THE PRODUCTION GAP · EPISODE 02 COMPANION

The handover guide.

How I clear long AI coding sessions without losing the thread, and the exact prompts to set it up in yours.

Manish Kumar
@manishkumarglobal

Version 1.1 · November 2026

BEFORE YOU START

What this is

This is the setup behind my reel about my AI agent picking up the wrong notes.

Long Claude Code sessions get expensive and fuzzy, so I clear them on purpose. Before I do, the agent writes a handover note, and the fresh session loads it. When that goes wrong, the agent works confidently from an old picture of your project, and nothing warns you.

The setup I run day to day is a bigger system tied to my machine. What you get here is a separate, simpler version of the same idea: three skills, one helper script that does the checking, one optional hook, and the prompts that install them. The files come in the companion folder that came with this guide. You paste a prompt into Claude Code, and it checks the files, shows you what it will change, and installs them once you say yes.

What's inside

- Why I clear sessions on purpose, and what that does and doesn't save
- The mistake I made, and the three rules it taught me
- What a good handover note contains, and how the check works
- Five copy-paste prompts and an uninstall prompt: install the kit and `/handoff`, install `/resume-handoff`, install `/compact-note` with its hook, check your setup, and write a handover for a teammate
- How to test it, move work between machines, undo it, and a one-page cheat sheet

How it was tested, and what it can't do, are on the last pages. Copy the prompts from `PROMPTS.md` in the companion folder, not from this PDF. Copying out of a PDF can mangle line breaks. Prompt 1 checks the kit's files against their checksums before and after copying them, so a damaged download stops the install instead of installing something broken.

01 · THE COST

Why I clear sessions on purpose

A long session costs more and remembers worse.

Every message you send goes to the model along with everything said before it. Caching makes the repeated part cheaper, but a long conversation still uses more of your limits with every turn.

It also gets worse at remembering. When the conversation fills up, Claude Code compacts it: it replaces the history with a summary and carries on. A summary keeps the storyline and drops detail. Let that happen a few times and you're working from a summary of a summary.

The details that fade first are the ones that hurt most:

- small decisions and the reason behind them, so it tries the approach you already rejected
- rules you gave once in the chat, like "don't touch that file"
- exact details, such as file names, error messages and what was already tested

And it sounds just as confident after the fifth compaction as it did before the first.

When I changed my compaction settings, the context carried into each turn of my own sessions went from about 346K tokens to about 219K on average, roughly 37% less. That's context carried, not my bill, and it isn't a promise for yours. It depends on how long your sessions run, the model, caching, how compaction behaves, the work and how often you resume. Sometimes letting Claude Code compact is the cheaper option.

So when a big piece of work is done, I write a handover note and clear the session. The new session starts small, with a note I can read and check, instead of a summary nobody wrote on purpose. A note is still a summary: it keeps the task state and decisions that matter, not every detail of the conversation.

02 · THE MISTAKE

My agent picked up the wrong notes

My setup wrote a good note every time. Reading it back was the problem.

My resume step looked for the latest note in the folder I started the new session in. When the work was in that same folder, it picked the right note in all eight cycles I could trace.

But sometimes I did the work in a separate clone of the same repository. The note was written there, but the next session started in my main folder and loaded the latest note it could see. In the three cycles like that I could trace, it never resumed the note that was meant. It even noticed the code didn't match, and blamed the code for changing.

That's too small a sample for a failure rate. What matters is how it failed: it looked like it worked. My handover check passed, because it checked the note from the folder where it was written. The next session reads from somewhere else.

> **“Check the latest doc” works, until there are two latest docs.**

Three rules came out of it, and the kit in this guide follows all three.

1 **Hand over by name.**
Every note gets an ID, and you resume with that ID. The next session never goes looking for whatever note happens to be newest.

2 **Check from the receiving side.**
A handover is only done once the reader can load the note by its ID and a script, not the model's judgement, confirms the code still matches it.

3 **When unsure, ask.**
If you resume without an ID, the skill lists the recent notes and asks you which one. If the code has moved on, it tells you. It never guesses.

03 · THE NOTE

What a good handover note contains

Write it for a smart colleague who has never seen this conversation, because that's exactly who reads it.

The header	When it was written, the folder the session started in, the folder where the work actually is, the branch and the commit. The helper writes this from the repository itself, so it's measured, not remembered. Keeping the two folders separate fixes my mistake.
Goal	One or two sentences.
Done means	What has to be true to call it finished. Without it, the next session decides for itself when it's done.
Where things stand	What's finished and what isn't, every claim tagged.
Decisions and why	The why, so the next session doesn't undo them.
Tried and rejected	What didn't work and why, so nobody pays for it twice.
Open problems	Known failures, risks and questions nobody has answered yet.
Next step	One concrete action, small enough to do in one go. "Continue" is not a next step.
Read these first	The few files that matter, in order.

Tag every claim

A fresh session can't tell a fact from a hunch unless you mark it:

`[checked]`	confirmed just now, while writing the note
`[from session]`	seen earlier in the session, not rechecked
`[guess]`	an inference that still needs checking
`[unknown]`	nobody knows yet

Never put passwords, tokens, keys or customer data in a note; the helper refuses one that looks like it does. Treat your notes folder like source code.

04 · THE CHECK

How the check works

A status is only worth what was actually compared, so the helper tells you exactly which one you got.

When you save a note, the helper records the branch, the commit, what's staged, and a SHA-256 fingerprint of every changed and untracked file. When you resume, it measures them again and compares. It stores fingerprints, never the files.

`VERIFIED_MATCH`	Everything recorded still matches. The code is where the note left it.
`DRIFT`	Something changed after the note, and it lists what. Treat the note as history.
`UNVERIFIABLE`	Part of the state couldn't be compared, like a changed `.env` file or a folder outside git. Never a match.
`REPO_MISSING`	The checkout isn't at the recorded path any more.
`REPO_MISMATCH`	The folder at that path is a different repository, or no longer a git repository.
`INVALID_HANDOFF`	The note or its record is incomplete or damaged.
`NOT_FOUND`	No note has that ID. Nothing else is loaded instead.

What it doesn't check

Ignored files aren't compared. Secret-looking files, like `.env` files and keys, are never fingerprinted, because even a hash can give a short secret away. If one has changed, you get `UNVERIFIABLE`, not a match.

A match means the code is where the note left it, not that the note is right.

Compact notes

`/compact-note` saves a note for one session, and the hook puts it back once after that session's next compaction. If it's over two hours old, or the code has changed, it comes back labelled as history, not current state. It never comes back twice, or into another session.

05 · SET IT UP

Five prompts, in order

Open Claude Code in any folder, paste a whole prompt, and let it do the work.

1. **Install the kit and /handoff.** Required. Copies the kit from the companion folder, checks it, and installs the skill that writes notes.
2. **Install /resume-handoff.** Required. Loads a note by its ID and checks it against the code.
3. **Install /compact-note and its hook.** Optional. Saves what must survive before you compact, and puts it back once afterwards.
4. **Check your setup.** Run it after installing. It changes nothing and tells you what's missing.
5. **A handover note for a teammate.** For people, not agents. Use it any time.

A few things to know

- You need Claude Code on macOS or Linux, with `python3` 3.9 or newer and `git`. On a Mac without the developer tools, the first use asks to install them.
- Download and unzip the companion folder first. Prompt 1 asks for its full path: in Finder, hold Option, right-click the folder and choose Copy as Pathname.
- Every install shows you a plan first and waits for your OK. It never overwrites a skill it didn't install: if you already have your own `/handoff`, it stops and tells you. It won't write through a symlinked skills folder or settings file either.
- Prompt 3 edits `~/.claude/settings.json`. It backs it up first into a new timestamped folder in `~/.claude/handover-kit-backups`, keeps every other setting and hook (its formatting may change), and won't add the hook twice.
- Changes to skills and hooks normally apply without a restart. If a command doesn't show up, run `/reload-skills` or restart Claude Code. Each skill runs the helper script, so Claude asks permission to run it. That's expected.
- On a work machine or a company repository, check your organisation's rules before you install hooks or keep project notes on your machine.
- Notes stay in `~/.claude/handoffs`, readable only by you. The kit uploads and syncs nothing, but a note's text is part of your conversation with Claude.

PROMPT 1 REQUIRED

Install the kit and /handoff

```
Set up the handover kit v1.1 and the /handoff skill for Claude Code.
Work carefully and stop at the first problem.
HELPER below means: python3 ~/.claude/handover-kit/handoff_kit.py

1. Ask me for the full path of the unzipped companion folder, the one
   with README.md and a kit folder in it. Do not search for it or guess
   it. Below, COMPANION means that folder.
2. Check that python3 (3.9 or newer) and git are installed. If either is
   missing, stop and tell me.
3. In COMPANION/kit run: shasum -a 256 -c SHA256SUMS
   (on Linux: sha256sum -c SHA256SUMS). Every line must end in OK. Show
   me the output and the contents of SHA256SUMS. If any line fails, stop.
   Never edit a file or SHA256SUMS to make the check pass.
4. If ~/.claude/handover-kit already exists, list what is in it and ask
   me before changing anything there.
5. Create ~/.claude/handover-kit and copy exactly these five files from
   COMPANION/kit into it, keeping their folders. Copy nothing else, and
   do not change them:
   handoff_kit.py
   skills/handoff/SKILL.md
   skills/resume-handoff/SKILL.md
   skills/compact-note/SKILL.md
   SHA256SUMS
6. In ~/.claude/handover-kit run the same checksum check again. Every
   line must end in OK.
7. Run: HELPER install handoff --dry-run
   Show me the plan exactly as printed and ask me to confirm.
8. Only after I confirm, run the same command without --dry-run and
   show me its output. If it refuses, show me why and stop.
```

Checksums of the kit files

Prompt 1 shows you `SHA256SUMS` from your download. These are the values for version 1.1. If any of them differ, don't install.

handoff_kit.py

`09ffa896f5ce5fa0015fc53ebb73f6f4d1f1ced7632ab42bfbe951f104d4f7e0`

skills/handoff/SKILL.md

`b6e24492028ac1cfbdb2cf015866fb40aeed8dcc7c3b038e48032e794714043a`

skills/resume-handoff/SKILL.md

`c2105eb0b2f532ff79bd8f73367bdb22a5f1a0e6cd57d83f1da0da724c9bceed`

skills/compact-note/SKILL.md

`d4b74aaca759323cf6e63dae428087bf277920c6ea39b8ea9e25abba3e9869b3`

PROMPT 2 REQUIRED

Install /resume-handoff

```
Install the /resume-handoff skill from my handover kit v1.1.
HELPER below means: python3 ~/.claude/handover-kit/handoff_kit.py

1. Check that ~/.claude/handover-kit/handoff_kit.py exists. If it does
   not, stop and tell me to run prompt 1 of the guide first.
2. Run: HELPER install resume-handoff --dry-run
   Show me the plan exactly as printed and ask me to confirm.
3. Only after I confirm, run the same command without --dry-run and
   show me its output. If it refuses, show me why and stop.
```

PROMPT 3 OPTIONAL

Install /compact-note and its hook

```
Install the optional /compact-note skill and its compaction hook from my
handover kit v1.1.
HELPER below means: python3 ~/.claude/handover-kit/handoff_kit.py

1. Check that ~/.claude/handover-kit/handoff_kit.py exists. If it does
   not, stop and tell me to run prompt 1 of the guide first.
2. Run: HELPER install compact-note hook --dry-run
   This adds one SessionStart hook with the matcher "compact" to
   ~/.claude/settings.json. Show me the plan exactly as printed, including
   any notes about existing or duplicate hooks, and ask me to confirm.
3. Only after I confirm, run the same command without --dry-run. It
   backs up settings.json first, keeps every other setting and hook as
   it was, and checks the result. Show me its output.
4. Show me the SessionStart part of ~/.claude/settings.json.
5. Tell me that hook changes normally apply to running sessions. If the
   hook does not seem to run, I should restart Claude Code.
```

PROMPT 4 READ-ONLY

Check your setup

```
Check my handover kit setup. This is read-only: do not change, install,
fix or delete anything.
HELPER below means: python3 ~/.claude/handover-kit/handoff_kit.py

1. Check that ~/.claude/handover-kit/handoff_kit.py exists. If it does
   not, stop and tell me to run prompt 1 of the guide first.
2. Run: HELPER check
3. Show me its table exactly as printed.
4. For every row that is not OK, explain in one line what it means and
   which prompt of the guide fixes it. Do not run that prompt.
```

06 · TEST IT

Test it from the receiving side

That's the whole lesson, so don't skip it. It takes five minutes.

1. In a git repository, do a small piece of work and run `/handoff`. Copy the ID it prints. The receiving-side check in its report should say `STATUS: VERIFIED_MATCH`.
2. Run `/clear`, then `/resume-handoff` followed by that ID. You should see `VERIFIED_MATCH` and a summary, and Claude should wait for you.
3. Now change a file you had already changed, without committing, and resume with the same ID again. You should see `DRIFT`, naming that file. That's the case my first version missed.
4. Run `/resume-handoff` with no ID. It should list your recent notes and ask which one to load. If it loads one by itself, something is wrong.
5. If you use worktrees or a second clone of a repository, try the case that broke mine. Start Claude Code in your main folder, do the work in the other copy, and run `/handoff`. The report should name the other copy as the task checkout. Then resume by ID in a fresh session.

If you installed the hook, run `/compact-note`, then the `/compact` line it gives you, and ask Claude what rules it is following. Compact again and ask again: the note should not come back as current.

The companion also has the kit's automated tests. Run `python3 -m unittest discover -s tests` inside the companion folder. They run in throwaway folders and never touch your own setup.

07 · MOVING WORK

Where a note can take you

A note is a file on one machine. Where you can resume depends on who can read that file.

Same machine, same checkout

Resume by ID. This is the normal case.

Same machine, another clone or worktree

Resume by ID from any folder. The report tells you which checkout the work is in, and the work belongs there.

Another machine

Notes stay on the machine that wrote them, and nothing syncs them. Uncommitted changes don't travel, so commit and push before you run `/handoff`. Copy the note's folder from `~/.claude/handoffs` yourself, in a way your company allows, and check out the same commit there. The recorded paths will be wrong there, so expect `REPO_MISSING`, then give Claude the new path and it checks the code there.

Another person

They don't have your notes or your session. Use prompt 5 to write a handover for a person, with the exact branch, commit and links, and share it the way your team shares documents.

08 · CHEAT SHEET

Every day

`/handoff`	Write the note and check it from the receiving side. Add a focus if you like: `/handoff the login bug`.
`/clear`	Start fresh. You can name the old conversation, like `/clear login-bug`, which makes it easy to find in `/resume` later.
`/resume-handoff <ID>`	Load that exact note, check it against the code, and wait for you.
`/resume-handoff`	No ID: list recent notes and ask which one.
`/compact-note`	Before compacting, save what must survive, then run the `/compact` line it gives you.

When to hand over

- A big piece of work is done, and the next one is different.
- The session has already compacted once or twice.
- You're stopping for the day.

Where things live

The helper is in `~/.claude/handover-kit/`, the skills in `~/.claude/skills/`, and the notes in `~/.claude/handoffs/`, one folder per ID, with compact notes in its `compact` folder.

How long to keep notes

About 30 days, or less for sensitive work. Nothing is deleted automatically. To clean up, run `python3 ~/.claude/handover-kit/handoff_kit.py prune --older-than 30`, check the list, then run it again with `--yes`.

If a note is wrong, the next session will be wrong in exactly the same way. Read it before you trust it.

09 · UNDO IT

Uninstall and roll back

The uninstall prompt removes only what this kit installed, and keeps your notes.

- It removes the three skills, but only if they're still the exact files the kit installed. A skill you edited, or one that isn't the kit's, stays put and is listed.
- It removes only this kit's hook from `settings.json`, after a backup, and checks that every other setting and hook is unchanged.
- It removes `~/.claude/handover-kit`.
- It keeps your notes in `~/.claude/handoffs` and the backups in `~/.claude/handover-kit-backups`. Delete those yourself if you want them gone.

To go back to your settings from before you installed, copy `settings.json` from the oldest folder in `~/.claude/handover-kit-backups` that has one. That also undoes any setting you changed since, so compare the two first. If no backup has one, you had no `settings.json` before, and the uninstall prompt is all you need.

Coming from version 1.0

Each install prompt recognises the v1.0 version of its own skill and replaces it after a backup, and prompt 3 also replaces the v1.0 hook entry. Your v1.0 notes still show up in the list, but they have no state record, so resuming one gives `UNVERIFIABLE`.

UNINSTALL IF YOU NEED IT

Uninstall and roll back

```
Remove the handover kit v1.1 from Claude Code. Remove only what this kit
installed, and keep my notes.
HELPER below means: python3 ~/.claude/handover-kit/handoff_kit.py

1. Run: HELPER uninstall --dry-run
   Show me the plan exactly as printed and ask me to confirm.
2. Only after I confirm, run: HELPER uninstall
   Show me its output.
3. Show me the SessionStart part of ~/.claude/settings.json, if the file
   exists, so I can see that my other hooks are still there.
4. Tell me where the backups are (~/.claude/handover-kit-backups) and
   that my notes are still in ~/.claude/handoffs. Do not delete either.
```

10 · FOR TEAMS

None of this is specific to AI

Every team runs handovers: shift changes, on-call rotations, someone going on leave, a project changing owners. The same three rules apply.

1 **Hand over by name.**
"It's in the doc" isn't a handover. Link the exact doc, the exact ticket, the exact dashboard.

2 **Check from the receiving side.**
Ask the person taking over what they'd do first. If they can't tell you, the handover isn't done yet.

3 **When unsure, ask.**
If there are two versions of the truth, say so out loud. Don't leave them to guess which one is right.

If you're handing work to a person, the next prompt gets you most of the way there.

PROMPT 5 ANY TIME

A handover note for a teammate

```
I'm handing this work over to a teammate. Write a handover note they
can act on without having to ask me anything. Use what you know from
this session, and ask me for anything you can't see.

Include:
- the goal, and what done looks like
- where things stand, marking what is confirmed and what is assumed
- the decisions we made, and why
- what we tried and dropped, and why
- open problems and risks
- the very next step, small enough to start on straight away
- the exact branch, commit, ticket or document each fact comes from
- who to ask about what

Do not include passwords, tokens, keys or customer data. Keep it to one
page, in plain language. Then list the three questions they would most
likely still ask, so I can answer them in the note before I send it.
```

ABOUT THIS VERSION

Version 1.1

Version 1.1 · 18 November 2026

Replaces version 1.0 from earlier the same month. If you installed 1.0, run the v1.1 prompts: they upgrade it in place.

What changed

- A helper script now does every check, so a status is measured, not judged by the model. The check now catches a file that changed again while its git status stayed the same.
- Statuses say exactly what was verified: `VERIFIED_MATCH`, `DRIFT`, `UNVERIFIABLE` and four failure states.
- Compact notes are restored once, only into their own session, and labelled as history when they're old or the code has changed.
- Safer installs: the kit comes as files with checksums instead of being retyped from a prompt, and every install shows a plan first, never overwrites, takes timestamped backups and avoids duplicate hooks. There's an uninstall.
- Private files: notes and folders are readable only by you, notes that look like they hold secrets are refused, and secret-looking files are never fingerprinted.
- Clearer claims about cost, about what a note keeps, and about moving work between machines.

Works with

Claude Code on macOS or Linux, `python3` 3.9 or newer, and `git`. Tested on macOS with Claude Code 2.1.327, Python 3.9 and 3.15, and git 2.54. Not yet tested on Linux. Windows isn't supported, and neither is a custom `CLAUDE_CONFIG_DIR`.

ABOUT THIS VERSION

How it was tested

Everything was tested on macOS, against a test project with two clones of the same repository, which is the situation that broke my setup. The automated tests in the companion run in throwaway folders. The live checks used real Claude Code sessions. Each layer was proven to a different degree, so here they are separately.

Installing	Every install and uninstall step was tested through the helper the prompts run, in throwaway home folders. The prompts haven't yet been pasted into a fresh Claude Code account end to end.
Writing a note	Tested in live sessions, with the work in a different clone from the one the session started in. The note named the right checkout.
Record integrity	Every note is checked against its recorded checksum before it's used. Damaged or edited records come back as `INVALID_HANDOFF`.
Checking the code	Automated tests for each kind of change, including changing an already-changed file again. Live, that case showed up as `DRIFT`.
Picking the right note	By ID only. Live, a missing ID listed the notes and asked, and a wrong or malformed ID loaded nothing.
Resuming	Tested with scripted sessions and a real compaction: the compact note came back once, then never again. The interactive `/clear` in the terminal wasn't scripted. A match never proves the note itself is right.

Limits

- A note is a summary. It keeps what the session wrote down, not the whole conversation.
- A match means the code is where the note left it, not that the note is right.
- Ignored files and secret-looking files aren't compared.
- The secret check catches common shapes of keys and passwords, not every one.
- Notes don't move between machines on their own.

Feedback

If something breaks or reads wrong, send me a DM on Instagram at @manishkumarglobal with what you ran and what you saw. Leave out anything private from your project.

I’m Manish Kumar.

I've spent years leading SRE and cloud teams as a director, and these days I build products as a founder and CEO.

The Production Gap is my series about what happens between a demo that works and a system you can trust in production.

If something in this guide breaks for you, tell me. I'd rather fix it than have you work around it.

Follow @manishkumarglobal for the rest of the series.