FROM FIRST LOGIN TO EXPERT

The Complete Guide to Claude. All of it.

Models · Prompting · Projects · Skills · Connectors · Plugins · Artifacts · Claude Code · Cowork

Arjit Lohani

20 PAGES · NO HYPE · KEEP IT OPEN WHILE YOU SET UP

CONTENTS

What's inside.

This guide runs in order of depth. Part I gets you oriented, Part II makes you fast, Part III is where most people never go, and Part IV is where Claude stops being a chatbot. Skip around freely, but if you are new, read pages 3 to 5 first. They change how you use everything else.

PART I · FOUNDATIONS

01

What Claude actually is.

Most people meet Claude as a chat box: type a question, get an answer. That is real, but it is the smallest version of the product. Claude today is a stack of seven connected pieces, and the people getting outsized results are simply using more of the stack.

The seven pieces

PIECE	WHAT IT IS	ONE-LINE JOB
Models	The AI engines themselves, in four sizes	The brain. Pick the right size.
Skills	Reusable playbooks Claude applies automatically	The training. It works your way.
Connectors	Live links into Gmail, Calendar, Drive, Slack and more	The hands. It reaches your tools.
Projects	Workspaces holding documents, instructions and chats	The desk. Context that stays put.
Plugins	Installable bundles of Skills, Connectors and commands	The kit. Workflows in one click.
Claude Code	An agent that works inside real codebases	The builder. Plain English to working code.
Cowork	A desktop app that works through files toward a goal	The colleague. You delegate, it finishes.

The mindset shift

The single biggest upgrade is not a feature. It is a reframe: **stop treating Claude like a search engine and start treating it like a capable new hire.** A new hire needs three things to be useful: training on how you work, access to the tools the job requires, and enough context to act without asking you fifty questions. The stack maps exactly onto those three needs. Skills are the training. Connectors are the access. Projects are the context.

> **THE TEST OF AN EXPERT**
>
> A beginner asks Claude questions. An intermediate user gives Claude tasks. An expert builds a setup where the same task never needs explaining twice. Everything in this guide moves you along that line.

PART I · FOUNDATIONS

02

The models: four sizes of brain.

Claude is a family of models. Same character, different capability and cost. The model picker at the top of every chat is the most consequential dropdown in the product, because your choice decides both answer quality and how fast you burn through your usage allowance.

MODEL	THINK OF IT AS	REACH FOR IT WHEN
Haiku	The sprinter. Fastest and lightest.	Summaries, quick questions, sorting and labelling data, high-volume repetitive jobs. Anything a human could do in two minutes.
Sonnet	The daily driver. Best balance of smart, fast and efficient.	Writing, coding, analysis, everyday business work. Roughly 80% of tasks belong here. This is your home base.
Opus	The specialist. Deep reasoning.	Genuinely hard problems: complex analysis, tricky code, long multi-step tasks. Powerful, but drains allowance fast.
Fable	The frontier. The newest, most capable tier, a step above Opus.	The work that actually deserves maximum intelligence. Availability and allowance depend on your plan.

How to choose, in practice

Start on Sonnet. Step down to Haiku when the task is clearly simple. Step up to Opus or Fable only when the task genuinely demands deep reasoning, or when Sonnet's answer to a hard problem feels shallow. Escalating a failed task to a bigger model is often cheaper than re-prompting the small one five times.

Two habits separate experts here. First, they match model to task within a piece of work: brainstorm and outline on Sonnet, then bring the hardest single decision to a bigger model. Second, they never leave a heavy model selected out of laziness. The dropdown resets your economics every time.

> **DURABILITY NOTE**
>
> Model names and versions change several times a year. The tier logic does not. Whatever the picker shows this month, there is a fast tier, a balanced tier, and one or two deep tiers. Map them to sprinter, daily driver, specialist, frontier and the strategy on this page keeps working.

PART I · FOUNDATIONS

03

Usage limits, and how to stretch them.

Paid plans meter usage in two layers: a session allowance that resets every five hours, and a weekly ceiling that applies across all models. Everything draws from the same pool, whether you are in chat, Claude Code or Cowork. A morning of heavy chatting leaves less for an afternoon of coding. Once you understand that tokens follow context, the rest is technique.

The six habits

- **Put reused documents in a Project.** Project files are cached, so asking about the same files again barely touches your limits. This is the single highest-leverage move on this page.
- **Batch your questions.** One message with five questions costs less than five messages, because every message re-processes the whole conversation history.
- **Front-load context.** Vague openers create expensive back-and-forth. Give the files, the constraints and the desired format up front, and get it right in one pass.
- **Start new chats for new topics.** A 100-message thread carries its entire history into every reply. Long conversations are the silent budget killer.
- **Match the model to the task.** Page 4. It matters more than every other habit combined.
- **Know your reset times.** Settings, then Usage, shows when your session and weekly windows refresh. Schedule heavy sessions right after a reset instead of hitting the wall mid-task.

> **Tokens follow context. Control the context, control the cost.**

WHEN YOU STILL HIT THE WALL

If you are consistently capped, that is usually a signal about workflow rather than plan size: a giant thread that should have been three chats, a heavy model doing light work, or documents pasted in repeatedly instead of living in a Project. Fix the workflow first, then consider the bigger plan.

PART II · WORKING WELL

04

Prompting fundamentals.

Prompting is not magic words. It is briefing quality. The same principles that make a good brief for a human contractor make a good prompt: context, constraints, and a clear picture of done.

The anatomy of a strong prompt

- **Role and situation.** "You are reviewing a supplier contract for a small cafe owner" beats "review this contract". Context changes which details matter.
- **The task, precisely.** One clear ask per prompt when quality matters. "Find the three biggest risks and explain each in plain English" beats "thoughts?"
- **Constraints.** Length, tone, format, what to avoid. "Under 300 words, no jargon, bullet the risks" saves a whole revision cycle.
- **The output shape.** Say what done looks like: a table, an email draft, a checklist, a document. Claude will match the shape you name.

Before and after

```
WEAK:  "Can you help with my presentation?"

STRONG: "I present to my leadership team Thursday: 10 minutes on
why we should adopt AI tools. Audience is skeptical about cost.
Draft a 6-slide outline. Slide titles plus 2 speaker points each.
Confident tone, no buzzwords, and slide 5 must address cost
directly with a simple payback framing."
```

The strong version is longer to write and dramatically cheaper to run, because it lands in one pass instead of six rounds of "actually, could you...". Experts spend their effort before the first reply, not after it.

> **THE ONE-PASS RULE**
>
> If you regularly need three or more follow-ups to get what you wanted, the prompt was missing context Claude could not guess. Write the follow-ups into your next first message. When the same correction appears twice in a week, that correction wants to become a Skill (page 9).

PART II · WORKING WELL

05

Advanced prompting.

Once the fundamentals are habit, four techniques account for most of the remaining gap between decent output and excellent output.

1 · Show, don't describe

One good example beats a paragraph of adjectives. Pasting a past report you loved and saying "match this structure and voice" outperforms any description of your style. Two examples, one good and one bad with a line on why, is even stronger.

2 · Ask for reasoning before answers

For anything with judgment in it, ask Claude to think it through first: "Walk through the trade-offs step by step, then give your recommendation." Reasoning first measurably improves the final answer on hard problems, and you get to audit the logic.

3 · Label the parts of a complex prompt

When a prompt contains multiple ingredients, mark them so nothing blurs together. Simple tags work well:

```
<background> the client, the deal, what has happened so far </background>
<task> draft the renewal email </task>
<rules> friendly but firm, mention the March price change once,
under 150 words </rules>
```

4 · Make it push back

Claude aims to be helpful, which can drift into agreeable. For decisions, invite the opposite: "Argue against this plan before you improve it" or "What would a skeptical CFO say?" You want a colleague, not a cheerleader, and one sentence buys that.

> **PERSONALISE THE DEFAULTS**
>
> Settings offers standing preferences (how you like answers formatted) and styles (tone presets). Five minutes there means every future chat starts pre-briefed. Preferences are prompting you only have to do once.

PART II · WORKING WELL

06

Projects: your permanent desk.

Every new chat starts with amnesia. A Project fixes that. It is a workspace that holds three things: uploaded documents, standing instructions, and every conversation that belongs to the same body of work. Start a chat inside the Project and Claude already knows the brief, the files, and the rules. No re-uploading, no re-explaining.

Setting one up properly

- **Create one Project per ongoing body of work.** A client, a job search, a product, a course. Not one giant Project for everything; the knowledge blurs.
- **Load the documents that keep coming up.** The brief, the style guide, the price list, the resume. If you have pasted it twice, it belongs here.
- **Write custom instructions.** This is the Project's standing brief: who you are in this context, what good output looks like, what to always avoid. It applies to every chat inside.
- **Keep related chats inside.** The point is that context accumulates in one place instead of scattering across your history.

The caching advantage

Project documents are cached. When a chat references them, you are not paying full price to re-read them each time, which makes Projects the single best way to stretch a paid plan (page 5). Heavy document work outside a Project is the most common expensive habit in the product.

A GOOD CUSTOM INSTRUCTION, WORD FOR WORD

```
"This project is for client work with Acme Retail. I am their marketing consultant. Always write in Australian English, keep drafts under one page, and flag anything that conflicts with the brand rules document before proceeding."
```

Specific, short, and it saves those four sentences from every single future prompt.

PART III · THE POWER LAYER

07

Skills: teach it once.

Here is the pattern Skills exist to kill: you have a prompt you paste every week. The tone rules, the format, the ten steps. Every paste is you re-training your AI from zero. A Skill moves that knowledge out of your clipboard and into Claude itself.

What a Skill is

A Skill is a small document (plus any reference files it needs) where you write down how you want a job done, once. Claude scans its available Skills at the start of a task, recognises when one matches, and follows it automatically. You do not invoke it. You do not remind it. You just ask for the report, and the report comes out your way.

Skill vs prompt vs Project: which one?

IF THE KNOWLEDGE IS...	IT BELONGS IN...
One-off	The prompt. Just say it.
About one body of work	A Project's custom instructions and files.
About how a TYPE of task is done, anywhere	A Skill. "How I write reports" applies in every project, every chat.

Signals you need one

- You keep a notes file of prompts you reuse. That file is a Skill waiting to be formalised.
- You give the same correction twice in one week ("shorter", "British spelling", "always include next steps").
- A task has a checklist you currently hold in your head, and quality drops when you rush it.
- You want a teammate, or future you, to get identical output without learning your prompting habits.

> **Prompting is a conversation. A Skill is infrastructure.**

PART III · THE POWER LAYER

08

Building your first Skill.

A Skill has two jobs: help Claude recognise when to use it, and tell Claude how to do the work. Structure yours around those two jobs and it will fire reliably.

The anatomy

NAME + DESCRIPTION	One clear sentence about what this Skill covers and when it applies. This is how Claude decides to use it, so name the trigger words a real request would contain.
THE PROCESS	Numbered steps, in order. What to do first, what to check, what to produce.
THE RULES	Always do / never do. Tone, length, formats, forbidden phrases, required sections.
EXAMPLES	One gold-standard sample output. If helpful, one bad example with a line on why it fails.

A complete worked example: the Weekly Client Update Skill

```
DESCRIPTION: Writing weekly status update emails to clients.
Use whenever asked for a client update, status email, or WIP summary.

PROCESS:
1. Open with one sentence on overall status: on track, at risk, blocked.
2. Three sections: Done this week / In progress / Needs your input.
3. Close with the single next milestone and its date.

RULES: Under 200 words. No jargon. Never promise dates we have not
agreed. Australian English. Subject line format: "Update · [Client] · [Date]".

EXAMPLE: [paste your best past update email here]
```

That is the whole thing. Twenty minutes to write, and every future update is one line: "write this week's update for Acme, here are my notes."

ITERATE LIKE AN EXPERT

First versions misfire. When output misses, do not fix the output, fix the Skill: add the rule that was missing, sharpen the description's trigger words. Three rounds of this and it becomes boringly reliable, which is the goal.

PART III · THE POWER LAYER

09

Skill patterns that work.

Skills cluster into a handful of patterns. Steal whichever matches your week.

The Formatter

Encodes a document type: your report template, your proposal structure, your meeting-notes layout. Rules plus one gold example. This is the easiest first Skill and the one with the most daily payoff.

The Checklist Reviewer

Encodes judgment: "review contracts against these 12 points", "check job applications for these red flags", "QA every spreadsheet for these five errors". Claude applies the full checklist every time, including the items a rushed human skips.

The Brand Voice

Encodes how your writing sounds: sentence length, warmth, banned words, how you open and close. Feed it three of your best pieces as examples. Works across posts, emails and documents at once.

The Router

Encodes a decision: "when a job ad arrives, decide which of my three resumes fits, then tailor the cover letter to match." One Skill, an entire repeatable workflow. Routers pair beautifully with Connectors, next page.

The Translator

Converts between formats you use constantly: meeting transcript to action list, data dump to executive summary, technical detail to client-friendly explanation. Define both ends precisely and the middle takes care of itself.

> **THE COMPOUNDING EFFECT**
>
> One Skill saves minutes. A library of eight changes your job. Experts build one Skill the first time a task repeats, and within a quarter their Claude behaves like a trained team member while everyone else's behaves like day one. The moat is not access to AI. Everyone has access. The moat is accumulated training.

PART III · THE POWER LAYER

10

Connectors and MCP.

On its own, an AI can only talk. Connectors give it hands. A Connector is a live, permissioned link between Claude and a tool you already use: Gmail, Google Calendar, Drive, Slack, Notion, Canva and a fast-growing directory of others. Connected, Claude can read the email thread, check your week, draft the reply and save the summary, instead of you ferrying text between apps all day.

MCP in sixty seconds

The technology underneath is MCP, the Model Context Protocol. It is an open standard, which matters for one practical reason: any company can build a connector for its product without waiting for Anthropic, which is why the directory keeps growing monthly. MCP is doing for AI tools what USB did for hardware: one plug shape, everything fits.

Control and safety, plainly

- **You approve each connection once**, through the tool's own sign-in. Claude never sees your password.
- **Claude asks before consequential actions.** Reading is routine; sending, deleting and posting get confirmed with you first.
- **Scope to what you use.** Connect the tools that carry your real work. Disconnect experiments you stopped using; fewer live connections is simply cleaner.
- **Treat connected content as data, not orders.** A good mental model, and the one Claude itself uses: an instruction inside a random email should never be treated like an instruction from you.

> **SETUP, ONCE**
>
> Settings, then Connectors. Connect your top three tools now, usually mail, calendar and file storage. That trio alone covers most of the workflows on the next page.

PART III · THE POWER LAYER

11

Connector workflows.

The value is not any single connector. It is the chain: one instruction that crosses several tools. Real patterns people run daily:

The Morning Triage

```
"Check my unread email from the last 24 hours. Summarise what
actually needs me, draft replies for the two most urgent, and
list anything that clashes with today's calendar."
```

Mail plus Calendar. Ten minutes of inbox anxiety becomes one review pass.

The Meeting Wrap

```
"Here are my raw notes from the Acme call. Turn them into clean
minutes, save them to the Acme folder in Drive, draft the
follow-up email to their team, and put a reminder in my calendar
to chase the contract Friday."
```

One instruction, four systems, zero copy-paste. This is the moment Connectors click for most people.

The Research Pull

```
"Find last quarter's pricing sheet in Drive, compare it with the
new supplier email in my inbox, and table what changed."
```

The Content Pipeline

```
"Take the final post text from my Notion drafts page and create
a matching visual in Canva using my brand template."
```

CHAIN + SKILL = AUTOMATION

Now combine layers. A "meeting minutes" Skill defines the format once; the Connector chain moves the result into Drive, email and Calendar. You have built a real automation without writing a line of code. This layering is the heart of expert-level Claude.

PART III · THE POWER LAYER

12

Plugins: workflows in a box.

By now you can see the assembly required: write the Skills, connect the tools, wire them together. It works, and it takes effort. A Plugin skips the assembly. It packages Skills, Connectors and ready-made commands for a whole job into one installable kit.

What arrives when you install one

Take a finance plugin as the example. One install and Claude suddenly carries month-end close checklists, reconciliation playbooks, journal-entry templates and reporting commands. An engineering plugin ships code review standards, incident response runbooks and standup formats. The domain knowledge someone senior would spend weeks teaching a new hire, pre-loaded.

When to install vs when to build

SITUATION	MOVE
A standard job in a standard domain	Install the plugin. Finance close is finance close; do not rebuild it.
Your process is genuinely your edge	Build your own Skills. Your voice, your checklist, your router.
Both	Install the plugin for the skeleton, then add your own Skills on top. They stack.

The team angle

Plugins are how AI setups spread through a team. One person encodes the department's way of working, everyone installs it, and suddenly the whole team's Claude behaves consistently. This is the same shift that happened when teams moved from personal spreadsheets to shared templates, and it rewards whoever builds the template first. That person becomes the reference point for how the team uses AI.

> **Install a workflow the way you install an app.**

PART III · THE POWER LAYER

13

Artifacts and file creation.

Ask Claude for anything substantial (a document, a webpage, an app, a diagram) and it opens an Artifact: a live workspace beside the chat where the thing itself takes shape. You iterate on the artifact in place ("make section two shorter", "change the colour scheme") instead of scrolling through a conversation for the latest version.

What Claude can produce as real files

- **Documents:** full Word files with proper formatting, contents pages and styles. Reports, proposals, letters.
- **Presentations:** complete PowerPoint decks, structured slide by slide.
- **Spreadsheets:** Excel files with working formulas, not just static tables.
- **PDFs:** polished, designed documents. The guide you are reading was produced this way.
- **Interactive pieces:** working web tools, calculators, quizzes, dashboards and games you can use immediately and share.

The upgrade most people miss

Artifacts can themselves contain AI. Claude can build you a small app that calls Claude: a writing coach tuned to your rules, a flashcard tutor for your course, a customer-reply generator for your store. You describe the tool; it builds the tool; the tool then works on demand without you re-prompting. Building small AI-powered utilities used to be a developer project. It is now a conversation.

> **PROMPTING FOR ARTIFACTS**
>
> Name the deliverable and the audience: "a one-page PDF checklist my new baristas can follow" beats "make a checklist". For anything long, approve an outline first, then let it build. Structure approved early is ten revisions saved late.

PART IV · EXPERT TERRITORY

14

Claude Code: the builder.

Claude Code is not a chat window. It is an agent that works inside a real codebase, from your terminal, VS Code, JetBrains or the desktop app. You describe the goal in plain English; it reads the project, edits the files, runs the tests, and shows you exactly what changed before anything is final.

```
$ claude
> fix the failing auth tests and tidy login.js
  reading project ............... 214 files
  editing src/login.js .......... done
  running test suite ............ 42 passed
  ready to commit. review changes?
```

Why it lands differently than chat

Pasting code into a chat gives Claude a keyhole view. Claude Code sees the whole room: every file, the structure, how pieces call each other. That context is why it can safely make changes that span a dozen files, and why its fixes tend to respect how your project actually works instead of how projects generically work.

The review loop keeps you in charge

It proposes; you approve. Every change is shown as a diff before it lands, tests run before commits, and you can interrupt or redirect at any point. The right mental model is a fast, tireless junior developer with you as the reviewer. Trust builds the same way it does with a human junior: small tasks first, then bigger ones as the track record grows.

PRACTICAL NOTES

Included with paid plans, drawing from the same usage pool as chat. Start it inside a project folder so it sees your real files. Big refactors burn allowance quickly, so scope the first tasks tight: one bug, one file, one feature.

PART IV · EXPERT TERRITORY

15

Claude Code without being a developer.

The quiet story of Claude Code is who else is using it. You do not need to write code to direct it, and "I can't code" now mostly means "I haven't watched this run yet." What non-developers actually build with it:

Small tools that remove daily friction

A script that renames and files the month's invoices. A tool that merges four spreadsheets into the weekly report. A folder-watcher that converts every incoming file to PDF. These are afternoon projects now: describe the annoyance, review what it builds, run it forever.

Fixing the scripts you inherited

Every workplace has a macro or script someone wrote years ago that nobody dares touch. Point Claude Code at it: "explain what this does in plain English, then fix why it breaks on files with commas in the name." Understanding legacy scripts is genuinely one of its strongest use cases.

Prototypes that make ideas real

"Build a simple page where staff can log maintenance issues, saved to a spreadsheet." A working prototype in an hour changes a meeting from abstract debate to concrete reaction. You are not shipping production software; you are making the idea touchable.

How to direct it well

- **Describe outcomes, not methods.** "I need X to happen whenever Y" is a perfect instruction. Let it choose the how.
- **Ask it to narrate.** "Explain what you changed like I'm not a developer" after each step. You stay in control of what you understand.
- **Keep asking why.** Six months of this and you will half-know how to code, as a side effect of getting your work done.

PART IV · EXPERT TERRITORY

16

Cowork: delegation, not conversation.

Cowork is Claude Code's power with none of the terminal. It is a desktop app built for everyone else: analysts, marketers, operations, admin. You point it at a folder, describe the outcome, and it works through the job step by step: reading documents, cleaning spreadsheets, drafting the output. You watch the steps and can stop or redirect at any point.

The shape of a Cowork task

```
TASK: "Clean this quarter's expense files and draft the summary"

  ● scan 34 files in /expenses
  ● normalise 3 spreadsheets
  ● flag 6 anomalies for review
  ○ draft summary.docx
```

Notice what you did not do: open a single file, explain a format twice, or copy anything anywhere. You defined done, then reviewed the work.

What it changes about your week

Chat compresses tasks that take minutes. Cowork compresses the tasks that eat afternoons: the folder of fifty documents that needs summarising, the messy export that needs restructuring, the research that needs pulling into one brief. The unit of delegation moves from "answer this question" to "finish this job", and that is the real difference between an assistant and a colleague.

Delegating well

- **Define done, concretely.** "A one-page summary per client, saved in the same folder, flag anything unusual" is a complete delegation.
- **Give it the whole folder.** Context is the fuel. It will find what matters.
- **Review the first run closely, then loosen.** Same trust curve as a new team member.
- **Pair it with your Skills.** Your formats and rules apply here too. Trained plus delegated is the endgame.

> **Chat answers questions. Cowork finishes tasks.**

PART IV · EXPERT TERRITORY

17

The expert operating system.

Everything in this guide compounds into a simple loop you run for any recurring work: **capture context in a Project, encode the method in a Skill, give it reach with Connectors, delegate the volume to Cowork or Code, and spend your own hours on judgment.**

A realistic expert day

MOMENT	WHAT RUNS
8:40	Morning triage chain: inbox summarised, two replies drafted, calendar clashes flagged. (Connectors, page 13)
10:00	Client report requested. The Formatter Skill produces it in house style, first try. (Skills, pages 9 to 11)
11:30	Hard pricing decision. Escalated to a deep model with "argue against me first". (Models, page 4; prompting, page 7)
14:00	Folder of supplier documents handed to Cowork: "one-page comparison, flag contract risks." (Page 18)
16:00	Twenty minutes in Claude Code improving the invoice-renaming tool. (Pages 16 to 17)
16:50	One Skill updated with today's correction, so tomorrow starts smarter than today.

The seven mistakes that keep people at beginner level

- One endless mega-chat instead of clean chats inside Projects.
- Heavy models on light tasks, then complaining about limits.
- Re-pasting the same instructions instead of writing the Skill.
- Never connecting tools, then manually ferrying everything Claude produces.
- Accepting first drafts of prompts that deserve a brief.
- Treating agreement as accuracy. Ask for pushback on anything that matters.
- Setting it all up once and never iterating. The setup is a garden, not a statue.

THE 30-DAY PATH

Week 1: model discipline plus one Project with real documents. **Week 2:** your first two Skills, a Formatter and a Reviewer. **Week 3:** connect three tools and run the meeting-wrap chain. **Week 4:** one full delegation to Cowork, one small tool in Claude Code. That is the whole ladder from this page to expert.

That's the whole stack. Now go build yours.

You now know more about Claude's full stack than the vast majority of its users. The gap between reading and expertise is about thirty days of the loop on page 19. Start with one Project and one Skill this week; the rest compounds on its own.

KEEP IN TOUCH

Arjit Lohani writes practical, no-hype guides to AI tools and the systems around them.

Found this useful? Connect and follow on LinkedIn for the next field guide, and share this PDF with the colleague who is still re-pasting the same prompt every morning.

THE CLAUDE STACK · FIELD GUIDE · 2026 EDITION

Product details such as model names and plan limits evolve; the methods in this guide are built to outlast them.