mc.js force-pending Auto-Expiry Sweep (MC #105894)

mc.js force-pending Auto-Expiry Sweep (MC #105894)

MC: #105894

Parent system: mc.js Force Approval Queue (MC #100818)
Builder: CodeCraft / Kleppmann (agent id kleppmann-105894)
Independent verifier: Proveo (agent id proveo-verify-105894)
Date shipped: 2026-07-20
Code: `~/system/tools/mc.js` lines 1765-1901 (sweep function), plus hook in the `force-pending` listing command and a new `force-sweep` command
Verdict: PASS (builder) + PASS (independent Proveo peer-verify) — 34/34 builder assertions, 47/47 independent assertions, live smoke tested

---

Problem

The force-pending approval queue (`~/system/state/force-pending.jsonl`, see mc.js Force Approval Queue (MC #100818)) accumulates entries whenever an agent runs `mc.js done --force`. Entries carry a 24h `expires_at` TTL, but nothing actually removed expired entries from the live queue — they just sat there forever, marked expired only in the *display* logic. By 2026-07-17 this had grown to 382 dead `pending_ceo_approval` records, requiring a manual cleanup. MC #105894 implements an automated, safe sweep so this does not recur.

What changed

File: `~/system/tools/mc.js`. Additive diff only — no existing function was modified, no other command's behavior changed. Pre-edit backup: `~/system/evidence/105894/mc.js.bak-pre-sweep-20260720` (sha256 `11b0d2ca1dbb42d2fa42ce4c2318b2d38a3c06f3eb1ae4b420a64e9a9d401e67`, independently re-verified by Proveo). Full diff: `~/system/evidence/105894/mc.js.diff-20260720`.

1. **`_forceSweepAtomicWrite(targetPath, content)`** (mc.js:1765) — tmp-file + fsync + rename atomic write helper, dedicated to the sweep (kept separate from the pre-existing `_atomicWrite` used by `start`, to avoid touching unrelated code).
2. **`sweepExpiredForcePending()`** (mc.js:1778-1901) — the sweep itself. Returns `{ swept, anomalies, skipped_locked, error }`. Never throws to the caller.
3. **`force-pending` listing command** — now calls the sweep first and always prints `auto-sweep: swept N ...` before showing the queue, plus any anomaly warnings. No silent mutation.
4. **New `force-sweep` command** (alongside `force-approve` / `force-deny`) — explicit standalone entry point; always prints the swept count, even when zero.
5. `sweepExpiredForcePending` added to the `module.exports.__test` seam (same pattern as MC #105599) for isolated testability. Not otherwise reachable outside the CLI.

Commands

Run an explicit sweep

```
node ~/system/tools/mc.js force-sweep
```

Output (nothing to sweep):

```
force-sweep: swept 0 expired pending_ceo_approval entries to force-pending-archive.jsonl
```

Output (entries swept):

```
force-sweep: swept 1 expired pending_ceo_approval entry to force-pending-archive.jsonl
```

Auto-sweep on listing (no separate step needed)

```
node ~/system/tools/mc.js force-pending
```

```
auto-sweep: swept 0 expired entries to force-pending-archive.jsonl
=== FORCE-PENDING QUEUE (P1.1 Reality Anchor) ===
Pending CEO approval: 4 | Expired: 0 | Processed: 159
[... remaining pending entries listed ...]
```

Every `force-pending` listing runs the sweep first and always prints the swept count — there is no code path where the queue is displayed without the sweep having run, and no code path where a sweep happens silently.

The 7 safeguards

Defined by Pi's independent verification doc (`~/system/evidence/105894/pi-queue-cleanup-verify-and-sweep-safeguards-2026-07-17.md`) after the 17.07 incident, and implemented exactly as specified. All 7 were independently re-confirmed by Proveo by reading the shipped code directly (mc.js:1765-1901), not by trusting the builder's description.

#	Safeguard	Implementation
1	Lock across the full cycle	`queuePath + '.sweep.lock'`, exclusive-create (`fs.openSync(lockPath, 'wx')`) held for read→classify→archive→rewrite, released in a `finally`. Stale-lock reclaim at 15s (mirrors the existing `appendBypassAttempt` lock idiom at mc.js:1732, but uses `wx` instead of stat-then-write to avoid that idiom's own TOCTOU gap).
2	Only pending + valid-expired swept	Explicit `status === 'pending_ceo_approval'` check, plus a valid parseable `expires_at <= now`, before anything is touched. Any other status passes through untouched.
3	Processed statuses never moved	`ceo_approved` / `consumed` / `ceo_denied` entries are unconditionally pushed back into `keptQueueLines`, never inspected for expiry, never written to the archive by this function.
4	Fail closed on invalid `expires_at`	Missing or unparseable `expires_at` is recorded into `result.anomalies` AND the entry stays in the live queue. It is never swept. Surfaced to the caller and printed by both `force-pending` and `force-sweep`.
5	Audit stamp on archive	`archived_at` + `archive_reason: 'auto_expiry_sweep'` stamped on every archived entry via `Object.assign`.

#	Safeguard	Implementation
6	Idempotent by `queue_id`	Existing archive `queue_id`s are loaded into a `Set` before appending; only entries not already present are appended. The queue rewrite still drops *all* `toArchive` entries (not just newly-appended ones), so a partially-completed prior run (archived but not yet removed from the queue) finishes correctly on re-run without a duplicate archive row. See known edge case below.
7	Atomic writes, archive before queue	Both files written via `_forceSweepAtomicWrite` (tmp file → fsync → rename). The archive is durably fsynced *before* the queue rewrite begins, so an interruption between the two steps leaves the entry safely duplicated into "archived and still in queue" — which safeguard 6 cleans up on the next run. Never data loss.

Operations

Files

- `~/system/state/force-pending.jsonl` — live queue (pending / approved / denied entries)
- `~/system/state/force-pending-archive.jsonl` — archive; swept entries land here with `archive_reason: 'auto_expiry_sweep'` and an `archived_at` timestamp
- `~/system/state/force-pending.jsonl.sweep.lock` — transient lock file, held only for the duration of a sweep cycle; auto-reclaimed if stale >15s. If you see this file outside of an active sweep, check for a crashed process before manually deleting it.

What an anomaly warning means

If `force-sweep` or the auto-sweep on `force-pending` prints an anomaly, it means an entry has `status=pending_ceo_approval` but a missing or unparseable `expires_at` field. The sweep deliberately leaves these entries untouched in the live queue (fail-closed — safeguard 4) rather than guessing whether they're expired. Investigate the flagged `queue_id` manually; do not assume it is safe to force-remove.

How to recover if something looks wrong

1. Do not hand-edit `force-pending.jsonl` or the archive while a sweep might be running — check for `force-pending.jsonl.sweep.lock` first.
2. Every live sweep operation described in the shipped verification took a pre-write sha256-verified backup. Follow the same pattern for any manual recovery: copy both files, sha256sum them, then act.
3. Reference backups from the 2026-07-20 rollout (useful as known-good comparison points, not for restore of current state): `~/system/evidence/105894/force-pending.jsonl.bak-pre-live-sweep-20260720T0810Z` and `~/system/evidence/105894/force-pending-archive.jsonl.bak-pre-live-sweep-20260720T0810Z`.
4. If the queue looks corrupted (malformed JSON lines), note that the sweep is designed to tolerate this — malformed lines are preserved verbatim, not dropped or crashed on (verified adversarially, see IND-3/IND-9 below). A crash during listing points elsewhere in the code, not at the sweep.

Known edge case (non-blocking, tracked as follow-up)

Safeguard 6 (idempotent by `queue_id`) is a no-op for any pending entry that has no `queue_id` field at all — e.g. a malformed/legacy row, or a hand-edited entry. If such a row is swept and then somehow reappears in the live queue (buggy re-enqueue, hand-edit, restore from an old backup), a second sweep will archive it again, producing a duplicate archive row for the same `task_id`. This is **not** a data-loss or live-queue-corruption issue — the entry is still correctly removed from the live queue either way — but it can leave a duplicate row in the archive. Every entry in production today has a `queue_id`, so this is not currently reachable through normal write paths. Tracked as follow-up **#106043** (fail-closed treat missing-`queue_id` as an anomaly, or fall back to a `task_id`+`expires_at` composite dedupe key).

Verification

Builder (Kleppmann) — 34/34 assertions, PASS. Tests ran against a byte-for-byte-identical extraction of the shipped function in a disposable fake-`HOME`, never touching live state:

- Test A — expired+valid mix across all statuses: correct sweep, correct exclusion of processed statuses.
- Test B — invalid `expires_at` (missing, garbage, far-future fixture): only the genuinely expired entry swept, invalid ones flagged as anomalies and left alone.
- Test C — interruption mid-sweep (archived but not yet removed from queue): re-run finishes cleanly, no duplicate archive row.
- Test D — concurrent/double invocation: second call correctly reports `skipped_locked: true` ; stale lock (>15s) correctly reclaimed.

Independent peer-verify (Proveo) — 47/47 assertions, PASS. Written from scratch, not derived from the builder's test script, run against fresh scratch `HOME` directories, spawned as real child processes (matching real `os.homedir()` resolution). Included everything the builder covered (re-derived independently) plus adversarial cases the builder did not test:

- Malformed JSON lines interleaved in the live queue file — no crash, malformed lines preserved verbatim.
- Pending entry with no `queue_id` at all — surfaced the known edge case above (Finding A), confirmed non-data-loss.
- TRUE concurrent race — two real child processes spawned simultaneously against the same scratch state (stronger than the builder's sequential lock-file simulation): no double-sweep, no duplicate archive `queue_id`s, no crash.
- Expiry boundary (`expires_at === now`): correctly swept per the coded `<=` comparison.
- Malformed line inside the *archive* file (not just the queue): dedupe-loading skips it gracefully, sweep still succeeds.
- No queue file / empty queue file: clean no-op, no crash.

Live smoke test (2026-07-20). Live state had drifted since the 17.07 manual cleanup (5 new pending entries had accumulated from normal operation by the time this shipped, one already expired). Rather than assuming a no-op, the actual state was checked first:

```
$ node ~/system/tools/mc.js force-sweep
force-sweep: swept 1 expired pending_ceo_approval entry to force-pending-archive.jsonl

$ node ~/system/tools/mc.js force-pending
  auto-sweep: swept 0 expired entries to force-pending-archive.jsonl
=== FORCE-PENDING QUEUE (P1.1 Reality Anchor) ===
Pending CEO approval: 4 | Expired: 0 | Processed: 159

$ node ~/system/tools/mc.js force-sweep     # idempotency re-check
force-sweep: swept 0 expired pending_ceo_approval entries to force-pending-archive.jsonl
```

Post-sweep: queue went from 164→163 lines (exactly the 1 expired entry removed, all processed-status counts unchanged); archive went from 386→387 lines (exactly 1 added, `archive_reason='auto_expiry_sweep'`, `task_id 105975`). All numbers independently recomputed by Proveo and matched exactly, including backup sha256 hashes.

Evidence

- `~/system/evidence/105894/kleppmann-sweep-2026-07-20.md` — builder verdict + full write-up
- `~/system/evidence/105894/proveo-peer-verify-2026-07-20.md` — independent Proveo verdict + full write-up
- `~/system/evidence/105894/mc.js.diff-20260720` — full unified diff

- `~/system/evidence/105894/mc.js.bak-pre-sweep-20260720` — pre-edit backup
- `~/system/evidence/105894/sweep-unit-tests-20260720.mjs` + output — builder's 34 assertions
- `/tmp/proveo-105894-verify/run-independent-tests.mjs` — Proveo's 47 independent assertions

Related

- **Parent MC:** #105894
- **Predecessor system:** mc.js Force Approval Queue (MC #100818) — defines the queue this sweep operates on
- **Follow-up:** #106043 — missing-`queue_id` dedupe gap (non-blocking, see Known Edge Case above)
- **Code:** `~/system/tools/mc.js` lines 1765-1901 (sweep + atomic-write helper), plus the `force-pending` command hook and new `force-sweep` command block

---

Revision #1
Created 2026-07-20 06:21:04 UTC by John
Updated 2026-07-20 06:21:04 UTC by John