Claude AI

Documentation — Manual — EN

1 Introduction

This document is a translation in substance. The reference is the original document in French; extensions and changes are always made there.

This manual is written for anyone who wants to use **CLAUDE** productively — to learn a subject, write documentation, analyse data or structure a project. It assumes no prior knowledge of *artificial intelligence*.

CLAUDE is not a search engine. It is a collaborator that can reason, rephrase, structure and adapt — provided it is given clear direction. This manual gives you the habits that get the most out of every session.

Note: *This manual is the companion to the Kit glossary, which defines every technical term used here. When a word is unclear, look there first.*

2 Understanding Claude before you start

2.1 What Claude is — and what it is not

CLAUDE is a *large language model* developed by **ANTHROPIC**. It understands and produces text in French, English, German and many other languages. It can read attached files — text, PDF, images — write code, draft structured documents and reason about complex problems.

It does not remember one session in the next, except through its *persistent memory*. It can be wrong, particularly on precise facts, recent dates or figures, and it usually says so itself when it is unsure.

It can, on the other hand, consult the web when a question calls for it — a recent fact, a price, a news item. That ability does not remove the need to check what matters: see §7.3.

💡 *What **CLAUDE** does best: summarising, structuring, drafting, explaining, rephrasing, comparing, planning. It excels as soon as the task is well defined and given its context.*

2.2 The context window — the session's working memory

Everything written in a *conversation* — your messages, the replies, the attached files — occupies the *context window*. That window is limited, and its size is expressed in units called *tokens*. The longer a *conversation* runs, the closer it comes to the limit.

When the limit is reached, the *conversation* is not cut off: it is summarised. The oldest exchanges are condensed into a synthesis that preserves the decisions and the state of the work, but loses the detail of the wording. That is good news — nothing disappears abruptly — but the fine grain degrades.

The practical consequence: for a long piece of work, opening a new *conversation* with an explicit summary of the decisions taken beats carrying on indefinitely in a session where the first instructions survive only in summary form.

Note: *A useful sign: if you find yourself restating a rule already set earlier in the same session, its detail has usually been condensed. That is the moment to close and start again cleanly.*

2.3 Choosing the right model

ANTHROPIC publishes several models side by side, each striking a different balance between power, speed and cost. The names and *generation* numbers change several times a year; the tiers themselves stay stable.

Tier	Usage profile	Note
Light tiers	Near-instant answers, minimal cost. Rephrasing, short questions, simple extraction.	The fastest
Balanced tier	The default working *model*. Drafting, structuring, everyday analysis, script *generation*.	Recommended for the Kit
Powerful tier	Deep reasoning, long agentic tasks, complex software engineering.	Keep for hard tasks
Top tier	The most advanced capabilities, sometimes with restricted access or extra safeguards.	Availability varies

The practical rule fits in one sentence: stay on the balanced tier by default, step up when a task resists, step down for simple repetitive volume.

Note: *This manual deliberately names no model. The range moves too fast for any list to stay correct, and an out-of-date table misleads more than it helps. For today's range, consult **ANTHROPIC**'s official documentation or the model selector in the interface, which describes each option in a line.*

3 First steps — opening a productive session

3.1 Choosing the right workspace

Two workspaces exist side by side: free conversations and projects. For any recurring work, or work tied to a specific domain, use a project.

A project gathers conversations that share the same reference files and the same *memory*. In the Kit, each documentation domain has a project of its own.

💡 *Always open a new conversation inside the right project. A conversation outside a project has neither the shared files nor the project memory.*

3.2 The opening prompt — setting the course from the start

The first message of a session is the most important one: it frames the whole *conversation*. A good opening *prompt* carries three things — the *context*, meaning who you are and what you are working on; the goal, meaning what you want to produce or learn; and the constraints of format, language, length or style.

```
Good evening. I'm working on the documentation for my home
automation system. For this session I want to write the chapter on
scenario management. Start by giving me an outline in four to six
sections, then we'll take each section one at a time.
```

A *prompt* of this kind is far better than a vague request: it states the expected deliverable, sets a working rhythm, and lets the answer be adjusted straight away.

3.3 Attaching files — enriching the context

CLAUDE can read attached files — Word documents, PDFs, images, text files. Attaching an existing file saves you copying its contents and lets the work rest directly on your real data.

For permanent reference files — stylesheets, guides, prompts — use the *project files*: they are available in every *conversation* without having to be attached each time.

Note: *Never attach a Word document to the project files expecting it to keep its formatting. The platform extracts the text from it. To modify an existing document, the binary file has to be dropped into the current conversation.*

4 Working in iterations

4.1 The principle of iteration

Working with **CLAUDE** is not a straight line. The final result rarely arrives on the first attempt, and that is not the aim. The natural process is to ask for a first version, judge it, correct or extend it, ask again. Each *turn* sharpens the result.

This holds for every kind of task — writing, script *generation*, planning, solving a technical problem. Move from the general to the particular: outline, then structure, then content, then detail.

💡 *Recommended method: ask for the outline, approve or adjust it, work through each section one at a time, consolidate at the end of the session. This produces far better results than a single all-in-one request.*

4.2 Framing an effective correction

When an answer misses the mark, the quality of your correction determines the quality of what follows. Three typical formulations.

- **Substance correction:** "This paragraph is too technical for my audience. Rewrite it, drop the jargon and keep only what is practically essential."
- **Structure correction:** "Points 3 and 4 overlap. Merge them into one and reorganise."
- **Tone correction:** "The tone is too formal. Rewrite it plainly, as if you were speaking to an experienced technician rather than to a board."

Avoid vague corrections of the "this isn't good, do it again" kind. You need to know why it is not good and in which direction to go.

4.3 Breaking down complex tasks

For an ambitious project — a thirty-page guide, the analysis of a full architecture, a series of documents — do not ask for everything in a single *instruction*. Break it into steps and approve each one before moving to the next.

4.4 Regenerating or correcting

Regenerating produces a new version without changing your last message. It helps when the form misses but the substance was on the right track.

If, on the other hand, it was your own *instruction* that was ambiguous or incomplete, write a new corrective message instead: regenerating would otherwise produce the same kind of result.

5 Managing memory over time

5.1 The three levels of memory

To work effectively on long projects, it helps to understand the three levels of *memory* available.

Level	Duration	Contents	Management
Working memory	Current session	The whole history of the active *chat*	Automatic
Persistent memory	Between sessions	Key facts drawn from conversations, kept separate per project	Viewable and editable
Project files	Permanent	Uploaded reference documents	Manual — by you

5.2 Persistent memory — understanding it and steering it

Important information is extracted automatically from your conversations and kept as notes. It is fed back into your future sessions, which gives the impression of continuity from one session to the next.

These notes can be viewed and edited in the settings. If something is wrong or out of date, saying so is enough: "update your *memory*, I now work with version 4.4, not 4.3".

Note: *Often misunderstood: memory is kept separate per project. Each project has its own space, and what was said in one is not known in the other. This is deliberate — it stops a professional context from spilling into a personal one — but do not count on continuity between projects.*

Note: *Memory is also switched off in private browsing. Those conversations leave no trace and do not appear in the history.*

5.3 Project files — structured memory

Files uploaded to a project are available in all of its conversations. It is the most reliable way to share rules, references and templates.

In the Kit, these files include the project *prompt* that carries the working rules, the stylesheets, the initialisation and update guides, and the *file inventory*. Together they form the project's permanent *memory*.

Note: *After every update to a project file, upload the new version and delete the old one. The version present is always the one that counts — two versions of the same file produce contradictory rules.*

6 Producing a document with the Kit

6.1 Overview of the production flow

The Documentation Kit rests on a six-step flow that produces consistent, professional documents which can be regenerated at any time.

Step	Action	Deliverable
1 — Framing	Define the subject, the audience and the structure wanted	Approved outline
2 — Content	Write section by section, in iterations	Text approved section by section
3 — Script	*Generation* of the production script	Script ready to run
4 — Upstream check	Static inspection of the script before it runs	Green light from the *linters*
5 — *Generation*	Running the script — production of the document	Timestamped document
6 — Validation	The *validator*, then a visual check	Deliverable document

Steps 4 and 6 are automatic and blocking: dedicated tools inspect the script before it runs, then the document produced before it is delivered. A failed check stops the chain — no file is delivered until the whole chain has passed.

💡 *This safety net has a pleasant consequence: a delivered document is structurally correct. Your proofreading can concentrate on the substance rather than the layout.*

6.2 The project prompt — the silent conductor

Every Kit project has a *prompt* file uploaded among its files. That document carries the permanent rules applied throughout the project — which *stylesheet* to use, naming conventions, language, author, expected behaviour.

The file is read automatically whenever a *conversation* opens in the project. You do not have to repeat these rules in every session.

💡 *If a rule seems forgotten in the middle of a long session, a short reminder is enough: "we use the Kit naming convention". Recalibration is immediate. It is also a sign that the session is getting long — see §2.2.*

6.3 File naming convention

Every Kit file follows a strict convention: a prefix, a category, a subject, then a *timestamp* in brackets and the extension. The *timestamp* is produced automatically by the script and reflects local time in Luxembourg.

A document published in several languages also carries a language code between the subject and the *timestamp*. The full detail is in Naming Convention.

Note: *Underscores are not allowed in file names, with the exception of the Kit tools, which follow a separate convention. The brackets around the timestamp are compulsory. The time is never typed by hand.*

6.4 Sequential delivery — one file at a time

The Kit follows a protocol of *sequential delivery*: each file is generated, validated and presented before moving to the next. This way of working avoids cascading errors and guarantees that every deliverable is correct before it joins the whole.

If several files are to be produced in one session, the full plan is announced first, then the files are delivered one by one in the agreed order.

7 Going deeper into a subject — learning with Claude

7.1 An adaptive tutor

Beyond producing documents, **CLAUDE** is an effective learning partner. It can explain a concept simply and then in progressively greater detail, answer follow-up questions, offer analogies, and set exercises or practical cases.

The key is to state your starting level and your goals. The better the *context* is known, the more relevant the explanations.

```
I'm new to virtual networks. First explain the concept in five
lines for a non-specialist, then give me a concrete example using
a home network.
```

💡 *Ask to be questioned rather than lectured at. This dialogue mode speeds up learning and keeps you engaged.*

7.2 Structuring a learning session

To learn a technical subject effectively, a typical session unfolds in five stages.

- Ask for an overview of the subject, in five to ten key points.
- Identify the points that remain unclear and ask for targeted clarification.
- Ask for one or two concrete examples tied to your real *context*.
- Test your understanding by rephrasing it yourself and asking to be corrected.
- Ask for a structured summary to keep as a reference.

7.3 Checking critical information

CLAUDE can be wrong, particularly on precise facts, software versions, legal references or recent events. For any content that will be published, shared or used in production, check against primary sources: official documentation, institutional websites.

This holds even when a web search has been made. A source found online is not necessarily reliable, and two sources can contradict each other. Asking where a piece of information comes from is always legitimate.

When there is doubt, it is usually flagged — "I'm not certain", "to be checked", or by an offer to search. Do not ignore those signals.

8 Closing a session methodically

8.1 Why close deliberately

Persistent memory holds facts, not the thread of an argument or the detail of the decisions taken. Without an explicit close, you risk losing rules adopted along the way, or emerging ideas that have not yet been written down.

Five minutes of closing let the next session start far more efficiently.

8.2 The session summary

At the end of a session, ask for a structured summary of the decisions taken, the files produced and the points left open. That summary can be kept in a tracking file or serve as the starting point for the next opening *prompt*.

```
Before we close, give me a summary of what we produced today, the
```

```
important decisions taken, and what is left to do.
```

8.3 Open points

During a session, ideas come up that there is no time to deal with. Ask for a list of open points to be kept as the session goes along, and presented at the end.

These points feed directly into the opening *prompt* of the next session and create a natural continuity, even without shared *working memory*.

8.4 Proactive suggestions

Beyond the summary, it is worth asking for suggestions about what comes next: angles not explored, inconsistencies in the documentation, related subjects, improvements to the overall structure.

```
Are there aspects we haven't covered that would be worth
including? Anything to watch out for in the next session?
```

These suggestions are not obligations, but they are often useful: the view of the session as a whole is still available, whereas yours may have dulled after an hour of intensive work.

8.5 Closing template

```
Before we close this session:

1. Summarise the files produced, with their exact names.
2. List the important decisions taken.
3. Set out the open points and what is left to do.
4. Offer two or three suggestions for the next session that you
   think would be useful but that we did not discuss.
```

9 Good practice and mistakes to avoid

9.1 The ten habits of an effective user

- **Frame it from the start:** give the *context*, the goal and the constraints in the first message.
- **Be specific:** "a five-line paragraph in a technical style" beats "write something".
- **Break it down:** for any complex task, start with the outline, not the content.
- **Iterate:** do not chase perfection on the first attempt. Refine over several turns.
- **Correct precisely:** say why an answer misses the mark and in which direction to go.
- **Use projects:** for any recurring work, work inside a project with reference files.

- **Check critical facts:** take nothing at face value where precise data is concerned.
- **Close your sessions:** summary, open points and suggestions — five minutes that save hours.
- **Keep the project files current:** after every major change, upload the new files and delete the obsolete ones.
- **Learn by documenting:** document what you learn in the same movement — structure and write at once.

9.2 The most frequent mistakes

- **A prompt that is too vague:** "tell me everything about home automation" yields nothing but a generic, unusable answer.
- **One session running too long:** beyond a certain point the *context window* fills up and the first instructions survive only in summary form. Several short, well-structured sessions are better.
- **No close:** failing to capture the decisions and the open points forces you to explain everything again in the next session.
- **Confusing regenerating with correcting:** regenerating without correcting your message produces a result of the same kind, and therefore just as unsatisfying.
- **Relying on the project files to modify a document:** the platform keeps only the text. To modify an existing document, drop its binary into the *conversation*.
- **Expecting continuity between projects:** *memory* is kept separate. What was established in one project has to be reintroduced in another.

10 Outlook and change

10.1 Generative AI moves fast

Models are updated regularly. New capabilities appear: a larger *context window*, better reasoning, new file types supported, built-in tools. This manual is updated alongside them.

The best way to stay current is to experiment. Try new formulations, new *prompt* formats, new kinds of attached files. You can also simply ask which capabilities are available.

It is also why this manual avoids naming models and quoting version numbers: documentation that describes principles stays correct longer than documentation that describes a state.

10.2 Going further

A few directions for users who have the basics in hand.

- **Advanced prompting techniques:** worked examples, explicit reasoning, role calibration, negative constraints. Covered in detail in the Guide Pratique.
- **Automation:** bring the *model* into your existing workflows through its programming interface.
- **A searchable knowledge base:** build a local corpus to query, where the answer rests on your own documents.
- **Agents:** hand over autonomous multi-step tasks, with access to tools.
- **Multimodal:** make use of image analysis and complex document analysis.

Note: *This manual is a living document. To propose a correction or an addition, open a conversation in the Kit project and flag the point to improve: the change will be integrated and the document regenerated with an up-to-date timestamp.*