A Voice-Driven Copyedit Pipeline for Quarto Blogs and Research Papers

Ronald G. Thomas

2026-09-01

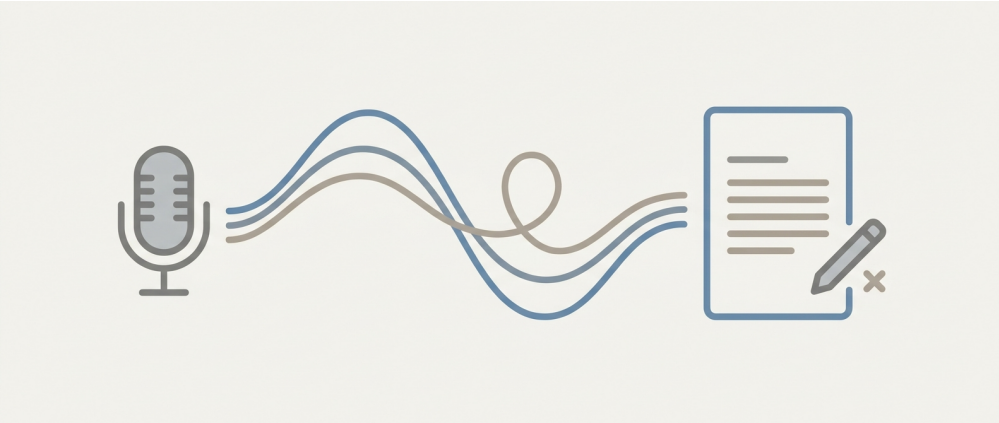


Figure 1: Speaking edits into a microphone instead of printing, annotating, and retyping them

Introduction

I didn't think dictating edits to 57 blog posts and 90 research-paper documents would end up teaching me more about failure modes than about speech recognition. I went in assuming the hard part would be Whisper accuracy. It wasn't. The hard part was everything *around* the transcription: knowing when a recording silently fell short, knowing when an anchor was located correctly but quoted incompletely, knowing that `index.qmd` is secretly a symlink in every single post I own, and knowing that "quote… end quote" doesn't always come back as the words "quote… end quote."

The problem I was solving is mundane and exhausting: render to PDF, print, annotate margins, open the source, find the passage, retype the edit. Repeated across a corpus large enough that no single sitting holds it in view. I replaced that loop with: read the rendered post, dictate edits aloud, machine transcription, a structured edit file, automated application, hunk-by-hunk diff review. The author retains full authorship throughout. No language model composes, rewrites, or suggests prose at any point. The machine performs three mechanical jobs: speech-to-text, resolving a spoken anchor to an exact location in the source, and applying a specified change. Every word that ends up in a post or paper is the author's.

This post documents the design, the real installed scripts, and a worked example run end to end for real, including three bugs found only by actually running it, and a live recording that failed three times before it succeeded once.

Motivations

- A print-annotate-retype loop that scales linearly with corpus size, and a corpus (57 blog posts, 90 candidate research-paper documents) large enough that the loop had become the bottleneck, not the reading or the deciding.
- A conviction that the mechanical part of copyediting (carrying a decision made on paper back into a source file without introducing errors) offers no editorial value and should be removed entirely rather than merely made faster.
- Skepticism, later vindicated, that "just let an LLM propose the edits" was the wrong design: authorship should stay with the author, and the machine's job should be mechanical, not editorial.
- Curiosity about where a voice pipeline like this actually breaks in practice, rather than in theory, which turned out to be a more productive question than "does Whisper work."

Objectives

By the end of this post, I will have:

1. Specified a repeatable process for reviewing and editing prose across three trees: a Quarto blog (`ryyblog`) and two research-compendium trees (`res`, `alz`), where dictation grammar, checkpoints, and failure controls apply identically regardless of which tree a document lives in.
2. Built and documented five real, installed shell/Python scripts (`voice-review-record`, `voice-review-transcribe`, `voice-review-tag`, `voice-review-dictate`, plus corpus-status and discovery helpers) that implement the pipeline, not just describe it.
3. Run the pipeline end to end for real against one real paragraph, with a genuine single-take recording, capturing every transcription error and every bug the run surfaced.
4. Catalogued the failure modes this pipeline actually hit (plausible transcription errors, silent anchor misapplication, symlinked target files, resolution-step anchor truncation) and the controls that catch each one.

What This Is Not

Worth stating plainly, since "voice pipeline" invites the wrong assumption:

- Not an AI copyediting tool. Nothing proposes changes to your writing.
- Not a replacement for reading. You still read every post or paper.
- Not fully automated. Two human checkpoints are mandatory, discussed below.

Repository Architecture

Understanding the layout is a prerequisite, because it determines where commands run and where review artifacts live.

```
~/prj/ryyblog/                      ← parent site repo
├── _quarto.yml
├── index.qmd                       ← site landing page
├── .gitignore                      ← contains `posts/*/`
└── posts/
    ├── _metadata.yml               ← tracked by parent repo
    ├── pp-body-mass-prediction/    ← independent git repo
```

```
│   ├── .git/
│   ├── index.qmd              ← the file under review
│   ├── index.pdf              ← pre-rendered, useful for reading
│   ├── analysis/  data/  tests/
│   ├── DESCRIPTION  NAMESPACE  Dockerfile  Makefile
│   └── renv.lock
└── ... 56 more
```

Three facts follow from this and shape everything downstream:

1. **Each post is its own git repository**, not a submodule. The parent `.gitignore` contains `posts/*/`, so the parent repo cannot see inside them. All version-control operations for prose happen *per post*.
2. **Posts are research compendia**, not simple markdown files. Each carries `renv.lock`, a `Dockerfile`, `DESCRIPTION`/`NAMESPACE`, tests, and figures. `index.qmd` is one file among many, and its code chunks execute against a pinned environment.
3. **56 of 57 have GitHub remotes** under `rgt47/`. This is the basis for the working-clone setup below.

Vocabulary priming (more on this shortly) does not live inside `ryyblog` at all: it lives at `~/prj/vocab`, a small shared repository used by ryyblog and by the `res`/`alz` research-compendium trees, since a specialty topic (AWS, clinical statistics, Shiny) is not particular to any one tree.

`index.qmd` is a symlink in every post, not a plain file. Verified across the full corpus:

```
find ~/prj/ryyblog/posts -maxdepth 2 -name "index.qmd" -type l | wc -l   # 57
find ~/prj/ryyblog/posts -maxdepth 2 -name "index.qmd" | wc -l           # 57
```

All 57 posts symlink `index.qmd` at the post root to `analysis/report/index.qmd`, the same layout the research-compendium trees use, discussed later. This is corpus-wide, not an occasional wrinkle. **Only reading follows the symlink transparently**: `cat index.qmd` or opening it in an editor shows the real content, because that resolution happens at the OS level. Editing, diffing, and staging do **not**: verified directly, `git diff --word-diff=color index.qmd` after a real edit produced no output at all, silently showing no change, because git tracks the symlink entry itself (unchanged) separately from the target file's blob (changed). The real work (diff, stage, `git add`) must target `analysis/report/index.qmd` explicitly. Diffing or staging the symlink path is not merely inconvenient, it is **silently wrong**: it reports a clean diff on a post that was, in fact, just edited.

Post categories

The slug prefixes group the corpus by type:

Prefix	Domain
`cln-`	Clinical trials infrastructure
`pp-`	Palmer Penguins teaching series
`pub-`	Publishing and Quarto
`rp-`	Research practice and review methodology

Prefix	Domain
`rl-`	R language topics
`sec-`	Security and secrets management
`sh-`	Shell
`shy-`	Shiny
`wf-`	Workflow and tooling
`zc-`	zzcollab / compendium tooling

Review by prefix group, not alphabetically. Posts within a group share terminology and voice, and cross-post inconsistencies (a term defined in one post and used undefined in another, drift in how a concept is named) are only visible when the posts are read consecutively. Alphabetical order interleaves the groups and hides exactly the problems a full-corpus review is for.

Working Clones

Editing happens in dedicated clones outside the parent site tree, not in `~/prj/ryyblog/posts/*` directly. `quarto render` writes to `_freeze/`, and an automated edit-and-render cycle running across 57 repositories multiplies the consequences of doing that in the tree also treated as the canonical copy.

Check for unpushed work before cloning from GitHub:

```
cd ~/prj/ryyblog
for d in posts/*/; do
  b=$(git -C "$d" rev-parse --abbrev-ref HEAD 2>/dev/null)
  ahead=$(git -C "$d" rev-list --count "origin/$b..$b" 2>/dev/null || echo "?")
  dirty=$(git -C "$d" status --porcelain | wc -l | tr -d ' ')
  [[ "$ahead" != "0" || "$dirty" != "0" ]] && \
    printf '%-45s ahead:%s dirty:%s\n' "$(basename $d)" "$ahead" "$dirty"
done
```

Create the clones:

```
mkdir -p ~/prj/review && cd ~/prj/review
for d in ~/prj/ryyblog/posts/*/; do
  url=$(git -C "$d" remote get-url origin 2>/dev/null) || continue
  git clone "$url"
done
```

All editing happens in `~/prj/review/`. When a post is finished, push, then `git -C ~/prj/ryyblog/posts/<slug> pull` to bring the parent tree's copy forward, so it sees one settled update per post rather than every intermediate state.

Remote protocol note. Remotes are a mix of SSH and HTTPS with no evident pattern. An unattended loop that pushes 57 repositories will prompt for credentials on each HTTPS remote unless a credential helper is configured. Either confirm the helper works, or normalize to SSH.

Pipeline Overview

```
 [1] READ            index.pdf, rendered, on screen, tablet, or paper
       |
 [2] DICTATE         structured spoken edits → .wav
       |
 [3] TRANSCRIBE      whisper-cli + vocabulary prompt → .notes.txt
       |
 [4] ◆ CHECKPOINT    human reads transcript; fixes misheard terms
       |
 [5] RESOLVE         anchors → exact source strings → edits.md
       |
 [6] ◆ CHECKPOINT    human verifies anchors; confirms uniqueness
       |
 [7] APPLY           edits.md → real source file
       |
 [8] VERIFY          word-diff, chunk integrity, quarto render
       |
 [9] STAGE           git add -p, per hunk
       |
[10] COMMIT & PUSH
```

The two ◆ checkpoints are not optional; their justification is in the Failure Modes section below. Stages 1–3 can be batched across many posts before any of stage 5 begins. Reading and applying demand different modes of attention, and interleaving them is a substantial part of what makes the old process tiring.

One-Time Setup

Software

```
brew install whisper-cpp ffmpeg
ls $(brew --prefix)/bin | grep -i whisper
```

The Homebrew formula installs a family of binaries. The one required here is **`whisper-cli`**. Others present, for reference: `whisper-server` (keeps the model resident in memory), `whisper-vad-speech-segments` (voice-activity detection, useful if silence-driven hallucination appears), `whisper-stream` (live transcription, less accurate, not recommended here), `whisper-quantize`, `whisper-bench`.

Model

Homebrew does not ship model files.

```
mkdir -p ~/.cache/whisper
curl -L -o ~/.cache/whisper/ggml-large-v3.bin \
  https://huggingface.co/ggerganov/whisper.cpp/resolve/main/ggml-large-v3.bin
ls -lh ~/.cache/whisper/ggml-large-v3.bin      # expect ~2.9G
```

`large-v3` is the default, on verified evidence, not `large-v3-turbo`. Both models were run head-to-head on the same 20-second clinical-terminology calibration clip, with the same core+clinical prompt: `turbo` failed to recover "estimand," "multiple imputation," or "MNAR" across three separate rounds of prompt tuning; each fix for one term either did nothing for the others or introduced a new regression (MNAR flipping between "MMNAR" and "mRNAR" depending on prompt wording). `large-v3`, run on the identical clip with the identical prompt, recovered every one of those terms correctly on the first attempt, at the cost of two minor cosmetic issues and roughly double the processing time. Domain terminology density in this corpus is high enough that the tradeoff favors accuracy. Keep `large-v3-turbo` available for speed-sensitive batches where a specialty's vocabulary has already proven reliable:

```
curl -L -o ~/.cache/whisper/ggml-large-v3-turbo.bin \
  https://huggingface.co/ggerganov/whisper.cpp/resolve/main/ggml-large-v3-turbo.bin
```

Verified performance on this machine (Apple M1): `large-v3-turbo` processed 13.4 seconds of audio in 4.53 s wall clock (1.72 s one-time model load); `large-v3` processed a 20-second clip in roughly 6–7 s per pass, including load. Metal acceleration engages automatically; no `GGML_METAL_PATH_RESOURCES` environment variable is needed at whisper-cpp 1.9.2. Model footprint: 1623.92 MB (`turbo`), ~2.9 GB (`large-v3`).

Vocabulary prompt

This is the single highest-leverage configuration item in the pipeline.

The problem, demonstrated. A test recording ending in domain terminology transcribed as:

> This is a test. One, two, three. Test, test, test. **Reuben, bitmap.**

Generic English was flawless. The domain terms were replaced with confident, fluent, entirely wrong alternatives. "Reuben" for "Rubin" is the failure mode to internalize: **terminology errors do not arrive garbled, they arrive plausible.** A reader skimming the transcript sees a name, not an error.

Size ceiling. Whisper's `--prompt` flag biases decoding toward supplied vocabulary, but the prompt is not unbounded. Whisper's decoder has a fixed text context of 448 tokens across all model sizes, and `whisper.cpp` reserves half of that for the prompt, roughly 220 tokens. Anything beyond that is silently truncated with no warning, and truncation drops from the **front** of the string, keeping the tail. A single file cumulative across every post, manuscript, and repo this pipeline touches would exceed this almost immediately.

Structure: a shared core plus topic-tagged specialty files, dictated by hand. This is not one blog with a fixed prefix taxonomy; it is `ryyblog` plus two research-compendium trees, each with its own naming conventions or none at all. A specialty is a **topic**, not a directory convention: AWS terminology learned from a ryyblog post about EC2 hosting is exactly as useful transcribing a research manuscript that happens to mention AWS. That argues for one vocabulary store shared across every tree, kept outside any of them:

```
mkdir -p ~/prj/vocab && cd ~/prj/vocab && git init

cat > ~/prj/vocab/core.txt <<'EOF'
Quarto, renv, qmd, end quote
EOF

cat > ~/prj/vocab/clinical.txt <<'EOF'
Rubin's rules, estimand, estimands, MMRM, multiple imputation, MNAR,
missing at random, missing not at random, ADAS-Cog, tipping point analysis,
biostatistics
EOF

git add core.txt clinical.txt
git commit -m "Initialize shared cross-tree vocabulary store"
```

No automatic derivation. State the specialty by hand, every time. There is no prefix or filename to mechanically derive a specialty from in the research-compendium trees, and forcing one back in for the blog alone would mean maintaining two different mechanisms for the same decision. So every transcription call, regardless of tree, states core plus whichever specialty file(s) apply, decided by the person doing the reviewing. A document can draw on more than one specialty, and nothing prevents the same specialty file from serving every tree alike.

Three points on content:

- Include **`end quote`** in `core.txt`. It is load-bearing as a structural delimiter and must transcribe consistently rather than as "and quote," so it belongs in every prompt regardless of topic.
- Treat each specialty file as **cumulative within its own topic**. A term mangled while reviewing a clinical document is appended to `vocab/clinical.txt`, not to `core.txt`.
- Within a file, put newly failing or currently unreliable terms **last**, since the front is what gets cut first if a specialty file grows past budget.

Calibration run

Before recording anything real, establish the baseline error signature on your own voice:

```
voice-review-record cal --seconds 20
```

`voice-review-record` shows a live countdown and automatically checks the resulting file's actual duration against the requested one: `-t 20` (or any `-t N`) is a request, not a guarantee. Observed directly: a `-t 45` recording once produced a 39.85-second file, several seconds short, with no error from `ffmpeg` itself; the wrapper surfaces that as a warning immediately.

Speak your fifteen hardest terms plus one "end quote," drawn from one specialty's vocabulary. Transcribe twice (without and with the prompt) and compare, iterating on the prompt until the terms land. Two or three rounds is typical.

Review workspace

```
mkdir -p ~/prj/review/_memos ~/prj/review/_notes ~/prj/review/_edits
```

The apply-only contract

Write this once and reuse it across every post and paper. It is the instrument that keeps authorship with the author.

```
cat > ~/prj/review/apply-contract.md <<'EOF'
# Editing contract

You are applying edits I dictated. You are a typist, not an editor.

## Absolute constraints

- NEVER compose, rewrite, smooth, condense, expand, or improve any prose.
- Text I marked between "quote" and "end quote" is MY verbatim wording.
  Insert it exactly as given. Do not adjust word choice or phrasing.
- Everything outside those markers is an instruction to you, not content.
- If an instruction is ambiguous, incomplete, or its anchor cannot be
  located, STOP and list it as a question. Never guess.
- Do not make any edit I did not request, however obvious the improvement.

## Do not touch

- Fenced code chunks: not one character
- Inline code, including inline `r` expressions
- The YAML header
- Citation keys, cross-references, figure/table labels, footnote markers
- Shortcodes, callouts, div fences, raw HTML
- Math, inline or display
- Any numeric result, statistic, or claim
- Whitespace outside the passages being edited
```

```
## Style of change

Minimal. Change exactly the span specified and nothing adjacent.
Preserve one-sentence-per-line formatting where present.
EOF
```

Optional: one-sentence-per-line reflow

`git add -p` splits hunks on lines. If a paragraph is a single long line, any one-word change makes the entire paragraph a single all-or-nothing hunk. Reflowing so each sentence occupies its own line makes hunks roughly sentence-sized. If adopted, do it as its own commit, across all posts, before any copyediting, so the reflow diff doesn't contaminate the prose diffs. Naive sentence-splitting breaks on abbreviations and inside code chunks; if the risk isn't worth it, skip the reflow and accept coarser hunks.

The Dictation Grammar

The core problem

Spoken editing instructions carry an ambiguity that written ones do not: the machine must distinguish **instructions addressed to it** from **prose to be inserted verbatim**. "Make that sentence shorter" and "replace it with, the estimand was underspecified" are the same acoustic stream.

The delimiter rule

Say **"quote ... end quote"** around any words that are your text. Never around anything else. Everything outside those markers is an instruction. Without this, the resolver has no reliable way to tell a directive from a payload.

Slot order

Four slots, always in the same sequence:

> **LOCATE → ANCHOR → OPERATION → PAYLOAD**

- **Locate**: coarse position: section name, paragraph number, "the bulleted list under Methods"
- **Anchor**: a distinctive phrase from the existing text
- **Operation**: replace, insert before/after, delete, transpose, split, join
- **Payload**: your new wording, delimited by quote / end quote

Examples

Replacement:

> "Section Results, second paragraph. The sentence beginning quote across five thousand replicates end quote. Replace quote ambiguous end quote with quote underspecified end quote."

Insertion:

> "Introduction, third paragraph. After the sentence ending quote in practice end quote, insert quote This is the assumption we relax below. end quote."

Deletion, reordering, and comment-only forms follow the same pattern: "Discussion, last paragraph. Delete the final sentence"; "Methods, the bulleted list. Transpose the second and third bullets"; "Whole post. Comment only, no edit: MMRM is used undefined here."

Session conventions

- **One audio file per post**, named with the post slug: the post is never spoken aloud, so a misheard slug cannot misroute an edit.
- **Say "next edit"** between items, for a reliable segmentation seam.
- **Do not dictate punctuation** inside payloads. Speak naturally, let punctuation be inferred, correct it at the diff.
- **Anchor on distinctive phrasing.** "The sentence beginning quote the model end quote" is a poor anchor; a specific numeric or unusual phrase is a good one.

Approximate anchors are expected

Spoken anchors will not match the source verbatim. Exact string matching fails on this, and it is precisely where a language model earns its place: resolving an approximate spoken description to the intended passage. This is also why the second checkpoint exists: the resolver's fuzzy match must be surfaced for human confirmation, never trusted silently.

Optional: numbered location markers

Instead of "the sentence beginning quote X end quote," LOCATE can simply be a count: "paragraph 5," "Figure 2," "Equation 3." This is safe precisely because it doesn't replace the anchor, it narrows where the anchor is checked: the resolver still verifies the quoted source text's occurrence count, just scoped to the numbered item. A miscounted paragraph produces a zero-occurrence flag, not a silent wrong-location edit.

For this to work, you and the resolver have to count identically, which means reading from a copy with the count actually printed on it. `voice-review-tag` (Python) writes a disposable, numbered sibling copy of a `.qmd`/`.Rmd`, and the real source is never touched:

```
voice-review-tag path/to/index.qmd
#  wrote: path/to/index.tagged.qmd
#    paragraphs=40 figures=9 tables=0 equations=0
```

Render *that* copy to get a numbered PDF, dictate from it, then resolve anchors against the real, untouched source. Clean up when done:

```
voice-review-tag path/to/index.qmd --clean
```

`--clean` matches `INPUT.tagged*`, broad enough to catch both `INPUT.tagged.qmd`/`.pdf`/`.html` and Quarto's `INPUT.tagged_files/` output directory (no dot before `_files`, which an earlier, narrower glob missed, leaving that directory behind after a real PDF render) and LaTeX auxiliary files from a PDF render. Verified by actually rendering a tagged copy to PDF and confirming `--clean` leaves nothing behind afterward.

Verified on a real post: found all 9 figures correctly (4 hand-written image references plus 5 `fig-cap:` chunk options), including two real multi-line cases a naive single-line regex missed on the first attempt: image markdown wrapping alt-text and attributes across several lines, and `fig-cap:` values continuing across multiple chunk-option lines. Also verified on a research document with a real display equation. **Not yet verified**: tables (no `tbl-cap` or pandoc table-caption example was found anywhere in this corpus to test against), and the counting is document-scoped only, no per-section reset yet.

Counting rules: skipped entirely, not numbered (YAML frontmatter, headings, fenced code chunk bodies, div fences including nested ones, list blocks, raw HTML). Everything else is a paragraph, table caption, figure, or equation. This is a pragmatic line-based heuristic, not a full CommonMark/Pandoc parser; if a count looks wrong, open the `.tagged` file directly rather than trusting the summary line.

Per-Post Procedure

Worked below on `pp-body-mass-prediction`, then run end to end for real against a different post later in this post.

Prepare the working copy

```
cd ~/prj/review/pp-body-mass-prediction
git status --short          # expect clean
git checkout -b copyedit
```

A dedicated branch means the entire review is revertible with a branch deletion, and the eventual merge is a single reviewable event.

Resolve the real target once, since `index.qmd` is a symlink in every post:

```
target=$(readlink -f index.qmd)   # analysis/report/index.qmd
```

Reading `index.qmd` directly is fine; that follows the symlink at the OS level. From verification onward, every `git diff`, `git add`, and `git restore` must target `$target`, not `index.qmd`.

Shortcut: `voice-review-dictate` combines dictation and transcription (and optionally numbered-location tagging) into one session:

```
voice-review-dictate pp-body-mass-prediction --seconds 45 --specialty penguin
```

It renders a numbered reading copy first (`--no-tag` to skip that), prompts you to confirm you're ready before it starts listening, records, transcribes, and opens the resulting transcript for Checkpoint 1, all in one command. Requires `--seconds` and `--specialty` explicitly; ryyblog posts only, since it assumes `$REVIEW/$SLUG/index.qmd` (the research-compendium trees have a different path and render wrapper and aren't supported by it yet). The manual steps below are still worth understanding for troubleshooting or when finer control is needed than the script gives.

Read the rendered post

Pre-rendered PDFs already exist in each post directory, so no render is needed for reading. Read on whatever surface supports careful reading: screen, tablet, or paper. The pipeline is indifferent; annotation is what it removes, not reading.

Dictate

Installed at `~/bin/voice-review-record`:

```
voice-review-record SLUG [--seconds N]

  --seconds N   Record for exactly N seconds, with a live countdown,
                then verify the actual file duration against N.
  (omitted)     Record until you press 'q' (open-ended, no countdown
                possible since the duration isn't known in advance).
```

```
voice-review-record pp-body-mass-prediction
```

Recording directly at 16 kHz mono eliminates a conversion step. Both `-ar 16000` and `-ac 1` are required and built into the script's fixed invocation. Omitting `-ac 1` when converting audio by hand is a common cause of degraded results.

Verifying a recording's actual duration is automatic with `--seconds N`: it checks the resulting file against N and warns if it fell more than a second short.

If a recording ends before you finish speaking, do not re-record everything. Record a short second memo for just the remaining edit(s), transcribe it separately, and append its transcript to the first one before resolving anchors, so the two edits still resolve in a single pass.

Transcribe

State the specialty by hand: decide which topic(s) this post draws on before transcribing. Batch mode cannot dictate the specialty for you, so write it as a one-line `.specialty` sidecar next to each memo:

```
echo "penguin" > ~/prj/review/_memos/pp-body-mass-prediction.specialty
```

Installed at `~/bin/voice-review-transcribe`:

```
voice-review-transcribe [--model PATH] [--force] [--seconds N]
                        [--specialty "name1 name2"] [SLUG]

  --model PATH        Use this whisper.cpp model instead of the default
                       large-v3.
  --force             Re-transcribe even if a transcript already exists.
  --seconds N         Only process the first N seconds of each memo.
  --specialty "LIST"  Space-separated specialty list, bypassing the
                       per-memo sidecar file entirely.
  SLUG                Transcribe only this one memo.
```

CHECKPOINT 1: read the transcript

```
cat ~/prj/review/_notes/pp-body-mass-prediction.txt
```

Correct any misheard domain terms by hand, and append them to the specialty file they belong to. Recall that errors here look plausible rather than garbled. This checkpoint takes under a minute and prevents a wrong edit from entering the pipeline with no later opportunity to catch it.

Resolve anchors

From within the post directory, prompt an assistant:

> Read `apply-contract.md` in `~/prj/review/`, then `index.qmd`, then `~/prj/review/_notes/pp-body-mass-prediction.txt`.
>
> The notes are dictated editing instructions. Text between "quote" and "end quote" is my verbatim wording; everything else is instruction to you.
>
> **Do not modify `index.qmd`.** Write `~/prj/review/_edits/pp-body-mass-prediction.md`. For each requested edit, record: the exact text from `index.qmd` that my anchor resolves to, quoted verbatim with its line number; the operation; my replacement or inserted text, verbatim as dictated; and the number of times that anchor string occurs in `index.qmd`.
>
> Flag any anchor occurring zero times or more than once. Do not choose among multiple matches. List ambiguous instructions as questions rather than resolving them.

Verify the anchors

CHECKPOINT 2: verify anchor resolution

Confirm for each entry:

- The quoted source text is the passage you meant
- The occurrence count is exactly 1
- Your replacement text is transcribed correctly
- **The quoted source text is complete**, not a truncated fragment: cross-check by reading the replacement text against the source: if the replacement contains a word that isn't present anywhere in the quoted source, that is the resolution step dropping part of the anchor, not a transcription error, and applying it as given produces a duplicated word.

This is a fast read: you are checking locations, not prose. It is the checkpoint that makes the automated application safe.

Apply

> Apply the edits in `~/prj/review/_edits/pp-body-mass-prediction.md` to `index.qmd` exactly as written, under the constraints in `apply-contract.md`. Change nothing else.

If your tooling refuses to write through a symlink, apply to `$target` instead of `index.qmd` directly.

Verify the result

Read the diff against `$target`, since diffing the `index.qmd` symlink path itself silently reports no change:

```
git diff --word-diff=color "$target"
```

You are verifying **obedience**, not judging suggestions: did it change exactly what `edits.md` specified, and nothing more? Confirm code chunks are byte-identical, and confirm the post still builds: reading `index.qmd` for the render is fine, since Quarto follows the symlink the same way `cat` does.

Stage selectively

Before staging, check whether this post's rendered output (`index.html` or similar) is conventionally committed together with source prose edits or separately:

```
git log --oneline -3 -- index.html "$target"
```

If a render step left `index.html` modified and the post's own history keeps it separate, restore it before staging. Walk the hunks:

```
git add -p "$target"
```

`y` accept, `n` reject, `s` split into smaller hunks, `e` open the patch in your editor to hand-modify it. Then discard everything not accepted:

```
git restore "$target"
```

`git restore <file>` without `--staged` overwrites the working tree from the index. Since the index now holds only accepted hunks, this removes every rejected edit. **Do not commit before this step**: reverting selected hunks post-commit inverts the polarity of `git add -p`'s keys, which is easy to get wrong 57 times consecutively.

Commit and push

```
git commit -m "copyedit: prose review pass"
git push -u origin copyedit
```

Merge to `main` when satisfied, then bring the parent tree's copy forward.

A Worked Example, Run for Real

Every step above is worked here for real: two edits to the opening paragraph of `pp-eda`'s Introduction, from **one** dictation take, run in this author's working clone (`~/prj/review/pp-eda`, branch `copyedit`), committed as `1f0dffb`. Every transcript and script output below is genuine output from this run.

The paragraph and the two edits

`analysis/report/index.qmd`, lines 83–88 before editing:

```
83  How much can a single morphometric measurement reveal
84  about an organism's body condition? The Palmer Penguins
85  dataset provides an opportunity to test the claim that
86  flipper length is a strong predictor of body mass in
87  penguins. The exercise turns out to be one of the most
88  instructive introductions to regression available.
```

Two edits, decided before recording:

- **Replace**: "test the claim that flipper length is a strong predictor of body mass in penguins" → "test whether flipper length predicts body mass in penguins."
- **Insert**, after "…instructive introductions to regression available.": "The dataset's modest size makes it practical for live coding."

Record and transcribe, in one session

```
voice-review-dictate pp-eda --seconds 45 --specialty penguin --no-tag
```

One take, real, unedited, 39.92 seconds captured against a 45-second request:

Introduction, first paragraph The sentence beginning, “The Palmer penguins dataset.” Replace, “Test the claim that flipper length is a strong predictor of body mass in penguins.” With, “Test whether flipper length predicts body mass in penguins.” Introduction, first paragraph. After the sentence ending, “Instructive introductions to regression available” end quote, insert quote “The dataset’s modest size makes it practical for live coding”. end quote.

Two things worth noticing, both real:

- **39.92 seconds captured against a 45-second request**, again short, though not enough to lose content this time; requested duration is never a guarantee.
- **“quote”/“end quote” rendered inconsistently.** Sometimes Whisper transcribed the literal words, sometimes it rendered the delimited span as typographic quotation marks instead, and once it did both in the same sentence pair. Both forms carry the same meaning to a human reader, but this widens what the resolver has to recognize as a delimiter, not previously observed in this form.

CHECKPOINT 1: read the transcript

One correction: “The Palmer penguins dataset” → “The Palmer Penguins dataset” (casing lost despite `vocab/penguin.txt` priming the exact phrase). Both payloads (the replacement and the inserted sentence) transcribed completely and correctly, apostrophe included. No content correction needed this time, only cosmetic.

This same document, tagged for numbered locations, finds **40 paragraphs, 9 figures**, and the edited paragraph is **P2, not P1**: a photo-credit line between the hero image and the Introduction heading claims P1, a miscount a human skimming the PDF could easily make.

Resolve anchors

A real casing subtlety the resolver had to get right: the transcript renders the anchor phrase capitalized (“Test the claim that…”, “Instructive introductions…”) because Whisper capitalizes the start of a quoted span stylistically. The actual source text is **lowercase** mid-sentence. Verified directly: searching for the transcript’s exact capitalization finds nothing; the real casing finds it once:

```
$ python3 -c "
import re
text = open('analysis/report/index.qmd').read()
norm = re.sub(r'\s+', ' ', text)
a = ('flipper length is a strong predictor of body mass in '
     'penguins')
print(norm.count('Test the claim that ' + a))  # transcript's casing
print(norm.count('test the claim that ' + a))  # real file's casing
"
```

```
0
1
```

`~/prj/review/_edits/pp-eda.md`, written to match the real file's actual casing, not the transcript's:

```
## Edit 1: replace (index.qmd, lines 85-87)
Source text: "test the claim that
flipper length is a strong predictor of body mass in
penguins"
Occurrences: 1
Replace with: "test whether flipper length predicts body mass in penguins"

## Edit 2: insert after (index.qmd, line 88)
Source text: "instructive introductions to regression available."
Occurrences: 1
Insert after: "The dataset's modest size makes it practical for live
coding."
```

> **CHECKPOINT 2: verify anchor resolution**
>
> Both occurrence counts read `1`. Both quoted source passages matched the real file, in the real file's actual casing, not the transcript's stylized capitalization, which would have found zero occurrences had it been copied verbatim. Both replacement/insertion strings matched the confirmed transcript payloads. No truncation this time.

Apply, verify, stage, commit

Three real symlink-related steps every ryyblog post needs:

```
target=$(readlink -f index.qmd)      # analysis/report/index.qmd
# ... apply the edit to $target, not index.qmd ...
git diff --word-diff=color "$target" # confirms exactly the two edits
quarto render index.qmd              # Output created: index.html
git restore index.html               # this repo commits rendered HTML
                                      # separately from source prose
git add "$target"
git commit -m "copyedit: tighten intro claim, add live-coding note"
```

Result: commit `1f0dffb` on branch `copyedit`. **Not pushed**: pushing and merging to `main` are separate, explicitly-confirmed steps.

What each script does underneath

`voice-review-dictate` is the orchestrator:

```
voice-review-tag "$TARGET"                    # optional, --no-tag skips it
quarto render "analysis/report/$tagged" --to pdf
read -r -p "Press Enter when ready to record ($SLUG, ${DURATION}s)... " _
voice-review-record "$SLUG" --seconds "$DURATION"
echo "$SPECIALTY" > "$REVIEW/_memos/$SLUG.specialty"
voice-review-transcribe --force "$SLUG"
"$EDITOR_CMD" "$REVIEW/_notes/$SLUG.txt"
```

Requires `--seconds` and `--specialty` explicitly, no defaults, since both are decisions a person has to make, not something the script should guess. The confirmation prompt is a real blocking wait for a keypress, not cosmetic: it exists so tagging and rendering, which can take tens of seconds, never eat into the recording window.

`voice-review-record` wraps `ffmpeg`, backgrounding the recording process and polling it once a second with `kill -0 "$pid"` (a signal-0 send that only checks whether the process is alive), printing the remaining count and catching early exit immediately. Afterward, `ffprobe` reads the real file length back and compares it to what was requested.

`voice-review-transcribe` wraps `whisper-cli`, concatenating `core.txt` with whichever specialty file(s) apply into the `--prompt` argument.

`voice-review-tag` is a regex-based block classifier written in Python (not a full CommonMark/Pandoc parser) that walks a document line by line and writes a disposable, numbered sibling file; the real source is opened read-only, never modified.

Batch Operation

Once the loop is validated on three posts, batch the mechanical stages.

Transcription

Installed at `~/bin/voice-review-transcribe`, scanning `~/prj/review/_memos/` for every memo with a `.specialty` sidecar written, skipping anything without one so the specialty decision is never silently guessed.

Bug found and fixed while using this for real: a split-recording memo's `-part2` slug has no working clone of its own (it shares the original slug's clone), so the directory check initially skipped it entirely. Fixed by stripping a trailing `-partN` suffix and checking the base slug's clone directory when the exact slug has none.

Corpus status

Installed at `~/bin/voice-review-status`: a table of memo/notes/edits/diff status across every working clone.

Recommended cadence

1. Read and dictate one prefix group in a sitting (5–10 posts), writing each memo's `.specialty` sidecar as you go.
2. Run `voice-review-transcribe` once for the group.
3. Read all transcripts in one pass (Checkpoint 1), updating the relevant specialty files.
4. Run resolution per post; read all edit files in one pass (Checkpoint 2).
5. Apply, diff, stage, commit per post.

Batching by stage rather than by post keeps you in one mode of attention at a time, which is the ergonomic argument for the whole design.

Scale estimate

At 57 posts, transcription is the only stage with meaningful machine cost: roughly 3 s of processing plus 1.7 s of model load per minute of audio on the M1. Fifty-seven five-minute memos is under half an hour of unattended compute. Every other stage is human time, which is the point: the pipeline reallocates human time from transcription to reading, rather than eliminating it.

Failure Modes and Controls

Plausible transcription errors. Domain terminology is replaced with fluent, incorrect alternatives that read as intentional. Observed: "Rubin's rules" → "Reuben"; an unidentified term → "bitmap." Controls: vocabulary prompt, Checkpoint 1, cumulative maintenance of the prompt file. Residual risk: a misheard term inside a payload that survives Checkpoint 1 will be inserted verbatim into the post; the final diff review is the last line of defence.

Silent anchor misapplication. An anchor phrase occurring twice in a post is resolved to the wrong occurrence. The edit is applied correctly, to the wrong place, and reads as intentional. Controls: mandatory occurrence count in the edit file; explicit instruction never to choose among multiple matches; Checkpoint 2; anchoring on distinctive phrasing. This is the single most important control in the pipeline: the reason resolution and application are separate stages with a human between them.

Composition drift. The model smooths, rewrites, or extends prose that was not part of a requested edit, precisely the outcome the pipeline exists to prevent. Controls: the apply-contract; separation of resolution from application; word-diff review of every hunk; `git add -p` requiring affirmative acceptance.

Silence hallucination. Whisper generates plausible text during extended silence. Controls: keep memos continuous; if the symptom appears, preprocess with `whisper-vad-speech-segments`.

Code chunk modification. An edit touches executable code, changing computed results in a compendium whose environment is pinned by `renv.lock`. Controls: explicit prohibition in the contract; mechanical byte-identity check; `quarto render` before commit.

Coarse hunk granularity. Paragraph-per-line formatting makes `git add -p` all-or-nothing at paragraph scale. Controls: optional one-sentence-per-line reflow, committed separately; use `e` within `git add -p` to hand-edit oversized hunks.

Resolution-step anchor truncation. The resolution step correctly locates the anchor but quotes an incomplete span of it. Observed directly: a source-text quote stopped one word short of the actual passage, which would have produced a duplicated word had it been applied as written. Distinct from silent anchor misapplication: the anchor is not ambiguous or misplaced, the quotation of it is simply cut short. Control: Checkpoint 2 cross-checks the replacement text against the quoted source.

Symlinked target files. `index.qmd` is a symlink in every ryyblog post. Diffing or staging the symlink path instead of its target does not error: it silently reports no change on a post that was in fact just edited. Control: resolve `$target` once per post and use it for every diff/stage/restore command.

Alternatives Considered

Visual editor. RStudio and Positron offer a near-WYSIWYG editing mode for `.qmd` that saves back to markdown, eliminating the transcription problem entirely with no AI involvement. Worth testing before building anything else: if reading in the visual editor is comfortable, that is the simplest possible answer. The case against, for this corpus: files here are embedded in compendia with executable chunks and complex Quarto features that visual editors render imperfectly.

CriticMarkup. A plain-text editorial markup standard (`{++ insertion ++}`, `{-- deletion --}`, `{~~old~>new~~}`) that is positionally exact, resolvable by a short script with no model involvement. Its disadvantage is that it requires typing at the keyboard with the source file open, which is the situation dictation was chosen to avoid. A viable hybrid: dictate, but have the resolution stage emit CriticMarkup rather than a prose edit file.

Proofreaders' marks on paper. The traditional standard is well-defined, but a caret encodes position spatially, and that survives only if the reader has the page: a photograph of hand-drawn annotations forces a model to infer word boundaries silently.

Deterministic tooling first. Independent of any of the above: Vale with a custom style for house rules, hunspell on prose regions, global terminology normalization via `grep`/`sed` in a single commit rather than 57 separate decisions.

An observation-only AI pass. The one AI-authored artifact worth keeping, because it proposes no text: a run that reads each post and emits observations only: "MMRM used undefined here; defined in the cross-validation post." This substitutes for some reading labor without touching

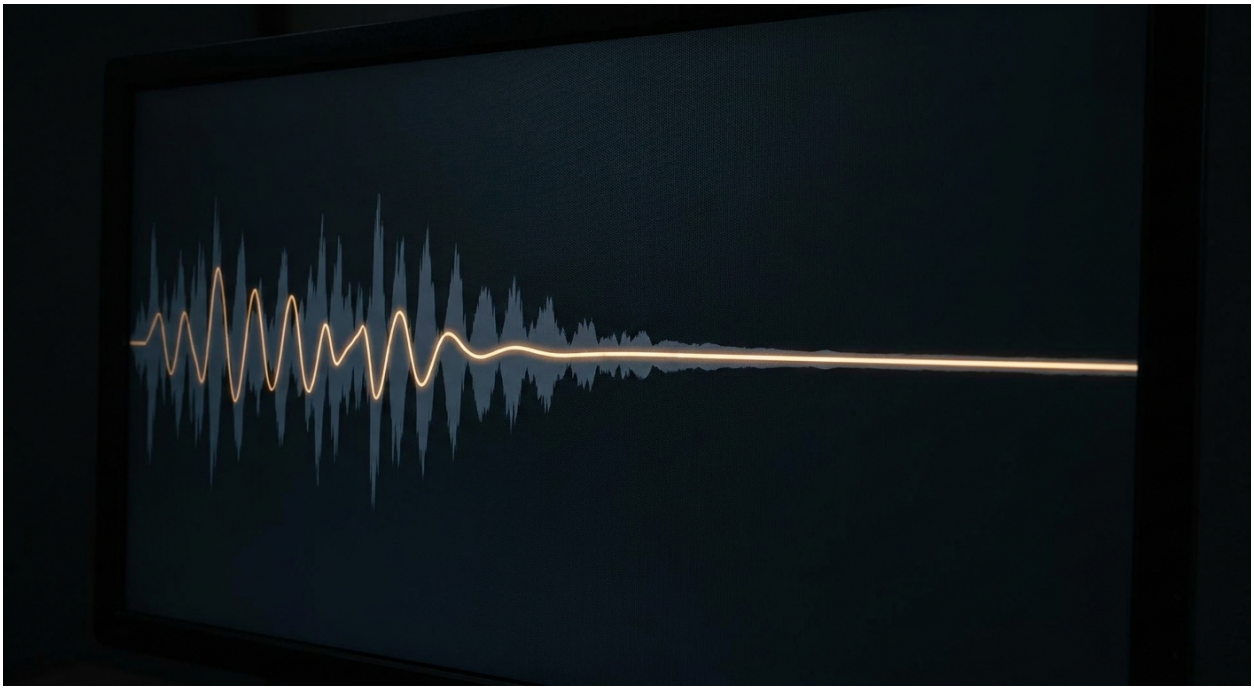
the prose, the closest machine equivalent to a second reader who marks a margin with a question mark.

Generalizing Beyond the Blog

This pipeline extends to two research-compendium trees holding actual papers and manuscripts. A third, confidential tree is explicitly out of scope: its repositories must never acquire a GitHub remote or be pushed anywhere, which the clone-and-push working-clone workflow can't accommodate as written.

Three structural facts break assumptions the blog-specific sections depend on: **no parent-site wrapper** (each numbered project directory wraps an independently named inner repo, which is the actual git repository); **no fixed target filename** (the report lives under `analysis/report/`, but the filename varies, and a repo is not one review unit; a document is, since some repos hold several manuscripts); **no prefix-group naming convention** (nothing analogous to `cln-`/`pp-`, which is exactly why the vocabulary specialty is hand-dictated per document rather than derived from anything structural).

A discovery script enumerates candidate review units, finding 90 candidate documents across both trees as they stand today (verified directly), correctly excluding presentation talking points, archived content, and a live Dropbox conflicted-copy artifact found during the survey. Both trees ship a `tools/render.sh` wrapper required in place of calling `rmarkdown::render()` directly, because it stamps a provenance footer and stages a dated, versioned copy tracked in a manifest file. Everything else (the dictation grammar, the apply-only contract, both checkpoints, the diff-and-stage discipline, the failure-mode controls) is domain-agnostic and applies exactly as written for a blog post.



Lessons Learned

Conceptual:

- Approximate anchors plus occurrence-count verification are safer than exact-match tagging, because a miscount fails loudly (a zero-occurrence flag) rather than silently.
- Vocabulary priming reduces but never eliminates plausible-sounding transcription errors; the residual risk always lands in the final diff review, not before it.
- Separating resolution from application (a human confirms before anything is applied) is the single most important structural decision in the whole design.
- A pipeline like this is a discipline as much as a set of scripts. The checkpoints exist because trusting quietly compounds risk across a corpus this size.

Technical:

- `index.qmd` is a symlink in every one of the 57 ryyblog posts; diffing or staging the symlink path itself is a silent no-op, not an error.
- `large-v3` beat `large-v3-turbo` head-to-head on domain terminology at roughly double the processing time, worth it for this corpus's jargon density.
- Whisper's `--prompt` has an effective ~220-token ceiling and truncates from the front; failing terms belong at the end of a specialty file.
- `ffmpeg`'s `-t N` is a request, not a guarantee: verified more than once with real shortfalls, including inside the final worked example.

Gotchas:

- A split-recording memo's `-partN` suffix has no working clone of its own: the transcribe script had to be taught to strip the suffix.
- The resolution step can locate an anchor correctly but quote it incompletely, producing a duplicated word if applied blindly.
- Whisper renders "quote"/"end quote" inconsistently (sometimes literal words, sometimes typographic punctuation) within the same utterance.
- `voice-review-tag`'s `--clean` initially missed Quarto's `_files/` output directory, because of a dot-versus-underscore glob mismatch.

Limitations

- Table numbering in `voice-review-tag` is unverified: no real `tbl-cap` or pandoc table-caption example exists anywhere in this corpus to test against.
- `voice-review-dictate` is ryyblog-only; the research-compendium trees have a different path and render wrapper it doesn't support yet.

- The research-compendium target-file discovery heuristic excludes `vignettes/*.Rmd` and `analysis/scripts/*.Rmd` on an unconfirmed assumption that they aren't "papers."
- The confidential tree is out of scope entirely: the clone-and-push workflow can't apply to material that must never touch a GitHub remote.
- Live recording through an orchestrating process, rather than a human directly running the tool, proved unreliable: automated attempts produced cut-off or silence-hallucinated takes; only a human-run session reliably worked.

Opportunities for Improvement

1. Extend `voice-review-dictate` to the research-compendium trees once the path and render-wrapper differences are handled.
2. Add per-section paragraph-number resets to `voice-review-tag` instead of document-wide-only counting.
3. Verify table numbering against a real `tbl-cap` or pandoc-caption example once one exists in the corpus.
4. Build the observation-only AI pass as a real script, not just a design sketch.
5. Investigate whether CriticMarkup as the resolution-stage output format reduces the anchor-truncation failure mode structurally, rather than relying on Checkpoint 2 alone.
6. A local-only variant of the pipeline for the confidential tree, with no push step, to bring that material into scope.

Wrapping Up

I started this expecting the interesting engineering to be in the speech recognition. It wasn't. The interesting engineering was in the seams between mechanical steps: a symlink that silently breaks `git diff`, a prompt window that truncates from the wrong end, a resolution step that quotes an anchor one word short. None of these are Whisper's fault, and none of them would have surfaced from reading the design on paper. They surfaced because the pipeline was actually run, against a real paragraph, more than once, until a take succeeded.

The two checkpoints are the part of this design I'd defend most stubbornly if asked to cut something for speed. Every real bug documented here (the truncated anchor, the casing mismatch, the split-recording clone lookup) was either caught by a checkpoint before it did damage, or found only because the checkpoint's own logic was interrogated closely enough to notice something was wrong. A pipeline that skips the human confirmation step to go faster is a pipeline that will eventually apply a confident, plausible, wrong edit, and never know it happened.

If there's one thing worth taking from this beyond the specific tooling: build for the failure you can't see, not just the failure you can. A silent no-op diff and a stylishly-capitalized anchor phrase are both far more dangerous than an obvious crash, because nothing tells you they happened.

Main takeaways: separate resolution from application; verify occurrence counts, not just presence; never trust a diff against a path you haven't confirmed is the real file; run the thing for real before trusting the design.

If you're trying this yourself: start with the calibration run on your own voice before touching a real document, and don't skip Checkpoint 2 even once: the anchor-truncation bug in this post's worked example was caught there, and nowhere else would have caught it.

See Also

- Code Review for AI-Assisted R Package Development: the same separation-of-concerns philosophy applied to reviewing generated code instead of dictated prose.
- LLM-Augmented Editing for the Workflow Construct: the broader pattern this pipeline's resolution step is one instance of.
- Constructing a Reproducible Blog Post Using zzcollab Tools: the compendium structure this post itself was built with.
- Palmer Penguins Part 1: Exploratory Data Analysis: the real post this pipeline was run against, worked example included.

Related posts in this cluster

This post is part of the *Workflow Extensions* series.

1. LLM-Augmented Editing for the Workflow Construct
2. A pocket terminal for your Linux laptop with ttyd and Tailscale
3. **A Voice-Driven Copyedit Pipeline for Quarto Blogs and Research Papers** (this post)