CLAUDE SKILLS · GUIDE 3 OF 3

Claude Skills Advanced patterns

Eleven ways to make the skills you already have fire at the right moment and give you the same standard of work each time.

Checked against Anthropic's docs on 10 November 2026.

Alex Kerss

START HERE

Before you start

This guide assumes you already make skills and know where to find them. Your skills are under **Customize > Skills** (Anthropic is renaming Customize to Plugins, so your app may show that instead).

Skills work on Free, Pro, Max, Ultra, Team and Enterprise, as long as **Create and edit files** is on under **Settings > Capabilities**. The developer docs list custom skill uploads on Pro and above only, so if you're on Free and can't upload, that's why. Sharing a skill with colleagues needs Team or Enterprise.

Every pattern below comes from Anthropic's own guidance on writing skills. Only the patterns that say so need any code.

BUILD IT

1 Baseline first
2 Trigger-ready description
3 Short front page
4 Script the fixed steps
5 Templates and examples

MAKE IT RELIABLE

6 Checklists and check loops
7 Small and focused
8 Skills with connectors and projects

KEEP IT WORKING

9 Test before you trust
10 Change safely
11 Know when not to

PATTERN 1

Baseline first

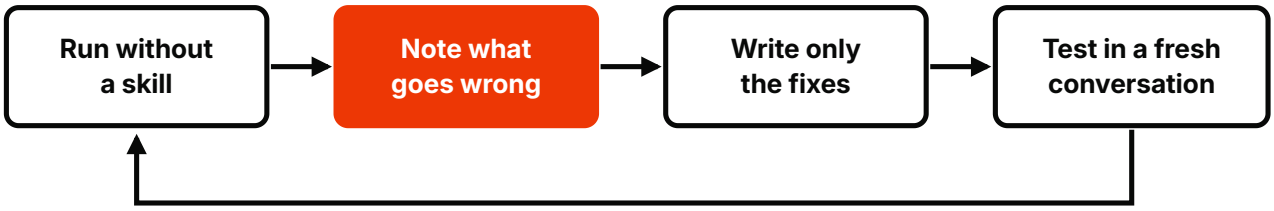
The best skills are made from Claude's mistakes, not from everything you know about the work. Anthropic's advice is to run Claude on a few typical examples with no skill at all, write down exactly where it goes wrong, and then write only enough to fix those gaps.

Claude already knows a great deal. A skill that explains things Claude knows costs space in the conversation and buries the instructions that do the work. So the question for every line is whether Claude would get this wrong without it.

When to reach for it: before you write a new skill, and whenever a skill has grown long and you're not sure which parts still earn their place.

Example: say you want a skill for replying to customer complaints. Before writing anything, give Claude three complaints from last month and ask for replies. Maybe the tone is fine, but it offers refunds you don't allow and never mentions your 14-day returns window. Your skill now needs two short rules, not a page about tone.

Anthropic suggests a two-conversation way of working here. In one conversation, work through the replies with Claude until they're right, then ask Claude to turn what you told it into a skill. Then open a fresh conversation, give it new complaints, and watch what the skill gets wrong. Take what you saw back to the first conversation and fix the skill there.



PATTERN 2

Trigger-ready description

Claude decides whether to load a skill from its description alone. It reads the name and description of every skill you have switched on, then matches your message against them. The rest of the skill stays unread until that match happens.

That makes the description the part of the skill most likely to fail without you noticing. A description that's too broad fires when you don't want it. One that's too narrow never fires at all.

A description that triggers well says two things: what the skill does, and when Claude should use it. Put in the words you'd actually type, including file types and the names you give things at work. Anthropic also says to write it in the third person ("Drafts replies to customer complaints..."), because the description goes into Claude's instructions and "I" or "you" can confuse the match.

When to reach for it: every skill, and first of all any skill that doesn't load when you expect it to.

Example:

- Weak: `Helps with emails.`
- Strong: `Drafts replies to customer complaints using our refund and returns policy. Use when the user pastes a complaint, mentions an unhappy customer, or asks for a reply to a refund or returns email.`

On length, the docs disagree. The Help Center says 200 characters, while the developer docs and the open skills standard allow up to 1,024, and the app accepts up to 1,024. Aim for a sentence or two either way, since every description is sent with every message.

To check whether a skill loaded, look at Claude's thinking for the skill being read. If it isn't there, change the description before you touch the instructions.

••• SCREENSHOT 1

a Claude reply with its thinking expanded, showing the skill being loaded before the answer.

PATTERN 3

A short front page that routes

A skill folder can hold far more than SKILL.md, and the extra files cost nothing until Claude opens one. So the pattern is to treat SKILL.md as the front page. It holds the steps that apply every time, and it points to separate reference files for everything else.

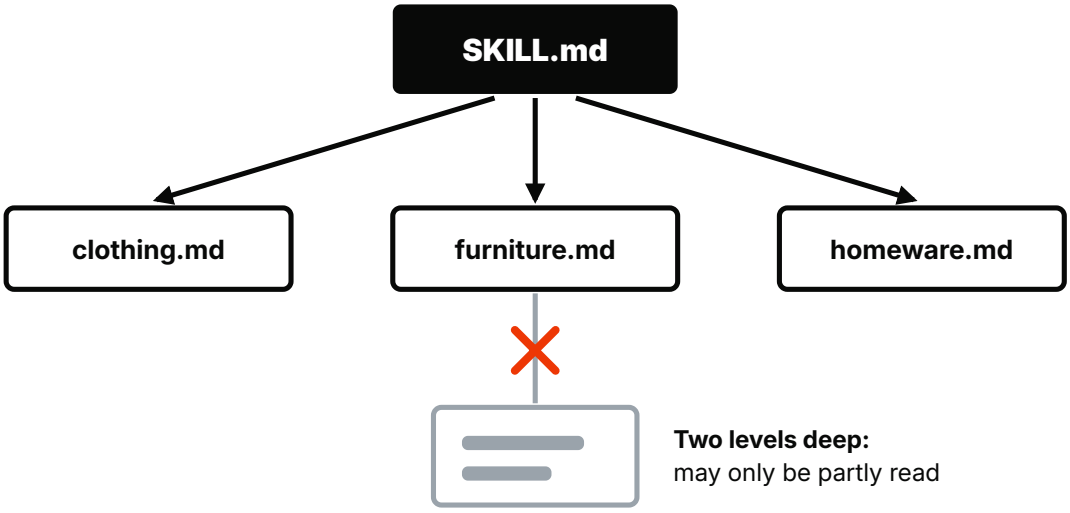
Anthropic's guidance is to keep the body of SKILL.md under 500 lines, or roughly 5,000 tokens. In Claude Code there's an extra reason to stay short and put the key rules near the top: after a long session is compacted, only the first 5,000 tokens of each skill are kept.

Three rules keep the routing working:

- **Link every reference file straight from SKILL.md.** Keep references one level deep. If a reference file points to a second file, Claude may only preview the start of that second file, and miss what's further down.
- **Give long files a contents list.** Any reference file over 100 lines should open with one, so Claude can see everything the file covers even from a partial read.
- **Split by topic and name files plainly.** Claude then opens only the file this conversation needs. `refunds.md` tells Claude what's inside, where `notes2.md` doesn't.

When to reach for it: once a skill covers more than one situation, or carries detail that only some conversations need.

Example: a skill for writing product descriptions for an online shop. SKILL.md holds the house style and the steps. It then says "For clothing, read `references/clothing.md`. For furniture, read `references/furniture.md`." When you ask for a sofa description, the clothing sizing notes never load.



PATTERN 4

Script the fixed steps

Some steps in a skill have one right answer, such as adding up totals or renaming files to a fixed pattern. Claude can work these out each time, but a short script that does the step is more reliable than code Claude writes fresh.

A script in a skill also costs very little. Claude runs it, and only the output enters the conversation. The script's own code never loads.

Anthropic describes this as choosing how much freedom to give Claude. Where many approaches are fine, plain written guidance is enough. Where the step is fragile or has to happen in a set order, give Claude an exact script and tell it not to change the command.

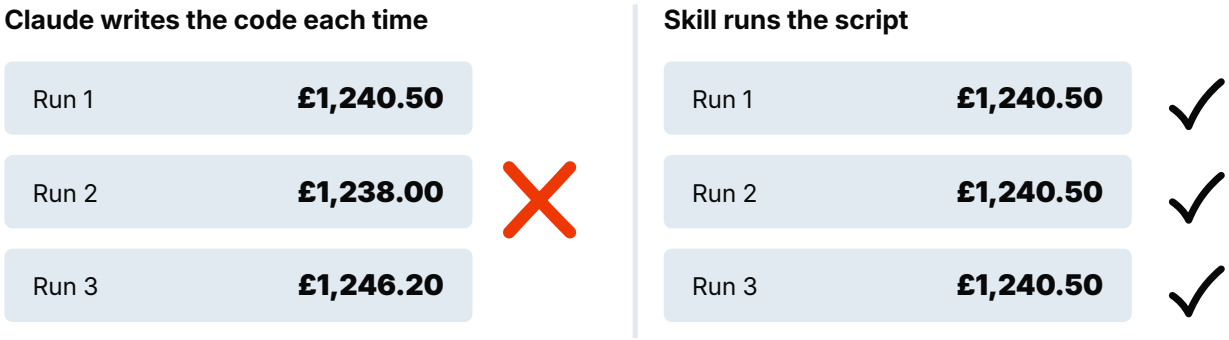
Be clear about what you get. The scripted step gives the same result from the same input every time. The parts of the skill that Claude writes will still vary a little, which is why the patterns in sections 5 and 6 help too.

A few rules from the docs:

- Say whether Claude should **run** the script or **read** it for reference. Running is usually what you want.
- Make the script deal with its own errors and print a clear message, rather than failing and leaving Claude to guess.
- List any packages the script needs. In the Claude app, whether a skill can install packages depends on your network settings.

When to reach for it: any step where a different result each time would be a mistake.

Example: a skill that prepares a monthly expenses summary. A script reads the export from your accounting software, totals each category and flags any claim over your limit. Claude then writes the short commentary around those figures.



PATTERN 5

Templates and examples as assets

Claude matches a pattern it can see far better than one it has to imagine from a description. So for any skill where the look or shape of the output counts, put a template or a finished example in the skill folder. The open skills standard keeps these in an `assets/` folder.

Anthropic suggests matching how strict the template is to how strict you need to be:

- **Strict:** "Always use this exact structure." For output something else reads, such as a form or a file that gets imported.
- **Flexible:** "Here's a sensible default, adapt it to fit." For writing where the content should shape the sections.

Examples work the same way. A couple of input and output pairs show the style and level of detail more clearly than a paragraph describing them.

When to reach for it: whenever you find yourself describing a layout or a tone in words.

Example: a skill for quotes at a small building firm. `assets/quote-template.docx` holds the layout, logo and terms. `references/examples.md` shows two finished quotes, one for a bathroom refit and one for a roof repair, each beside the notes it was written from.

PATTERN 6

Checklists and check-your-own-work loops

On a long piece of work, Claude can skip a step it would normally take. Anthropic's fix is a checklist in the skill that Claude copies into its reply and ticks off as it goes. You can then see which steps it did.

The stronger version is a loop. Claude checks its work and fixes what's wrong, then checks again, and only finishes when the work passes. Anthropic says this pattern greatly improves the quality of what comes back.

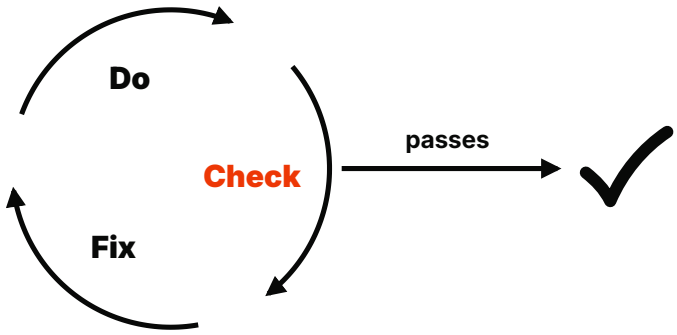
The check doesn't need code. It can be a reference file Claude compares its work against, such as a style guide or a list of rules. Where the check can be scripted, a script is more dependable, and its error messages should say exactly what's wrong so Claude can fix it.

For changes that are hard to undo, Anthropic adds one more step. Claude writes a plan first, the plan gets checked, and only then are the changes made.

When to reach for it: work with several steps, and anything where one missed step spoils the result.

Example: a skill that tidies a customer mailing list before a campaign.

```
Copy this checklist and tick each step:
- [ ] Remove duplicate email addresses
- [ ] Remove anyone marked "unsubscribed"
- [ ] Check every row has a first name
- [ ] Run scripts/check_list.py and fix anything it reports
- [ ] Run the check again. Finish only when it reports no problems
```



PATTERN 7

Small and focused, not one giant skill

The Help Center's advice is to make separate skills for separate kinds of work. Skills can't call each other directly, but Claude can load more than one skill at once and combine them on its own. That makes several small skills more flexible than one that tries to cover everything.

A giant skill has its own problems. Its description has to cover so much that it fires at the wrong moments. Its body loads in full even when you need one small part. And it's harder to test, because a change for one situation can break another.

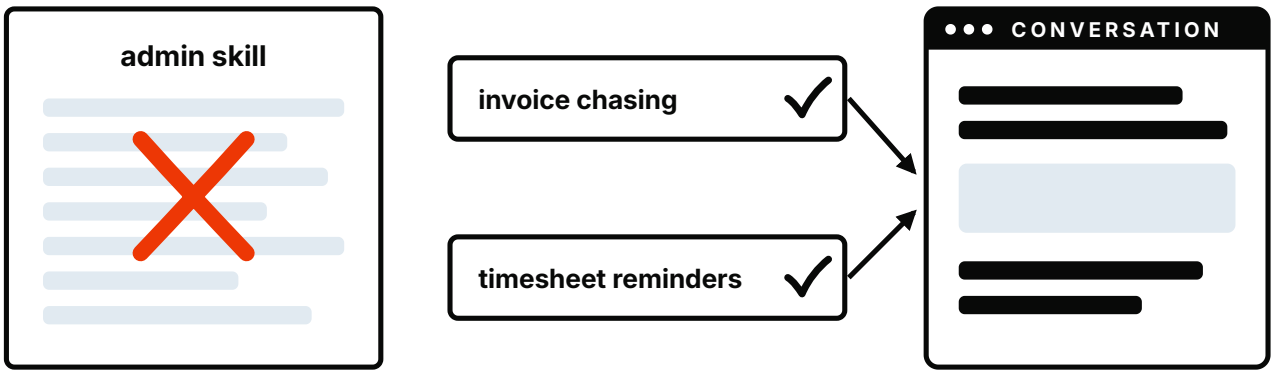
Too many skills cause problems too, so focused doesn't mean endless. Every skill you have switched on sends its name and description with each message, at roughly 100 tokens each. Two skills with overlapping descriptions compete, and the wrong one can win. In Claude Code, the list of skill

descriptions has a fixed budget, and once it's full the least-used skills lose their descriptions and stop loading on their own.

Anthropic's guidance for teams is to start narrow and merge later. Combine two skills into one only when testing shows the merged skill does as well as the pair did.

When to reach for it: when a skill's description needs "and" more than once, or when two skills keep firing for the same message.

Example: rather than one "admin" skill, keep a skill for chasing unpaid invoices and a separate one for weekly timesheet reminders. Switch off the skills you haven't needed in months.



PATTERN 8

Skills alongside connectors and projects

A skill holds the method. Connectors and projects supply what the method needs, and each pairing has its own pattern.

With a connector

A connector gives Claude access to an outside service, such as your CRM or your calendar. A skill teaches Claude how you want that access used. Anthropic describes the two as made to work together.

- **Name the tool.** If the skill relies on a connector, say which connector and which of its tools to use. Anthropic recommends writing the connector's name in front of the tool's name, so Claude doesn't pick the wrong one when several connectors are on.
- **Keep passwords and keys out.** Never write login details into a skill. Let the connector handle access.
- **Expect to switch both on.** A plugin can bundle skills with a connector. If you remove the plugin, the connector stays connected until you disconnect it yourself.

Example: with your CRM connected, a skill for logging sales calls says which CRM tool to call, which fields to fill and which ones to leave alone.

With a project

A project's instructions and files are loaded every time you start a conversation inside that project. A skill loads only when the work matches its description, and it works across all of Claude. So keep background facts in the project and the repeatable method in the skill.

Example: a project for one client holds their brand guide and contract. A general skill for writing monthly client reports runs inside that project and picks up the client's details from the project files. The same skill works for every other client project without changes.



PATTERN 9

Test before you trust

A skill that seems to work on one example can still fail on the next one. Anthropic's skill-creator can test a skill properly, and it's available in the Claude app and as a plugin you install in Claude Code.

Ask skill-creator to write evals for your skill. An eval is a test prompt with a description of what good output looks like, plus any files the prompt needs. Anthropic suggests three to five per skill. Include messages that should trigger the skill and some that shouldn't, plus a borderline one or two.

skill-creator can then:

- **Run a benchmark.** It runs your evals and reports the pass rate, with the time and tokens each run took. Every run starts in a clean context, so one test can't affect the next.
- **Run a blind comparison.** A separate agent compares two versions of the skill, or the skill against no skill, without knowing which output came from which. That tells you whether a change helped or only felt like it did.
- **Tune the description.** It tests the description against sample prompts and suggests changes so the skill fires when it should and stays quiet when it shouldn't.

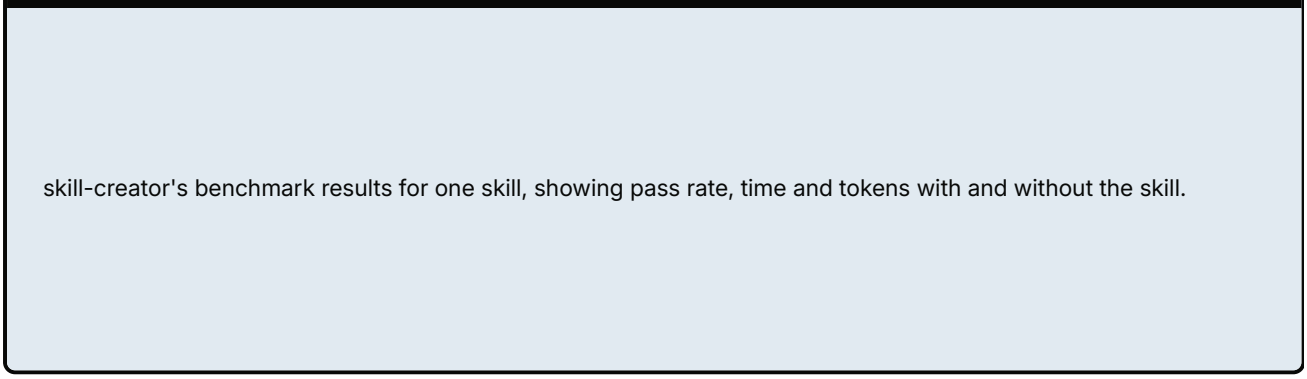
Anthropic also says to test with each model you plan to run the skill on. A smaller model may need more guidance than a larger one, and a larger one may not need the explanations at all.

Evals take far more time and usage than a normal reply, because several agents run at once. Save them for skills you rely on.

When to reach for it: before you share a skill, after any big change, and after a new Claude model arrives.

Example: you've rewritten your complaint-reply skill. A blind comparison against the old version on five past complaints shows whether the new one actually writes better replies.

••• SCREENSHOT 2



skill-creator's benchmark results for one skill, showing pass rate, time and tokens with and without the skill.

PATTERN 10

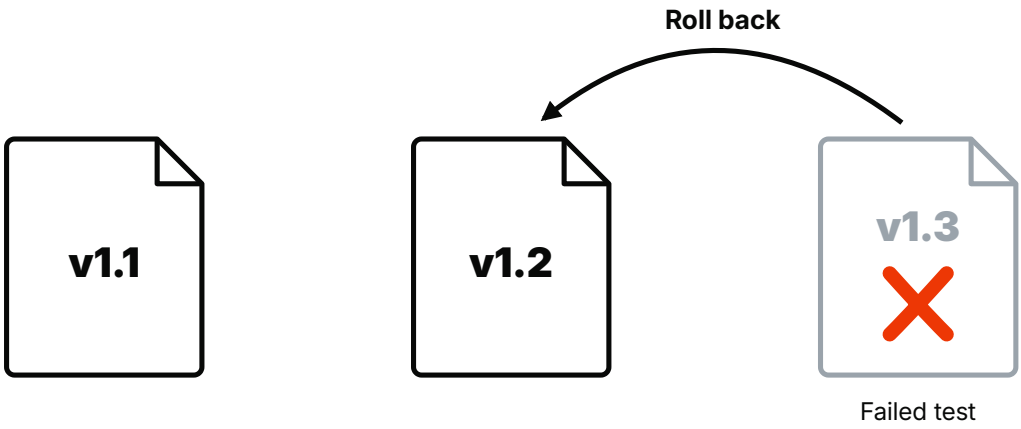
Change safely and keep versions

Changing a skill can break it, and on Team and Enterprise the change can reach other people straight away.

- **Keep a copy before you change it.** Download the skill file first. If you delete a skill, the only way back is to upload the file again.
- **Change one thing at a time.** The Help Center's advice is to test after each significant change, rather than building lots of changes at once.
- **Know who gets the change.** On Team and Enterprise, people you've shared a skill with get your updated version automatically the next time they use the skill. A skill published to your organisation is different: everyone stays on the approved version until your update has been approved.
- **Read a proposed update before saving it.** If you record your screen and the recording overlaps a skill you already have, Claude proposes an update to that skill instead of a new one. Open the content and read it before you click Update.
- **Note the version.** The open skills standard has an optional `metadata` field where you can store a version number, such as `version: "1.2"`. Keep the previous version's file so you can go back if the new one fails its tests. If you work with Git, Anthropic recommends keeping skill folders there for history and rollback.
- **Move outdated steps aside.** Avoid instructions with dates in them ("before March, use the old form"). Anthropic suggests putting the current method in the main text and any old method in a short "Old patterns" section at the end.

When to reach for it: any edit to a skill other people rely on, or one you'd struggle to rebuild.

Example: you add a new discount rule to a quoting skill your whole team shares. You keep a copy of the current file, make the one change, run your evals, then save it.



PATTERN 11

Know when not to make a skill

A skill is the right tool for repeatable work with a method Claude wouldn't follow on its own. For everything else, something simpler works better.

WHAT YOU WANT	USE THIS INSTEAD
Help with something once	An ordinary message
A preference in every conversation	Your custom instructions
Background facts for one area of work	A project
Access to an outside service	A connector
A rule that must never be broken, in Claude Code	A hook, which always runs, where a skill is read and interpreted

Also skip a skill when Claude already does the work well. Run the baseline from section 1 first. If Claude gets it right with no skill, a skill only adds to every message.

The same goes for skills you already have. Anthropic notes that when a skill teaches Claude something it couldn't do before, a newer model may manage it unaided. If a new model passes the skill's evals without the skill loaded, the skill isn't broken, but you may no longer need it. Skills that capture your own way of working, such as your firm's steps for approving an order, tend to last longer.

Example: a skill you made last year for formatting spreadsheet tables. Run your evals with the skill switched off. If the results come out the same, switch it off for good.

AT A GLANCE

Quick reference

••• QUICK REFERENCE

- **Baseline first.** Run the work with no skill and write down what goes wrong. Write only those fixes.
- **Trigger-ready description.** Say what the skill does and when to use it, in the third person, with the words you'd type. The app accepts up to 1,024 characters; keep it to a sentence or two.
- **Short front page.** SKILL.md under 500 lines. Reference files linked straight from SKILL.md, one level deep. A contents list in any file over 100 lines.
- **Script the fixed steps.** Only the script's output enters the conversation. Say "run", and list the packages it needs.
- **Templates and examples.** Put them in `assets/`. Say whether the template is strict or a starting point.
- **Checklists and loops.** Claude copies the checklist and ticks it off. Do, check, fix, check again.
- **Small and focused.** Separate skills for separate work. Each one on costs about 100 tokens per message. Switch off what you don't need.
- **Connectors and projects.** Connector for access, project for facts, skill for method. No passwords in a skill.
- **Test before you trust.** skill-creator evals: three to five prompts, including ones that shouldn't trigger. Benchmark, blind comparison, description tuning.
- **Change safely.** Download a copy first. One change, then test. Shared skills update for others at next use.
- **Know when not to.** One-off work, always-on preferences, background facts and access each have a better home.