CLN WORK

The Low-Effort Guide to Claude Fable 5.2

Anthropic's guidance for its newest model, tested on our own code. One measured test, eight prompt changes, and the order to apply them.

01

Why low effort

Effort controls how much the model thinks and how many tool calls it makes. The default is `high`. Anthropic's advice is to start at `high` and move down wherever your own tests show that quality holds.

We moved down. Anthropic reports that in its testing, Fable 5.2 at `low` scored close to what Fable 5.1 reached at higher levels. Our own test in section 03 points the same way for a well-specified task.

Effort barely changes how long the replies are. Set length and tone in the prompt, not with the effort level.

02

The effort ladder

EFFORT	ANTHROPIC RECOMMENDS IT FOR	KNOW THIS
`low`	Quick back-and-forth, well-specified tasks, subagents, high volume	More likely to stop early on long tasks.
`medium`	Interactive coding, chat, everyday knowledge work	Same early-stop risk on long tasks.
`high`	Hard coding, research, long unattended runs	Default. Thinks before almost every reply, even a greeting.
`xhigh`	Only the hardest long-running work	Use only where you measured a gain.
`max`	Measuring the ceiling, open-ended science and math	Slow. Many more tokens than xhigh.

03

Our test: low vs high

We took one real bug fix from our client portal: a change to how delivery feedback is settled, touching 15 files including tests. We reset the code to the state before the fix and gave Fable 5.2 the ticket text. Three runs at `low`, three at `high`, each in a fresh copy, with no human in the loop.

We graded each run with the tests from the real fix, which the model never saw: 84 unit tests in three files, plus the type check. Before the fix, 5 of the 84 fail.

RUN	TIME	COST	OUTPUT TOKENS	TESTS PASSED	TYPE ERRORS
low 1	4:50	$3.62	28,046	84 / 84	0
low 2	5:59	$4.20	32,895	84 / 84	0
low 3	5:36	$4.21	34,740	84 / 84	0
high 1	13:34	$9.32	72,919	84 / 84	0
high 2	15:07	$10.78	86,059	84 / 84	0
high 3	15:04	$10.40	82,368	84 / 84	0

All six runs passed all 84 tests with no type errors. On average, `low` took 5:28 and cost $4.01. `high` took 14:35 and cost $10.17. For the same test result, `high` needed about 2.5 times the cost and 2.7 times the time.

Cost is the API list price for the tokens used. Time is wall-clock time for the whole run. All six runs ran at the same time, which can slow each one down. Measured on 17 November 2026.

04

Where low falls short

- Our test is one task, three runs per level. It shows that low held here. It does not show that low holds everywhere.
- The ticket was well specified. That is the case Anthropic names for `low`. We did not test a vague task.
- The hidden tests check behavior, not design. We did not compare the quality of the code structure between the levels.
- Anthropic reports that the model is more likely to stop early at `low` and `medium` on long, multistep tasks. For long unattended runs it recommends `high`.

So test on your own tasks before you change a default. Judge cost per finished task, not per request.

EIGHT PROMPT CHANGES. ADD A BLOCK ONLY WHERE YOU SEE THE MATCHING BEHAVIOR.

05

Finish the whole task

You see: the model ends a turn before the work is done. It offers to continue, asks permission, or stops at a timeout. This is the one change that matters most at `low`.

In Anthropic's testing at `medium`, a version of this text left about half as many sessions with doable work unfinished. It removed most endings that ask permission or offer to continue. It did not remove endings that list what remains and then stop. Output tokens rose by about 30%.

```
# Finishing work
Please avoid stopping while you still owe work the user asked for. Be
persistent and do not stop unless needed.

While status notes are welcome, as is clarifying with the user on open
decisions, carry on with work that does not depend on the user's answer
without unnecessary stops.

If you notice yourself inviting the user to redirect you or offering to
wait, instead proceed on the next part of the task. You may meet ordinary
obstacles like errors, timeouts, locked files, empty results or failing
tools. Diagnose them first, and when they are not real blockers, work
through them with the access you have (wait and retry, fix the request,
use another tool or source) rather than stopping unnecessarily.

If someone clearly decided something is a hard blocker, such as a file
marked not to be touched, access that was intentionally withheld, or a
safety guardrail, leave it alone, say plainly what you found, and look
for another safe way to finish the task. Stopping before the task is
complete is very rarely merited, as in when the task can't move forward
without user input, or where the blocker is deliberate and should not be
worked past. This does not override the need for confirmation on risky or
destructive actions.
```

The model asks less often with this text. Keep your own confirmation step for risky or irreversible actions.

06

Delete the "think less" lines

Lines such as "answer directly without deliberating" worked on Fable 5.1. In Anthropic's testing they did not stop Fable 5.2 from thinking at `medium` and above. Remove them and lower the effort level.

07

Name what is approved

Tell the model what is approved instead of sending a bare "continue". For example: "Yes, run the migration on the staging database".

Approval rules in a system prompt are applied as written, so name the exact actions they cover. In Anthropic's testing, a rule that required approval for "sending messages to people outside this conversation" made the model hold back about half of its routine requests to colleagues.

08

Say who owns the code

You see: the model adds to the existing structure, repeats code, and reports nearby problems instead of fixing them. With this sentence it improves the structure as it works.

```
You are the owner and long-term maintainer of this codebase, and its
design is up to you.
```

The price, from Anthropic's testing on small coding tasks: about 1.7 to 1.8 times the output tokens, and changes nobody asked for, many of them tests and documentation. The closing summary often left out small changes. Review the diff, not only the summary. Leave the sentence out where you want the code changed as little as possible.

09

Updates on long runs

You see: long silence while the model works. Use this where a person watches the work. In Anthropic's testing the model then spoke before its first tool call every time, and output tokens per task rose by about 20%.

```
Before you start, say in a line what you're about to do; brief updates
while you work help the user follow along. Close with a short recap that
stands on its own (what you found, what you did, and what's next), so a
reader who only sees the last message has the full picture.
```

10

Subagents and cost

How much the model delegates depends on your setup. Where prompts invite delegation, it delegates readily, and subagents can be a large part of the cost of a session.

1. Review the lines in system prompts, skills and agent files that tell it to delegate.
2. Check which model your subagents run on by default.
3. Only if it still delegates too much: rewrite the description of the tool that starts subagents. Say what a subagent costs, when delegation is worth it, and how to brief one. Put cost first.

In Anthropic's testing at `high` effort, a longer description focused on cost roughly halved the number of subagents compared to a shorter version.

11

Old notes and "never" rules

Cautious instructions build up in memory files, notes and instruction files. This model follows them closely, so it may ask for approvals it does not need or hold routine work.

- Remove notes whose reason no longer holds.
- Name the steps that need no approval. Telling the model to act more freely is too vague, so list the routine steps it can take on its own.
- Write the exception into the rule. The model treats an instruction that contains "never" as absolute. Write "ask for approval before deleting files" instead of "never delete files".
- Check your stopping conditions. A condition such as "stop and review at the first error" can block autonomous work.
- Keep your own confirmation step for risky or irreversible actions.

12

Readable final reports

The person who reads the final report has not seen the tool calls behind it. First remove hard style rules you no longer need, because the model follows them closely enough to make the prose choppy. Then add this and adjust it to your readers.

```
# Communicating clearly
Write for an intelligent but low-context reader. The person reading has
not seen your tool calls, your reasoning or the files you opened, and may
be switching in from other work. Messages should generally be self-
contained: the reader should understand what you mean without having to
dig for other context. Put the key points in the first few sentences.
Practice progressive disclosure in your sentence order, and do not use
undefined references and ideas in line 1 that won't make sense until the
reader reaches line 12.

Use plain words, and write in clear, understandable full sentences. Do
not use a term or abbreviation that you made up during the work. Say what
the thing is in a way any reader would understand. Write mostly in the
style of ASD-STE100 Simplified Technical English: short sentences, one
idea per sentence, active voice, and consistently using the same word for
the same thing. Use a paragraph of flowing prose where ideas connect, and
keep bullets for lists of separate items. Leave out unnecessary details
unless the reader asks and practice concision, but do ensure that your
messages have the key context. Aim to make your user-facing text as
human-readable and clear as possible.
```

13

The order to apply it

This is Anthropic's order, with the step down to `low` added.

1. Set effort to `high` explicitly. Run your own test tasks at `medium` and `low` as well. Move down where quality holds.
2. Read everything you carried over: system prompts, skills, agent files, memory, notes. Remove or test what was written for older models.
3. Run your longest tasks. Note where the model stops early, asks for approvals it does not need, goes quiet, or delegates more than the task needs.
4. Add one change per behavior you saw. Measure its effect and its token cost each time.

14

Audit your prompts with one command

Section 13 says to read everything you carried over. In Claude Code you do not have to do that by hand. One command does the first pass.

```
/claude-api prompt-audit
```

What it checks. Prompts, tool descriptions, skills and agent configuration files such as CLAUDE.md, rule files, commands and subagents. It looks for text written for older models, for instructions the project has outgrown, and for instructions that contradict each other.

What you get. A report with each finding: the file and line, the pattern, why it is obsolete, and how confident the audit is. Plus a proposed diff.

What it does not do. It changes nothing unless you explicitly ask for the edits. You review the diff first.

How to run it. It runs without stopping to ask questions. So name the scope and the target model in the request:

```
/claude-api prompt-audit ~/projects/my-agent and ~/.claude/CLAUDE.md,
target Fable 5.2
```

A related command, `/claude-api migrate`, is for something else: it moves code that calls the Claude API to a newer model. For prompts, skills and agent files, use the audit.

Beyond your laptop

Everything in this guide makes one person with one agent faster. Getting AI into production is a different job: real workflows, real data, several people, and a human in control of every decision that matters.

That is what CLN Work does. Senior, founder-led AI implementation for small and mid-sized companies, delivered in small steps. If you want AI in production or your company workflows improved, talk to us: reply to the LinkedIn message this guide came in, or go to clnwork.com.

Sections 01, 02 and 05 to 13: figures and the four prompt texts are Anthropic's, from its page "Prompting Claude Fable 5.2" as of 17 November 2026. The prompt texts are quoted from that page. Section 03 is our own measurement. Section 14 describes a command in Claude Code and is not from that page. Read Anthropic's original.