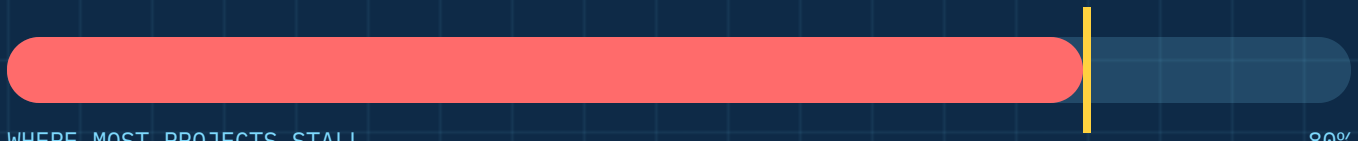
FREE CHECKLIST · FOR CLAUDE CODE PROJECTS

Past the 80% wall

The 5 habits that take a Claude Code app from "almost done" to actually shipped, with the exact prompts and templates to paste in.



p.2	The one-page checklist: print this one	ALL 5 HABITS
p.3	Spec before code · One phase at a time	HABITS 1–2
p.4	Project memory · Checks before "done"	HABITS 3–4
p.5	Bug reports + save points · Command cheat sheet	HABIT 5
p.6	Have you outgrown doing it alone? 4 questions	SELF-CHECK

How to use this: print page 2 and keep it next to your keyboard. When you start a new project, phase, feature or bug, open the matching page and copy the prompt. Anything in `[brackets]` is yours to fill in.

The one-page checklist

Tick these in order. If you skip one, that's usually where the 80% wall comes from.

1 Spec before code

ONCE, BEFORE YOU BUILD

- [ ] Claude interviewed me and wrote the answers to `SPEC.md` (prompt on p.3)
- [ ] The spec says who uses it, what they must do, what's **not** in version one, and what "done" looks like
- [ ] I started a **fresh session** and pointed Claude at the spec

2 One phase at a time

EVERY PHASE

- [ ] Phase 1 is the smallest thing a real customer can actually use
- [ ] Everything else is written on the "Not now" list
- [ ] Plan mode on (`Shift + Tab`). I read the plan and approved it before any code
- [ ] New ideas mid-build went on "Not now", not into my next prompt

3 Project memory

EVERY SESSION

- [ ] `CLAUDE.md` exists (`/init`) and is short: what the app is, key decisions, my rules
- [ ] Every decision worth remembering became one new line in it
- [ ] One session per task. `/clear` before starting something unrelated

4 Checks before "done"

EVERY FEATURE

- [ ] I wrote 3–5 plain-English checks **before** asking for the feature
- [ ] Claude proved each check with test output or a screenshot, not "it works"
- [ ] I tried at least one unhappy path myself (two users, bad input, expired card)

5 Bug reports + save points

EVERY BUG

- [ ] I reported it properly: what I did, what I expected, what happened, the exact error
- [ ] I asked for the **root cause**, not a quick patch
- [ ] Claude committed to Git the last time something worked
- [ ] Corrected the same thing twice? I stopped, ran `/clear` , and re-prompted better

HABITS 1–2

Spec before code. One phase at a time.

Most projects start with one sentence, so Claude fills the gaps with confident guesses. Ten minutes of questions saves weeks of wrong turns.

1 Get Claude to interview you

PROMPT · START OF A PROJECT

```
I want to build [one-line description of the app]. Before writing any code, interview me in detail. Ask one question at a time about who will use it, what they must be able to do, what is out of scope for version one, and how we'll know it's done. When you have enough, write the full spec to a file called SPEC.md.
```

What a good SPEC.md looks like:

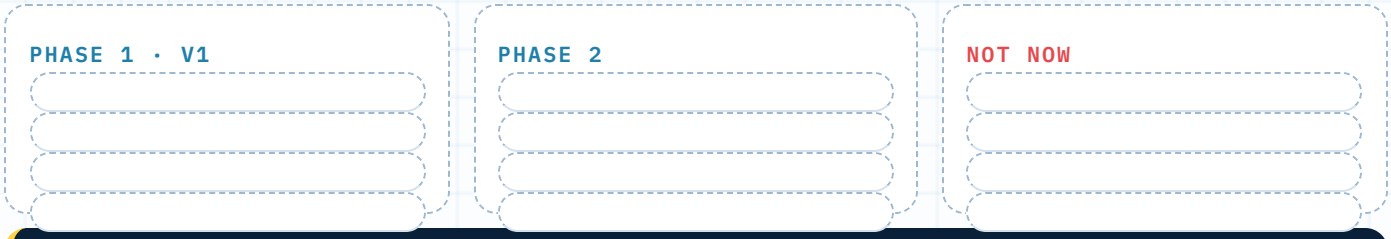
```
# [App name]: spec
## Who uses it
Local plant buyers, on their phones
## Must be able to (v1)
Browse, add to cart, pay, pick up
## Not in version one
Shipping, reviews, wishlists
## Done looks like
A real customer can order and pay
## Open questions
```

PROMPT · FRESH SESSION

```
Read SPEC.md. Propose how to split it into phases. Phase 1 must be the smallest version a real customer can actually use. Put everything else on a "Not now" list.
```

Why a fresh session? The interview chat is full of back-and-forth. A clean session that reads only the spec starts from the decisions, not the noise.

2 Draw the finish line: your phase board



PROMPT · PLAN MODE (SHIFT + TAB FIRST)

```
Plan only Phase 1 from SPEC.md. List what you'll build, the files you'll touch, and how we'll test it. Don't write any code yet.
```

Then: read the plan → push back on anything you don't understand → approve → let it code. A new idea mid-build goes on "Not now", never into the next prompt. That's how the finish line stops moving.

HABITS 3–4

Project memory. Checks before "done".

Decisions from week one quietly disappear by week four, and "done" usually means it worked once, for one user. Both are fixed by writing things down before Claude needs them.

3 Give the project a memory: CLAUDE.md

Claude Code reads `CLAUDE.md` at the start of every session. Type `/init` to create one, then trim it to something like this:

```
# What this app is
[Who it's for and what it does, in one line]

# Key decisions
- Prices are in US dollars
- Guests can check out without an account

# Rules
- Never change the checkout flow without asking me
- Never touch the order emails
- Show test results before saying "done"
```

Keep it working:

- Made a decision worth remembering? Add **one line**.
- Keep it lean. If it gets long, Claude starts skimming it. Delete lines that are no longer true.
- If Claude keeps breaking a rule, make the rule shorter and more specific, not louder.
- **One session per task.** Type `/clear` before starting something unrelated.

The memory belongs to the **project**, not the chat. Anything only said in a chat will eventually be forgotten.

4 Define "done" before you ask for the feature

```
Feature: [name]
Check 1: When [situation], then [result]
Check 2: When ______, then ______
Check 3: When ______, then ______

Example: checkout
Two people buy the last item at once → only one succeeds, the other sees "Sold out"
Expired card → clear error, no order created
Order paid → confirmation email arrives
```

Unhappy paths worth a check:

- Two people doing the same thing at once
- Empty, wrong or very long input
- Declined or expired payment
- The same button clicked twice
- Logged out or offline halfway through

PROMPT · WITH EVERY FEATURE

```
Here are the acceptance checks for this feature: [paste checks]. Build it, then prove each check: run tests and show me the output, or show me a screenshot. Don't tell me it works. Show me.
```

The rule: if you can't verify it, it isn't done.

HABIT 5

Bug reports + save points

"Fix it" describes a symptom. Claude patches the symptom, the cause moves, and something else breaks. Report bugs like a project manager, and never work without a way back.

5 Report the bug, not the feeling

What I did:
Added 2 plants, applied code SPRING10
What I expected:
Total: $63
What actually happened:
Total: $70, discount ignored
Exact error (copy-paste it):
TypeError: cannot read 'rate' of undefined

PROMPT · EVERY BUG

```
[Paste the bug report.] Before changing anything, find the root cause and explain it to me in two sentences. Then fix the cause, not the symptom, and tell me what else this change could affect.
```

Save points: always have a way back

PROMPT · EVERY TIME SOMETHING WORKS

```
This works now. Commit it to Git with a message that says what works.
```

Then, when something breaks: **"Go back to the last commit where checkout worked."**

Claude Code also keeps checkpoints of its own edits. Type `/rewind` or press `Esc` twice to roll back.

Corrected the same thing twice? Stop. Type `/clear` and start fresh with a better prompt that includes what you just learned.

Command cheat sheet

Shift + Tab	Plan mode: Claude plans before it writes any code
/init	Creates CLAUDE.md, the project's memory file
/clear	Fresh session. Use one per task
/rewind · Esc Esc	Roll back to an earlier checkpoint
SPEC.md	What you're building and what "done" means
CLAUDE.md	What the app is, key decisions, your rules

SELF-CHECK

Have you outgrown doing it alone?

These five habits get most projects over the wall. Most of the time, a few fixes really is all you need. Be honest with these four.

1	**Are real customers trusting you with their money or their data?**	☐ YES	☐ NO
2	**Are fixes still breaking other things, even with these habits in place?**	☐ YES	☐ NO
3	**Are you spending more time fixing the app than running your business?**	☐ YES	☐ NO
4	**Honestly: could you tell if it wasn't secure?**	☐ YES	☐ NO

For question 4, tick YES if the honest answer is "no, I couldn't tell".

0–1 yes: keep going

The habits are enough. Run the checklist on your next session and the wall usually moves.

2+ yes: not failure, just the next stage

This is where an experienced delivery team earns its fee. Someone who has shipped this before can usually tell you quickly whether you need a few fixes or a rebuild.

If you bring in help, ask these first

- **Have you shipped apps that take real payments or real data?** Ask for one example.
- **How do you decide something is "done"?** You want to hear "agreed checks, tested", not "when it works".
- **Will you review what I have before quoting?** A good team looks at the code and SPEC.md first.
- **Who owns the code and accounts?** The answer should be you, always.
- **What happens in the first two weeks?** Expect a review, a phase plan, and checks, before any big rebuild.

WANT A SECOND OPINION?

I'm a delivery project manager who has shipped software for US clients for years. If you answered yes twice, I'm happy to take a look.

Book a free session →

calendar.app.google/UJVDaVoiGHBNNRGB8