// THE OPERATOR'S MANUAL

V1 · EDITION 2026 · BY AVIK BHANJA

Claude Code.

How I ship real products, every day, with one CLI.
Slash commands, hooks, MCP servers, and the workflows I actually use.

@techplus.avik

A free field guide · Not a tutorial

// FOREWORD

Read this *first.*

This is not a Claude Code tutorial. There are a hundred of those, and the docs are good. This is the field guide I wish someone had handed me a year ago — the part nobody puts in the docs because it sounds too opinionated.

I use Claude Code 8–10 hours a day. I ship for US clients, build content for 600+ devs, and run my own products from a laptop in India. Everything in here is muscle memory now. I hope it saves you a few months.

Read it in one sitting (it's short on purpose), then keep it open the next time you sit down to ship something. Skip around — the chapters are independent.

No fluff. No "and finally, the future of AI coding." Just what's wired up on my machine right now.

— *Avik*

// CONTENTS

What's *inside.*

// CHAPTER 01

The mental model.

Most devs treat Claude Code like a smarter autocomplete. That's the mistake. It's a pair, not a plugin.

// 01 · THE MENTAL MODEL

Cursor types with you. Claude Code builds for you.

Same model under the hood, totally different posture. Pick the tool to match the task — not the other way around.

YOU WANT TO…	CURSOR / COPILOT	CLAUDE CODE
Type code faster	✓ Best fit	—
Refactor 12 files	—	✓ Best fit
Plan a feature	—	✓ Best fit
Run + verify tests	—	✓ Best fit
Inline tab-complete	✓ Best fit	—
Multi-step debugging	—	✓ Best fit

RULE OF THUMB

If the task involves **more than one file**, more than one step, or any decision-making — open Claude Code. If it's "finish this line," stay in your editor.

Three jobs I delegate to Claude Code daily

- → Reading a codebase I've never seen before — "explain this repo in 10 bullets."
- → Writing the boring half — auth scaffolds, CRUD endpoints, test setups.
- → Reviewing my own PRs before a human sees them — catches what I glaze over.

// CHAPTER 02

Setup that *actually* matters.

Skip the cosmetic stuff. These are the three files that change how Claude behaves on your machine — forever.

// 02 · SETUP · 1 OF 2

Your CLAUDE.md is the most leverage you have.

It's read on every conversation start. Treat it like a system prompt for your codebase. Short, declarative, no fluff.

// CLAUDE.MD (PROJECT)

```
# Project Rules

## Stack
- Next.js 15, App Router, TS strict
- Postgres (Drizzle), Tailwind v4
- Resend for email, Stripe for billing

## Conventions
- Server components by default
- No barrel files (../index.ts)
- Use zod for ALL form input

## Don't do
- Don't use any. Use unknown + narrow.
- Don't add a lib without asking first.
```

// WHAT GOES IN HERE

- → **Stack & conventions** — so Claude doesn't pick the wrong library.
- → **Anti-patterns** — the "don't do this" rules nobody documents.
- → **Glossary** — internal terms (e.g. "Workspace = top-level account container").
- → **Test runner** — the exact command you'd run in CI.

What does NOT go here

- → Long histories of past decisions — use commit messages.
- → Anything > 200 lines — Claude will skim. Keep it dense.

ONE-LINER THAT PAYS FOR ITSELF

Add this to your CLAUDE.md: *"When unsure about scope, ask one clarifying question before writing code."* It saves an hour a week.

// 02 · SETUP · 2 OF 2

Settings & permissions: set once, forget.

Two files. Five minutes. Then your shell does the right thing every time you launch.

~/.claude/settings.json

```
{
"model": "claude-opus-4-7",
"permissions": {
"allow": ["Bash(git status:*)", "Bash(npm test:*)"],
"deny": ["Bash(rm -rf:*)", "Bash(git push --force:*)"]
},
"statusLine": { "type": "command", "command": "~/.claude/sl.sh" }
}
```

- **Permissions** — auto-approve the boring stuff. Block the destructive stuff. You'll tab through 50% fewer prompts.
- **Status line** — show git branch + token usage. A 10-line shell script changes how much you trust the session.
- **Model pin** — pinning the model means consistent behavior across machines.

> PRO MOVE
>
> Keep two CLAUDE.md files: **~/.claude/CLAUDE.md** (your global preferences — tone, style, what to avoid) and **./CLAUDE.md** (per-project). Global rules persist across every project you ever open.

// CHAPTER 03

The 12 commands I run *every day.*

Eight you'll use within a week. Four that feel weird at first, then become indispensable.

// 03 · COMMANDS · 1 OF 3

The starters. Every Claude Code user uses these.

If you only learn four, learn these. They cover 80% of a normal day.

/init

01

ESSENTIAL

Generates a starter CLAUDE.md by reading your repo. Best 30 seconds you'll spend in a new project.

```
$ /init
✓ CLAUDE.md created
```

/clear

02

ESSENTIAL

Wipes the session and starts fresh. Use the moment a task is done — don't carry context across tasks.

```
$ /clear
→ new conversation
```

/compact

03

ESSENTIAL

Squeezes the conversation into a summary so you can keep going. Cheaper than /clear when context is precious.

```
$ /compact
✓ 42k → 3k tokens
```

/agents

04

ESSENTIAL

List, create, or edit subagents. Build one specialist (e.g. "test-runner") and Claude calls it instead of doing the work itself.

```
$ /agents
› create new
```

> *"The dev who masters four commands beats the dev who installs four IDE plugins."*
>
> — FIELD NOTE, WEEK 3

// 03 · COMMANDS · 2 OF 3

The daily drivers. Use 'em without thinking.

These four show up in 80% of my sessions. None are flashy. All are load-bearing.

/memory

05

COMMON

View and edit Claude's persistent memory about you and your projects. Surprisingly the most underused command.

```
$ /memory
› edit user.md
```

/model

06

COMMON

Switch model mid-session. Drop to Haiku for greppy work, Opus for design decisions. Saves real money.

```
$ /model haiku
✓ switched
```

/review

07

COMMON

Run a code review on your last changes. I run this before every push. Catches dumb stuff humans miss.

```
$ /review
⚠ 3 issues found
```

/design

08

COMMON

Routes any UI/UX request to the right path — visual mockup, production code, or design review. Stops "vibe code, ugly UI."

```
$ /design pricing page
→ spawns frontend-artisan
```

HABIT TO BUILD

End every coding session with **/review** → fix → **/clear**. Two commands, ninety seconds, dramatically better code shipped.

// 03 · COMMANDS · 3 OF 3

The power moves. Less common, more leverage.

These four are custom or rarely-used. They're the ones that make people watching me code go "wait, what was that?"

/loop

09

POWER

Re-runs a task on a schedule or until a condition is met. The closest thing to a junior who never sleeps.

```
$ /loop "check CI"
→ every 5m until green
```

/powerup

10

POWER

Bumps Claude into deeper-thinking mode for the next message. Use before any architectural decision.

```
$ /powerup
› design auth flow
```

/ultrareview

11

POWER

Spawns a multi-agent code review on the current branch. Catches what you glaze over at 11pm.

```
$ /ultrareview
→ 3 agents, 1 verdict
```

/schedule

12

POWER

Runs an agent in the background on a cadence — daily standup, weekly cleanup, monthly audit. Hands-free maintenance.

```
$ /schedule cleanup
✓ weekly · Mondays
```

> *"Custom commands are how a CLI becomes a co-worker."*
>
> — A TRUTH ONLY OBVIOUS IN HINDSIGHT

// CHAPTER 04

Hooks that *save hours.*

Hooks are shell scripts that fire on lifecycle events. The cheapest automation in the entire stack.

// 04 · HOOKS · THE LIFECYCLE

Five points where you can inject yourself.

A hook fires before or after specific events. Want to block writes to `/etc`? Auto-format on save? Send a Slack ping when work finishes? One hook each.

01

UserPromptSubmit

Inject context (e.g. current git branch) into every prompt before Claude sees it.

02

PreToolUse

Block, modify, or approve a tool call. Best place to enforce safety rules.

03

PostToolUse

Run something after a tool call — auto-format, lint, push to remote.

04

Stop

Fires when Claude stops responding. Great for "ping me when it's done" notifications.

05

Notification

Fires on permission requests or idle waits — pipe to push notifications, sound, anything.

MENTAL SHORTCUT

If you're typing the same command after Claude finishes — *that's a PostToolUse hook waiting to be written.*

// 04 · HOOKS · REAL EXAMPLES

Two hooks I'd never code without.

1. Auto-format after every write

Saves a manual "now run prettier" every time Claude touches a TS file.

```
# settings.json
"hooks": {
"PostToolUse": [{
"matcher": "Write|Edit",
"hooks": [{ "type": "command", "command": "npx prettier --write
$CLAUDE_FILE_PATHS" }]
}]
}
```

2. Block writes outside the project root

A two-line guardrail that has saved me from "wait, why is there a node_modules in my home folder."

```
# .claude/hooks/guard.sh
case "$CLAUDE_FILE_PATHS" in
"$PWD"*) exit 0 ;;
*) echo "blocked: outside repo" >&2; exit 2 ;;
esac
```

> WHEN NOT TO USE A HOOK
>
> Hooks fire silently — debugging is painful. If logic needs branching or AI judgement, write a subagent instead. Hooks are for one-line, deterministic glue.

// CHAPTER 05

Subagents — when to *spawn.*

Most people overuse them. A few people underuse them. Almost nobody uses them right.

// 05 · SUBAGENTS · THE DECISION

Spawn one when, and only when...

A subagent has its own context window. That's a feature (focus, parallelism) and a cost (tokens, no shared memory). Use the rule below.

- The task is **independent** of the main thread (research, lookup, broad search).
- Or the task would **pollute** your context (giant logs, big file reads, browse + summarize).
- Or you need **parallel work** — two agents researching two topics in the time it takes one.
- Or you need a **fresh perspective** — independent code review, second opinion on a design call.

Don't spawn one when...

- The work depends on the conversation so far. The agent won't see it.
- The task is small (< 30 seconds of work). The spawn overhead isn't worth it.
- You'd need to babysit. If you check in every minute, do it yourself.

> THE PATTERN THAT PAYS
>
> For any non-trivial coding task, run **Plan** → **Build** → **Review** with one reviewer subagent at the end. Two prompts, one reviewer, dramatically better code. (Next page.)

// 05 · SUBAGENTS · THE PATTERN

Plan → Build → Review. The only workflow you need.

This sounds simple because it is. But the gap between knowing it and running it on every task is the gap between average and shipping.

Step 01

Plan

Ask Claude for an implementation plan first. No code yet. Read it. Approve or redirect.

Step 02

Build

Once the plan is right, write all the code in one go. Don't review mid-stream — it interrupts flow.

Step 03

Review

Spawn ONE reviewer subagent acting as a senior dev. Critiques quality, security, UX. Apply feedback in one pass.

Why one reviewer, not three

- → Three reviewers triple your tokens for marginal gain.
- → Conflicting reviewer opinions waste a turn arbitrating.
- → One smart prompt to one reviewer beats a committee.

CAP THE LOOP

Maximum two reviewer calls per task. One review + one verification pass. Anything more is rabbit-holing.

// CHAPTER 06

The MCP stack I keep *installed.*

There are 5,000+ MCP servers now. You need five. Here are mine.

// 06 · MCP · THE FIVE

Five servers. Everything else is noise.

Each one earns its keep weekly. Install only what you'll actually use — every server is context overhead.

F

Filesystem

Lets Claude read/write files outside your project directory — config files, scratch notes, cross-repo work. The default MCP everyone should have.

```
claude mcp add filesystem ~/projects ~/.claude
```

G

GitHub

Browse PRs, file issues, read repo READMEs without leaving the CLI. Pairs perfectly with /review on someone else's PR.

```
claude mcp add github
```

P

Playwright

Headless browser. Claude can open a URL, take a screenshot, fill a form, scrape — debug your own UI without DevTools.

```
claude mcp add playwright
```

// CHAPTER 07

Five workflows I run *weekly.*

Patterns, not one-offs. Each one took me months to refine. Steal them.

// 07 · WORKFLOWS · 1 OF 2

The first three. Daily-ish cadence.

WORKFLOW 01

Codebase audit (new repo onboarding)

TRIGGER	First time opening any unfamiliar codebase.
STEPS	/init → ask "explain this repo's architecture in 10 bullets" → ask "where would I add feature X?" → /review on the README.
OUTPUT	A CLAUDE.md tuned to the repo, a mental map, and an entry point.

WORKFLOW 02

PR review before pushing

TRIGGER	After every "build" phase, before git push.
STEPS	/review on the diff → fix issues → run tests → /ultrareview if the change touches anything sensitive (auth, billing, migrations).
OUTPUT	Cleaner PR. Fewer reviewer comments. Less embarrassment.

WORKFLOW 03

Refactor a legacy module

TRIGGER	"This file is 800 lines and I hate it."
STEPS	Plan first ("propose 3 refactor strategies") → pick one → /powerup → build → review → run tests after every chunk.
OUTPUT	Same behavior, half the code. Tests prove it.

// 07 · WORKFLOWS · 2 OF 2

The last two. Plus what NOT to do.

WORKFLOW 04

Debug a production issue

TRIGGER	Bug report from a user. You're not sure where to start.
STEPS	Paste the error → ask "what are the 5 most likely causes, ranked?" → investigate top 2 → fix → write a regression test.
OUTPUT	Fix + test that ensures it doesn't come back.

WORKFLOW 05

Weekly hygiene sweep

TRIGGER	Friday afternoon, low energy, no new features.
STEPS	/schedule a weekly agent that runs: "audit unused deps, flag stale TODOs, list flaky tests." Read its report Monday.
OUTPUT	A codebase that doesn't rot. Monday-you thanks Friday-you.

Three anti-patterns I see weekly

- **Treating Claude like Stack Overflow.** If you're copy-pasting answers, you've lost the leverage. Have it write the file.
- **Never running /clear.** Stale context = worse output. End every task with /clear.
- **No CLAUDE.md.** You're paying for the same explanation every conversation. Write it once.

// CHAPTER 08 · THE END

Now go ship *something.*

// INSTAGRAM
@techplus.avik

// SITE
avikbhanja.tech

// GITHUB
github.com/avikbhanja

// CALENDLY
15-min consult — free