Workflow Quality Control System

Audit rubric, release pipeline, and intensive checklist — Gumroad-ready

July 14, 2026

Gumroad Release Pipeline
  Pipeline overview
  Stage 1 — Intake
  Stage 2 — Preflight (deterministic checks)
  Stage 3 — Intensive checklist + rubric audit
  Stage 4 — Fix loop
  Stage 5 — Independent verification
  Stage 6 — Package for Gumroad
  Stage 7 — Release record
  Deliverables checklist
Intensive Release Checklist
  A. Metadata & frontmatter
  B. Structure & file integrity
  C. Instruction quality
  D. Inputs & prerequisites
  E. Reliability & error handling
  F. Output definition
  G. Bundled code (`scripts/`) — skip section if no scripts
  H. Security, privacy & trust
  I. Portability & generality
  J. Buyer packaging (Gumroad)
  L. Spanish documentation (localization)
  K. Final gate
Workflow Quality Rubric
  1. Triggering (weight 3)
  2. Clarity (weight 2)
  3. Completeness (weight 3)
  4. Reliability (weight 3)
  5. Output definition (weight 2)
  6. Efficiency (weight 1)
  7. Safety & trust (weight 2)
  8. Generality (weight 2)
  Severity definitions for findings
Gumroad Packaging Specs
  Files
  Cover image
  What’s inside the product zip
  Listing fields to prepare (paste-ready in listing-copy.md)
Templates: Buyer README and Gumroad Listing Copy
  Buyer README template (ships inside the zip as 01-README.md)
  Gumroad listing copy template (listing-copy.md, paste into Gumroad)
  Spanish versions (01-README.es.md, listing-copy.es.md)
  Writing rules for both
Audit Report Template
Quality Audit: [workflow name]
  Scorecard
  What’s working
  Findings
    F1 · [Critical/Major/Minor] · [Dimension] — [one-line title]
    F2 · …
  Quick wins
  Re-audit trigger
Portfolio summary template (multi-workflow audits only)
Workflow Portfolio: Quality Summary
  Systemic patterns
  Suggested order of work

Gumroad Release Pipeline

Takes a workflow from “draft” to “ready to upload to Gumroad” through a release gate that nothing passes with known defects.

An honest note on “100% no errors”: no process can guarantee a workflow is perfect — what this pipeline guarantees is that **nothing ships with a detected error**. It combines deterministic machine checks (which catch mechanical defects with certainty) with a rubric audit and an independent verification pass (which catch judgment-level defects), and the gate only opens when all three come back clean. A workflow that can’t be made clean gets reported back to the owner with exactly what’s blocking it — it never gets waved through.

Pipeline overview

```
Intake → Preflight → Audit → Fix ⟲ (until clean) → Independent
verify → Package → Release record
```

Run every stage per workflow. When releasing several workflows, run each through the full pipeline independently — one workflow’s pass never covers another.

Stage 1 — Intake

Identify the workflow’s complete file set (SKILL.md or workflow definition, `references/`, `scripts/`, `assets/`). Confirm with the owner: product name, intended price tier (free products on Gumroad are limited to 250 MB; paid files can be far larger), and license terms. Snapshot the files before touching anything so every change in the fix loop is diffable.

Stage 2 — Preflight (deterministic checks)

Run the bundled checker:

```
python3 scripts/preflight.py <path-to-workflow-folder>
```

It machine-verifies what a human reviewer skims past: frontmatter parses and has name + description, every file the instructions reference actually exists in the bundle, no personal residue (author usernames, emails, `/Users/<name>` paths), no TODO/FIXME/placeholder markers, no broken local links, and the bundle zips cleanly. Exit code 0 = pass; anything else prints the exact failures. **A preflight failure is always a Critical finding.** Fix and re-run until it exits 0 — this is the “no errors” floor, checked by code rather than by judgment.

Stage 3 — Intensive checklist + rubric audit

Work through `references/release-checklist.md` item by item — roughly 90 binary properties across metadata, structure, instructions, inputs, reliability, outputs, bundled code, security, portability, and packaging. Every item is pass, fail, or N/A-with-reason; there is no “mostly”. Items tagged [auto] are already covered by preflight; the rest you verify by reading and by executing the workflow.

In parallel, score the workflow against `references/rubric.md` (8 weighted dimensions, evidence required for every deduction) — the checklist finds the defects, the rubric sizes them. If the companion `workflow-quality-audit` skill is installed, use its persona walkthroughs and report template too. Output: the completed checklist plus a findings list with severities and concrete fixes.

Stage 4 — Fix loop

Apply the fixes, most severe first. After each round, re-run **both** preflight and the audit — fixes routinely introduce new defects, and a stale pass is worthless.

Exit condition: preflight exits 0, **every** checklist item passes (or is N/A with a stated reason), **and** the audit reports **zero Critical and zero Major findings**. Minors are recorded in the release record but don't block release.

If three rounds don't converge, stop and escalate to the owner with the remaining findings — a workflow that resists three fix rounds usually has a design problem the owner needs to decide on, and grinding further wastes effort on the wrong layer.

Stage 5 — Independent verification

The fixer must not grade their own homework. Have a fresh agent (a subagent with no memory of the fix loop) re-run preflight and the rubric audit from scratch. The gate opens only if this independent pass also finds zero Critical/Major findings. If it finds something the fix loop missed, return to Stage 4 — and note what the miss was, since it usually reveals a blind spot worth adding to the rubric.

Stage 6 — Package for Gumroad

Read `references/gumroad-packaging.md` for current specs and build the bundle:

1. **Product zip** — the workflow folder, packaged (a `.skill` file for Cowork/Claude skills, plus a plain `.zip` fallback since buyers may not know what a `.skill` file is).
2. **Buyer README, English + Spanish** — installation, first run, requirements, support. Template in `references/listing-template.md`. Write both for someone who has never used a Claude skill. The Spanish version (`01-README.es.md`) must mirror the English section-for-section; keep commands, code, and file names untranslated (checklist section L has the full localization rules).
3. **LICENSE.txt** — the terms the owner chose at intake.
4. **Listing copy** — title, subtitle, description, bullets, FAQ. Template in `references/listing-template.md`.
5. **Cover-image spec** — 1280×720 PNG/JPEG (16:9), key content centered so Gumroad's square thumbnail crop doesn't cut it off. Generate the image if asked, or hand the spec to the owner's designer.

Stage 7 — Release record

Write `release-record.md` into the bundle's parent folder: workflow name and version, date, final rubric scores, preflight output, independent-verify confirmation, SHA-256 of the product zip, and any accepted Minor findings. This is the proof of QC — it's what distinguishes "we checked" from "we think we checked", and it's what you diff against when the next version ships.

Deliverables checklist

Every completed run hands the owner exactly five things:

- ☐ `<name>.skill` + `<name>.zip` (the product)
- ☐ `01-README.md` + `01-README.es.md` for buyers (inside the zip)
- ☐ `LICENSE.txt` (inside the zip)
- ☐ `listing-copy.md` (+ `listing-copy.es.md` if selling to Spanish-speaking buyers)
- ☐ `release-record.md` (the QC proof, including the Section L localization tally)

If any box is unchecked, the run isn't done.

Intensive Release Checklist

Every property below is binary: pass or fail. A workflow is release-ready only when **every** item passes or is explicitly marked N/A with a reason. Work through it top to bottom during Stage 3–4; the independent verifier (Stage 5) re-checks every item marked pass.

Items tagged **[auto]** are verified by `scripts/preflight.py`; the rest are checked by reading and by executing the workflow.

A. Metadata & frontmatter

- [ ] A1 [auto] SKILL.md exists at the bundle root
- [ ] A2 [auto] Frontmatter parses (valid `---` block)
- [ ] A3 [auto] `name` present, non-empty, kebab-case, matches folder name
- [ ] A4 [auto] `description` present, 15–1500 characters
- [ ] A5 description states WHAT the workflow does in the first sentence
- [ ] A6 description states WHEN to use it (contexts, not just capability)
- [ ] A7 description includes the literal phrases users actually say
- [ ] A8 description disambiguates from adjacent workflows ("not for X — use Y")
- [ ] A9 all when-to-use information lives in the description, none only in the body
- [ ] A10 name is outcome-oriented enough for a buyer to guess what it does

B. Structure & file integrity

- [ ] B1 [auto] every file referenced in any .md exists in the bundle
- [ ] B2 [auto] no zero-byte files
- [ ] B3 [auto] no `.icloud` eviction stubs
- [ ] B4 [auto] the bundle zips and the zip verifies clean
- [ ] B5 SKILL.md body under ~500 lines; overflow pushed to `references/`
- [ ] B6 every reference file is pointed to from SKILL.md with when-to-read guidance
- [ ] B7 reference files >300 lines have a table of contents
- [ ] B8 no orphan files (present in bundle but never mentioned or used)
- [ ] B9 folder names follow the standard anatomy (`scripts/`, `references/`, `assets/`)
- [ ] B10 no build junk (.DS_Store, **pycache**, .git, editor swap files)

C. Instruction quality

- [ ] C1 instructions are imperative ("Run X"), not descriptive musing
- [ ] C2 steps are numbered wherever order matters
- [ ] C3 every term a novice wouldn't know is defined at first use
- [ ] C4 non-obvious instructions explain WHY (rationale travels better than rules)
- [ ] C5 no contradictions between any two sections
- [ ] C6 no instruction depends on context that exists only in the author's head
- [ ] C7 examples given for anything format-sensitive
- [ ] C8 a first-time reader could execute end-to-end without asking a question
- [ ] C9 one-time setup clearly separated from every-time usage
- [ ] C10 first-run expectations set where behavior differs from steady state

D. Inputs & prerequisites

- [ ] D1 every required input enumerated
- [ ] D2 for each input: where the user finds it if they don't know it
- [ ] D3 environment requirements declared (OS, tools, accounts, plan tiers)
- [ ] D4 required credentials listed, with the safe way to supply each
- [ ] D5 behavior defined when an input is missing or wrong format
- [ ] D6 "When NOT to use this" section present and accurate

- [ ] D7 no assumed folder locations without a discovery step

E. Reliability & error handling

- [ ] E1 known failure modes documented as symptom → cause → fix
- [ ] E2 destructive operations are opt-in flags, never defaults
- [ ] E3 running twice in a row is safe (idempotent or explicitly warned)
- [ ] E4 interruption mid-run leaves a recoverable state (or the risk is documented)
- [ ] E5 network/service dependency failures have guidance
- [ ] E6 an explicit verification step confirms success after the main action
- [ ] E7 the most likely user mistake is anticipated and handled
- [ ] E8 error messages a user will see are quoted so they can be recognized

F. Output definition

- [ ] F1 the deliverable is named: format, structure, location
- [ ] F2 templates provided for any structured output
- [ ] F3 at least one example of correct output included where format matters
- [ ] F4 the user is told upfront what they'll receive
- [ ] F5 success output vs. warning output distinguishable by the user

G. Bundled code (`scripts/`) — skip section if no scripts

- [ ] G1 every script runs on a clean machine (stdlib-only, or deps declared)
- [ ] G2 scripts actually execute without error on valid sample input
- [ ] G3 scripts exit non-zero on failure (no silent success)
- [ ] G4 usage/help text present (`--help` or module docstring)
- [ ] G5 no hardcoded paths, hosts, ports, or credentials
- [ ] G6 config uses an example file (`*.example.json`), never a filled-in real one
- [ ] G7 every script the docs mention is bundled; every bundled script is mentioned

H. Security, privacy & trust

- [ ] H1 [auto] no email addresses anywhere in the bundle
- [ ] H2 [auto] no `/Users/<name>` or other personal paths
- [ ] H3 [auto] no author name or blocklisted words (run preflight with `--blocklist`)
- [ ] H4 no API keys, tokens, passwords, or real hostnames — including in examples
- [ ] H5 secrets guidance uses env-var/keychain indirection, never paste-into-chat
- [ ] H6 files with credentials instructed to be permission-locked (chmod 600)
- [ ] H7 any data leaving the user's machine is explicitly called out
- [ ] H8 the workflow does nothing its description doesn't say (no surprises)
- [ ] H9 destructive/irreversible actions require explicit user confirmation

I. Portability & generality

- [ ] I1 zero author-specific context in shipped files (names, habits, anecdotes)
- [ ] I2 works for the full range of inputs the description promises, not just the demo case
- [ ] I3 examples are illustrative, not load-bearing
- [ ] I4 OS-specific commands flagged as such, with alternatives where reasonable
- [ ] I5 [auto] no TODO/FIXME/PLACEHOLDER/TBD markers
- [ ] I6 nothing references the author's other private files, sessions, or memory

J. Buyer packaging (Gumroad)

- [ ] J1 product zip contains: buyer README, .skill file, LICENSE, skill-source copy
- [ ] J2 buyer README covers install, first run, requirements, and support contact
- [ ] J3 README written for someone who has never installed a Claude skill
- [ ] J4 LICENSE.txt present and matches what the listing promises
- [ ] J5 files named to sort sensibly (01-, 02-, 03- prefixes)
- [ ] J6 total size under 250 MB if the product can be gotten for $0
- [ ] J7 cover image 1280×720, key content inside the center square, <1 MB
- [ ] J8 listing description includes explicit requirements (prevents refund reviews)
- [ ] J9 listing promises nothing the workflow doesn't do — audit claims against A5–A8
- [ ] J10 support channel and response expectation stated

L. Spanish documentation (localization)

Buyer-facing documents ship in English **and** Spanish. The workflow's internal instructions (SKILL.md body) stay English-only unless the owner decides otherwise — Claude reads those; buyers read these.

- [ ] L1 Spanish buyer README exists (`01-README.es.md`) alongside the English one
- [ ] L2 section-for-section parity: every heading, step, and warning in the English README appears in the Spanish one — no silently dropped content
- [ ] L3 translation is natural Spanish, not word-by-word machine output (read it aloud: would a native speaker write this?)
- [ ] L4 commands, code, file names, and UI button labels are NOT translated — only the prose around them (translated commands break copy-paste)
- [ ] L5 UI labels the buyer must click are given as shown in the product's actual interface, with the Spanish gloss in parentheses where helpful
- [ ] L6 troubleshooting section translated, including the quoted error messages kept verbatim in their original language so buyers can match them
- [ ] L7 Spanish listing copy prepared (`listing-copy.es.md`) if the owner sells to Spanish-speaking buyers on the same listing or a second one
- [ ] L8 support section states which languages support is answered in — never imply Spanish support exists if it doesn't
- [ ] L9 numbers, dates, and currency formatted for the locale (1.234,56 € style where appropriate) in prose — but never inside code or config
- [ ] L10 parity re-verified after every fix-loop round that touched the English docs: an English edit without the matching Spanish edit is a **Major** finding
- [ ] L11 [auto] preflight run with `--require-spanish` exits 0 (checks each buyer-facing doc has a non-empty, non-identical `.es.md` counterpart)

K. Final gate

- [ ] K1 preflight.py exits 0 on the final bundle
- [ ] K2 full rubric audit shows zero Critical and zero Major findings
- [ ] K3 independent verifier re-confirmed every section above from scratch
- [ ] K4 release-record.md written (scores, checksums, date, accepted Minors)
- [ ] K5 the exact uploaded zip is the exact audited zip (SHA-256 match)

Counting rule: the release record must state the tally, e.g. "95 checked / 3 N/A (no scripts) / 0 failing." Any failing item = the gate stays closed. There is no "mostly passing."

Section L may only be marked N/A as a whole, with the owner’s explicit, recorded decision to ship English-only — never silently skipped.

Workflow Quality Rubric

Score each dimension 1–5. Record evidence (a quote or a named gap) for every score below 5. Weighted total = Σ(score × weight) / Σ(weights).

Weights reflect what actually drives user ratings: a workflow that never triggers or fails silently gets a 1-star review no matter how elegant its prose is.

#	Dimension	Weight
1	Triggering	3
2	Clarity	2
3	Completeness	3
4	Reliability	3
5	Output definition	2
6	Efficiency	1
7	Safety & trust	2
8	Generality	2

---

1. Triggering (weight 3)

The description in the frontmatter is the *only* thing Claude sees before deciding to use the workflow. If it doesn't fire, nothing else matters.

Check: - Does the description state both *what it does* AND *when to use it*? - Does it list the concrete phrases/contexts users actually say ("push my site", "upload these files"), not just abstract capability ("file synchronization")? - Is it appropriately "pushy"? Claude under-triggers by default; descriptions should tell it to trigger even when the user doesn't name the workflow. - Are near-miss cases disambiguated (when a *similar* workflow should win instead)? - Is all "when to use" information in the description, not buried in the body?

5 = description covers what + when + user phrasings + disambiguation. **3** = what it does is clear but trigger contexts are thin or live in the body. **1** = capability-only description; Claude will rarely invoke it unprompted.

2. Clarity (weight 2)

Check: - Imperative instructions ("Run X, then check Y"), not passive description. - Every term a novice wouldn't know is explained at first use or linked to a reference. - Steps are ordered and numbered where order matters. - The *why* behind non-obvious instructions is explained (models and humans both follow instructions better when they understand the reason). - No contradictions between sections.

5 = a first-time reader can execute without a single follow-up question. **3** = executable but requires guessing at 1–2 points. **1** = ambiguous ordering, unexplained jargon, or contradictory guidance.

3. Completeness (weight 3)

Check: - Required inputs enumerated (what to ask the user for, and where the user finds each item if they don't know it). - Prerequisites and one-time setup separated from everyday usage. - Edge cases addressed: empty/huge/malformed input, missing files, first run vs. repeat run, offline dependencies. - "When NOT to use this" section present — misuse is a top source of bad ratings. - Every file, script, or asset the instructions mention is actually bundled or its location is precisely specified.

5 = inputs, setup, edges, and anti-use-cases all covered; no dangling references. **3** = happy path complete; edges thin. **1** = assumes context the user doesn't have; references missing files.

4. Reliability (weight 3)

Check: - Known failure modes listed with symptom → cause → fix (a troubleshooting table is the gold standard). - Destructive operations are opt-in, never default. - Idempotence: what happens if it's run twice? Interrupted midway? - External dependencies (network, credentials, services) have failure guidance. - Verification step: how does the user *confirm* it worked?

5 = troubleshooting present, non-destructive defaults, verify step defined. **3** = some failure guidance but silent-failure paths remain. **1** = no failure handling; errors will strand the user.

5. Output definition (weight 2)

Check: - The deliverable is pinned down: format, structure, location, naming. - Templates provided for structured outputs (report skeletons, config examples). - Examples of good output included where format matters. - The user is told what they'll receive before the work starts (expectation-setting is half of a good rating).

5 = exact output template/example present. **3** = output described in prose but format left to chance. **1** = output entirely unspecified; every run produces something different.

6. Efficiency (weight 1)

Check: - SKILL.md under ~500 lines; detail pushed to `references/` with clear pointers on when to read each file. - Repetitive/deterministic work captured in bundled `scripts/` rather than regenerated every run. - No dead weight: every section earns its place. - Large reference files (>300 lines) have a table of contents.

5 = lean body, progressive disclosure, scripts bundled where warranted. **3** = works but bloated or misses obvious script-bundling opportunities. **1** = monolithic wall of text; wasted tokens every invocation.

7. Safety & trust (weight 2)

Check: - Secrets never pasted into chat or stored in plaintext when an env-var or keychain pattern is available; file permissions locked down where relevant. - No surprising behavior: the workflow does what its description says, nothing more. - Data leaving the user's machine (uploads, API calls) is called out explicitly. - Destructive or irreversible actions require explicit confirmation.

5 = credential hygiene, explicit data flows, confirmed destructive actions. **3** = mostly safe but one loose practice (e.g., plaintext config suggested first). **1** = handles secrets carelessly or does undisclosed destructive/network actions.

8. Generality (weight 2)

A workflow sold or shared must work for *any* buyer, not just its author.

Check: - No hardcoded names, personal paths, or author-specific context in the instructions (personal notes like "Greg's folder is X" belong in the user's own memory/preferences, not in a distributed workflow). - Examples are illustrative, not load-bearing — the workflow generalizes past them. - Environment assumptions declared (OS, tools, account types) rather than silent. - Reasonable behavior across the *range* of inputs the description promises.

5 = fully portable; a stranger could run it today. **3** = generalizes with minor edits; a few author-specific residues. **1** = only works for the author's exact setup.

Severity definitions for findings

- **Critical** — will cause failed runs or non-triggering for a typical user (broken trigger, missing bundled file, silent destructive default, hardcoded personal context in a sold product).
- **Major** — user completes the task but with real friction or degraded output (missing troubleshooting for a common failure, undefined output format, unexplained prerequisite).
- **Minor** — polish (tone, ordering, redundancy, formatting).

One Critical finding caps the overall verdict at “Fix majors” regardless of score.

Gumroad Packaging Specs

Verified against Gumroad's help center, July 2026. If a listing behaves unexpectedly, re-check gumroad.com/help — specs do change.

Files

- Free ($0 / $0+) products: **250 MB** total file limit. Paid products allow much larger files (multi-GB); Claude workflow bundles are tiny, so this only matters if you bundle video walkthroughs.
- Upload the product as a **zip**. Include both the `.skill` file and an unzipped copy of the folder — buyers unfamiliar with `.skill` files can still inspect the contents.
- Gumroad shows buyers a "Download all" experience; name files so they sort sensibly: `01-README.md`, `02-<name>.skill`, `03-LICENSE.txt`.

Cover image

- **1280 × 720 px** (16:9), PNG or JPEG, ideally under 1 MB.
- Gumroad crops a **centered square** for thumbnails — keep the product name and key art inside the middle square or it's cut off in browse views.
- Profile banner (if updating the storefront too): 1600 × 400 px.

What's inside the product zip

```
<product-name>/
├── 01-README.md          ← buyer-facing: install, first run,
requirements, support
├── 01-README.es.md       ← the same, in Spanish (checklist section
L governs quality)
├── 02-<name>.skill       ← the installable skill package
├── 03-LICENSE.txt        ← usage terms
└── skill-source/         ← unzipped copy of the skill folder
(transparency sells)
```

Verify the localization mechanically before shipping:

```
python3 scripts/preflight.py <bundle-folder> --require-spanish
```

Listing fields to prepare (paste-ready in listing-copy.md)

- **Name** — outcome-first, not mechanism-first ("Deploy your site to Hostinger in one command", not "FTPS sync tool").
- **Description** — problem → what it does → what's included → requirements → refund/support policy. Buyers of workflow products are burned often; explicit requirements ("needs Claude with skills support; macOS/Linux") prevent the refunds and 1-star reviews that vague listings earn.
- **Price** — remember the 250 MB free-tier limit if using $0+ pricing.
- **FAQ / support** — where buyers reach the owner; response expectation.

Templates: Buyer README and Gumroad Listing Copy

Buyer README template (ships inside the zip as 01-README.md)

```
# [Product name]

Thanks for your purchase! This is a Claude skill — a packaged workflow that
teaches Claude to [one-line outcome].

## What you need

- [Claude plan/app requirement, e.g. "Claude desktop app or Claude Code with skills support"]
- [OS requirements, if any]
- [Accounts/credentials the workflow uses, if any]

## Install (2 minutes)

1. Unzip this download.
2. [Exact install steps for the buyer's platform — click Save skill / copy the
   folder to the skills directory / etc.]
3. Confirm it's installed: ask Claude "[a phrase that should trigger it]".

## First run

[Walk the buyer through their first successful use, including the one-time
setup if any. End with what success looks like.]

## Troubleshooting

[Top 3 issues and fixes. Link to the workflow's own troubleshooting section
for the rest.]

## Support

Questions or problems: [contact channel]. I typically reply within [expectation].
```

Gumroad listing copy template (listing-copy.md, paste into Gumroad)

```
# Title
[Outcome the buyer gets, ≤60 chars — "…in one command", "…without the errors"]

# Subtitle / one-liner
[Who it's for + the pain it removes]

# Description

**The problem:** [2-3 sentences the target buyer instantly recognizes]

**What this does:** [3-4 sentences, concrete. Mechanism only after outcome.]

**What's included:**
- [The skill itself]
- [Buyer README with install + first-run guide]
- [Bundled scripts/templates, if any]
- [License terms summary]

**Requirements:** [Every requirement, explicitly. This line prevents refunds
and 1-star reviews — do not soften it.]

**FAQ**
- *Does this work with [common adjacent thing]?* — [honest answer]
- *What if it doesn't work for me?* — [refund/support policy]
```

```
# Tags
[3-6 marketplace search terms buyers would type]
```

Spanish versions (01-README.es.md, listing-copy.es.md)

Use the same templates above, translated — not adapted loosely. Rules that matter (full list in `references/release-checklist.md`, section L):

- Mirror the English structure heading-for-heading; a buyer comparing the two should find every warning and step in both.
- Never translate commands, code, file names, or things the buyer must type or click — only the prose around them. A translated command breaks copy-paste.
- Keep quoted error messages verbatim in their original language (that's what the buyer's screen will show), with a Spanish explanation after.
- State plainly which languages support responds in.
- Write natural Spanish. If a sentence reads like a word-substituted English sentence, rewrite it.

Writing rules for both

- Claims must match the audit: anything promised here must map to a passing checklist item. The listing is part of the QC surface — an oversold listing turns a passing product into a 1-star review.
- Write requirements as exclusions too ("not for X") — the wrong buyer refunding is worse than no sale.
- Keep the buyer README self-sufficient: assume the buyer never contacts support and never reads the listing again after purchase.

Audit Report Template

Use this exact structure for every per-workflow report. One report per workflow, one file per report, named `audit-<workflow-name>.md`.

Quality Audit: [workflow name]

Audited: [date] · **Version audited:** [file hash, date, or version if known] **Weighted score:** X.X / 5 · **Verdict:** Ship / Fix minors / Fix majors / Rework

Scorecard

Dimension	Weight	Score	One-line reason
Triggering	3		
Clarity	2		
Completeness	3		
Reliability	3		
Output definition	2		
Efficiency	1		
Safety & trust	2		
Generality	2		
Weighted total		**X.X**	

What’s working

[2–4 bullets. Real strengths, not padding — the owner needs to know what NOT to touch during the rework.]

Findings

[Ordered by severity, then by dimension. 5–12 findings is the healthy range.]

F1 · [Critical/Major/Minor] · [Dimension] — [one-line title]

Evidence: [quote the offending line, or name the specific gap] **Why it hurts ratings:** [the user-visible consequence, one sentence] **Fix:** [concrete action — replacement text, section to add, file to bundle. Written so the owner can apply it in under an hour without follow-up questions.]

F2 · …

Quick wins

[The 2–3 fixes with the best ratings-impact-per-minute, pulled from the findings above. This is what the owner should do today.]

Re-audit trigger

[When to re-run this audit: after applying Critical/Major fixes, before the next release, or on a cadence if the workflow changes often.]

Portfolio summary template (multi-workflow audits only)

Workflow Portfolio: Quality Summary

Audited: [date] · **Workflows:** N

Workflow	Score	Verdict	Top fix

Systemic patterns

[Issues appearing in 2+ workflows — these are process problems. Fix the process (e.g., an authoring checklist) once, not each workflow separately.]

Suggested order of work

[Which workflows to fix first, weighing severity against how much each one is used or how visibly it’s rated.]