Copy Audit

Read every string a repo ships, judge it against the content standards, and rewrite only the words — never the code around them.

Produces	Per-unit verdicts (keep, rewrite, flag, delete) with an exact rewrite for each, plus a cross-cutting holistic report — nothing written to disk until you approve.
Use it when	A branch or a marketing site has drifted, and the copy, docs, and microcopy need a plain-language, inclusive, and voice pass. Or you want a comment-slop and test-name sweep of the code itself.
Never	Change code, structure, meaning, product names, numbers, or URLs. Only the human-readable text span ever moves, and nothing writes to disk before you say go.

What it is

Copy drifts. A landing page written in one voice grows a second one; a button says “Click here to get started” where “Get started” would do; a doc picks up “simply” and “just” and “sanity check” over a long branch; an error message explains the system instead of the reader’s next move. No single reviewer keeps up, because the strings are scattered across markdown, templates, JSON, YAML, and a dozen source languages, and each one reads fine in isolation. The drift is only visible when someone reads all of it against a standard, in one pass.

That is the work: find every string a repo ships to a human, and judge each one in the context of the file it lives in. Then decide — keep it, rewrite it, flag it for a human call, or delete it. The output is a set of verdicts with a concrete rewrite for each, ordered so the reader sees what changed and why. It is a content review, not a linter run: the judgment is about whether the words serve the reader, not whether they pass a regex.

The hard boundary is that you rewrite words, never code. A copy edit that reflows a JSON structure, drops a quote, or shifts a line number is not a copy edit — it is a bug you introduced while tidying prose. So every change is a surgical splice of the human-readable span alone, and you prove afterwards that not one line of code moved. And because a wrong copy edit is visible harm shipped to users, nothing is written to disk until a person has read the proposed rewrites and approved them.

What you produce, concretely: for each string, a verdict — `keep`, `rewrite`, `flag`, or `delete` — a category and severity, a short note, and for a rewrite the exact replacement text. Plus a holistic report for the problems that only appear across strings: terminology drift, a voice that wanders, the same sentence three times on one page.

How you’d do it by hand

The tooling wraps two deterministic steps — pull the strings out, splice the approved ones back — around the one step that needs judgment: reading each string against the standard. Strip the scripts away and the craft is the same. Here it is end to end.

Pull out the copy, and only the copy

Walk the tree (or a diff range) and, for each file, separate the words a human reads from the code around them. This is the step that quietly decides everything downstream, because a string you misclassify as copy becomes a rewrite that breaks a build, and a string you miss never gets reviewed.

The rule that keeps it honest: a raw string is copy only when it reads like human language. It has a letter, and either whitespace, sentence punctuation, or a copy-carrier key (`title`, `label`, `description`, `placeholder`, `alt`, `message`, and their kin). URLs, file paths, identifiers, class lists, `CONSTANT_CASE`, hex colours, numbers, and code-shaped strings are dropped. When you are unsure, skip it. A conservative miss costs you one unreviewed string; an aggressive false positive costs you a corrupted file.

Different file types carry copy in different places, and you read each on its own terms:

- **Markdown** — headings, prose paragraphs, list items, blockquotes, image alt text, and the copy values in the frontmatter.
- **Templates** (HTML, Astro, Svelte, Vue, JSX, TSX) — visible text nodes and the copy attributes (`alt`, `title`, `placeholder`, `aria-label`), with `<script>`, `<style>`, comments, and the component fence masked out so you scan structure without ever capturing code.
- **Data** (JSON, YAML, TOML) — string values under copy-carrier keys, never the keys themselves and never config values that only look like words.
- **Source** (JS/TS and 40 languages) — string literals judged by context; a bare label like `"Save"` counts when it is the first argument to a UI marker (`Text(`, `Button(`, `NSLocalizedString(`, `t(`), and template literals with interpolation are left alone because they are unsafe to rewrite.

Record, for each unit, the exact character offsets of the editable payload and a hash of the file it came from. Those two facts are what make the write-back safe later.

Judge each string against the standard

This is the part no tool does for you. Read every unit in the context of its whole file — a heading reads differently above its section, a button label differently next to its siblings — and judge it against four pillars.

1. **Plain language and readability.** Short sentences (flag anything past 30 words), one idea per sentence, everyday words over jargon, active voice, present tense, second person for instructions. Front-load the point. Cut filler: `in order to` → `to`, `utilize` → `use`, `at this point in time` → `now`. Prefer positive phrasing over double negatives.
2. **Inclusive and bias-free language.** No gendered defaults ("guys" for a group, "manpower", "chairman"). No ableist idioms — "sanity check" → "quick check", "dummy" → "placeholder". Retire exclusionary tech pairs — master/slave → primary/replica, blacklist/whitelist → blocklist/allowlist. Watch "simply", "just", "obviously", "easy" — they presume the reader's effort. When a call needs house-style or community context you do not have, flag it rather than guess.
3. **UX microcopy.** A control names the action it performs — "Get started", not "Click here". An error says what happened and what to do next, in the reader's terms, never a raw code. Labels are consistent across the surface. Sentence case unless the house style says otherwise.
4. **Voice and grammar.** One consistent voice across the corpus. Correct grammar, spelling, and punctuation. Parallel structure in lists. No trailing throat-clearing.

For each unit you land on one of four verdicts. `keep` — it already serves the reader. `rewrite` — you can state the exact better words, and the change is safe and mechanical. `flag` — there is a real problem but the fix needs a human: a brand term, a person's self-description, a claim you can't verify. `delete` — the string adds nothing (and never on a test name, where deleting the first argument breaks the runner).

The discipline that separates a review people trust from one they learn to ignore: preserve meaning, product names, numbers, URLs, and the reader's language, always. A rewrite that is cleaner but says something different is a defect, not an improvement.

Read the whole corpus once, for what per-string review can't see

Some problems are invisible one string at a time. The word "workflows" in three places and "skills" in four, for the same concept. The same value proposition stated on the hero, the features section, and the footer in three different voices. A page that front-loads on one route and buries the point on the next. Read a whole page or route in reading order and rank what you find — terminology drift, repetition, inconsistent voice — as advisory findings. These don't get auto-applied; they are the map for the human editor.

Write back the approved rewrites, and only the text

Once a human has read the verdicts and approved, splice the rewrites in — and here every guard earns its place.

- **Refuse a file that moved.** Before touching a file, re-hash it. If the hash differs from what you recorded at extract time, someone else changed the file underneath you: stop, defer that file to the next sweep, and never force the write.
- **Confirm the span.** Check that the bytes at the recorded offsets still equal the string you reviewed. A second belt against drift.
- **Splice the payload, escaped for its context.** Replace only the text span, re-escaped for where it lives — a JSON string re-escaped as JSON, an HTML text node HTML-escaped, a YAML scalar quoted for its style. The surrounding quotes, tags, `#` and `-` markers, and JSON commas are byte-identical before and after. Apply bottom-up by offset so earlier edits never invalidate later ones.
- **`keep` and `flag` never write.** A flag is surfaced in the report for a human; it is not a silent edit.

Prove you only touched words

The last step is the one that lets anyone trust the rest. For every file you changed, every hunk of the real diff must fall inside a span you recorded as copy. If a single changed line sits outside those spans, you moved something you shouldn't have — stop and treat it as a fault, not a warning. Run the repo's own formatter or linter afterwards and report its result. A copy audit that can't prove it changed only copy is just an edit you're asking people to take on faith.

Why it's built this way

Every rule here exists because a specific failure mode is common and expensive.

Extract conservatively. The whole audit rests on the line between copy and code. Lean toward copy and you eventually rewrite an identifier and break a build — the one outcome that would make a team turn the whole thing off. Lean toward code and you miss a string, which costs one unreviewed unit and nothing else. The asymmetry is the reason the classifier skips when unsure, and the reason the write-back re-checks the hash and the span even though extraction already looked.

Judge in context, not in a list. A string ripped out of its file loses the information you need to judge it — who the reader is, what the button sits next to, whether the heading earns its weight. Reviewing each unit with its full file is slower than a flat wordlist and it is the only way the verdicts are any good.

Rewrite words, never structure. The reason to splice a character span instead of regenerating a file is that regeneration cannot promise it changed only the copy. A copy tool that occasionally rewrites code is worse than no tool. The SHA guard, the span check, and the verify pass are three independent proofs of the same one invariant, because that invariant is the entire trust model.

Nothing writes without approval. Copy is subjective and high-stakes: a rewrite ships to every reader, and "better" is a judgment a person has to own. Auto-applying even confident rewrites would trade that ownership for speed, so the flow always stops for review, and `flag` exists precisely for the calls a machine should not make alone.

Meaning is the invariant, not word count. The point is not shorter text; it is text that serves the reader without changing what it says. Product names, numbers, URLs, and the reader's own language are load-bearing, and a rewrite that quietly alters any of them has failed no matter how clean it reads.

The through-line: a copy audit is trustworthy exactly to the degree that it changes only what it claims to change, and improves the reader's experience without altering the meaning. Everything above is in service of that.

At a glance

Pre-audit checklist:

- Separate copy from code before judging anything; when unsure whether a string is copy, skip it.
- Read every unit in the context of its whole file, not as a flat list.

- Judge against four pillars: plain language, inclusive language, UX microcopy, voice and grammar.
- One verdict per unit: keep, rewrite, flag, delete. Flag the calls that need a human; never delete a test name.
- Preserve meaning, product names, numbers, URLs, and the reader's language — always.
- Read the whole corpus once for cross-cutting drift: terminology, repetition, wandering voice.
- Get the verdicts approved before a single byte is written.
- Splice only the payload span, escaped for its context; leave quotes, tags, and markers byte-identical.
- Refuse any file whose hash moved since extract; defer it, never force it.
- Prove it: every changed hunk must fall inside a recorded copy span, or it isn't a copy edit.