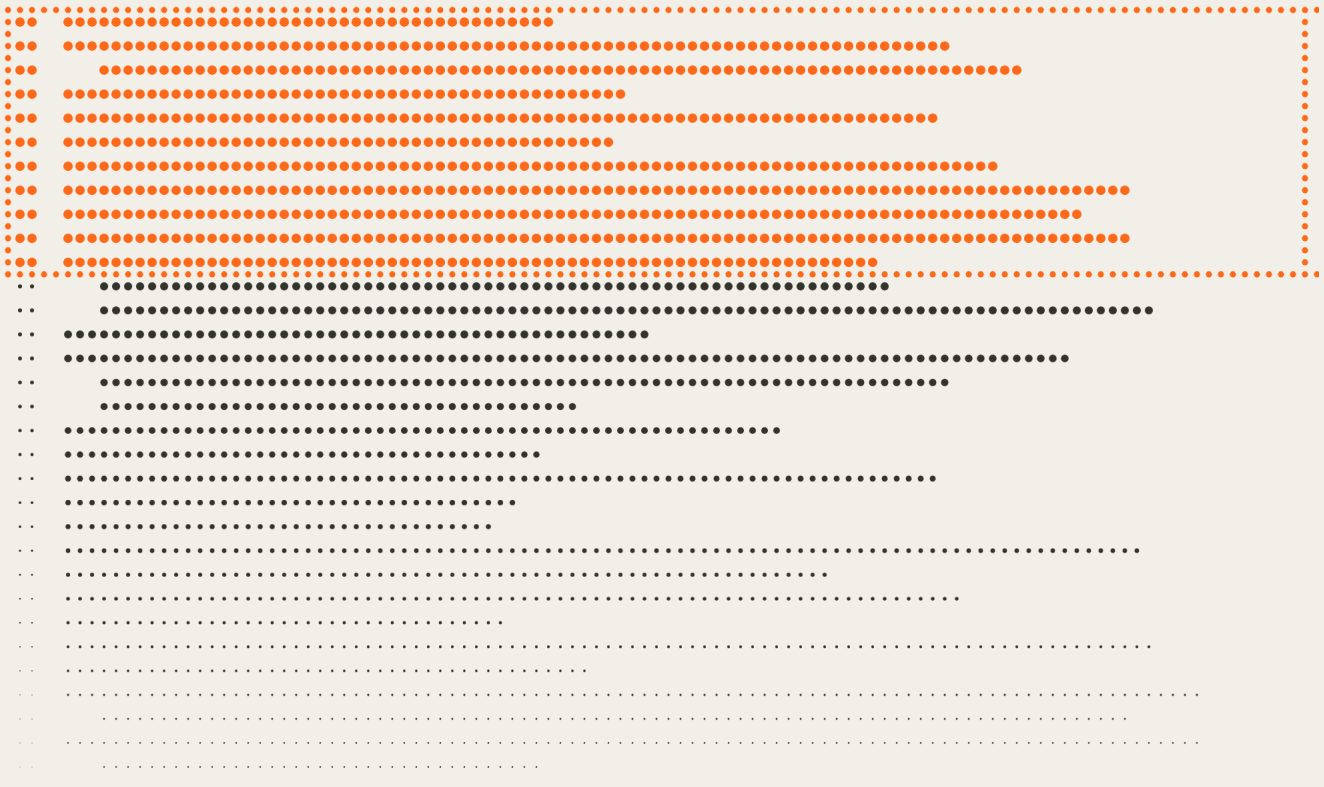
The 10 SKILL Rules

for Claude's 5.5 models, with a prompt that audits them all

A ROBONUGGETS GUIDE

Every rule from Anthropic's official skill-writing guide that the video covers, with the exact source words, how to check your own skills, the fix, and one prompt that audits every skill you have and shows you what it would change first.



README

WHAT THIS IS - the 10 skill rules from the video, plus three more from Anthropic's guides. Every rule comes from Anthropic's official skill-writing guide or the Claude Code docs, quoted word for word with a link to the section, then how to check a skill for it and the fix.

HOW TO USE IT - the audit prompt is on page 3. Send this whole PDF to your AI agent with that prompt, and it audits every skill in your folder against every rule, shows you what it would change, and changes only the fixes you approve. Pages 4 to 9 explain the 10 rules, page 10 has three more, and page 11 is skill-creator-plus, a free skill that already follows all of them.

THE COMMUNITY - skool.com/robonuggets, where we build and sell AI systems, and master Claude Code and Claude Design along the way.

How to use it

A skill is a folder with a SKILL.md file that teaches Claude how to do a job the same way each time. The audit works on a whole folder of them at once.

1. **Back up your skills folder.** A git commit or a plain copy is enough. The audit changes nothing until you say so, but a backup makes any fix easy to undo.
2. **Send this PDF to your agent with the prompt on page 3.** Claude Code, the Claude app or any agent that can open your skills folder. Swap the orange part for your folder: personal skills in Claude Code live in `~/.claude/skills`, and a project's skills in `.claude/skills`.
3. **Read the report.** One table first, then every skill rule by rule, with the file and line for each problem and the fix it would make.
4. **Approve fixes by number.** Reply with the numbers you want. The agent applies only those, checks the changed skills again, and lists what it changed.

> **The prompt is also a text file.** The link in the video description opens the same prompt as plain text, ready to paste.

The rules at a glance

THE 10 RULES FROM THE VIDEO

1	Progressive Disclosure	p4
2	Content Lists	p5
3	Degrees of Freedom	p5
4	Testing skills on the models you use	p6
5	Optimise your skill writing	p6
6	Workflows and checklists	p7
7	Feedback loops	p7
8	Common patterns	p8
9	Build for shareability	p8
10	Hooks for must-hold rules	p9
+3	Three more from Anthropic's guides	p10
+	skill-creator-plus, the free skill	p11

THE AUDIT One prompt for every skill you have

Send this PDF to your agent, then paste the prompt with your folder in the orange spot. It reports first and changes nothing until you approve fixes by number.

01 · THE SKILL AUDIT · PASTE THIS WITH THE PDF

Audit every skill in my skills folder against the skill-writing rules below, which come from Anthropic's official skill-writing guide and the Claude Code docs. Show me what you would change before you change anything.

My skills folder: **[the path, for example ~/.claude/skills, or .claude/skills inside a project]**

Check every skill, and every file its SKILL.md links to, against these 10 rules:
1. Progressive Disclosure. The SKILL.md body is under 500 lines, details sit in their own files, and every file is linked straight from SKILL.md, one level deep. Flag any file Claude can only reach through another file.
2. Content Lists. Every reference file over 100 lines starts with a short contents list.
3. Degrees of Freedom. The detail matches the risk: exact scripts or commands for fragile or costly steps, plain instructions where many approaches work. Flag risky steps left loose and simple steps locked down.
4. Testing on the models you use. Haiku needs enough guidance, Sonnet needs clear and efficient steps, and Opus and Fable should not be over-explained. Flag instructions that look written for older models and may now be too prescriptive, as candidates to test, not to delete.
5. Optimise your skill writing. Keep only what Claude can't work out on its own, like company details, prices, terms and internal rules. Flag paragraphs that explain general knowledge. The description is in the third person and says what the skill does and when to use it.
6. Workflows and checklists. Jobs with many steps have clear numbered steps, a checklist Claude can copy and tick off, and a go-back line after each check step.
7. Feedback loops. Quality-critical work runs check, fix, repeat until it passes (a style guide counts as the check), and scripts give specific error messages that name the problem.
8. Common patterns. Templates say how strict they are, output that depends on style has input and output examples, and jobs that branch have a clear fork.
9. Build for shareability. Every package or tool the skill needs is named with its install line (like pip install pypdf) and listed in SKILL.md. No paths that only exist on my computer, and forward slashes in every path.
10. Hooks for must-hold rules. Find every rule written as never, always or must that has to hold every single time, and suggest moving it into a hook (in Claude Code, a hook can live in the skill's hooks frontmatter). Say which event it would run on.

Also check these three from Anthropic's guides:
11. The skill has at least three test prompts, and a baseline run without the skill.
12. The most important instructions sit near the top of SKILL.md.
13. No instruction asks Claude to write out or echo its reasoning in the reply. A short explanation of the answer is fine.

How to report:
- Change nothing yet.
- Start with one table: a row per skill, its number of fails, and its worst problem.
- Then, skill by skill, go rule by rule: pass, fail or not applicable, with the file and line number as evidence, and the exact fix for every fail.
- End with every fix in one numbered list, biggest impact first.

Then wait for me. Apply only the fixes I approve by number. After applying them, check each changed skill against the same rules again, and if anything still fails, go back and fix it. Finish with a short list of what changed, file by file.

> **Big folder?** Start with your five most-used skills, or the ones that send money, delete things or talk to clients. Those are where a missed rule costs the most.

01 Progressive Disclosure

Keep the main SKILL.md under 500 lines, split everything else into its own files, and link every one of them straight from SKILL.md.

> "Keep SKILL.md body under 500 lines for optimal performance."
>
> "SKILL.md serves as an overview that points Claude to detailed materials as needed, like a table of contents in an onboarding guide."
>
> ANTHROPIC'S GUIDE · best-practices#progressive-disclosure-patterns

Claude only loads what it needs, when it needs it. Take a skill for invoices. The main file has the steps, and next to it sit a pricing file, a clients file and a scripts folder. When Claude raises an invoice, it opens only the pricing file. The scripts never get read at all.

```
invoicing/
  SKILL.md          the steps, and a link to each file below
  pricing.md        rates, rush fees, discounts
  clients.md        who gets billed, and how
  scripts/          run, never read
```

THE CATCH: FILES BEHIND FILES

> "Claude may partially read files when they're referenced from other referenced files. When encountering nested references, Claude might use commands like `head -100` to preview content rather than reading entire files, resulting in incomplete information."
>
> "Keep references one level deep from SKILL.md. All reference files should link directly from SKILL.md to ensure Claude reads complete files when needed."
>
> ANTHROPIC'S GUIDE · best-practices#avoid-deeply-nested-references

So if SKILL.md points to a second file, and that file points to a third, Claude might only preview the first hundred lines of the third. Anything further down may as well not exist.

```
Too deep
SKILL.md      links to advanced.md
advanced.md   links to details.md
details.md    the real information,
              maybe only previewed
```

```
One level deep
SKILL.md  links to advanced.md
SKILL.md  links to reference.md
SKILL.md  links to examples.md
          every file read in full
```

HOW TO CHECK

- Count the lines in SKILL.md, after the frontmatter at the top.
- List every file SKILL.md links to.
- Flag any file that is only linked from another file, never from SKILL.md.

THE FIX

- Near 500 lines: move detail that only some jobs need into its own file.
- A nested file: link it straight from SKILL.md, or fold it into the file that uses it.

02 · FROM THE VIDEO · A QUICK FIRST CHECK

```
Go through my five most-used skills. For each one, tell me if SKILL.md is over 500 lines, list every file
it links to, and flag any file that is only reachable through another file. Show me the list before you
change anything.
```

02 Content Lists

Any file over a hundred lines gets a short contents list at the very top.

> "For reference files longer than 100 lines, include a table of contents at the top. This ensures Claude can see the full scope of available information even when previewing with partial reads."
>
> ANTHROPIC'S GUIDE · best-practices#structure-longer-reference-files-with-table-of-contents

It pairs with rule 1. Even when Claude only previews the top of a file, the list tells it what's further down, so it can jump to the section it needs. The pricing file below now shows that discounts sit near the bottom.

```
# Pricing
## Contents
- Standard rates
- Rush jobs
- Payment terms
- Discounts          down near line 240
```

HOW TO CHECK

- Count the lines in every file in the skill folder.
- For each one over 100, look for a contents list in its first few lines.

THE FIX

- Add a "Contents" heading at the top with one line per section, in the same order as the file.

03 · FROM THE VIDEO

```
Go through my skills folder and add a contents list to every file over 100 lines.
```

03 Degrees of Freedom

Match the detail to the risk. Risky steps get exact steps and guardrails, low-stakes steps get room to be creative.

> "Match the level of specificity to the task's fragility and variability."
>
> "Narrow bridge with cliffs on both sides: There's only one safe way forward. Provide specific guardrails and exact instructions (low freedom)." "Open field with no hazards: Many paths lead to success. Give general direction and trust Claude to find the best route (high freedom)."
>
> ANTHROPIC'S GUIDE · best-practices#set-appropriate-degrees-of-freedom

LEVEL	WHAT ANTHROPIC SAYS IT IS	USE IT FOR, SAY
High freedom	Text-based instructions. "Multiple approaches are valid"	Brainstorming ways to automate your business
Medium freedom	Pseudocode or scripts with parameters. "A preferred pattern exists"	A weekly report: a set shape, with room for new findings
Low freedom	Specific scripts, few or no parameters. "Operations are fragile and error-prone"	Raising an invoice, or tax documents

HOW TO CHECK

- For each step, ask what a mistake there would cost.
- Flag costly steps written loosely, and simple steps locked down hard.

THE FIX

- Lock costly steps to an exact script or command. Loosen the rest to a plain goal. One skill can mix all three: the invoice email wide open, the step that creates the invoice locked to a script.

04 Testing skills on the models you use

A skill is only as good as the model running it, so test it on the models you actually run it with.

> "Skills act as additions to models, so effectiveness depends on the underlying model. Test your Skill with all the models you plan to use it with."
>
> ANTHROPIC'S GUIDE · best-practices#test-with-all-models-you-plan-to-use

MODEL	ANTHROPIC'S QUESTION TO ASK
Haiku (fast, economical)	"Does the Skill provide enough guidance?"
Sonnet (balanced)	"Is the Skill clear and efficient?"
Opus (powerful reasoning)	"Does the Skill avoid over-explaining?" Fable sits at the top tier with Opus.

> "Skills developed for prior models are often too prescriptive for Claude Fable 5 and can degrade output quality. Review and consider removing older instructions if default performance is better."
>
> PROMPTING CLAUDE FABLE 5 · prompting-claude-fable-5#recommended-scaffolding-changes

It works the other way too. A skill you always run on the smartest model may be detailed enough that Haiku does the job. Testing every skill on every model burns tokens, so test the skills you rely on most, and the ones where one mistake costs you.

HOW TO CHECK

- Note which model runs each skill.
- Run three real tasks on it and ask that model's question above.

THE FIX

- Older, step-by-step instructions: test with them removed, and keep the cut only if the output is as good or better.

05 Optimise your skill writing

Keep only what Claude can't work out alone, and write the description in the third person.

> "The context window is a public good." "Only add context Claude doesn't already have." Ask: "Does Claude really need this explanation?" and "Does this paragraph justify its token cost?"
>
> ANTHROPIC'S GUIDE · best-practices#concise-is-key

> "Always write in third person. The description is injected into the system prompt, and inconsistent point-of-view can cause discovery problems." The description "should include both what the Skill does and when to use it."
>
> ANTHROPIC'S GUIDE · best-practices#writing-effective-descriptions

HOW TO CHECK

- Flag any paragraph that explains general knowledge, like what an invoice is.
- Read the description: does it say "I can" or "you can"?

THE FIX

- Cut to your own facts: prices, terms, clients, internal rules.
- Write "Creates client invoices and sends payment reminders", not "I can help you with invoices and payment reminders".

06 Workflows and checklists

Lay long jobs out as clear steps, give Claude a checklist it copies and ticks, and add a go-back line after every check.

> "Break complex operations into clear, sequential steps. For particularly complex workflows, provide a checklist that Claude can copy into its response and check off as it progresses."
>
> "Clear steps prevent Claude from skipping critical validation." Anthropic's own example ends: "If verification fails, return to Step 2."
>
> ANTHROPIC'S GUIDE · best-practices#use-workflows-for-complex-tasks

```
Copy this checklist and track your progress:
- [ ] Step 1: Pull this month's completed jobs
- [ ] Step 2: Match each job to the client and the price
- [ ] Step 3: Create the invoices
- [ ] Step 4: Check every total against the job list
- [ ] Step 5: Send them out
If any total doesn't match the job list, go back to Step 2.
```

The go-back line is the part that matters. Without it, a wrong invoice can go out just because a box got ticked.

HOW TO CHECK

- Find every job with more than three or four steps.
- Look for numbered steps, a checklist, and what happens when a check fails.

THE FIX

- Number the steps, add the copy-this checklist, and put a go-back line under each check step.

07 Feedback loops

Have the skill check its own work: run a check, fix what fails, and repeat until it passes.

> "Common pattern: Run validator → fix errors → repeat. This pattern greatly improves output quality."
>
> The check doesn't have to be code. In Anthropic's first example, "The 'validator' is STYLE_GUIDE.md, and Claude performs the check by reading and comparing."
>
> ANTHROPIC'S GUIDE · best-practices#implement-feedback-loops

> "Make validation scripts verbose with specific error messages such as "Field 'signature_date' not found. Available fields: customer_name, order_total, signature_date_signed" to help Claude fix issues."
>
> ANTHROPIC'S GUIDE · best-practices#create-verifiable-intermediate-outputs

HOW TO CHECK

- Find the work where quality matters: client copy, documents, data.
- Is there a check, and does a fail send Claude back to fix it?
- Do the scripts say what broke, or just "Error"?

THE FIX

- Add a short checklist to review against, a revise step, and "only finish when every check passes".
- Make error messages name the problem and what does exist.

08 Common patterns

When the output needs a set shape, use a template, examples, or a fork in the road.

PATTERN	ANTHROPIC'S WORDS	THEIR EXAMPLE
Template	"Provide templates for output format. Match the level of strictness to your needs." Strict: "ALWAYS use this exact template structure". Flexible: "Here is a sensible default format, but use your best judgment".	A report: title, executive summary, key findings, recommendations
Examples	Where quality depends on seeing examples, "provide input/output pairs just like in regular prompting." "Examples convey the desired style and level of detail to Claude more clearly than descriptions alone."	Three commit messages, each a change and the summary they want
Conditional workflow	"Guide Claude through decision points." "Creating new content? → Follow "Creation workflow" below." "Editing existing content? → Follow "Editing workflow" below."	A Word document skill: create or edit

ANTHROPIC'S GUIDE · #template-pattern · #examples-pattern · #conditional-workflow-pattern

HOW TO CHECK

- Templates: does each one say if it's strict or a default?
- Style-heavy output: are there input and output pairs?
- Jobs that branch: is the fork written out?

THE FIX

- Add the strict or default line, two or three real example pairs, and an "if this, follow these steps" fork.

09 Build for shareability

Never assume the tools your skill needs are installed on someone else's computer. Name them, with the install line.

"Don't assume packages are available." Bad: "Use the pdf library to process the file." Good: "Install required package: `pip install pypdf`".

ANTHROPIC'S GUIDE · best-practices#avoid-assuming-tools-are-installed

"List required packages in your SKILL.md." On claude.ai, skills "Can install packages from npm and PyPI and pull from GitHub repositories". On the Claude API, a skill "Has no network access and no runtime package installation".

ANTHROPIC'S GUIDE · best-practices#package-dependencies

Your invoicing skill works on your computer. The first time a teammate runs it, it can't even create the PDF, because the package isn't there. The same guide also says to use forward slashes in paths (`reference/guide.md`), never backslashes, so a skill made on Windows still works on a Mac.

HOW TO CHECK

- List every package, command line tool and script the skill uses.
- Look for paths that only exist on your machine, and backslashes.

THE FIX

- Add a short "Needs" list to SKILL.md with each install line.
- Swap personal paths for paths inside the skill folder.

10 Hooks for must-hold rules

If a rule has to hold every single time, move it out of the words and into a hook.

Say your invoicing skill has one rule that can never be broken: never send an invoice above ten thousand dollars without your sign-off. Written in capitals, Claude follows it most of the time. For a rule like this, most of the time isn't enough.

A hook is a bit of code that Claude Code runs on its own at a set moment, like right before Claude runs a command, and it runs whether or not Claude is following the skill. Hooks are a Claude Code feature, so this rule is for skills you run there.

> "Claude skipped a rule that must hold every time: move the rule into a hook. Claude Code runs a hook every time its event occurs, such as before each file edit, whether or not Claude is following the skill. To keep the rule with the skill, define the hook in the skill's hooks frontmatter. That hook applies from the time the skill is invoked until the session ends."
>
> CLAUDE CODE DOCS · skills#claude-stops-following-a-skill

> The `hooks` frontmatter field: "Hooks that Claude Code registers when the skill is invoked and keeps running for the rest of the session."
>
> CLAUDE CODE DOCS · skills#frontmatter-reference · FORMAT AND EXAMPLES: hooks#hooks-in-skills-and-agents

Anthropic's example in the hooks docs is a skill called secure-operations. Its frontmatter holds a hook that runs a security check script before every Bash command, so the check happens every time, not most of the time.

THE INVOICING RULE, AS A HOOK

PART	WHAT IT DOES
When it runs	Right before Claude runs the command that sends an invoice
What it checks	The invoice total
What happens	Above 10,000 dollars, it stops the send and holds it for your sign-off. Below that, the send goes ahead.
Where it lives	In the skill's `hooks` frontmatter, so it travels with the skill

HOW TO CHECK

- Search the skill for never, always, must and words in capitals.
- For each one, ask: would one miss cost money, data or a client?

THE FIX

- Pick the few rules where one mistake really costs you, and make those your hooks.
- Keep the rest as plain words, with the reason beside them.

04 · TURN ONE RULE INTO A HOOK · CLAUDE CODE

```
Look through [the skill name] for rules that must hold every single time, like anything written as never,
always or must. List them, and for each one tell me what a single miss would cost. Then, for the one I
pick, write a hook in the skill's hooks frontmatter that enforces it, using the hooks docs at
https://code.claude.com/docs/en/hooks#hooks-in-skills-and-agents. Show me the hook and the script it runs
before you save anything, then test it on one case that should pass and one that should be blocked.
```

BONUS

Three more from Anthropic's guides

These didn't make the video, but the audit prompt checks them too.

11 Build the test before the skill

Write three test tasks and see how Claude does without the skill first. Then write the smallest skill that fixes what went wrong.

> "Create evaluations BEFORE writing extensive documentation. This ensures your Skill solves real problems rather than documenting imagined ones." The steps: run Claude without a Skill, "Build three scenarios that test these gaps", measure that baseline, then "Create just enough content to address the gaps and pass evaluations".
>
> ANTHROPIC'S GUIDE · best-practices#build-evaluations-first

> Collect a few realistic prompts, then "run each one in a fresh session with the skill available and again with it turned off, and compare the results. A fresh session matters because leftover context from authoring the skill will mask gaps in the written instructions."
>
> CLAUDE CODE DOCS · skills#evaluate-and-iterate-on-a-skill

The fix: keep three real test tasks next to each skill you rely on. Run them in a fresh chat with the skill on and off after every big change.

12 Put your most important instructions at the top

In a long Claude Code session, only the start of a skill may survive, so the rules that matter most go first.

> "After compaction, Claude Code can keep only the start of an invoked skill, so put the most important instructions near the top of SKILL.md."
>
> When a long chat is summarised, Claude Code "re-attaches the most recent invocation of each skill after the summary, keeping the first 5,000 tokens of each."
>
> CLAUDE CODE DOCS · skills#claude-stops-following-a-skill · skills#skill-content-lifecycle

The fix: open SKILL.md with the few rules that must never be missed. Background, history and long examples go lower down or into their own files.

13 Never tell Claude to write out its reasoning

On the newest models, a skill that asks Claude to echo its thinking in the reply can get the request declined.

> "Prompts, skills, or harness instructions that tell the model to echo, transcribe, or explain its internal reasoning as response text may trigger the reasoning_extraction refusal category on Claude Fable 5." "Audit existing skills and system prompts for reflection or show-your-thinking instructions when migrating." "You can still ask for a short explanation of the answer or a summary of the actions taken."
>
> PROMPTING CLAUDE FABLE 5 · prompting-claude-fable-5#recommended-scaffolding-changes

The 5.5 guides agree. Opus 5.5: such requests "may be declined with the reasoning_extraction category". Sonnet 5.5: those instructions "invite reasoning_extraction declines". **The fix:** find lines like "think step by step and write out your reasoning" and change them to ask for the answer, plus a short explanation if you need one.

THE FREE SKILL skill-creator-plus

A free Claude skill that audits, builds and improves other skills against Anthropic's own skill-writing rules, including what changed for the newest Claude models. It already follows every rule in this guide.

WHY WE MADE OUR OWN

Anthropic's own skill-creator was last changed in March 2026 (checked 5 October 2026), and it still asks for contents lists on files over 300 lines, where the skill-writing guide says 100. It also has nothing on one level deep, degrees of freedom, testing on Haiku, Sonnet and Opus, checklists, third-person descriptions or hooks. skill-creator-plus covers all of them, plus the Fable 5, Opus 5.5 and Sonnet 5.5 guidance.

What it does

MODE	SAY SOMETHING LIKE	YOU GET
Audit	"Audit my deploy skill against Anthropic's rules." Or: "Check every skill in ~/.claude/skills and tell me what to fix first."	A report: one row per rule, fails first, ranked fixes. Nothing is edited until you pick.
New	"Make this a skill, we write release notes like this every Friday."	Test prompts first, then a new skill folder that passes the validator.
Refine	"The report skill keeps forgetting the date filter. Fix the skill."	The rule rewritten around its reason, usually shorter than before.

It checks 46 rules, each with its file and line as evidence, and a validator script checks the mechanical ones in a second.

Install it in Claude Code

Put the folder in your skills folder and name it `skill-creator-plus`. Downloading the ZIP from GitHub and copying the folder there works too.

FOR ALL YOUR PROJECTS

```
git clone https://github.com/robonuggets/skill-creator-plus ~/.claude/skills/skill-creator-plus
```

OR FOR ONE PROJECT ONLY, FROM THE PROJECT FOLDER

```
git clone https://github.com/robonuggets/skill-creator-plus .claude/skills/skill-creator-plus
```

On Windows, `~/.claude` is the `.claude` folder inside your user folder. **On claude.ai:** zip the folder (`SKILL.md` , `references/` , `scripts/` and `LICENSE`) and upload it under Skills in claude.ai settings. The validator needs code execution turned on.

The validator on its own

EVERY SKILL IN A FOLDER, ONE TABLE · RUN IT INSIDE THE SKILL-CREATOR-PLUS FOLDER

```
python3 scripts/validate_skill.py --all ~/.claude/skills
```

Python 3.8 or newer, nothing else to install. Free and MIT licensed: github.com/robonuggets/skill-creator-plus

Where this goes next

The easy part is writing a skill. One prompt and Claude makes you one. The skill that pays is building skills that run real jobs the same way every time, then turning them into AI systems a business will pay for.

That's what we practise every week inside RoboNuggets: building AI systems you can sell, and mastering Claude Code and Claude Design along the way.

THE COMMUNITY

Bring the report your audit made, and fix your skills with people doing the same work. That's skool.com/robonuggets.



The one-page recap

1 **Progressive Disclosure.** SKILL.md under 500 lines, every file linked straight from it, one level deep.

2 **Content Lists.** A contents list at the top of any file over 100 lines.

3 **Degrees of Freedom.** Exact scripts where mistakes cost, plain goals where they don't.

4 **Test on the models you use.** Older skills can be too prescriptive for the newest models. Test what you rely on most.

5 **Optimise your writing.** Only what Claude can't work out alone. Descriptions in the third person.

6 **Workflows and checklists.** A checklist Claude copies and ticks, plus a go-back line.

7 **Feedback loops.** Check, fix, repeat. Error messages that name the problem.

8 **Common patterns.** Templates that say how strict they are, examples, and forks.

9 **Shareability.** Name every package with its install line. Forward slashes.

10 **Hooks.** A rule that must hold every time goes in a hook, not in capitals.

11 **Test first.** Three tasks and a baseline before you write the skill.

12 **Top first.** The most important instructions near the top of SKILL.md.

13 **No reasoning echo.** Ask for the answer and a short explanation, not the thinking.

Every source in this guide

platform.claude.com/docs/en/agents-and-tools/agent-skills/best-practices - Anthropic's skill authoring best practices, the source for rules 1 to 9 and 11

code.claude.com/docs/en/skills - the Claude Code skills docs, for rules 10, 11 and 12

code.claude.com/docs/en/hooks#hooks-in-skills-and-agents - hooks inside a skill, with the secure-operations example

platform.claude.com/.../prompting-claude-fable-5 - the Fable 5 guide, for rules 4 and 13

platform.claude.com/.../prompting-claude-opus-5-5 and .../prompting-claude-sonnet-5-5 - the 5.5 guides, for rule 13

github.com/robonuggets/skill-creator-plus - the free skill

skool.com/robonuggets - the RoboNuggets community

Quotes taken from Anthropic's pages as they read on 3 October 2026.