

MINI PACK

Make Your Claude Code Limit Last

Get more done before "You've hit your session limit" shows up.

For Claude Code · on Pro, Max, Team, and Enterprise plans

10 prompts · 4 sections

Welcome! How to Use This Pack

You sit down to build, and an hour later Claude Code says you've hit your limit. Usually it isn't one big request that did it. It's a chat that stayed open all day, a pile of files Claude didn't need, or a heavy model doing a light job.

Here's the main idea behind this pack: Claude Code sends your whole conversation with every request. A one-line question at the end of a long session still draws usage for all the history in front of it. So the biggest savings come from keeping each session small and focused.

This pack shows you where your usage goes, how to start sessions lean, how to keep them small while you work, and what to do when you're close to the limit.

Claude Code changes often. Every tool-specific tip here comes from Claude Code's official docs as of October 2026. If a command or screen looks different for you, run `claude update` or check the current docs.

How each prompt is laid out

- **Title:** what the prompt is.
- **Use this when...:** one or two sentences on what the prompt is for and when to reach for it.
- **Fill in:** the blanks to replace before you paste.
- **The prompt:** the box you copy.
- **Tips to customize:** optional ways to change it.

Anything in **[BRACKETS IN CAPS]** is a blank to fill in.

Works best with: Claude Code, in the terminal, an editor extension, or the desktop app. Prompts 1.1 and 1.2 only need what you paste, so you can also run them in any AI chat.

The commands, in one line each. `/usage` shows your plan limits and when they reset. `/context` shows what's filling your context right now. `/clear` starts a fresh conversation. `/compact` shrinks the current one into a summary. `/model` and `/effort` pick the model and how hard it thinks.

Using the free Claude Code prompts on our site too? Those help you write a CLAUDE.md and plan a feature. This pack is about stretching your usage so you can keep building.

Quick Start: Find Your Problem

- **No idea what's eating your usage?** Prompt 1.1.
- **One session burned way more than it should have?** Prompt 1.2.
- **Every session starts slow and heavy?** Prompts 2.1 and 2.2.
- **Using the strongest model for everything?** Prompt 2.3.
- **One long chat that never ends?** Prompt 3.1.
- **Test runs and logs flooding the chat?** Prompt 3.2.
- **Lots of add-ons and MCP servers connected?** Prompt 3.3.
- **A big feature that never fits in one sitting?** Prompt 4.1.
- **Close to the limit right now?** Prompt 4.2.

Golden rule: One task per session. When the task changes, run `/clear`.

What's Inside

1. Find the Leak (2 prompts)
2. Start Lean (3 prompts)
3. Stay Lean While You Work (3 prompts)
4. Work With the Limit, Not Against It (2 prompts)

SECTION 1

Find the Leak

Before you change how you work, see where your usage actually goes.

1.1 Read My Usage Report

Use this when you keep hitting your limit and don't know why. It's for turning the numbers from `/usage` and `/context` into a plain explanation and three changes to try first.

Fill in: [MY /USAGE BREAKDOWN] (paste the breakdown part of `/usage`), [MY /CONTEXT OUTPUT] (paste the output of `/context` from a typical session), [HOW I USE CLAUDE CODE] (e.g. "one long session all day on a Next.js app, Opus as my model").

I keep hitting my Claude Code usage limit. Help me understand why. Don't suggest anything that needs a paid add-on.

My `/usage` breakdown: [MY /USAGE BREAKDOWN]

My `/context` output from a typical session: [MY /CONTEXT OUTPUT]

How I use Claude Code: [HOW I USE CLAUDE CODE]

Please:

1. Explain in plain words what each part of these numbers means. If something isn't clear from what I pasted, say so instead of guessing.
2. Tell me which three things are most likely using up my limit, biggest first.
3. For each one, give me one specific change to try this week, like a command to run or a habit to change.
4. Tell me what to check in `/usage` next week to see if it worked.

 **Tips to customize:** On a Pro, Max, Team, or Enterprise plan, `/usage` shows your usage bars plus a breakdown of where usage went (skills, subagents, plugins, and MCP servers) and flags habits like long context or cache misses when one makes up 10% or more of your recent usage. Press `d` or `w` to switch between the last 24 hours and the last 7 days. The breakdown only counts sessions on this computer. Paste just the breakdown and the bars. The dollar "Total cost" at the top is meant for API users, so you can ignore it on a subscription.

1.2 Why Did This Session Burn So Much?

Use this when one session used far more of your limit than the work you did in it. It's for checking your session against the usual reasons usage climbs, so you know which habit to change.

Fill in: [WHAT I DID IN THE SESSION] (e.g. "fixed a login bug, then built a settings page, then asked about deploys"), [HOW LONG IT WAS OPEN] (e.g. "about 6 hours, with a lunch break"), [ANYTHING RUNNING IN

THE BACKGROUND] (e.g. "a /loop checking my tests every 10 minutes", "subagents", or "nothing I know of").

One Claude Code session used much more of my usage limit than I expected. Help me find out why.

What I did in the session: [WHAT I DID IN THE SESSION]

How long it was open: [HOW LONG IT WAS OPEN]

Anything running in the background: [ANYTHING RUNNING IN THE BACKGROUND]

Here's what I know: Claude Code sends the whole conversation with every request, and when I come back after a break the saved copy of the conversation (the prompt cache) may have expired, so the next message reprocesses everything.

Please:

1. Check my session against these common causes: one long conversation used for unrelated tasks, coming back after breaks, repeating tasks like /loop that run while I'm away, subagents, and compacting a very large conversation.
2. For each cause, say "likely", "possible", or "unlikely" for my session, with one line explaining why.
3. Give me a short checklist of habits for my next session, in order of how much they'll save.

 **Tips to customize:** Claude Code's docs list these as the main reasons a long session uses more than you'd expect. A repeating task from `/loop` sends your full context each time it runs, even while you're away, so stop the ones you don't need. Compacting also costs something, because Claude has to read the whole conversation to summarize it. If you want a fresh start rather than a summary, `/clear` costs nothing.

SECTION 2

Start Lean

What loads at the start of a session loads into every request after it.

2.1 The Lean Session Kickoff

Use this when you start a new Claude Code session for a task. It's for giving Claude exactly the files and goal it needs, so it doesn't read half your project to figure out what you meant.

Fill in: [THE TASK] (one task, e.g. "add an email field to the signup form"), [FILES INVOLVED] (paths, e.g. `src/app/signup/page.tsx`, or "not sure"), [DONE MEANS] (e.g. "the form saves the email and the existing tests pass"), [DON'T TOUCH] (e.g. "the database migrations folder").

```
Task for this session: [THE TASK]
```

```
Files involved: [FILES INVOLVED]
```

```
Done means: [DONE MEANS]
```

```
Don't touch: [DON'T TOUCH]
```

```
Ground rules for this session:
```

1. Start with the files I listed. Only open other files if you need them, and tell me which ones and why before you do.
2. If I wrote "not sure" for the files, do one quick, targeted search to find them, show me the list, and wait for my OK.
3. Don't paste whole files back to me. Show only the lines you changed.
4. Stop when "done" is met and tell me how you checked it. Don't start on related improvements.

 **Tips to customize:** Type `@` in Claude Code to autocomplete a file path instead of typing it out. Claude Code's docs give this example: a vague request like "improve this codebase" makes Claude scan broadly, while "add input validation to the login function in auth.ts" lets it work with only a few file reads. Don't paste big files or logs into the chat yourself. Give the path and let Claude read only what it needs.

2.2 Move Big Workflows Out of CLAUDE.md

Use this when your CLAUDE.md holds long, step-by-step instructions for jobs you only do sometimes, like deploys or database changes. It's for moving those into skills, which load only when you use them, so every session starts lighter.

Fill in: [MY CLAUDE.MD PATH] (usually `CLAUDE.md`), [JOBS I DO SOMETIMES] (e.g. "deploying", "writing database migrations", "reviewing pull requests").

```
Don't change any files yet. Read [MY CLAUDE.MD PATH] and help me make it lighter.
```

```
Jobs I only do sometimes: [JOBS I DO SOMETIMES]
```

```
Please:
```

1. Find every section that only matters for one of those jobs, or for any other occasional job you notice.
2. For each one, propose a skill: a short name, a one-line description of when to use it, and the steps it should contain.
3. Show me the slimmer CLAUDE.md with those sections removed, keeping only what matters in every session.
4. Add a short "Compact instructions" section that tells you what to keep when the conversation is summarized, based on how this project works.
5. List the new files you'd create in `.claude/skills/` and wait for my OK.

 **Tips to customize:** CLAUDE.md loads at the start of every session, but a skill's full instructions only load when it's used, and just its short description loads up front. Claude Code's docs suggest keeping CLAUDE.md under 200 lines. To see which skills take up the most room, run `/skills` and press `t` to sort by token count. If you also use other AI tools, our One Rules File pack covers trimming and syncing rules files across them.

2.3 Right Model, Right Effort

Use this when you use the same top model and settings for everything. It's for matching each kind of task to a model and effort level, so heavy reasoning goes only where it pays off.

Fill in: [MY PLAN] (e.g. "Pro", "Max"), [MY USUAL TASKS] (e.g. "renaming things, small UI tweaks, fixing bugs in old code, planning new features"), [WHAT I USE NOW] (e.g. "Opus on high for everything", or "not sure").

```
Help me pick a Claude Code model and effort level for each kind of task I do, to make my usage limit last longer without getting worse results.
```

```
My plan: [MY PLAN]
```

```
My usual tasks: [MY USUAL TASKS]
```

```
What I use now: [WHAT I USE NOW]
```

```
Please:
```

1. Make a table with each task, a suggested model (sonnet, opus, or haiku), a suggested effort level, and a one-line reason.
2. Point out which tasks need the strongest settings and which ones are wasting them right now.
3. Tell me the exact commands to switch: `/model` for the model and `/effort` for the effort level.
4. If you aren't sure which models or effort levels my version offers, tell me to check `/model` instead of guessing.

 **Tips to customize:** Claude Code's docs say Sonnet handles most coding tasks well and costs less than Opus, so save Opus for complex design decisions or hard multi-step problems. Lower effort is faster and cheaper for simple tasks like a rename, while higher effort suits work like fixing a bug in existing code. There's also an `opusplan` setting that uses Opus while you plan in plan mode and switches to Sonnet to write the code. Available models and effort levels change over time, so check `/model` for what you have.

SECTION 3

Stay Lean While You Work

Every message carries everything before it. Keep the pile small.

3.1 Clear or Compact? Decide and Get Ready

Use this when a session has gotten long, or you're about to switch to something different. It's for deciding whether to start fresh with `/clear` or shrink the session with `/compact`, and for saving what matters first either way.

Fill in: [WHAT I'M DOING NEXT] (e.g. "keep working on the same checkout bug", or "start the profile page"), [HANDOFF FILE] (e.g. `notes/handoff.md`).

```
I'm about to free up context. What I'm doing next: [WHAT I'M DOING NEXT]
```

Please:

1. Tell me whether I should use `/clear` (fresh start) or `/compact` (keep a summary), and why, in one or two sentences.
2. If `/clear`: write a short handoff note to [HANDOFF FILE] with what we finished, decisions we made and why, files changed, what's left, and anything that turned out to be a dead end. Keep it under 200 words.
3. If `/compact`: write me the exact `/compact` line to run, with focus instructions that keep what I need for the next step and drop the rest.
4. Then tell me the first message to send after I run it.

 **Tips to customize:** Claude Code's docs recommend `/clear` between unrelated tasks, and starting fresh with a more specific prompt after you've corrected Claude more than twice on the same thing. Run `/rename` first if you might want this session back, then find it later with `/resume`. A focused `/compact`, like `/compact focus on the checkout bug fix`, keeps what you choose instead of what the automatic summary guesses is important. Your project-root `CLAUDE.md` is reloaded after compacting, so you don't need to repeat its rules.

3.2 Send the Noisy Work to a Subagent

Use this when you're about to run tests, read long logs, or search through docs, and you only care about the result. It's for having a subagent do the noisy part in its own space, so only a short summary lands in your main conversation.

Fill in: [THE NOISY JOB] (e.g. "run the full test suite", "read the last 500 lines of the server log"), [WHAT I WANT BACK] (e.g. "only the failing tests and their error messages").

```
Use a subagent for this, and keep the details out of our main conversation: [THE NOISY JOB]
```

```
Report back only: [WHAT I WANT BACK]
```

Rules:

1. Keep the report short: no full logs, no whole files, no passing results.
2. If the job turns up something unexpected, say so in one line and wait for me before digging further.
3. Don't change any files.

 **Tips to customize:** Claude Code's docs suggest exactly this for test runs, docs lookups, and log files, because the long output stays in the subagent's context and only the summary comes back. The subagent still uses your plan's usage, so save this for big outputs, not quick checks. If you do this often, ask Claude to create a project subagent for it in `.claude/agents/` with `model: haiku`, which is a smaller, cheaper model that suits simple jobs like this.

3.3 Turn Off Tools I'm Not Using

Use this when you've connected several MCP servers or plugins and `/context` shows a lot loading before you type anything. It's for finding what you actually use and turning off the rest.

Fill in: [MY /CONTEXT OUTPUT] (from the start of a fresh session), [MY MCP SERVERS] (the list from `/mcp`), [WHAT I'M WORKING ON THIS WEEK] (e.g. "front-end work on the dashboard").

```
Help me trim what loads into every Claude Code session.
```

```
My /context output from a fresh session: [MY /CONTEXT OUTPUT]
```

```
My MCP servers: [MY MCP SERVERS]
```

```
What I'm working on this week: [WHAT I'M WORKING ON THIS WEEK]
```

Please:

1. List each MCP server and say whether I'm likely to need it this week: keep, turn off for now, or not sure.
2. For any server that does something a command-line tool already does (for example GitHub and the `gh` tool), tell me whether the command-line tool would work instead.

3. Point out anything else in my `/context` output that looks bigger than it should be, like a long `CLAUDE.md` or many skills.
4. Give me the exact steps to turn off the servers I don't need, using `/mcp`.

💡 **Tips to customize:** Claude Code loads only the names of MCP tools (plus short server instructions) until it actually uses one. Its docs still recommend turning off servers you aren't using with `/mcp` and preferring command-line tools like `gh` when they're available, since those add no tool list at all. You can turn a server back on any time.

SECTION 4

Work With the Limit, Not Against It

Plan around the limit so it never stops you halfway through.

4.1 Split a Big Job Into Small Sessions

Use this when you're starting a feature that's too big to finish in one sitting. It's for breaking it into small sessions, each with one clear goal and a short starting note, so you never carry one huge conversation for days.

Fill in: [THE FEATURE] (e.g. "user accounts with profile pages and avatar upload"), [WHAT ALREADY EXISTS] (e.g. "login works, no profile page yet"), [PLAN FILE] (e.g. `notes/profile-plan.md`).

Don't write any code yet. Break this feature into small Claude Code sessions.

Feature: [THE FEATURE]

What already exists: [WHAT ALREADY EXISTS]

Please:

1. Split the work into sessions. Each one should have one goal, end in something I can test, and touch as few files as possible.
2. For each session, write: the goal, the files involved, how I'll know it's done, and a two-line starting message I can paste after `/clear`.
3. Put the riskiest session (data changes, new libraries, payments) early, and mark it so I review it closely.
4. Save the plan to [PLAN FILE] so each new session can read it instead of me re-explaining.

 **Tips to customize:** Run this in plan mode (press Shift+Tab to cycle modes), so Claude explores and proposes without editing anything. Claude Code's docs say planning first prevents expensive rework when the first direction is wrong. Start each new session with `/clear` and the starting message, then name the plan file by path so Claude reads only what it needs. Commit after every session so you can always roll back.

4.2 Wrap Up Before the Limit Hits

Use this when Claude Code warns that you've used most of your session limit, or `/usage` shows you're close. It's for finishing cleanly, so you don't get cut off halfway through a change and come back to a broken app.

Fill in: [WHAT WE'RE IN THE MIDDLE OF], [HANDOFF FILE] (e.g. `notes/handoff.md`).

```
I'm close to my usage limit. Help me stop at a safe point.
```

```
What we're in the middle of: [WHAT WE'RE IN THE MIDDLE OF]
```

```
Please:
```

1. Finish only the smallest step that gets the app back to a working state. If that's more than a few small edits, undo the unfinished change instead and tell me what you undid.
2. Run the quickest check that proves it works, like the build or the related tests, and show me the result.
3. Write a short handoff note to [HANDOFF FILE]: what's done, what's half-done, the exact next step, and anything I shouldn't forget.
4. Suggest a commit message for what's finished. Don't start anything new.

 **Tips to customize:** You can see the warning coming. To watch your limits all the time, run `/statusline` and ask for a status line that shows your context use and your 5-hour and 7-day limits. If you do hit the limit, the message tells you when it resets. In newer versions, if you're signed in with a Claude subscription, Claude Code can wait in the open session and pick the task back up after the reset. If the message names one model family, like "You've hit your Opus limit," you can switch to a different family with `/model` and keep working. The general session and weekly limits cover all models, so switching won't help there.

Bonus: Magic Phrases for Saving Usage

- "Only open the files I listed."
- "Show me only the lines you changed."
- "Use a subagent and report back only the failures."
- "Stop when it works. Don't add extras."
- "Write a handoff note before I clear."
- "Ask before reading anything else."

Make Your Claude Code Limit Last

That's the whole pack: **10 prompts across 4 sections**. Find what's eating your usage, start every session lean, keep the conversation small while you work, and plan big jobs around your limit instead of running into it.

Happy building!