

# Build apps with Claude Code in 7 steps

Prompts alone ≠ an app. Use this workflow, in this order. Copy the prompts, replace the [brackets].

1

## /init — give Claude your project

Open Claude Code in your project folder and run the command. It reads your project and writes a CLAUDE.md file so every new session keeps the context.

```
/init
```

2

## PRD prompt — idea → full plan

```
I want to build [your app idea] for [who it is for].  
Write a PRD: goal, user stories, core features (MVP only),  
screens, data model, and what is out of scope.  
Ask me up to 5 clarifying questions first.
```

3

## Plan mode — think before code

Press `Shift` + `Tab` until it shows plan mode. Claude reads the code and proposes a plan without changing anything. Approve it, then build.

```
Using the PRD, plan the build in small steps. List the files you will create  
or change for each step.
```

4

## Debug prompt — real cause, then fix

```
Do not fix yet. Find the root cause of this bug: [error / what happens].  
Show the evidence (file + line), explain why it happens, then propose the  
smallest fix.  
After fixing, run /code-review on the change.
```

5

## Skills — save a prompt you repeat

Create `.claude/skills/code-review/SKILL.md` in your project (or `~/claude/skills/` for all projects):

```
---
name: code-review
description: Use when reviewing changes before commit
---

Review the diff for bugs, missing error handling,
security issues and unclear names. Give a short list,
most important first.
```

6

## Hooks — rules Claude cannot break

**BLOCKED**

A hook runs your script before a tool runs. Exit code 2 blocks the action and tells Claude why. Add to `.claude/settings.json`:

```
{ "hooks": { "PreToolUse": [{
  "matcher": "Bash",
  "hooks": [{ "type": "command",
    "command": ".claude/hooks/protect.sh" }] }] } }
```

```
# .claude/hooks/protect.sh (chmod +x it, needs jq)
cmd=$(jq -r '.tool_input.command')
if echo "$cmd" | grep -qE 'rm -rf|\.env'; then
  echo "Blocked: protected files" >&2; exit 2
fi
```

7

## Git commit — a save point after every step

Finish a step, test it, then commit. If something breaks, go back to the last save point.

```
Commit the current changes with a clear message
(what changed and why). Do not commit .env or secrets.
```



## The loop

/init → PRD → plan mode → build one step → debug prompt if needed → skill for repeats → hook for rules → commit. Repeat for each feature.