TEACH CLAUDE NEW SKILLS

A practical guide to skills and MCP for UK professionals

BOUCH

bouch.dev | April 2026

CONTENTS

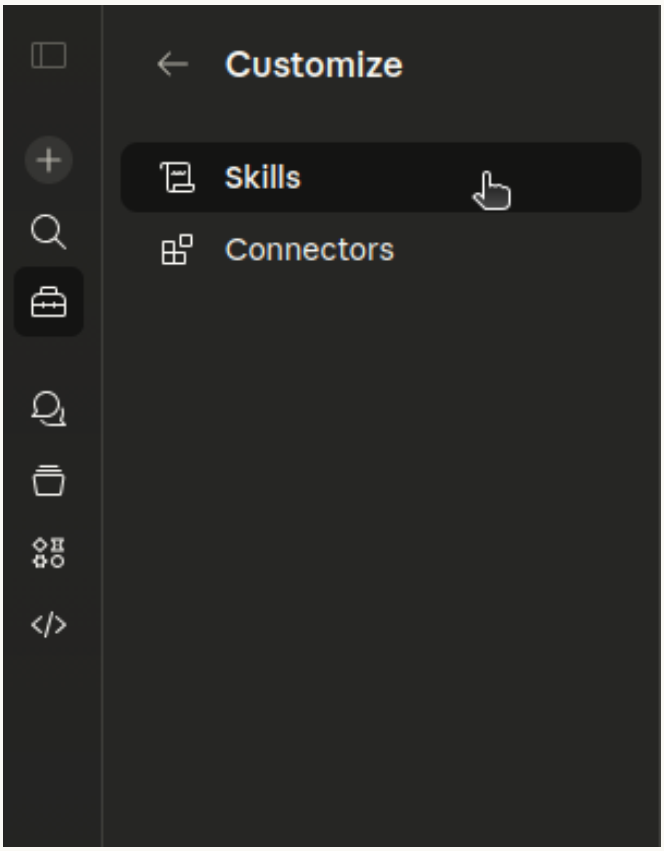
PART 1: START HERE

1. YOUR FIRST SKILL IN 2 MINUTES

Download the humaniser folder from this guide's bundled files, or grab it from bouch-skills on GitHub.

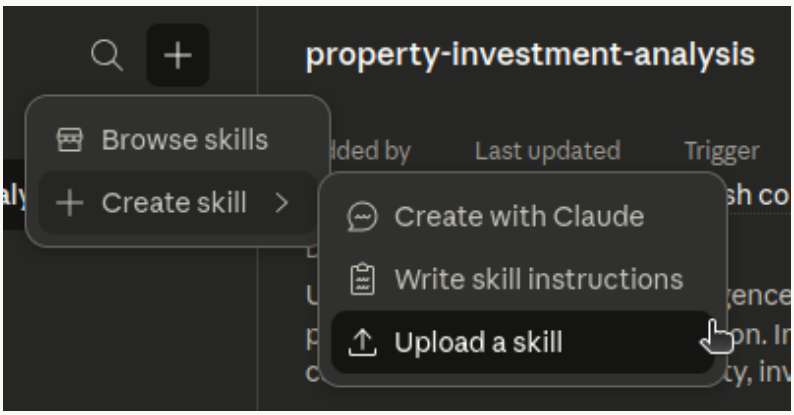
Install it in claude.ai:

1. Open claude.ai
2. Click your name (bottom-left) > **Customize**
3. Click **Skills** in the sidebar



Customize sidebar — click Skills

4. Click the **+** button > **Upload a skill**



Upload a skill from the + menu

5. Select the humaniser skill
6. Toggle the skill on

Try it:

Paste this into Claude:

> *Rewrite this to sound less AI-generated: "We're excited to leverage cutting-edge solutions to optimize your workflow and deliver exceptional value at scale."*

Claude loads the humaniser skill and strips the corporate filler. You taught Claude a new capability. It now knows how to spot and remove AI writing patterns from business text. The skill sits quietly until Claude sees a request that matches, then it activates automatically.

2. HOW SKILLS STAY LIGHTWEIGHT

You might wonder what happens when you install 20, 30, 50 skills. The answer: almost nothing.

Claude uses a three-level loading system called **progressive disclosure**:

Level 1: Metadata (always loaded). When you install a skill, Claude reads just the name and description into its working memory. Roughly 100 tokens per skill. Fifty installed skills cost about 5,000 tokens of overhead. Trivial.

Level 2: Instructions (loaded when triggered). When you make a request that matches a skill's description, Claude reads the full SKILL.md body into context. Anthropic recommends keeping this under 5,000 tokens. Before that trigger, the file sits on disk consuming nothing.

Level 3: Resources (loaded on demand). Skills can bundle extra files in scripts/, references/, or assets/ directories. Claude only reads these when the SKILL.md instructions reference them. A stamp duty rate table, a validation script, an output template. None of it enters context unless the workflow needs

it.

The result: 50 installed skills cost ~5,000 tokens. One triggered skill costs under 5,000 more. Everything else stays dormant.

Skills are capacity, not overhead.

PART 2: HOW SKILLS WORK

3. ANATOMY OF A SKILL

Every skill is a folder containing one required file: SKILL.md. That file has two parts.

The frontmatter (YAML, between -- markers):

```
---
name: workflow-auditor
description: |
  Analyse a business workflow to find where time is actually lost and
  recommend specific improvements. Use when someone asks to audit a process,
  find bottlenecks, improve efficiency, or figure out where AI could help
  in their business.
allowed-tools:
  - Read
  - Grep
  - Glob
  - AskUserQuestion
---
```

The body (Markdown, after the second --):

Step-by-step instructions for how Claude should handle the task. Workflow steps, output format, rules, examples.

Required fields:

Field	Rules
name	Lowercase, hyphenated (kebab-case). Max 64 characters. Must be unique. Cannot contain "claude" or "anthropic".
description	What it does AND when to use it. Max 1,024 characters. Must include trigger phrases.

allowed-tools is optional but recommended. It lists which tools the skill can use: built-in tools like Read, Write, Edit, or MCP tools for external data.

The description is what makes or breaks a skill. Section 12 covers how to write one that works.

The four parts of a good instruction body:

Every bundled skill in this guide follows the same structure. Open any of them and you will see:

1. **Opening role statement.** First line after the frontmatter. "You are a writing editor who strips AI patterns from business text." This sets Claude's scope

immediately. Without it, Claude defaults to being a general assistant.

2. **Numbered workflow steps.** "Step 1: Map the Process. Step 2: Identify the Five Types of Time." Explicit steps with explicit data handoffs between them. "Extract the median price from Step 2. Pass it as purchase_price to Step 3."
3. **Output template.** What the final result looks like. A table, a heading structure, a report format. If you do not specify this, Claude will invent its own format every time, and it will be different every time.
4. **Negative rules.** "Do NOT add actions that were not in the notes." "This skill does NOT provide mortgage advice." These prevent false activation and stop Claude from wandering outside the skill's scope.

4. THREE TYPES OF SKILLS

Type 1: Standalone (no external dependencies)

Uses only Claude's built-in tools: Read, Write, Edit, Grep, Glob, AskUserQuestion.

Examples from this guide's bundled skills:

- **humaniser** — strips AI writing patterns. Uses Read, Write, Edit, Grep, Glob.
- **workflow-auditor** — finds the real bottleneck in a business process. Uses AskUserQuestion for multi-turn conversation.
- **meeting-actions** — turns raw meeting notes into structured actions with owners and deadlines.

These are the simplest to build. They work everywhere Claude works. They encode knowledge and process, not data access.

Some standalone skills handle multiple sub-tasks. When they do, a routing table at the top prevents Claude from picking the wrong approach. Anthropic's docx skill opens with a three-row table: Read content → use pandoc. Create new → use docx-js. Edit existing → unpack XML, edit, repack. Claude reads the table, picks a path, and follows only the relevant instructions. If your skill has distinct modes (read vs create vs edit, or analyse vs generate vs compare), put the decision tree first.

Type 2: MCP-Connected (uses external data)

Uses MCP servers (called "Connectors" on claude.ai) to access real-world data and APIs.

Examples:

- **property-report** — pulls comparable sales, EPC ratings, rental yields, and stamp duty from the Property MCP server. One request produces a complete

investment analysis.

- **bailii-case-law** — searches UK court decisions through the Legal MCP server. Finds relevant case law for a legal question.

These require an MCP server to be connected. The skill teaches Claude the workflow; the MCP server provides the data. More on this in Part 3.

Type 3: Multi-MCP (coordinates multiple services)

Orchestrates tools from several MCP servers in a single workflow.

Concept example: a **deal-screener** skill that checks a property address against three services in sequence:

1. Property MCP: pull comps, yields, EPC
2. UK Due Diligence MCP: check Companies House for freeholder, planning applications
3. UK Legal MCP: search for relevant property law issues

Each MCP provides raw data. The skill knows the order, the logic, and what to do with the results. Section 9 shows the full implementation of this deal-screener skill.

Start standalone. Add MCP when you need external data. Most useful skills are Type 1 or Type 2.

PART 3: CONNECT TO THE REAL WORLD

5. CONNECTING MCP ON CLAUDE.AI

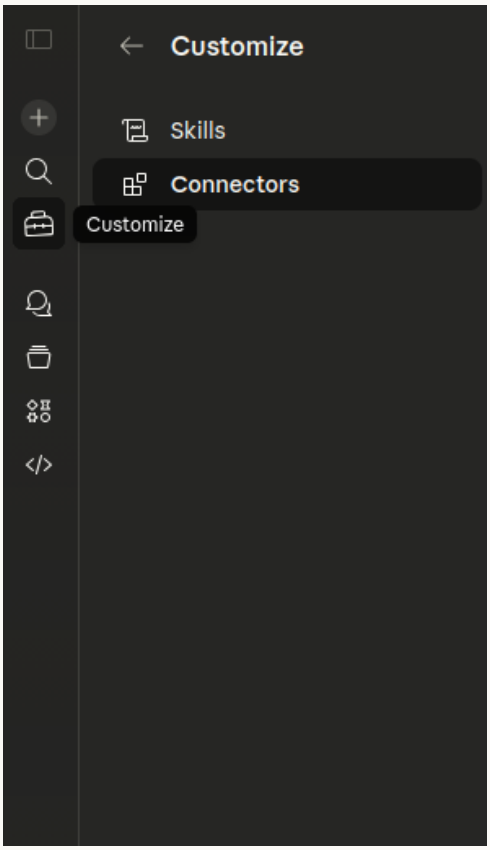
An MCP server gives Claude access to external tools and data. A skill teaches Claude how to use those tools in the right order for a specific task.

Without a skill: you ask Claude to "analyse this property investment" and then spend 10 minutes telling it which data to pull, what calculations to run, and how to structure the output.

With a skill: you ask the same question and get a complete report. The skill already knows the workflow.

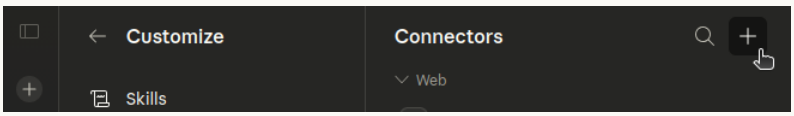
On claude.ai, MCP servers are called **Connectors**. Connecting one takes 30 seconds.

1. Click your name (bottom-left) > **Customize**
2. Click **Connectors** in the sidebar



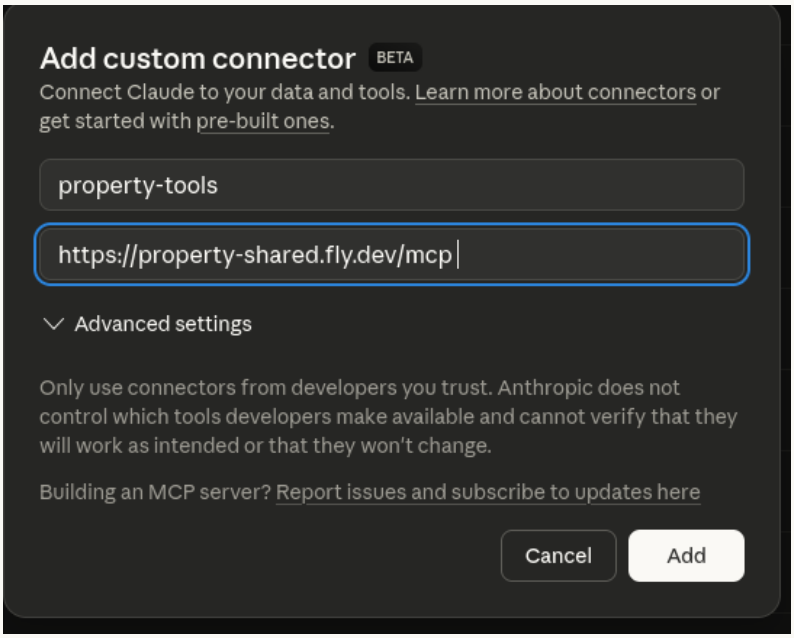
Customize sidebar — click Connectors

3. Click the **+** button



Click the + button to add a connector

4. Enter a **Name** and **Remote MCP server URL**
5. Click **Add**



Add custom connector dialog with name and URL fields

Example: connecting the Property MCP server

Field	Value
Name	property
URL	https://property-shared.fly.dev/mcp

Once connected, Claude can use all the tools that server exposes: comparable sales, EPC lookups, rental analysis, stamp duty calculations, yield estimates.

Example: connecting UK Legal Tools

Field	Value
Name	uk-legal-tools
URL	*(your server URL)*

This gives Claude access to legislation lookup, BAILII case law search, and other legal research tools.

Test it: after connecting the Property MCP, ask Claude "Fetch comparable sales for SW1A 1AA". If you get back real Land Registry data, the connection works.

6. CLAUDE CODE: FULL CONTROL (FOR TECHNICAL USERS)

If you use Claude Code (Anthropic's CLI tool), you get finer control over MCP connections and tool access.

Connecting an MCP server:

```
claude mcp add --transport http property https://property-shared.fly.dev/mcp
```

Tool naming convention:

In Claude Code, each MCP tool gets a namespaced identifier:

```
mcp__[server_name]__[tool_name]
```

For example: `mcp__property__property_comps`, `mcp__property__stamp_duty`

The server name comes from your config (the name you chose when adding the server), not the URL.

Per-skill tool scoping:

In Claude Code, you can restrict which tools a skill uses via the `allowed-tools` field:

```
allowed-tools:
  - mcp__property__property_comps
  - mcp__property__property_epc
  - mcp__property__rental_analysis
  - mcp__property__stamp_duty
```

This is a Claude Code feature. On claude.ai and Claude Desktop, all tools from a connected server are available to all skills automatically. There is no per-tool scoping.

Installing skills in Claude Code:

No ZIP, no upload. Create a folder in `~/.claude/skills/` (personal) or `.claude/skills/` (project-level, shareable via Git).

PART 4: BUILD YOUR OWN

7. TUTORIAL: BUILD A MEETING ACTIONS SKILL

Let's build a skill from scratch. We will create meeting-actions, which turns raw meeting notes into structured actions with owners and deadlines.

Step 1: Create the folder and file

```
mkdir meeting-actions
```

Create a file called SKILL.md inside it. The filename must be exactly SKILL.md (case-sensitive).

Step 2: Write the frontmatter

```
---
name: meeting-actions
description: |
  Turn meeting notes, transcripts, or voice memos into structured actions
  with owners and deadlines. Use when someone pastes meeting notes, shares
  a transcript, says "what came out of that meeting", or asks to extract
  actions from any unstructured text about a conversation or discussion.
allowed-tools:
  - Read
  - Write
  - Edit
---
```

Notice the description: it says what it does ("turn meeting notes into structured actions") and when to use it ("Use when someone pastes meeting notes, shares a transcript..."). Multiple trigger phrases cover different ways people might ask.

Step 3: Write the instructions

After the closing --, write the body in Markdown:

```
# Meeting Notes to Actions

You take raw meeting notes, transcripts, or voice memo text and produce
a clean, structured output: decisions made, actions with owners and
deadlines, and open questions.

## Workflow

### Step 1: Read and Parse

Read the full text. Identify:
- Who was in the meeting (names mentioned)
- Topics discussed
- Decisions made (anything agreed or confirmed)
- Actions (anything someone said they would do)
- Open questions (anything unresolved)
- Deadlines (any dates or timeframes mentioned)

### Step 2: Extract Actions

For each action, determine:
- What needs to happen (specific, not vague)
- Who is responsible (use the name from the notes, or "TBC" if unclear)
- By when (use the deadline from the notes, or "No deadline set")

### Step 3: Structure the Output

## Meeting Summary
**Date:** [if mentioned]
**Attendees:** [names from the notes]
**Topic:** [1-sentence summary]

### Decisions
1. [Decision, stated clearly]

### Actions
| # | Action | Owner | Deadline |
|---|--------|-------|----------|
| 1 | [Specific action] | [Name] | [Date] |

### Open Questions
- [Thing raised but not resolved]

## Output Rules

- Actions must be specific. "Send the proposal to James" not "Follow up."
- Do not add actions that were not in the notes.
- If the notes are too vague, say so and ask the user to clarify.
- British English throughout.
```

Why each section matters:

- **The opening line** ("You take raw meeting notes...") sets Claude's role and scope. Without this, Claude treats the skill as generic advice rather than a specific task.
- **Output Rules** prevent Claude from waffling. Without explicit constraints, Claude adds preambles, summaries of what it's about to do, and unsolicited advice.

- **Negative rules** ("Do not add actions that were not in the notes") stop Claude from being "helpful" in ways that corrupt the output. Claude will invent plausible-sounding actions if you do not tell it not to.

The humaniser skill in the bundled files adds a "What Good Looks Like" section. The workflow-auditor adds "What This Skill Does NOT Do". Both are worth studying as patterns.

Building verification into the skill itself. The meeting-actions skill above includes "Do not add actions that were not in the notes." That is a constraint. But you can go further and add a verification step directly in the workflow: "After generating the output, read it back against the original notes. Check that every action has an owner, every deadline is a specific date, and no actions appear that were not in the notes. If anything fails, fix it before presenting the output." Anthropic's pptx skill takes this further. It converts slides to images, spawns a fresh subagent with no prior context, and has it inspect for problems. The principle is the same: assume there are errors, build in a check.

Step 4: Upload and test

Upload the meeting-actions folder via Customize > Skills > + > Upload a skill.

Test it:

> *"I met with Sarah and John about Q2 roadmap. We decided to push mobile app to May and prioritise the API rewrite. John will update the timeline by Friday. Sarah is checking whether we need a new staging environment."*

Expected output: a structured summary with two decisions, two actions (John: update timeline by Friday; Sarah: check staging environment), and the topic clearly identified.

Step 5: Test properly

Three checks before you share a skill with anyone:

Trigger test. Try five different phrasings of the same request. "Extract actions from these notes." "What came out of that meeting?" "Turn this transcript into a task list." "Summarise this call." "What did we agree?" Aim for the skill to activate at least 9 out of 10 times. Then try three unrelated requests ("help me write an email", "what's the weather", "review this code"). The skill should NOT trigger on any of them.

Output consistency. Run the same request three times. The structure should be identical every time: same sections, same table format, same level of detail. The content will vary (different wording, different emphasis) but the shape should not. If the output format changes between runs, your template is not specific enough.

Before/after comparison. Do the task once without the skill. Just prompt Claude directly with "turn these meeting notes into actions." Then do it with the skill enabled. Compare: how many messages did it take? Did you need to correct anything? Was the output better structured? If the skill does not clearly improve at least one of these, it is not earning its keep. Go back to the instructions and tighten them, or consider whether this task needs a skill at all (see Section 11).

Common fixes if something is off:

- **Doesn't trigger:** add more trigger phrases to the description
- **Output too verbose:** add "Be concise" to your output rules
- **Missing sections:** make them explicit in the template
- **Triggers on the wrong things:** add "Do NOT use for..." to the description
- **Output format inconsistent:** add a stricter output template with exact headings

You just built a skill. No code. No API. A markdown file in a folder.

8. PATTERNS THAT WORK

From building 29 skills across property, legal, and project management domains, six patterns appear consistently. Understanding them will save you from reinventing wheels.

Pattern 1: Sequential Workflow

Step 1 feeds into Step 2 feeds into Step 3. Each step has clear inputs, outputs, and validation. The skill encodes a dependency chain the user does not need to know about.

UK example: the property-report skill follows this chain:

1. Pull comparable sales (gets the median price)
2. Pass median price into rental analysis (calculates yield)
3. Pass purchase price into stamp duty (calculates SDLT)
4. Pull current Rightmove listings (market context)
5. Synthesise everything into a structured report

The key design detail is **explicit data handoffs** between steps. Here is what this looks like in the actual SKILL.md:

```
### Step 2: Pull Comps First

Start with `property_comps` to get comparable sales. This gives you
the **median price** which feeds into subsequent calls.

Extract and note:
- Median price (this becomes the purchase_price for yield calculations)
- Median price per sqft (from EPC enrichment)

### Step 4: Pull Rental Data

**Pass `purchase_price`** to rental_analysis using the median comp
price from Step 2. This populates `gross_yield_pct` in the response.
```

Without that explicit "median price from Step 2 becomes purchase_price in Step 4" instruction, Claude will sometimes make up a number or ask the user for a price they already provided. The handoff language is what makes sequential skills reliable.

Pattern 2: Iterative Refinement

Draft, validate against source, improve, present. The skill checks its own output before showing it to the user.

UK example: a policy-briefing skill searches Hansard debates, drafts a summary of parliamentary reception, then cross-checks each claim against the actual debate text. Without the validation step, Claude produces confident-sounding claims that are not in the source. One test run stated "the Minister expressed strong support" when the actual debate showed lukewarm acknowledgement. The validation step caught it.

The fix: write the validation step as a mandatory gate with the instruction "Do NOT present the briefing until this step is complete." Without that gate, Claude will sometimes skip validation when the draft looks reasonable. The gate forces the check. For each claim, the skill requires Claude to confirm it is directly supported by the source, reword it if partially supported, or remove it entirely.

Pattern 3: Domain Intelligence

The skill embeds expert knowledge that most people would not think to apply. This is where skills earn their keep: encoding the non-obvious thing that an experienced practitioner knows.

UK example: the property-report skill separates student lets from professional lets before calculating rental yields. If you mix weekly student prices (£170/week) with monthly professional lets (£950/month) into one average, the number is meaningless. The skill knows this and handles it automatically:

```
**Segment student vs professional lets.** Look for signals:
- Weekly pricing = almost certainly student
- "students" in listing text
- Multiple rooms advertised separately

If both are present, report them separately. Exclude student lets
from yield calculations unless the user specifically asks.
```

Second example: a due diligence skill checks The Gazette for insolvency notices *before* pulling Companies House officer data. Why? If the company is in an insolvency proceeding, the officer list is misleading. The appointed insolvency practitioner is running the show, not the listed directors. Checking Gazette first changes how you interpret everything that follows. Without that ordering, the skill would present a clean officer list for a company that is actually being wound up.

This kind of knowledge is invisible to someone who has not done the work before. That is exactly what skills should encode.

Pattern 4: Guardrail Skills

Some skills exist mainly to stop Claude from doing something wrong. The humaniser is this pattern. Its job is to remove AI writing patterns, not add anything. The "What This Skill Does NOT Do" section is a guardrail. Negative rules ("Do NOT add actions that were not in the notes") are guardrails.

This pattern is underappreciated. Consider building a guardrail skill when:

- Claude consistently makes the same mistake in a domain (e.g. mixing student and professional rents)
- The cost of getting it wrong is high (legal research returning advice instead of case law)
- Your team needs consistency more than creativity (report formatting, compliance checks)

A guardrail skill can be short. Sometimes 20 lines of "do not do X" rules are more valuable than 200 lines of workflow steps.

Pattern 5: Script Toolkit

Some skills need complex processing that does not belong in natural language instructions. The solution: bundle helper scripts and teach Claude the interface, not the internals. The skill says "Run `python scripts/convert.py input.pdf`". Claude calls it as a black box.

Anthropic's pptx skill bundles five scripts (thumbnail, unpack, pack, clean, add_slide) that handle XML manipulation. The SKILL.md never explains how they work. It just says when to call each one. Anthropic's webapp-testing skill goes further: "Always run scripts with `-help` first. Do NOT read the source." This prevents the script from polluting Claude's context window.

If your skill needs deterministic processing (format conversion, data validation, calculations), wrap it in a script. Teach Claude the inputs, outputs, and when to use it. Let the script handle the complexity.

Pattern 6: Anti-Pattern Inline Examples

When Claude makes the same mistake repeatedly, showing the wrong way is more effective than just showing the right way.

Anthropic's xlsx skill does this for spreadsheet formulas:

```
❌ WRONG: calculating in Python and hardcoding the result:
   total = df['Sales'].sum()
   sheet['B10'] = total

✅ CORRECT: letting Excel calculate:
   sheet['B10'] = '=SUM(B2:B9)'
```

The side-by-side format makes the mistake unmissable. Use this when you have tested a skill and found Claude falling into the same trap. Add the ❌/✅ pair directly in the workflow step where the mistake occurs, not in a separate "common mistakes" section at the end.

When to add extra files:

Directory	When to use	Example
scripts/	Validation that must be deterministic (code, not language)	Postcode format checker
references/	Detailed docs too long for SKILL.md	Stamp duty rate tables
assets/	Output templates, schemas	Report template

Keep SKILL.md under 5,000 words. Move detailed reference material to references/ and link to it from the instructions.

9. MULTI-MCP IN PRACTICE

Section 4 introduced Type 3 skills: those that coordinate multiple MCP servers. Here is what one actually looks like.

The scenario: a deal-screener skill that checks a property investment against three data sources in sequence. Not a concept sketch. A real SKILL.md you could upload today.

Frontmatter:

```
---
name: deal-screener
description: |
  Screen a UK property deal by pulling comps, checking the freeholder
  on Companies House, and searching for relevant legal issues. Use when
  someone asks to "screen this deal", "check this property", or "run
  due diligence on this address". Requires Property, UK Due Diligence,
  and UK Legal Tools MCP servers.
allowed-tools:
  - mcp__property__property_comps
  - mcp__property__property_epc
  - mcp__property__rental_analysis
  - mcp__property__stamp_duty
  - mcp__due_diligence__company_search
  - mcp__due_diligence__company_profile
  - mcp__due_diligence__gazette_insolvency
  - mcp__legal__case_law_search
  - mcp__legal__legislation_search
---
```

On claude.ai, skip the `allowed-tools` field. All connected servers expose all their tools automatically.

The workflow (abbreviated):

```
### Step 1: Property Data (Property MCP)

Pull comps and EPC for the address. Extract:
- Median price, price per sqft, EPC rating
- Whether it is leasehold or freehold (from EPC data)

If leasehold, note the freeholder name for Step 2.

### Step 2: Freeholder Check (Due Diligence MCP)

If leasehold: search Companies House for the freeholder company
using the name from Step 1.

Check:
- Is the company active? (company_profile → company_status)
- Any insolvency notices? (gazette_insolvency)
- Who are the officers? (company_officers)

If the freeholder is dissolved or in insolvency, flag this
prominently. It affects service charges, lease extensions,
and the ability to sell.

### Step 3: Legal Context (Legal MCP)

Search for recent case law related to the property type and area.
For leasehold flats, search for:
- "service charge" + "unreasonable" (Landlord and Tenant Act 1985)
- "lease extension" + building name if known

For any property near major development, search planning law.

### Step 4: Synthesise

Present a one-page deal screen:
- Property fundamentals (price, yield, condition)
- Freeholder status (clean / flags)
- Legal context (relevant cases, if any)
- Go / No-go / Investigate further
```

The design principle: each MCP provides raw data. The skill provides the logic: knowing that a dissolved freeholder is a red flag, knowing which legal searches are relevant to leasehold flats, knowing to check Gazette before trusting officer data.

Handling partial failure:

MCP servers go down. Fly.io instances sleep. Add this to the workflow:

```
If any MCP server is unavailable or times out after one retry:
- Note which data source is missing in the output
- Continue with the remaining servers
- Mark the deal screen as "Partial — [source] unavailable"

A partial screen with two data sources is still more useful than
no screen at all.
```

This is important because without it, Claude will apologise and stop the moment one server fails. The explicit instruction to continue with partial data changes

Claude's behaviour completely.

Do not attempt rollbacks. If Step 3 fails after Steps 1 and 2 succeeded, do not try to undo the earlier steps. Present what you have: "Property data and freeholder check completed. Legal search failed (server timeout). Here is what we have so far. Run the legal check separately when the server is back." Build this language into the skill itself so Claude handles failures gracefully without being told each time.

10. MISTAKES I HAVE MADE SO YOU DO NOT HAVE TO

Nine real mistakes from building 29 skills. The first five are about writing SKILL.md. The last four are operational. Things that only break in production.

1. Description too broad

A skill described as "help with property tasks" triggered on everything from mortgage questions to interior design. Claude could not tell when to use it versus three other property skills. Fix: descriptions need to be specific about what the skill does AND what it does not do.

2. SKILL.md over 8,000 words

The first version of the property-report skill was enormous. Claude would follow steps 1-5 carefully, then forget the output formatting rules at the end. The instructions fell off the back of Claude's working memory. Fix: keep the body under 5,000 words. Move reference material (rate tables, data dictionaries) to a `references/` folder.

3. Hardcoded Claude Code tool names in a skill meant for claude.ai

The body referenced `mcp__property__property_comps` as a tool name. This is a Claude Code convention. On claude.ai, tools have plain names (`property_comps`). The skill still worked because Claude matched the intent, but the instructions were confusing and sometimes Claude tried to call the wrong tool name. Fix: if your skill needs to work on claude.ai, use plain tool names in the body and only use `mcp__` names in the `allowed-tools` frontmatter field (which is ignored on claude.ai anyway).

4. No "What This Skill Does NOT Do" section

A legal research skill kept activating when people asked for legal advice. It was designed to find case law, not give opinions. Without explicit boundaries, Claude treated every legal question as an invitation to use the skill. Fix: add a "What This Skill Does NOT Do" section. Be specific: "Does NOT provide legal advice. Does NOT interpret case law for a specific situation."

5. Testing only with the happy path

The skill worked perfectly for "analyse this property". It failed when someone said "what's this house worth?" or "is this a good investment?". The description only contained one trigger phrase. Fix: test with at least five different phrasings of the same request before publishing. Real people do not use your exact words.

6. Two skills with overlapping descriptions

A "property analysis" skill and an "investment summary" skill both matched "analyse this property". Claude sometimes picked the wrong one. Sometimes it loaded both and merged their instructions, producing a Frankenstein output. The descriptions were different but the trigger phrases overlapped.

Fix: if two skills could match the same request, they need to carve out non-overlapping territory. Either merge them into one skill, or make the descriptions mutually exclusive: "Use for quick valuations under 2 minutes" vs "Use for full investment reports with rental yield and stamp duty." The trigger phrases must not collide.

7. MCP server cold start

The Property MCP runs on Fly.io. After 15 minutes of inactivity, the instance sleeps. The first tool call after a sleep takes 8-10 seconds and sometimes times out. Claude interprets the timeout as "this tool is broken", apologises, and stops the workflow. The user thinks the skill is broken.

Fix: add a retry instruction to the workflow. "If the first MCP call fails with a timeout, wait 5 seconds and retry once. Fly.io instances wake on the first request. The retry will succeed." This is a one-line addition that prevents the most common failure mode for any skill using a hosted MCP server.

8. Too many tools in allowed-tools

The first property-report skill listed every tool the Property MCP server offers: comps, EPC, rental, stamp duty, yield, Rightmove search, Rightmove listing, planning search, company search, block analysis. Fourteen tools in total. Claude would call tools that were not relevant to the request, burning tokens on planning searches when the user just wanted a quick valuation. The output was bloated and slow.

Fix: restrict `allowed-tools` to only what the workflow actually uses. The property-report workflow needs six tools, not fourteen. Fewer tools means Claude stays focused. If you later need a tool you excluded, add it, but start lean.

9. No exit conditions on multi-turn skills

I built a workflow-auditor skill that kept asking questions forever. It would dig deeper and deeper into the process, asking about edge cases and exceptions, until the user gave up and said "just give me the answer." The skill had no concept of "enough information gathered."

Fix: add explicit stage transitions. "When the user has answered at least 5 questions and you understand the workflow end-to-end, move to the recommendation step." Anthropic's doc-coauthoring skill models this well. It has three named stages (Context Gathering, Refinement, Reader Testing), each with a specific exit condition. Without these gates, multi-turn skills become interrogations.

11. WHEN NOT TO WRITE A SKILL

Not everything should be a skill. Here are five signs you should just prompt Claude directly.

1. You will do this once

Skills pay off through repetition. If you need a one-off analysis, a custom email, or a single report in an unusual format, just ask Claude. Writing a skill for a one-time task is like writing a recipe for a meal you will never cook again. The break-even point is roughly five uses, or one use shared across a team.

2. The task needs constant human judgment

Negotiation strategy. Creative direction. Choosing which of three valid approaches fits this specific client. If the "right answer" changes with every input and depends on context that cannot be written down, a skill adds friction. Skills encode repeatable processes. If the process is not repeatable, the skill will fight you.

3. Claude already does it well without help

Summarisation. Translation. Explaining code. Basic data analysis. If Claude handles the task reliably without a skill, you do not need one. Skills should encode YOUR specific process or domain knowledge: the non-obvious steps, the edge cases, the "we tried it the other way and it did not work" lessons. If Claude's default behaviour is good enough, do not add a skill just to have one.

4. The workflow is still changing

If you are changing the process every week, adding steps, removing steps, rethinking the output format, encoding it in a skill locks in something premature. Get the workflow right by doing it manually five times. Notice which steps you repeat, which you skip, where you add the same instructions. When the

process stabilises, that is when you write the skill.

5. The "just ask Claude" test

Before writing any skill, try the task by prompting Claude directly. If the result is 80% of what you want, the skill only captures the last 20%. Is that 20% worth maintaining a file for? Sometimes yes: when consistency matters, when a team needs the same output format, when the domain knowledge is hard to remember. Sometimes no: when you are the only user and you can type the extra instructions in 30 seconds.

A skill is worth writing when the alternative is explaining the same workflow to Claude for the fifth time.

6. The tool-first trap

Some skills start from a problem: "My team wastes 2 hours on meeting follow-ups." Others start from a tool: "I have this Property MCP server. What useful skills could I build on top of it?" Problem-first skills tend to write themselves because the trigger phrases come naturally from how people actually ask for help. Tool-first skills often fail the "just ask Claude" test because you are working backwards from capability to use case.

If you are staring at an MCP server wondering what skill to build, start by asking: what question would someone actually ask that this server could answer? If you cannot think of one that passes the tests above, the server might not need a skill yet.

12. THE DESCRIPTION CHECKLIST

The description is your activation UI. Use this formula:

```
[What it does] + [Specific output] + Use when [trigger 1], [trigger 2], [trigger 3] +
[Dependencies if needed]
```

Dissecting a working description:

```
description: |
  Analyse a business workflow to find where time is actually lost and
  recommend specific improvements. Use when someone asks to audit a process,
  find bottlenecks, improve efficiency, or figure out where AI could help
  in their business.
```

- "Analyse a business workflow to find where time is actually lost" — specific verb, clear domain, tangible output
- "recommend specific improvements" — outcome stated
- "Use when someone asks to..." — explicit activation trigger

- "audit a process, find bottlenecks, improve efficiency" — multiple trigger phrases covering different ways people ask

Before uploading any skill, check:

- [ ] Description says WHAT it does (verb + noun)
- [ ] Description says WHEN to use it ("Use when...")
- [ ] At least three trigger phrases covering different ways people ask
- [ ] Dependencies stated if MCP is required
- [ ] Under 1,024 characters

Test method:

Ask Claude: "When would you use the [skill-name] skill?" If Claude answers clearly, your description works. Try five different phrasings of the same request. Aim for the skill to activate at least 9 out of 10 times.

Common mistakes:

Mistake	Fix
Too vague ("helps with projects")	Add specific verbs and outputs
No trigger phrases	Add "Use when..." with 3+ phrases
Too narrow ("calculates gross yield")	Use the language real people use ("is this a good investment?")
Missing dependencies	Add "Requires [X] MCP server"

PART 5: GOING FURTHER

13. WHERE TO GO FROM HERE

Skills work across all Claude surfaces. The SKILL.md format is the same everywhere. Installation differs.

Surface	How to install	MCP tool scoping?	Sharing
claude.ai	Upload via Customize > Skills	No (all tools available)	Individual only
Claude Desktop	Upload via settings	No	Individual only
Claude Code	Folder in `~/.claude/skills/` or `.claude/skills/`	Yes (`allowed-tools`)	Via Git or plugins
Claude API	Upload via `/v1/skills` endpoint	Yes	Organisation-wide

Skills uploaded to one surface do not sync to others. You manage them separately. Start with uploads on claude.ai. Move to folders in Claude Code when you are ready. The SKILL.md file is the same; only the location changes.

Use what you have:

The eight skills bundled with this guide are yours. They are yours to modify, rename, or use as templates. The four original skills (humaniser, workflow-auditor, meeting-actions, property-report) demonstrate the core patterns. The four document-format skills (pdf-handler, pptx-handler, xlsx-handler, docx-handler) demonstrate the Decision Tree Router, Script Toolkit, QA Verification, and Anti-Pattern patterns from Section 8. The `meeting-actions` skill took 15 minutes to build. Most standalone skills take less than 30 minutes once you understand the pattern.

Explore more skills:

- anthropics/skills — Anthropic's official skills repository (16 skills). The document-format skills (pdf, pptx, xlsx, docx) are particularly worth studying — they demonstrate the Script Toolkit pattern (bundling helper scripts as black boxes) and QA verification loops (spawning subagents to inspect output with fresh eyes). The `skill-creator` and `doc-coauthoring` skills show staged workflows with explicit exit conditions.
- More BOUCH skills and free MCP servers are listed at the end of this section. See "Free MCP servers" and "Stay connected" below.

Use the skill-creator:

Claude has a built-in skill called skill-creator that helps you build new skills. Ask Claude "Help me build a skill using skill-creator" and it will walk you through drafting, testing with sample prompts, evaluating results, and iterating. Worth using if you are building more than a couple of skills.

For teams: organisation-wide deployment:

Since December 2025, workspace admins can deploy skills organisation-wide via the Claude API. This means one person writes the skill and everyone on the team gets it. If you are a team lead, this is the path from "I built a useful skill" to "my whole team uses it."

Composability: making skills play nicely together:

When you have 10+ skills installed, they need to coexist without colliding. Two skills with overlapping descriptions will confuse Claude (see Mistake 6 in Section 10). Test that new skills do not accidentally steal triggers from existing ones. Try your standard requests and check that the right skill activates each time. If two skills fight over the same request, make their descriptions explicitly non-overlapping: "Use for quick valuations" vs "Use for full investment reports."

Skills are just markdown files that encode how you already work. MCP connects them to real data. That is the whole thing.

Free MCP servers: connect these today:

All of these are live, free, and maintained. Connect them as custom connectors in claude.ai (Section 5) and start asking questions immediately.

Server	URL	What it does
UK Property	https://property-shared.fly.dev/mcp	Comparable sales, EPC data, yield calculations, Rightmove listings
UK Legal Tools	https://uk-legal-mcp.fly.dev/mcp	Legislation, case law, Hansard, parliamentary bills, HMRC guidance
UK Due Diligence	https://uk-due-diligence-mcp.fly.dev/mcp	Companies House, Charity Commission, Land Registry, Gazette, VAT
P6 Schedule (pyp6xer)	https://pyp6xer-mcp.fly.dev/mcp	Primavera P6 XER analysis, critical path, earned value, health checks
PineScript	https://pinescript-mcp.fly.dev/mcp	TradingView Pine Script v6 documentation and linting

New tools and servers are added regularly. Star or follow the repos below to get notified.

Stay connected:

- bouch.dev/tools — full catalogue of free skills and MCP servers
- bouch-skills on GitHub — standalone skills for UK business use
- bouch-mcp-skills on GitHub — MCP-connected skills for property, legal, and project management
- LinkedIn — updates, new tools, and the occasional useful post
- GitHub: paulieb89 — follow for new repos and releases

Want help building skills for your team?

This guide gives you everything you need to build your own. If you would rather have someone do it with you, or for you, that is what BOUCH does.

- Free self-service audit — seven questions, five minutes, a clear picture of where you are with AI
- In-person workflow audit — £349, half a day on-site, leave with a prioritised action plan
- Remote workflow audit — £249, same depth, done over video call

No obligation beyond the audit. If it makes sense to keep going, we talk about training or build work. If not, you still walk away with a plan.

paul@bouch.dev | 07711 822 499

APPENDIX A: CHECKLIST

Use this before uploading any skill.

Frontmatter:

- [ ] `name` is lowercase, hyphenated, unique
- [ ] `name` does not contain "claude" or "anthropic"
- [ ] `description` includes what, when, and trigger phrases
- [ ] `description` is under 1,024 characters
- [ ] `allowed-tools` lists the tools the skill needs (or is omitted)
- [ ] No XML angle brackets (< >) in frontmatter
- [ ] File starts with `--` at line 1, no leading spaces

Body:

- [ ] Opening role statement sets scope
- [ ] Clear numbered workflow steps
- [ ] Output template or format specified
- [ ] Negative rules ("Do NOT..." / "What This Skill Does NOT Do")
- [ ] British English throughout

Testing:

- [ ] Activates on at least 5 different phrasings (aim for 90%+)
- [ ] Does not trigger on unrelated requests
- [ ] Output quality is consistent across multiple runs
- [ ] MCP connections work (if applicable)

APPENDIX B: UK MCP SERVERS REFERENCE

These are publicly accessible MCP servers built for UK-specific domains. Connect them as Connectors on claude.ai (Customize > Connectors > +) or via `claude mcp add` in Claude Code.

PROPERTY MCP (PROPERTY-SHARED)

URL	https://property-shared.fly.dev/mcp
Docs	https://property-shared.fly.dev/docs
What it does	UK property data: Land Registry comparables, EPC certificates, Rightmove listings, rental analysis, stamp duty, yield calculations

Example tools:

- `property_comps` — comparable sales with EPC enrichment for any UK postcode
- `property_epc` — energy performance certificate data for a specific address
- `rental_analysis` — rental market aggregates with yield calculation
- `stamp_duty` — SDLT calculation for primary or additional property
- `rightmove_search` — current Rightmove listings (buy or rent)

What you get back (abbreviated):

```
{
  "postcode": "NG1 1AA",
  "transaction_count": 12,
  "median_price": 245000,
  "median_price_per_sqft": 187,
  "epc_match_rate": 0.67,
  "transactions": [
    {"address": "14 HIGH STREET", "price": 250000, "date": "2025-08-15",
     "floor_area_sqm": 85, "epc_rating": "C"}
  ]
}
```

Example request after connecting:

> *"Pull comparable sales for NG1 1AA, terraced houses only"*

UK LEGAL TOOLS

What it does	UK legislation lookup, BAILII case law search, Hansard debates, parliamentary bills, HMRC guidance, committee evidence

Example tools:

- `legislation_search` — search UK Acts and Statutory Instruments by keyword
- `case_law_search` — search BAILII for UK court decisions and tribunal rulings
- `parliament_search_hansard` — search parliamentary debates by topic
- `hmrc_search_guidance` — search HMRC guidance documents

Example request after connecting:

> *"Find case law on Section 21 eviction notices from the last 2 years"*

UK DUE DILIGENCE

What it does	Companies House searches, Charity Commission, Land Registry title searches, Gazette insolvency notices, VAT validation

Example tools:

- `company_search` — find a UK company by name or number
- `company_profile` — full company details from Companies House
- `company_officers` — directors and secretaries
- `gazette_insolvency` — check The Gazette for insolvency notices
- `vat_validate` — verify a UK VAT number

Example request after connecting:

> *"Look up the freeholder company for this building on Companies House"*

P6 (PYP6XER)

What it does	Primavera P6 XER schedule analysis, earned value, health checks, critical path, float analysis

Example tools:

- `pyp6xer_schedule_health_check` — analyse a P6 schedule for quality issues
- `pyp6xer_earned_value` — calculate earned value metrics from schedule data
- `pyp6xer_critical_path` — identify critical path activities
- `pyp6xer_float_analysis` — total float distribution across the schedule

Example request after connecting:

> *"Run a health check on this P6 schedule file"*

APPENDIX C: TROUBLESHOOTING

Skill will not upload

- The file must be named exactly `SKILL.md` (case-sensitive)
- The skill folder must contain SKILL.md at its root, not nested in a subfolder
- Frontmatter must start at line 1 with `---`, no leading spaces or blank lines before it
- Keep ZIP files a reasonable size

Skill does not trigger

- Ask Claude: "When would you use the [skill-name] skill?" If Claude cannot answer clearly, the description needs work.
- Check the description includes "Use when..." with specific trigger phrases
- Verify the skill is toggled on (Customize > Skills)
- Verify code execution is enabled (Settings > Capabilities)

Skill triggers when it should not

- Add negative boundaries to the description: "Do NOT use for..."
- Check for overlapping descriptions with other installed skills (see Mistake 6 in Section 10)
- Make the description more specific. Narrow the verb and noun

Connector (MCP) not working

- Verify the connector is listed and active (Customize > Connectors)
- Test the connector directly without the skill: ask Claude to call one tool by name
- For hosted servers (Fly.io, Railway): the first call after a sleep period may timeout. Retry once (see Mistake 7 in Section 10)

Description character limit

- The skills specification and claude.ai both allow up to 1,024 characters
- If your description is approaching the limit, tighten the language and move detail to the SKILL.md body

APPENDIX D: SECURITY

Skills are markdown files that tell Claude what to do. Treat them like software you install.

Before using a skill from an unknown source:

- Read the full SKILL.md. Check that the instructions match the stated purpose
- If the skill includes a `scripts/` folder, open the files and look for unexpected network calls or file operations
- Check `references/` and `assets/`. Verify the content is what the skill claims
- If the skill connects to an MCP server you have not used before, test the server independently first

Risks to be aware of:

- A skill's instructions control what Claude does with your connected services. A malicious skill could misuse your connectors
- Scripts bundled with skills run on your machine with your permissions (Claude Code only; claude.ai runs scripts in a sandbox)
- Skills that reference external URLs could fetch content you did not expect

The simple rule: only use skills from sources you trust. The bundled skills in this guide are open for inspection. For anything else, read before you install.

BUILD. SHARE. REPEAT.

bouch.dev

Skills are just markdown files that encode how you already work.