CODING TIPS · 17 PAGES

Claude Code tips I *actually* use

24 habits, shortcuts and settings that changed how I work with Claude Code. In plain words, with the lines ready to copy. My own mods and tools are in the back.

Yair · finebutwhy.com · October 2026 · checked on Claude Code 2.1.296

INSIDE

01 Set it up once p. 3
02 Every session p. 6
03 Bigger jobs p. 9
04 Safe and cheap p. 12
05 Cheat sheet p. 14
A My own add-ons p. 15

BEFORE THE TIPS

Start here

I run Claude Code all day. These are the things I would tell a friend in their first week, checked against Anthropic's docs and on version 2.1.296.

Lines that start with `/` you type inside Claude Code. Lines that start with `claude` you type in your normal terminal. The keys are for the terminal; the desktop app and VS Code have buttons for most of the same things.

Short on time? Do these *five*

1 **Give it a way to check its work** — tip 07
A test, a build, a screenshot. Then it keeps going until the check passes, not until it looks done.

2 **Plan first on anything bigger than a typo** — tip 06
Shift+Tab into plan mode. It reads and proposes, and changes nothing until you say go.

3 **Clear between tasks** — tip 12
`/clear`. A fresh context with a good prompt beats a long one full of old detours.

4 **Keep AGENTS.md short** — tip 01
Only what Claude would get wrong without it. A long file gets half ignored.

5 **Stop early, rewind freely** — tip 11
Esc the moment it drifts. Esc twice to go back to before it went wrong.

Words I use

CONTEXT **What Claude has in mind**
Your messages, every file it read, every command output. It fills up, and quality drops as it does.

MODE **How much it may do alone**
From asking before every edit to running everything with a safety check.

SKILL **A saved way of working**
A folder with instructions. Claude loads it when needed, or you run it as `/name`.

AGENTS.MD **The house rules**
A file Claude Code and other agents read at the start of every session.

SUBAGENT **A helper with its own context**
It does the reading somewhere else and hands back only the answer.

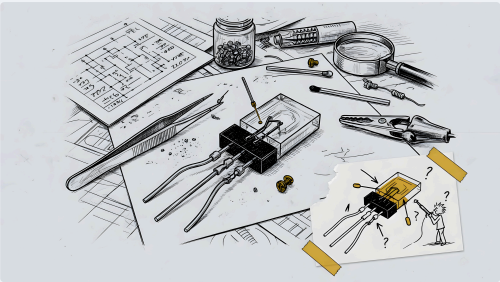
HOOK **A command that always runs**
At a fixed moment, like after every edit. Rules can be ignored; hooks can't.

01

CHAPTER ONE

Set it up *once*

Ten minutes now, and every session after starts better.



01 Write an *AGENTS.md*, then cut it down

AGENTS.md is the file Claude Code reads at the start of every session, and Codex, Cursor and Copilot read the same one. Ask Claude to write a first one from your repo. Then go through it line by line and ask: would Claude get this wrong without it? If not, delete it. A long file buries the rules that matter, and Claude starts skipping them.

A GOOD ONE IS SHORT

```
# Commands
- dev: bun run dev · test: bun test
- run one test file, not the
whole suite

# Rules
- TypeScript strict, no any
- never edit migrations/ by hand
- commit only when I ask
```

WHERE IT LIVES

- `./AGENTS.md` for the project, shared through git
- `api/AGENTS.md` extra rules, read when Claude works in that folder
- `~/.claude/CLAUDE.md` your own rules, in every project

GOOD TO KNOW

- Keep it under about 200 lines. `@docs/style.md` pulls in another file.
- Claude keeps skipping one rule? Put IMPORTANT in front of that line only.

> **WATCH OUT**
> Claude Code reads AGENTS.md only when there is no CLAUDE.md or CLAUDE.local.md in the folder or above it. `/init` writes a CLAUDE.md, so skip it here. `/memory` shows which file loaded. Needs Claude Code 2.1.277 or newer.

02 Let it remember, and check what it kept

Auto memory is on by default. While it works, Claude writes its own notes about your project: a build command it found, a fix you showed it. To add one yourself, just say it.

```
remember that we deploy with bun run
deploy, never wrangler deploy
```

`/memory` opens the notes and has the switch to turn auto memory off.

GOOD TO KNOW

- Notes live in `~/.claude/projects/`, one folder per project, with `MEMORY.md` as the index.
- The first 200 lines or 25 KB of the index load into every session.

> **WATCH OUT**
> Memory sits on your machine only. Rules your whole team needs belong in AGENTS.md.

Docs code.claude.com/docs/en/memory AGENTS.md, imports and auto memory

01 · SET IT UP ONCE

03 Pick a permission mode with *Shift+Tab*

The mode decides what Claude may do without asking you. Shift+Tab switches, and the line under the prompt says which one is on. Since version 2.1.283 new sessions start in auto mode.

MODE	RUNS WITHOUT ASKING	GOOD FOR
Manual	Reading only. It asks before every edit and command.	Sensitive work
Accept edits	Reading, file edits, and simple file commands like `mkdir` and `mv`	Code you review as it lands
Plan	Reading. It writes a plan and changes no files.	Anything bigger than a typo
Auto	Everything, while a second model checks each action and blocks what looks risky	Long tasks, fewer prompts

From auto, Shift+Tab goes to Manual, then Accept edits, then Plan, then back to Auto.

SAY YOUR LIMITS OUT LOUD

In auto mode, "don't push" or "wait for me before you deploy" makes the checker block those actions until you lift it in a later message.

WATCH OUT

There is also a mode that skips every check. Anthropic's docs say it is for isolated containers and VMs only. Leave it there.

04 Pre-approve the boring stuff, block the secrets

In Manual mode the tenth prompt is a click, not a review. `/permissions` lets you allow the commands you trust and deny what Claude should never touch. The rules count in auto mode too, and a deny always beats an allow.

.CLAUDE/SETTINGS.JSON

```
{
  "permissions": {
    "allow": [
      "Bash(npm run test *)",
      "Bash(npm run lint)",
      "Bash(git diff *)"
    ],
    "deny": [
      "Read(./.env)",
      "Read(./.env.*)",
      "Read(./secrets/**)"
    ]
  }
}
```

WHERE RULES GO

- `.claude/settings.json` for the project, shared through git
- `.claude/settings.local.json` just for you. "Yes, don't ask again" writes here.
- `~/.claude/settings.json` for every project

Try it. Run `/permissions` after a day of work and turn the prompts you clicked most into allow rules.

Docs code.claude.com/docs/en/permission-modes **Rules** code.claude.com/docs/en/permissions

01 · SET IT UP ONCE

05 Small settings that *pay off*

Each one takes a minute. Together they make every session calmer.

TYPING

New lines with Shift+Enter

Works out of the box in Ghostty, Kitty, iTerm2, WezTerm, Warp, Apple Terminal and Windows Terminal. In VS Code, Cursor and Zed, run `/terminal-setup` once. Anywhere else: type `\` then Enter, or Ctrl+J.

CONTEXT

Compact at 250k

On a 1M model Claude Code summarises at about 967k tokens. Until then every reply drags the whole session along. At 250k it stays faster, cheaper and on topic.

```
/autocompact 250k
```

ATTENTION

A ping when it needs you

Add this to `~/.claude/settings.json` on Linux. The docs have the macOS and Windows lines. Or ask Claude to write it.

```
"hooks": { "Notification": [{ "matcher": "",
  "hooks": [{ "type": "command", "command":
  "notify-send 'Claude Code' 'Claude Code needs
your attention'" }] }] }
```

ANSWERS

Shorter replies

`/output-style concise` leads with the result and leaves out the preamble and the recap. `explanatory` and `learning` do the opposite, for when you want to learn the code.

HISTORY

Keep sessions longer

Old conversations are deleted after 30 days. Want to resume or search further back? Raise it in `~/.claude/settings.json`.

```
"cleanupPeriodDays": 90
```

KEYS

Vim, or your own shortcuts

Vim editing is a switch under `/config`, Editor mode. `/keybindings` opens `~/.claude/keybindings.json` to change any shortcut.

And a status line

One row under the prompt with what you would otherwise ask for. Mine shows context, cost, branch, the usage limits and whether the prompt cache is still warm.

```
12:09:56 47m shop fix/discount [Opus 5.5 · high] 18% [183K/1M] $4.12
+126/-38 you ⟨work⟩ 5h: 34% wk: 61%
cache ● 1h 40m left · hit 94% · misses 1
```

COPY-PASTE: LET CLAUDE BUILD YOURS

```
/statusline show the folder, git branch, model and effort, a 10-block context bar with
tokens used, session cost, and small bars for the 5-hour and weekly limits. Green under
70%, yellow under 90%, red above. Refresh every 30 seconds.
```

`/statusline` writes the script and adds it to your settings. My real script, fed with demo data above.

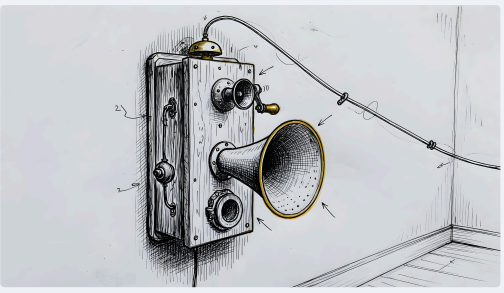
Docs code.claude.com/docs/en/terminal-config · code.claude.com/docs/en/statusline · code.claude.com/docs/en/output-styles

02

CHAPTER TWO

Every *session*

Habits for the work itself: how you ask, how you check, and how you steer.



06 Plan first, unless the change fits in one sentence

In plan mode Claude reads, asks and proposes, and changes nothing. Press Shift+Tab until the bar says plan mode, or start a single prompt with `/plan`. Press Ctrl+G to edit the plan in your own editor, then approve it and Claude starts.

Could you describe the diff in one sentence, like a typo or a log line? Skip the plan and just ask.

FOR A BIGGER FEATURE: LET IT INTERVIEW YOU

```
I want to build [short description].
Interview me in detail with the
AskUserQuestion tool. Skip the obvious
and dig into the hard parts: edge
cases, UI, tradeoffs. Keep going until
we've covered everything, then write
the spec to SPEC.md.
```

Then start a fresh session that builds from SPEC.md.

07 Give it a way to *check its own work*

Claude stops when the work looks done. Give it something that says pass or fail, like a test, a build, a linter or a screenshot to compare against, and it keeps going until the check passes. Anthropic's docs call it "the difference between a session you watch and one you walk away from".

INSTEAD OF	ASK
fix the login bug	login fails after a session timeout. Look at src/auth/. Write a failing test that shows it, then fix it.
make the dashboard look better	[screenshot] build this. Screenshot your result, compare it with mine, list the differences and fix them.
the build is failing	the build fails with [error]. Fix the cause, don't silence it, and show me the build passing.

Ask for proof, not "done". The test output, the command and what it printed, a screenshot. Reading proof is faster than checking it all again yourself.

Docs code.claude.com/docs/en/best-practices verify, explore then plan, interviews

02 · EVERY SESSION

08 Show it, don't *describe* it

The fastest way to get a good answer is to hand Claude the real thing.

A FILE

Type `@` and part of a path, and Tab completes it. Claude reads the file before it answers.

```
how does @src/auth/session.ts
refresh tokens?
```

A SCREENSHOT

Ctrl+V pastes an image from the clipboard (Cmd+V in iTerm2, Alt+V on Windows and WSL). It shows up as `[Image #1]`.

A LOG

Pipe it in from your normal terminal.

```
cat error.log | claude -p "why did
this fail"
```

A LINK

Paste the docs URL. Allow that site once in `/permissions` and it stops asking.

A CLI

"Use `foo --help` to learn foo, then..." Claude picks up tools it has never seen.

09 Run a command yourself with *!*

Start a line with `!` and it runs as a plain shell command. The output lands in the conversation, and Claude reads it and responds. Quicker than asking Claude to run it, and you see exactly what it sees.

```
! npm test
! git status
```

Ctrl+S puts a half-typed prompt aside, so you can run a command first.

10 Ask a side question with */btw*

Mid-task and you need a detail? `/btw` answers in a panel, and the question and answer never enter the conversation. Your context stays clean and Claude stays on the job.

```
/btw which pytest flag shows
print output?
```

Press `f` in the answer to hand it to a subagent with tools.

New to a codebase? Ask it like a *colleague*

No special prompting needed. Ask what you would ask the engineer who has been there for years, and point at examples instead of explaining them.

- How does logging work here?
- How do I add a new API endpoint?
- Why does this call `foo()` instead of `bar()` on line 333?
- Look through the git history of this file and tell me how its API came to be.

```
look at how the widgets on the home
page are built. HotDogWidget.php is a
good example. Follow that pattern for a
calendar widget.
```

Docs code.claude.com/docs/en/interactive-mode every key and prefix

02 · EVERY SESSION

11 Stop early, *rewind* freely

Esc stops Claude mid-step. Nothing is lost, so just say what you want instead. Correcting early beats correcting late.

Esc twice on an empty prompt, or `/rewind`, opens the rewind menu. Every prompt you sent is a checkpoint. Go back to one and restore the code, the conversation, or both. That even works after you close the terminal and resume.

Use it to be brave. Let Claude try the risky idea. If it fails, rewind.

> **WATCH OUT**
>
> Checkpoints only cover Claude's own file edits. What a shell command does, like `rm`, `mv` or a migration script, can't be rewound.
>
> Commit before anything big. Checkpoints are not git.

Want Claude to undo its last change? "Undo that" works too.

12 Keep the context *clean*

Everything Claude read this session sits in its context, and the fuller it gets, the more it forgets and slips. Treat it like a desk.

- `/clear` between unrelated tasks. The most underused command there is.
- `/compact keep the plan and the open todos` summarises now, and you say what must survive.
- `/context` shows what is taking up the space.
- Esc twice, pick a message, then "Summarize from here" squeezes only the part after it.
- Tell compaction what to keep, once, in AGENTS.md: "When compacting, always keep the list of changed files and the test commands."

> **THE TWO-STRIKE RULE**
>
> Corrected it twice on the same thing? The context is full of failed attempts. `/clear` and start over with a better first prompt that includes what you learned.

13 Name your sessions, pick them up later

```
claude -c              # the last one here
claude -r              # pick from a list
claude -n oauth-fix    # start with a name
claude --from-pr 123   # sessions of PR 123
```

Inside a session: `/rename` gives it a name, and `/resume` switches to another one. Treat named sessions like branches: one per piece of work, each with its own context.

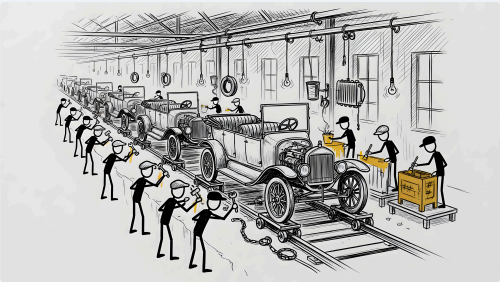
Docs code.claude.com/docs/en/checkpointing · code.claude.com/docs/en/sessions

03

CHAPTER THREE

Bigger *jobs*

When one chat is not enough: helpers, saved workflows, rules that always run, and sessions side by side.



14 Let *subagents* do the reading

A subagent works in its own context and only its answer comes back to yours. Use one for anything that reads a lot: a big folder, a long log, a second look at a diff. Just ask.

```
use subagents to find out how we
refresh tokens, and whether we already
have OAuth helpers I should reuse
```

Built in: Explore, Plan and general-purpose. `/agents` no longer opens a manager; ask Claude to write one, or add a file yourself.

YOUR OWN: .CLAUDE/AGENTS/CODE-REVIEWER.MD

```
---
name: code-reviewer
description: Reviews a diff for bugs.
  Use after writing code.
tools: Read, Grep, Glob
model: sonnet
---
Review only the changed lines. Report
bugs and missing tests, not style.
Name the file and line for each.
```

Then: "use the code-reviewer subagent on my last commit".

15 Turn a prompt you repeat into a *skill*

A skill is a folder with a SKILL.md. Claude loads it when the description fits the task, or you run it as `/name`. Whatever you type after the name lands in `$ARGUMENTS`.

AGENTS.md loads every session; a skill only when it is needed. Move the instructions you only need now and then out of AGENTS.md and into a skill.

`disable-model-invocation: true` means only you can start it. Use that for anything with side effects. Old `.claude/commands/` files still work.

.CLAUDE/SKILLS/FIX-ISSUE/SKILL.MD

```
---
name: fix-issue
description: Fix a GitHub issue
disable-model-invocation: true
---
Fix GitHub issue $ARGUMENTS:
1. read it with gh issue view
2. write a failing test, then fix it
3. run the tests and the linter
4. commit and open a PR
```

Run it with `/fix-issue 1234`.

Docs code.claude.com/docs/en/sub-agents · code.claude.com/docs/en/skills

03 · BIGGER JOBS

16 Use a *hook* for what must happen every time

AGENTS.md is advice, and advice gets skipped. A hook is a shell command Claude Code runs at a fixed moment, every time. This one formats every file Claude edits.

.CLAUDE/SETTINGS.JSON

```
{
  "hooks": {
    "PostToolUse": [{
      "matcher": "Edit|Write",
      "hooks": [{ "type": "command",
        "command": "jq -r '.tool_input.file_path' | xargs npx prettier --write" }]
    }]
  }
}
```

Needs `jq` and Prettier. `/hooks` shows what is set up.

THE MOMENTS

PreToolUse before a tool runs, and it can block it · **PostToolUse** after it ran · **Stop** when Claude thinks it is done · **Notification** when it needs you · **SessionStart** when a session opens

Easier: ask for it. "Write a hook that runs eslint after every edit", or "write a hook that blocks writes to the migrations folder".

17 Connect your tools: a *CLI* first, MCP when there is none

Anthropic's docs call CLI tools the most context-efficient way to reach outside services. Install `gh` and Claude handles issues and pull requests with it. Same for `aws`, `gcloud` or `az`.

No CLI? Connect an MCP server. `/mcp` shows which ones are connected and signs you in.

```
claude mcp add --transport http notion
https://mcp.notion.com/mcp
```

Add `--scope project` to share it with your team through `.mcp.json`, or `--scope user` for all your projects.

Which one do I *need*?

YOU WANT	USE	WHERE
Claude to always know a rule	**AGENTS.md**	`./AGENTS.md`
A way of working you need now and then	**Skill**	`.claude/skills/`
Something to happen every single time	**Hook**	`.claude/settings.json`
Heavy reading without filling your context	**Subagent**	`.claude/agents/`
Claude to reach an outside service	**CLI or MCP**	`claude mcp add`
To share all of it with others	**Plugin**	`/plugin`

Docs code.claude.com/docs/en/hooks-guide · code.claude.com/docs/en/mcp

03 · BIGGER JOBS

18 Run sessions *side by side*

EACH IN ITS OWN COPY OF THE REPO

```
claude -w feature-auth
```

Starts a session in its own git worktree, at `.claude/worktrees/feature-auth` on a new branch. Two sessions never edit the same files. Add `.claude/worktrees/` to `.gitignore`.

WRITER AND REVIEWER

Let one session write and a fresh one review. The reviewer isn't attached to code it just wrote.

IN THE BACKGROUND

```
claude --bg "find out why the
  checkout test is flaky"
claude agents      # all of them
claude attach <name>
```

`/bg`, or `←` on an empty prompt, sends the session you are in to the background. Ctrl+B does the same for one long command.

19 Script it with *claude -p*

```
claude -p "list every API endpoint" --output-format json
claude -p "fix all lint errors" --permission-mode auto
claude -p "write the changelog" --max-budget-usd 2
```

`-p` runs one prompt with no chat and prints the answer, so it fits in scripts, CI and git hooks. `--max-budget-usd` caps what a run may spend.

In CI, `--bare` skips hooks, plugins, MCP servers and auto memory so it starts faster. It also skips your subscription login, so it needs an API key.

FAN OUT OVER MANY FILES

```
for file in $(cat files.txt); do
  claude -p "Migrate $file to the new API. Return OK or FAIL." \
    --allowedTools "Edit,Bash(git commit *)" \
    --permission-mode dontAsk
done
```

Have Claude write the list of files first. Run the loop on two or three, fix the prompt, then run the rest. `dontAsk` refuses anything you didn't allow, so an unattended run never hangs on a prompt.

Inside a session, `/batch` does this for you: it splits a change over 5 to 30 subagents, each in its own worktree.

REPEAT IT WHILE YOU WORK

```
/loop 5m check the deploy and tell me
when it is green
```

`/loop` repeats a prompt while the session runs. Without an interval, Claude picks one. `/schedule` runs it in the cloud, even with your laptop closed.

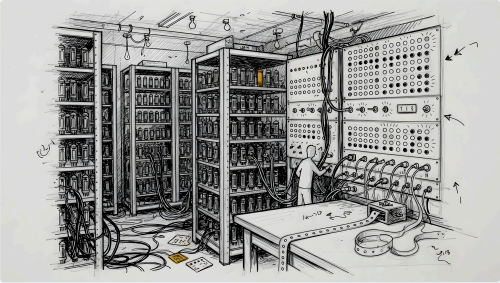
Docs code.claude.com/docs/en/worktrees · code.claude.com/docs/en/agent-view · code.claude.com/docs/en/headless

04

CHAPTER FOUR

Safe and *cheap*

Spend your limits where they matter, keep your keys out of the chat, and check before you push.



20 Match the model and the *effort* to the job

- `/model` or Alt+P switches the model. Option+P on a Mac.
- `/effort low` up to `max` sets how hard it thinks. Low for renames and lookups, high for a nasty bug.
- Put `ultrathink` in a prompt for one deep turn, without changing your setting.
- Subagents that only read can run on a cheaper model: `model: haiku` in their file.

WATCH OUT

`/fast` gives faster output and costs extra. On a subscription it is billed to usage credits.

21 Know what a session costs

`/usage` shows the session's cost, how much of your plan limits you have used, and your activity. `/cost` and `/stats` open the same screen.

On a subscription the dollar figure is not what you pay. The limit bars are what count.

WHAT SAVES THE MOST, IN MY EXPERIENCE

- `/clear` between tasks
- Compacting at 250k instead of near a million
- Subagents for the heavy reading
- Lower effort for the easy work

Before you walk away from a *long run*

Five checks that let you leave Claude alone for an hour and come back to something you can trust.

1 Commit first
So you can always get back, whatever a command did.
tip 11

2 Give it a check it can run
Tests or a build that must pass before it may say done.
tip 07

3 Say your limits
"Don't push, don't deploy, don't touch the database."
tip 03

4 Lock the secrets
Deny rules for `.env` and your keys folder.
tip 04

5 Ask for proof at the end
The test output, the commands it ran, a screenshot.
tip 07

Docs code.claude.com/docs/en/model-config · code.claude.com/docs/en/costs · code.claude.com/docs/en/fast-mode

04 · SAFE AND CHEAP

22 Keep your *keys* out of the chat

A key you paste into the chat stays in the transcript on disk. Put it in an environment variable or your keychain, and tell Claude to use `$OPENAI_API_KEY` instead of the value.

Block reads of `.env` with a deny rule (tip 04), so it can't read the file either.

FEWER PROMPTS, STILL SAFE

`/sandbox` puts the shell commands Claude runs behind a wall your OS enforces: they can only write to your project folder and a temp folder, and every connection is checked against the sites you allow. Choose auto-allow and sandboxed commands run without asking. Reading is still open by default, so keep your deny rules. macOS, Linux and WSL2.

BEFORE YOU INSTALL ANYONE'S PLUGIN

```
claude plugin validate ./the-plugin
```

A plugin runs with your permissions. Validate checks it without running it. For mods it also lists every event they listen to and every call they make. Read that before you install.

23 Get a second pair of eyes *before you push*

- `/code-review` reviews your diff for bugs in a fresh context. Add `--fix` and it applies what it found.
- `/security-review` checks your branch for security problems. It needs an `origin` remote.
- `/code-review ultra` runs a review with several agents in the cloud. Pro and Max get 3 free runs, then it is billed.

OR CHECK IT AGAINST YOUR PLAN

```
use a subagent to review this diff
against PLAN.md. Check that every
requirement is done and the edge cases
have tests. Report gaps that affect
correctness, not style.
```

24 When something feels *off*

- `/doctor` checks your setup. `/doctor prompt-audit` finds outdated or conflicting lines in AGENTS.md, skills and subagents.
- `claude --safe-mode` starts with AGENTS.md, skills, plugins, hooks and MCP servers off. If the problem is gone, one of those is the cause.
- `/export` saves the conversation to a file, handy for a bug report.
- `claude update` gets the newest version. Check yours with `claude --version`.

Docs code.claude.com/docs/en/code-review · code.claude.com/docs/en/cli-reference

05 · CHEAT SHEET

One page to *keep open*

Everything from the tips, on one page. `?` on an empty prompt shows all the keys.

Keys

Shift+Tab	switch mode
Esc	stop Claude
Esc Esc	rewind menu
Ctrl+G	edit the plan
Ctrl+V	paste an image
Ctrl+J	new line
Ctrl+R	search history
Ctrl+S	put a draft aside
Ctrl+O	full transcript
Ctrl+T	Claude's task list
Ctrl+B	command to background
Alt+P	switch model
←	session to background
@	add a file
!	run a shell command
?	all shortcuts

Inside Claude Code

/config	all settings
/memory	rules and notes
/permissions	allow and deny
/plan	plan this prompt
/clear	start fresh
/compact	summarize now
/context	what fills it
/rewind	back to a checkpoint
/btw	side question
/rename	name this session
/model	pick the model
/effort	how hard it thinks
/usage	cost and limits
/autocompact	compact window
/code-review	review the diff
/loop	repeat a prompt
/hooks	show hooks
/mcp	connected tools
/doctor	check the setup

In your terminal

claude -c	continue
claude -r	pick a session
claude -n name	named session
claude -w name	own worktree
claude --bg "…"	background
claude agents	see them all
claude -p "…"	one-shot
--permission-mode	start in a mode
--effort high	start with effort
--add-dir ../api	a second folder
--from-pr 123	sessions of a PR
--safe-mode	no customizations
claude mcp add	connect a tool
claude plugin validate	check a plugin
claude doctor	diagnostics
claude update	newest version

Where things *live*

`./AGENTS.md`	project rules, every agent
`api/AGENTS.md`	rules for one folder
`~/.claude/CLAUDE.md`	your rules in every project
`.claude/settings.json`	project settings, shared
`.claude/settings.local.json`	your own project settings
`~/.claude/settings.json`	your settings in every project
`.mcp.json`	MCP servers, shared
`.claude/skills/`	skills
`.claude/agents/`	subagents
`.claude/worktrees/`	sessions started with claude -w
`~/.claude/projects/`	auto memory notes
`~/.claude/keybindings.json`	your shortcuts

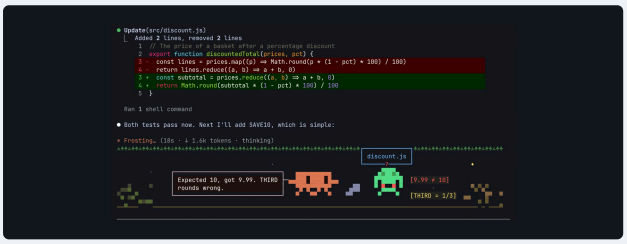
In the project folder, the files without `local` are meant for git, so your team gets them too. Files under `~/.claude` are yours alone.

APPENDIX · MADE BY ME

My own *add-ons*

The tips work without any of this. These are the mods and tools that run on my own machine, in case you want to try them.

> **BEFORE YOU INSTALL ANYTHING, MINE INCLUDED**
>
> A mod is code that runs inside Claude Code, and it is not sandboxed. It can read your files, see your prompts and spend your usage. Clone it and run `claude plugin validate` on it first (tip 22). Mods need Claude Code 2.1.287 or newer, and draw in the terminal, not in the VS Code chat panel.



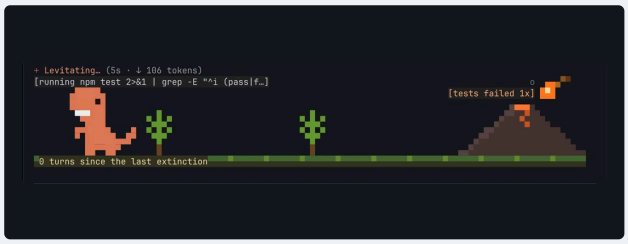
MOD · USES YOUR PLAN

Clawd Cartoons

Clawd acts out every step in a live cartoon under the spinner, with a speech bubble naming the real file and result.

About 4 small model calls per 30 second turn. Pick haiku in `/plugin` to pay less.

```
/plugin marketplace add falkoro/clawd-cartoons
/plugin install clawd-cartoons@clawd-cartoons
```



MOD · FREE

Clawd Park

A pixel T-rex under the spinner. Every failing test run brings the meteor closer; three in a row and Rex is a fossil.

No model calls. Nothing leaves your machine.

```
/plugin marketplace add falkoro/clawd-park
/plugin install clawd-park@clawd-park
```

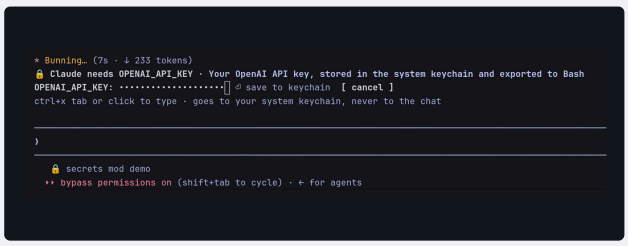
MOD · FREE OFFLINE

Scenes

A tiny text-art scene of the work in place of the spinner, and the line under it names the real command.

Live mode lets Haiku draw new scenes, about a third of a cent each.

```
/plugin marketplace add falkoro/claude-scenes
/plugin install scenes@claude-scenes
```



MOD · FREE

Secrets

Claude asks for an API key in a masked field. It goes to your OS keychain, and Claude uses it as `$NAME` without seeing the value.

On Linux you need `secret-tool` from libsecret.

```
/plugin marketplace add falkoro/claude-
code-secrets
/plugin install secrets@falkoro
```

APPENDIX · TOOLS I USE

And the tools around it

One plugin and one app of mine, and two tools by Kun Chen that I use every day.

PLUGIN · MINE

Azure

Pipelines, approvals, pull requests and work items in Azure DevOps, through the `az` CLI and the sign-in you already have. Reading never asks; anything that changes something does.

```
az extension add --name azure-devops
claude plugin marketplace add falkoro/claude-code-azure
claude plugin install azure@claude-code-azure
```

github.com/falkoro/claude-code-azure

APP · MINE

Agent Office

Your agents as pixel characters in an office in the browser: at a desk while they work, on the couch while they wait, and a bell when one needs approval. Works with Claude, Codex, Grok, OpenCode and more.

github.com/falkoro/agent-office · WSL and Windows

TOOLS · BY KUN CHEN

axi tools

GitHub and Chrome CLIs built for agents: short output, a hint for the next command. A SessionStart hook briefs Claude on the repo, its issues and PRs when a session opens.

```
npm install -g gh-axi chrome-devtools-axi
gh-axi setup hooks
chrome-devtools-axi setup hooks
```

github.com/kunchenguid/gh-axi · axi.md

TOOL · BY KUN CHEN

no-mistakes

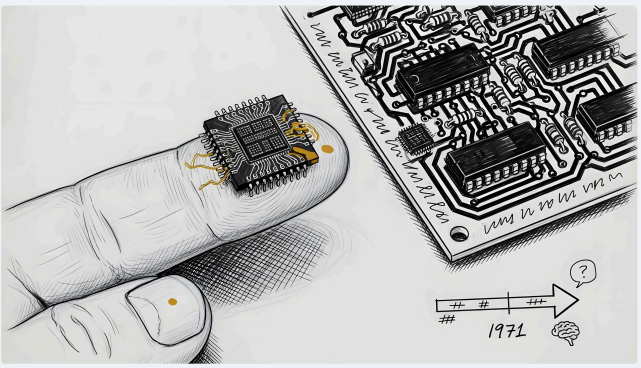
Push to `no-mistakes` instead of `origin`. Review, tests and lint run in a throwaway worktree, and your branch only goes out when all are green. Each gate runs an agent, so it uses some of your plan.

```
curl -fsSL
https://raw.githubusercontent.com/kunchenguid/no-mistakes/main/docs/install.sh | sh
no-mistakes init
```

github.com/kunchenguid/no-mistakes

Make your own mod

Claude Code ships with a skill for writing plugins. That is how I built all four of mine. Open Claude Code in an empty folder, fill in the brackets, paste.



COPY-PASTE: BUILD A MOD

```
build me a Claude Code mod: [what you want to see or change, for example: count failing test runs in a row and show the streak next to the spinner, red from 3]
- read https://code.claude.com/docs/en/plugins/mods/overview first and use the plugin-authoring skill
- no network calls and no model calls unless I ask for them
- write tests and run claude plugin validate --strict on it
- tell me how to try it for one session with --plugin-dir, and how to turn it off
```

YOUR SETUP

Which tips fit *yours*?

Give this PDF to Claude Code with the prompt below. It looks at how you have things set up now, and changes nothing until you say so.

COPY-PASTE: SEND WITH THIS PDF

```
I've attached a PDF with 24 Claude Code tips. Read it, then check my setup against it.
- look for yourself: my Claude Code version, AGENTS.md and CLAUDE.md files and how long
they are, my permission rules, hooks, status line, autocompact setting and MCP servers
- ask me the rest: what I mostly use you for, whether I work in a team, and what annoys
me most
- give me the 5 tips that would help me most, best first, with one line on why
- for each one, show exactly what you would change
- change nothing yet. Wait for my go on each one
```

Where it comes from

Best practices	code.claude.com/docs/en/best-practices
AGENTS.md	code.claude.com/docs/en/memory
Modes	code.claude.com/docs/en/permission-modes
Permissions	code.claude.com/docs/en/permissions
Hooks	code.claude.com/docs/en/hooks-guide
Subagents	code.claude.com/docs/en/sub-agents
Skills	code.claude.com/docs/en/skills
Background	code.claude.com/docs/en/agent-view
Scripts	code.claude.com/docs/en/headless
Costs	code.claude.com/docs/en/costs
My repos	github.com/falkoro
Compact video	finebutwhy.com/p/auto-compact-at-250k

FINE BUT WHY

More at *finebutwhy.com*

Essays, coding tips with a short video each, and the odd benchmark. Get new posts by email at **finebutwhy.com/newsletter**, or follow **@finebutwhy** on X.

Found a tip that is out of date? Claude Code moves fast. Tell me on X and I will fix the guide.