FREE PROMPT PACK

The Claude Code prompt pack 12 prompts



MobiStack

mobistack.co/free/claude-code-prompts

Every prompt here comes from Anthropic's Best practices for Claude Code, checked against the live docs. Swap the parts in angle brackets for your own.

Every source opened on 30 October 2026. The page with every link: mobistack.co/free/claude-code-prompts

01 Fix a bug the specific way

When: you know the symptom and roughly where it lives.

```
users report <what fails> after <when it happens>. look at <the area> in <path>, write a failing test that reproduces it, then fix it
```

Why: the docs: "If you name the file, describe the symptom, and say what 'fixed' looks like up front, you usually save yourself a round of corrections later." Their own example: "users report login fails after the session times out. look at token refresh in src/auth/, write a failing test that reproduces it, then fix it".

02 Build it the way your code already works

When: a new feature should look like the ones you have.

```
follow the pattern in <ExistingFile> and build <the feature> that <what it does>. no new libraries
```

Why: the docs, on "a preference between two reasonable approaches": "Claude can't read that from the code, so if you care which one it picks, say so up front." Their example: "follow the pattern in HotDogWidget.php and build a calendar widget that lets you pick a month and page between years. no new libraries".

03 Ask for the full version

When: you want the edge cases handled, not the smallest thing that works.

```
build <the screen> from the mockup. be ambitious: empty and error states, the save flow wired up, keyboard navigation, not just the layout
```

Why: the docs: "Claude tends to read a request conservatively and build the smallest thing that satisfies it" and "One phrase that does still help is `be ambitious`."

04 Let it interview you first

When: you don't know exactly what you want yet.

```
I want to <goal> but haven't thought it through yet. Interview me until you could write the spec, then write it to SPEC.md.
```

Then start a fresh session to build from SPEC.md.

Why: the docs: Claude "comes back with multiple-choice questions about the parts you haven't pinned down", and "Once `SPEC.md` exists, start a fresh session to build from it, so the build begins with a clean context and a written definition of done."

05 Plan before it edits

When: the change touches several files, or you want to see the approach first.

```
/plan <the task>
```

Why: the docs: "Enter plan mode by pressing `Shift+Tab` or prefixing a single prompt with `/plan`." In plan mode "Claude reads files, runs shell commands to explore, and writes a plan, but does not edit your source."

06 Give it a check to pass

When: always, if you have tests, a build or a linter.

```
run the <name> tests after, and keep going until they pass. then show me the command you ran and its output
```

Why: the docs: "Give Claude something it can run that comes back pass or fail." The first half is their own example, "run the auth tests after, and keep going until they pass". The second half asks for what they call evidence: "the command it ran and its output, or the screenshot."

07 Set a goal for a long run

When: a long task you won't watch.

```
/goal all tests in <folder> pass and lint is clean
```

Why: the docs: `/goal` "sets a condition that a separate model checks against the conversation after every turn, and Claude keeps going until it's met." It "matters for long background runs".

08 Review the work against your spec

When: a feature is built from a spec.

```
use a subagent to review the <feature> diff against SPEC.md, check every requirement is implemented and the listed edge cases have tests, report gaps that affect correctness and skip style
```

Why: the docs, word for word except the feature. And: "Don't drop the `skip style` part of that prompt. A reviewer that's asked to find problems will always find *something*."

09 Get a fresh pair of eyes

When: before you open the pull request.

```
/code-review
```

Why: the docs: `/code-review` "reviews the diff in a fresh subagent that never saw your conversation. It isn't swayed by the reasoning that produced the code."

10 Start CLAUDE.md, then cut it

When: a new repo, or a CLAUDE.md that keeps growing.

```
/init
```

Then:

```
go through CLAUDE.md and list every line you could work out by reading the code and would actually go and look for. I'll remove those.
```

Why: the docs: "`/init` generates a starting version from your codebase. From there the work is mostly removing lines." Their test for every line: "1. Could it find this by reading the code? 2. Would it actually go and look? If both are yes, leave it out." The second prompt is our wording of that test. `/doctor` also "suggests cuts".

11 Make a rule unbreakable

When: something must never happen, whatever the model decides.

```
write a hook that blocks edits to the <folder> folder
```

Why: the docs: "If something has to happen every time, make it a hook, a deny rule, or a managed setting." Their example prompt, "write a hook that blocks edits to the migrations folder", "works as a prompt, and `/hooks` shows the result."

12 Start over, the right way

When: you have corrected the same thing twice and it is still wrong.

```
/clear
```

Then one prompt that folds in what you learned:

```
<the goal>. last time it went wrong because <what happened>. keep <the constraint>. done when <the check> passes
```

Why: the docs: "after two failed corrections run `/clear` and write a better prompt instead of a third", because "a third correction mostly adds to a conversation that already contains two failed attempts". The template is ours.

Three small ones

- Hand it the error: `cat error.log | claude -p "what's going on here"` (the docs' own example).
- Shrink a long session but keep what matters: `/compact keep the API changes and the test commands`.
- Just exploring? A vague prompt is fine: `what would you improve in this file?`

Before you leave it running

The docs' five questions: "Does it know what done looks like?" "Can it check its own work?" "What's it allowed to touch?" "What stops it?" "Where does the result go, and who signs off?"

Save it next to your terminal.

Sources, opened 30 October 2026

1. Anthropic, Best practices for Claude Code
   code.claude.com/docs/en/best-practices
2. Anthropic, Permission modes
   code.claude.com/docs/en/permission-modes

Follow us for more like this on Instagram, Facebook and LinkedIn.

Made by MobiStack (mobistack.co). We take over, fix and build apps and web platforms.