Hooks · Subagents · Skills

Three layers that make a repo agent-ready.

Prakash Poudel Sharma · 2026

The thesis, in one line

> ***Hooks enforce rules.***
> ***Subagents review work.***
> ***Skills are how the team actually invokes them.***

Three layers. Three different jobs. The trick is knowing which to reach for.

The problem

You write a rule in `CLAUDE.md`. The agent reads it. Two prompts later it forgets.

You leave the same PR comment on three branches in a week:

- "Don't import from the barrel."
- "Don't use `crypto.randomUUID` — we have a helper."
- "Don't edit `dist/`."

Mechanical reminders. The same mistake. You're the regex now.

Why CLAUDE.md isn't enough

`CLAUDE.md` is advisory. It's the agent's onboarding doc.

- It teaches.
- It doesn't enforce.
- It's read once per session — and forgotten under load.

For mechanical rules, you need something that fires on every write. For judgement rules, you need something that reads the code. Different layers, different tools.

Layer 1 — Hooks

Shell commands wired to lifecycle events. The agent tries to do a thing; the hook decides whether the thing happens.

- **Deterministic.** No model in the loop.
- **Fast.** Exits in milliseconds.
- **Exit code 2 = block.** The agent sees stderr and retries.

If you can write a `grep`, you can write a hook.

What hooks are good at

- Reject barrel imports (`from '@/components'`).
- Reject `crypto.randomUUID` — point at your helper.
- Reject edits to `dist/` and other generated paths.
- Run `prettier --write` on every `Edit` .
- Block destructive `bash` (`rm -rf` , force-push to main).

The pattern is always the same: **read the tool input, decide, exit.**

A hook in 20 lines

```
#!/usr/bin/env bash
# .claude/hooks/no-barrel-imports.sh
set -euo pipefail

input=$(cat)
content=$(jq -r '.tool_input.new_string // empty' <<<"$input")

if grep -qE "from ['\"]@/components['\"]" <<<"$content"; then
  echo "Barrel imports are banned. Import from the file directly:" >&2
  echo "  from '@/components/Button'  ✓" >&2
  echo "  from '@/components'         ✗" >&2
  exit 2
```

That's the whole pattern. Read stdin, grep, exit 2.

Layer 2 — Subagents

Specialist workers. The main agent delegates a task; the subagent reads the code, forms a judgement, returns a structured report.

- **Read-only by default.** Restrict their tool surface.
- **Pinned output format.** Markdown, table, JSON — your choice, but fixed.
- **Scoped.** One job. Type safety. Accessibility. Cache keys.

Hooks block patterns. Subagents make judgement calls.

What subagents are good at

- "Is this `any` actually justified, or did the agent give up?"
- "Does this React Query hook have a sensible cache key?"
- "Are these `useEffect`s causing re-renders we missed?"
- "Does this component meet our a11y bar?"
- "What barrel imports are still in the codebase?"

You can't grep a judgement. You can ask a subagent.

A subagent, sketched

```
---
name: barrel-import-auditor
description: Find remaining barrel imports and suggest concrete fixes.
tools: [Read, Grep, Glob]
---

You are a read-only auditor. Find every import of the form
`from '@/components'` (and similar barrel re-exports).

For each, return a row in this table:

| File | Line | Current import | Suggested rewrite |
```

Tool surface is the contract. Output shape is the contract.

Layer 3 — Skills

Slash commands. The user types `/de-barrel`; a 60-line prompt fires, loading the right docs, calling the right subagent, walking the right edits.

- **Discoverable.** Tab-complete in the prompt.
- **Composable.** Skills call subagents. Hooks guard the edits skills perform.
- **Team-shaped.** A skill is how knowledge spreads without a meeting.

The user typed eight words. The skill did the workflow.

What skills are good at

- `/migrate-list Orders` — migrate one list view to the new pattern.
- `/de-barrel` — sweep barrel imports, file by file, with the hook guarding each edit.
- `/audit-rerenders` — run the re-render subagent, report top offenders.
- `/perf-audit` — bundle size, lazy boundaries, hydration cost.

Same primitive: a markdown file in `.claude/commands/`. The leverage is in the workflow it encodes.

A skill, sketched

```
---
name: de-barrel
description: Remove barrel imports across the codebase, one file at a time.
---

1. Invoke @barrel-import-auditor. Get the table.
2. For each row, top-to-bottom:
   - Open the file.
   - Replace the barrel import with the direct import the auditor suggested.
   - Save. The hook will reject any remaining barrels — fix, don't bypass.
3. Run `pnpm typecheck`. If clean, summarise rows touched.
```

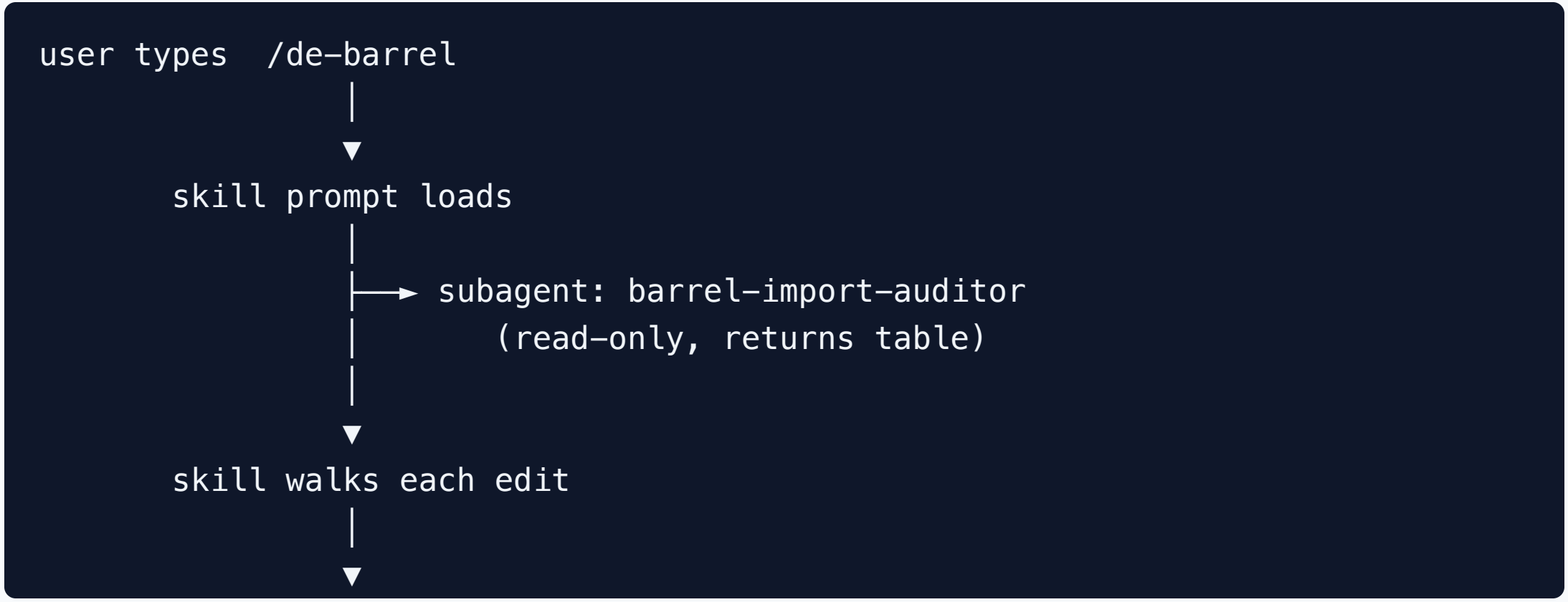
Auditor finds. Skill drives. Hook guards. Three layers, one job.

The decision matrix

	Hook	Subagent	Skill
Mechanical rule?	✓		
Needs judgement?		✓	
Team-discoverable?			✓
Reads code?		✓	(via subagent)
Edits code?			✓
Blocks edits?	✓		

Three different jobs. Three different layers. Don't ask one to do another's work.

How they compose



The skill is the verb. The subagent is the eye. The hook is the gate.

When to add which

A simple heuristic for any rule:

1. **Can a `grep` decide?** → hook.
2. **Does it need to read the surrounding code?** → subagent.
3. **Will a teammate forget to invoke it?** → wrap it in a skill.

Start at the bottom. Promote a rule up the stack only when the layer below can't carry it.

What you don't have to build

- Not every rule needs all three layers.
- A single sharp hook beats a vague skill.
- A subagent without a pinned output format is just another chat.
- A skill that only invokes one tool isn't a skill — it's a shortcut you typed once.

Build the layer that earns its keep. Resist the urge to make a `/do-the-thing` for every workflow.

The hands-on workshop

This deck is half the story. The other half is a 65-minute workshop:

- **Lab 1 — Hook (15 min):** write a `PreToolUse` hook that blocks barrel imports.
- **Lab 2 — Subagent (20 min):** author `barrel-import-auditor` with a pinned report shape.
- **Lab 3 — Skill (15 min):** wire `/de-barrel` to run the auditor and walk the edits while the hook guards each one.
- **Capstone (10 min):** design the hook/subagent/skill triple for a rule you keep repeating.

Starter repo. One planted bug. Three labs.

Read the series

Agent-ready React — the four posts behind this talk:

- The Claude Hooks Lifecycle Primer
- From Audit to Hook Enforcement
- Subagents That Catch What Hooks Can't
- Skills: The User-Facing Workflow Layer

Workshop: sharmaprakash.com.np/workshop/hooks-subagents-skills/

Thanks

Scan to open the workshop.

sharmaprakash.com.np