



How I Work : Skills & Claude Design

Where my skills came from, how they break, and how I keep them alive.



A skill is a trigger and a pointer, not a manual.

TRIGGER

The sentence that decides when Claude reaches for it. You are not in the room for that decision.

POINTER

Where the real material lives. It loads only after the skill fires, so it costs nothing until it is needed.



Most people write the manual. The whole talk is about the other two.

WHAT JUST HAPPENED

I never named a skill.

01 I TYPED

"Make me a Flamingo ad from
ad-typographic..."

Plain English. No slash command, no file
path, no setup.

02 CLAUDE MATCHED

`flamingo-design`

It read one thing to decide: the skill's
description. Nothing else was in context
yet.

03 THE SKILL RAN

`make-ad.py` → `1080×1080 PNG`

Rendered offline in 8 seconds. No Figma,
no Claude Design, no network.

The whole point: I did not have to remember that any of this existed.

Four things people mix up

THING	WHO DECIDES IT RUNS	WHAT IT IS
Skill	Claude, from your words	A folder of instructions that loads itself when the moment matches.
Slash command	You, explicitly	The same thing, invoked by name when you want to force it.
Subagent	Claude, for big lookups	A separate context that goes away and returns a summary.
Routine	A schedule	A prompt that runs on its own, with nobody watching.

A skill can be used by all four. Write it once, and the routine gets it too.

Two parts. Only one of them decides.

```
---  
name: link-builder  
description: "Evaluate external domains for inbound link placement...  
    Use this skill whenever the user mentions link building, backlinks,  
    link exchange, three-way swap, finding placements..."  
---  
  
# everything below here is the body
```

The **description** is the only part Claude reads to decide.
The body is only read *after* it has already fired.

Which is why almost every skill problem is a description problem.

THE GOOD NEWS

You already know how to write one

Google's prompting course teaches five steps. A skill is those five steps written down once instead of retyped every time.

GOOGLE'S FIVE STEPS

WHERE THEY ALREADY LIVE IN `flamingo-content`

Task	The persona and the format. "Your no-BS tech mate", plus the voice principles and formatting rules.
Context	The positioning section. What Flamingo is, and what it is not.
References	Two to five examples. <code>seo-diamond</code> keeps three reference files; every good and bad example pair in the voice section.
Evaluate	The quality checklist at the bottom of the skill.
Iterate	The rewriting workflow, and every fix in the next section of this talk.

A skill is a prompt you only have to write once.

Your session vs. a subagent

YOUR SESSION · WHAT IT STARTS WITH

Your CLAUDE.md and standing rules

Every skill description [the menu](#)

Everything you have said so far

Every file you have opened

It knows what exists.

A SUBAGENT · WHAT IT STARTS WITH

The prompt you wrote for it

no skill descriptions

no conversation history

nothing you said before

It knows what you told it.

```
// the first 5 seconds of my blog routine, this morning
init: model=claude-fable-5 cwd=/home/user/claude-skills
assistant: First, let me find the ClickUp MCP tools and check the repo skills.
tool_use ToolSearch: {"query":"+clickup tasks filter resolve assignees"}
tool_use Glob:      {"pattern":"**/SKILL.md"} ← going looking
```

It had to **search the disk** for its own skills. That one line is why a routine prompt is long, and why you name the skill in it.

THE PRACTICAL QUESTION

Do I have to name the skill?

No. Describe the work, not the tool.

YOU TYPE

WHAT LOADS

"Write a post about SOC as a service"

flamingo-content

"Turn this into a spreadsheet"

flamingo-xlsx

"Reply to this customer"

flamingo-customer-comms

"Find keywords for MSP backup"

seo-diamond

SAY IT IN THREE CASES

Writing a routine – that agent starts blank, so always. · **Two brands fit** – say "an OpenMSP ad", not the skill name. · **It did not fire** – name it to get unstuck, then go fix the description.

Thirteen skills, two plugins

design 8

flamingo-design · **openframe-design** · **openmsp-design** – the three brands

flamingo-docx · flamingo-xlsx · flamingo-report – branded files

blog-og-preview · flamingo-blog-diagrams – blog imagery

marketing 5

flamingo-content – the brand voice

flamingo-customer-comms – customer replies

seo-diamond – keyword and SERP research

link-builder – backlink placements

mini-michael – founder voice

The repo also holds engineering and QA skills, and three vendored third-party ones. Those are other people's. There is a shared **core** underneath the three design systems that nobody invokes directly.

Two of them do almost everything: **flamingo-content** writes, **flamingo-design** draws. The other eleven exist because those two kept needing them.

THE PART THAT LOWERS THE BAR

Most useful skills are one file

6 of 13

are one file and nothing else. No assets, no scripts, no build step.

3

design systems carry 77 of my 78 MB. They are the exception, not the shape.

160

lines in `link-builder`, one of the most-used skills I have.

You do not need to build a design system to start.
You need one file and a good first sentence.

The one that holds judgment

400+

lines, and growing

34

sections

0

assets, scripts or images

It carries nothing you could look up. It carries what we learned the expensive way:

Words we never use

AI tells - hard blocks

Word frequency limits

Voice principles

Channel tone map

MSP language translation

Lead with the opportunity, never blame the reader

Positioning - what Flamingo is

Quality checklist

This is where my corrections stopped being corrections and became permanent.

Written in the words people type

Triggers include: **blog posts**, **SMM posts** (LinkedIn, Instagram, Facebook, X, etc.), **Reddit comments**, **newsletters**, **case studies**, **website copy**, **email copy**, **Slack/Discord messages**, **webinar scripts**, **PR/press**, or any content referencing **MSPs**, **IT automation**, **vendor lock-in**, **RMM/PSA consolidation**, or **Flamingo/OpenFrame**.

WHY IT WORKS

Every **pink** word is a **surface someone types**. Every **cyan** word is a subject they type instead of naming a surface.

THE HABIT

Write the description by listing **the words you would have used** when you asked for this by hand. Not what the skill is. What its moment sounds like.

Eleven lines govern 521 files

```
--- (name + description) ---
```

```
Read the README.md file within this skill, and  
explore the other available files.
```

```
If creating visual artifacts (slides, mocks,  
throwaway prototypes, etc), copy assets out and  
create static HTML files for the user to view.  
[...two more sentences...]
```

```
Key rules: dark-first UI (#161616/#212121/#FAFAFA),  
yellow #FFC008 only for CTAs/highlights, Azeret Mono  
headings + DM Sans body, sharp 3px radii, 2px strokes,  
no shadows, custom stroke icon set via core/icons.
```

11

lines of SKILL.md

521

files it governs

17 MB

tokens, components, icons, fonts

It does not contain the design system.

It **points at it**, and names the handful of rules you would otherwise get wrong.

THE RULE

What goes in the file itself?

From `seo-diamond`, which keeps a 17 KB `SKILL.md` and points at 105 KB of reference beside it:

"Domain Rating is **deliberately not recorded here**. It moves, and a stale number silently mis-sizes every keyword filter downstream."

STAYS IN `SKILL.MD`

What breaks if it is missing. Rules, refusals, and the thing everyone gets wrong on the first try.

GOES IN A FILE YOU NAME

Anything you can look up once you know you need it. It names the **section** too, so only that part gets read.

Six times more material outside the skill than inside it, and none of it costs anything until the skill fires.

A pointer can be a program

FLAMINGO-DESIGN / SKILL.MD

CORE/TOOLS/MAKE-AD.PY

```
## Running the tooling
```

```
cd to the directory containing this  
SKILL.md, then confirm before doing  
real work:
```

```
python3 ../core/tools/make-og.py --help
```

```
If that prints usage, you are in the  
right place. If it prints  
can't open file ... No such file or  
directory, you are not.
```

```
for o in overflow:  
    print(f'  OVERFLOW: slot "{o["id"]}" '  
          f'renders {o["h"]}px against '  
          f'a {o["max"]}px box - shorten '  
          f'the copy')
```

```
return 1 if overflow else 0
```

A pointer does not have to be prose. It can be a program.

The skill's job is to say **when to run it**, and how to tell you are holding it wrong.

The failure mode is documented on purpose. Every renderer path in that skill is relative to the skill's own directory, so the first thing it teaches you is how to know you are in the wrong one.

BOTH ARE CORRECT

Two opposite shapes, one rule

FLAMINGO-CONTENT

400+ lines, 0 files

Everything inline, because it is all **judgment**. There is nowhere else to look this up. If it is not in context, the writing is wrong and nobody notices.

OPENFRAME-DESIGN

11 lines, 521 files

Almost nothing inline, because it is all **reference**. The files are right there and self-describing. Pasting them in would only crowd out the rules.

Same question, opposite answers: what breaks if this is not already in context?

WHEN IT FIRES WRONG

Four ways I have broken this

SYMPTOM	CAUSE, AND THE FIX
Wrong skill fires	Descriptions said what a skill <i>is</i> . Name the surface, and say when not to use it.
No skill fires	It does not sound like the request. Write it in the words your team types.
Acts on stale facts	A number was hardcoded. Replace the fact with where to look it up.
Fires, then fails	It points at files that moved. Use your own skill, and let CI check the paths.



All four are real. All four are in our git history. One per slide from here.

Three skills, one description

```
// flamingo-design
Use this skill to generate well-branded interfaces and assets for Flamingo,
either for production or throwaway prototypes/mocks/etc...
```

```
// openframe-design
Use this skill to generate well-branded interfaces and assets for OpenFrame
(Flamingo's open-source MSP platform), either for production or throwaway
prototypes/mocks/etc...
```

```
// openmsp-design
Use this skill to generate well-branded interfaces and assets for OpenMSP,
either for production or throwaway prototypes/mocks/etc...
```

THE SYMPTOM

Ask for an OpenMSP post, get Flamingo pink. Three near-identical descriptions left Claude **nothing to route on**.

THE TELL

Two of the three carried the same typo, **protoyping**. They were copy-pasted and never re-read.

Name the surface. Point at the others.

```
// openframe-design, today  
OpenFrame product design system - dark-first UI (#161616/#212121/#FAFAFA),  
yellow #FFC008 reserved for CTAs and highlights, Azeret Mono headings +  
DM Sans body, sharp 3px radii, 2px strokes, no shadows, custom stroke icon  
set. Use when building or mocking the OpenFrame product interface, app  
screens, dashboards or openframe.ai.
```

```
For flamingo.run marketing surfaces use flamingo-design;  
for openmsp.ai open-source content use openmsp-design.
```

A good description says when **not** to fire.

Every skill that has a neighbour should name that neighbour.

A skill nobody could trigger

We shipped a skill called `x-api` whose description and body were written **entirely in Japanese**.

WHAT HAPPENED

It worked. It just never ran. Every request in the workspace was in English, so it **matched almost nothing** and sat there for weeks.

THE LESSON

The description is matched against **what your team types**. Not what the skill does, not what is correct – what gets typed.

Removed in the same commit that fixed the design systems. Sometimes the fix is deleting it.

The skill was confidently wrong

DR 17

what seo-diamond had written down for
flamingo.run

DR 45

what it was by then

1,878

keywords already ranking in positions 1-3
while it assumed we were a beginner

That one stale number drove a $KD < 15$ filter on every search, so keyword discovery was **systematically under-targeting our main blog** – quietly, and for months.

Never hardcode a fact that changes.
Write down **where to look it up** instead.

Three paths that pointed at nothing

<code>core/icons/Icon.d.ts</code>	does not exist - names are in <code>names.js</code>
<code>core/icons/<Name>.svg</code>	claimed 1,692 files; there are zero
<code>uploads/openfram-brand-kit.pdf</code>	not shipped

HOW I FOUND IT

By trying to build a mockup with my own skill. The buttons rendered as the word "Button" three times before I gave up and read the file.

THE PRACTICE

Use your own skill from scratch, occasionally. A skill that points at files is only ever as good as its pointers, and prose rots slower than paths.

A pointer-heavy skill needs the same maintenance as code. It just does not fail loudly.

Three copies drifted in three days

BEFORE

The same skills lived in my home folder, in one repo, and in another repo. Nothing synced them. I edited whichever I opened first, and they came apart **twice in three days**.

AFTER

One canonical repo. Everything in `~/ .claude/skills` is a **symlink** into it. Editing a skill *is* editing the tracked file – there is no copy left to forget.

```
$ ls -la ~/.claude/skills
flamingo-content    → ~/src/claude-skills/plugins/marketing/skills/flamingo-content
openframe-design    → ~/src/claude-skills/plugins/design/skills/openframe-design
... 15 of them, no real folders
```

Bonus: symlinks are why `../core/tools/` resolves at all. Copies would silently break the shared core.

The git SHA is the version

Versioning

No CHANGELOGs. No `version` field – the manifest is validated to **omit** it. The git SHA is the version, so nobody has to remember to bump anything.

Changing one

Branch, edit the tracked file, open a PR. Same as code, because it is code that happens to be prose.

WHAT CI REJECTS ON EVERY PR

- A skill whose name and folder disagree
- Empty or missing descriptions
- A manifest carrying a version
- **User-specific absolute paths**

That last one is the number one way a skill that works for you breaks for everyone else.

Watch how Claude reads your skill

WHAT YOU SEE IN THE TRANSCRIPT

WHAT IT IS TELLING YOU

It explores in an order you did not expect

Your structure is not as intuitive as you thought

It fails to follow a reference

The link needs to be more explicit, or more prominent

It reads the same file every time

That content belongs in SKILL.md

It never opens a file at all

The file is unnecessary, or badly signposted

THIS MORNING, IN MY OWN ROUTINE

seo-diamond pointed at its own framework file. The path did not resolve, and the agent went and found it in the repo instead. Signal two, live, in a run that still looked successful.

Source: Anthropic's Agent Skills best practices. The transcript is the test suite you already have.

A routine is a habit that runs while you sleep

How it picks up skills

On Claude Code for web, tick the repo under **Select repositories**. Skills load straight from the repo – no install, no symlinks, nothing on your laptop.

THE GOTCHA THAT COST ME A WEEK

My best skills **ask questions**. An intake checkpoint, an outline to approve. Nobody is there at 6am, so the routine just stops.

```
# two of the section headings inside flamingo-content
## Intake: gather what's missing (MANDATORY - do this before anything else)
## After delivery (MANDATORY - always collect feedback via questionnaire)
```

An unattended run has to **bypass those checkpoints** and allow the `Skill` tool explicitly. Interactive by default is right for you and wrong for a routine.

Faster, and I could not share it

THE SAME 200 BLOG IMAGES

CLAUDE DESIGN

THESE SKILLS, IN CLAUDE CODE

4 images for one post

5 to 7 min

15 to 20 min

50 posts, 200 images

4 to 6 hours

12 to 17 hours

SO WHY DID I MOVE

You cannot export a design system out of Claude Design, or share it with the team, without an organisation subscription. The whole brand lived in **one account**. Mine.

WHAT I DID INSTEAD

Took the Figma files, turned them into CSS tokens, wrapped them in skills, and put those on GitHub. **Two to four times slower per image.** Twenty-six people can run it.

I gave up most of a working day per batch so that it stopped being
a thing only I could do.

What they can and cannot do

CAN, TODAY, OFFLINE

- 28 ad layouts at 1080x1080, with your copy
- Blog OG card and hero preview, 1200x640
- Inline blog diagrams and meme canvases
- OpenMSP deck, podcast cover, social image
- Branded .docx, .xlsx and HTML reports
- 164 clay objects to pull from

CANNOT

- **Invent a layout.** Templates are fixed. A structural change means copying the folder and editing the HTML
- **Shrink copy that does not fit.** It refuses, and tells you by how many pixels
- **Supply photography.** The system ships none, on purpose
- **Run from anywhere.** The renderers live in a sibling skill, so paths only resolve from the skill's own directory

The limits are the point. A template you cannot restyle by accident is a brand that survives being used by twenty people.

It tells you the budget before you write

```
$ make-ad.py --template ad-typographic --slots
```

```
Ad - all type    1080×1080
  headline      2 lines at 85px, 920px wide - about 18 characters
                  per line, 36 in total. The shipped copy is 24.
  subhead       3 lines at 85px, 920px wide - 54 in total.
  body          2 lines at 38px, 920px wide - 80 in total.
```

THEN EITHER

Say it in plain English – "Make me an ad from ad-typographic, headline: Eight modules. One dollar." The skill fills the slots.

OR DRIVE IT DIRECTLY

```
--set "headline=Eight modules. One dollar."
--set "subhead=Per device, per month."
--out ad.png
```

Write to the number. Azeret Mono is monospace, so **character count maps straight to line count** – there is no guessing whether it will fit.

One command, four seconds

```
$ make-og.py \  
  --title "A Skill Is a\nTrigger and a\nPointer" \  
  --cyan "Trigger" --pink "Pointer" \  
  --clay bear-trap --out og.png
```

og.png (223KB, 1200×640)

WHAT IT ASSEMBLED

The 60/40 split, the flamingo watermark, the clay object on its pink radial glow, the logo lockup, and the two accent words picked out of the headline.

WHAT I DID NOT DO

Open Figma. Open Claude Design. Pick a colour. Touch the network.
None of it was a design decision – the decisions are already in the files.

Every blog post ships two of these. That used to be the reason a post sat in draft.

Where it joins the automation

STAGE	WHAT RUNS IT	AUTOMATED?
Pick the task	ClickUp, in the routine	Yes, 07:00 weekdays
Research and brief	seo-diamond	Yes
Write the post	flamingo-content	Yes
Insert as draft	Supabase, in the routine	Yes
OG card and preview	blog-og-preview	Not yet - I run it
Inline diagrams	flamingo-blog-diagrams	Not yet - I run it

The writing is unattended. The pictures are two commands away from joining it.
That is the next thing I am wiring up, and it is a good first contribution if you want one.

BEFORE FRIDAY

Pick one thing you
corrected Claude on twice
this week.
Turn it into a five-line
skill.

STEP 1 ADD THE MARKETPLACE, ONCE

```
/plugin marketplace add flamingo-stack/claude-skills
```

STEP 2 INSTALL, EITHER SURFACE

```
/plugin install design@flamingo-stack
```

Or in the app: + beside the prompt box → **Plugins** → **Add plugin**.

TWO SOURCES

[Anthropic, Agent Skills best practices](#) · [Google, Prompting Essentials](#)

