



SEVEN THINGS EVERYONE MIXES UP

Skill, plugin, hook, subagent. They all look the same and do different things. The difference comes down to one thing: when it loads and who triggers it.

1. ALWAYS

Claude has it in its head from the first second of every conversation.

2. ON CALL

Sits off to the side. Claude knows it exists and opens the whole thing only when it's needed.

3. ON ITS OWN

Runs when an event happens, even if nobody remembers to ask.

SEVEN LAYERS, IN PLAIN ENGLISH

01 CLAUDE.MD

ALWAYS

A text note for Claude. It reads it at the start of every conversation, whatever you're working on.

EXAMPLE "Reply in English. Never delete files without asking."

02 SKILL

ON CALL

A saved procedure. It loads in full only when needed, so it can easily run three pages.

EXAMPLE The full steps for making an invoice. You call it with one word.

03 SLASH COMMAND

ON CALL

A command with a slash at the start of a message. It runs a built-in function or your skill.

EXAMPLE /clear clears the conversation. /invoice runs your procedure.

04 SUBAGENT

ON CALL

A helper with its own head. You send it off to do the messy work and it hands back only the result.

EXAMPLE Reads 100 files, writes 3 sentences. Your desk stays clean.

05 HOOK

ON ITS OWN

An automatic safeguard. An event happens and the hook runs, even if Claude doesn't remember.

EXAMPLE Every time Claude saves a file, a check runs. No asking.

06 MCP

ON CALL

A cable into someone else's app. Without it, Claude knows nothing about your Notion or email.

EXAMPLE You connect Gmail and Claude drafts the reply right away.

07 PLUGIN

WRAPPER

A box for everything above. It usually can't do anything extra, but it packs it up and ships it to a coworker.

EXAMPLE You install a ready-made plugin and have new commands in a minute.

AND ONE SAFEGUARD

A rule in text is a request. Claude usually follows it, but it's not guaranteed.

REMEMBER If something must always hold, it belongs in a hook, not in a note.

WHAT PEOPLE MIX UP

CLAUDE.MD VS SKILL A note is always read. A skill only when needed.

SKILL VS COMMAND The skill is the content. The slash is just the switch.

SKILL VS SUBAGENT A skill is a guide. A subagent is a worker.

SKILL VS HOOK Claude reads a skill and decides. A matching hook always fires.

MCP VS SKILL MCP is the cable to the app. A skill is the how-to for using it.

PLUGIN VS THE REST It mostly just boxes it all up.

START HERE

Don't set up seven things on day one. Write a CLAUDE.md and put three rules in it. The second time Claude makes the same mistake, add a line. The third time you paste the same procedure into chat, turn it into a skill. The rest can wait until something annoys you.

CLAUDE.MD



ONE FILE YOU’LL STOP EXPLAINING

A text file in your project folder. Claude reads it on its own at the start of every conversation, so you don’t have to keep repeating the same things. **You write it in normal sentences and it’s done in five minutes.**

WITHOUT IT

You start every new conversation by explaining. Claude guesses where things are and keeps making the same mistake.

WITH IT

It starts with your rules in its head. You write the task, not an intro.

HOW TO CREATE IT

All in the Claude app, with a mouse and one sentence.

STEP 01

Open the Claude app and click the **Code** tab.

STEP 02

Choose **Local**, click **Select folder** and find your project folder.

STEP 03

Type this command in the box. Claude scans the files and drafts it.

`/init`

STEP 04

Read it. **Delete what doesn’t fit. Add what only you know.** This is the important part.

The Code tab opens only on a paid account (Pro, Max, Team or Enterprise). And if you already have a CLAUDE.md in the project, **/init won’t overwrite it**, it only suggests what to add.

WHAT GOES IN IT

This is what a finished file looks like. Feel free to copy it.

File **CLAUDE.md** in the project folder

Website for The Rooster café

How to talk to me

- Reply in English.
- Explain things simply, I’m not a coder.
- Always ask before you delete anything.

Site rules

- The colors are dark green and cream.
Don’t add any others.
- Don’t edit the photos, they’re from a photographer.
- The phone number 777 123 456 is correct.

Things you can’t figure out yourself

- The owner approves the menu prices.
Never change them, only suggest them to me.

HOW TO TALK TO ME

Language and style. No jargon you don’t understand.

WHAT IT MUST FOLLOW

Things that hold every time. And what it must not touch at all.

SPECIFIC NUMBERS AND NAMES

Phone, address, color. Written down, so it doesn’t guess.

WHAT ONLY YOU KNOW

Why things are the way they are. Nobody can read that from the project. The most valuable lines.

HEADINGS WITH A HASH

They keep the text tidy. Claude finds its way around like you do.

WHERE IT’S SAVED

Rules for this one project

CLAUDE.md

In the project folder, next to your other files. Start here, almost everything belongs here.

Your rules for all projects

~/ .claude/CLAUDE.md

Applies to every project on your computer. No need to hunt for it, /memory opens it for you.

When you have several files at once, Claude reads **all** of them. They don’t overwrite each other, they stack.

WHEN TO OPEN IT AGAIN

Don’t write it up front. Add to it when something annoys you.

Claude made the same mistake twice. Write one line so it doesn’t make it a third time.

You’re typing a correction into chat that you typed last time. That exact sentence belongs in the file.

You’d have to explain it to someone new. What they need, Claude needs too.

THREE MISTAKES EVERYONE MAKES

“I’ll just write everything in there now.”

Don’t. The longer the file, the worse Claude sticks to it. **Stay under 200 lines.**

“I’ll keep it general so it always applies.”

“Do it properly” means nothing. **“Ask before you delete anything” does.**

“It’s written down, so it applies.”

Claude treats it as a request, not a law. **It usually listens. Not always.**

/memory

Lists your files and opens them for editing. **No need to dig through folders.**

/context

In the **Memory files** list you see what actually loaded. Not there? Claude can’t see it.

START HERE

Run **/init** and let Claude write the base. Then write nothing more. The second time the same thing annoys you, add one line. In a month you have twenty lines that really hold, instead of two hundred made-up ones.



A ROUTINE YOU WRITE ONCE AND THEN JUST CALL

A folder with one text file. Inside is a routine you'd otherwise paste into the chat over and over. You write it once, and after that a slash and one word is enough. **Claude opens it when it's needed, so you stop dictating the steps.**

WHEN TO MAKE IT Don't make one in advance. You'll know it's time in any of these three cases.

- You paste the same routine** into the chat a third time. The second time you let it slide. The third time you don't.
- A how-to grew inside your CLAUDE.md.** A rule belongs there. A five-step routine doesn't.
- You have a checklist** you never remember in full. Let Claude remember it.

HOW TO CREATE IT You install nothing. Claude makes the folder and the file.

- STEP 01**

In the Claude app, open the **Code** tab and your folder. Same place as for CLAUDE.md.
- STEP 02**

Describe the routine in plain words, then add this sentence and a short name.

make this a skill
- STEP 03**

Claude writes the file and shows it to you. **Depending on the mode, it asks before saving or saves right away.**
- STEP 04**

Add the rules only you know. Then type a slash and the name, and try it for real.

If this is the very first skill in a project and Claude can't find it, type `/reload-skills`. **Later skills load on their own.**

WHAT A FINISHED ONE LOOKS LIKE The Rooster café answers reviews. Here's how it's written.

File **SKILL.md** in the **review-reply** folder

```
---
description: Writes a reply to a guest review.
  Use it when a new Google review comes in.
---

# Review reply
1. Find one specific thing in the review
  and respond to it.
2. Write three to four sentences. No more.

## Rules
- Keep it polite. Sign as Jana, the owner.
- No "we're sorry to hear that". Be specific.

## Leave this to me
- Never offer a discount or a voucher.
- Reviews under two stars: just show me.
```

THREE DASHES

Three dashes above and below mark the header. The short description for Claude goes inside.

DESCRIPTION

All Claude sees up front, with the name. Say what the skill does **and when to use it.**

STEPS

The order you do things in. Ten steps is fine, they fit here.

RULES

Lines you retype in the chat every time. Put them here.

LIMITS

What it must not touch. The key section for anything that goes out to a client.

WHY IT CAN BE LONG This is the whole difference between the two files.

CLAUDE.MD

Loads in full at the start of every conversation, whatever you're doing. That's why it stays under two hundred lines.

SKILL.MD

Claude keeps only that short description in mind. The rest loads when you actually use the skill. **That's why it can hold five hundred lines.**

WHERE IT'S SAVED The folder name is what you type after the slash.

Routine for this project only	<code>.claude/skills/review-reply/SKILL.md</code>	In the project folder. Start here. If you share the folder with a colleague, they get the skill too.
Routine for all projects	<code>~/ .claude/skills/review-reply/SKILL.md</code>	Works everywhere on your computer. Put what you use all the time here.

Next to **SKILL.md** you can drop a template, a sample of finished output, or a price list. Claude opens them only when it needs them.

HOW IT RUNS Two ways and one safeguard.

- YOU CALL IT**

Type a slash and the name. It runs, whatever Claude would have decided.
- CLAUDE PICKS IT UP**

You write "we got a review" and Claude uses it. It knows from the **description**.
- IF YOU DON'T WANT THAT**

Add the line `disable-model-invocation: true` to the header. Then it starts only when you call it.

A skill is still just text that Claude reads. **Most of the time Claude follows it to the letter, but that's not guaranteed.** For anything that goes out, read the result.

THREE MISTAKES EVERYONE MAKES

- "I'll put a title in the description."

Just "Reviews" isn't enough. **Use the words you'd say out loud to a colleague.**
- "It didn't run on its own, it's broken."

Usually it isn't. First check that Claude can see it at all. **Then add to the description when to use it.**
- "I'll cram in everything I do."

One skill is one routine. **Stay under five hundred lines** and put the rest next to it as a second file.

- /skills**

Lists what you have. You can switch off most skills here too.
- "Which skills do you have?"**

A plain question. If it doesn't list your new skill, **it can't see it.**

START HERE

Don't plan a list of skills. Open your last conversation and find the routine you typed for the third time. Tell Claude to make it a skill. **In ten minutes you have your first one and you know if it's any use.**

/COMMANDS



SHORTCUTS YOU DON'T HAVE TO SPELL OUT

Type a slash in the box and the app offers you a list. Most of these aren't tasks for Claude, they're levers that act right away. **You need four of them daily. The rest you'll remember when you have to.**

FOUR YOU NEED DAILY If you remember only four lines from this whole poster, make it these.

/clear Starts a fresh conversation. Your CLAUDE.md rules stay. A new task deserves a clean desk.	/context A colored grid shows what fills its head. When it runs low, Claude gets careless.	/compact Condenses the chat to free up space. Keeps a summary instead of the whole chat. Use it when you can't start fresh.	/rewind Rolls file edits and the chat back to an earlier point. Terminal-command changes aren't undone.
---	--	---	--

THE REST, BY SITUATION Don't memorize any of this. Read it and come back when the situation hits.

Starting a new project	/init /memory	Claude reads your files and writes the first CLAUDE.md. /memory opens it for editing.
Want to think it through first	/plan /model	Plan mode proposes steps, touches no files. /model switches the model for later chats too.
Something went wrong	/rewind /doctor	/rewind undoes edits and chat. /doctor finds what's broken or forgotten in your setup.
Returning to unfinished work	/resume /diff	/resume reopens an old conversation. /diff shows what Claude changed in your files.
Connecting your apps	/mcp /permissions	/mcp shows connected apps and their status. /permissions sets what Claude may do unasked.
Don't know what you have	/help /skills	/help lists all commands. /skills shows your own routines and can switch them off.
Need the result out	/copy /export	/copy puts the last answer on your clipboard. /export saves the whole conversation as text.

HOW TO TYPE IT Four messages in a row: clean up, set up, run my skill, copy the result.

The box where you type to Claude	THE SLASH COMES FIRST It has to be at the start of the message. In the middle of a sentence it's plain text and nothing runs.
/clear	THE APP SUGGESTS Type two or three letters and it offers the rest. You don't need to know it by heart.
/model opus	TEXT AFTER THE SPACE Whatever you add after the command, it becomes the task. Here the review goes straight into your skill.
/review-reply Great room, weak breakfast.	THEN ENTER A built-in command runs right away. With your skill, Claude starts up and gets to work.
/copy	

THREE KINDS THAT LOOK THE SAME They all start with a slash. Inside, each one is something different.

BUILT-IN /clear /model /context The logic is written right into the app. /clear and /model do the same thing every time , Claude has no say.	BUNDLED SKILL /doctor /code-review A ready-made task that Claude receives. Looser, because Claude has to think it through.	YOUR OWN /review-reply /offer A folder with a SKILL.md file that you wrote. The slash is just a switch , the content is the skill.
---	---	--

THREE MISTAKES EVERYONE MAKES

<i>"If it goes wrong, I'll hit /clear."</i> That clears the conversation, not your files. To undo changes, use /rewind.	<i>"I'll keep one conversation going all day."</i> The longer it gets, the worse Claude holds on to the task. New task, new /clear.	<i>"I have to memorize these commands."</i> You don't. Type just the slash and the app lists everything you have.
---	---	---

/ Just a slash at the start of a message. It shows the list and suggests as you keep typing.	/status Shows the version, model, account, and connections you're running. The first thing to check when Claude acts weird.
--	--

START HERE

Learn just one command: **/clear**. When you finish a task, type it and start the next one fresh. Most complaints that Claude "stopped listening" are just a crowded conversation nobody cleaned up.



SUBAGENT

A HELPER THAT TAKES THE MESS ELSEWHERE

Claude calls in a helper for a side task. The helper does the work in its own head and sends back only the result. **You won't end up with fifty files in your conversation that you'd never read anyway.**

WHY YOU WANT IT

Three reasons, and each one saves you something.

SAVES HEAD SPACE Searching and reading happen off to the side. You get three sentences back instead of thirty listings.	YOU CAN RESTRICT IT You decide what it may do. Read-only means it can't overwrite or delete anything.	CAN RUN CHEAPER For mechanical work you set a weaker model. Often good enough, faster and cheaper.
---	--	---

THREE YOU ALREADY HAVE

You create nothing. Claude sends these on its own when it fits.

EXPLORE A fast finder. It reads files and reports back. It can't write or delete , so run it without worry.	PLAN A scout for plan mode. It gathers material before Claude presents you a plan. Read-only too.	GENERAL-PURPOSE The all-rounder for complex work that includes changes. It has almost all the tools.
--	---	--

WHAT YOUR OWN LOOKS LIKE

A café keeps an eye on the competition nearby. This one visits the websites for it.

<pre>File competitor-research.md in the .claude/agents folder --- name: competitor-research description: Goes through the websites of nearby cafés and returns prices and opening hours. tools: Read, WebFetch, WebSearch model: haiku --- You are a scout. For each café, find out: 1. the price of an espresso and filter coffee 2. weekend opening hours 3. what it brags about on the homepage Return a table, not paragraphs. No commentary. If you can't find the site, say so and move on.</pre>	NAME The name you call it by. Lowercase letters and hyphens, no spaces.
	DESCRIPTION This is how Claude knows when to send it. Same rule as for a skill.
	TOOLS Reading and the web only, so it can't change anything.
	MODEL The cheapest model is enough. Leave thinking to the main chat.
	BODY TEXT Its instructions. It can't see your chat , so spell out the obvious.

WHERE IT'S SAVED

One file, same logic as skills.

A helper for this project only	<code>.claude/agents/name.md</code>	In the project folder. Share the folder and your colleague gets it too.
A helper for all projects	<code>~/ .claude/agents/name.md</code>	Works on your whole computer. For the one you always want on hand.

You don't have to write the file by hand. **Tell Claude who you need and it writes it for you.** If it's your first helper in a project, close and reopen the app once afterward.

HOW TO SEND IT

Two ways for you, one for Claude.

SAY IT IN A SENTENCE Send competitor-research to these 3 sites Claude usually obeys. The easiest way.	THE @ SIGN @competitor-research Type @ and pick it from the list. This forces it.	CLAUDE PICKS IT "What are the prices next door?" It knows from the description. So write it as if a colleague were reading it.
---	--	--

WHEN NOT TO USE IT

<i>"We need to sort this out together."</i> A helper can't do ping-pong. If you expect follow-up questions, stay in the main conversation.	<i>"It's a two-minute job."</i> It starts with an empty head and has to get its bearings. For a small thing that's a delay.	<i>"Let it remember this for later."</i> By default it remembers nothing. It returns a summary and ends. What should stay belongs in CLAUDE.md.
--	--	---

<code>/tasks</code> Lists what is currently running in the background.	<code>Ctrl + B</code> Sends the running task to the background. You keep typing, the helper keeps working.
--	---

START HERE

Don't create one. Next time Claude buries you in output you didn't want to read, tell it: **"next time, send this through Explore"**. Write your own helper once you've wanted the same search three times.



MCP

THE CABLE BETWEEN CLAUDE AND YOUR APPS

On its own, Claude knows nothing about your Gmail, calendar, or Notion. Through MCP it connects to them and can read from them and write to them. **The acronym stands for Model Context Protocol, but you don't need to remember that.**

WHAT YOU CAN DO WITH IT You'll know it's for you if you keep copying things from one window to another.

YOU STOP COPYING

"Pull the brief from Notion and write a proposal for it."

Claude loads it on its own. You don't switch windows and you lose nothing along the way.

PREPARES REPLIES

"Go through my unread emails and prepare replies."

The drafts wait for you in Gmail. By default, it asks before sending anything.

SEES YOUR DATA

"How many meetings do I have next week?"

Calendar, spreadsheets, databases. It answers from reality, not from a guess.

HOW TO CONNECT IT The easiest way doesn't go through Claude Code but through the claude.ai website.

STEP 01

On **claude.ai**, open **Customize**, then **Connectors**.

STEP 02

Pick a service from the list and sign in to it. You approve what it can access.

STEP 03

Open Claude Code signed in with your claude.ai account. The connector **shows up on its own**.

STEP 04

Check it. It has to show as connected in the list.

`/mcp`

Gmail, Calendar, and Microsoft 365 can be connected **only this way**, you can't connect them directly from Claude Code. Other services can also be added with a command in the terminal, but you don't need to know how.

WHAT YOU CAN CONNECT Examples named right in the docs. There are hundreds of servers today.

Email and calendar	Gmail, Google Calendar, Microsoft 365. Reading, searching, and preparing replies.
Notes and tasks	Notion, Slack, Figma. It reads the brief where you actually keep it.
Company systems	GitHub, Stripe, HubSpot, Sentry, databases. Usually someone from IT handles these.

WHAT IT LOOKS LIKE A simplified flow of one request, not the exact look of the app.

Connected **Gmail**, one message and what happened

You Find Jane's email about the September date and prepare a reply.

Claude gmail · thread search
found 1 thread, Aug 3

gmail · draft creation
draft ready, I did not send it

YOU SEE WHAT IT TOUCHED

Every step lists which service it used and what it did there.

YOU APPROVE

On an action that changes something, it stops and waits. **Silence doesn't mean yes.**

THE DRAFT WAITS FOR YOU

It stays in Gmail. Sending is your decision.

BEFORE YOU CONNECT ANYTHING This section matters more than everything else on the poster.

YOU LET AN OUTSIDER INTO YOUR DATA

The server sees what you allow. **Connect only servers you know**, and check what they ask of you.

OUTSIDE TEXT IS JUST TEXT

A stranger's email or a website can hide an instruction for Claude. **It is never a command from you.**

WATCH WHAT GOES OUT

For anything that goes out, keep approvals on. What may happen without asking, you set in **/permissions**.

THREE MISTAKES EVERYONE MAKES

"I'll connect everything, just in case."

Every server costs something and sees something. **Connect one and use it for a month.**

"If it has a connector, it can do everything."

It does what the server offers. Check whether it only reads or also writes and sends.

"It doesn't work, it's broken."

Check the status first. Usually the **sign-in just expired** and you only need to sign in again.

`/mcp`

A list of connected services and their status. **The first place to look.**

`/permissions`

You set what Claude may do without asking and what it must always ask you about.

START HERE

Connect just one service, the one you copy from most often. For a month, only read, write nothing. **If after a month you forget you have it, disconnect it again.**



HOOK

A RULE YOU CAN'T TALK YOUR WAY AROUND

Claude reads a rule in CLAUDE.md and usually follows it. A hook runs on its own, no matter what Claude happens to think. **It doesn't wait for anyone to read a bullet point, and you can't talk it out of it.**

REQUEST VERSUS LOCK

This is the whole reason hooks exist.

A RULE IN CLAUDE.MD

"Never edit the passwords file." Claude reads that as a request. It usually obeys. Not always.

HOOK

Claude touches that file and the hook rejects the action. **Nobody weighs in on the decision, not Claude, not you.**

WHEN IT FIRES

There are over thirty events. These are the ones people set hooks up for.

When a session starts	SessionStart	Injects context that should be on hand every time, even after the conversation is cleared.
Before your message goes out	UserPromptSubmit	You can add context to it, or stop it right there.
Before Claude runs something	PreToolUse	This is where you can forbid a specific action before it happens. That's where the power is.
After an action finishes	PostToolUse	Formatting, tests, writing to a log. Too late to forbid it, it already happened.
When Claude wants to stop	Stop	You can send it back to work if it left something unfinished.

WHAT IT LOOKS LIKE

It goes into settings.json. This one guards against deleting files.

File .claude/settings.json in the project folder

```
"PreToolUse": [
  {
    "matcher": "Bash",
    "hooks": [
      {
        "type": "command",
        "if": "Bash(rm *)",
        "command": "${CLAUDE_PROJECT_DIR}/.claude/hooks/block-rm.sh"
      }
    ]
  }
]
```

PRETOOLUSE

When it fires. Here, right before Claude runs a command.

MATCHER

The coarse filter. React only to running commands, not to writing files.

IF

The fine filter. Of all commands, watch only deletes. The rest goes through.

COMMAND

Your script. It prints **deny** and the action doesn't happen, or **ask** and it asks you.

BEFORE YOU TURN IT ON

This section matters more than everything else on the poster.

IT'S A PROGRAM WITH YOUR RIGHTS

Nothing isolates it. It sees your files, variables and keys. **Write it as carefully as anything else.**

SOMEONE ELSE'S REPO, SOMEONE ELSE'S HOOK

Project hooks run as soon as you trust the folder. **Before you trust a folder that isn't yours, read its .claude/settings.json.**

A MODEL IS NOT A LOCK

A hook can also decide with a prompt. That can be fooled. A hard limit belongs in a script, not in reasoning.

THREE MISTAKES EVERYONE MAKES

"I'll put it in CLAUDE.md, so it applies."

It doesn't. If it has to apply every time, **it's a hook, not a bullet point.**

"The script returns 1 and that stops it."

It doesn't. Only **exit code 2** blocks. Exit code 1 just logs an error and things carry on.

"I'll add a hook for everything."

Each one adds a delay and one more thing that can break. Start with one.

/hooks

Lists every hook and which file it comes from. **Read before you trust.**

deny

The answer a script gives to reject an action. Next to it are **allow** and **ask**. **defer** works only in **claude -p**.

START HERE

Don't set up hooks just in case. Pick one thing you can't afford to lose and lock it. **One hook that holds is worth more than ten bullets in CLAUDE.md that Claude sometimes reads and sometimes doesn't.**



PLUGIN

A BOX FOR EVERYTHING YOU'VE SET UP

You pack skills, subagents, hooks and app connections into one folder. A colleague installs it with one command and has exactly what you have. A typical plugin can't do anything extra. It just packs, versions and hands things out.

WHEN A FOLDER IS ENOUGH AND WHEN YOU NEED THE BOX

Recommendation from the docs: start on the left, move to the right when someone else wants it.

A PLAIN .CLAUDE/ FOLDER Your stuff in a project. Quick experiments, personal workflows. The skill is called /proposal . You share it by copying.	PLUGIN Versioned, installable, shared. The skill is called /my-plugin:proposal , so it can't clash with someone else's.
---	--

WHAT FITS INSIDE

All of it optional. Feel free to pack just one skill.

skills/	Workflows. Each in its own folder with a SKILL.md file, exactly as outside a plugin.
agents/	Subagents with a mind of their own. One file each.
hooks/hooks.json	Hooks that fire on events. This is why other people's plugins deserve a close look.
.mcp.json	Connections to apps. Your colleague has them right away, no clicking needed.
bin/	Executable programs the plugin adds to your path while it's enabled.

WHAT IT LOOKS LIKE INSIDE

One folder, usually with a manifest, the rest optional.

Folder my-plugin anywhere on disk my-plugin/ ├─ .claude-plugin/ │ └─ plugin.json ├─ skills/ │ └─ proposal/SKILL.md ├─ agents/ │ └─ research.md └─ hooks/hooks.json	PLUGIN.JSON Name, description, version. The name becomes the prefix of every skill inside. NOTHING ELSE IN .CLAUDE-PLUGIN The most common mistake. Only the manifest goes inside. All other folders belong one level up. THE VERSION DECIDES THE UPDATE If the manifest has a version, others won't get updates until you bump the number. TRY IT WITHOUT INSTALLING You start Claude Code pointing at the folder and the plugin loads for that session only.
---	---

WHERE THEY COME FROM

Anthropic runs two main marketplaces. The rest of the internet isn't curated.

CLAUDE-PLUGINS-OFFICIAL Anthropic's main marketplace, most of its plugins are built by partners. It's added automatically on first launch.	CLAUDE-COMMUNITY Third-party plugins that authors submitted to Anthropic. You add it by hand.
--	---

BEFORE YOU INSTALL ANYTHING

A plugin isn't just text. It can carry code that runs.

YOU'RE INSTALLING SOMEONE ELSE'S CODE Hooks inside run on their own and MCP servers can reach your data. Install only what you trust.	CHECK WHAT IT BROUGHT After installing, look in /hooks and /mcp . You'll see exactly what you got.	A MARKETPLACE IS NO GUARANTEE Even a plugin from Anthropic's marketplace runs with your rights. You stay responsible for what you turn on.
--	--	--

THREE MISTAKES EVERYONE MAKES

<i>"A plugin will be able to do something extra."</i> Usually it can't. It's the same stuff as in .claude/ , just packaged. A box, not an engine.	<i>"I installed it and nothing's there."</i> Check the message after installing. Sometimes it also needs /reload-plugins .	<i>"I'll build a plugin right at the start."</i> Wasted work. Use your setup for a month first, then pack it.
/plugin The manager. Browse marketplaces, install, enable and disable, plus a tab for errors.	/reload-plugins Loads changes without a restart. You use it while building your own.	

START HERE

Don't build one. Open **/plugin**, browse the curated list and install one that saves you work. **Pack your own only when someone asks for your setup a second time.**