A CLAUDE CODE TIP · CARSON · 2026

Stop Claude Code from getting *dumber*.

Your context window rots. Here are three commands and one workflow that fix it — in under ten minutes.

MOVES 3

READ TIME
8 minutes

COMMANDS
/context · /statusline · handoff.md

SKILL LEVEL
Beginner

Contents

Long Claude Code sessions get dumber over time. Not because the model changes — but because everything you've tried, every dead end, every wrong turn stays in the conversation. ***And /compact won't save you.*** It summarizes the rotted conversation — and the shape of the rot rides along into the summary. This guide is the workflow that fixes it.

OVERVIEW

Three commands. One workflow.

Master `/statusline`, `/context`, and the `handoff.md` pattern. Catch context rot before it pulls your work under, then reset cleanly with zero lost state.

01 Watch context fill in real time without leaving the terminal.

02 See exactly which subsystem is bloating your window — and what's prunable.

03 Write a clean briefing the next session reads in 30 seconds.

04 Recover from rot in under a minute, with zero lost progress.

TOTAL SETUP TIME

Under ten minutes. From zero to a workflow you'll run every day.

PREREQUISITES

Before you start

Everything in this guide ships with Claude Code. You don't need a plan upgrade, an MCP server, or any external tool — just a terminal and a project you're actively working in.

NEED	WHAT FOR	HOW TO VERIFY
Claude Code v1.0.86+	The `/context` command was introduced in this release	`claude --version`
A project	Anywhere you'd run `/clear`	Toy project is fine
Terminal & shell	To customize `/statusline` with a script	bash, zsh, or fish
~ 8 minutes	Reading + setup	Pour the coffee

> **TIP**
>
> If `claude --version` shows 1.0.85 or older, run `claude upgrade` first. The `/context` command landed in 1.0.86 — earlier versions only have `/compact`, which is the trap this guide is about.

READ THIS FIRST

Compact preserves the rot.

The most common reflex when context fills up is also the wrong one.

THE ONE RULE

`/compact` summarizes the same broken conversation. ***Every dead end, every wrong turn, every bad guess*** gets pulled forward into the next reply — just in shorter form.

Compaction looks like cleanup. It isn't. It's a lossy summary of the rotted thinking that already steered Claude wrong — and the summary still tilts the next reply toward the same dead ends.

What you actually want is a clean break. Write down what's salvageable, run `/clear`, and start fresh in a new session. That's the entire workflow this guide covers.

```
~ claude code · 142k / 200k tokens

> /compact

  Compacting conversation...

  ⚠  still preserves: dead ends · wrong turns · bad guesses
```

The fix, in three commands

Three commands, in order. `/statusline` runs always-on at the bottom of every session. `/context` diagnoses when something feels slow. And `handoff.md` — written by Claude itself, in 30 seconds — is the briefing the next session reads after you `/clear`. The rest of this guide walks through each one.

01

STEP ONE

Write a handoff.md

A six-section markdown file that gives the next session everything it needs to pick up where you left off — in 30 seconds of reading.

A — The template

Before you run `/clear`, ask Claude to write a `handoff.md` at the project root with these six sections. The order matters: Goal first, because it's what the next session loads its mental model around. Failures near the end, because they're the cheap-to-skip part if context is tight.

~ claude code

```
> Write a handoff.md in the project root with these six sections:
  Goal, Current state, Files in play, What's changed, What's failed (don't retry),
What to try next.
```

B — A worked example

Here's the file Claude would produce for an in-flight auth refactor. Note the specificity — every line names a file, a command, or a decision with its reason. That's the bar. Vague handoff = vague resumption.

handoff.md · auth-refactor

```
# Handoff — auth refactor

## Goal
Ship cookie-based auth to replace the JWT flow before Friday's compliance review.

## Current state
Cookie path is ~80% wired. Login + logout work locally; middleware doesn't yet read
the new cookie. JWT tests still pass; no cookie tests yet.

## Files in play
- src/auth/session.ts (cookie set/clear — done)
- middleware/auth.ts (needs cookie reader wired in)
- tests/auth.test.ts (only old JWT cases)

## What's changed (with why)
- JWT → session cookies (legal flagged token-storage compliance)
- Dropped Redis cache (added 200ms p95 — not worth it)

## What's failed — don't retry
- Secure + SameSite=None on localhost → cookies dropped silently
- next/headers in middleware → edge runtime doesn't expose it

## What to try next
Wire the cookie reader into middleware/auth.ts:42, then run
pnpm test:auth. If the session-bound test still fails, check
the cookie attribute combo first — Secure=true + SameSite=Lax
is the only combo that works locally and in preview.
```

TIP

Put `handoff.md` in the project root, not in `~/.claude/`. The file moves with the repo, so a teammate or a second machine picks up the same state automatically.

02

STEP TWO

Inspect with /context

See exactly where your tokens are going, broken down by subsystem. Run it when something starts to feel slow.

A — Run the command

The `/context` command (introduced in Claude Code v1.0.86) breaks down every component eating your context window — system prompt, tools, MCP servers, skills, auto-memory, and the conversation itself. Type it at any prompt and read the breakdown.

~ claude code · /context output

```
> /context

Context usage — 142,310 / 200,000 tokens (71%)

Component                  Tokens     %
System prompt               4,210    2.1%
Tools (built-in)            8,840    4.4%
MCP tools (listed)            120    0.1%
Auto memory (MEMORY.md)       680    0.3%
Skills (descriptions)       3,200    1.6%
Conversation history      125,260   62.6%   ← the rot zone

Suggestion: handoff + /clear if you're seeing repeated mistakes.
```

B — How to read it

Three bands matter most. **System prompt + tools + MCP** are fixed overhead (~5–15%) — you can't trim these mid-session. **Memory + skills** load on demand and stay modest. **Conversation history** is the variable. If it's over 40%, you're entering the rot zone. Over 60%, write the handoff now.

CAVEAT (GITHUB ISSUE #28167)

The displayed percentage counts *input* tokens only. The actual context-limit check includes output and cache tokens too — so "20% used" in `/context` can still hit the wall mid-response. Trust `/statusline`'s live indicator more than `/context`'s static read for the urgency signal.

03

STEP THREE

Customize /statusline

A shell script you write once. Lives at the bottom of every Claude Code session and color-codes your context usage live.

A — The fast path

The quickest setup: run `/statusline` with a description of what you want, and Claude Code generates the script and writes it into your settings. No bash required.

~ claude code

```
> /statusline show model name, git branch, context % with a color bar, and session
cost

✓ Wrote ~/.claude/statusline.sh
✓ Updated ~/.claude/settings.json
Restart Claude Code to see it.
```

B — Manual config (for full control)

If you want to hand-tune, drop a script at `~/.claude/statusline.sh` and reference it in `~/.claude/settings.json`. The script receives JSON session data on stdin (model, cwd, transcript path, cost, context usage) and prints whatever you want on stdout. Color-code the context band — green < 40%, yellow 40–70%, red > 70% — so the rot zone has a visual cue.

~/.claude/settings.json

```
{
  "statusLine": {
    "command": "~/.claude/statusline.sh"
  }
}
```

TIP

The community ships pre-built statuslines if you don't want to start from a shell script. `ccstatusline` is the most polished (themes + powerline support). `claude-statusline` has TOML config and cost tracking. Start with `/statusline`'s generated script and swap if you want themes later.

RUN IT EVERY DAY

The combined workflow

One loop, every long-session day. The three commands play different roles — monitor, diagnose, reset — and you only spend ~30 seconds on the reset.

WHEN	MOVE	WHY
Always on	Glance at `/statusline` at every commit	Catches the rot early — before Claude starts repeating mistakes
~ 50% context	Run `/context` to diagnose	Tells you what's prunable. MCP bloated? Drop unused servers. Skills exceed budget? Switch to "name-only" in `skillOverrides`
~ 70% or first repeat	Write `handoff.md`	Don't wait for desperation. The moment you notice Claude looping on the same fix is the moment to capture state
After handoff	`/clear`, then read `handoff.md` in a new session	Fresh agent, ~5% context, 30-second briefing. Same project, no rot

THE WHOLE LOOP, IN PLAIN ENGLISH

Monitor with */statusline*. Diagnose with */context*. Reset with *handoff.md*.

FURTHER READING

If you want to go deeper

THE RESEARCH

Chroma's *Context Rot* paper is the source of the problem framing. 18 frontier models tested; all degrade as input grows. Claude Opus 4 had the most pronounced gap between focused and full prompts on LongMemEval. `trychroma.com/research/context-rot`

THE OFFICIAL DOCS

`/statusline` : `code.claude.com/docs/en/statusline` · Context window: `code.claude.com/docs/en/context-window` · The percentage caveat lives in GitHub issue #28167.

PRE-BUILT STATUSLINES

If `/statusline` 's generated script isn't enough: `ccstatusline` (themes + powerline), `claude-statusline` (TOML + cost tracking), `ClaudeCodeStatusLine` (compact single-line).

PRACTITIONER PLAYBOOKS

Session-capture conventions and handoff variants: `claudecodehq.com/playbooks/session-capture-handoff` . Token-counting deep dive: `claudefa.st/blog/guide/mechanics/context-management` .

handoff.

Built from Chroma's research on context rot, the Claude Code docs, and many ugly sessions where /compact pretended to clean things up.

Carson · 2026-05-19

· MADE WITH CLAUDE CODE · TIKTOK · INSTAGRAM · YOUTUBE ·