final (de 200) | líneas JSONL inválidas |
|---|---|---|
| A y B, más C ejecutando `forget` | 24, 27, 35 | 0 |
| control: solo A y B | 0, 0, 0 | 0 |

*Observaciones de apoyo (arnés):*

- Con `forget` se perdieron 3, 30, 54 y 12 entradas, y ninguna en los controles.
- También se observaron pérdidas con el `ContextLake.forget` de los adaptadores de Hermes y Pi, que siguen el mismo patrón de recargar y después reescribir (`adapters/hermes/__init__.py` líneas 268–282; `adapters/pi/rlm.ts` líneas 258–277).

**No demostrado.**

- Con qué frecuencia ocurre en sesiones reales. La prueba usa un bucle deliberadamente ajustado y el número de pérdidas varía entre ejecuciones; en otras máquinas una ejecución podría no perder ninguna.
- El comportamiento en otros sistemas operativos o sistemas de archivos.
- El `forget` del adaptador de OpenCode bajo concurrencia: sus resultados en el arnés quedaron dominados por F-1.

**Impacto, si ocurre en uso real.** Las entradas desaparecen sin aviso siempre que un proceso añade entradas a un lake mientras otro ejecuta `forget` sobre el mismo lake. Por ejemplo, el adaptador de Hermes arranca un kernel por sesión (`_get_kernel`, líneas 360–365) y apunta cada kernel al lake de su directorio de trabajo (línea 91). Por tanto, dos sesiones en el mismo proyecto comparten un lake.

**Dirección sugerida** (una opción entre varias).

- Serializar el ciclo de leer, modificar y escribir, por ejemplo con un bloqueo de archivo.
- Alternativamente, no reescribir al borrar: añadir un marcador de borrado y compactar bajo bloqueo.

### 6.2 F-3 — El `ContextLake` de Hermes mantiene una caché obsoleta; un `forget` posterior eliminó la entrada de otro proceso

- **Prioridad · tipo:** P1 · Alta · defecto
- **Nivel de evidencia:** 2 · confirmado solo a nivel de adaptador (`ContextLake` importado fuera de Hermes)
- **Versiones:** rlm-agent `5178d62` · Python 3.13.15 · macOS 26.6.2 arm64
- **Ubicación:** `adapters/hermes/__init__.py`.
    - `ContextLake._ensure_loaded` (líneas 203–216) lee el archivo una vez por instancia y nunca más.
    - `ContextLake.forget` (líneas 268–282) reescribe el archivo desde esa caché mediante `_rewrite` (líneas 223–226).

    - El plugin mantiene una instancia por directorio de proyecto durante toda la vida del módulo (`_lakes` y `_lake_for`, líneas 349–357).
- **Reproducción mínima:** `python3 REPRO/f3_hermes_stale_cache.py <checkout>`
- **Evidencia:** `artifacts/f3_hermes_stale_cache.txt`

**Esperado.** Las entradas que otro escritor añade al lake permanecen en él salvo que coincidan con el patrón de `forget`.

- Las otras tres implementaciones recargan cuando cambia el mtime del archivo: el kernel (`_LakeBridge._ensure`, `kernel/kernel.py` líneas 339–349), OpenCode (`ensureLoaded`, `adapters/opencode/rlm.ts` líneas 302–314) y Pi (`ensure`, `adapters/pi/rlm.ts` líneas 180–196).
- Un comentario del adaptador de OpenCode explica el motivo: las entradas que escribe el kernel "must be visible to the plugin tools" (líneas 305–306).

**Observado.**

1. La instancia P1 guardó `seed`, lo que cargó su caché.
2. Un segundo proceso de Python guardó `from-p2` en el mismo archivo del lake. El archivo contenía entonces `seed` y `from-p2`.
3. `P1.get("from-p2")` no encontró la entrada.
4. `P1.forget("^seed$")` eliminó 1 entrada.
5. El archivo del lake quedó **vacío**: `from-p2` se había eliminado junto con `seed`.

*Observación de apoyo (arnés):* el mismo resultado en 3 de 3 ejecuciones.

**No demostrado.**

- El comportamiento dentro de un host Hermes en ejecución: cuánto tiempo vive el estado del módulo del plugin y qué `cwd` pasa el host a las herramientas.
- El caso del kernel como escritor, descrito en *Impacto*, se deduce del código; no se ejecutó. El script usa un segundo proceso de Python como escritor.

**Impacto, si se confirma en Hermes.**

- El adaptador arranca el kernel de cada sesión con `RLM_LAKE_FILE` apuntando al lake de su `cwd` (línea 91).
- Las herramientas del lake lo buscan con la misma expresión de `cwd` (por ejemplo, líneas 418–419).
- Por tanto, las entradas escritas desde el kernel con `rlm_lake.store`, o por otro proceso de Hermes, serían invisibles para las herramientas, y un `rlm_forget` posterior podría borrarlas.

**Dirección sugerida.** Recargar cuando cambie el mtime, como hacen las demás implementaciones, y volver a leer justo antes de reescribir (ver también F-2).

### 6.3 F-1 — Bun 1.4.0: `Bun.write()` con la opción que usa el adaptador de OpenCode no conservó las entradas anteriores del lake

- **Prioridad · tipo:** P1 · Alta · defecto observado en Bun 1.4.0. El impacto en OpenCode depende de la versión de Bun que realmente ejecute.
- **Nivel de evidencia:**
    - comportamiento del runtime (F-1a): 1 · confirmado en el runtime probado;
    - comportamiento del adaptador (F-1b): 2 · confirmado solo a nivel de adaptador;
    - efecto en los usuarios de OpenCode: 3 · requiere verificación en la integración con el host. El factor del host es la versión de Bun que ejecuta OpenCode y cómo trata esa versión esta llamada.
- **Versiones:** Bun 1.4.0, revisión `1381054db427439f5d669f28aa1984faa26d8e94` · rlm-agent `5178d62` · macOS 26.6.2 arm64
- **Ubicación:** `adapters/opencode/rlm.ts`, `ContextLake.append` (líneas 335–337), llamado desde `ContextLake.store` (línea 361). Ejecuta `await Bun.write(this.file, JSON.stringify(entry) + "\n", {`

`append: true }`).

- **Reproducción mínima:**
  - `bun REPRO/f1a_bun_write_append.ts` (solo runtime; no carga rlm-agent)
  - `bun REPRO/f1b_opencode_lake_store.ts <checkout>` (adaptador)
- **Evidencia:** `artifacts/f1a_bun_write_append.txt` · `artifacts/f1b_opencode_lake_store.txt`

**Esperado.** Cada entrada guardada permanece en el lake:

- el método se llama `append` y pasa `{ append: true }`;
- cada respuesta de `rlm_store` indica al modelo que recupere los datos más tarde con `rlm_get`, `rlm_search` o `rlm_find`.

**Observado.**

1. **Runtime (F-1a).** `Bun.write(file, "line-1\n", { append: true })` seguido de `Bun.write(file, "line-2\n", { append: true })` dejó el archivo con solo `"line-2\n"`.
2. **Adaptador (F-1b).** `rlm.ts` se cargó fuera de OpenCode, con solo `@opencode-ai/plugin` sustituido por un stub. Tres llamadas a `rlm_store` respondieron cada una `Stored "entry-N" …`. Después, el archivo del lake contenía solo `entry-3`.
3. **Misma sesión.** `rlm_stats` informó entonces `1 entries`.
   - El lake se recarga cuando cambia el mtime del archivo (líneas 302–314), así que la vista de la propia sesión también se reduce a la última entrada.
   - `rlm_get` y `rlm_search` leen las mismas entradas en memoria (líneas 365–386); el script no las llama.

*Observaciones de apoyo:*

- En el arnés, 18 llamadas sucesivas a `store` dejaron 1 entrada.
- F-1a también se ejecutó una vez fuera del entorno restringido, en la misma máquina, con el mismo resultado. Esa salida no se incluye.

**No demostrado.**

- Qué versión de Bun ejecuta OpenCode y, por tanto, si afecta a sus usuarios.
- Cómo tratan otras versiones de Bun `{ append: true }` en `Bun.write`: solo se probó la 1.4.0. Esta revisión no demuestra si esa opción forma parte de la API de `Bun.write` en alguna versión.
- El comportamiento dentro de OpenCode.

**Impacto, si OpenCode ejecuta una versión de Bun que se comporte así.** El lake de OpenCode no puede contener más de una entrada: cada `store` descarta sin aviso los datos guardados antes, mientras informa de éxito.

**Dirección sugerida.**

- Añadir las entradas mediante una API documentada para escribir al final del archivo en el runtime que usa OpenCode. Un ejemplo es `appendFileSync` de `node:fs`, que ya usa el adaptador de Pi (`adapters/pi/rlm.ts` línea 211).
- Añadir una prueba que guarde dos entradas y vuelva a leer el archivo.

---

## 7. Observaciones secundarias

### 7.1 B-1 — la salida de `print()` del kernel lleva `"id": null` y llega después de `result`

- **Prioridad · tipo:** P2 · Media · causa de una limitación documentada, y desviación del protocolo documentado del kernel
- **Nivel de evidencia:** 3 · requiere verificación en la integración con el host. El comportamiento del protocolo se observó directamente en el kernel; que la salida se pierda o no para los usuarios depende de cómo trate estos eventos cada cliente y cada host.

- **Versiones:** rlm-agent `5178d62` · Python 3.13.15 · macOS 26.6.2 arm64
- **Ubicación:** `kernel/kernel.py`.
  - La variable de contexto `_current_cell` (líneas 84–86) se lee al enviar la salida capturada (`_StreamWriter._flush`, línea 153), pero nunca se asigna.
  - `_handle_request` envía `result` (línea 482) y solo después vacía la salida capturada, en su bloque `finally` (líneas 517–521).
  - La ruta de `%%bash` envía su salida con el id de la petición (línea 234).
- **Reproducción mínima:** `python3 REPRO/b1_kernel_stdout_attribution.py <checkout>`
- **Evidencia:** `artifacts/b1_kernel_stdout_attribution.txt`

**Esperado.**

- El docstring del módulo enumera eventos `stdout`/`stderr` con el id de la petición, seguidos de `result` o `error` y después `done` (líneas 16–22).
- El docstring de `_StreamWriter` dice que las escrituras se atribuyen "to the cell running at write time via the _current_cell contextvar" (líneas 117–118).

**Observado.**

- `print("CANARY-PRINT")`, tres veces: `result (id=req-N)` → `stdout (id=null, text="CANARY-PRINT\n")` → `done (id=req-N)`.
- Comparación, una celda `%%bash`: `stdout (id=req-4)` → `result` → `done`.

**No demostrado.**

- El comportamiento de extremo a extremo en OpenCode, Pi o Hermes.
- El efecto en los clientes de OpenCode y Pi se deduce de leer su código; no se ejecutó.
  - Ambos buscan los eventos `stdout` por id de petición y eliminan la petición pendiente cuando llega `result` (`adapters/opencode/rlm.ts` líneas 183–192; `adapters/pi/rlm.ts` líneas 99–107), así que descartarían esta salida.
  - Asignar el id no bastaría por sí solo: la salida también tendría que vaciarse antes de `result`.
  - Para el cliente de Hermes, ver B-2.

**Contexto.** Las notas de uso del proyecto ya dicen que `ipython` no captura `print()` (`adapters/hermes/skills/rlm-usage/SKILL.md` línea 38; `adapters/pi/AGENTS.md` línea 33). Esta observación localiza una causa a nivel de protocolo.

**Dirección sugerida.**

- Asignar `_current_cell` durante cada petición y vaciar la salida capturada antes de enviar `result`.
- La comprobación «stdout capture» de `tests/test_kernel.py` pasa hoy porque el cliente de prueba concatena todos los eventos `stdout` sin comprobar su `id` (líneas 58 y 97–100). Comprobar el id detectaría el problema.

### 7.2 B-2 — el cliente de kernel del adaptador de Hermes no expone los eventos `stdout`

- **Prioridad · tipo:** P2 · Media · causa de una limitación documentada
- **Nivel de evidencia:** 3 · requiere verificación en la integración con el host. La clase `Kernel` y `_format_execute` del adaptador se ejecutaron fuera de Hermes; no se observó cómo presenta Hermes los resultados de las herramientas.
- **Versiones:** rlm-agent `5178d62` · Python 3.13.15 · macOS 26.6.2 arm64
- **Ubicación:** `adapters/hermes/__init__.py`.
  - `Kernel._handle_line` solo actúa sobre los eventos `result`, `error` y `names` (línea 128).
  - `Kernel.execute` devuelve siempre `"stdout": ""` (línea 163).
  - `_format_execute` (líneas 368–377) construye el texto de la herramienta a partir de ese resultado.
- **Reproducción mínima:** `python3 REPRO/b2_hermes_kernel_client_output.py <checkout>`
- **Evidencia:** `artifacts/b2_hermes_kernel_client_output.txt`

**Esperado.**

- La descripción de la herramienta `ipython` dice "Use %%bash for shell commands" (línea 570), lo que sugiere que se devuelve la salida del shell.
- Sin embargo, las notas de uso indican que la salida de `%%bash` no se captura (`adapters/hermes/skills/rlm-usage/SKILL.md` línea 39). El síntoma está, por tanto, documentado; esta observación localiza su causa.

**Observado.**

- `print("CANARY-PRINT")` devolvió el texto `→ None`.
- Una celda `%%bash` que imprime `CANARY-BASH` devolvió `→ exit code 0`.
- En ambos casos `execute()` devolvió `stdout: ''`.
- Control: `1 + 2` devolvió `→ 3`.

**No demostrado.** Si Hermes expone la salida del kernel por algún otro medio, o si posprocesa los resultados de las herramientas de un modo que cambie lo que ve el modelo.

**Impacto, si se confirma en Hermes.**

- La salida impresa y la del shell nunca llegan al texto que devuelve la herramienta `ipython`.
- `print(...)` produce el engañoso `→ None`, que es el valor de la expresión `print` final.
- Es un problema distinto de B-1: incluso los eventos `stdout` correctamente atribuidos, como los que produce `%%bash`, se descartan.

**Dirección sugerida.** Acumular los eventos `stdout` y `stderr` por id de petición en el cliente de Hermes, como hacen los clientes de OpenCode y Pi. Mostrar la salida de `print()` depende también de B-1.

### 7.3 D-1 — el README describe la compactación de Hermes vía `register_context_engine`; `register()` no la registra

- **Prioridad · tipo:** P2 · Media · discrepancia
- **Nivel de evidencia:** 2 · confirmado solo a nivel de adaptador. `register()` se ejecutó fuera de Hermes, con un objeto de contexto que registra cada llamada a método.
- **Versiones:** rlm-agent `5178d62` · Python 3.13.15 · macOS 26.6.2 arm64
- **Ubicación:** `adapters/hermes/__init__.py`, `register()` (desde la línea 402).
  - Una función local `_on_compact` se define en la línea 604 y nunca se registra.
  - Los únicos registros de hooks están en las líneas 619–620.
  - Documentación: `README.md` línea 61 y el docstring del módulo del adaptador, línea 9.
  - `adapters/hermes/plugin.yaml` solo enumera `on_session_end` en `provides_hooks` (líneas 19–20).
- **Reproducción mínima:** `python3 REPRO/d1_hermes_registered_hooks.py <checkout>`
- **Evidencia:** `artifacts/d1_hermes_registered_hooks.txt`

**Esperado.** La línea 61 de README.md indica soporte de compactación "(Hermes via `register_context_engine`; OpenCode via the compaction hook)". El docstring del módulo del adaptador (línea 9) dice lo mismo.

**Observado.**

- `register(ctx)` registró 12 herramientas, los hooks `on_session_end` y `transform_tool_result`, y una sección del system prompt.
- No hizo ninguna llamada de registro de compactación ni de context engine.
- `_on_compact` está entre las funciones definidas dentro de `register()`.

**No demostrado.** Si Hermes ofrece integración de compactación mediante algún mecanismo no ejercitado aquí.

**Impacto, si se confirma en Hermes.** Los usuarios de Hermes no recibirían, al compactar, el snapshot del kernel y el resumen de variables que describe el README.

**Dirección sugerida.** Registrar el callback de compactación mediante la API correspondiente de Hermes, o alinear el README y el docstring con el comportamiento actual.

### 7.4 F-4 — tras un `store` fallido, la misma sesión sigue sirviendo el valor

- **Prioridad · tipo:** P3 · Baja · defecto (consistencia)
- **Nivel de evidencia:** 1 · confirmado en el código/runtime probado (procesos del kernel)
- **Versiones:** rlm-agent `5178d62` · Python 3.13.15 · macOS 26.6.2 arm64
- **Ubicación:** `kernel/kernel.py`, `_LakeBridge.store` (líneas 374–392). La entrada en memoria se asigna (línea 390) antes de escribirla en el archivo (línea 391), y nada la revierte si la escritura lanza una excepción.
- **Reproducción mínima:** `python3 REPRO/f4_kernel_write_failure.py <checkout>` (como usuario no root)
- **Evidencia:** `artifacts/f4_kernel_write_failure.txt`

**Esperado.** No documentado. Suposición nuestra: un `store` que lanzó un error no se presenta después como guardado.

**Observado.**

1. Un kernel guardó `ok-entry`.
2. El archivo del lake se puso en solo lectura.
3. `store('failed-entry', 'NEVER-WRITTEN')` lanzó `PermissionError`, que llegó al llamante.
4. En el **mismo** kernel, `get('failed-entry')` devolvió `'NEVER-WRITTEN'` y `stats()` contó 2 entradas.
5. Un kernel **nuevo** sobre el mismo `RLM_LAKE_FILE` devolvió `None` para `failed-entry` y `'OK'` para `ok-entry`.

*Observación de apoyo (arnés):* el `ContextLake.store` de los tres adaptadores también asigna la caché antes de escribir (Hermes líneas 235–236; OpenCode líneas 360–361; Pi líneas 209–211), y en cada uno se observó el mismo comportamiento.

**No demostrado.** Qué fallos de escritura reales (disco lleno, permisos, errores de codificación) ocurren en la práctica. El disparador de esta prueba es artificial.

**Impacto.**

- Bajo, porque el error llega al llamante.
- Después, `get`, `search` y `stats` presentan en esa sesión la entrada como guardada hasta que se recarga el lake.

**Dirección sugerida.** Actualizar la caché en memoria solo después de que la escritura tenga éxito, o revertirla si falla.

### 7.5 F-5 — la selección del lake depende de la cadena de ruta del proyecto

- **Prioridad · tipo:** P3 · Baja · limitación
- **Nivel de evidencia:** 3 · requiere verificación en la integración con el host. El comportamiento de la función se observó directamente; no se observó si los editores pasan rutas de proyecto no canónicas.
- **Versiones:** rlm-agent `5178d62` · Python 3.13.15 · macOS 26.6.2 arm64
- **Ubicación:** `adapters/hermes/__init__.py`, `_lake_file_for` (línea 71).
  - El nombre del archivo son los primeros 16 dígitos hexadecimales del `sha256` de la cadena de ruta (línea 73).
  - `lakeFileFor` usa la misma fórmula en `adapters/opencode/rlm.ts` (líneas 72–73) y en `adapters/pi/rlm.ts` (líneas 51–52).
- **Reproducción mínima:** `python3 REPRO/f5_path_string_lake.py <checkout>`
- **Evidencia:** `artifacts/f5_path_string_lake.txt`

**Esperado.** No documentado más allá de que el lake es "per-project" (README.md línea 57). Suposición nuestra: a un directorio le corresponde un lake.

**Observado.** `_lake_file_for` devolvió **tres archivos de lake distintos** para un mismo directorio: su ruta simple, la misma ruta con barra final y un enlace simbólico a él. Las tres resuelven al mismo directorio real.

*Observación de apoyo (arnés):* con el código de los adaptadores de Hermes, Pi y OpenCode, las variantes con barra final y con enlace simbólico buscaron cada una un lake distinto, así que no encontraron las entradas guardadas bajo la ruta simple.

**No demostrado.** Si los editores pasan alguna vez rutas no canónicas.

**Impacto, si lo hacen.** Las entradas anteriores parecen haber desaparecido cuando se accede al mismo proyecto con otra escritura de la ruta: un espacio de trabajo enlazado simbólicamente, un separador final o una ruta de montaje distinta.

**Dirección sugerida.** Normalizar el directorio del proyecto antes de calcular el hash: resolver los enlaces simbólicos y quitar los separadores finales. Los archivos de lake existentes necesitarían una vía de migración.

### 7.6 P-3 — diferencias de implementación (informativo)

- **Prioridad · tipo:** P3 · Baja · diferencia
- **Nivel de evidencia:** 1 · confirmado en el código/runtime probado (`rlm_lake` del kernel y `ContextLake` de Hermes)
- **Versiones:** rlm-agent `5178d62` · Python 3.13.15 · macOS 26.6.2 arm64
- **Ubicación:**
  - `kernel/kernel.py`: `_LakeBridge.search` solo compara contenido y clave (línea 407), y `store` escribe segundos de `time.time()` (línea 381).
  - `adapters/hermes/__init__.py`: `ContextLake.search` también compara las etiquetas (línea 251), y `store` escribe milisegundos (línea 230).
  - Solo según el código: OpenCode y Pi también comparan las etiquetas (`adapters/opencode/rlm.ts` línea 380; `adapters/pi/rlm.ts` línea 230) y escriben milisegundos con `Date.now()` (líneas 351 y 200).
- **Reproducción mínima:** `python3 REPRO/p3_kernel_vs_adapter_differences.py <checkout>`
- **Evidencia:** `artifacts/p3_kernel_vs_adapter_differences.txt`

**Esperado.** No documentado. Suposición nuestra: las implementaciones que hay detrás de las mismas herramientas se comportan igual.

**Observado.** Se guardó la misma entrada etiquetada mediante ambas implementaciones:

| comprobación | `rlm_lake` del kernel | `ContextLake` de Hermes |
|---|---|---|
| `search` de un valor presente solo en las etiquetas | 0 coincidencias | 1 coincidencia |
| `created` escrito en el archivo del lake | `1789123937.747145` (segundos, float) | `1789123937761` (milisegundos, int) |

**No demostrado.** Si la unidad del timestamp causa errores visibles. Por ejemplo, la cabecera de `rlm_get` en OpenCode formatea `updated` como fecha (`adapters/opencode/rlm.ts` línea 804); no se probó con una entrada escrita por el kernel.

**Impacto.** Los resultados difieren según se consulte el lake desde el kernel o mediante una herramienta, y las entradas escritas por el kernel llevan timestamps en otra unidad.

**Dirección sugerida.** Alinear la semántica de búsqueda y las unidades de timestamp entre implementaciones, o documentar la diferencia.

---

## 8. Metodología de reproducción

- **Un script por hallazgo; F-1 tiene dos.**
  - Cada script imprime la versión del runtime y expresa su resultado en líneas `RESULT:`.
  - Los scripts que cargan rlm-agent imprimen también su commit y el sha256 de los archivos que ejercitan.
  - B-1 y B-2 imprimen una línea `RESULT:` por caso y una línea `SUMMARY:` final.
- **Runtime separado del adaptador (F-1).** F-1a no carga rlm-agent; F-1b ejecuta la propia herramienta `rlm_store` del adaptador.

- **Escritores concurrentes (F-2, F-3).** Un proceso separado del sistema operativo actúa como el otro escritor.
- **Controles.** Se enumeran en §9.
- **Aislamiento.**
  - Los scripts escriben solo en directorios temporales.
  - `HOME` se redirige allí donde el código del adaptador deriva rutas a partir de él.
  - El adaptador de OpenCode se ejecuta desde una copia idéntica byte a byte junto a un stub mínimo de su paquete del host.
  - No se lee ni escribe ninguna configuración de editores.
- **Evidencia de referencia.** Los scripts y sus salidas. Los resultados del arnés aparecen solo como observaciones de apoyo etiquetadas.

## 9. Controles de calidad y correcciones

**Proceso de revisión.**

- Un único revisor diseñó las pruebas, escribió ambos caminos de código y redactó este informe.
- Ningún segundo revisor independiente comprobó los hallazgos, y ninguna otra parte hizo una revisión adversarial.
- Los controles siguientes reducen el riesgo de error, pero no sustituyen esa independencia. Los hallazgos se apoyan en scripts que cualquiera puede volver a ejecutar.

**Controles y comparaciones dentro de los scripts.**

| script | control o comparación | resultado en nuestra ejecución |
|---|---|---|
| F-2 | los mismos dos escritores sin el bucle de `forget`, 3 ejecuciones | ninguna entrada perdida en ninguna ejecución |
| F-1a, F-1b | sonda del runtime separada de la prueba del adaptador | ambas reproducen |
| B-1 | una celda `%%bash` en el mismo kernel | la salida lleva el id de la petición y llega antes de `result` |
| B-2 | la expresión `1 + 2` en el mismo cliente | `→ 3` |
| F-4 | un proceso nuevo del kernel lee el mismo archivo del lake | la entrada fallida no está en disco; la anterior sí |
| F-5 | cada variante de la ruta se resuelve a su directorio real | las tres son el mismo directorio |
| D-1 | el contexto de registro acepta cualquier nombre de método | una llamada de registro inesperada también habría quedado registrada |

F-1a también se ejecutó una vez fuera del entorno restringido, en la misma máquina, con el mismo resultado. Esa salida no se incluye.

**Correcciones hechas durante la revisión.**

1. **Error de codificación en el arnés.**
   - La primera versión del arnés incrustaba el contenido de prueba en código Python generado usando escape JSON.
   - Con un carácter fuera del Plano Multilingüe Básico (un emoji), eso produjo code points sustitutos sin pareja, y el kernel no pudo escribir la entrada en el archivo del lake.
   - La comprobación de ida y vuelta del arnés leyó el valor desde la misma sesión, que aún lo servía desde memoria, y registró un falso aprobado.
   - Se corrigió la incrustación y se volvió a ejecutar la batería. Investigar ese falso aprobado destapó F-4.
2. **Script de D-1.** Su primera versión solo inspeccionaba los nombres a nivel de módulo e informaba de que `_on_compact` no estaba definida. La función está definida dentro de `register()` y nunca se registra. El script se corrigió, y la salida incluida procede del script corregido.

3. **Resultados confundidos excluidos.** Los resultados de concurrencia del arnés para el adaptador de OpenCode quedaron dominados por F-1, así que no se usan como evidencia de F-2 ni de F-3.

## 10. Evidencia e integridad

- **Salidas de los scripts.** `artifacts/*.txt` contiene la salida de cada script en nuestra ejecución; `artifacts/EXIT-CODES.txt` enumera los códigos de salida.
- **Integridad.** `EVIDENCE-MANIFEST.sha256` cubre 26/26 archivos del paquete, es decir, todos salvo el propio manifiesto.
  - Una comprobación correcta demuestra que los archivos no han cambiado desde que se empaquetaron.
  - No valida los hallazgos: para eso hay que volver a ejecutar los scripts.
- **No incluido.** El arnés y sus registros, que contienen detalles específicos del entorno.

## 11. Limitaciones

- **No es a nivel de editor.** OpenCode, Pi y Hermes no se instalaron; los adaptadores se ejercitaron fuera de sus hosts con stubs mínimos.
- **Cobertura de plataforma estrecha.** Una plataforma (macOS arm64), una versión de Bun (1.4.0) y una versión de Python (3.13.15).
- **Pruebas de concurrencia pequeñas.** Establecen el mecanismo, no con qué frecuencia ocurre en uso real.
- **Sin sesiones dirigidas por un LLM.** Las herramientas se llamaron directamente.
- **Un solo commit.** Los resultados aplican solo al commit `5178d62`.
- **Un único revisor.** Ver §9.
- **Expectativas supuestas.** Cuando el comportamiento no está documentado (F-4, F-5, P-3), el comportamiento esperado es una suposición nuestra, y el hallazgo lo indica.
- **Restricción de red parcial.** No cubrió el proceso de Bun; ninguna prueba necesitaba red.

## 12. Sugerencias para los mantenedores

Son direcciones posibles; otros diseños pueden encajar mejor en el proyecto.

1. **F-2 / F-3:** hacer que todas las implementaciones del lake recarguen de forma consistente, y serializar el ciclo de leer, modificar y escribir de `forget` (un bloqueo de archivo, o marcadores de borrado que solo se añaden y se compactan bajo bloqueo).
2. **F-1:** añadir las entradas del lake mediante una API documentada para escribir al final del archivo en el runtime de OpenCode, y añadir una prueba de ida y vuelta con dos entradas.
3. **B-1 / B-2:**
   - asignar `_current_cell` en cada petición y vaciar la salida capturada antes de `result`;
   - comprobar los ids de los eventos en `tests/test_kernel.py`;
   - tratar los eventos `stdout`/`stderr` en el cliente de kernel de Hermes.
4. **D-1:** registrar el callback de compactación de Hermes, o alinear el README y el docstring del adaptador con el comportamiento actual.
5. **F-4:** actualizar la caché en memoria solo después de que la escritura en el archivo tenga éxito.
6. **F-5:** normalizar la ruta del proyecto antes de convertirla, mediante el hash, en nombre de archivo del lake.
7. **P-3:** alinear, o documentar, la búsqueda sobre etiquetas y las unidades de timestamp entre implementaciones.

## 13. Qué NO se probó

- **Editores:** OpenCode, Pi y Hermes, ni ningún comportamiento a nivel de editor.
- **Versiones de Bun:** la versión de Bun que ejecuta OpenCode, ni `Bun.write` en ninguna versión distinta de la 1.4.0.
- **Uso dirigido por un LLM.**
- **Subagentes:** `rlm`, `rlm_result`, `rlm_list`, `rlm_delete`, ni los límites de recursión.

- **Estado y compactación:** `rlm_snapshot` / `rlm_restore`, ni la compactación en ejecución.
- **Manejo de salidas:** el «watch guard» de Hermes y la captura automática de salidas de OpenCode.
- **Instalación:** el instalador (`install.js`) y el actualizador (`update.js`).
- **Propiedades de seguridad** del kernel.
- **Cifras de rendimiento y de tokens** citadas en README.md.
- **Otros entornos:** otros sistemas operativos, ni otras versiones de Python o Node.

# Apéndice A — Guía de reproducción

## Guía de reproducción

Cada script de `REPRO/`:

- es autónomo: crea en línea cualquier stub mínimo que necesite y no usa ningún otro código de prueba;
- no necesita **red, credenciales, editor ni LLM**;
- se ejecuta sobre un **checkout limpio** de rlm-agent y nunca lo modifica;
- escribe solo en un directorio temporal nuevo, y apunta `HOME` a un directorio temporal allí donde el código del adaptador deriva rutas de `HOME`, así que no se lee ni escribe ninguna configuración real de editores;
- imprime la versión del runtime. Los scripts que cargan rlm-agent imprimen también su commit (si el checkout es un clon de git) y el sha256 de los archivos fuente que ejercitan;
- expresa su resultado en líneas `RESULT:`.

Un resultado reproducido no es una afirmación sobre el comportamiento a nivel de editor. El nivel de evidencia de cada hallazgo está en `FINDINGS.md`.

### Requisitos

| elemento | probado con | notas |
|---|---|---|
| checkout de rlm-agent | commit `5178d62ffece0679746ad3f7a14cd3aa4c30bf59` | los resultados describen solo este commit |
| Python | 3.13.15 | solo usa la biblioteca estándar |
| Bun | 1.4.0 (revisión `1381054db427439f5d669f28aa1984faa26d8e94`) | solo para los dos scripts de F-1 |
| sistema operativo | macOS 26.6.2, arm64 | no se probaron otras plataformas |
| usuario | no root | F-4 necesita que un archivo de solo lectura rechace escrituras |

### Preparación

```
git clone https://github.com/nicolasramos/rlm-agent
```

```
git -C rlm-agent checkout 5178d62ffece0679746ad3f7a14cd3aa4c30bf59
```

En los comandos siguientes, `rlm-agent` es el directorio del checkout y `REPRO/` es el directorio de este paquete.

### Comandos

Primero van los hallazgos principales.

| ID | comando | qué hace el script | control o comparación |
|---|---|---|---|
| F-2 | `python3 REPRO/f2_kernel_forget_race.py rlm-agent 3` | procesos del kernel que comparten un `RLM_LAKE_FILE`: dos escritores (100 `store` cada uno) más un bucle de `store` y `forget` (10 ciclos); 3 ejecuciones | los mismos dos escritores sin el bucle de `forget`; 3 ejecuciones |
| F-3 | `python3 REPRO/f3_hermes_stale_cache.py rlm-agent` | `ContextLake` de Hermes en este proceso más un segundo proceso de Python que escribe en el mismo archivo del lake; después, `forget` en la primera instancia | — |

| ID | comando | qué hace el script | control o comparación |
|---|---|---|---|
| F-1a | `bun REPRO/f1a_bun_write_append.ts` | llama a `Bun.write(file, "line-1\n", { append: true })`, luego lo mismo con `"line-2\n"`, y lee el archivo; no carga rlm-agent | separado de la prueba del adaptador (F-1b) |
| F-1b | `bun REPRO/f1b_opencode_lake_store.ts rlm-agent` | carga una copia de `adapters/opencode/rlm.ts` con un stub mínimo de `@opencode-ai/plugin` (`tool(def)` devuelve `def`), llama tres veces a la herramienta real `rlm_store` en un proceso hijo de Bun cuyo `HOME` es un directorio temporal, y después lee el archivo del lake | — |
| B-1 | `python3 REPRO/b1_kernel_stdout_attribution.py rlm-agent` | envía `print(...)` tres veces e imprime la secuencia de eventos | una celda `%%bash` |
| B-2 | `python3 REPRO/b2_hermes_kernel_client_output.py rlm-agent` | `Kernel` y `_format_execute` del adaptador de Hermes sobre `print` y sobre una celda `%%bash` | la expresión `1 + 2` |
| D-1 | `python3 REPRO/d1_hermes_registered_hooks.py rlm-agent` | `register(ctx)` con un objeto de contexto que registra cada llamada a método | el registrador acepta cualquier nombre de método |
| F-4 | `python3 REPRO/f4_kernel_write_failure.py rlm-agent` | `store`, poner el archivo del lake en solo lectura, `store` de nuevo y `get` en el mismo kernel | un kernel nuevo lee el mismo archivo del lake |
| F-5 | `python3 REPRO/f5_path_string_lake.py rlm-agent` | `_lake_file_for` con una ruta, la misma ruta con barra final y un enlace simbólico | cada variante se resuelve a su directorio real |
| P-3 | `python3 REPRO/p3_kernel_vs_adapter_differences.py rlm-agent` | la misma entrada etiquetada, guardada mediante el kernel y mediante el `ContextLake` de Hermes | — |

## Líneas de resultado y códigos de salida

| script | defecto o discrepancia reproducido | no reproducido |
|---|---|---|
| F-1a, F-1b, F-3 | `RESULT: FAIL …`, código de salida 1 | `RESULT: PASS …`, código de salida 0 |
| F-4 | `RESULT: FAIL …`, código de salida 1 | `RESULT: PASS`, código de salida 0; `RESULT: INCONCLUSIVE …` si el `store` no falló (por ejemplo, al ejecutarlo como root) |
| B-1, B-2 | un `RESULT: FAIL` por cada caso afectado y después una línea `SUMMARY:`; código de salida 1 | todos los casos `RESULT: PASS`; código de salida 0 |
| F-2 | `RESULT: REPRODUCED`, código de salida 1 | `RESULT: NOT REPRODUCED`, código de salida 0; `RESULT: INCONCLUSIVE` si una ejecución de control también pierde entradas |
| D-1 | `RESULT: DISCREPANCY …`, código de salida 1 | `RESULT: CONSISTENT`, código de salida 0 |
| F-5 | `RESULT: … LIMITATION OBSERVED`, código de salida 0 | `RESULT: … NOT OBSERVED`, código de salida 0 |
| P-3 | `RESULT: informational …`, código de salida 0 | — |

**Cómo leer B-1 y B-2.** Se espera que sus casos de comparación y de control impriman `RESULT: PASS`. El código de salida es 1 cuando falla cualquier otro caso.

**Dependencia del tiempo.** F-2 depende del tiempo. El número de entradas perdidas varía entre ejecuciones, y en otras máquinas una ejecución podría no perder ninguna. El escenario de control separa el efecto de `forget` del de los añadidos concurrentes por sí solos.

### Cómo se produjeron las salidas incluidas

Las salidas de `artifacts/` proceden de ejecutar los comandos anteriores sobre el commit probado.

**Entorno:**

- un entorno mínimo (`env -i`) con `HOME` y `TMPDIR` apuntando a un directorio de trabajo temporal;
- acceso a red denegado a los procesos de Python a nivel del sistema operativo (el método usado no cubre Bun; ningún script necesita red).

**Códigos de salida:** se enumeran en `artifacts/EXIT-CODES.txt`.

**Integridad:** `EVIDENCE-MANIFEST.sha256` cubre 26/26 archivos del paquete, es decir, todos salvo el propio manifiesto. Una comprobación correcta demuestra que los archivos no han cambiado desde que se empaquetaron. No valida los hallazgos; para eso hay que volver a ejecutar los scripts.

**Verificar el paquete:**

```
shasum -a 256 -c EVIDENCE-MANIFEST.sha256
```

# Apéndice B — Salidas de los scripts en nuestra ejecución

Contenido literal de `artifacts/`. Son salidas de programas y se reproducen sin traducir.

## artifacts/EXIT-CODES.txt

```
b1_kernel_stdout_attribution exit=1
b2_hermes_kernel_client_output exit=1
d1_hermes_registered_hooks exit=1
f1a_bun_write_append exit=1
f1b_opencode_lake_store exit=1
f2_kernel_forget_race exit=1
f3_hermes_stale_cache exit=1
f4_kernel_write_failure exit=1
f5_path_string_lake exit=0
p3_kernel_vs_adapter_differences exit=0
```

## artifacts/b1_kernel_stdout_attribution.txt

```
rlm-agent functional review — b1_kernel_stdout_attribution.py
python 3.13.15 — Darwin arm64 — rlm-agent commit 5178d62ffece0679746ad3f7a14cd3aa4c30bf59
sha256 c4830814df5d79b3906456f5d98369fdc4768c14e33eb766409c88e125e32100  kernel/kernel.py

[print #1] request id: req-1
  event sequence (event, id): [('result', 'req-1'), ('stdout', None), ('done', 'req-1')]
  stdout event at position 1: id=None text='CANARY-PRINT\n'
  RESULT: FAIL

[print #2] request id: req-2
  event sequence (event, id): [('result', 'req-2'), ('stdout', None), ('done', 'req-2')]
  stdout event at position 1: id=None text='CANARY-PRINT\n'
  RESULT: FAIL

[print #3] request id: req-3
  event sequence (event, id): [('result', 'req-3'), ('stdout', None), ('done', 'req-3')]
  stdout event at position 1: id=None text='CANARY-PRINT\n'
  RESULT: FAIL

[%%bash (comparison)] request id: req-4
  event sequence (event, id): [('stdout', 'req-4'), ('result', 'req-4'), ('done', 'req-4')]
  stdout event at position 0: id='req-4' text='CANARY-BASH\n'
  RESULT: PASS

SUMMARY: print PASS 0/3 — %%bash PASS
```

## artifacts/b2_hermes_kernel_client_output.txt

```
rlm-agent functional review — b2_hermes_kernel_client_output.py
python 3.13.15 — Darwin arm64 — rlm-agent commit 5178d62ffece0679746ad3f7a14cd3aa4c30bf59
sha256 67a3914f6e9b4e5cdd681ea3b33315d26c5f42f37829f538da5b994ee25c6cfa  adapters/hermes/__init__.py
sha256 c4830814df5d79b3906456f5d98369fdc4768c14e33eb766409c88e125e32100  kernel/kernel.py

[print] code='print("CANARY-PRINT")'
  text returned by the adapter: '→ None'
  execute() returned stdout field: ''
  RESULT: FAIL

[%%bash] code="%%bash\nprintf 'CANARY-BASH\\n'"
  text returned by the adapter: '→ exit code 0'
  execute() returned stdout field: ''
  RESULT: FAIL

[expression (control)] code='1 + 2'
```

```
  text returned by the adapter: '→ 3'
  execute() returned stdout field: ''
  RESULT: PASS

SUMMARY: print FAIL — %%bash FAIL — control PASS
```

## artifacts/d1_hermes_registered_hooks.txt

```
rlm-agent functional review — d1_hermes_registered_hooks.py
python 3.13.15 — Darwin arm64 — rlm-agent commit 5178d62ffece0679746ad3f7a14cd3aa4c30bf59
sha256 67a3914f6e9b4e5cdd681ea3b33315d26c5f42f37829f538da5b994ee25c6cfa  adapters/hermes/__init__.py
sha256 9668ba662b239dbd880bc87b1a08bfb395791e5f839eae3a973b6ed0cd5029ec  README.md

tools registered (12): ['ipython', 'rlm_store', 'rlm_get', 'rlm_search', 'rlm_find', 'rlm_stats',
'rlm_forget', 'rlm_snapshot', 'rlm_restore', 'rlm', 'rlm_result', 'rlm_list']
hooks registered: ['on_session_end', 'transform_tool_result']
other registration calls: [('register_system_prompt_section', 'rlm')]
functions defined inside register(): ['<lambda>', '_ipython', '_on_compact', '_rlm', '_rlm_find',
'_rlm_forget', '_rlm_get', '_rlm_list', '_rlm_restore', '_rlm_result', '_rlm_search', '_rlm_snapshot',
'_rlm_stats', '_rlm_store']
_on_compact defined inside register(): True
compaction / context-engine registration calls: []
README mentions register_context_engine: True

RESULT: DISCREPANCY (README describes a compaction hook that register() does not register)
```

## artifacts/f1a_bun_write_append.txt

```
rlm-agent functional review — f1a_bun_write_append.ts
bun 1.4.0 (revision 1381054db427439f5d669f28aa1984faa26d8e94) — darwin arm64
calls: Bun.write(file, "line-1\n", { append: true }); Bun.write(file, "line-2\n", { append: true })
file content after both calls: "line-2\n" (1 line(s))
RESULT: FAIL (only the last write is present: the append option was not honored)
```

## artifacts/f1b_opencode_lake_store.txt

```
rlm-agent functional review — f1b_opencode_lake_store.ts
bun 1.4.0 — darwin arm64 — rlm-agent commit 5178d62ffece0679746ad3f7a14cd3aa4c30bf59
sha256 55490fee09b9928b3eee264b4ec0a68def4374145380ea120994c3e8453ea36a  adapters/opencode/rlm.ts

rlm_store replies returned to the model:
  Stored "entry-1" (18 chars, tags: none). It is NOT in the LLM context — retrieve with rlm_get /
rlm_search / rlm_find.
  Stored "entry-2" (18 chars, tags: none). It is NOT in the LLM context — retrieve with rlm_get /
rlm_search / rlm_find.
  Stored "entry-3" (18 chars, tags: none). It is NOT in the LLM context — retrieve with rlm_get /
rlm_search / rlm_find.

rlm_stats: "Context lake: 1 entries, 18 chars total.\nKeys:\n- entry-3"
lake files: 1 — keys in lake file: ["entry-3"]

RESULT: FAIL (1 of 3 entries remain in the lake file)
```

## artifacts/f2_kernel_forget_race.txt

```
rlm-agent functional review — f2_kernel_forget_race.py
python 3.13.15 — Darwin arm64 — rlm-agent commit 5178d62ffece0679746ad3f7a14cd3aa4c30bf59
sha256 c4830814df5d79b3906456f5d98369fdc4768c14e33eb766409c88e125e32100  kernel/kernel.py

  with forget run 1: lost 24/200 A/B entries — invalid JSONL lines 0 — C entries left 0/10 — cell errors []
  with forget run 2: lost 27/200 A/B entries — invalid JSONL lines 0 — C entries left 0/10 — cell errors []
  with forget run 3: lost 35/200 A/B entries — invalid JSONL lines 0 — C entries left 0/10 — cell errors []
```

```
  control     run 1: lost 0/200 A/B entries - invalid JSONL lines 0 - cell errors []
  control     run 2: lost 0/200 A/B entries - invalid JSONL lines 0 - cell errors []
  control     run 3: lost 0/200 A/B entries - invalid JSONL lines 0 - cell errors []

RESULT: REPRODUCED
```

## artifacts/f3_hermes_stale_cache.txt

```
rlm-agent functional review - f3_hermes_stale_cache.py
python 3.13.15 - Darwin arm64 - rlm-agent commit 5178d62ffece0679746ad3f7a14cd3aa4c30bf59
sha256 67a3914f6e9b4e5cdd681ea3b33315d26c5f42f37829f538da5b994ee25c6cfa  adapters/hermes/__init__.py

1-2. keys in lake file after P1 store and P2 store: ['seed', 'from-p2']
3.   P1.get('from-p2') finds the entry: False
4.   P1.forget('^seed$') removed: 1
5.   keys in lake file after P1 forget: []

RESULT: FAIL (entry written by P2 was removed by P1 forget)
```

## artifacts/f4_kernel_write_failure.txt

```
rlm-agent functional review - f4_kernel_write_failure.py
python 3.13.15 - Darwin arm64 - rlm-agent commit 5178d62ffece0679746ad3f7a14cd3aa4c30bf59
sha256 c4830814df5d79b3906456f5d98369fdc4768c14e33eb766409c88e125e32100  kernel/kernel.py

K1 store('ok-entry'):              ('ok', {'key': 'ok-entry', 'chars': 2, 'tags': []})
K1 store('failed-entry') read-only: ('error', "PermissionError: [Errno 13] Permission denied:
'<tmp>/lake.jsonl'")
K1 get('failed-entry'):            ('ok', 'NEVER-WRITTEN')
K1 stats():                        ('ok', {'entries': 2, 'chars': 15, 'keys': ['failed-entry', 'ok-
entry']})
K2 get('failed-entry'):            ('ok', None)
K2 get('ok-entry'):                ('ok', 'OK')

RESULT: FAIL (the failed store is still served by the same session but is not on disk)
```

## artifacts/f5_path_string_lake.txt

```
rlm-agent functional review - f5_path_string_lake.py
python 3.13.15 - Darwin arm64 - rlm-agent commit 5178d62ffece0679746ad3f7a14cd3aa4c30bf59
sha256 67a3914f6e9b4e5cdd681ea3b33315d26c5f42f37829f538da5b994ee25c6cfa  adapters/hermes/__init__.py

  directory path         -> lake file 0a2dfc7b0ee7e76b.jsonl   (same real directory: True)
  trailing slash         -> lake file 3f292abce7bb2de6.jsonl   (same real directory: True)
  symlink to directory   -> lake file cdb94e317b3e4e57.jsonl   (same real directory: True)

RESULT: 3 distinct lake file(s) for 1 real directory (LIMITATION OBSERVED)
```

## artifacts/p3_kernel_vs_adapter_differences.txt

```
rlm-agent functional review - p3_kernel_vs_adapter_differences.py
python 3.13.15 - Darwin arm64 - rlm-agent commit 5178d62ffece0679746ad3f7a14cd3aa4c30bf59
sha256 c4830814df5d79b3906456f5d98369fdc4768c14e33eb766409c88e125e32100  kernel/kernel.py
sha256 67a3914f6e9b4e5cdd681ea3b33315d26c5f42f37829f538da5b994ee25c6cfa  adapters/hermes/__init__.py

1. search for a value present only in tags
   kernel rlm_lake.search        -> 0 match(es)
   hermes ContextLake.search     -> 1 match(es)

2. `created` value written to the lake file
   kernel rlm_lake.store         -> 1789123937.747145 (float)
   hermes ContextLake.store      -> 1789123937761 (int)
```

```
RESULT: informational — tags searched: kernel=False, hermes=True; timestamp magnitude: kernel~1e9,
hermes~1e12
```

## Apéndice C — Hashes de los archivos al generar este PDF

La lista completa de archivos del paquete, con sus hashes, está en `EVIDENCE-MANIFEST.sha256`.

| archivo | sha256 |
|---|---|
| REPRO/b1_kernel_stdout_attribution.py | 8755cb0e800825e25096ee3f651c44cce6434520db776fac1e8ad1d2d2163e2e |
| REPRO/b2_hermes_kernel_client_output.py | de8173adb0c6cb4959875d73a5209f2380d7b39dec0ab6458b0d989aebaa4376 |
| REPRO/d1_hermes_registered_hooks.py | df7bd3ef27f8e1366a82cea36be0d603de6df3b758da9b139ccdb5faa4938114 |
| REPRO/f1a_bun_write_append.ts | 103850ed69a7d4bdc088fe7f89dba4d648d2085a73e22fa3d2c4dc6f9c9e0386 |
| REPRO/f1b_opencode_lake_store.ts | d6578756ea8bc52147a5e0faec97202ac22b4603a2fb0a2439fbdba4de1dcc2b |
| REPRO/f2_kernel_forget_race.py | 296bc3d8142ec231394c23006b8045561b704e3812bbb245fffb4337f3ab0e57 |
| REPRO/f3_hermes_stale_cache.py | d5ec3e2d082e2cb58aeb997fb87a96bdccbb5cdc8e6a576b1d3dc7bf44c38361 |
| REPRO/f4_kernel_write_failure.py | 59a38e6012dbc9cda410f17b652b37dc0e98b229a5683d704d01ad09df2200c0 |
| REPRO/f5_path_string_lake.py | a88ce640ac06f8ebc60a7ea2d8447bf6a4137706270f81c6beb838eaeb8a27e9 |
| REPRO/p3_kernel_vs_adapter_differences.py | 22f148125b6665c6e4186822200dca15bddb7519ded596173fa2972baf20de76 |
| artifacts/EXIT-CODES.txt | 18bc3a1508898ce587788d63e07d3c290cc8a590345834a8ee6f4b0546876867 |
| artifacts/b1_kernel_stdout_attribution.txt | fe468029cf5e88e55ebe04e0a677aab5ab9498417c92b04b47910a56849bf537 |
| artifacts/b2_hermes_kernel_client_output.txt | 8fca4b35d57c7a8b2b27235c79aa05a0c9e57c77d108c4a4c3d9c3b0374a5d1e |
| artifacts/d1_hermes_registered_hooks.txt | eaa71d4e429fb4dab829e53d4581aec0ba9974dfd3da2faca41afeca9ab7d46c |
| artifacts/f1a_bun_write_append.txt | 0d67fa801d59e82affd8879755dc21b8c99b663d83bd4c63e20b7f4ac309e14f |
| artifacts/f1b_opencode_lake_store.txt | 902a8da83c21de0613ed1a70bb7ff6c3250a349c76ef6f265669364512b36192 |
| artifacts/f2_kernel_forget_race.txt | eef1fd10754360763a10b997b2c899f7e550003431ee16d985e9909fb0a409ef |
| artifacts/f3_hermes_stale_cache.txt | 145e24f190b0026646fe9ea576e84fbd2c263dbb94b30d4b460f2b8d5f2aeab8 |
| artifacts/f4_kernel_write_failure.txt | 5e1e3c279e99ae6690f7d7b934e88fd22c0afe6c8f77db37325bbb8edb3261be |
| artifacts/f5_path_string_lake.txt | e1091c488075a1e5aaae1a9efd972dca68b4594b4d87bee40d2f4c5294ae3869 |
| artifacts/p3_kernel_vs_adapter_differences.txt | 348c34393cf1ef1a6f109c99e558658a1b6cceb459e1f5c190976e553143ce65 |