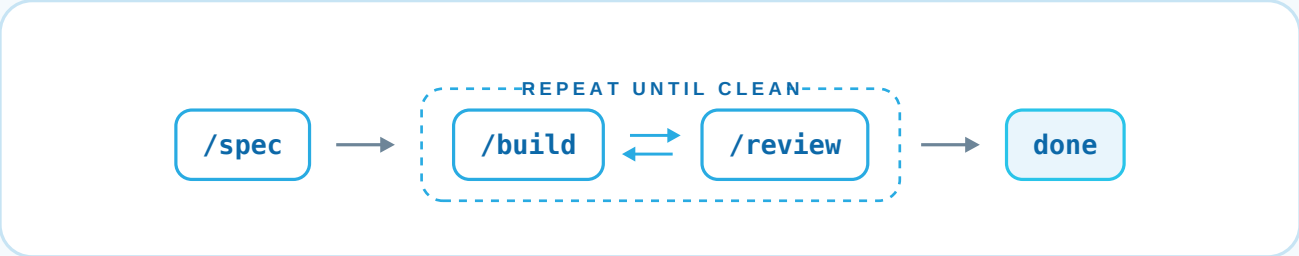
THE LOOPS BUILD GUIDE · IgCE ACADEMY

Build a loop that fixes its own work.

Three small skills turn Claude Code into a loop: it specs your idea, builds it, reviews its own build against the spec, and fixes whatever fails, until it passes clean.



FIRST, WHAT A LOOP IS

One prompt is a guess. A loop is a system.

A prompt is one shot: you ask, Claude answers, you take whatever comes back. A **loop** is a cycle: the model does the work, grades its own output against a standard, fixes the weak parts, and repeats until it's actually right.

It beats single prompts because scoring and rewriting are different acts. Forcing Claude to critique its own work before it rewrites turns blind editing into directed search. And with Fable 5 able to run for hours, the loop keeps going on its own until the job is done.

STEP 1 · BUILD THE SKILLS

Paste these into Claude Code

Each prompt builds one skill. Run all three once and you'll have `/spec`, `/build`, and `/review` installed.

01 /spec

Turns your idea into a plan

Interviews you until it understands what you want, then writes a detailed spec it can build against.

```
Create a Claude Code skill called "spec".

When I run /spec, interview me about the feature or app I want to build. Ask one
focused question at a time until you fully understand the goal, the must-have
requirements, the constraints, and what "done" looks like. Do not start building.

When you have enough, write a clear, detailed spec and save it to specs/<name>.md.
The spec must include: the objective, the exact requirements, the edge cases to
handle, and a concrete definition of done.
```

02 /build

Builds straight from the spec

Reads the spec and builds exactly what it says. No extra features, no scope creep.

```
Create a Claude Code skill called "build".

When I run /build, read the spec in specs/<name>.md and build exactly what it
describes. Do not add features, refactor unrelated code, or invent requirements
that aren't in the spec. When you finish, list which spec requirements you covered
so the review step can check them.
```

03 /review

Grades the build, sends back failures

Checks the build against the spec line by line, lists every gap, hands fixes back to /build.

```
Create a Claude Code skill called "review".

When I run /review, compare the current build against specs/<name>.md. Go
requirement by requirement and list every gap, bug, or missing piece, naming the
exact spec item each one fails. If anything fails, write the specific fixes needed
and hand them back so /build can address them. Only pass the build when every
requirement in the spec is fully met.
```

STEP 2 · RUN IT

Point the loop at your idea

1. Run `/spec` once and answer its questions. It turns your idea into a detailed spec.

2. Then paste this to start the loop:

```
Loop /build and /review: build from the spec, review the build against the spec, fix whatever fails, then repeat until the review passes clean. Keep going on your own until it passes.
```

3. Put Fable 5 on `high effort`, kick it off, and walk away. Claude builds, reviews its own work, fixes what fails, and repeats until the review passes.

EASY MODE

Or install a loop that’s already built

Feel a loop in 30 seconds. Install the free plan-optimizer skill (idea by @goodalexander, packaged by @seangeng). Hand it any plan and it scores, critiques, and rewrites itself until it can’t improve. One command:

```
curl -fsSL https://seangeng.com/plan-optimizer-skill.zip -o /tmp/plan-optimizer-skill.zip && unzip -o /tmp/plan-optimizer-skill.zip -d ~/.claude/skills/ && rm /tmp/plan-optimizer-skill.zip
```

NO INSTALL

The one-paragraph loop

Paste this after any task and Claude will loop on its own output. Works on writing, plans, code, anything with a quality bar.

```
After you finish, score your own work 0-100 against a clear rubric you write first. List the weakest parts and why they cost points. Rewrite to fix the top weaknesses while keeping what scored well. Repeat until the score stops improving by a real margin, then give me the best version and the score trajectory.
```