FIELD HANDBOOK · VOL. I OF II

Building jlegal.pro with AI Coding Agents

How one non-developer, working with Claude Code (and briefly Cursor), turned a single legal report into a 54-page evidentiary website in roughly seven weeks — the workflow, the guardrails, and the exact habits worth copying.

SUBJECT

jlegal.pro repository

WINDOW STUDIED

May 8 – Jul 3, 2026

METHOD

5-agent parallel repo audit

712 COMMITS	**54** HTML PAGES
~30 SOLO PRS MERGED	**11** LIVE WORKTREES

CONTENTS

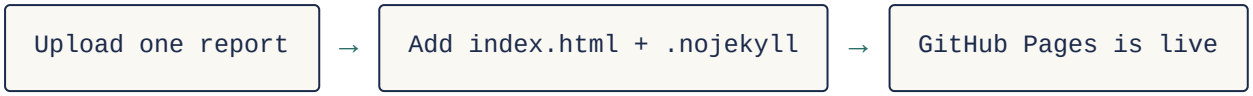
HOW TO READ THIS

This handbook describes *process* — the mechanics of using AI coding agents to build and maintain a large, fast-moving, high-stakes static website without being a professional developer. It does not cover the site's subject matter, arguments, or claims. For an assessment of the site's investigative and rhetorical strategy, see Volume II, *Strategy, Tactics & Skills*.

01 · FOUNDATION

The Origin: One Document, One Day

The repository's first real commit isn't a scaffold or a template — it's a single uploaded HTML report. Within minutes, a second commit adds `index.html` and a bare `.nojekyll` marker so GitHub Pages would serve it with no build step. A third adds a README. That's the entire "setup phase": three commits, same day, and the site was live.



Everything that followed — 711 more commits, 54 pages, interactive graphs, an OSINT database — was grown from that one page by **iterative accretion**, not planned architecture. The commit-message verb counts make the point bluntly: "Add"-type commits outnumber "Fix"-type commits roughly 4-to-1 across the whole history. This is a site that was built by continuously asking an agent to add the next thing, not by drafting a spec and executing it.

WHY THIS MATTERS FOR A NON-DEVELOPER

You do not need an architecture before you start. The lesson from jlegal.pro's first three commits is that the fastest path to a real, live, working site is: write the first real document, hand it to an agent, ask for hosting, and iterate from there. Structure gets imposed later, deliberately, once there's enough content to see what it should be (see §8).

The Three-Phase Arc

Read across the full commit history, the project has three distinct phases, each triggered by a specific kind of commit rather than a calendar boundary.

PHASE 1 — MAY 8 TO ~MAY 19

THE MONOLITH

A single consolidated report grows by direct edits: new timeline entries, addenda, section additions ("Add section 4.7.A: Reframing as Frame-Up Predicate"). Two large kitchen-sink documents are created on day one — `master-record.html` and the Consolidated Report — and everything for the next two weeks is appended to them.

PHASE 2 — ~MAY 20 TO MID-JUNE

THE DECOMPOSITION

The monolith is broken into topic pages. `story.html`, `three-surfaces.html`, `convergence.html`, and a dozen others are spun out one at a time. One commit in this window shows the extraction caught mid-act: 812 lines removed from the Consolidated Report, 955 lines added to a new standalone page. In the same window, the site starts speaking to machines as well as people — `sitemap.xml`, `robots.txt`, and `llms.txt` appear together in one commit, followed shortly by `agents.md` and `ai-agents.json`.

PHASE 3 — MID-JUNE TO PRESENT

THE CONSOLIDATION

With 40+ pages now diverging in small ways (different nav markup, different footer structure), a single patch script (`scripts/enhance_chrome.py`) is written and run once across the whole site to force every page onto one shared header/footer component. This is the only site-wide "refactor" in the project's life, and it's a 500-line Python regex tool, not a framework migration.

> *The site didn't start structured and get built out. It started as one page and got organized in place, twice: once into many pages, once back into visual consistency.*

The Checkpoint Rule

The single most important governance artifact in the whole repository is a file that no longer exists in the working tree — `CLAUDE.md`, created June 12 and deleted twice shortly after (it was a living work-order document, not meant to persist). Recovered from git history, its "Workflow & Checkpoints" section is the clearest statement anywhere of how the user actually directed the agent:

> For each CRITICAL item: 1. AI reads the requirement 2. AI identifies the exact files and lines to change 3. AI **ASKS** the user for explicit approval before proceeding 4. User approves or adjusts 5. AI makes the change 6. AI shows the result and asks for confirmation.
>
> **Do NOT:** Commit changes without checkpoints. Make changes to files not mentioned in this document. Skip asking about optional items.
>
> CLAUDE.MD, COMMIT A41DF6A (RECOVERED VIA GIT SHOW)

The same document contains an explicit protocol for subjective, taste-based changes — the kind an agent has no ground truth for:

> This is subjective. Before changing it: take a screenshot. Ask the user whether it feels better or worse. Only commit if they approve.
>
> CLAUDE.MD — SUBJECTIVE-CHANGE PROTOCOL

And it explicitly deferred large, high-risk, low-urgency refactors rather than letting the agent "improve" things unprompted:

> Extract Shared CSS/JS into External Files — High leverage long-term but medium effort; touches every file; high breakage risk if not careful. Save for a dedicated refactoring session.
>
> CLAUDE.MD — "LOW PRIORITY: DON'T DO THESE YET"

THE TRANSFERABLE RULE

Write down, in a file the agent reads first, which changes require your sign-off before a commit and which don't. Critical/user-facing changes: ask first, show the diff, wait. Small additive changes: proceed and show the result. Anything that touches every file at once: schedule it deliberately, never let it happen as a side effect of an unrelated request.

Branches, PRs, and Worktrees — For an Audience of One

This is a solo project. There is no second human reviewer, no team, no CI pipeline (`.github/` does not exist anywhere in the repository's history). And yet it has merged roughly 30 pull requests and, as of the most recent commits, has **11 separate git worktrees** checked out simultaneously, each on its own branch, each mid-task.

WHY PRS, SOLO

Every merged PR traces to a Claude-Code-generated branch name — `claude/add-legal-correspondence-U2Kxe`, `claude/add-critical-findings-section-7E94o` — the agent's own task-branching convention. The PR is not for a second reviewer; it's a forced diff view the author reads before merging. It substitutes for peer review when there is no peer.

WHY WORKTREES, SOLO

Each worktree (`fair-housing-page`, `footer-greystar-link`, `mobile-chrome-consistency` …) is an isolated checkout for one agent task in flight. Running several at once means one agent's half-finished edit never collides with another's, even with a single person directing all of them.

One branch, `cursor/website-mobile-footer-fixes-3586`, and 16 commits authored by `Cursor Agent <cursoragent@cursor.com>` confirm a second AI coding tool was used alongside Claude Code at least once — the workflow isn't married to one vendor, it's married to the branch-review-merge discipline itself.

THE TRANSFERABLE RULE

Even alone, don't let an agent commit straight to your live branch. Let it open a branch (or worktree) per task and show you a diff before it lands. The review step costs you thirty seconds and catches the mistake before it's public.

Briefing the Agent: `agents.md`

Every new agent session starts with no memory of the last one. jlegal.pro solves this with a 39KB briefing document, updated in place as the case evolves, that any fresh agent reads first. It contains a case summary, a full annotated inventory of all 54 pages, party/entity tables, legal-citation tables — and, most importantly, a section that exists purely to keep agents from making the site worse:

> **Must-NOT-Claim List** — "#1 worst" / "most violations in its class": 6 luxury buildings have more open violations. [Every forbidden claim ships with the specific reason it's false.]
>
> AGENTS.MD, §6 "VERIFIED L&I / PROPERTY DATA (LOCKED)"

This single section is doing more work than it looks like. It's a standing fact-check that travels with the project itself, so a future agent session — weeks later, with no memory of why a claim was rejected the first time — can't reintroduce it. The document closes by addressing the reader directly:

> If you are an AI agent reading this, you now have comprehensive context to review, analyze, or assist with any aspect of this case.
>
> AGENTS.MD, CLOSING LINE

THE TRANSFERABLE RULE

For any project that will span more than one agent session, keep one file — read first, every session — that holds facts the agent must get right, claims it must never make again, and a map of what already exists. Treat it as the project's memory, because the agent has none of its own between sessions.

Making Agents Check Each Other

Twice in the project's history, the user had one agent formally audit another agent's work rather than accepting either output at face value.

Case 1 — Rejecting a refactor

One agent proposed a 15-point "tone softening" pass: a filter to flag words like *criminal*, *conspiracy*, *motive*, and *bad faith*, plus new hedging language throughout. A second agent (Claude, Opus) was assigned to assess the proposal before anything was touched, and rejected nearly all of it:

> The proposal treats the site as though it were a legal filing that needs to meet evidentiary standards of neutrality. It is not. The site's credibility comes from its specificity, directness, and documentary rigor.
>
> `AI-REFACTOR-ASSESSMENT.MD, JUNE 6`

Only 3 of the 15 proposed changes survived review.

Case 2 — A structured six-agent panel

A separate self-audit used a deliberately adversarial panel structure, run entirely with AI agents standing in for a review team that didn't otherwise exist:

> Reviewed by 6 AI agents (2 pro, 2 con, 2 judges). Two agents assessed strengths and opportunities. Two assessed weaknesses, risks, and technical problems. Two weighed the findings and ranked recommendations by impact, effort, and risk.
>
> `SITE-REVIEW-JUN6-2026.MD`

The output was a ranked, ten-item punch list scored by effort and impact — including technical hardening items like self-hosting a hotlinked image (so it can't be swapped or pulled by its host) and pinning CDN scripts with subresource-integrity hashes. It also explicitly declined several plausible-sounding changes as not worth the effort yet — an example of scope discipline, not just change-making.

THE TRANSFERABLE RULE

When a change is structural, tonal, or otherwise hard to evaluate in isolation, don't let the agent that proposed it also be the only one that approves it. Ask a second agent — or the same agent in a fresh, skeptical session — to argue against the change before it ships. Panels of 4-6 (pro / con / judge) surface risks a single pass misses, including risks to the project's own credibility.

Burst-and-Quiet Cadence

712 commits land across 52 distinct days spanning roughly seven weeks — not a steady daily drip, but sharp bursts separated by near-silence.

BUSIEST SINGLE-DAY COMMIT COUNTS

DATE	COMMITS	WHAT SHIPPED IN THAT SITTING
Jun 19	63	An entire "About the Author" feature, scaffold to nav integration, in 44 small increments
May 28	45	A full narrative-page rebuild driven by a 4-persona review (journalist, reader, lawyer, designer)
May 21	37	Countdown banner and header redesign iteration
Jun 18	34	Continued page build-out
May 22	31	Continued page build-out

By contrast, several days in the same window show a single commit, or none. The pattern reads as one continuous sitting per burst — a single focused session where dozens of small, individually reviewable commits converge on one feature — rather than dozens of independent decisions to open the laptop.

THE TRANSFERABLE RULE

Don't measure progress by commits-per-day. A 63-commit day and a 1-commit day can represent the same amount of "sitting down to work" — the difference is how granularly the agent checkpoints its own progress within a session. Small, frequent commits inside one sitting cost nothing and make any single step easy to revert.

Fixing Drift Without a Framework

By late June, 40+ hand-grown HTML pages had quietly diverged — different nav markup, one page (`kane-retaliation.html`) with an entirely bespoke header. There was no component system to blame; there never had been one. The fix wasn't to adopt React or a static-site generator. It was a single, purpose-built, idempotent script.

WHAT `SCRIPTS/ENHANCE_CHROME.PY` ACTUALLY DOES

A regex-based Python patcher that (1) checks each page for a marker string proving it's already been updated, skipping it if so; (2) replaces old hand-written nav markup — matched against the exact old inline-style strings, including a special case just for the one bespoke page — with one canonical `jl-compact-nav` snippet; (3) injects one shared `<style>` block before the footer if it isn't already there. It can be re-run safely at any time.

THE TRANSFERABLE RULE

You don't need a build system to keep 50 static pages consistent. You need one small, idempotent script that can be re-run without double-applying itself, run once across the whole site when drift accumulates. This is a far smaller lift than migrating to a framework, and it matches the actual scale of a personal site.

The Whole Technical Stack

Notable for what it doesn't include. No React, no Vue, no static-site generator, no CSS framework, no bundler, no transpiler, no linter, no CI. The entire toolchain:

LAYER	CHOICE	WHY IT'S ENOUGH
Hosting	GitHub Pages, `CNAME` + `.nojekyll`	Zero-config static serving straight from `main`
Markup	Plain HTML, one file per page	No template engine needed at 54 pages
Styling	Inline `<style>` per page	Zero build step; consistency enforced by the patch script (§8) instead of a shared stylesheet
Interactivity	ApexCharts, D3.js, Chart.js — loaded per-page via CDN, only where needed	No page pays for a library it doesn't use
Only local dependency	`playwright`	Renders ~40 branded Open Graph share-images and runs local screenshot QA — not part of the deploy path at all
Site-wide consistency tool	`scripts/enhance_chrome.py`	One regex patcher, run on demand (§8)

The Reusable Playbook

1 Start with the real document, not a scaffold.

Write (or paste) the actual first piece of content. Ask the agent to make it live. Structure comes later, once you can see what you actually built.

2 Write your checkpoint rule down before you need it.

One short

file: which changes need your yes/no before committing, which don't, and what's explicitly off-limits for now.

3 Let the agent use branches even when you're the only reviewer.

A branch-and-diff step costs seconds and is the cheapest mistake-catcher available.

4 Keep one briefing file that survives between sessions.

Facts to get right, claims never to repeat, a map of what exists — because the next agent session remembers none of it on its own.

5 When a change is subjective or structural, get a second, skeptical opinion

— a different agent, or the same agent told

explicitly to argue against the idea — before it ships.

6 Don't reach for a framework to fix drift.

One small, idempotent script, run when things have visibly diverged, is usually enough at the scale of a personal project.

7 Measure sittings, not days.

Commit small

and
often
within
a
session;
don't
worry
about
the
gaps
between
sessions.

SOURCES

Compiled from a five-stream parallel audit of the `jlegal.pro` repository: full commit history (712 commits, May 8 – Jul 3 2026), recovered governance documents (`CLAUDE.md` , `agents.md` , `ai-refactor-assessment.md` , `site-review-jun6-2026.md`), build tooling (`scripts/` , `package.json`), and live repository state (branches, worktrees). This document describes the build *process* only. See Volume II for an assessment of the site's investigative and rhetorical content.