

Moving Claude to Another PC

How to take the memory, the rules and the skills with you — for any project

Version 2.4 · 27 October 2026

Hans Tandt

Contents

1. What this is for	3
2. Where Claude keeps its brain	3
2.1 The five things that matter most	4
2.2 The parts people skip, and what each one is for	5
2.3 Three different things, easy to confuse	5
2.4 Keep a session note, and update it every session	6
3. Before you copy anything	6
4. Harvest it with two prompts	7
4.1 The seven questions the export asks	8
4.2 What the export leaves on the drive	8
4.3 Check before you unplug	9
4.4 It writes what is missing	9
5. What to leave behind	10
6. Putting it back on the new PC	10
6.1 The four questions the import asks	11
6.2 It merges, it does not overwrite	11
6.3 The memory folder name will be different	12
6.4 Programs and environment variables	12
7. Check that it worked	13
7.1 If something is still missing	13
8. A short checklist	13
9. Glossary	14

1. What this is for

You have worked with Claude on one PC for months. Claude has learned how you work, what this project is, and which mistakes to avoid. Now you need that same Claude on a different PC.

Copying the project folder is not enough. Most of what Claude has learned is stored somewhere else, and a folder copy leaves it behind. This guide shows what to take and how to put it back.

NOTE — Works for any project

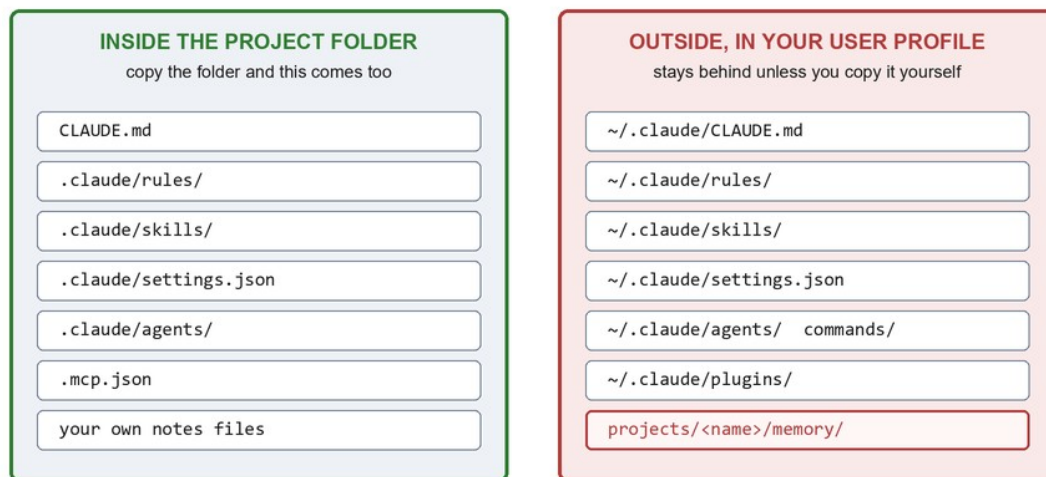
Nothing here is specific to one project or one company. Follow the same steps for a manual, a program, or a folder of documents.

NOTE — Who made this

These prompts and this manual were created by Hans Tandt (hanstandt.be). They are free under the MIT licence: you may use, copy and adapt them, as long as you keep this line.

2. Where Claude keeps its brain

Claude stores what it knows in two places. Only one of them travels when you copy the project folder.



Most of the brain is on the right.

Copying the project folder takes the left side only. The right side is the part people lose.

The right-hand side lives in your user profile, in a hidden folder called `.claude`. On Windows that is `C:\Users\<your name>\.claude`. It is shared by every project on the PC, which is exactly why a project copy does not include it.

2.1 The five things that matter most

What	What it holds	Where it lives
Auto memory	What Claude worked out about you and the project, written by Claude itself	<code>~/.claude/projects/<project>/memory/</code>
Skills	Your methods: how to build a manual, how to check work before saying it is done	<code>~/.claude/skills/</code>
Personal CLAUDE.md	Your rules for every project on that PC	<code>~/.claude/CLAUDE.md</code>
Project CLAUDE.md	The rules for this one project	In the project folder
Your notes file	Decisions and the reasons behind them, written by you	In the project folder

TRAP — Skills are the easiest thing to lose

Skills live outside the project. If you copy only the project folder, every skill you ever wrote stays on the old PC. They are small files and they carry the most work, so take them first.

2.2 The parts people skip, and what each one is for

Those five matter most. The profile usually holds more than that, and what follows is the part people skip, because the name means nothing to them. Each is a small folder. Each is work you already did.

What	What it holds	What you lose without it
<code>rules/</code>	Personal rules split by topic, taken out of your personal <code>CLAUDE.md</code> to keep it short	They carry the same force as <code>CLAUDE.md</code> itself. Nothing announces that they are gone.
<code>agents/</code>	Subagents you wrote — a helper with its own instructions for one kind of job	They stop existing. Nothing errors; the work just gets done the ordinary, slower way.
<code>commands/</code>	Your own commands, each a saved instruction you use often	A habit stops working. You type it and nothing happens.
<code>output-styles/</code>	How answers are shaped — length, tone, how much explaining	Claude sounds different on the new PC and you cannot say why.
<code>workflows/</code>	Jobs of several steps, saved and run as one	Usually the longest thing in this table to rebuild from memory.
<code>plugins/</code>	Packages of skills and commands that came from a marketplace	These are installed, not copied. Take the list, not the folder.
<code>.mcp.json</code>	Which MCP servers a project may reach — the outside tools it can use. This one sits in the project, not the profile	The project loses its tools and looks broken for no visible reason.

NOTE — The same names exist on both sides

A project can have its own `.claude\` folder holding `rules\`, `skills\`, `agents\`, `commands\` and `workflows\`, alongside `.mcp.json` and `CLAUDE.local.md`. Those travel with a copy of the folder. The ones in your user profile do not. When the export lists both, check which side of the line each one is on.

2.3 Three different things, easy to confuse

	Who writes it	How long it lasts
Transcript	Recorded automatically	Deleted after a while. Do not rely on it.
Auto memory	Claude, for itself	Stays, but lives outside the project
Session note	You, with Claude's help	Stays, and travels with the project

The session note is the only one of the three that you control, that survives, and that a new Claude can read on the first day. It is the most valuable file in the handover, and it is the one nobody remembers to write.

2.4 Keep a session note, and update it every session

A session note is one file in the project. It records what was done, what was decided and why, and what is still open. Write it in plain sentences, for a reader who was not there.

Make it a habit: at the end of every working session, ask Claude to update the session note. It has the whole session in front of it and will draft in a minute what would take you twenty. Then read what it wrote — you are the one who knows whether it is true.

Put in it	Leave out
What changed, and on what date	Anything Claude can read from the files themselves
What was decided, and the reason behind it	A blow-by-blow account of the conversation
What was tried and rejected, and why	Praise, filler, or a summary of what you already know
What is still open, and the next step	Anything you are not sure is true

TIP — The reason matters more than the decision

"We use LibreOffice for the PDF" is worth little on its own. "We use LibreOffice because Word hangs when it runs unattended" stops the next person undoing it.

TRAP — Do not let the note grow into a diary

A note nobody reads is the same as no note. Keep it to decisions, reasons and open points. Move finished detail into the project's own documents.

3. Before you copy anything

Do these three things on the source PC first. They take ten minutes and they decide how much you actually get.

1. Make sure the project has a `CLAUDE.md`. If it has none, ask Claude to run `/init`, then read what it wrote and correct it. This is the file that explains the project to a fresh Claude.
2. Update the session note. Ask Claude to bring it up to date, then read it. On the new PC this is the first thing read, so it must be right.
3. Ask Claude to tidy its memory. Open `/memory`, remove what is out of date, and merge anything written twice. Do not carry stale facts to the new PC.

TIP — Why tidy first

A wrong fact that travels to the new PC is worse than a missing one. You will trust it, and it will be wrong.

4. Harvest it with two prompts

Two prompts do the work. They are plain text files, made to be pasted into Claude Code — not documents to read on screen. This guide describes version 3.2 of them.

File	Run it on	What it does
1_EXPORT_PROMPT.txt	The source PC	Asks seven questions, then works through five tasks: it lists the project, collects everything that lives outside it, writes whatever is missing, copies it all, and checks the result
2_IMPORT_PROMPT.txt	The new PC	Asks four questions, then works through seven tasks: it reads what the export left on the drive, puts each part back in the right place, proves it arrived, and goes back to the old PC for whatever is still missing

Open Claude Code with the project folder as the working directory, and paste the prompt as one message. Each file has a row of `v` characters and a row of `^` characters inside it. Paste only what lies between them — the marker rows are not part of the instruction.

Both ask their questions before touching anything, so nothing is discovered halfway through — such as a drive with no room left. After that they work one task at a time and show you the result before starting the next.

NOTE — They discover, they do not assume

Neither prompt has a fixed list of your files. Each looks at what is really there and reports it, so you do not need to know in advance which skills or settings exist.

NOTE — The export changes nothing

The export prompt treats the project as read-only. The only new things it makes are the notes files it is asked for and the copies on your drive. It does not build, install or set anything, and it never copies your login.

NOTE — They read your files, they do not obey them

A file in the project, or on the drive, could contain text that reads like an instruction. Both prompts are told to show you that text instead of acting on it. Only you give instructions.

4.1 The seven questions the export asks

Have these answers ready. Nothing starts until you have given them, and once you have, it checks there is enough free space at the destination and stops if there is not.

1. Is the folder I am in the project you want exported?
2. Where should the export go? A drive letter or a full path. It creates a folder called `Handover_Export` there.
3. Is this a move, a backup, or a second machine that stays in use?
4. Do you want the big folders included? It tells you the total size first, and names any single folder over 100 MB, so you can decide.
5. Which file is the main entry point — the thing that gets run or built? It offers its own guess for you to confirm.
6. Is there a session note or a decisions file it must be sure to include?
7. Is there anything outside the project folder and outside `.claude` that the work depends on — a prompt library, a folder of templates, a shared drive? It cannot find these on its own.

4.2 What the export leaves on the drive

Everything lands in one folder. Carry the whole folder to the new PC, with its contents and its names as they are: the second prompt asks you where it is and then reads inside it.

Handover_Export\	
project\	the project, minus what must not travel
brain\	
skills\	every skill from the old PC
memory\	the memory .md files
rules\ agents\	commands\ output-styles\ workflows\
CLAUDE.md	your personal rules, for every project
settings.json	settings and hooks
HANDOVER_MANIFEST.md	every file, with a checksum
HANDOVER_BRAIN.md	what was never written down anywhere
SOURCE_FACTS.md	the old paths, programs and variables
robocopy_log.txt	proof of what was copied

What the export produces. This folder is the handover.

The three notes files	What it is for
HANDOVER_MANIFEST.md	Every file in the project with its size and a checksum, so you can prove on the new PC that you have the right version and that nothing was lost on the way
HANDOVER_BRAIN.md	Decisions and their reasons, things agreed out loud and never written down, questions never answered. Read this one yourself — it is the part no folder copy contains
SOURCE_FACTS.md	The old project path, the old memory folder name, the programs, the environment variables, and what must be merged rather than replaced. The import prompt reads this file

NOTE — It is a folder, not a zip file

Nothing here is compressed. The import prompt reads `SOURCE_FACTS.md` straight off the drive, and the check before you unplug opens copied files to prove they arrived whole — both are easier on a plain folder, and a zip that is subtly damaged gives you nothing at all. If you are sending the export over a network instead of carrying it, zip the whole `Handover_Export` folder for the journey and unzip it before you run the import.

4.3 Check before you unplug

The export ends with a check. Read it rather than skim it. While the drive is still in the source PC you can fix whatever it finds; once you have walked away, that drive may be the only copy you have.

TRAP — A copy that reports success is not proof

A copy tool can report success and still have written a file that is truncated. The export opens two or three of the copied files — a `SKILL.md` and a memory `.md` — and confirms they are complete, not empty and not cut off.

It also confirms the following before you go — and if something is missing, this is the moment to find out:

- Each of the three notes files is there, with its size
- `project\`, `brain\skills\` and `brain\memory\` all arrived
- Nothing that must not travel reached the drive — `.credentials.json` above all
- The file count and total size match what the copy itself reported
- A plain listing of everything on the drive, so you see the whole harvest at once

4.4 It writes what is missing

Chapter 3 asks you to put `CLAUDE.md` and the session note in order before you start. People do not always do it, and a thin session note travels just as badly as none at all.

So the export no longer only complains. Its third task offers to write what is missing, from the session you are already in: a `CLAUDE.md` if there is none, and one new dated entry for the session note. It shows you every line first and writes nothing until you say go.

NOTE — It only ever adds

It may create a file that does not exist, or append to one that does. It will never rewrite or remove a line you wrote. If you say no, it leaves the file alone and carries on — a gap you know about is fine.

TIP — This is the one thing worth doing even if you are not moving

Run the export prompt's third task on a project that is staying exactly where it is, and you end up with a `CLAUDE.md` and a session note that were written while the work was still fresh. That is worth having whether you move or not.

5. What to leave behind

Some things must never be copied. They are either private, or they are rebuilt automatically on the new PC and copying them causes faults.

Do not copy	Why
<code>.credentials.json</code>	Your login. Sign in on the new PC instead. Never put it on a stick.
API keys, tokens, passwords	Never copied as a value. The export names them and writes SECRET - set by hand on the new PC. You set them yourself there.
<code>.venv</code> , <code>node_modules</code>	Built for one PC. Recreate them with <code>py -m venv</code> and <code>npm install</code> .
<code>__pycache__</code> , <code>build</code> , <code>dist</code> , <code>_internal</code>	Generated files. They come back on their own.
<code>cache</code> , <code>shell-snapshots</code> , <code>telemetry</code> , <code>debug</code> , <code>sessions</code> , <code>file-history</code> , <code>history.jsonl</code>	Working files of the tool itself. Of no use to you.
<code>_archive</code> , very large <code>.zip</code> files	Usually old snapshots of what you are already copying.
<code>.git</code>	Large, and better restored by cloning from your remote. Read the warning below first.

TRAP — If the project is a git repository

The export leaves `.git` out, so the history does not travel with it. Clone the project from its remote on the new PC instead.

If the project has no remote, its history exists only on the old PC. Copy `.git` across by hand before you wipe anything.

6. Putting it back on the new PC

The second prompt does this work. You prepare the PC, paste the prompt, and answer what it asks. It then works through seven tasks: what is missing on this PC, the project files (checking the main entry point against the checksum in the manifest, so you know you have the right version), the skills and settings, the memory, the environment variables, a checklist, and finally whatever is still missing — showing you each result before it moves on.

1. Plug in the drive that holds `Handover_Export`.
2. Create the folder where the project will live. Empty is fine.
3. Open Claude Code with that folder as the working directory. This matters: it makes Claude create the memory folder under the correct new name.
4. Paste `2_IMPORT_PROMPT.txt` as one message, without the marker rows.

TIP — Let the prompt do the fiddly steps

Putting the memory back means finding a folder whose name changed with the path. Restoring the settings means merging two files rather than replacing one. Both are exactly the kind of step to hand to a machine rather than do by hand at the end of a long day.

6.1 The four questions the import asks

It asks these four, then reads `SOURCE_FACTS.md` and shows you the old project path, the old memory folder name, the programs and variables you will need, and anything that must be merged rather than replaced. If the destination folder turns out not to be empty, it lists what is in it and stops to ask you.

1. Where is `Handover_Export`? A drive letter or a full path.
2. Is the folder I am in where the project should live?
3. Is this PC fresh, or does it already have Claude skills and settings worth keeping?
4. Should I install the missing programs, or only list them?

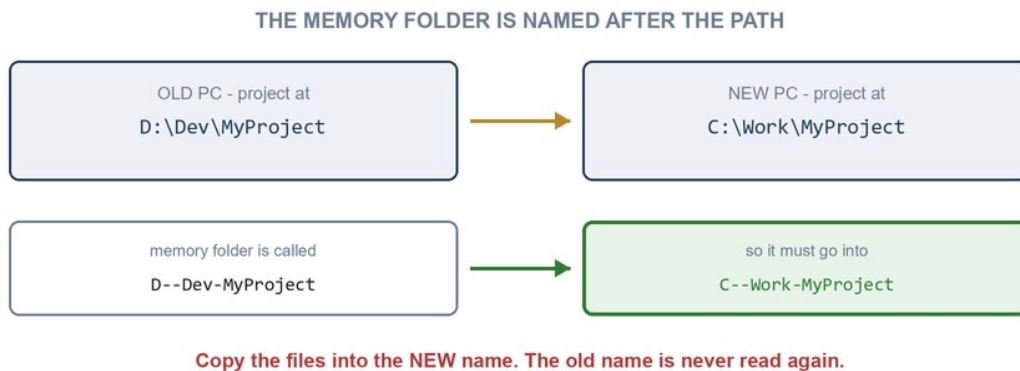
6.2 It merges, it does not overwrite

The new PC may already have a Claude set-up of its own. Replacing files wholesale would throw away permissions and hooks built up there. The prompt is written not to.

What	How it is handled
Skills	Folders, so they sit side by side. A skill of the same name is never overwritten: you are shown both — dates, sizes and any changelog — and you choose.
Personal <code>CLAUDE.md</code>	A single file. If this PC has none, it is copied in. If it has one, both are shown in full and a merged version is proposed. Nothing is written until you approve it.
<code>settings.json</code>	A single file, treated the same way: both versions, the keys that differ, and a proposed merge for you to approve.
<code>rules\</code> , <code>agents\</code> , <code>commands\</code> , <code>output-styles\</code> , <code>workflows\</code>	Copied file by file. Anything that already exists here is skipped, and you are told which.
Machine-specific files inside a skill	<code>node_modules</code> , <code>__pycache__</code> and <code>profile.json</code> are left behind on purpose. They belong to the old PC — <code>profile.json</code> is the file that names who signs documents there.
Plugins	Not copied. It lists which plugins and which marketplaces the old PC had, and gives you the command to install each one here. Plugins belong installed, not copied.

6.3 The memory folder name will be different

This is the step that goes wrong most often. The memory folder is named after the full path of the project. Put the project somewhere else and the name changes.



Copy the memory files into the folder named after the NEW path, not the old one.

The prompt handles this in its fourth task. It finds the folder for this PC's path, confirms it is not the old name recorded in `SOURCE_FACTS.md`, copies the `.md` files into it, then reads `MEMORY.md` back and tells you how many entries arrived. It also lists any memory that is now wrong here — an old drive letter, an old path, a note about the source PC — and leaves the decision to you.

To do it by hand, or to check the prompt's work:

1. Start Claude Code once in the project folder. This creates the folder with the correct name.
2. Look in `C:\Users\<your name>\.claude\projects\` and find the new name.
3. Copy the `.md` files from the old PC into the `memory\` folder inside it.
4. Restart Claude Code and ask: "what do you remember about this project?"

TRAP — Copying into the old name does nothing

A folder named after the old path is never read again. The files will sit there, look correct, and be invisible to Claude.

6.4 Programs and environment variables

Files arriving is not the same as a project that runs. Two things never travel at all: installed programs and environment variables. Both have to be put in place by hand.

The first task of the import prompt reports which programs and packages are missing and prints the exact command for each. Its fifth task lists every environment variable from the old PC, tells you whether it is already set here and to what, and gives you the `setx` command with the path corrected for this PC.

TRAP — A new variable needs a new terminal

`setx` writes the variable, but a window that is already open will never see it. Close Claude Code and open it again afterwards.

7. Check that it worked

Do not assume. Ask, and compare against what you brought.

TRAP — Restart before you check anything

Memory is only read when a session starts. Close Claude Code and open it again first, or a set-up that is perfectly correct will answer as though it remembers nothing.

Ask Claude	You should get
"What do you remember about this project?"	The facts from the old PC, not a blank answer
"Which skills do you have?"	The same list as on the old PC
"Which instruction files did you load?" (/context)	Your project and personal <code>CLAUDE.md</code>
Run the project's own build or test	The same result as on the old PC

The import prompt finishes by printing four lists: what was copied, what was skipped and why, what still needs your decision, and what could not be done at all. Read that before you trust the new PC.

TIP — Keep the stick until this passes

Do not wipe the source PC or the stick until every row above is answered correctly. Until then it is the only copy you have.

7.1 If something is still missing

Do not treat the checklist as the end. If an answer came back wrong, or the project will not run, you are not finished — and the old PC is still sitting there, able to answer. The import prompt's seventh task uses it.

1. It tells you exactly what it needs and does not have — a named file, a skill, a setting, a fact.
2. It splits that in two: what the old PC can still give you, and what only you can do here (install a program, add a plugin, set a variable).
3. For the first group it writes you a prompt to paste into Claude Code on the OLD PC, naming the project and exactly what it needs.
4. You bring the answer back, and the checklist runs again. Round that loop as many times as it takes.

TRAP — This is why nothing gets deleted yet

The whole loop depends on the old PC still being there to ask. Keep it, and keep the drive, until the checklist comes back clean. Both prompts now say so in their last line.

8. A short checklist

- Project folder, without `.venv`, `node_modules`, `build`, `dist`, `__pycache__`
- The session note, brought up to date on the day you copy

- Project `CLAUDE.md` and your notes files (inside the folder already)
- The whole `~/.claude/skills/` folder
- `~/.claude/CLAUDE.md` and `~/.claude/rules/`, if they exist
- `~/.claude/settings.json`
- `~/.claude/agents/`, `commands/`, `output-styles/`, `workflows/`, if they exist
- The `.md` files from `~/.claude/projects/<project>/memory/`
- A written list of the plugins installed — the list, not the folder
- A written list of environment variables and installed programs
- `HANDOVER_MANIFEST.md`, `HANDOVER_BRAIN.md` and `SOURCE_FACTS.md`
- `.git`, only if the project has no remote to clone from
- Never: `.credentials.json`

9. Glossary

Term	Meaning
Auto memory	Notes Claude writes for itself about you and the project.
Session note	A file in the project where you record what was decided and why. You write it, and it travels with the folder.
CLAUDE.md	A file you write telling Claude how to work. One per project, plus a personal one.
Skill	A folder holding a written method, with a <code>SKILL.md</code> inside.
Plugin	A set of skills and commands from a marketplace. It is installed, not copied.
Handover_Export	The folder the export prompt creates on your drive. It holds the project, the brain and three notes files.
SOURCE_FACTS.md	The file the export writes and the import reads: old paths, programs and variables.
User profile	Your own folder on the PC: <code>C:\Users\<your name></code> . Contains the hidden <code>.claude</code> folder.
Transcript	The recording of one session. Deleted automatically after a while.
Environment variable	A setting stored in Windows, not in a file. It does not travel with a copy.
Hook	A command Claude runs by itself at a set moment, set up in <code>settings.json</code> .
MCP server	An outside tool a project is allowed to reach, named in <code>.mcp.json</code> .

Revision history: Version 1.0 — 10 September 2026 — First version, replacing a handover prompt written for one project with a general method. Version 2.0 — 11 September 2026 — Brought in line with version 3.0 of the two prompts: the questions each asks, what the export leaves on the drive, merging instead of overwriting, git history, programs and variables. Version 2.1 — 11 September 2026 — Everything the two prompts handle is now explained, with the reason for taking it: the profile folders that had only ever been listed (2.2), the check before you unplug (4.3), and the rule that the prompts read your files without obeying them.

Version 2.2 — 26 October 2026 — Brought in line with prompts version 3.1: the export now writes a missing `CLAUDE.md` or session note instead of only complaining about it (4.4), it asks about folders that live outside the project (4.1), the import can go back to the old PC for what is missing (7.1), and nothing may be deleted until that passes. Version 2.3 — 26 October 2026 — Says plainly in 4.2 that the export is a folder and not a zip file, and when zipping it is still the right thing to do. Version 2.4 — 27 October 2026 — Brought in line with prompts version 3.2: passwords, keys and tokens are named but never written to the drive (chapter 5). Who made it, and the MIT licence (chapter 1). Dutch and German versions.