# Caveman for Claude Code

Guide to installing Caveman, activating it inside Claude Code, and testing the real token savings for yourself.

| | |
|---|---|
| **What is it?** | A Claude Code skill that strips filler words from AI responses |
| **Claimed saving** | 75% of output tokens |
| **Actual saving** | ~4% of a full 100K session (only prose is compressed) |
| **Worth using?** | Yes — for speed & satisfaction. Not as a billing fix. |
| **Install time** | Under 60 seconds |

This guide is for everyone, as no coding background is required for the install steps.

**Part 1 — What Is Caveman?**

Caveman is a **Claude Code skill**—a small instruction file that changes how Claude writes its responses. When active, it drops polite filler and gets straight to the point.

| | **Normal Claude** | **Caveman Claude** |
|---|---|---|
| **Example** | "Sure! I'd be happy to help you with that. The reason your component is re-rendering is likely because you're creating a new object reference on each render cycle. I would recommend using useMemo to memoize the object." | "New object ref each render. Inline prop = new ref = re-render. Wrap in useMemo." |
| **Token Count** | 69 tokens | 19 tokens (73% less) |

**Important:** Caveman only compresses prose (conversational text). Code blocks, error messages, and tool calls are written normally.

**Parts 2 & 3 — Installation Guide**

Before starting, ensure you have:

1. **Claude Code installed** (The AI coding tool from Anthropic, install at: claude.ai/code).

2. **A terminal / command line** (Terminal on Mac/Linux or Command Prompt/PowerShell on Windows).

**Installation Steps (Under 60 seconds):**

1. Open your terminal and run the single command:
   claude install-skill JuliusBrussee/caveman
   - This downloads the skill file from GitHub. You only need to do this once.
   - *Expected Output:* Installing skill: JuliusBrussee/caveman followed by Skill installed successfully.
2. Open Claude Code in a project folder. If you don't have one, create an empty folder and run Claude:
   mkdir my-test-project followed by cd my-test-project and claude

**Part 4 — Activating Caveman**

Caveman is not on by default and must be activated per session.

| Action | Trigger Phrase(s) |
|---|---|
| **Activate Caveman** | /caveman, talk like caveman, caveman mode, or less tokens please |
| **Deactivate Caveman** | stop caveman or normal mode |

Claude confirms activation with a message like: "ok. caveman mode. short answers. no fluff. brain still big."

**Part 5 — Testing the Real Token Savings (Controlled Comparison)**

To verify the 60–75% token reduction on prose responses in isolation:

1. **Get Baseline:** In your Claude Code session, type a detailed question (e.g., Explain what a React hook is and when to use it. Be thorough.) and note the token count (Step A).
2. **Activate Caveman:** Send the command /caveman (Step B).
3. **Compare:** Ask the exact same question again and note the token count (Step C).

You should observe roughly 60–75% fewer tokens in the response compared to Step A, confirming the claim is accurate for prose in isolation.**Part 6 — Testing the Full Session Reality**

The overall session savings land around **~4%** because prose responses are only ~6% of all tokens in a real session.

To see this:

1. **Create Test Project:** Create a small project with files for Claude to read.
2. **Trigger File Read:** Ask Claude to read the files and suggest improvements

(Read all files in this project and explain what the code does. Then suggest 3 improvements.). Note the high token count from file reads (the real token hog).
3. **Repeat with Caveman:** Activate Caveman (/caveman) and ask the same thing.

You will notice that the file-read tokens and code suggestions remain identical; only the surrounding explanation text is shorter.

**Part 7 — What Actually Saves Tokens**

For real cost savings, focus on controlling the input context, as Caveman is primarily useful for speed and satisfaction.

| Strategy | Command/Example | Savings/Notes |
|---|---|---|
| **Targeted file read** | Only look at src/auth/login.js | Saves 20–40K tokens per session by limiting the codebase Claude reads. |
| **Summarize long sessions** | Summarise what we've built so far in 5 bullet points | Clears conversation history (20K tokens typical) by starting a fresh session with a summary. |
| **Scope your prompts tightly** | Good: Fix the null check on line 42 | Specific prompts reduce verbose answers and prevent Claude from touring the codebase. |
| **Use Caveman for explanations** | /caveman then ask your conceptual question | Delivers real savings when you genuinely need long explanations (where prose tokens are consumed). |

**Token Breakdown Reminder**

| Token Type | Typical Amount | Caveman Helps? |
|---|---|---|
| File reads / tool outputs | 50,000 (50%) | No |
| Conversation history | 20,000 (20%) | No |
| Tool call outputs | 10,000 (10%) | No |
| Code blocks | 9,000 (9%) | No |
| System prompt | 5,000 (5%) | No |
| **Prose responses** | **6,000 (6%)** | **YES — saves ~4,000 ...** |

**Quick Reference Card**

| Action | Command | Notes |
|---|---|---|

| | | |
|---|---|---|
| **Install Caveman** | claude install-skill JuliusBrussee/caveman | One-time setup |
| **Turn on Caveman** | /caveman or caveman mode | Per session |
| **Turn off Caveman** | stop caveman or normal mode | Per session |
| **Check token usage** | Shown in session output after each message | |
| **Summarise a long session** | Summarise this session in 5 bullets | Start fresh after |
| **Targeted file read** | Only read [filename] | Saves 20–40K tokens |