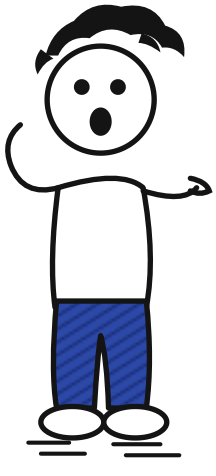
Everyone is building amazing things with Claude right now.

And I don't understand a word of it.



?

CLAUDE.md

Skills

Plugins

MCP

Hooks

Subagents

/Commands

Memory

Checkpoints

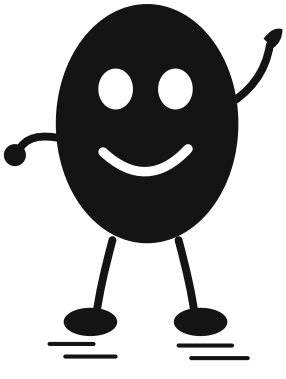
Cowork

Routines

Plan mode

/loop

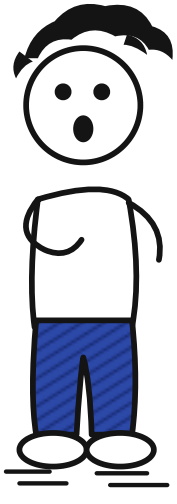
Come on, I'll explain.

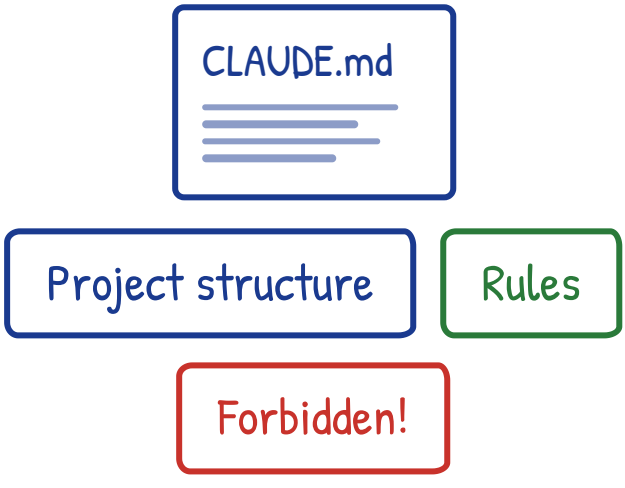


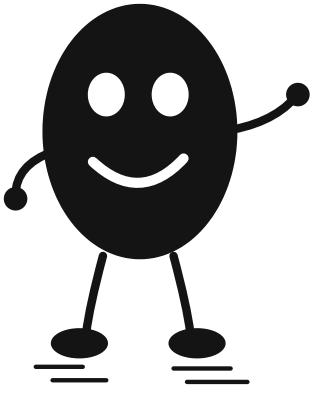
Let's start with CLAUDE.md. Why does everyone have one?

CLAUDE.md is basically a user manual that you hand to Claude.

It says how your project works, which rules apply – and what Claude must never do.



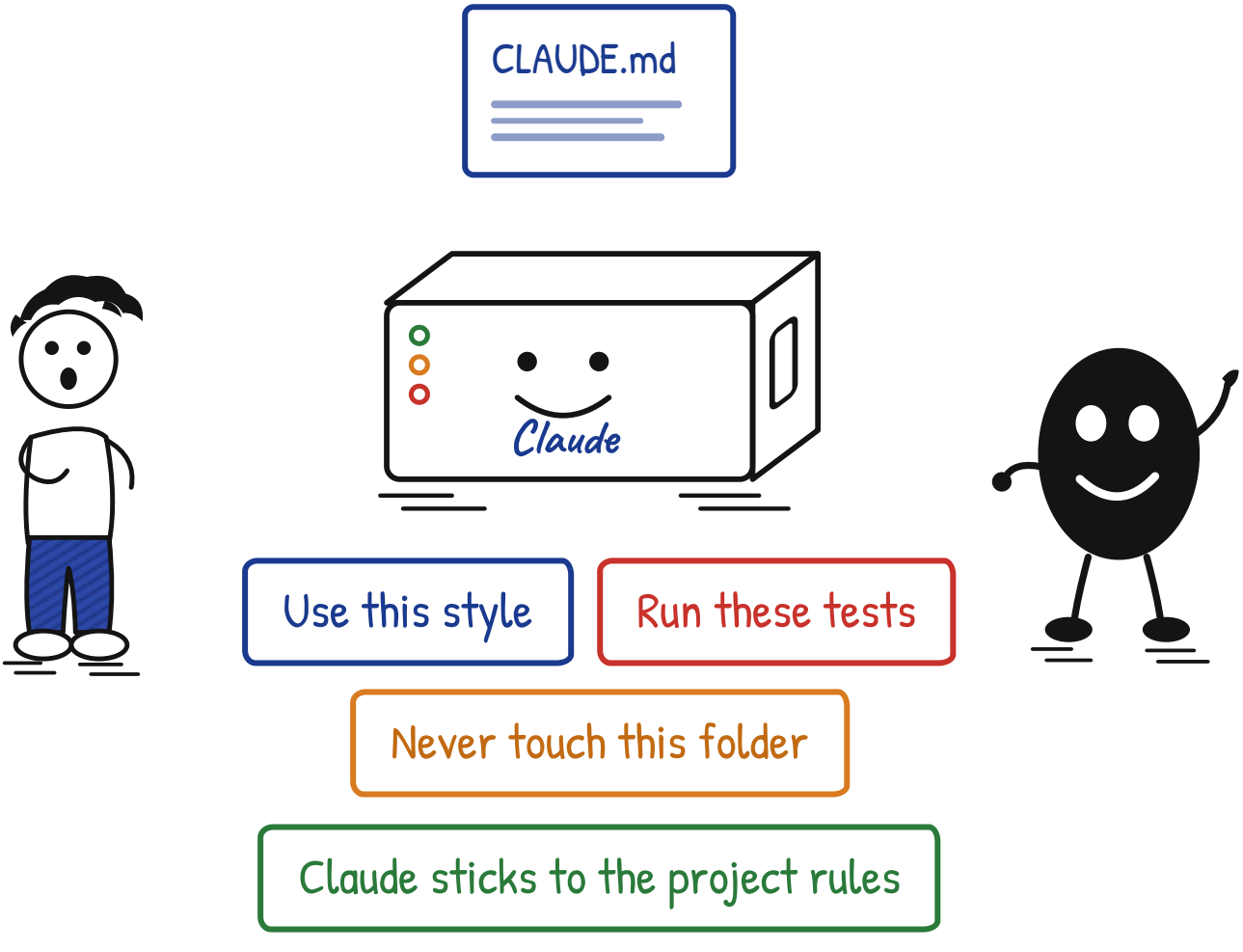




So I don't have to explain the same thing every session?

Exactly. Claude reads the file automatically as soon as you start inside the project.

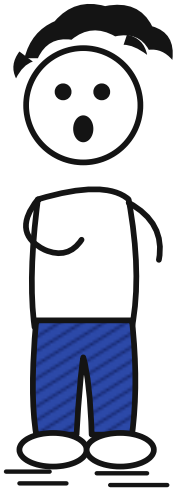
There are even three levels: globally in `~/.claude/`, in the project folder and per subfolder. `/init` writes you a first draft.

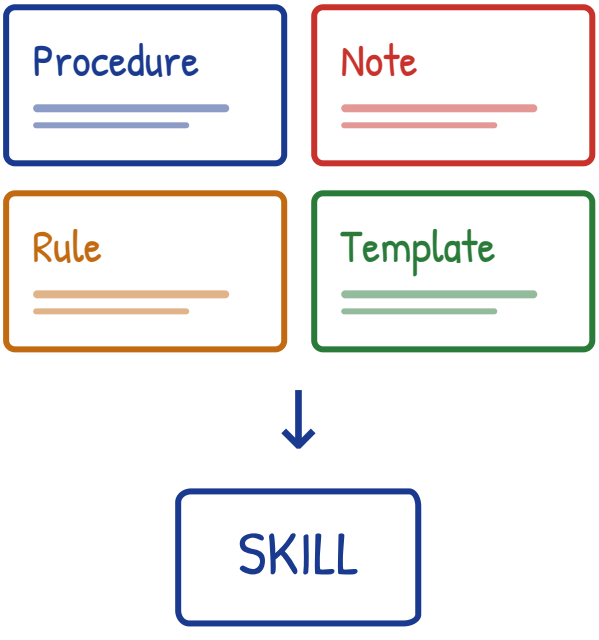


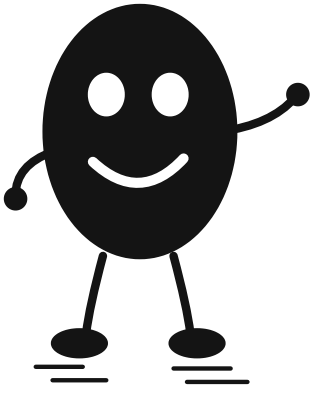
Okay. So what are skills, then?

A skill teaches Claude **how** a particular job is done properly.

Instead of explaining your whole procedure every time, you turn it into a reusable skill.



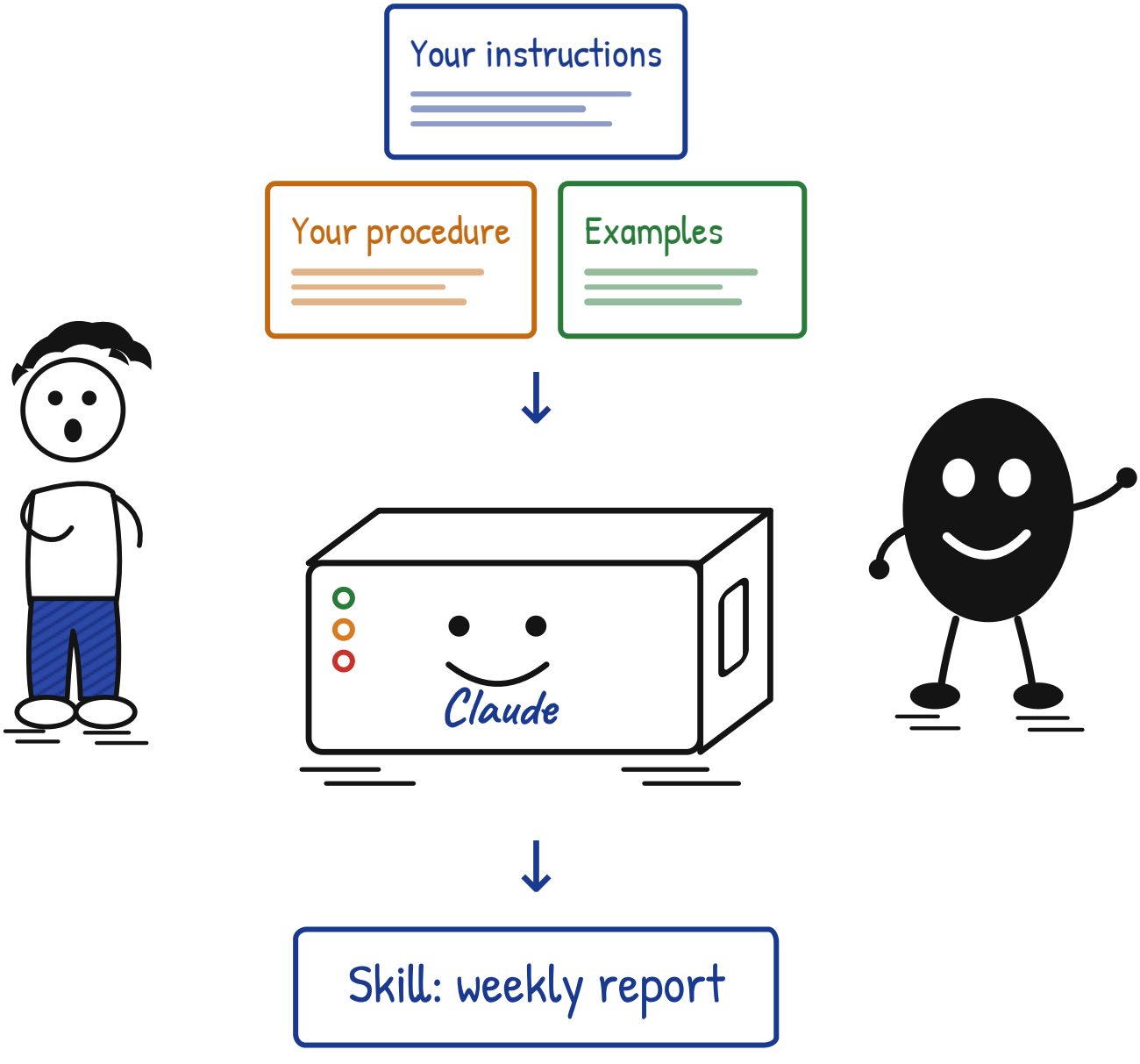




Give me an example.

Say you want the same analysis report every week.

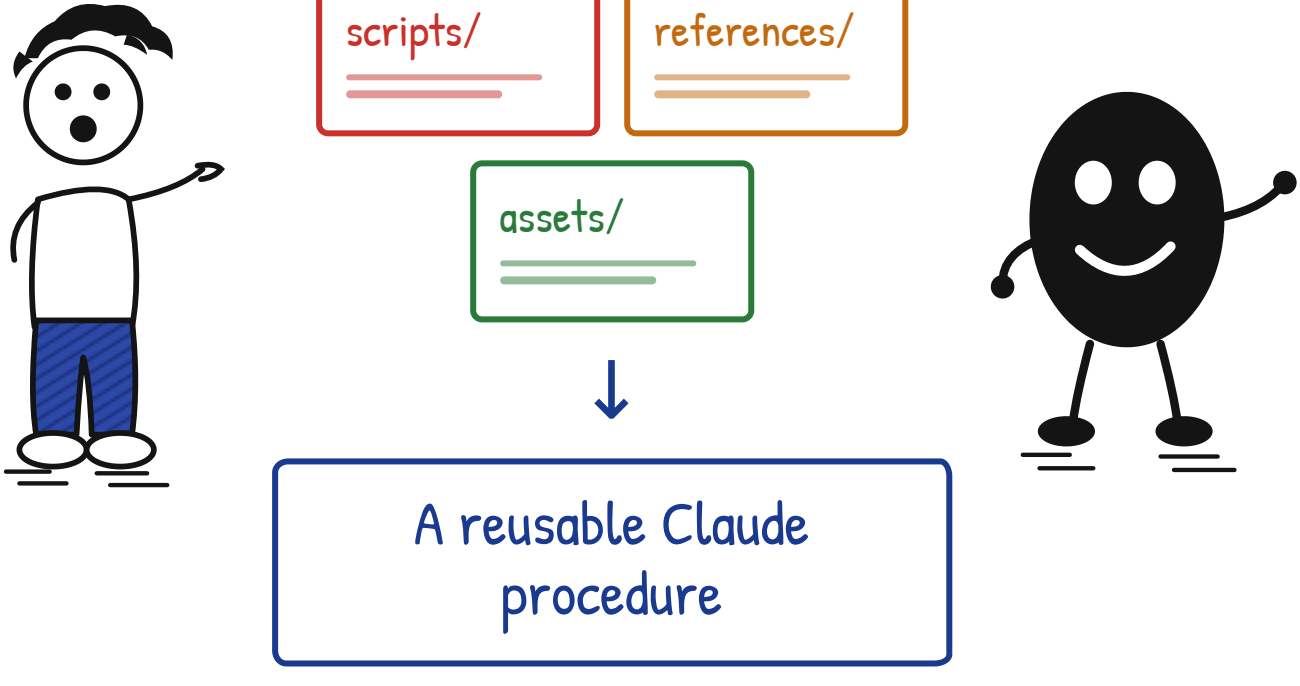
Your instructions, your procedure, your examples – that becomes a skill you simply call.



And those SKILL.md files everyone downloads from GitHub – they're just instructions?

Pretty much. A skill is a folder with a `SKILL.md` in it, plus scripts, templates and reference files.

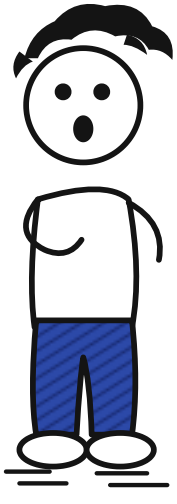
Claude first reads only the name and description – the rest only when it really needs it. You can also call it yourself with `/skill-name`.

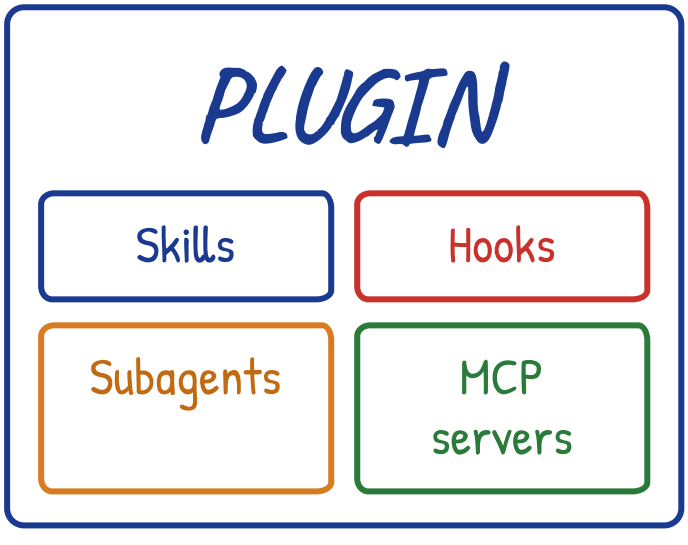


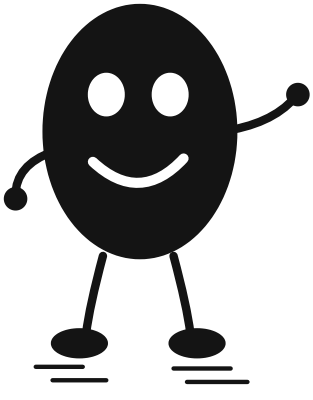
And plugins? Aren't those just skills as well?

Not quite. A skill is **one** ability.

A plugin is more of a **bundle**: skills + hooks + subagents + MCP servers + custom commands in one go. You install it from a marketplace with `/plugin`.





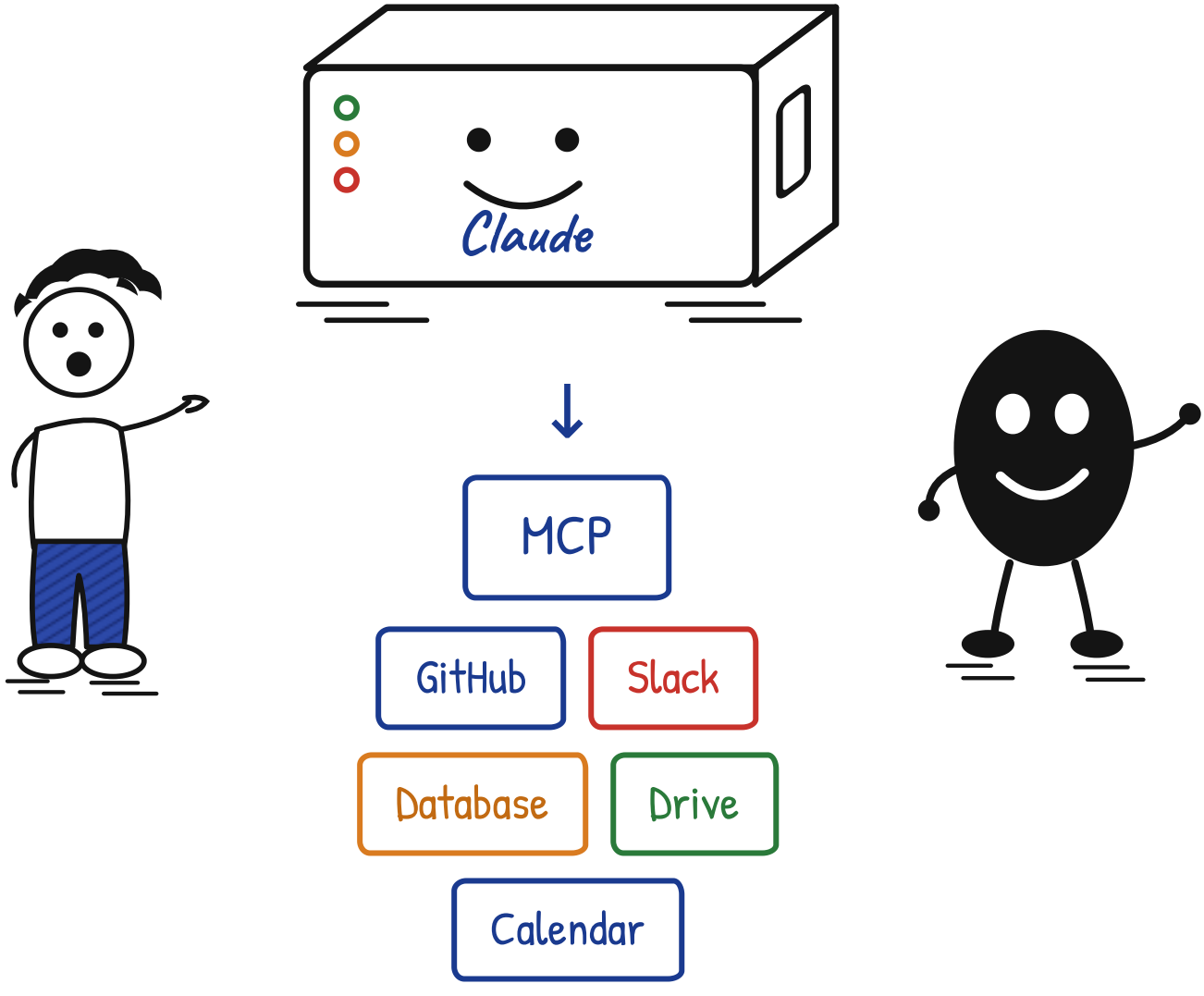


Skill

Hold on - MCP servers. What are those?

MCP is basically how Claude connects to things **outside** of Claude.

One standard plug for other people's programs and services.



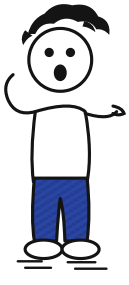
So I don't have to paste everything in by hand any more …

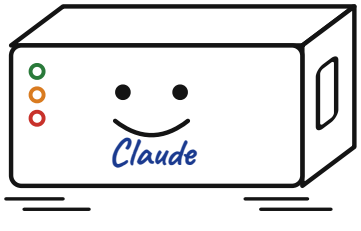
Claude connects on its own, fetches the information – and may act there too, if you allow it.

`/mcp` shows you what is currently connected.

Without MCP

GitHub



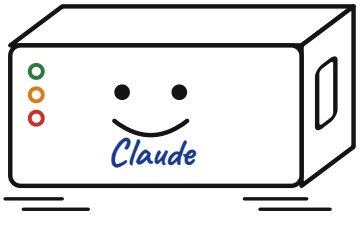


you copy everything by hand

With MCP

GitHub

MCP

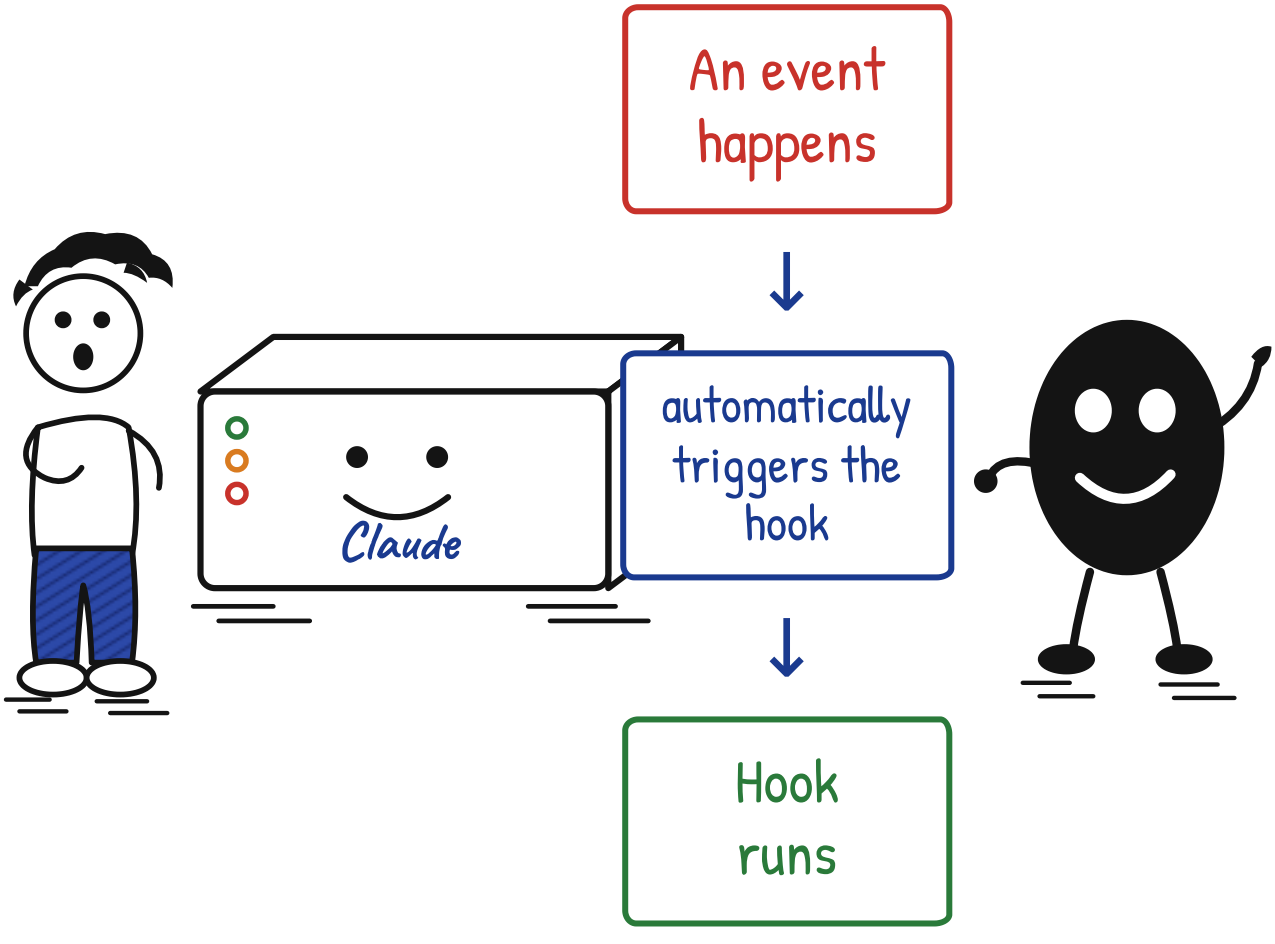


Claude fetches it itself

Right. And hooks? That sounds even more confusing.

Hooks are different because **Claude doesn't** decide whether they happen.

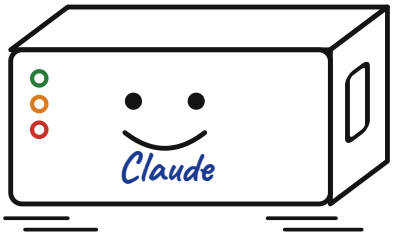
They fire automatically as soon as a certain event occurs. Set up in `settings.json` or through `/hooks`.

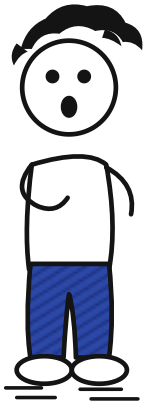


For example?

A hook can run your formatter after every file change – or stop Claude **before** a dangerous command runs.

Typical events: `PreToolUse`, `PostToolUse`, `UserPromptSubmit`, `SessionStart`, `Stop`.





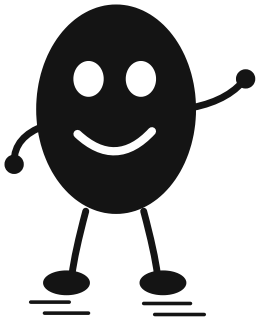
Claude changes a file

↓

HOOK runs automatically

Format | Test | Check

Report | STOP

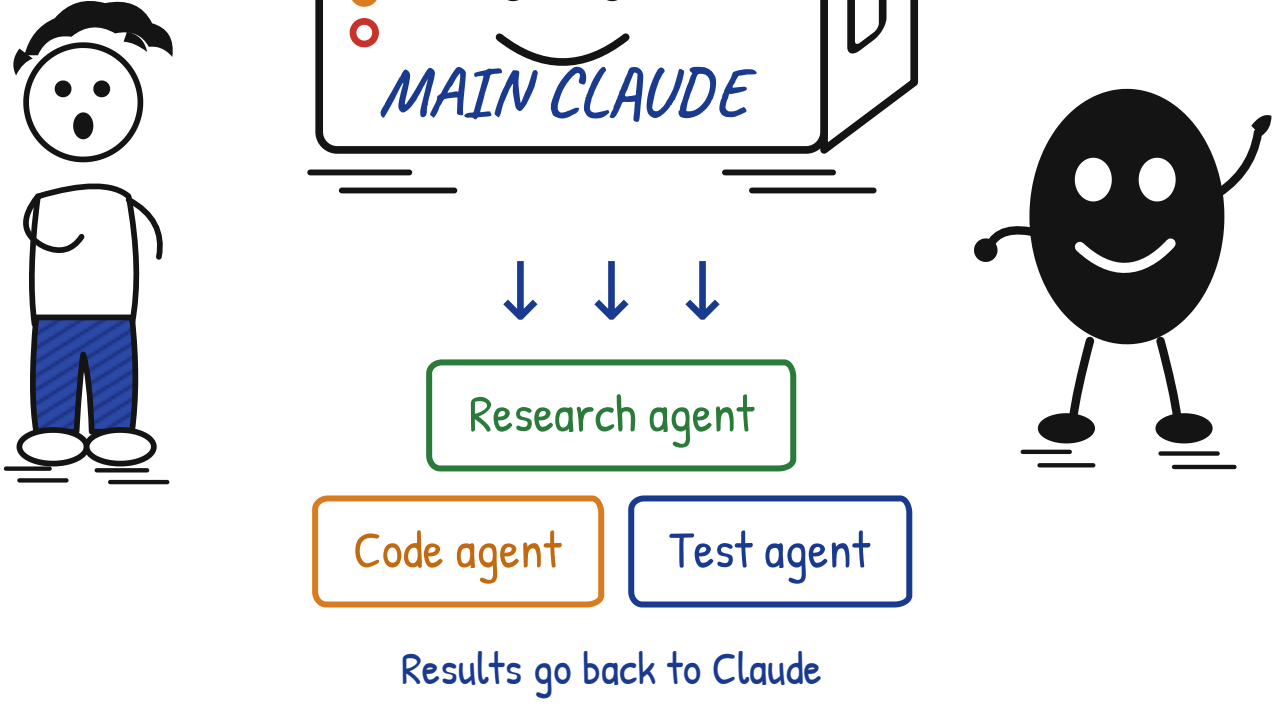


Stops before dangerous commands

What about subagents? Are those literally several Claudes?

They are independent workers Claude can hand clearly defined jobs to – each with its own context window and its own model.

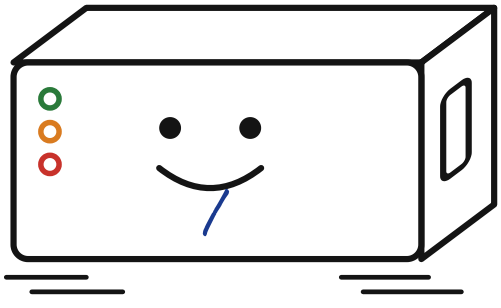
One researches, one reviews code, one tests. Then they report back to the main Claude. You create them as files under `.claude/agents/` – or you let Claude do it for you.

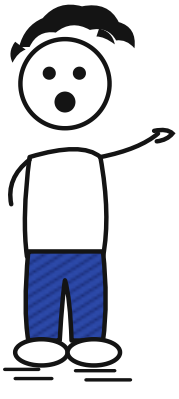


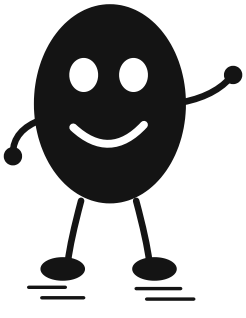
And what are all those /something commands everyone uses?

The slash is Claude Code's shortcut menu.

There are built-in ones like `/clear` and `/compact` – and skills bring their own commands that you can call yourself.







/clear
start over

/compact
boil the context down

/context
what eats the space?

/rewind
rewind

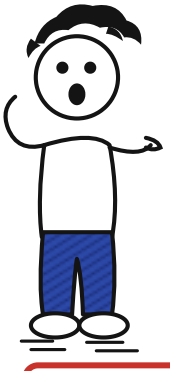
/model
switch model

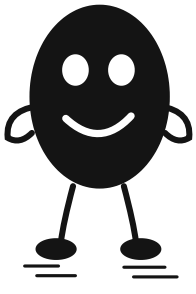
/usage
check your usage

Is Claude allowed to do anything at all on my machine?

On the paid plans Claude Code now starts in **auto mode**: it works straight through, and instead of you a second model reviews every action.

You can switch at any time with `Shift`+`Tab`: **manual** goes back to asking before every change, **plan** only looks and puts a proposal in front of you.





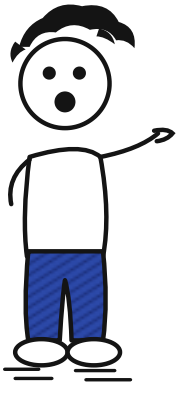
Auto	works straight through – the reviewer watches, the default
Manual	reads on its own, asks before every change
Accept edits	may touch files, everything else still asks
Plan	only looks and proposes – changes nothing

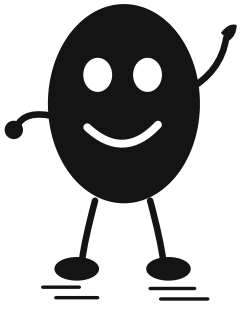
switch with Shift + Tab

Are there shortcuts worth knowing?

Four characters save most of the typing – and two keys are the emergency exit.

Remember Esc above all: you don't have to wait until Claude is done. Stop it, put it right, carry on.





`@file`
pull a file into the conversation

`!command`
run something yourself

`/`
open the command menu

`Shift+Tab`
switch permission mode

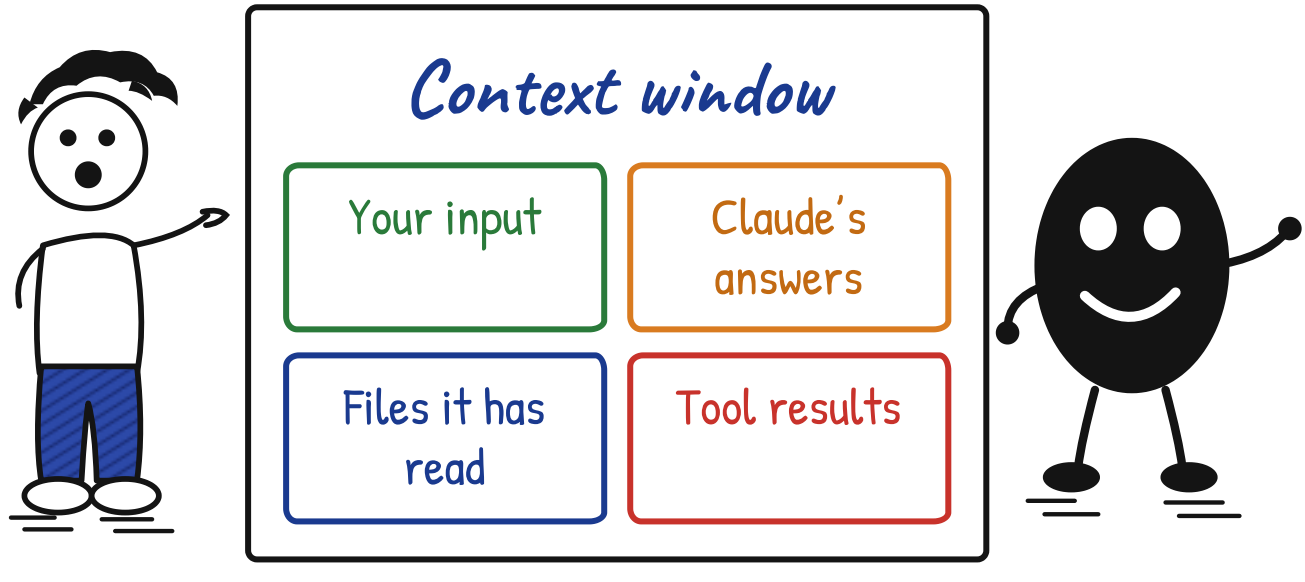
`Esc`
stop Claude right now

`Esc Esc`
rewind

Last question: what exactly is this context window?

That is Claude's working memory for the conversation you are having.

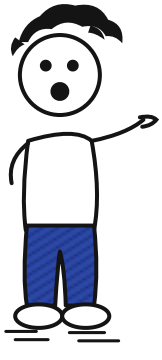
Your input, Claude's answers, files it has read and tool results all take up room in it.

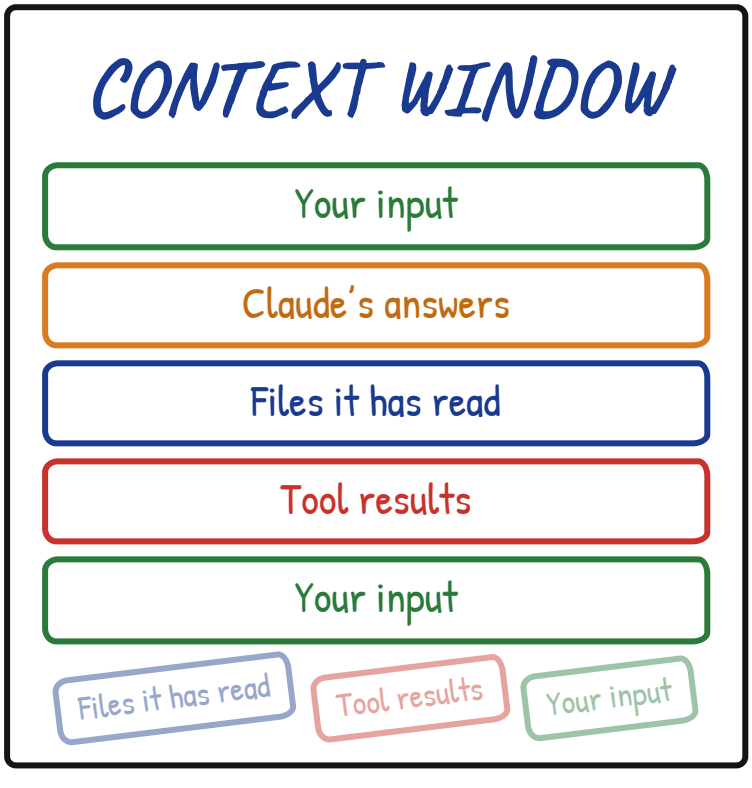


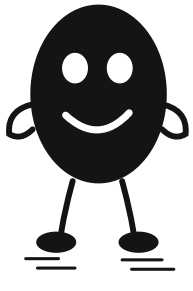
So when people say "Claude lost the plot" ...

... then usually so much has piled up that it no longer all fits into the working memory at once.

Claude Code then summarises automatically. `/context` shows you what is taking up the space, `/clear` gives you a clean start.





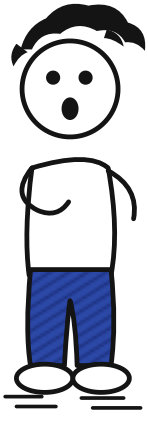


What Claude currently has in mind

And if Claude messes up? Is everything lost?

No – Claude Code sets checkpoints automatically before it changes files.

`Esc` twice or `/rewind`, and you roll back: just the conversation, just the files, or both.

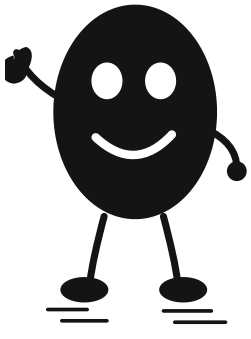


State A → State B → State C

this is where it went wrong

← ← ←

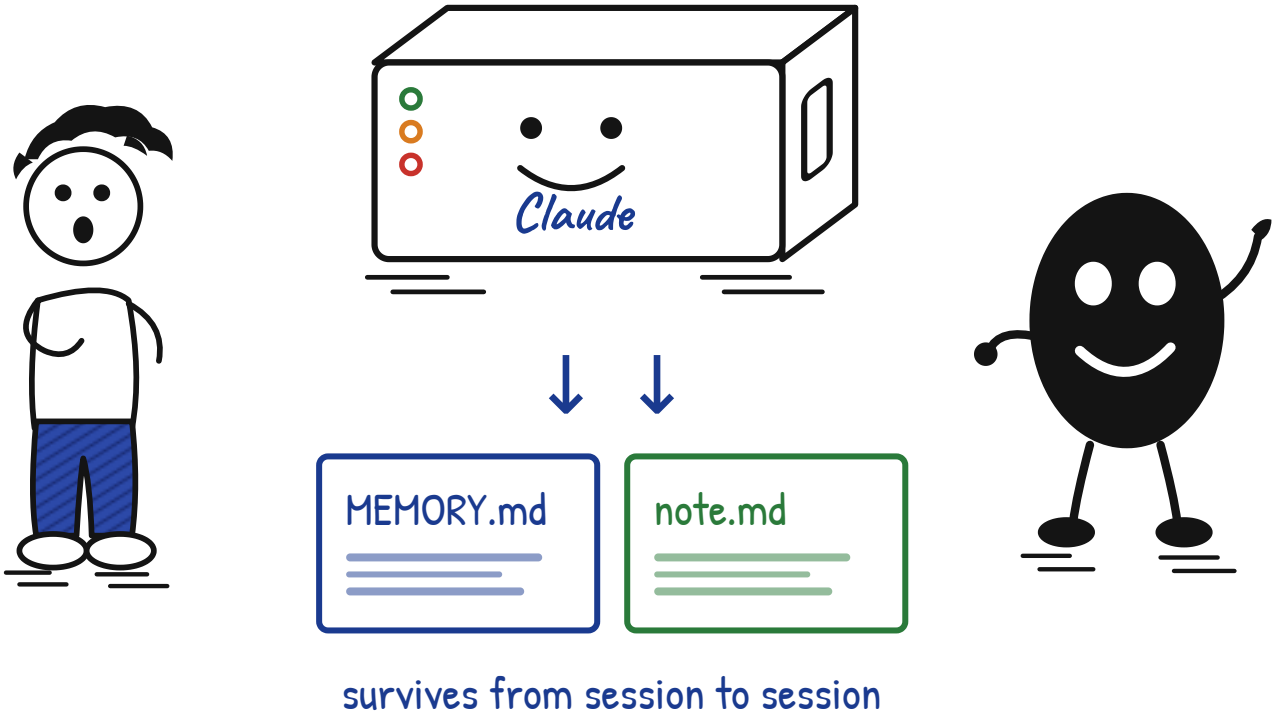
/rewind – back to state B



Does Claude remember anything beyond a single session?

Yes – that is what memory is for: small note files Claude writes itself and reads again the next time it starts.

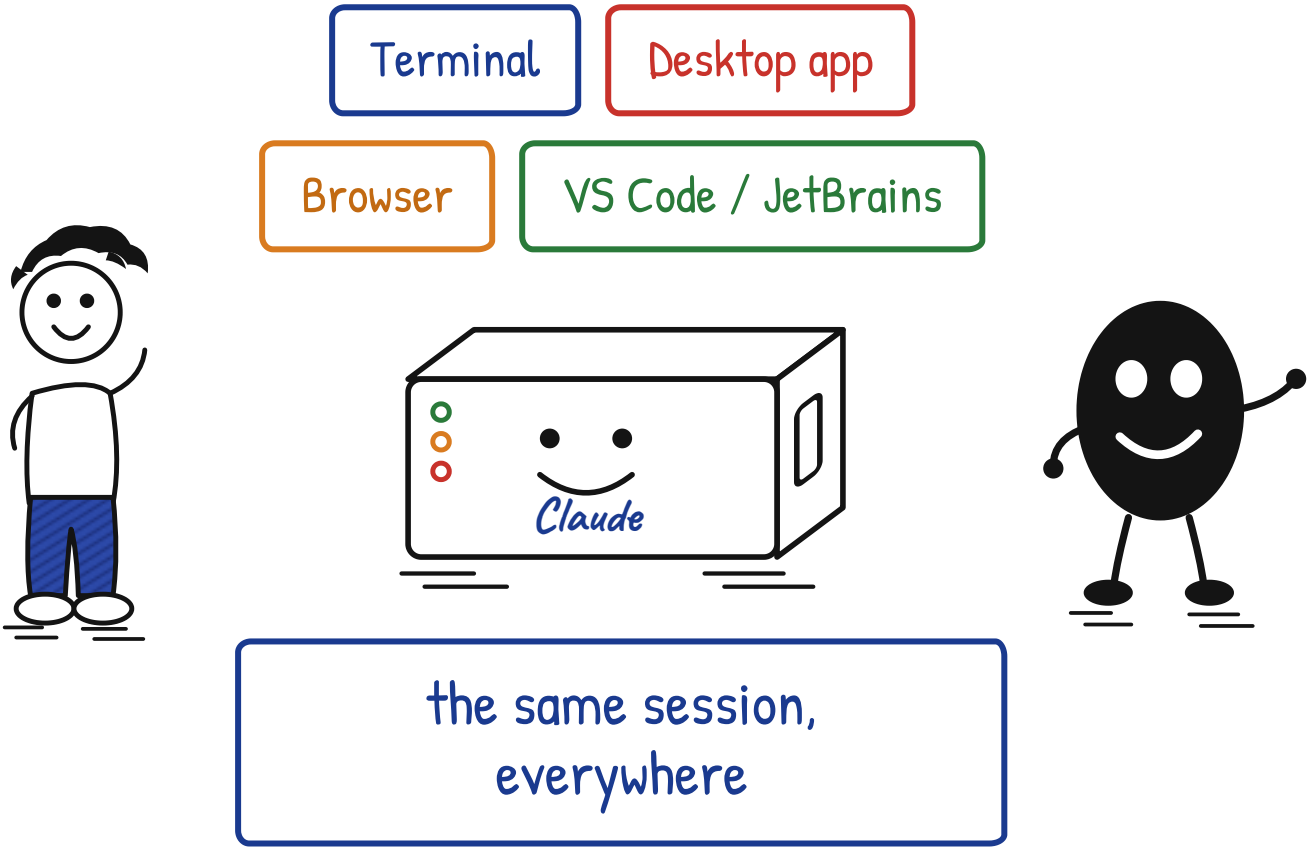
Your preferences, where a project stands, decisions that keep coming back. Unlike CLAUDE.md, Claude maintains this itself.



And all of this only runs in the terminal?

Not for a long time now. Claude Code comes as a terminal command, as a desktop app, in the browser at `claude.ai/code` and as an extension for VS Code and JetBrains.

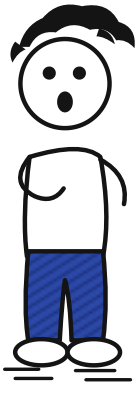
On top of that, agents that keep working in the background or in the cloud while you do something else.

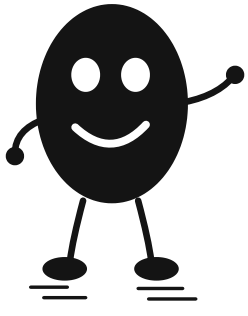


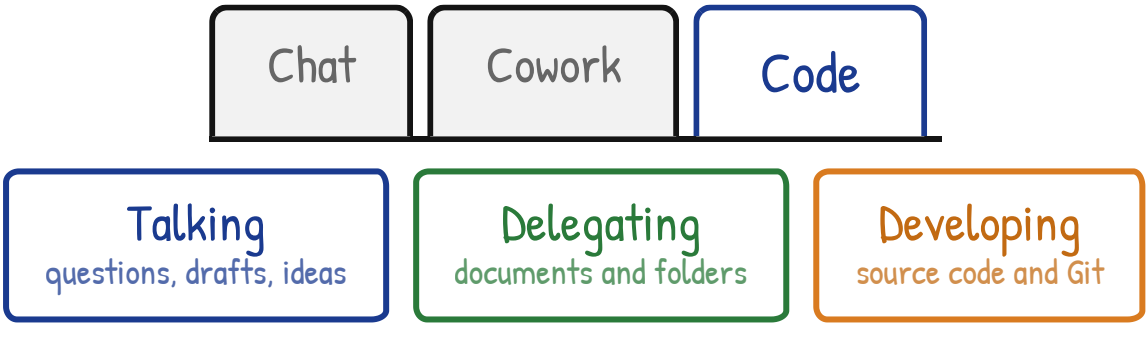
Is there a version of all this to click on?

Yes. The Claude desktop app has three tabs: **Chat** is the conversation, **Cowork** does office work, **Code** is Claude Code with a user interface.

It is available for Mac and Windows, and as a beta for Linux.



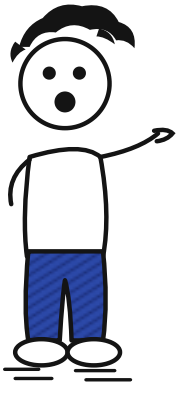


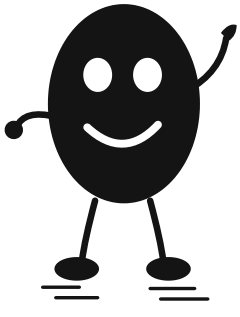


What can the app do that the terminal can't?

Several sessions side by side – each in its own Git worktree. That way they don't get in each other's way, even when they work on the same spot.

Plus editor, terminal, preview and the review of the changes in the same window. You can even kick off a session from your phone.





Session 1
its own worktree

Session 2
its own worktree

Session 3
its own worktree

↓

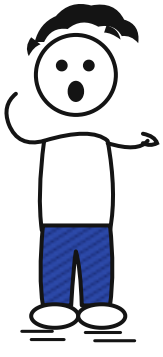
Editor | Terminal | Preview | Review changes

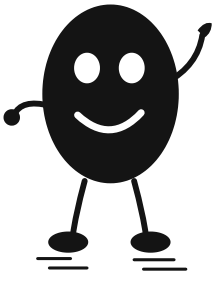
everything in one window

And what if a change runs through the whole project?

That is what `/batch` is for: Claude thinks the rebuild through once, then hands it out to many agents that start at the same time – each in its own working copy.

At the end each one puts up its own set of changes. You see them side by side and accept them one at a time.





think it through once, then split it up

↓

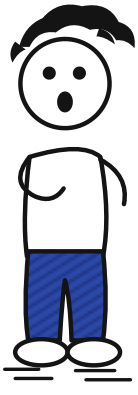
Agent 1	Agent 2	Agent 3	and so on
its own copy	its own copy	its own copy	up to thirty

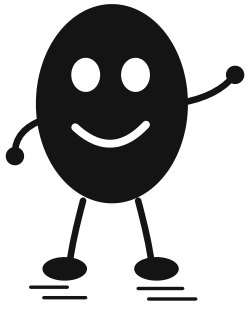
one set of changes each, to review

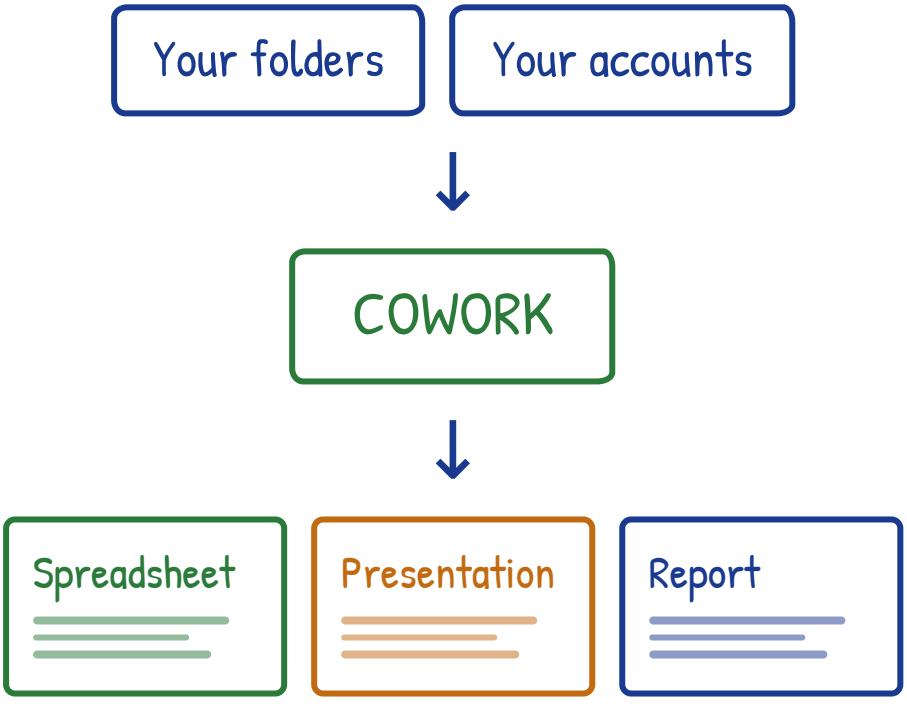
So what is Cowork, then?

The same machinery as Claude Code – only in the ordinary Claude app instead of the terminal, with nothing to set up.

You share folders, describe the goal, and Claude gets going: spreadsheets with real formulas, presentations, finished documents.



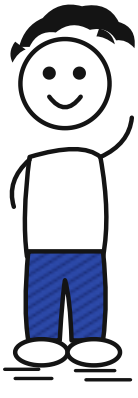


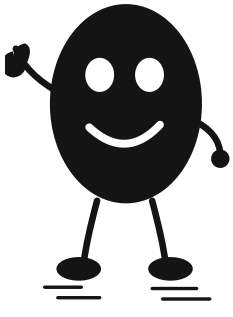


So what does it actually do?

Turn a folder full of receipt photos into an expense report. Write a report from a pile of notes. Tidy up a filing system. If need be it will drive Chrome for that – clicking, typing, filling in forms.

How much it may decide on its own is up to you.





Receipts → expense report

Notes → report

Tidy up the filing

and in the browser if that's what it takes

Asks every time
you approve everything

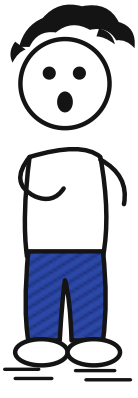
Decides for itself
with a safety net

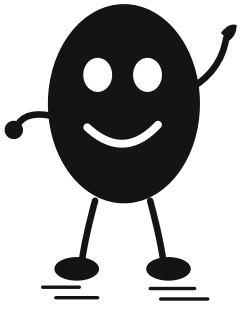
Just gets on with it
without asking

When do I use which?

The question is always how far Claude may go into your things. Cowork gets individual folders, runs sealed off and needs no setting up.

Claude Code gets the whole machine. Only here can you teach it tools of your own – and only here does it keep going when you are away.





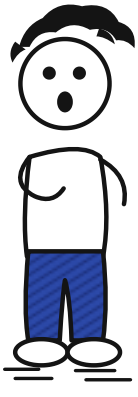
1 · In the conversation	nothing to set up – you upload, you download
2 · Cowork	desktop app: Claude works in your folders
3 · Claude Code	you are here: the whole machine, your own tools, runs without you

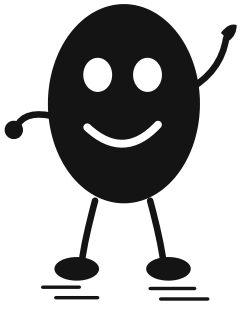
getting things done → Cowork · building tools that run by themselves → Code

And what if something should run regularly on its own?

Then you create a routine: task, project and accounts saved once – after that it runs without you.

And it does so in the cloud. Your machine may be switched off. You create it with `/schedule`, in the app or in the browser.





/schedule review the new pull requests daily at 9am

↓

Task | Project | Accounts

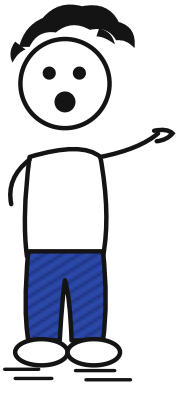
↓

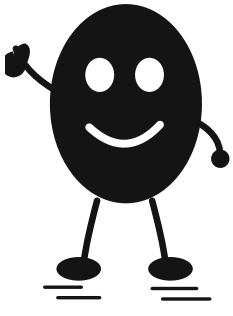
runs in the cloud – machine off

And what sets a routine off?

Three things, alone or combined: a schedule, a call from outside, or an event on GitHub.

The shortest interval is one hour. One-off appointments work too – `/schedule in two weeks …` The whole thing is still in preview.





Schedule
hourly to weekly

A call
from your own tools

GitHub
new pull request

↓

Routine runs

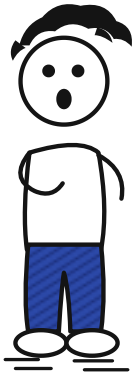
↓

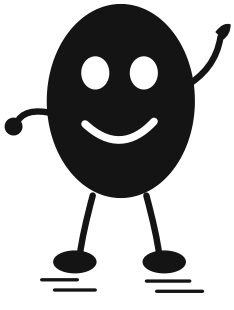
the result is waiting in the morning

And if it's only meant for the next hour?

Then use `/loop`. It repeats a task inside the open session – on your machine, with your files.

Leave out the interval and Claude picks it itself: short while something is happening, longer when things are quiet. Esc ends the loop.





`/loop 5m check whether the build has finished`

↻ ↻ ↻

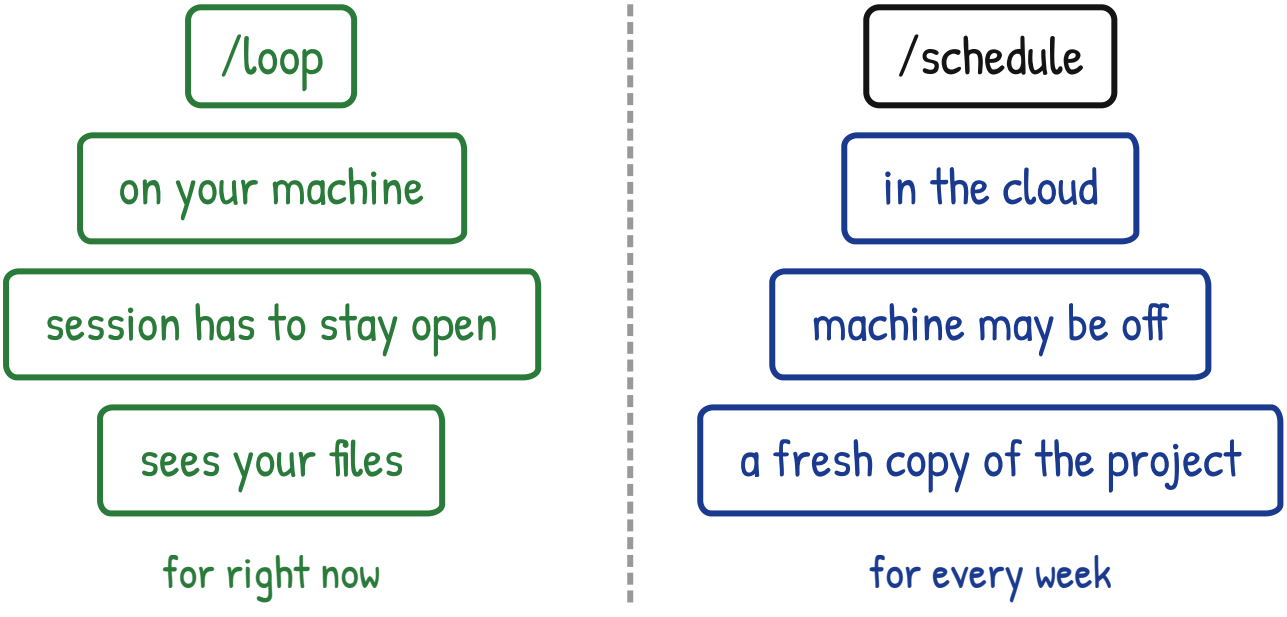
with an interval	without one	with no task at all
every 5 minutes	Claude decides	it tidies up by itself

runs only while the session is open – and ends after seven days

That sounds almost like a routine.

The difference is where it runs. The loop needs your session open and your machine switched on – in return it sees your files.

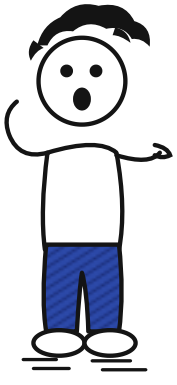
The routine runs in the cloud, without you. Rule of thumb: loop for this afternoon, routine for every Monday.

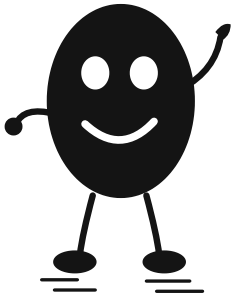


And what if I don't know yet what it should look like?

Then start with `/design`. Claude lays several drafts out as boards on a canvas – click them, move them, edit the text right there.

Once the direction is right, you hand it back to Claude Code, and that is where it gets built. Still an early preview.





describe it → look at drafts → adjust it yourself → have it built

↓

several proposals side by side

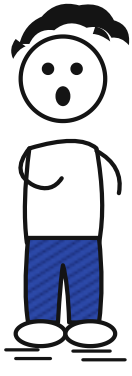
click instead of describe

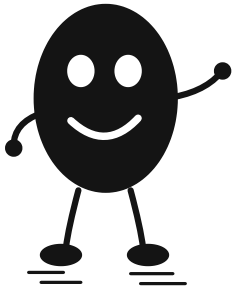
still an early preview – may change

And which model is actually doing the work?

As of August 2026: **Fable 5** is the strongest and built for long runs, **Opus 5** takes the heavy lifting, **Sonnet 5** the everyday work, **Haiku 4.5** the fast and cheap jobs.

Switch with `/model`. With `/fast` Opus answers more quickly – it is not swapped for a smaller model in the process.





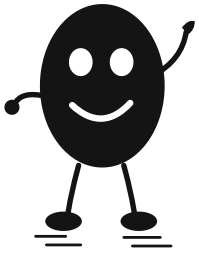
Fable 5	Opus 5	Sonnet 5	Haiku 4.5
long agent runs	the heavy jobs	everyday work	fast & cheap
1M context	1M context	1M context	200K context

switch with /model

And how thoroughly it works – can I say that too?

Yes, that is the second dial next to the model: `/effort`. First pick the model by how hard the task is, then the effort by how thorough you want it.

The default is **high**. Going down means faster and cheaper, going up means more thinking, more tool calls, longer runs.



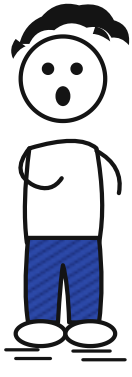
low	short, simple jobs – the fastest
medium	the thrifty middle ground
high	the default – for anything demanding
xhigh	agent runs lasting hours
max	everything it has, never mind the spend

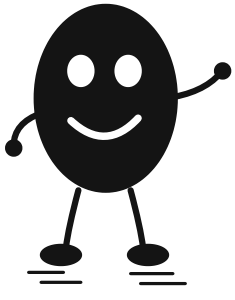
set it with /effort

And how do I best put it?

State the goal, not the individual steps – Claude finds the way itself. And say how you can tell when it is done.

And give the context: who it is for, what for, and what must not happen under any circumstances. The clearer the task, the fewer rounds.





rather not

"Make the table look nicer."

↓

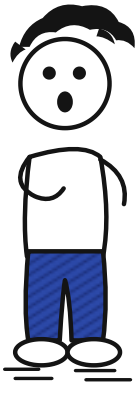
better

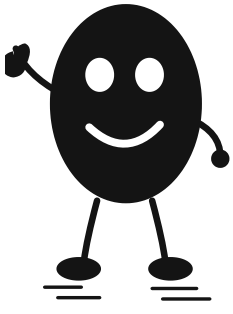
"The table goes to the board. Numbers right-aligned, totals row in bold, no colours. Check at the end that the totals add up."

Can I actually rely on it?

Usually yes – but Claude can also be convincingly wrong. Have it show you what it did instead of just believing it.

And whatever can be checked, let Claude check itself: run the tests, redo the arithmetic, name the sources.





Look at the changes
not just the summary

Run the tests
automatically if you can

Ask for sources
where does that number come from?

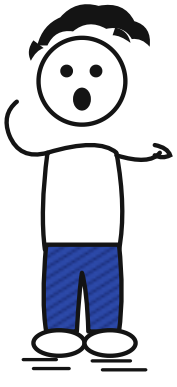
When in doubt, rewind
Esc Esc costs nothing

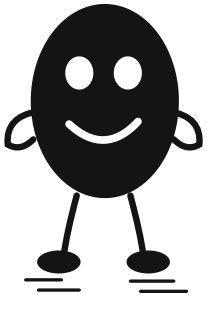
Trust is fine. Checking is faster than repairing.

And what if a file says "delete everything"?

Good question – that is exactly where the trap is. Claude reads web pages, e-mails, tickets and other people's source code. None of that is an instruction from you.

Instructions come from you, from your project rules and from what you allow. Everything else is material to read – which is why the permissions are not there to annoy you.





Your instruction
counts

CLAUDE.md
counts

Web page
material only

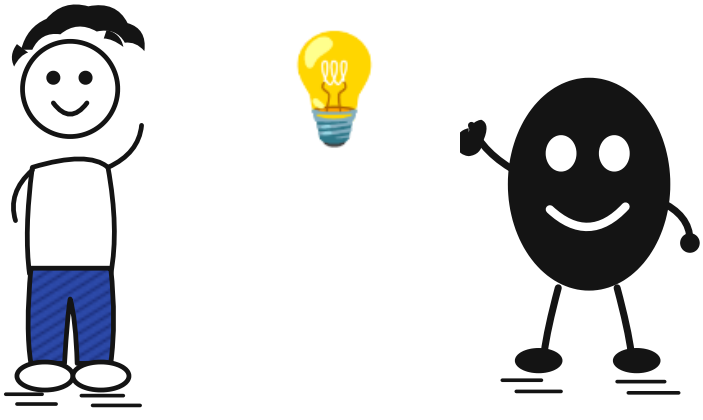
E-mail
material only

someone's ticket
material only

The further out Claude may reach, the shorter you keep the leash

Ohh. I think I've got it now.

Glad to hear it.



CLAUDE.md – the rules of your project
Skills – how a job is done properly
Plugins – all of that as one bundle
MCP – the connection to the outside
Hooks – run automatically on events
Subagents – helpers with a mind of their own
Context window – the working memory
Checkpoints & memory – rewind and remember
Cowork – the same inside your folders, without a terminal
Routines – run to a schedule in the cloud
Permissions – automatic today, stricter whenever you want
Effort – how thoroughly the work should be done
/loop & /batch – repeat it, or split it across many
/design – draft it first, then have it built