

N9LYA TECHNICAL GUIDES

Building With Claude

A Field Guide to AI Partnership

Jerome M. Kutche

N9LYA

Second Edition · September 2026

Introduction: From Asking Questions to Running Projects

There's a moment that happens to almost everyone who starts using an AI assistant seriously. It starts small — you ask it to explain something, fix a typo, summarize a document. Then one day you catch yourself doing something different: you're not asking a question anymore. You're delegating a piece of a real project and expecting it back, done.

That shift — from question-asking to project-running — is what this book is about.

Most books about “AI collaboration” are written in the abstract. They talk about prompting technique, productivity hacks, or speculative futures. This book isn't that. It's a field guide built from an actual, ongoing technical partnership: a home lab running a dozen virtualized services, a ham radio network with its own routing and infrastructure, monitoring pipelines, credential systems, and a slow, deliberate migration from “I do everything myself” to “we build this together.”

The projects in this book are real. A Proxmox server with more than a dozen containers and virtual machines, each with its own job — DNS filtering, file sync, voice processing, a firewall doing double duty as a VPN gateway, a workflow automation engine stitching a dozen small services into one system. A packet radio network that predates most commercial internet infrastructure in the area, still being extended and documented today. A newsletter that goes out to a radio club, built and rebuilt by hand until it became a repeatable pipeline. A series of technical books, produced chapter by chapter over months.

None of this is hypothetical. It's the material this book is drawn from.

If you're a systems administrator, a homelab hobbyist, a ham radio operator, a maker, or anyone who runs real infrastructure and has started leaning on an AI assistant for more than syntax help, this book is for you. It won't teach you “10 prompts to be more productive.” It will show you, chapter by chapter, how a real technical partnership between a person and an AI actually develops — where the labor gets divided, where trust gets earned and sometimes lost, what it looks like when things break in the middle of a project, and where the human hand is still, stubbornly, required.

By the end, you'll have a practical framework you can apply to your own projects — not a theory of AI collaboration, but a field-tested one.

How to Use This Book

Parts I and II are the framework — division of labor, verification, trust, and the honest boundaries of the relationship. Read them in order.

Part III is practice — real incidents, documentation, multi-session workflows, and producing actual artifacts together.

Part IV is scale and limits — what changes as projects grow, where the collaboration genuinely stops working, and when not to use it at all.

Part V distills everything into something you can apply immediately.

Each chapter ends with a **Field Check** — questions to answer or things to try against your own projects. They aren't rhetorical. The framework in this book only becomes useful when it's been applied to something you actually own, and the checks are the fastest way to do that.

The appendices contain the templates: a project start checklist, the four documents that constitute shared memory, a briefing template, and a credential exposure procedure. **Those are the parts you'll come back to.**

A Note on What This Book Assumes

It assumes you have real work to do, some technical competence, and enough judgment to know when something looks wrong. It doesn't assume you're a programmer, though several examples involve code.

It also assumes you're using a capable general-purpose AI assistant with the ability to hold context across a long conversation. The specific product matters less than the practice; the framework here has been developed against real tools and doesn't depend on which one.

It does not assume the AI is always right. A substantial portion of this book is about the opposite.

About the Author

Jerome M. Kutche holds the amateur radio callsign N9LYA, first licensed in 1991. He spent over three decades in industrial electrical and controls work — Allen-Bradley and Siemens PLC systems, motor control, plant electrical maintenance and supervision — holding NFPA 70E and thirty-nine other professional certifications. From 1990 to 1996 he built RF test and measurement equipment for the telecommunications and cable television industries.

He serves as Indiana's 44Net/AMPRNet coordinator and is trustee for two amateur radio clubs. His home network runs segmented VLANs behind a MikroTik router, with a virtualized DNS filter, local AI inference, reverse proxying, and rather more automation than is strictly necessary.

The projects described in this book are his own, and the incidents in Chapter 10 all happened.

Part I — Foundations

Chapter 1: Why This Book — The Shift From Asking to Running

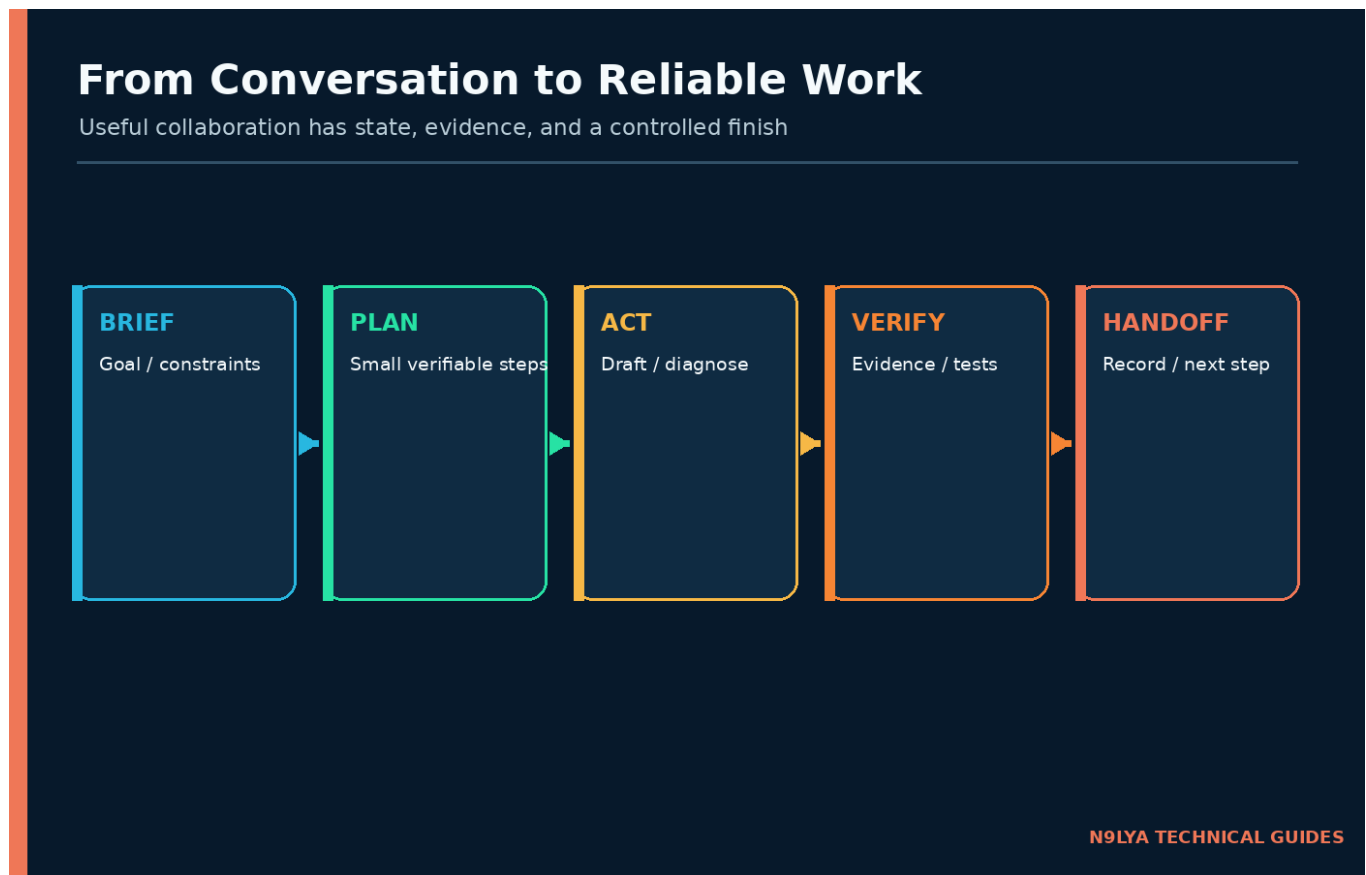


Figure 1: From conversation to reliable work

Every technical collaboration has an origin point, and it's rarely dramatic. It's usually a small task that goes well enough that you hand over a slightly bigger one next time.

For a lot of people, that first task is something like: “explain this error message.” Then it’s “write this script.” Then, months later, it’s “here’s my entire network topology, here’s what I want to build, help me plan the whole thing” — and somewhere in that progression, the relationship changes shape. You stop treating the AI like a reference tool and start treating it like a collaborator with a defined role on the project.

This chapter is about naming that shift explicitly, because most people drift into it without noticing, and drifting means you don’t get the benefits of doing it deliberately.

The Three Stages of AI Use

In practice, most long-term users of an AI assistant move through three recognizable stages:

Stage one: Lookup. You ask factual or syntax questions. “What’s the flag for this command?” “What does this error mean?” The AI is a faster, more conversational search engine. There’s no project context carried between sessions — each question stands alone.

Stage two: Assistance. You bring project context. “Here’s my script, why isn’t it working?” “Here’s my network diagram, what am I missing?” The AI now needs to hold state about your specific system, even if only for the length of one conversation. You’re doing the driving; it’s doing focused repair work.

Stage three: Partnership. You bring an entire initiative — not a bug, not a snippet, but a goal — and the AI takes an active role in planning, building, and tracking it across multiple sessions. This is where a home lab audit stops being “help me check one thing” and becomes a structured, multi-phase project with its own findings, its own documentation, and its own follow-up list.

Most people stall at stage two indefinitely. It’s genuinely useful there, and there’s no obligation to go further. But if you’re running any kind of real infrastructure — a network, a set of services, a physical system with software behind it — stage three is where the real leverage shows up, because it’s the only stage where the AI can carry meaningful project memory and act as a second set of hands across time, not just across one conversation.

Recognizing Which Stage You’re Actually In

A useful diagnostic: how much context do you have to re-establish at the start of a session?

If the answer is “none, because each question is self-contained” — stage one. Fine, and nothing here applies.

If the answer is “a few paragraphs of background every time” — stage two. You’re doing real work and paying a recurring tax to do it.

If the answer is “I point at the documentation and we continue” — stage three. The project has memory that lives outside any single conversation.

That last distinction is the whole thing. A partnership isn’t defined by how sophisticated the tasks are; it’s defined by whether the project’s state persists somewhere both of you can reach. Chapter 11 is entirely about that.

A second diagnostic, sharper: could a different session — one with no memory of any conversation you’ve had — pick up this project and continue meaningfully from what’s written down?

If yes, you have a partnership. If no, you have a series of assisted conversations that feel like a partnership because you remember all of them.

What Changes When You Move to Partnership

Three things change once a collaboration reaches partnership level, and all three require deliberate choices rather than passive drift:

Division of labor becomes explicit. In stage two, division of labor doesn’t need to be discussed — you ask, it answers, you decide what to do with the answer. In stage three, someone has to decide,

out loud, what the AI actually builds versus what stays in human hands: who writes the deployment script, who verifies it against production, who holds the credentials, who pulls the trigger on a change that could take down a service. Chapter 3 covers this in full.

Trust becomes a live variable. At stage two, if the AI gives you a wrong answer, you shrug and try again. At stage three, if the AI writes a firewall rule that blacklists your own SSH connections, or if a credential gets exposed in a shared terminal session, that's a live incident with real consequences — and how you both handle it either builds durable trust or erodes it. Chapter 5 is about that loop.

Documentation becomes shared infrastructure, not an afterthought. When it's just you, your undocumented mental model is enough. Once an AI is acting as a returning collaborator across sessions, the project's documentation is the only thing that lets it pick up where it left off. A findings log, a task tracker, a network map — these stop being “nice to have” and become the shared memory the whole partnership runs on. Chapter 11 covers this in depth.

The Honest Case for Doing This

It's worth being upfront about why this is worth the effort, because none of it is free. Setting up explicit labor division, maintaining documentation, and building trust deliberately all cost time up front.

The payoff is leverage on projects that would otherwise not get done at all, or would take far longer. A full audit of a dozen interconnected services — checking every port, every credential, every automation dependency — is the kind of project that's easy to say you'll do “eventually” and never start, because it's tedious and the payoff is diffuse (you're looking for problems that mostly aren't there). Handing structured pieces of that work to an AI partner, while you retain verification and judgment, is often the difference between “eventually” and “done this weekend.”

The second payoff is less obvious and possibly larger: it changes which projects you consider possible.

There's a category of work that never gets started because the tedium-to-payoff ratio is bad. Auditing every firewall rule. Documenting a network that grew organically over five years. Rewriting a pile of ad-hoc scripts into something maintainable. Producing a technical manual for a system only you understand.

None of these are hard. All of them are long. **A partnership that reliably handles the long-and-not-hard portion changes the calculus on every one of them** — and the projects you end up doing are different from the projects you would have done alone, not just faster.

The Honest Case Against

Three costs worth naming before you commit:

The verification burden is real and it doesn't go away. Every chapter in this book that says “the human verifies” is describing work. If you hand over building and don't do the verifying, you haven't built a partnership — you've built a system that produces plausible output nobody checked. Chapter 4 is about doing this efficiently, and efficient is not free.

Documentation discipline is genuinely hard to sustain. It's the first thing to slip when a project gets busy, and it's the thing that makes the partnership work. Chapter 11 is honest about this.

And there's a skill-atrophy question worth taking seriously. If the AI writes all your scripts, your script-writing gets rusty. Whether that matters depends entirely on whether you need the skill for its own sake. **For a professional, it might. For someone automating their own homelab, it probably doesn't** — the same argument applied to calculators and to compilers, and mostly resolved in favor of the tool.

Be deliberate about which skills you want to keep sharp, and keep doing those yourself even when delegating would be faster. That's a legitimate reason to do work the slow way.

Field Check

1. Which stage is your current AI use actually at — lookup, assistance, or partnership?
2. How much context do you re-establish at the start of a typical session? Estimate the time.
3. Could a session with no memory of your conversations pick up your most important project from what's written down?
4. Name one project you've been meaning to do "eventually" that's long but not hard. That's your candidate.
5. Name one skill you want to keep sharp regardless of what could be delegated.

Chapter 2: Briefing — Getting a Project Into Shared Context



Figure 2: The project brief

Before any division of labor, before any building, there's a step most people do badly or skip: establishing what the AI actually knows about your situation. This chapter is about doing that deliberately.

Why the Briefing Matters More Than the Prompt

An enormous amount of advice about AI use focuses on prompt wording. **For project work, wording is a minor variable and context is the dominant one.**

A precisely worded request against an incomplete picture of your system produces a precisely wrong answer. A roughly worded request against an accurate picture produces something useful you can refine.

The failure mode this prevents is the most common one in AI-assisted technical work: the AI fills a gap with a reasonable-sounding assumption, you don't notice because the assumption is reasonable, and the error surfaces three steps later when something doesn't match reality.

What Belongs in a Project Briefing

The environment, concretely. Not “I have a home server.” What hardware, what hypervisor, what operating systems, what versions. What’s actually installed and running.

The network, with addresses. Segments, address ranges, what lives where, what can reach what. **Vague network descriptions produce advice that doesn’t apply to your network.**

What’s already been tried. This one is consistently underprovided and consistently valuable. “I already restarted it and checked the logs” saves a round trip and prevents the AI from proposing what you’ve already ruled out.

Constraints. Hardware you can’t change, software you’re stuck with, a maintenance window, a budget, a service you can’t take down, a family member who needs the internet working.

What you’re actually trying to accomplish — the goal behind the request. “Help me write a firewall rule” and “help me stop this device reaching the internet” produce different answers, and the second is what you wanted.

What you know is unreliable. If your documentation is stale, say so. If you’re not certain a device is where you think it is, say so. **An AI told “this may be out of date” treats it differently than one told nothing.**

What to Leave Out

Credentials, always. Chapter 6.

Irrelevant detail. A briefing that’s a complete system inventory when the question is about one container wastes context and buries the relevant parts.

Speculation stated as fact. If you think the problem is DNS but don’t know, say you think it. Stating a guess as an observation is how an investigation goes down the wrong path with confidence.

The Standing Context Problem

For an ongoing project, re-briefing every session is the tax that stage two never stops paying.

The solution is to make the briefing a document rather than a message. **A project brief that lives in a file, gets updated as things change, and gets pointed at rather than retyped** is the difference between a partnership and a series of restarts.

What a standing brief contains:

- The environment and network, current
- The project’s goal and current phase
- What’s been decided and why
- What’s known to be unreliable
- Open questions
- **The correction list** — Chapter 11

Update it when things change, not eventually. A brief that describes a system as it was six months ago is worse than no brief, because it produces confident wrong answers instead of questions.

Briefing for a Specific Task

Beyond the standing context, an individual task benefits from a short, structured framing:

The goal, stated as an outcome rather than a method.

The constraints specific to this task.

What done looks like — how will you know it worked? **This is the most-skipped element and the one that most improves the result**, because it forces you to define success before work starts and it gives the AI something to check its own output against.

The blast radius — is this a scratch script or something that touches production? **Say which.** An AI told “this will run against my live router” behaves differently than one told nothing, and it should.

An Example Worth Studying

Poor briefing:

“My packet node isn’t forwarding. Help.”

Adequate briefing:

“My LinBPQ node stopped forwarding to one of its partners about three days ago. Other partners still work. Nothing changed on my end that I know of. Where do I start?”

Good briefing:

“LinBPQ node, running on a Pi, four forwarding partners. One partner stopped accepting our forwarding about three days ago — the others are fine. The link is AXIP over the internet, not RF. I’ve confirmed our internet is up and the config file hasn’t been edited since June. I haven’t looked at the log yet. Partner’s operator hasn’t responded to email. Goal is to determine whether the problem is on my end before I chase him again. This is a live node other people use, so I’d rather not restart it unnecessarily.”

The third version contains everything needed to give a useful answer on the first pass — the scope (one partner, not all), the transport (internet, not RF), what’s been ruled out, what hasn’t been checked, the actual goal (determine whose problem it is), and a constraint (don’t restart casually).

It took ninety seconds to write and it saves three rounds of clarifying questions.

Briefing an Agent Differently

When the AI can execute rather than only advise (Chapter 7), the brief needs additional elements.

State the scope of action explicitly. “Investigate and report, change nothing” is a complete and clear instruction, and it’s respected. **The absence of such a statement is not the same as saying it.**

Name what’s off limits. Specific hosts, specific directories, specific services that must not be touched or restarted.

State the stop condition. “If the first fix doesn’t work, report rather than trying something else” prevents the cascading-fix failure from Chapter 7.

Set limits. Iterations, time, and — where applicable — spend.

And say what to announce. “Tell me before anything that changes state” establishes a convention that holds across the session without needing repetition.

These belong in the standing brief, not restated each time. Appendix C.

When the Briefing Reveals the Problem

A frequent and useful outcome: you can’t write the brief.

You sit down to state the goal and discover you don’t know what done looks like. Or you list the constraints and realize two of them conflict. Or you go to describe the environment and find you’re not certain what’s actually running.

That’s not a failure of the exercise — it’s the exercise working. A request you can’t brief is usually a request that isn’t ready, and the twenty minutes spent discovering that is cheaper than the two hours spent building the wrong thing.

The same applies to clarifying questions. A question you can’t answer is pointing at a gap in your own picture. **Treat it as information rather than an obstacle.**

The Briefing Habit

Write the brief before you write the request. Even mentally, even for a small task — the discipline of stating environment, goal, constraints, and what done looks like catches your own unstated assumptions as often as it informs the AI.

And notice when you can't state the goal clearly. A request you can't brief is frequently a request you haven't thought through, and the clarifying questions you get back are doing you a favor.

Field Check

1. Write a standing project brief for your most active project. Keep it under two pages.
2. Take a request you're about to make and rewrite it with goal, constraints, and "what done looks like."
3. Identify one thing in your documentation you know is stale. Mark it as unreliable.
4. For your next task, state the blast radius explicitly before starting.

Chapter 3: Setting the Division of Labor

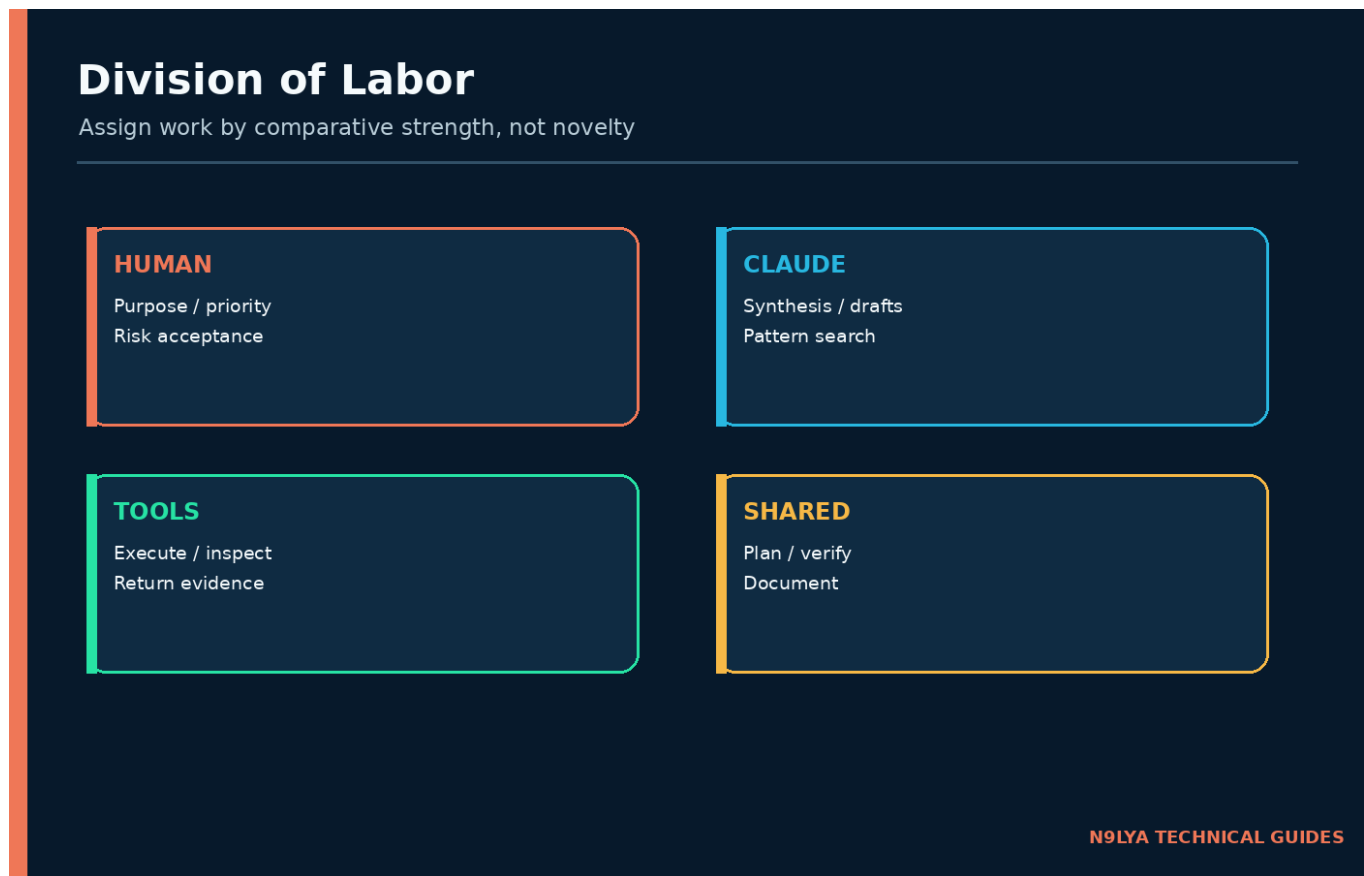


Figure 3: Division of labor

The single most important design decision in any human-AI project partnership is deciding, explicitly, who does what. Get this wrong and you either bottleneck on the human (defeating the purpose of the partnership) or hand over things that shouldn't be handed over (creating real risk). Get it right and the work moves fast without anyone losing control of the parts that matter.

The Core Split: Build vs. Verify vs. Deploy

A useful mental model, drawn from actual practice, splits any technical task into three roles:

Build — designing the solution, writing the script, drafting the configuration, producing the first version of something.

Verify — checking that what was built actually matches reality: does this config apply to the right host, does this script account for the actual (not assumed) network layout, does this credential still work.

Deploy — actually pushing the change into a live system: running the command on the production box, applying the firewall rule, restarting the service.

A workable default for a lot of home-lab-scale work: **the AI builds, the human verifies and deploys.** This isn't a rule carved in stone — there are tasks where the AI can safely deploy too, especially in sandboxed or clearly reversible situations — but it's the right starting default, because it puts a human decision point between "this looks right" and "this is now live."

In practice, this looks like: an AI-driven coding environment does the actual investigative work on a server — checking configurations, running diagnostics, drafting fixes — while a second AI session serves purely as a coordination and planning layer, and the human remains the one who reviews findings and decides what actually gets changed. That three-way split (execution agent, coordination layer, human decision-maker) scales surprisingly well once a home lab has more than a handful of interconnected services, because it means the human isn't the bottleneck for every single diagnostic step, but is still the bottleneck for every actual change.

Chapter 12 covers that structure in detail.

Deciding What Never Gets Handed Over

Some categories of work should stay firmly on the human side regardless of how much trust has been built:

Credential rotation and secrets management. An AI can help you design a rotation plan or write the script that changes a password, but the actual entry of new secrets into a vault, and the decision about which have been exposed and need rotating, is worth keeping as a human-confirmed step every time — especially after any session where secrets were visible on screen. Chapter 6.

Anything with a blast radius beyond one component. A firewall rule change on a router that sits in front of your entire network is a different risk class than restarting one containerized service. The bigger the blast radius, the more the "verify before deploy" gap matters.

Irreversible or hard-to-reverse actions. Deleting data, wiping a filesystem, or applying a change with no clean rollback path deserves a deliberate human "go" — not because the AI is untrustworthy, but because irreversible actions deserve friction regardless of who's doing them.

Anything with consequences outside the system. Money, legal exposure, a relationship with a vendor or a neighbor, an obligation to a served agency. Chapter 18.

Deciding What Should Be Handed Over More Freely

On the other end, some work is safe to hand over almost entirely, because the cost of being wrong is low and reversal is trivial:

Read-only investigation. Checking configurations, comparing running state against documentation, testing connectivity, or reading logs is close to zero-risk. This is exactly the kind of work an AI-driven coding agent can do continuously without a human hovering over every step — the actual audit and diagnostic work described in Chapter 10 largely happened this way.

Drafting, not applying. Scripts, configs, and documentation drafts cost nothing until they're actually run or committed. There's no reason to gate this behind a verification step; verification belongs at the deploy boundary, not the draft boundary.

Anything explicitly staged in a sandbox or test environment. If a system is genuinely isolated from production, the stakes of an AI making a mistake there are low enough that faster iteration is worth more than a verification bottleneck.

Analysis and synthesis. Reading a long log, comparing two configurations, summarizing what changed, organizing findings by severity. **Low risk, high tedium, and exactly the shape of work the partnership is for.**

The Gradient, Not the Binary

In practice the split isn’t three boxes — it’s a gradient, and the useful question is how much friction to put between proposal and effect.

Action	Appropriate friction
Read a log file	None
Draft a script	None
Run a script in a sandbox	None
Restart a non-critical container	Announce, then do
Change a config on a live service	Human reviews the diff first
Change a firewall rule on the edge router	Human reviews and applies
Rotate a credential	Human does entirely
Delete data	Human does, after a stated confirmation

Set this explicitly at the start of a project, and revisit it as trust develops. **A partnership where every action requires the same approval is one where approval becomes a reflex,** which defeats the purpose of having it.

A Worked Example: The Weekend Audit

Consider a full audit of a multi-service home lab — a dozen-plus containers and virtual machines running on a hypervisor, a couple of managed switches, a router doing firewall duty, several automation workflows tying it all together. The goal: find every misconfiguration, every orphaned dependency, every silent failure, and document all of it in one place.

The division of labor that makes this tractable looks like this: an execution-focused AI agent, working directly against the infrastructure, does the actual poking — checking every container’s network binding, testing every credential, tracing every automation’s dependencies, watching for services that quietly stopped talking to each other months ago and nobody noticed. A second, coordination-focused AI session takes the raw findings and organizes them: what’s urgent, what’s cosmetic, what needs a hardware fix versus a config fix, what can be resolved this weekend versus what needs to be tabled.

The human’s job in this picture isn’t to do the checking — that would take the better part of a week done manually, checking a dozen-plus services one at a time. **The human’s job is to review the findings, decide what actually gets fixed and in what order, and to be the one who pulls the trigger on anything that touches a live, in-use system.**

That’s the leverage: hours of tedious verification work compressed into an afternoon of AI-driven checking, followed by an evening of human judgment on what to do about it.

This exact structure — audit agent, coordination layer, human reviewer — is how a comprehensive multi-phase infrastructure audit gets done in a weekend instead of never getting done at all.

When the Default Is Wrong

Two situations where “AI builds, human verifies and deploys” should be adjusted:

When the human can’t meaningfully verify. If you’re asking for something outside your competence, “human verifies” is a formality. **Be honest about this.** The options are to learn enough to verify, to have someone else verify, to test it somewhere it can’t hurt anything, or to accept the risk knowingly. **What doesn’t work is pretending you verified because you read it and it looked fine.**

When verification costs more than the work. For genuinely trivial, genuinely reversible things, a full verification pass is theater. **Reserve the discipline for where it earns its cost**, or it degrades into ritual everywhere.

The Division-of-Labor Checklist

Before starting any AI-assisted project, it's worth explicitly answering four questions:

1. **What's the blast radius if this goes wrong?** (One component? The whole network? Irreversible data loss?)
2. **Is this read-only investigation, or does it change live state?**
3. **If it changes live state, is there a clean rollback?**
4. **Who is the last set of eyes before this goes live — and is that actually happening, or just assumed?**

Answering these explicitly, every time, is the difference between a partnership that scales safely and one that eventually causes an incident nobody meant to cause.

Field Check

1. For your current project, write down the build/verify/deploy split explicitly.
2. List three things that will never be delegated, and why.
3. List three things you're currently verifying that don't need it.
4. Identify one task where "human verifies" is currently a formality. Decide what to do about it.

Chapter 4: Verification — The Human's Core Skill



Figure 4: The verification stack

End of the free sample

Building With Claude — Second Edition · by Jerome M. Kutche

That was the first 12 of 70 pages. The full book continues with the practical detail, worked examples and recovery steps that don't fit in a sample.

Get the full book (\$6.99, PDF + EPUB):

<https://booksrus.biz/product/building-with-claude/>

Also on Gumroad:

<https://n9lya4.gumroad.com/l/xvfbbo>



All 17 Second Edition guides in one bundle for \$89.99:

booksrus.biz/product/n9lya-technical-guides-complete-bundle

This sample may be shared freely in unmodified form. © Jerome M. Kutche. Second Edition.