PROMPTS & SKILLS FOR CLAUDE CODE

Ten prompts for the session that drifted, hallucinated or dug itself into a hole.

Terminal Craft

INDEPENDENT FIELD GUIDE

The Debug & Recover Pack

Ten prompts for the session that drifted, hallucinated or dug itself into a hole.

TERMINAL CRAFT

WELCOME

Before You Begin

Twenty minutes ago something small went wrong. The obvious fix did not work, so you asked for another attempt, and then another, and each one edited a little more. Now the original problem is buried under four speculative changes nobody can account for, the test suite is failing in a new way, and you are seriously considering throwing the whole session away. That escalation is not a failure of the tool and it is not a failure of yours. It is what happens when the instinct to keep going meets a system that will always produce another plausible attempt. What breaks the pattern is not a cleverer instruction. It is stopping, taking an inventory of what has already changed, and refusing to accept any claim that has not been shown to exist in your repository. This pack is ten prompts for exactly that. There is an hour-by-hour plan for the first ten minutes, the questions that force claims to be backed by quoted lines, the move that breaks a loop, and a recovery method that separates the work worth keeping from the changes to revert, because most bad sessions contain much more salvageable work than the frustration suggests. Two notes. This is an independent publication with no affiliation to any AI vendor, and no software is included. And it is the 2026

Edition: what drifts and how it drifts shifts between model versions, so the final chapter is about the setup that makes drift rare rather than a list of symptoms that will date.

— The Terminal Craft Team

HOW TO USE THIS GUIDE

Getting the Most From This Book

This is a reference, not a reading list. The prompts work best adopted a few at a time until they are automatic. Here's how to get the most from the pages ahead.

1. **Use it as a reference.**
   Every chapter ends with a *Try This* exercise and a set of *Key Takeaways*. Read the chapter once, then come back to the prompts when you need them.

2. **Adopt a few at a time.**
   Take four prompts and use them until they are automatic. Reaching for forty at once is the fastest way to use none of them.

3. **Read the 'not for' line.**
   A structured prompt on a trivial task is pure overhead. Knowing when not to use one matters as much as knowing the wording.

4 **Adapt, don't discard.**

When a model version changes and a prompt stops landing, rebuild it from the four-part frame rather than hunting for a new download.

Whenever you're ready, turn the page and let's begin.

CONTENTS

What’s Inside

CHAPTER 1

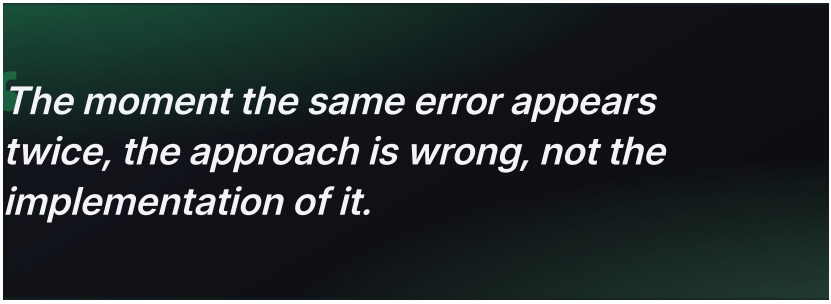
How Sessions Go Wrong

> Three failure modes cover almost everything. Recognising which one you are in decides which prompt to reach for, and reaching for the wrong one is how ten minutes becomes forty.

Drift

Drift is when each step is individually reasonable and the sequence is not. A fix requires a small refactor, the refactor exposes an inconsistency, fixing that touches a shared helper, and forty minutes later the change spans nine files and none of it was requested.

The signature is that you cannot answer 'why is this file in the diff?' for at least two of them. Nothing errored, nothing looked wrong at any single step, and the result is still not what you asked for.



Loops

A loop is the same approach tried repeatedly with cosmetic variation. Attempt one fails, attempt two renames a variable and fails identically, attempt three adds a conditional and fails identically again. The tell is that the failure message stops changing.

Loops are cheap to detect and expensive to sit through. The moment the same error appears twice, the approach is wrong, not the implementation of it.

Confident invention

The third mode is a claim about your codebase, your framework or an API that is fluent, specific and untrue. A flag that does not exist, a method with the wrong signature, a file that was never there. It is the most dangerous because it is the most persuasive.

The defence is structural rather than clever: require a quoted line for every claim, and give explicit permission to answer 'no evidence'. Instructions not to invent things work far less well than making verification the easier path.

WORKING OUT WHICH FAILURE YOU ARE IN

The Mode Check

Before we continue: is this drift (files changed that I did not ask about), a loop (the same failure twice), or an unverified claim? Answer with one word and the evidence for it.

WHY IT WORKS

The three modes need different responses, and reaching for the wrong prompt wastes the ten minutes that matter most.

USE WHEN The moment something feels wrong

PAIRS WITH The Full Stop

Situations where the cause is already obvious.

Try This: Name Your Last One

Pattern recognition is faster when you have named the pattern once.

1. Think of the last session that went badly.
2. Decide which of the three modes it was.
3. Note what you did next, and whether it helped.
4. Read the chapter for that mode first. It is the one you are prone to.

KEY TAKEAWAYS

- Drift: every step reasonable, the sequence not. You cannot explain two files in the diff.
- Loop: the failure message stops changing. The approach is wrong, not its implementation.
- Invention: fluent, specific and untrue. Require quoted lines rather than asking for honesty.

CHAPTER 2

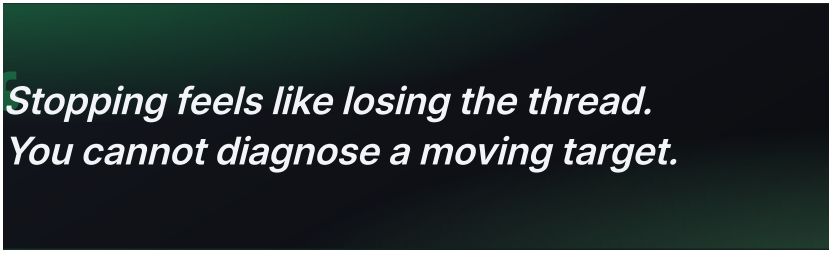
The First Ten Minutes

> The first move in every bad session is the same, and it is the hardest one: stop. Not pause to think of a better instruction. Stop editing, and find out what is currently true.

Why stopping feels wrong and is right

Stopping feels like losing the thread, which is exactly backwards. You cannot diagnose a moving target, and every further speculative edit enlarges the eventual cleanup while burying the original problem deeper.

There is also a sunk-cost pull: twenty minutes in, abandoning the current approach feels like wasting them. Those twenty minutes are already spent. The only question is whether the next twenty join them.



Inventory before diagnosis

Before asking what went wrong, establish what changed. A file-by-file list with a one-line reason for each is the single most useful artefact in a bad session, because it converts a vague mess into something you can reason about.

It is also where drift becomes undeniable. When two of the reasons are 'to support the change in the previous file', you have found your sequence.

Then, and only then, diagnose

With edits stopped and an inventory in hand, ask what failed, with the requirement that every claim cites a real line. Not 'the router was misconfigured' but 'router.ts:44 sets the path before the guard runs'.

If a claim cannot be cited, it is a hypothesis. Hypotheses are fine, but they must be labelled, because an uncited hypothesis is exactly what the next twenty speculative minutes will be built on.

THE FIRST MOVE, ALWAYS

The Full Stop

Stop editing immediately. List every file you have changed this session with a one-line reason for each. Change nothing else until I reply.

WHY IT WORKS

You cannot diagnose a moving target, and the inventory turns a vague mess into something you can reason about.

USE WHEN The moment you notice trouble

PAIRS WITH The Change Audit

Sessions that are going fine. It interrupts good flow.

SEPARATING ASKED-FOR FROM INVENTED

The Change Audit

For each changed file, mark whether I explicitly asked for it, or whether it followed from another change. For the second group, name the change it followed from.

WHY IT WORKS

It exposes drift as a chain rather than a list, and the chain shows exactly where the sequence left your request.

USE WHEN Right after The Full Stop

PAIRS WITH The Full Stop

Single-file sessions.

FINDING OUT WHAT ACTUALLY FAILED

The Cited Diagnosis

What failed, and why? Every claim must cite file:line from this repository. Label anything you cannot cite as HYPOTHESIS. Do not fix anything yet.

WHY IT WORKS

Labelling uncited claims stops the next twenty minutes being built on something that was never checked.

USE WHEN After the inventory **PAIRS WITH** The Change Audit

Failures with an obvious, single, visible cause.

Try This: Paste It From Memory

The prompt you have to look up is the prompt you will not use at minute twenty.

1. Copy The Full Stop somewhere you can paste in two seconds.
2. Use it the next time a session starts feeling wrong, before anything clever.
3. Read the inventory properly rather than skimming it.
4. Count how many files you could not have named beforehand.

KEY TAKEAWAYS

- The twenty minutes are already spent. The only question is whether the next twenty join them.
- Inventory first, diagnosis second, and the inventory is what makes drift undeniable.
- Uncited claims get labelled as hypotheses, or they become the next bad decision.

CHAPTER 3

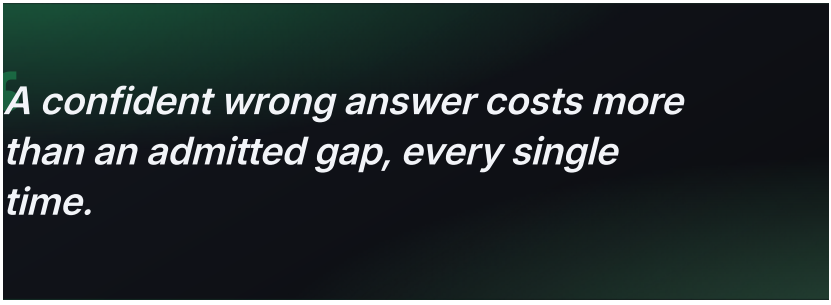
The Verification Prompts

> This chapter is about making claims checkable. It is the highest-value habit in the pack, because a confident wrong answer costs more than an admitted gap every single time.

Citations beat instructions

Telling a model not to make things up is weak. Requiring every claim to be accompanied by a quoted line from your repository is strong, because it changes what a compliant answer looks like rather than appealing to intent.

The mechanism is simple: with a citation requirement, producing a confident invention requires also producing a fake citation, and the fake citation is trivially checkable in a way the prose was not.



Give an escape hatch

Always pair the requirement with explicit permission to fail it. 'If you cannot find the line, say no evidence' produces honest answers much more reliably than any prohibition, because it makes the honest response an acceptable one rather than an admission of failure.

Without the hatch, the pressure to produce something pushes toward invention. With it, 'no evidence' becomes a normal, correct output.

Spot-check the citations

Open two or three of the cited lines yourself, at random, especially early in a session with an unfamiliar repository. You are not auditing everything. You are calibrating how much the citations can be trusted at all.

That calibration is worth doing once per repository. It takes ninety seconds and it tells you how sceptical to be for the rest of the day.

ANY CLAIM ABOUT YOUR CODEBASE

The Evidence Rule

For every claim you make about this codebase, quote the exact file and line that supports it. If you cannot find one, write 'no evidence' instead of explaining.

WHY IT WORKS

It changes what a compliant answer looks like, and a fake citation is checkable in a way confident prose is not.

USE WHEN Any unfamiliar territory **PAIRS WITH** The API Reality Check

Discussions of approach, where there is nothing to cite.

A FLAG, METHOD OR OPTION YOU HAVE NOT SEEN

The API Reality Check

For each API, flag or option you just referenced: show me where it is defined in this repo or quote the documentation. If you are inferring it exists, say so explicitly.

WHY IT WORKS

Invented parameters are the most common form of confident invention and the easiest to catch by demanding a definition.

USE WHEN Any unfamiliar library surface **PAIRS WITH** The Evidence Rule

Standard-library calls you already know.

CALIBRATING HOW MUCH TO TRUST CITATIONS

The Spot Check

Pick three of the citations you just gave at random and quote the surrounding five lines of each, verbatim.

WHY IT WORKS

It is a cheap test of whether the citations are real, and it calibrates your scepticism for the rest of the session.

USE WHEN Once per unfamiliar repository

PAIRS WITH The Evidence Rule

Repositories you know line by line.

Try This: Verify Three

Calibration takes ninety seconds and changes how you read everything after it.

1. Run The Evidence Rule on your next question about an unfamiliar repo.
2. Open three of the cited lines yourself.
3. Note how many were accurate.
4. Use that number to decide how much to check for the rest of the day.

KEY TAKEAWAYS

- Requiring citations beats instructing honesty, because it changes what compliance looks like.
- Always pair the requirement with permission to answer 'no evidence'.
- Spot-check three citations once per repository to calibrate your scepticism.

CHAPTER 4

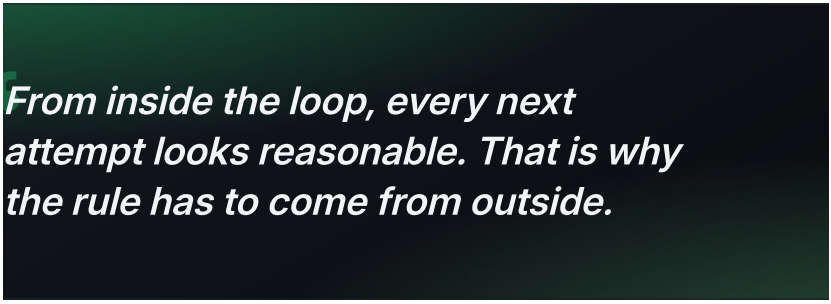
Breaking Loops

> A loop is the cheapest failure to detect and the most tedious to sit through. The fix is a rule you apply from outside, because from inside the loop every next attempt looks reasonable.

Forbid the approach, not the attempt

Saying 'try again' produces the same approach with different names. Saying 'do not attempt this approach again' is what actually changes the search. The distinction sounds pedantic and it is the whole difference between attempt three and a solution.

Follow it with a demand for alternatives: three different hypotheses about the root cause, each with the cheapest test that would confirm or kill it. Three forces different thinking; one just produces the same idea restated.



Reduce before you theorise

When hypotheses are not converging, stop theorising and reduce. The smallest input and shortest path that still reproduces the failure usually reveals the cause on its own, and it always makes verification of a fix reliable.

Reduction is also the honest test of whether the problem is understood. If it cannot be reduced, it is not yet understood, and any fix at that point is a coincidence waiting to be discovered later.

Change the frame

If three hypotheses all fail, the framing is probably wrong. Ask what would have to be true for the current behaviour to be correct. The question forces a different starting point rather than another variation of the same one.

This is where surprising causes surface: a cached artefact, a different environment, a test asserting the wrong thing, a file that is not the file being loaded.

THE SECOND IDENTICAL FAILURE

The Loop Breaker

You have tried this approach twice and it failed the same way both times. Do not attempt it again. Give three different hypotheses for the root cause and the cheapest test for each.

WHY IT WORKS

Forbidding the approach, rather than the attempt, is what changes the search. Three hypotheses force different thinking.

USE WHEN The second identical failure

PAIRS WITH The Minimal Reproduction

The first failure, where one retry is usually legitimate.

HYPOTHESES THAT WILL NOT CONVERGE

The Minimal Reproduction

Reduce this to the smallest input and shortest code path that still reproduces the failure. Remove everything not required. Show me the reduced case before proposing any fix.

WHY IT WORKS

Reduction usually exposes the cause by itself, and it is the only reliable way to know a fix actually worked.

USE WHEN When theorising stalls

PAIRS WITH The Loop Breaker

Failures that only appear under real load or timing.

THREE HYPOTHESES, ALL WRONG

The Reframe

What would have to be true for the current behaviour to be CORRECT? List every assumption that would make this the right output rather than a bug.

WHY IT WORKS

It forces a different starting point, which is where caching, environment and wrong-file-loaded causes finally surface.

USE WHEN After three failed hypotheses

PAIRS WITH The Minimal Reproduction

Straightforward bugs with a visible cause.

Try This: Say It Out Loud

The rule is easy to state and hard to apply from inside the loop.

1. Next time a fix fails twice identically, stop before the third attempt.
2. Paste The Loop Breaker instead of rephrasing your request.
3. Read all three hypotheses before choosing one.
4. Run the cheapest test first, not the most interesting one.

KEY TAKEAWAYS

- Forbid the approach, not the attempt. 'Try again' returns the same idea renamed.
- If it cannot be reduced, it is not yet understood, and a fix would be a coincidence.
- When three hypotheses fail, the framing is wrong. Ask what would make this correct.

CHAPTER 5

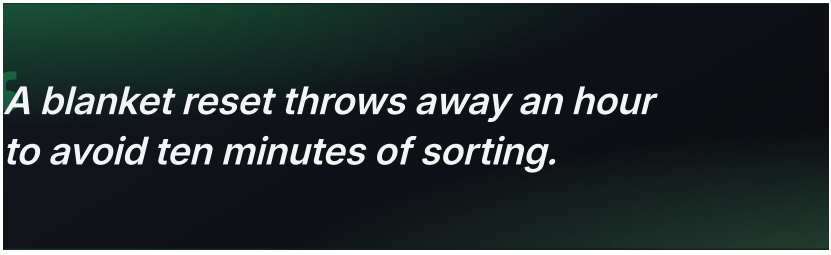
The Recovery Move

> The session is stopped, the inventory exists and the cause is understood. Now the question is what survives, and the answer is almost always more than the frustration suggests.

Sort before you revert

The instinct after a bad session is to reset everything and start clean. Sometimes that is right. Far more often a good portion of the work was fine and only the last few speculative changes are the problem, and a blanket reset throws away an hour to avoid ten minutes of sorting.

Three lists: keep, revert, unsure. The unsure pile is usually small and is where your judgement is worth spending. Everything else is mechanical.



Make the revert precise

A vague intention to undo the recent changes becomes an over-broad revert under pressure. Ask for the exact file list or command that undoes only the speculative work, leaving the earlier changes intact.

Precision here is what makes partial recovery happen rather than being a nice idea you abandon because the mechanics were unclear at the wrong moment.

Sometimes a clean session is the recovery

When the context is polluted with contradictory instructions, half-applied approaches and a long argument about a wrong hypothesis, a fresh session with a good brief beats continuing. The brief is the important part: goal, constraints, what was learned, what to avoid.

Written that way, starting over stops being a loss. The learning survives even though the transcript does not, which is the only thing that made the bad session worth anything.

DECIDING WHAT SURVIVES

The Salvage Sort

Split every change from this session into three lists: keep, revert, unsure. For each item in 'unsure', give the one question that would settle it.

WHY IT WORKS

Most bad sessions contain more good work than the frustration suggests, and the unsure pile is where judgement is worth spending.

USE WHEN Before reverting anything **PAIRS WITH** The Revert Boundary

Sessions where almost nothing was completed.

UNDOING ONLY THE SPECULATIVE PART

The Revert Boundary

Give me the exact command or file list that undoes only the changes in the 'revert' list, leaving everything in 'keep' untouched. Show it before running anything.

WHY IT WORKS

Vague intentions become over-broad reverts under pressure; precision is what makes partial recovery real.

USE WHEN Right after the sort **PAIRS WITH** The Salvage Sort

Cases where a full reset is simpler.

WHEN THE CONTEXT IS BEYOND REPAIR

The Fresh Start Brief

We are restarting this in a clean session. Write the brief I should paste there: goal, constraints, what we learned, what to avoid, and which files are already correct.

WHY IT WORKS

It carries the learning across the reset, which is the only thing that made the bad session worth anything.

USE WHEN Contradictory or polluted context

PAIRS WITH The Salvage Sort

Sessions needing only a course correction.

Try This: Sort Before You Reset

Ten minutes of sorting routinely saves an hour of redoing.

1. The next time you are about to reset everything, run The Salvage Sort first.
2. Count how many changes land in 'keep'.
3. Answer the questions attached to the 'unsure' items.
4. Only then decide whether a full reset was actually warranted.

KEY TAKEAWAYS

- Sort into keep, revert and unsure before touching anything.
- Ask for the exact revert boundary, because vague intentions become over-broad resets.
- A fresh session with a written brief carries the learning across the reset.

CHAPTER 6

Preventing The Next One

> Recovery is a skill worth having and a poor substitute for not needing it. Almost every bad session in this book traces back to something that was not established before the work started.

Most drift is an unset boundary

Drift happens when the agent is optimising for a good result and nobody drew a line around what 'result' includes. A single sentence naming what must not change prevents the large majority of it, and it costs four seconds.

Make it permanent rather than remembering it. The sessions where you forget to type the boundary are exactly the tired ones where drift is most likely.

Noticing a loop from inside is the hard part. That's the whole argument for making the rule structural.

Most loops are an unset stopping rule

A stopping rule stated in advance turns a twenty-minute loop into a two-minute one: after two identical failures, stop and propose alternatives. Installed as a standard it applies without you having to notice the loop, which is the hard part.

Noticing is difficult from inside. That is the entire argument for making the rule structural rather than a matter of vigilance.

Most invention is an unasked-for citation

A standing requirement that claims about the codebase carry a quoted line removes the majority of confident invention before it reaches a decision. It is the cheapest standard in this whole system and the one people install last.

Between a boundary, a stopping rule and a citation requirement, the sessions this book is about become rare rather than routine. That is a better outcome than being excellent at recovery.

MAKING BAD SESSIONS RARE

The Three Standards

Add these as permanent project standards: (1) never change anything outside the stated task, (2) after two identical failures stop and propose three alternatives, (3) every claim about this codebase cites file:line or says 'no evidence'.

WHY IT WORKS

The three of them address drift, loops and invention structurally, rather than relying on you noticing in the moment.

USE WHEN Once, per project **PAIRS WITH** The Post-Mortem

Nothing. This is the closing recommendation of the pack.

LEARNING FROM A BAD SESSION

The Post-Mortem

That session went badly. In three lines: what was the first moment it left my request, what would have caught it, and what standard would have prevented it?

WHY IT WORKS

It converts an annoying hour into one specific, installable change rather than a vague resolution to be more careful.

USE WHEN After every bad session **PAIRS WITH** The Three Standards

Minor hiccups that resolved in a minute.

Try This: Install the Three

Fifteen minutes now, against the next four bad sessions.

1. Add the boundary, the stopping rule and the citation requirement as project standards.
2. Run a week of ordinary work without changing anything else.
3. Note whether you reached for this book at all during that week.
4. After the next bad session, run The Post-Mortem and install what it names.

KEY TAKEAWAYS

- Most drift is an unset boundary. Most loops are an unset stopping rule.
- Most invention is an unasked-for citation, and that standard is installed last by almost everyone.
- Being excellent at recovery is worse than rarely needing it.

PUT IT INTO PRACTICE

Your Action Plan

The emergency procedure on one page. Most of it is preparation done before anything goes wrong, which is exactly why it works.

- [ ] Copy The Full Stop somewhere you can paste it in two seconds.
- [ ] Put The Evidence Rule directly underneath it.
- [ ] The moment something feels wrong, run The Mode Check: drift, loop, or invention?
- [ ] Stop editing before diagnosing. Always inventory first.
- [ ] Mark which changed files you asked for and which followed from another change.
- [ ] Require file:line citations, and allow 'no evidence' as an answer.
- [ ] Spot-check three citations once per unfamiliar repository.
- [ ] On the second identical failure, forbid the approach. Do not rephrase the request.
- [ ] If hypotheses stall, reduce to the smallest reproduction before theorising further.
- [ ] Sort into keep / revert / unsure before undoing anything.
- [ ] Ask for the exact revert boundary rather than reverting broadly under pressure.
- [ ] Install the three standards from chapter six so these sessions become rare.

FINAL WORDS

Where You Go From Here

The most valuable thing in this pack is not any of the ten prompts. It is the first ten minutes, the moment when the instinct says keep going and the procedure says stop, inventory, and find out what is true. Every session that ended badly in your experience escalated through one of those moments. Keep The Full Stop and The Evidence Rule where you can paste them without looking, because at minute twenty you will not go hunting for a book. Sort before you reset, because most sessions contain more good work than they feel like they do. And when it is over, spend three minutes on the post-mortem: the first moment it left your request, what would have caught it, what standard would have prevented it. Then install that standard. Do that four or five times and this book becomes something you rarely open, which is the outcome it was written for. Being excellent at recovery is a consolation prize. Not needing it is the actual goal.

Breathe. Begin. Return.
