The AI Bootcamp

33

Make One Job Run Across Several Apps

In this chapter you'll learn

One cross-application chain, the arithmetic that says keep it short, and what it costs to run

The AI Bootcamp

THE PROBLEM

Most AI education is a pile of disconnected tips. A LinkedIn carousel here, a YouTube tutorial there, a $2,000 cohort course that goes out of date before the refund window closes. None of it is built to be read in order, so nothing compounds. You finish twelve videos and can recite twelve facts, not perform one new skill.

THE FIX

The AI Bootcamp is built the other way. 90 skills, packed 2 to 3 per chapter, 36 chapters, 3 tracks. Every chapter states what it needs from you and what it hands to the chapter after it, so skill 47 only works because skill 12 already landed. Read it in order and by the end you are not collecting facts about AI, you are running work through it.

LETTER FROM THE EDITOR

We built this because we kept meeting smart people who felt behind on AI for no reason other than nobody had organized the material for them.

This book does not assume you are technical, and it does not water anything down either. Every chapter earns its place: a real skill, a real exercise, a one-page companion you can flip back to in month three when you have forgotten the details but still need the answer.

If a chapter loses you, or a chapter changes how you work, I want to hear about it either way.

Kris

Editor, AI Central

Write to me at **editor@thecentral.ai**

THE 3 TRACKS

AI 101

Get running

Pick a tool, set it up, write a clear instruction, use it safely at work.

AI 202

Prompt properly

The framework behind every good instruction, and catching AI when it is wrong.

AI 303

Build without watching it

Connect AI to your tools, and automations that run while you are not looking.

IN THIS LESSON YOU'LL LEARN

By the end of this chapter you can design a workflow that starts in one application, does its thinking in the middle and lands in a third, work out before you build it how often the whole thing will finish clean, and make every step fail loudly rather than pass bad data down the line.

Chapter 28 got one automation running and Chapter 29 found you the work worth automating. But the jobs that actually eat your week are rarely one trigger and one action. They start in one system, need judgement in the middle, and have to end up somewhere a colleague will look. Those are genuinely buildable now. They are also the ones that break quietly, and nobody warns you that a chain fails far more often than any single step in it does.

CONTENTS

33.1 A chain is the same sequence, with you taken out of it

33.2 One chain, built with its failures already in it

33.3 A chain fails more often than its steps suggest

33.4 Two steps that work beat eight that mostly do

TOPICS REQUIRED

Chapters 26 through 32. You need at least one connector working, one automation you have already shipped, and the recurring job from your Chapter 29 audit that takes more than a single trigger and action to finish.

AFTER THIS CHAPTER

Chapter 34 assumes you have a chain crossing at least two applications, and that you have already found the one step in it you would not be comfortable leaving unwatched.

33.1 A chain is the same sequence, with you taken out of it

Chapter 16 had you run a long job as a sequence of steps, in one conversation, present at every step. This is that sequence again, crossing several applications, happening while you are somewhere else.

The manual sequence

Chapter 16. You run the steps yourself, in one conversation, and you read every intermediate result.

TRAITS

· You paste the input in by hand
· You see each step before the next one starts
· You are the error handling, without calling it that

PROS

+ A bad step gets caught the second it appears
+ You can change your mind halfway through

CONS

− It only happens when you sit down and make it happen
− The same 20 minutes, every time, forever

The chain

The same steps, wired between applications, triggered by an event rather than by you.

TRAITS

· Starts on its own when something happens in system A
· Passes each result straight into the next step, unread
· Writes into system C whether or not the middle went well

PROS

+ Happens 40 times a month at no marginal cost to you
+ Runs on the days you are travelling or ill

CONS

− Nobody says no to a bad intermediate result
− A wrong answer is now written down somewhere official

Fig 33.1 The same steps in both columns. What changes is who notices when one of them goes wrong, and that single difference is what the rest of this chapter is about.

33.1

The trigger. Something happens in system A without you: a file lands, a form comes in, a date arrives.

The thinking. The one step only an assistant can do: read it, judge it, turn it into the fields the next step needs.

The landing. The result is written into system C, where somebody finds it without being told to look.

Here is the difference, and it is not a technical one. In Chapter 16 you took a job that was too big for one message and ran it as a sequence, yourself, in one conversation. You pasted each result into the next step and you read every one of them on the way past. That works, and for a lot of work it is still the right answer. It also means the job only ever happens on the days you sit down and make it happen.

A chain is that same sequence with you removed from the middle of it. Something happens in one application. Your assistant does the part that needs judgement. The result is written into a third application, where somebody else will find it. You are not in the room for any of it, and on a good week you never think about it at all.

Read the figure again and notice what did not change. The steps are the same steps. What changed is who checks them. In a conversation you are the quality control without ever calling it that: you see a summary that has missed the point and you say no, do that again. In a chain there is nobody to say no. Whatever step two produces, step three accepts, and step four writes down.

There are three places a chain like this can live, and you should pick before you start building. Inside your assistant, using the connectors from Chapter 26, which is the cheapest way to try something and the weakest at running on a schedule and telling you when it broke. On a dedicated automation platform such as Zapier, Make or n8n, which exist precisely to run unattended and keep a record of every run. Or inside the suite your company already pays for, where Microsoft and Google both build automation of their own, and where your IT team may already have an opinion you would rather hear now than later.

One thing genuinely has changed underneath all three, and it is worth knowing the name of. The Model Context Protocol is an open standard for connecting assistants to outside systems, with a spec dated July 2026 and support across Claude, ChatGPT and most of the developer tooling around them. The practical effect for you is that connecting an assistant to a system is increasingly a setting rather than a project. The practical cost is that a connector you install is code you did not write: Anthropic warns in its own documentation that custom connectors reach services nobody has verified, that a malicious one can feed your assistant instructions you never typed, and that in some modes tools get invoked without asking you first. Chapter 27 covered what it means to grant that permission. Grant it to fewer things than you are tempted to.

Before you open any tool, write down three things on paper: what happens in system A, what only an assistant can do with it, and where the result has to land. If you cannot fill in all three in one line each, you do not have a chain yet. You have a wish.

33.2 One chain, built with its failures already in it

A real job, wired end to end. The parts that handle things going wrong are in the design from the start, because a chain that gets them bolted on afterwards never gets them at all.

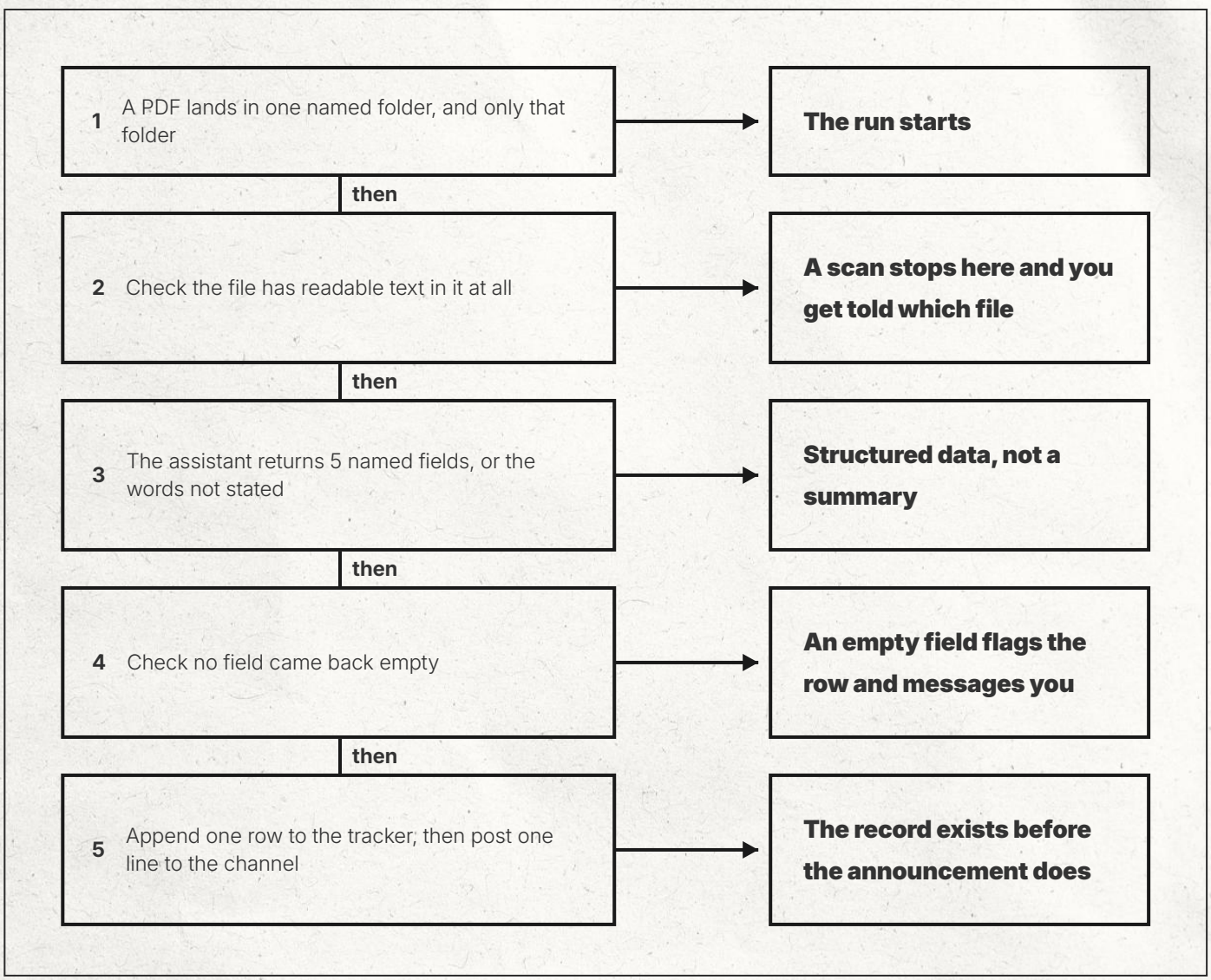


Fig 33.2 Two of these lines exist only to stop a bad run before it writes anything, and they were drawn at the same time as the rest. On most platforms a check like that is the cheapest thing in the chain.

33.2

Narrow the trigger. One folder, one file type. A trigger that fires on anything is the most common reason a chain runs constantly and costs money.

Ask for named fields. Five fields, each named, with "not stated" allowed. Never "summarise this contract" when something downstream has to read the answer.

Record before announcement. Write the row first, post the message second. If the message fails you still have the record. The other order loses the work.

Take a job most companies do badly and nobody owns. Signed agreements arrive as PDFs, from suppliers, from customers, from whoever your legal counsel last emailed. Somebody saves them somewhere. Nobody logs them. Six months later a renewal fires and no one can say when the notice period was, because the answer is inside a 14-page PDF that only exists in one person's downloads folder.

The chain starts with a PDF landing in one named folder in the shared drive, and only that folder. That narrowness is doing real work. A trigger set to fire on any new file anywhere will pick up screenshots, half-finished drafts and the same document uploaded twice, and each of those is a full run you pay for and then have to explain.

The step in the middle is the one only an assistant can do, and how you ask for it decides everything downstream. Do not ask it to summarise the contract. Ask it for 5 named fields: counterparty, start date, end date, annual value, notice period. Tell it in the instruction that anything it cannot find in the document comes back as the words "not stated", and that it must never infer a value from context. That is the Chapter 13 framework applied to a step whose reader is not a person but a spreadsheet, and the format half of it matters more here than it ever did in a conversation.

Then the two checks, which are in the design now rather than added after the first bad week. Before the assistant sees anything, check the file has extractable text at all: a scanned page is an image, the assistant will read almost nothing off it, and the honest outcome is to stop and send you one message naming the file. After the assistant answers, check that no field came back empty. An empty field is different from "not stated": "not stated" is an answer, and empty means something went wrong upstairs. When a field is empty, still write the row, but mark it and tell you, because a flagged row someone will fix beats a silent gap nobody sees.

The last two steps are in that order on purpose. Append the row to the tracker, then post the one-line note into the channel where finance and legal actually look. If the note fails you still have a complete record and a missing announcement, which you can fix in 30 seconds. Reverse them and a failure leaves you with an announcement about a row that does not exist, which is worse than nothing, because now somebody trusts it. The craft of writing well into shared documents and channels is Chapter 31; what belongs here is only the ordering.

Draw yours before you build it. Three real steps, two checks, one notification, on one sheet of paper, and only then open the tool.

33.3 A chain fails more often than its steps suggest

Every step can be individually reliable and the whole thing still be unreliable. This is arithmetic, not bad luck, and it is the single most useful thing to understand before you build anything long.

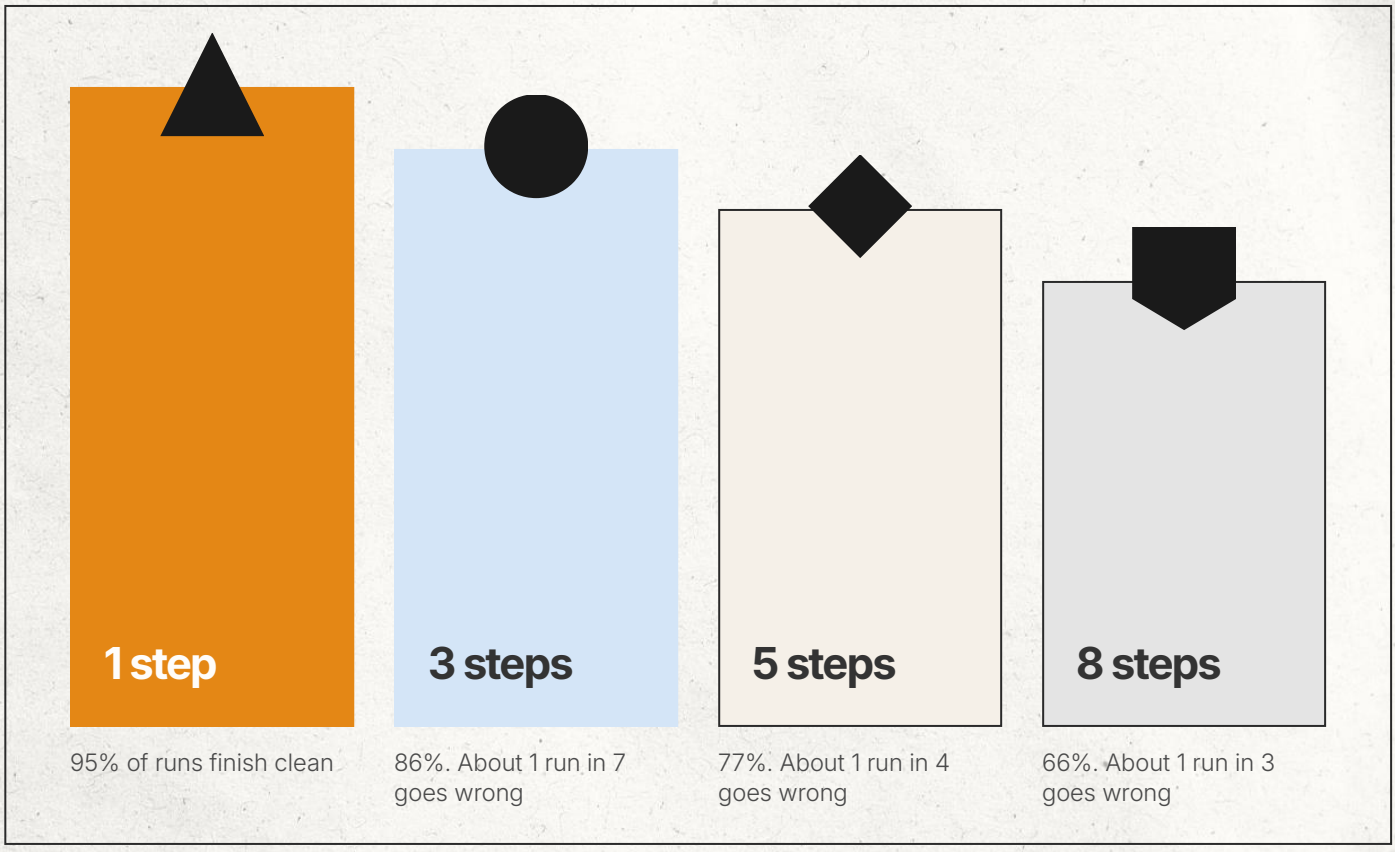


Fig 33.3 Assume each step finishes correctly 95% of the time, which is generous. The bar heights here do encode the numbers: this is how often the whole chain finishes clean, and nothing in it is broken.

33.3

The multiplication. 8 steps at 95% each finish clean about 66% of the time. At 100 runs a month that is 34 broken runs, from 8 good steps.

Loud beats quiet. A step that stops costs you 10 minutes. A step that passes a wrong value costs you the six weeks before anyone notices.

The usual culprit. Almost never the assistant. Almost always an input that arrived in a shape the chain had never seen before.

Chapter 25 introduced compounding error as an idea. Here is what it actually costs you. Say every step in your chain does the right thing 95% of the time, which is generous for anything touching a real inbox, a real PDF or a real person filling in a form. Three steps finish clean 86% of the time. Five steps, 77%. Eight steps, 66%.

Translate that out of percentages, because percentages hide it. A chain that runs 100 times a month with 8 steps at 95% breaks 34 times a month. You did not build a broken chain. You built 8 good steps, and the multiplication did the rest. It is also why raising the quality of each step pays so much better than it feels like it should: at 99% a step, those same 8 steps finish clean 92% of the time. Fewer steps and better steps are the only two levers there are, and on the day you design it, the number of steps is the one you control.

Real chains do worse than that arithmetic, for a reason worth knowing. The multiplication assumes each step fails independently, and they do not. One oddly shaped input breaks step two, which hands something strange to step three, which then fails for a reason that looks unrelated. The same shape shows up when reliability is measured properly on longer work: METR's research on how long a job an AI agent can finish found close to a 100% success rate on tasks taking a person under 4 minutes, and under 10% on tasks taking more than about 4 hours. That measured AI agents doing long tasks rather than plumbing between apps, so take it as the shape of the curve rather than a number about your chain. The shape is the point. Length is what kills reliability.

Now the distinction that actually changes how you build. There are two ways a step goes wrong, and they cost wildly different amounts. A step that fails loudly stops the run, writes nothing, and sends you an email. That is annoying and it is cheap: you lose one run and 10 minutes. A step that fails quietly returns something that looks fine, and the chain accepts it, and step five writes it into the tracker, and the run history shows a clean green tick. You find out in week six, when somebody asks why three contracts show a renewal date in 1926.

So the design goal is not to complete. It is to fail early and visibly. Set every step to stop the run by default rather than carry on. Both of the big platforms let you attach a route that only runs when a step fails: use it to notify you, not to skip past the problem. Zapier will automatically retry an errored run on its paid tiers, which is genuinely useful for a service that was briefly down, and useless for a document that was always going to be unreadable. Add explicit checks whose only job is to turn a quiet wrong answer into a loud stop, which is exactly what the two checks in 33.2 are for.

And know where the quiet failures come from, because it is nearly always the same place: an input arrived in a shape the chain had never seen. A date written 3/4/26 that could be March or April. A value typed as "1.2m" where the next step wanted a number. A name entered as "Smith, Jane" that gets written into the tracker backwards forever. A PDF that turned out to be a photograph of a page. An out-of-office auto-reply that tripped the trigger. An attachment buried three replies deep in a forwarded thread. None of those produce an error. Every one of them gets written down. Chapter 19 warned you about trusting it with arithmetic; this is the other half, and it is about shape rather than sums.

33.4 Two steps that work beat eight that mostly do

The honest advice about chain length is unglamorous and it survives contact with real use. Cut until it hurts, then check what the meter is going to charge you.

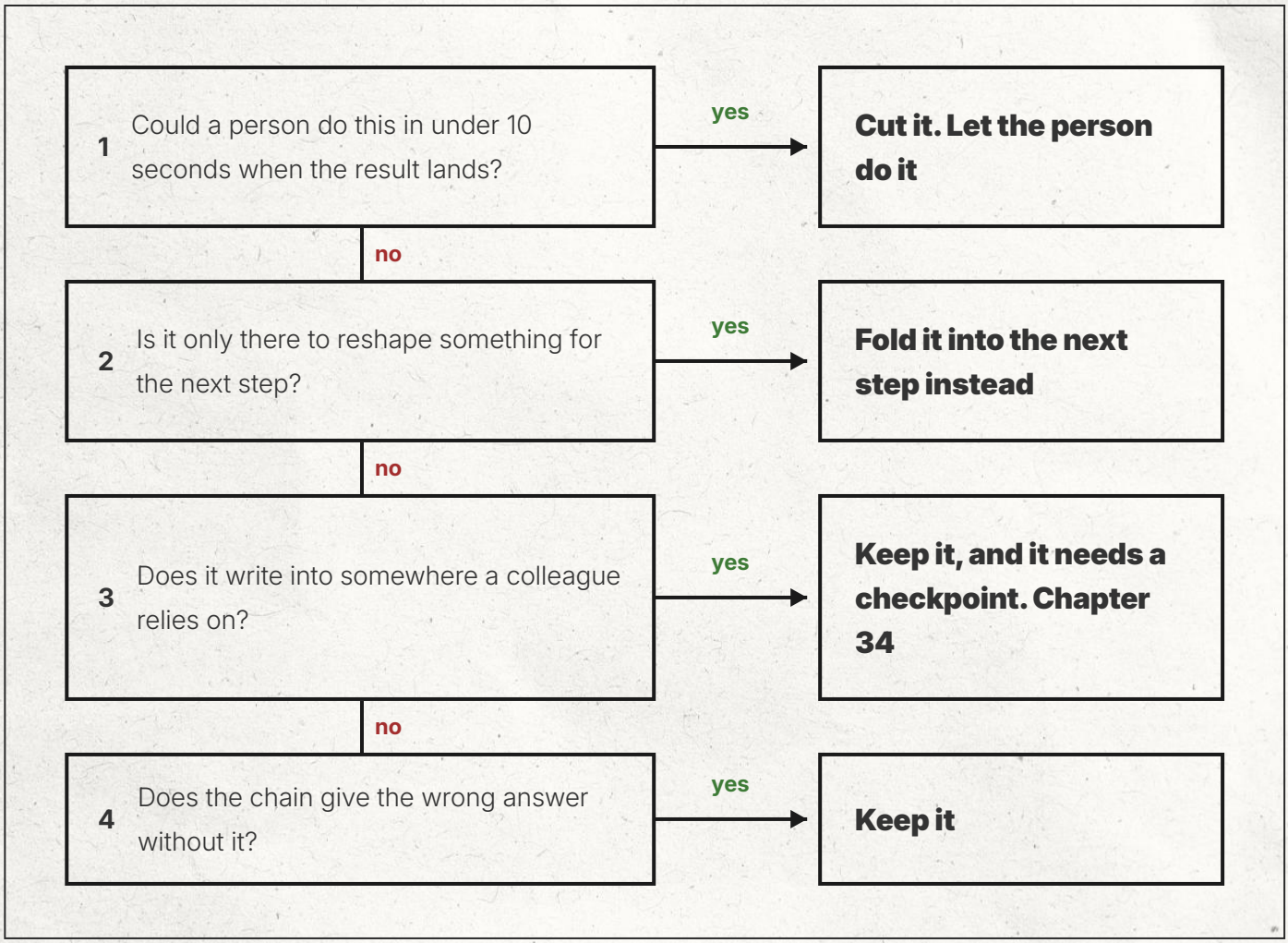


Fig 33.4 Run every step past this before you build it, reading down until one line is true. Most steps people add are the first two, and most chains get better the moment those come out.

33.4

Zapier. Meters every successful action step. Free: 100 a month. Professional from $29.99 for 750. Triggers, filters and paths are not charged.

Make. Meters every module action as a credit. Free: 1,000 a month. Core $12 for 10,000. Routers and error handlers are not charged.

n8n. Meters whole runs, not steps. Starter EUR 20 for 2,500 runs a month, with a free self-hosted edition. All prices checked Sept 2026.

Here is the rule, and it is worth more than anything else in this chapter: 2 or 3 steps that work beat 8 that mostly do. Not because long chains are inelegant, but because of the arithmetic you just read. Every step you add is a step that can fail quietly, and it takes something off the end-to-end reliability of everything you have already built.

The figure above is the cut you should make on paper, before you build. Two kinds of step come out almost every time. The first is work a human would do in 10 seconds anyway once the result is in front of them, which you automated because you could rather than because it was worth it. The second is a step that exists only to reshape data for the next step, which usually belongs inside that next step as an instruction rather than standing alone as another thing that can break.

Then look at what running it will cost, because these platforms meter by volume and the bill arrives later than the enthusiasm. Zapier counts every successful action step as a task, with triggers, filters and paths free. Make counts every module action as a credit, with routers and error handlers free. n8n counts whole runs and lets the steps inside be as many as you like. Same chain, three completely different shapes of bill.

Work an example through, because the gap is larger than people expect. Take a 5-action chain firing 20 times a business day: roughly 400 runs and 2,000 successful actions a month. On Zapier that is 2,000 tasks, which is nearly three times the 750 included in the $29.99 tier. On Make it is 2,000 credits, comfortably inside the 10,000 that the $12 plan includes. On n8n it is 400 executions against a 2,500 allowance, because the steps inside a run cost nothing there. Nothing about the chain changed between those three sentences. If your company already pays for Microsoft's automation it is priced per person instead, at $15 a user a month on an annual plan, which is a different trade entirely: better if half your department will use it, worse for one chain.

What actually produces a surprising bill is almost never a chain that is too long. It is a trigger that is too wide. A chain wired to every new file in a shared drive rather than one folder will fire on every screenshot anybody saves, and each of those is a full run charged in full. Narrow the trigger first, then count the steps. And note the useful asymmetry in the meters: the checks that make a chain safe are usually the part you are not charged for, so there is no cost argument for leaving them out.

One last thing before you build. Look down your cut list at the step that writes into something other people rely on, and ask whether you are genuinely willing for it to happen while you are asleep. If the answer is not an easy yes, that step needs a person in the loop, and that is what Chapter 34 is for. Mark it now, while you can still see the whole chain on one page.

READING ENDS HERE

Everything before was reading. Everything after is doing.

3 exercises, About 15 minutes and one real decision that opens Chapter 34.

Do these before Chapter 34

3 exercises. About 15 minutes

Take the recurring job from your Chapter 29 audit and write it as a chain. One line each, before you open any tool.

WHAT HAPPENS IN SYSTEM A	WHAT ONLY AN ASSISTANT CAN DO WITH IT	WHERE THE RESULT HAS TO LAND

Now design the failures in, rather than waiting for them. Answer all three before you build anything.

THE WRONG-SHAPED INPUT YOU CAN ALREADY PICTURE	THE STEP THAT MUST STOP THE WHOLE RUN	WHO GETS TOLD, AND HOW THEY GET TOLD

Count the steps in what you just drew. If there are more than 4, run each one past the cut rule in 33.4 and write out the shorter version. Then write the one step you would not want running unwatched, and carry it into Chapter 34.

Chapter 33 on one page

THE ANATOMY	A trigger in system A, one thinking step in the middle, a durable write in system C. Anything else is plumbing you can probably cut
THE COMPOUNDING TABLE	At 95% a step: 3 steps finish clean 86% of the time, 5 steps 77%, 8 steps 66%. At 100 runs a month that is 14, 23 and 34 failures
FAIL LOUD, NOT LATE	Default every step to stop and notify. Never set it to ignore the error and continue. A green run that wrote a wrong value is worse than a red one
THE SHAPE CHECK	Dates, currency, names reversed, empty fields, scans with no text, auto-replies that trip the trigger. Check the shape before the assistant sees it, and again before anything is written
WHAT IT COSTS	Zapier meters successful actions, Make meters module actions, n8n meters whole runs. The same chain can differ several times over. Narrow the trigger before you cut steps

BRING TO CHAPTER 34

Your chain drawn on one page with its steps counted, and the one step in it you would not want running unwatched. Chapter 34 is about exactly that step.

BOTTOM LINE

You can build a workflow that crosses several applications on its own, and say before it ever runs where it will break and how you will hear about it.

NEXT CHAPTER

The One Check That Keeps Automations Honest

Chapter 34 takes the step you just flagged and puts a person back beside it, without putting you back in the middle of the whole chain: where a checkpoint earns its cost, what a good approval actually asks you, and the short list of things that should never run unattended at all.

BRING WITH YOU

- ☐ The chain you designed in this chapter, with its steps counted
- ☐ The one step in it you would not be comfortable leaving unwatched

GO FURTHER

The Ultimate AI Library

Every guide, prompt pack and playbook AI Central has published, in one place. The reference shelf this course keeps pointing you to.

explore the library ↗



WHO MAKES THIS

AI CENTRAL

AI Central Media is a London-based editorial team. Our flagship publication covers practical AI for senior professionals, and 100+ AI companies, SaaS platforms and growth teams have advertised with us since 2023.

100+

AI companies, SaaS platforms, education brands and growth teams have advertised with us since 2023

613K

Accounts reached every month, across the 7 channels we operate

151

Countries our readers are in, plus all 50 US states

PROUD PARTNER OF

GAMMA · IIElevenLabs · guidde. · Outskill · Luma AI

HubSpot · attio · Synthflow · Notion · Replit

We turn attention into pipeline.

visit thecentral.ai ↗