100

Claude Code Tips

. . .

Everything I learned on my journey with Claude Code — from the basics that changed how I work to advanced techniques most people never discover.

Compiled by

Manthan · Phaze AI

May 2026

Why This Guide Exists

I started using Claude Code the same way most people do — cautiously, unsure of what I could actually hand off to it. I asked for small things. I double-checked everything. I fixed bugs myself because it felt faster.

I was wrong on all counts.

Over hundreds of hours and millions of tokens, I discovered a completely different way of working. Not just using an AI assistant, but building a genuine workflow around it — with skills that auto-trigger, hooks that format and verify, agents that work in parallel, and context strategies that keep quality high across long sessions.

This guide is everything I wish I had on day one. It draws from my own experiments, from ykdojo's 45 tips, from shanraisshan's 84-tip repository, from the Claude Code cheat sheet, and from the community that keeps building on top of all of it.

100 tips. Organized into 11 categories. Start from the beginning or jump to whatever is most relevant to where you are in your own journey.

The goal isn't to make you dependent on a tool. It's to show you how deep the leverage actually goes.

— Manthan, Phaze AI

Table of Contents

Planning & Thinking

Tips 01–10

TIP 001

Always start with Plan Mode

Before Claude writes a single line, switch to Plan Mode (Shift+Tab). Let it lay out the full approach — architecture, edge cases, file structure — then review and execute. This one habit eliminates most back-and-forth.

TIP 002

Ask clarifying questions before planning

Prompt: 'Before you make a plan, ask me all the clarifying questions you have in detail.' Forces ambiguity to the surface early instead of mid-implementation when it's costly to fix.

TIP 003

Only ask for step one

Never say 'implement the whole feature.' Say 'what's step one?' Review it, then ask for step two. Tedious but it keeps Claude on rails and saves hours of course-correction.

TIP 004

Use ultrathink for hard problems

Add the keyword ultrathink to your prompt to trigger maximum reasoning effort. Reserve it for complex architectural decisions, tricky logic, or high-stakes output where you need Claude's deepest thinking.

TIP 005

Spin up a second Claude to review your plan

After Claude creates a plan, paste it into a fresh Claude instance and ask it to review the plan as a staff engineer. Or use a cross-model review — Gemini or Codex — for an independent second opinion.

TIP 006

Prototype over PRD

Build 20–30 quick versions instead of writing long specs. The cost of building with Claude is so low that iterating beats planning. Take many shots, learn fast, then formalize what works.

TIP 007

Phase-wise gated plans

Ask Claude to produce a phase-wise gated plan where each phase has unit, integration, and automation tests before moving to the next. Gates prevent cascading failures across a large build.

TIP 008

Use ASCII diagrams for architecture clarity

Ask Claude to represent your system architecture as ASCII diagrams. It forces clarity on both sides and gives Claude a stable visual anchor when working with complex structures.

TIP 009

Use Opus for planning, Sonnet for coding

Opus 4.7 gives you the deepest reasoning for architectural decisions. Sonnet 4.6 is the fastest and best for actual code generation. Combine them: plan in Opus, execute in Sonnet.

TIP 010

Reduce ambiguity before execution

The more specific your spec, the better the output. Before handing work to Claude, eliminate every 'it depends' and 'we'll figure it out later.' Ambiguity compounds in agentic runs.

Context Management

Tips 11–20

TIP 011

Compact at 50% — don't wait for degradation

Don't wait for auto-compaction. Run /compact manually when context hits ~50%. Claude's output quality degrades past that point before you notice. Use /context to watch usage as a colored grid.

TIP 012

Write a HANDOFF.md before clearing context

Before starting a fresh session, ask Claude: 'Write HANDOFF.md — what you tried, what worked, what didn't, so the next agent can pick up from here.' Then start fresh pointing only to that file.

TIP 013

Start a new conversation per topic

AI context is like milk — fresh and condensed is best. Start a new session for every new task or when you feel Claude's responses getting vague. Don't drag unrelated context into new problems.

TIP 014

Use /clear between unrelated tasks

Don't carry irrelevant context. /clear resets everything. Use it aggressively when switching tasks. Irrelevant context doesn't just waste tokens — it actively confuses Claude's responses.

TIP 015

Half-clone long conversations

When a conversation gets too long, use the half-clone script to keep only the later half. Preserves recent work and decisions while cutting tokens in half. A hook can auto-trigger this at 85% usage.

TIP 016

Fork conversations to branch safely

Use /branch [name] to fork the current conversation and preserve the original. Experiment with risky approaches without losing your good state. Return with /resume claude if the branch fails.

TIP 017

Use /compact with focus instructions

Don't just run /compact — give it focus instructions: '/compact focus on the auth module changes.' This steers what survives compaction and keeps the most relevant context.

TIP 018

Search past session history

Past sessions are stored as .jsonl files in ~/.claude/projects/. Ask Claude to search them directly for decisions, code snippets, or approaches you used in earlier sessions.

TIP 019

Use /context to visualize context health

/context shows current context usage as a colored grid with optimization suggestions. Green is safe, yellow is caution, red is danger. Check it before starting any long autonomous task.

TIP 020

Use /btw for side questions

/btw asks a quick side question without adding it to the main conversation flow. Great for quick lookups or clarifications that don't belong in the main thread.

CLAUDE.md & Memory

Tips 21–28

TIP 021

Keep CLAUDE.md under 200 lines

Only include what Claude couldn't possibly know on its own — business context, domain knowledge, naming conventions, internal data models. Everything else is noise that dilutes signal.

TIP 022

Only add things you repeat

Don't fill CLAUDE.md upfront. Start empty. When you catch yourself giving Claude the same instruction twice, add it. Let the file grow organically from real friction.

TIP 023

Use tags

Wrap domain-specific rules in tags to stop Claude from ignoring them as the file grows longer. Conditional tagging keeps rules from being treated as boilerplate.

TIP 024

Use .claude/rules/ for large projects

Split long CLAUDE.md instructions into separate rule files under .claude/rules/. Keeps things organized, modular, and easier to maintain as your project evolves.

TIP 025

Review CLAUDE.md periodically

Instructions go stale. What made sense 3 months ago may now be wrong, outdated, or redundant. Schedule a review and prune aggressively. A shorter file is a better file.

TIP 026

Test CLAUDE.md with a fresh start

Any developer should be able to launch Claude, say 'run the tests', and have it work on the first try. If it doesn't, your CLAUDE.md is missing essential setup, build, or test commands.

TIP 027

Use multiple CLAUDE.md files for monorepos

Ancestor + descendant loading means each subdirectory can have its own CLAUDE.md. Use this for monorepos where different packages have different rules, stacks, and conventions.

TIP 028

Create AGENTS.md for portability

AGENTS.md is becoming the standard across all AI coding tools. Put your core instructions there so they work across Claude Code, Cursor, Copilot, and others. Keep CLAUDE.md short and import AGENTS.md with @/AGENTS.md.

Prompting Mastery

Tips 29–38

TIP 029

Challenge Claude — don't babysit

'Prove to me this works.' 'Grill me on these changes and don't make a PR until I pass your test.' Push Claude to verify its own output instead of accepting the first response.

TIP 030

After a bad fix — ask for the elegant solution

'Knowing everything you know now, scrap this and implement the elegant solution.' Gets you out of iterative patch hell and forces a clean architectural rethink.

TIP 031

Ask Claude to double-check every claim

'Double-check everything you produced and make a table of what you were able to verify.' Works especially well for research, data, and technical claims where accuracy matters.

TIP 032

Let Claude fix its own bugs

Don't fix bugs yourself. Claude loses context of what went wrong. Paste the bug, say 'fix', and let Claude work. When it corrects its own mistakes, it builds a better mental model of your codebase.

TIP 033

Use voice input for faster prompting

Local transcription tools like Wispr Flow, MacWhisper, or superwhisper let you communicate 3–5x faster than typing. Claude handles transcription errors well — be loose and conversational.

TIP 034

Use @ to point to files directly

Instead of making Claude search around, use @/path/to/file.ts to load files directly into context. Otherwise Claude reads files in chunks, which is slower and uses more tokens.

TIP 035

Use ! to execute shell commands

If you need to run tests, typecheck, or build — type the CLI command with ! prefix instead of asking Claude to do it. Faster and more direct for things you already know how to run.

TIP 036

Paste screenshots when stuck

Ctrl+V pastes images directly. Share screenshots of bugs, error UI, console output, or design mockups. Much faster than describing what you're seeing — and Claude often catches things you miss.

TIP 037

Use Cmd+A to paste any page content

Select all content from Gmail, YouTube transcripts, Reddit threads, or any webpage and paste it directly into Claude. Combine with Playwright MCP for pages that require login.

TIP 038

Use Ctrl+G for long prompts

Ctrl+G opens your external editor for writing long, complex prompts. Write in a proper text editor with spellcheck and formatting, then submit. Better prompts for anything over 5 sentences.

Skills, Commands & Agents

Tips 39–48

TIP 039

Create skills for repetitive workflows

If you repeat the same instruction more than once a day, turn it into a skill. The real trick: write a precise description so Claude triggers it automatically without being asked.

TIP 040

Skill description = trigger, not summary

Write the skill description from the model's perspective: 'When should I fire this?' Not 'What does this do?' That framing is what enables auto-triggering and makes skills genuinely intelligent.

TIP 041

Build a Gotchas section in every skill

The highest-signal part of any skill. Add Claude's known failure points over time. This is how skills get smarter session by session — institutional memory for AI.

TIP 042

Don't railroad Claude in skills

Give goals and constraints, not prescriptive step-by-step instructions. Claude performs better with direction than with a script. Prescriptive instructions make skills brittle.

TIP 043

Use slash commands for inner-loop workflows

Anything you do multiple times a day becomes a command. /techdebt, /review, /handoff — check them into git under .claude/commands/ so the whole team benefits automatically.

TIP 044

Use subagents to keep main context clean

Say 'use subagents' to offload parallel work. One agent writes code while another reviews it. Separate context windows improve quality — one agent causes bugs, another finds them with fresh eyes.

TIP 045

Use context: fork in skills

Run a skill in an isolated subagent — main context only sees the final result, not all intermediate tool calls. Keeps your primary context lean and prevents skill noise from polluting the thread.

TIP 046

Feature-specific agents, not generic ones

Don't create a generic 'QA agent' or 'backend engineer.' Create agents with specific context and skills: 'auth-reviewer', 'database-migrator', 'api-contract-tester.' Specificity improves output dramatically.

TIP 047

Use skills in subfolders for monorepos

Skills live in .claude/skills/ by default but can be organized in subfolders. In monorepos, keep package-specific skills alongside their code so context stays relevant.

TIP 048

Include scripts and libraries in skills

Bundle helper scripts, reference examples, and library snippets inside skill folders. Claude composes from what's there rather than reconstructing boilerplate from scratch every time.

Hooks

Tips 49–54

TIP 049

PostToolUse hook for auto-formatting

Claude generates well-formatted code but the hook handles the last 10% — runs prettier, eslint, black, gofmt etc. after every file edit. Prevents silent CI failures from formatting drift.

TIP 050

Use a Stop hook to verify work

Add a Stop hook that nudges Claude to keep going or self-verify at the end of every turn. Catches lazy stopping mid-task and ensures Claude confirms results before handing back control.

TIP 051

Route permission requests to Opus via hook

Use a PreToolUse hook to send permission requests to Opus, which scans them for prompt injection attacks and auto-approves safe ones. Faster and smarter than manual approval every time.

TIP 052

Measure skill usage with PreToolUse hooks

Track which skills trigger often and which never fire. Use data to tune descriptions, fix broken triggers, and cut dead weight. You can't improve what you don't measure.

TIP 053

On-demand hooks for dangerous operations

Create /careful and /freeze hooks that block destructive commands or lock edits to specific directories. Use them when handing autonomous runs to Claude on sensitive codebases.

TIP 054

Embed !command in SKILL.md for dynamic output

Embed shell commands in skill files with !command syntax. Claude runs them on invocation and only sees the result — inject live git status, env vars, or API data into the skill context dynamically.

Workflow & Execution

Tips 55–66

TIP 055

Run /simplify before code review

Claude has a bias toward writing more code than needed. Before doing a review, run /simplify first. Three parallel review agents clean up bloat. You spend less time reviewing over-engineered output.

TIP 056

Commit often — at least once per hour

Commit as soon as a task completes, not at the end of the day. Clean checkpoints let you rewind safely when Claude goes off-track. One hour of work is acceptable to lose; a day is not.

TIP 057

Use Esc Esc to rewind, not fix-in-place

When Claude goes off-track, don't try to recover in the same context — it compounds errors. Hit Esc Esc, rewind to before it went wrong, and try again with a better prompt.

TIP 058

Do a retro at the end of each session

Ask: 'What did you learn during this session?' Save the output. This is how you build institutional knowledge over time — a growing record of what worked, what failed, and why.

TIP 059

Use /rename to label sessions

When running multiple Claude instances simultaneously, /rename each one: [refactor-auth], [fix-ci], [new-feature]. Makes context-switching between instances fast and clear.

TIP 060

Use /loop for recurring monitoring

/loop runs a prompt repeatedly while the session stays open — poll deployments, watch CI, babysit a PR. Runs up to 3 days. Pair with exponential backoff for token efficiency.

TIP 061

Run /security-review regularly

Run /security-review every so often on your current branch. Checks for injection vulnerabilities, auth issues, and data exposure in pending changes. A first pass — not a replacement for proper security review.

TIP 062

Audit approved commands with cc-safe

Run npx cc-safe . periodically to scan your settings for risky approved commands like rm -rf, sudo, or curl | sh. One bad approval can wipe your project. Review quarterly.

TIP 063

Use /doctor to diagnose installation issues

/doctor diagnoses installation, authentication, and configuration issues automatically. Run it first whenever Claude Code behaves unexpectedly — before spending time debugging manually.

TIP 064

Update Claude Code daily

Run the update command daily and start your day reading the changelog. Claude Code ships features fast. Missing even a week of updates means missing tools that could 10x your productivity.

TIP 065

Keep codebases clean — finish migrations

Partially migrated frameworks confuse models which may pick the wrong pattern. If you start migrating from one pattern to another, finish it. Half-migrated codebases produce inconsistent AI output.

TIP 066

Invest in product verification skills

Build signup-flow-driver, checkout-verifier, and other end-to-end verification skills. Spend a week perfecting them. Autonomous Claude runs need to verify their own work — these skills make that possible.

Parallel & Advanced

Tips 67–74

TIP 067

Use git worktrees for parallel work

Each worktree = a branch + a separate directory. Run multiple Claude instances on different features simultaneously without conflicts. Combine with the terminal tab cascade for max throughput.

TIP 068

Use sandbox mode for isolation

/sandbox gives Claude file and network isolation. Internally this reduces permission prompts by ~84%. Use it for exploratory, risky, or experimental tasks where mistakes must be contained.

TIP 069

Use /permissions with wildcards

Instead of --dangerously-skip-permissions, use targeted wildcards: Bash(npm run *), Edit(/docs/**). Safer, more controlled, and you can audit exactly what you've approved in settings.json.

TIP 070

Run risky tasks in containers

For long-running or experimental work, run Claude inside a Docker container with --dangerously-skip-permissions. If something goes wrong, it's sandboxed. Your local machine and repo stay safe.

TIP 071

Use exponential backoff for long jobs

For Docker builds, CI runs, or deploys — ask Claude to check status with increasing sleep intervals (1m, 2m, 4m, 8m...). More token-efficient than continuous polling and kinder to your context budget.

TIP 072

Multitask with terminal tab cascade

Open new terminal tabs to the right for new tasks, sweep left to right as tasks complete. Keep it to 3–4 active Claude instances at a time. More than that and you lose track of state.

TIP 073

Use Ctrl+B to send commands to background

Ctrl+B sends a running bash command to the background. Combine with subagents running in parallel to analyze different parts of a large codebase simultaneously without blocking your main thread.

TIP 074

Use /batch for large-scale parallel changes

/batch decomposes large tasks into 5–30 units and spawns one agent per unit in isolated git worktrees. Perfect for large-scale refactors, bulk file changes, or applying a pattern across a monorepo.

Debugging & Verification

Tips 75–82

TIP 075

Share screenshots when stuck

Paste screenshots of bugs, error UI, or console output directly with Ctrl+V. Claude sees and interprets them. Much faster than describing what you're seeing — and Claude often spots things you miss.

TIP 076

Use Playwright MCP for browser tasks

Playwright uses the accessibility tree instead of pixel coordinates — more reliable, faster, and less brittle. Only fall back to Claude's native browser integration when you need a pre-authenticated session.

TIP 077

Run terminal as a background task for logs

When debugging, ask Claude to run the process as a background task. It monitors logs periodically without blocking everything else. More efficient than watching logs in real time during a long run.

TIP 078

Use Chrome DevTools MCP for console logs

Pair Claude in Chrome MCP with Chrome DevTools to let Claude read console logs, network requests, and errors on its own. No more copying and pasting error stacks — Claude sees them directly.

TIP 079

Use cross-model QA

Use Gemini or Codex to review Claude's implementation — and vice versa. Models catch different classes of bugs. Same-model review has blind spots; cross-model review finds more issues.

TIP 080

Use /diff for interactive diff review

/diff opens an interactive diff viewer showing uncommitted changes and per-turn diffs. Review exactly what Claude changed before committing. Makes autonomous runs auditable and safe.

TIP 081

Use git bisect with Claude for bug hunting

Give Claude the write-test-then-run-bisect loop. It writes a failing test for the bug, commits it, then runs git bisect to find the exact commit that introduced the regression. Surgical precision.

TIP 082

Write tests before asking Claude to implement

Write failing tests first, commit them, then ask Claude to implement code that makes them pass. Claude can verify its own work. TDD with Claude eliminates entire categories of subtle bugs.

Mindset & Meta

Tips 83–92

TIP 083

The billion token rule

Don't think about becoming good at Claude Code in hours — think in tokens consumed. The best way to improve is to use it constantly. Every conversation teaches you how to prompt better.

TIP 084

Small investments compound

An hour spent improving CLAUDE.md, writing a skill, or building a hook pays back every session forever. The ceiling on your productivity rises each time you invest in the workflow itself.

TIP 085

Choose the right abstraction level

Vibe coding is fine for throwaway scripts. For production code, go deeper — read individual lines, question Claude's decisions, stay in control. Match scrutiny to stakes.

TIP 086

Be braver in the unknown

Iterative problem solving with Claude lets you tackle unfamiliar codebases, languages, and domains. Control the pace — go fast when exploring, slow when understanding something critical.

TIP 087

The era of personalized software is here

Build tools just for yourself. Custom dashboards, voice apps, data pipelines — Claude Code makes personal software cheap and fast. Stop waiting for a SaaS to solve your niche problem.

TIP 088

Automation of automation

Whenever you repeat a task, automate it. Put it in CLAUDE.md, make a skill, write a script. Keep pushing the abstraction higher. The goal is to automate your own automation process.

TIP 089

Share your knowledge

Teaching others solidifies your own understanding and surfaces tips you wouldn't have found otherwise. The Claude Code community compounds knowledge — what you share comes back multiplied.

TIP 090

Claude Code as universal interface

Storage cleanup, video editing via ffmpeg, data analysis, CI debugging, contract drafting — Claude Code is the first place to go for any digital problem. Treat it as a universal interface.

TIP 091

Do a retro, save institutional knowledge

At session end: 'What did you learn?' Save it. Over months, this builds a compounding knowledge base about your codebase, your patterns, your mistakes — things no onboarding doc captures.

TIP 092

Follow the community

r/ClaudeAI, r/ClaudeCode, and the growing X/LinkedIn community ship new tips weekly. The tooling moves so fast that community knowledge consistently outpaces official documentation.

Tools & Setup

Tips 93–100

TIP 093

Customize your status line

Set up a status line showing model, git branch, uncommitted file count, and a token usage progress bar. Knowing your context health at a glance is critical for long sessions.

TIP 094

Set up terminal aliases

Short aliases save thousands of keystrokes over time: c for claude, ch for claude --chrome, q to jump to your projects folder. Friction reduction compounds.

TIP 095

Use iTerm/Ghostty/tmux over IDE terminals

Dedicated terminal apps give you more control, better tab management, and are essential for the tab cascade and parallel worktree workflow. VS Code's integrated terminal is a bottleneck.

TIP 096

Use voice input tools

Wispr Flow (claimed 10x productivity), MacWhisper, or superwhisper for local transcription. Voice is faster than typing for long prompts and Claude handles transcription errors gracefully.

TIP 097

Slim down the system prompt

Patch Claude Code's CLI bundle to cut the default system prompt from ~19k to ~9k tokens at startup — ~50% token savings on every session. Also enable lazy-loading of MCP tools so they load on demand.

TIP 098

Use Gemini CLI as fallback

For sites Claude can't access directly — Reddit, paywalled content, region-restricted pages — route requests through Gemini CLI via a tmux skill. Expands Claude's reach without breaking workflow.

TIP 099

Install the dx plugin

Two commands install /dx:gha, /dx:handoff, /dx:clone, /dx:half-clone, /dx:reddit-fetch, and the review-claudemd skill all at once. The highest return-on-investment setup step in the ecosystem.

TIP 100

Use /release-notes to discover features

/release-notes shows what's new in your current Claude Code version. Claude Code ships fast — new commands, MCP tools, and capabilities appear weekly. Make reading it a daily habit.

That's 100.

Not everything will apply to your workflow right now.

Pick 3 tips that you can use tomorrow.

Then come back.

. . .

If this helped you, share it. The Claude Code community gets better when we all share what we learn.

Phaze AI · Manthan · phazeai28@gmail.com