CaliberVaultII — Build / Test / Publish / Verify Process

Overview

This document explains the end-to-end automation process used in the CaliberVaultII repository.
Two command-line tools handle the entire cycle:

- cvpipe — runs the overlay → install → test → publish → verify pipeline
- cvmemo — builds a Famous-ready memo using the results from the most recent run

Step 1 — Prepare the Overlay

Purpose: Replace local code with the latest drop from Famous.

```
OV=/Users/ghogue/Desktop/CaliberVaultII-From-Famous-Nov-9
REPO=/Users/ghogue/Desktop/CaliberVaultII
```

OV points to the unzipped overlay folder received from Famous.
REPO points to your active working repository.

Step 2 — Dependency Installation

Purpose: Install or refresh all NPM packages deterministically.
npm ci or npm install ensures consistent environment setup.

Step 3 — TypeScript Compilation

Purpose: Validate that the TypeScript codebase compiles before tests run.

Step 4 — Test Execution

Purpose: Run all Vitest suites and capture results.

Step 5 — Commit and Publish

Purpose: Push the tested state to the public mirror for Famous.

Step 6 — Verification

Purpose: Confirm the public repo contains the expected artifacts.

Step 7 — Build the Memo

Purpose: Generate a formatted Markdown memo that summarizes the run.

Step 8 — Famous Response Loop

Purpose: Famous confirms by replying with verification values and log lines.

Step 9 — Harness Policy Summary

vitest.setup.ts is not frozen. Famous may modify it each drop but must document changes.

Directory Layout Quick Reference

```
CaliberVaultII/
■■■ src/test/vitest.setup.ts
■■■ scripts/pipeline.sh (cvpipe)
■■■ scripts/famous-memo.sh (cvmemo)
■■■ test-artifacts/vitest.out.txt
■■■ test-artifacts/famous-memo..md
```

Typical Usage Flow

```
cvpipe # runs overlay → install → test → publish → verify
cvmemo # builds memo with hashes and status
```

Glossary

Overlay - code package received from Famous
Harness - the vitest.setup.ts environment stubs
Artifact - output files published to the public repo
Commit SHA - unique identifier of the public main branch after publishing
SHA-256 - file fingerprint verifying exact content identity