The AI Bootcamp

32

Build Your Own AI Assistant

In this chapter you'll learn

The test for whether packaging is worth it, one assistant built end to end, and what a colleague can see inside it

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 A LinkedIn carousel here, a YouTube tutorial there, a $2,000 cohort course that goes out of date before the refund window closes.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 tell a job that needs a saved template from a job that needs a packaged assistant, build the second kind with its own name, instructions and reference material, test it against the uses you did not have in mind, and hand it to a colleague knowing exactly what they will be able to see inside it.

Everything you have built in this track so far runs on you. The instruction lives in your head, or in a file on your machine, so the moment a colleague needs the same job done they either interrupt you or do it worse. This chapter is about the object that fixes that, and it is equally about the more common case, where building one would waste an afternoon and a saved template was already the right answer.

CONTENTS

32.1 You have met two of these already, and this is the third

32.2 Most repeated jobs do not need one of these

32.3 Four parts, and the third one is where people go wrong

32.4 It fails differently once it is not you holding it

TOPICS REQUIRED

Chapter 31, and Chapters 11 and 17 in particular. You need one job you have already run several times and got right, the final version of the instructions you used, and the 2 or 3 files that job always needs. Bring the version that worked, not the first one you typed.

AFTER THIS CHAPTER

Chapter 33 assumes you have one packaged assistant that works and that at least one other person has used. It treats that package as a single step inside a longer run, rather than as the whole thing.

32.1 You have met two of these already, and this is the third

Chapter 11 configured your assistant around you. Chapter 17 gave you prompts that sit still until you fetch them. Both of those stop at the edge of your desk, and this one does not.

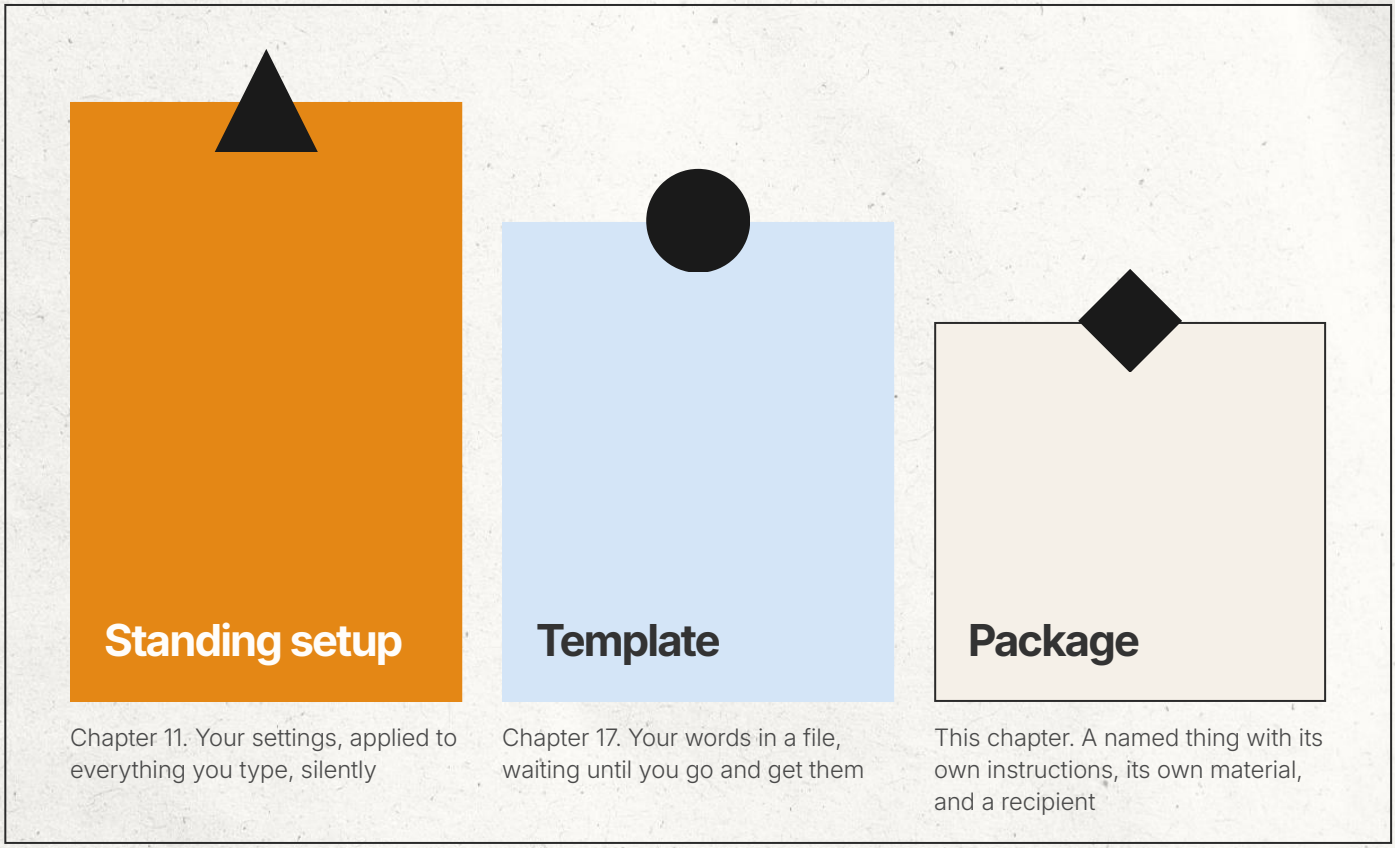


Fig 32.1 The 3 answers to "I keep doing this again". They differ on one question: who else is standing there. The heights are visual rhythm, not a ranking. The third is not an upgrade on the first two, it is what you build when the answer to that question is somebody other than you.

32.1

Level one. Standing instructions and memory. Your configuration, running on everything you type whether or not you remember it exists. Chapter 11.

Level two. A template. Your words, dormant in a document, as narrow and opinionated as you like because nothing fires until you fetch it. Chapter 17.

Level three. A package. A name, a one-line description, instructions and reference material in a single object, with a share control on the side of it.

Here is the situation this chapter exists for. There is a job you have done often enough and well enough that you could describe it in your sleep. Then a colleague asks how you do it, and the honest answer turns out to be a paragraph of instructions, plus 3 files, plus 2 corrections you would only remember while watching them get it wrong. So you do it for them, again, which is exactly the arrangement you were trying to escape.

Two things you already own solve neighbouring problems, and they get muddled with this one constantly. A standing instruction, from Chapter 11, is configuration: it applies to every conversation you have, automatically, and it is about who you are. A template, from Chapter 17, is text you deliberately go and fetch, and it is about a task you repeat. Both are yours. Neither of them travels.

A package is a different kind of object. It has a name and a one-line description, so somebody can tell what it is for without asking you. It carries its own instructions, which apply inside it and nowhere else. It carries its own reference material, attached once rather than pasted every time. And it has a share control, which is the part that makes the whole exercise worth an afternoon.

Notice what is not on that list. Nothing about packaging makes the answers better. Run the same input through a good template and a good package and you get the same output, because the instructions doing the work are the same instructions. What changes is who else can get that output without going through you, and how consistently they get it. That is the entire value, and being clear-eyed about it is what stops you building 6 of these and using none.

The names for this move constantly, so learn the shape rather than the label. ChatGPT calls it a plugin: GPTs, the old name, retire on 11 December 2026. Claude calls the everyday version a project and the more portable version a skill. Gemini calls it a Gem. Microsoft calls it an agent and gives you Agent Builder to make one. All 4 have been renamed or restructured in the past 2 years, and one of them moved while this book was being written, so keep your instructions in a plain text file you own. What has not changed in any of them is the anatomy: a name, a description, instructions, attached material, and a boundary saying who can open it.

Before the next section, write down the one job somebody has actually asked you how to do. Not a job you think would be useful to automate. One where a real person asked.

32.2 Most repeated jobs do not need one of these

This is the most over-recommended feature in the whole subject. Nearly everything written about it assumes you want one, and for nearly everything you repeat, a saved template already wins.

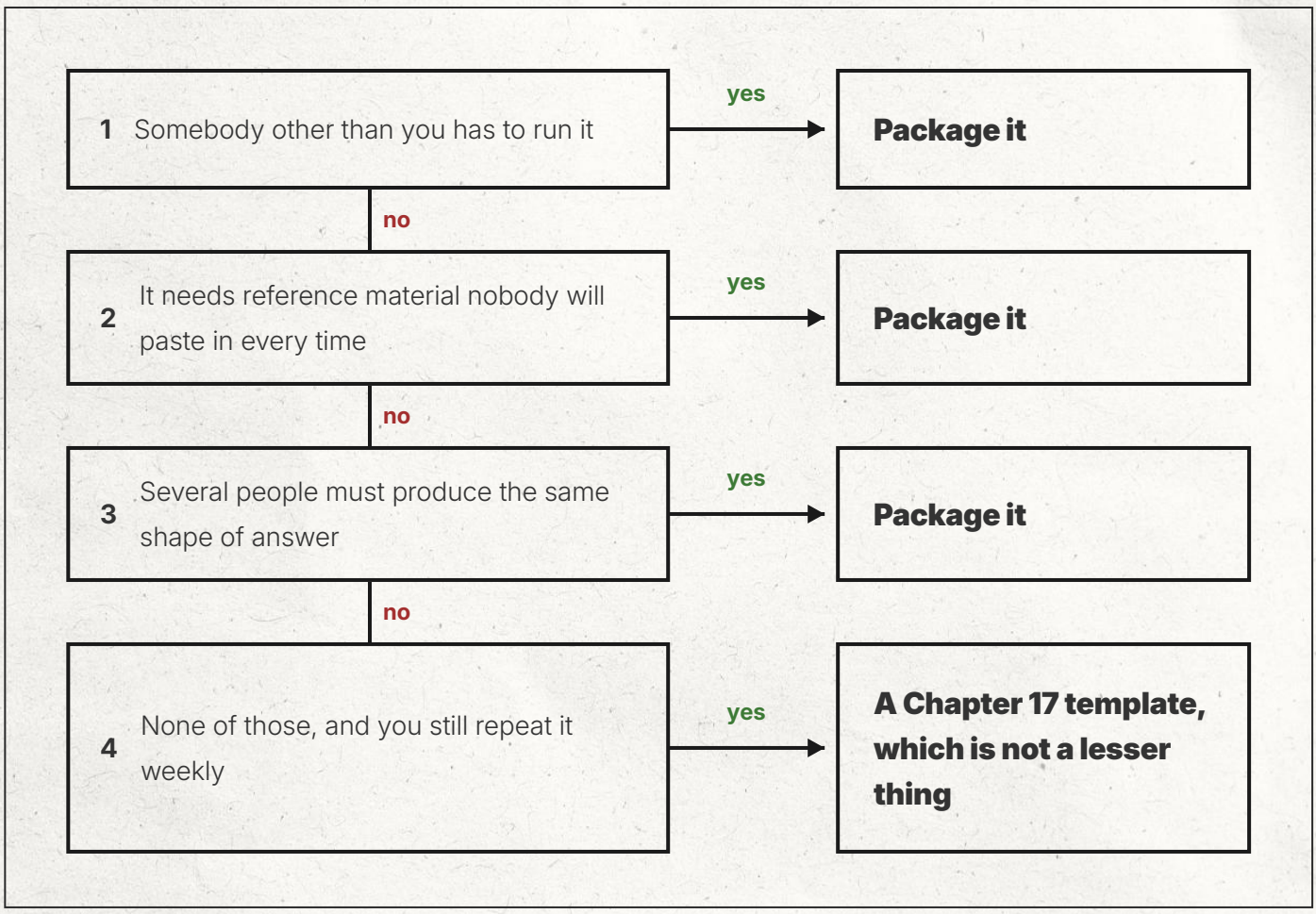


Fig 32.2 Read down until a line is true. Notice that the first 3 lines are about other people or about material, and none of them is about how often you do the job. Frequency is what makes a template worth writing. It is not what makes a package worth building.

32.2

The honest count. In a normal senior role, 1 or 2 jobs clear this bar. If your list has 9 on it, you have not run the test, you have listed your annoyances.

The audience-of-one trap. Built for yourself, opened twice, abandoned. It lost to the template on speed, which is a fair fight it was always going to lose.

It becomes yours to maintain. When the underlying rule changes, a package sitting in 6 people's sidebars is now wrong in 6 places, and none of them will notice.

The case for packaging is almost never the one people give. The reason offered is usually "I do this all the time", and that reason is wrong on its own, because doing something all the time is the argument for writing it down once, which is Chapter 17 and takes about a minute. Frequency justifies a template. It does not justify an object with a name, a description, a maintenance obligation and an audience.

Three things justify that, and the figure lists them. The first is that somebody else has to run the job: a colleague, a team, a new starter who will be asked this in week 2. The second is that the job depends on reference material of its own, a standard, a rulebook, 2 worked examples, something nobody is realistically going to attach by hand every time. The third is that consistency between people matters more than any single answer being the best possible one, which is the case whenever the output goes to the same audience from several hands.

That third one is worth sitting with, because it is the one senior people underrate. If 6 directors each write to the board in their own shape, the board spends its attention on the variation rather than the content. A package does not make any one of those 6 papers better than the best of them. It makes the worst of them recognisably the same object as the best, which is usually what you actually wanted.

The cost side is real and rarely stated. The moment other people use it, the thing has your name on it. When the standard changes in March, your package is quietly out of date in every sidebar it reached, and nobody who is using it will notice, because it will keep answering confidently in the old shape. A template that goes stale wastes your own minute. A package that goes stale spreads a wrong answer with your authority attached, which is a genuinely worse failure and the reason to build very few of them.

So be blunt with your own list. Most senior professionals reading this have 1 or 2 genuine candidates, not 10. The weekly report you assemble alone, in your own head, with no attached material, is not one of them however tedious it is. The thing 4 people ask you to check before they send it almost certainly is.

Run the test down your list now, before you build anything. If nothing clears a line, go back to Chapter 17 and write a better template, and treat that as the chapter working rather than the chapter failing.

32.3 Four parts, and the third one is where people go wrong

Take one candidate that cleared the test and build it end to end. It takes about 20 minutes, and most of that is writing the instructions you have already been giving people verbally.

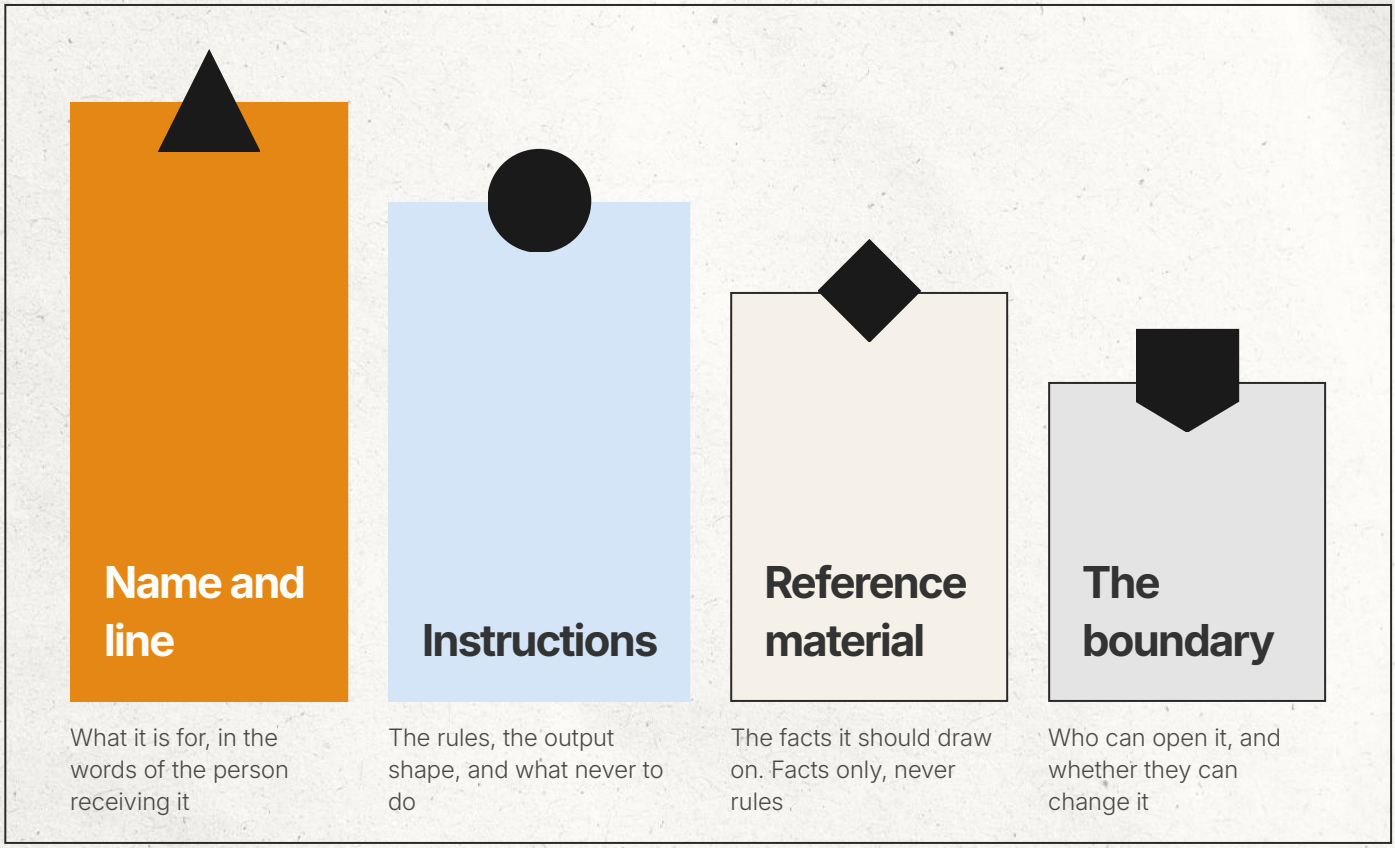


Fig 32.3 The 4 parts of any package, in the order worth writing them. Bar heights are visual rhythm only. The third part is where most first attempts fail, by putting rules into the attached files where the assistant treats them as facts about the world rather than as orders.

32.3

ChatGPT Plugins. GPTs retire on 11 December 2026 and new ones stopped being made in October; an existing GPT migrates to a plugin, carrying its instructions and files but not its custom actions. Building needs Go, Plus or Pro. Free can use a plugin but not make one.

Claude Projects carry their own instructions and knowledge base on every plan, free included and capped at 5 there. Skills are the more portable version, on every plan, and need code execution switched on in settings.

Gemini Gems. Free to build with a personal Google account: a name, instructions, and knowledge files from your device, from Drive, or from a NotebookLM notebook.

Copilot Agent Builder inside Microsoft 365 Copilot, on the web and in Teams but not on mobile. Building is covered by your Copilot licence, and your administrator can switch the whole feature off.

First, check you can build one at all, because this is the part that has moved most. On Gemini and Claude you can, on any plan including the free ones. On Copilot you can if your organisation has left Agent Builder switched on. On ChatGPT the answer moved twice in one year: GPTs were taken off personal accounts in August 2026, then retired altogether, and what you build now is a plugin, which any paid personal plan can do. If your plan leaves you without a builder, your nearest equivalent is a project holding the instructions and the files, which does the same job for you and simply does not travel.

Now the worked example, which is a real shape of job rather than an illustration. You are chief operating officer of a manufacturer with around 400 people. Six directors write papers for the board, 2 or 3 times a year each, and every one of those papers comes past you first. You have said the same 5 things about every one of them for 2 years. That clears the test on all 3 lines: other people run it, it depends on a standard and some worked examples, and consistency between the 6 of them is the whole point.

Part one, the name and the line under it, written for the person receiving it rather than for you. "Board paper check, before it comes to me" tells a director exactly when to open it. "Document reviewer v2" tells them nothing and will not be opened. The description line does the rest of the work: "Checks a draft board paper against our house standard and tells you what to fix. Paste the whole draft."

Part two, the instructions. This is the package. Written out in full, so you can see how ordinary it is: "You check draft board papers for a mid-sized manufacturer before they go to the board. The people using you are directors and heads of function who write 2 or 3 papers a year, not professional writers. Your reference files are our one-page board paper standard and 2 papers that were accepted without rework. When someone gives you a draft, say in one line whether it is ready to send. Then list what is missing against the standard, in the standard's own order, naming the section each time. Then list up to 5 sentences that would not survive a question from the board, and say what the question would be. Then stop. Keep the whole reply under 400 words. Quote the draft's own words when you name a problem, so the writer can find it. Do not rewrite the paper for them. Do not comment on formatting. If what you have been given is not a board paper, say so and ask what they wanted, instead of reviewing it anyway."

Part three is reference material, and it is where first attempts fail. Attach the facts: the one-page standard, the 2 accepted papers, the list of questions your board reliably asks. Do not attach the rules. OpenAI puts this well in its own guidance to builders, and the same holds on all 4: use knowledge for reference material, and put rules, tone and workflow into the instructions. A rule buried in an uploaded file gets treated as information about the world rather than as an order, so it gets mentioned when relevant and ignored when not, which is the single most common reason a package works in your hands and drifts in everybody else's. Attach 3 files, not 30. A folder of everything makes the answers vaguer, not richer.

Part four is the boundary, and it decides more than it looks like it does. On Gemini you share a Gem by email as Viewer or Editor, or by link. On Claude, project sharing is a Team or Enterprise feature: by email, in bulk, or organization-wide, as view or edit. On ChatGPT, a plugin is yours alone unless you are on Pro, the one personal plan that can put a plugin in front of a workspace, and on Business, Enterprise and Edu an administrator decides which plugins exist at all. On Copilot, an agent goes to people in your organisation and no further. Give people view, not edit, unless you actively want them changing the instructions. Then test it in the preview pane before you share anything, and go and do that now, because the next section is about the tests you would not think to run.

32.4 It fails differently once it is not you holding it

Two things change the moment a colleague opens it. They use it in a way you never pictured, and everything you put inside becomes readable by them.

Copied in

Files uploaded into the package itself: a ChatGPT plugin's reference files, Gemini Gem knowledge, the files in a Claude project.

TRAITS

· The file travels with the package, everywhere it goes

· Frozen at the version you uploaded

· Readable, in practice, by everyone you share with

PROS

+ Works for any recipient, with no permissions to arrange first

+ The material cannot change under you between uses

CONS

– Anything confidential inside is now shared as widely as the package

– A file that goes out of date does so silently

Pointed at

The package reads a source that stays where it lives, under the permissions it already carries. A Copilot agent grounded on SharePoint works this way.

TRAITS

· Always the current version, with nothing to re-upload

· Each reader sees only what they could already open

· Sharing the package grants nobody access to the files

PROS

+ Confidential material stays governed by the system that governs it

+ No stale copies anywhere to remember to replace

CONS

– A colleague without access gets a thinner answer and no explanation why

– Somebody has to arrange the permissions, and Microsoft documents that this is not automatic

Fig 32.4 The 2 ways reference material gets attached, and the one question worth asking before you press share: does this file leave the building with the package, or does it stay behind it?

32.4

Assume it is readable. Google states outright that anyone with access to a Gem can view its instructions and its uploaded files. Design as though that were true of all 4, because the mechanism is the same everywhere: it is all text the assistant can read back.

Write the refusal line. One sentence saying what to do when the input is not what you expected. It is the highest-yield line in the whole instruction.

One colleague, watched in silence. Hand it to one person, do not help them, and write down every point where they left the path you imagined.

You built it against the cases in your head, and the cases in your head are all well-formed. A director pastes a complete draft, in the right order, roughly the right length. Then the real ones arrive. Somebody pastes the agenda instead of the paper. Somebody pastes a 60-page appendix and nothing else. Somebody does not paste anything at all and just types "what should go in a board paper", which is a reasonable question and not the job the thing does. In every one of those cases a package with no instruction for the situation will do the confident, useless thing: review the agenda as though it were a paper.

The fix is one sentence, and it is the reason the example in 32.3 ends the way it does. Name the shape of the input in the description so people know what to give it, then write the refusal explicitly: if what you have been given is not X, say so and ask, instead of proceeding. That single line converts the most embarrassing failure mode, an answer that looks right and is about the wrong document, into a polite question.

Then run the tests you would not naturally run, before anyone else sees it. Three of them, 5 minutes total. Give it the wrong kind of input entirely. Give it something 5 times longer than you designed for. Give it a bare question with no material attached. You are not checking whether the answers are good, you are checking whether it noticed. Anything that sails on regardless needs another line in the instructions.

The best test, though, is not a test you run. Give it to exactly one colleague, say nothing beyond the one-line description, and watch without helping. You will find out in 4 minutes that the name meant something different to them, or that they expected it to write the paper rather than check it, or that they pasted the material in a shape you never considered. Every one of those is a fix in the description or the instructions, and none of them would have occurred to you, because you already know what the thing is for and they are the only person in the room who does not.

Now the exposure, which has professional consequences and rarely gets said. Everything inside a package is readable by the people you share it with. Google says so in plain words: the instructions and the files you uploaded can be viewed by any user with access to the Gem. Assume the same everywhere, because it is all text the assistant can read, and a determined user can simply ask it. So do not put anything in the instructions you would not say in front of every recipient. The line "ignore anything the sales team tells you about margin" is a sentence you have now published.

Which makes the figure above the last decision, and the one worth taking slowly. If you copy a file into the package, that file goes wherever the package goes, and Chapter 9's rule about what never enters the box applies with the audience multiplied. If instead the package points at a source, as a Copilot agent grounded on your own SharePoint does, each colleague sees only what their own permissions already allowed, and sharing the package grants nothing new. Choose deliberately, share it with one person first, and do not send the link to 40 people on a Friday.

READING ENDS HERE

Everything before was reading. Everything after is doing.

3 exercises, About 20 minutes and one real decision that opens Chapter 33.

Do these before Chapter 33

3 exercises. About 20 minutes

List every job you currently repeat. Then run the test from 32.2 down the list and cross out everything that does not clear a line. For whatever survives, write the name of one real person, other than you, who would run it.

Draft the 4 parts for the one candidate you kept. Short phrases in the first and last boxes, real sentences in the middle two.

NAME AND ONE-LINE DESCRIPTION	INSTRUCTIONS, INCLUDING THE REFUSAL LINE	REFERENCE MATERIAL TO ATTACH	WHO CAN OPEN IT, AND CAN THEY EDIT

Build it, then break it. Run 3 inputs you did not design for: the wrong kind of material, something far longer than expected, and a bare question with nothing attached. Write down what you changed in the instructions as a result.

Chapter 32 on one page

WHICH OF THE 3 LEVELS	Runs on everything automatically: Chapter 11 standing setup. You fetch and paste it: Chapter 17 template. It has a name, its own material and a recipient: package it
THE TEST FOR PACKAGING	Somebody else runs it · it needs reference material nobody will paste each time · several people must produce the same shape of answer. None true and you still repeat it: that is a template, and a template is not a lesser thing
THE 4 PARTS	Name and one-line description in the recipient's words · instructions carrying the rules, the output shape and the nevers · reference material carrying facts only · the sharing boundary, view rather than edit
WHAT IT IS CALLED, AND WHO CAN BUILD	ChatGPT: plugins, built on Go, Plus or Pro and shared to a workspace on Pro, replacing GPTs from 11 Dec 2026 · Claude: projects on every plan, sharing on Team and Enterprise, plus skills · Gemini: Gems, free to build and to share · Copilot: Agent Builder, inside your organisation only
BEFORE YOU PRESS SHARE	Write the refusal line · run 3 inputs you did not design for · watch one colleague use it in silence · assume every recipient can read the instructions and the attached files

BRING TO CHAPTER 33

Your one packaged assistant, and the note you made when a colleague used it in a way you had not pictured. Chapter 33 treats that package as one step in a longer run rather than as the finished thing.

BOTTOM LINE

You can tell when a repeated job has outgrown a template, package it into a named assistant with its own instructions and reference material, and hand it to a colleague knowing how it will fail and what they will be able to see inside it.

NEXT CHAPTER

Make One Job Run Across Several Apps

Chapter 33 puts several steps in a row across different applications: one run that reads from one place, decides something, and writes to another, with the package you just built sitting inside it as a single step.

BRING WITH YOU

- [ ] The packaged assistant you built, and the instructions inside it
- [ ] The note from the colleague who used it in a way you had not pictured

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 ↗