FIELD GUIDE · JULY 2026 EDITION

Claude Code

From first session to power user · the complete engineer's guide

By Franklin Tranié · blog.tranie.org

For an engineer who has never used a coding agent: install, models, daily workflows, parallelism, security, plus a seven-day action plan.

Claude Code lives in the terminal: here, its original welcome screen in Anthropic's colors.

Sources: official docs code.claude.com/docs · claude.com/pricing (24 July 2026) · support.claude.com · feature "AI for Software Engineers: The July 2026 Field Guide" (blog.tranie.org)

CONTENTS

What you'll be able to do by the end

≈ 21 pages · reads standalone · every chapter ends with a concrete action

Claude Code is a model in a loop, not a chatbot

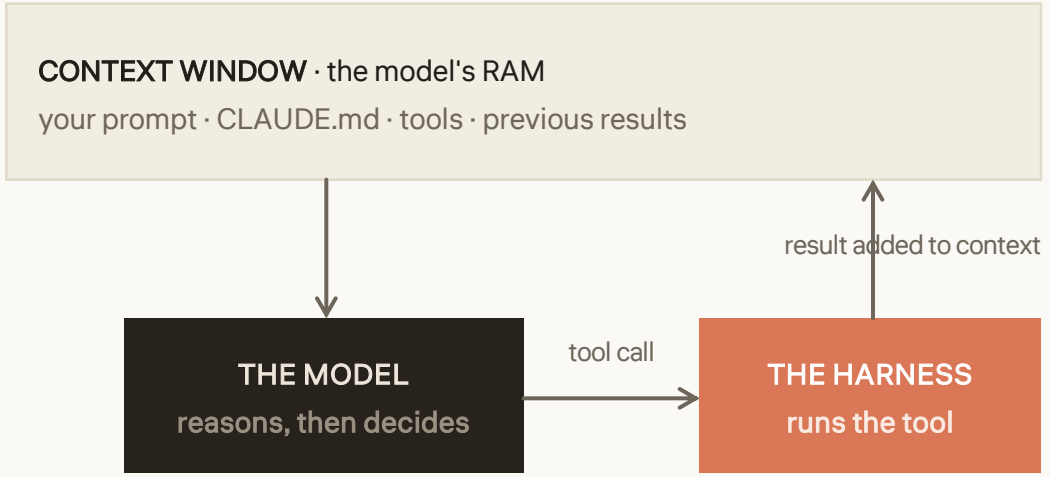
A coding agent = a language model run in a loop, with tools.

The model gets the context (your request, CLAUDE.md, the conversation), picks an action (a **tool call**), the terminal runs it, and the result comes back into the context. Then the loop starts again, until the work is done.

The core of an agent fits in ~400 lines of code (Thorsten Ball demonstrated it). Claude Code is that loop, refined: better tools, permissions, memory, checkpoints, parallel sessions.

What this changes for you: it works in your real repo, with your shell and your tests, not in a sandbox.

Takeaway: output quality depends as much on the harness (tools, memory, checks) as on the model. This whole guide is about configuring that loop better.



The harness tools: bash · read/edit files · git · run tests and linters · web search · MCP servers

The loop runs for minutes, sometimes hours, and that's what separates an **agent** from a plain chat reply.

Source: blog.tranie.org, fig. 1 (July 2026) · Thorsten Ball, "How to Build an Agent"

One agent, four surfaces: the terminal is still the reference

Terminal (CLI)	Full repo access + shell, long multi-step tasks, scriptable, memory via CLAUDE.md.	**When:** daily work, this is where it all happens.
IDE extension	VS Code and others: visual diffs, code selection as context, review inside the editor.	**When:** exploring and reviewing changes closely.
Desktop and web	Supervised sessions from the Claude app or the browser, including remote work you kick off.	**When:** delegating a task and watching it remotely.
Headless (claude -p)	No interface: one command, JSON output. Same engine, driven by your scripts (see ch. 4).	**When:** CI, cron, pipelines, automation.

INSTALLATION · 30 SECONDS

```
› curl -fsSL https://claude.ai/install.sh | bash # macOS · Linux · WSL
› winget install Anthropic.ClaudeCode # Windows
› claude # start a session in the project folder
```

Source: code.claude.com/docs · setup (July 2026) · automatic updates with the native install

Your first session in five minutes

1. **Install** with the command from the previous page, then check: **claude --version** (**claude doctor** checks the config).

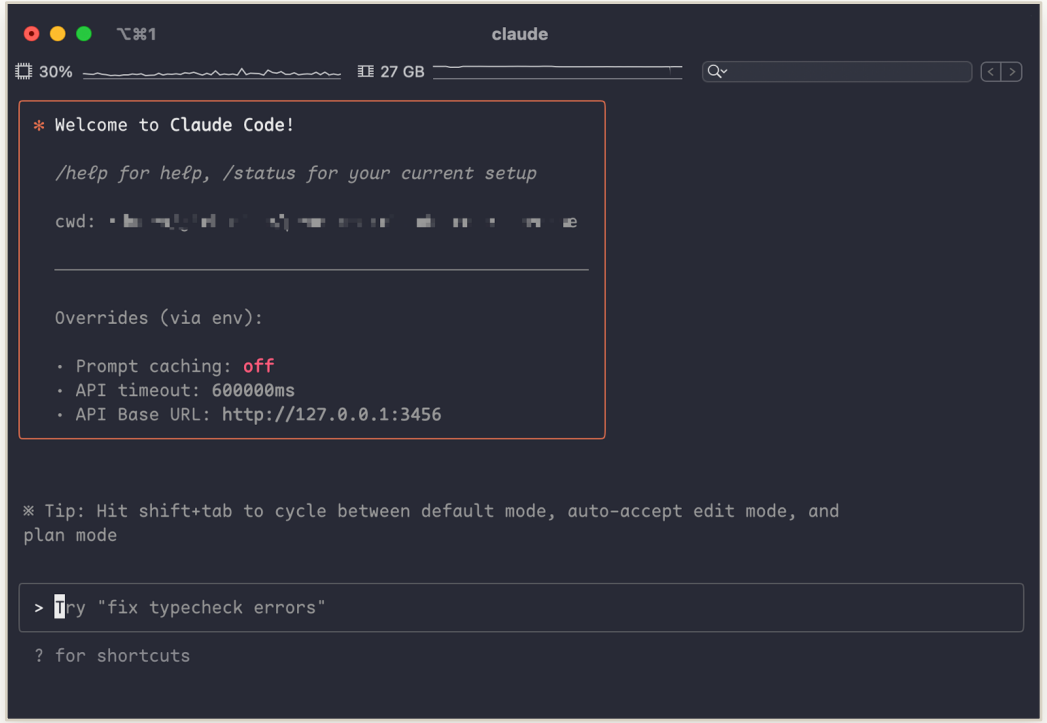
2. **Work inside a project**: **cd** to the repo, then **claude**. Claude only sees that folder and its subfolders.

3. **Log in** on first launch: Pro / Max / Team / Enterprise subscription or a Console account. The free plan doesn't include Claude Code.

4. **Initialize memory**: **/init** writes a first CLAUDE.md, the file Claude re-reads every session (chapter 3).

5. **Explore before you edit**: "explain how authentication works in this repo". Interrogating your own code is the best prompting school.

⚠ **Never run claude at the root of your disk.** It can read and change everything it sees, so give it a project folder, not your whole machine.



The welcome screen of a session: greeting, current folder (cwd), and the **Shift+Tab** tip for switching modes, more on page 9.

Source: code.claude.com/docs · quickstart (July 2026)

Four models, four roles: Opus 5 just changed everything

Model	Price (in / out, per MTok)	Role	When to use it
Opus 5 · new	$5 / $25	The new default for coding	Shipped 24 July 2026: beats Fable 5 on most public benchmarks at half the price. Max plan default. Fast mode ×2.5 at 2× the price
Fable 5	$10 / $50	Mythos class: the top	Only if your own measurements show Opus 5 falls short: it still leads on a few extreme agentic evals
Sonnet 5	$2 / $10 (intro → $3 / $15 on 1 Sept.)	The workhorse	Volume, repetitive tasks, subagents: intro pricing until 31 August 2026
Haiku 4.5	$1 / $5	Speed	Exploration, summaries, small tasks at scale, running hooks

Control in session: /model switches model on the fly; **/effort** sets reasoning depth (low → max). Same task, same price per token: effort changes how many tokens burn, so the cost per task.

The "best model" has a shelf life of a few weeks. Re-test monthly on your own tasks and keep the one whose output you'd ship, not the one in the launch posts.

Simple rule: Opus 5 by default, Sonnet for volume, Haiku for speed, Fable 5 only on measured proof.

Source: anthropic.com/news/claude-opus-5 (24 July 2026) · claude.com/pricing (26 July 2026) · benchlm.ai

Subscription or API: two ways to pay, one tradeoff to know

SUBSCRIPTION · FOR INTERACTIVE

Pro: $20/month. Claude Code included, Sonnet 5 by default with Opus 5 access, 5 h usage windows. The baseline for engineers.

Max 5× / 20×: $100-200/month. Opus 5 by default, much higher caps, Fable 5 via usage credits. For heavy daily agent work.

Subscriptions subsidize tokens heavily: the same interactive work would cost several times more at API rates.

Rule: all your interactive work lives on the subscription, the cheapest frontier compute on the market.

API · FOR PIPELINES

Billed per token, no session cap. Opus 5: $5/$25 · Sonnet 5: $2/$10 (intro) · Haiku 4.5: $1/$5 per million tokens. 1M window at no extra cost.

For: CI, cron, headless, automation. A real coding task burns 400 K-2 M tokens in total, because the conversation is resent every turn.

Rule: anything that runs without you (claude -p, SDK) burns API credits, so budget for it.

THREE LEVERS · TO CUT THE API BILL

1 · Prompt caching

Up to **-90%** on repeated context (system prompt, tools, large files re-read).

2 · Batch API

-50% on anything asynchronous: overnight jobs, bulk classification, evals.

3 · Routing by task

Haiku/Sonnet for volume, Opus 5 for judgment: often **10-50× cheaper** on high-volume steps.

Source: claude.com/pricing (26 July 2026) · blog.tranie.org, part 06 · "The economics"

CLAUDE.md: the highest-return 30 minutes of setup

What it is: the file Claude re-reads automatically at the start of every session. It's the project memory, the difference between a generic assistant and a briefed teammate.

What goes in:

· build, test, lint commands, so Claude can verify on its own
· code conventions and architecture in five lines
· off-limits areas and known repo traps
· the definition of "done" (green tests, diff read…)

Golden rules:

· **/init** writes a first version, then rework it
· **under 200 lines**, beyond that it dilutes focus
· **committed to git**, the whole team benefits
· **nested** CLAUDE.md per subfolder for specifics

EXAMPLE · ADAPT IT TO YOUR REPO

```
# CLAUDE.md
## Commands
build : pnpm build
test  : pnpm test --run
lint  : pnpm lint --fix
## Conventions
TypeScript strict: never any
function components + hooks
commit messages in English
## Never
never touch /config/prod
no migration without backup
## Done when
tests + lint green, diff read
```

Treat CLAUDE.md as onboarding docs for a brilliant new hire: that's exactly what it is, and it reads them every session.

Source: Anthropic, Claude Code Best Practices · support.claude.com

Plan mode: think before writing a single line

THE MODE CYCLE · SHIFT+TAB

Normal: every action asks for your approval.

Auto-accept: file edits go through without confirmation.

Plan mode: read only. Claude explores and plans, never writes.

A WORKFLOW THAT WORKS

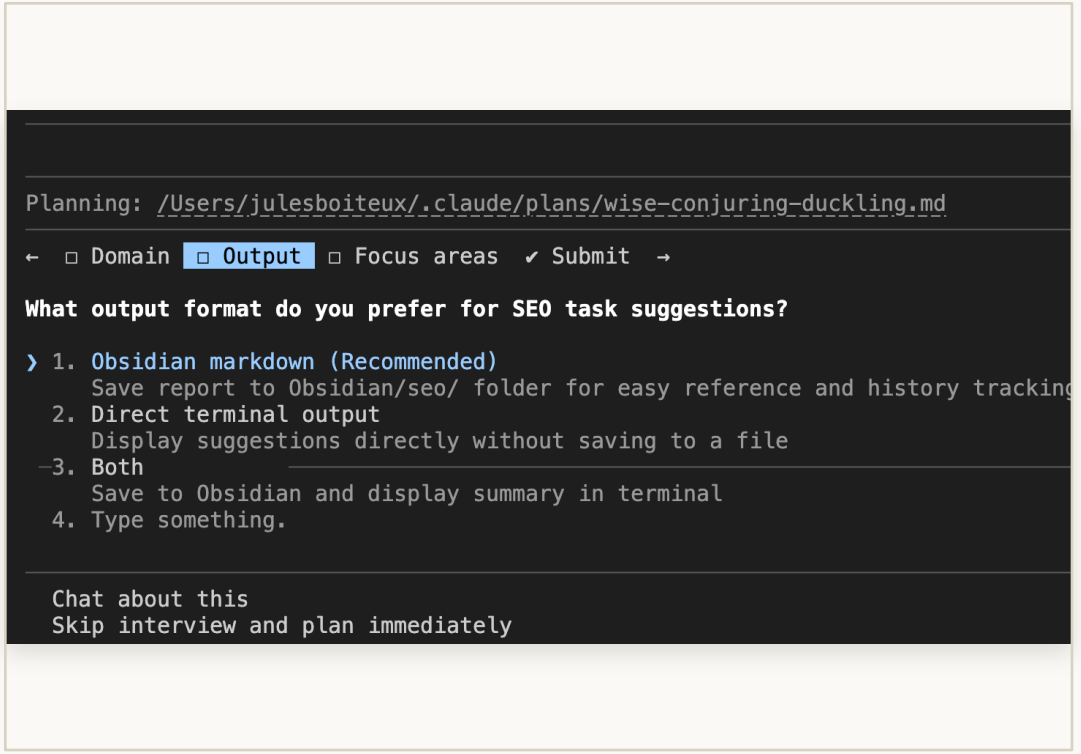
1. Describe the expected result and constraints (the spec).

2. Let Claude explore the code and write a plan.

3. Review and fix the plan: this is where you add the most value.

4. Back to auto-accept → run it in one go.

"The spec is the durable artifact; the model is just the compiler."
(Sean Grove, OpenAI, AI Engineer World's Fair)



In plan mode, Claude questions you first (format, scope, priorities), then writes a plan in a dedicated file that you approve before anything runs.

Put your effort into the plan. Precise plan = one-shot implementation; vague plan = round trips, polluted context.

Source: support.claude.com · power user tips (June 2026) · blog.tranie.org, part 05

The slash commands that change your day

Command	Effect	When to use it
/init	Creates a starter CLAUDE.md	First day on a repo
/clear	Resets the conversation completely	Between unrelated tasks
/compact	Summarizes old turns, frees context	Session getting long
/rewind	Goes back to an earlier point: code, conversation or both	After 2 failed fixes (or Esc ×2)
/resume	Picks up a past session	Continuing interrupted work
/context	Shows how full the context window is	When Claude "gets weird"
/diff	Every change made this session	Before every commit
/model · /effort	Switch model · reasoning depth	Depending on task difficulty
/btw	Side question without stopping current work	While Claude is running
/goal	Sets a goal Claude pursues across turns	Long, autonomous tasks
/batch	Fan-out of agents in worktrees for a migration	Big migrations (ch. 4)
/usage	Usage tracking (alias: /cost)	Regularly, without guilt

Type **/help** in session for the full list. The commands are the control layer, not advanced options.

Source: DataCamp: Claude Code slash commands (June 2026) · code.claude.com/docs

Context is RAM: manage it, or it manages you

Every session starts with an empty window that fills up: conversation, command output, files read. When it overloads, instructions drown and quality drops. That's "context rot".

THE FIVE HABITS

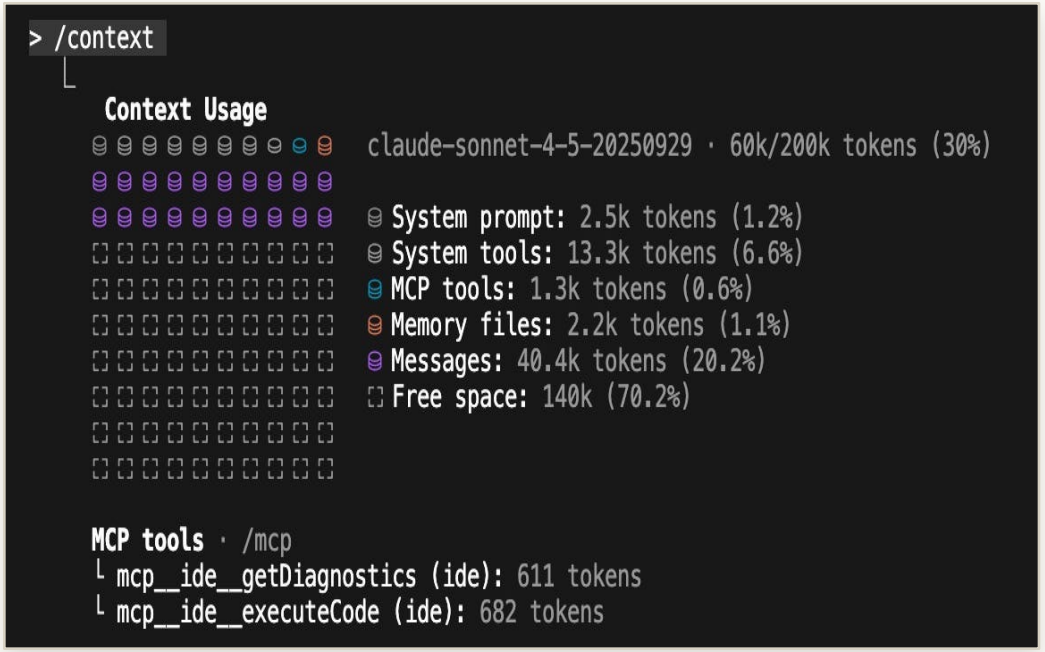
1 · One task = one session. The "kitchen sink session" (mixing it all) is enemy number one.

2 · /clear between topics with nothing in common. Continuity only helps if the thread holds.

3 · /compact when it gets long, or "Summarize" from the /rewind menu.

4 · After 2 failed fixes: stop. /rewind, then rewrite the original prompt. Piled-up failures poison the context more than they inform it.

5 · Stay under ~250k of effective context. Even with a 1M token window, reasoning quality drops off clearly past that (context rot). Watch **/context** and run /compact or start a new session before you reach that zone.



/context: the window occupancy grid, by category: system prompt, MCP tools, memory files, messages.

"Claude started well then got weird" = almost always a context problem, not a model one. In practice, the high-quality zone ends well before the window's official limit.

Source: support.claude.com · Anthropic, "Effective Context Engineering" (via blog.tranie.org, part 02)

Subagents and skills: from prompt to team

SUBAGENTS · SPECIALISTS WITH THEIR OWN CONTEXT

· Plain **.md in .claude/agents/**: name, description, allowed tools, model, color.

· Each subagent works in **its own context window**: search and exploration no longer pollute your main conversation.

· Use cases: squad PR review (one agent for logic, one security, one perf), exploring a big repo, batch tasks with **isolation: worktree**.

· **/agents** to create the first one: Claude writes the file for you.

SKILLS · YOUR PROCEDURES, REUSABLE

· A folder **.claude/skills/<name>/SKILL.md**: instructions + scripts, loaded on demand.

· **Progressive disclosure**: only the name and description cost context at startup; the body loads only when the task matches.

· Golden rule: done more than once a day → make it a skill. Share with the team via git.

· Open standard (agentskills.io): the same skill works in Codex, Cursor, Copilot…

HOW TO CHOOSE

Subagent = a role that works on its own with its own context. **Skill** = a procedure Claude follows in the current conversation. The two compose: a skill can say "use this subagent".

Audit community skills and agents like any other dependency: they're executable instructions.

Source: support.claude.com · power user tips · agentskills.io (via blog.tranie.org, part 03)

Hooks: guaranteed automation that doesn't rely on the model

A hook is a **shell command fired by the agent's lifecycle** (before or after a tool, on notification, at session end). It runs every time, unlike an instruction the model can forget.

Event	Hook example	What it guarantees
PostToolUse (EditIWrite)	prettier --write on the edited file	Code is always formatted, without thinking
PreToolUse (Bash)	Block rm -rf, git push --force, .env access	Dangerous commands never get through
PreToolUse (Edit)	Refuse any edit in /config/prod	Off-limits areas really stay off-limits
Notification	Sound or message when Claude waits for an answer	Essential from 2 parallel sessions on
Stop / SessionEnd	Log usage, archive the transcript	Traceability and costs in check

CONFIG · .claude/settings.json

```
"matcher": "EditIWrite",
"command": "prettier --write $FILE"
```

No need to learn the syntax: ask Claude to write the hook, it knows its own config format.

Source: support.claude.com · code.claude.com/docs · hooks · next page: MCP

MCP: write the tool once, use it in every agent

The **Model Context Protocol** standardizes how external tools connect: a server exposes tools (functions), resources (data) and prompts. Claude Code, Codex, Cursor and other clients all plug into it: the USB-C of agents. **/mcp** manages your servers.

MCP server	What the agent can do	Real usage example
GitHub	Open PRs, read issues, comment, review	"Open a PR summarizing the diff and link issue 214"
Postgres	SQL queries, schema inspection	"Check the DB for why this client has two subscriptions"
Slack	Read channels, post messages	"Summarize last night's #incidents alerts"
Playwright / browser	Click, fill forms, capture pages	"Run the app, reproduce the bug, show me the screenshot"
Filesystem / Drive	Read docs, specs, exports outside repo	"Follow the v3 spec in the shared folder"
Internal APIs (FastMCP)	Expose your business services in a few Python decorators	Every improvement benefits all your agents at once

NO MCP SERVER FOR YOUR TOOL? THE CLI IS ENOUGH

Claude Code calls any CLI through bash: **gh, docker, kubectl, aws, jira, vercel, stripe…** Most modern tools ship a rich CLI, no server to write.

Wrap it in a **skill** (key commands, examples, pitfalls, when not to use it): you get the equivalent of an MCP server, portable across every agent.

Source: modelcontextprotocol.io · code.claude.com/docs · mcp · blog.tranie.org, part 03

Save MCP for tools without a CLI, large data that needs paging, and integrations shared by the whole team.

Parallelize: the real productivity multiplier

1. Git worktrees: 3 to 5 parallel sessions

Each session in its own worktree, on its own branch (**claude --worktree**, native support). Name the worktrees, add shell aliases, color the terminal tabs and turn on notifications so you know when a Claude is waiting on you.

2. /batch: massive migrations

Claude interviews you about the migration, then launches as many agents in isolated worktrees as needed (dozens, or more). Each one tests its own changes and opens its own PR.

3. Agent Teams: coordinated sessions (experimental)

Several Claude Code sessions cooperate on a shared codebase: a lead splits the work, teammates execute. Powerful, but still experimental: keep control of the review.

THE ORCHESTRATION APPS

Conductor (conductor.build)
Mac, isolated workspaces, Claude Code / Codex / Cursor together, built-in review and merge. Slickest.

Superset (superset.sh)
Agent-agnostic, 10+ parallel agents, diff viewer, per-workspace ports, remote hosts. Free tier, Pro tiers.

T3 Code
Free and open source, parallel projects/threads, native worktrees, Linux from day one.

The condition: only parallelize independent, verifiable tasks. When generation multiplies, the bottleneck becomes your review capacity, not the model.

Path: one session → two worktrees → /batch.

Source: support.claude.com · conductor.build · superset.sh · code.claude.com/docs · agent teams

Headless: Claude works while you sleep

HEADLESS MODE · NO UI, SCRIPTABLE OUTPUT

```
› claude -p "fix the typecheck errors and commit" # -p = print, noninteractive
› claude -p "summarize the build errors" --output-format json # structured output
› claude -p "review this PR" --max-turns 20 # guardrail: bounded turn count
```

WHERE TO USE

· **Scripts, cron**: recurring tasks, reports, cleanups.

· **CI**: official GitHub Actions (PR review, issue resolution, generation triggered by repo events).

· **Agent SDK** (Python / TypeScript): the same agent loop inside your own programs.

THE GUARDRAILS

· **--dangerously-skip-permissions** removes every confirmation: only inside a container or a sandboxed VM, never with production access.

· Always bound it: --max-turns, limited permissions, logs kept.

· **Budget**: headless burns API credits, not your subscription.

⚠ **The over-automation warning:** every automated job is code nobody reads anymore. When the team can no longer explain what runs, or why, or how to fix it at 3 a.m., you don't have a productivity gain: you have debt. Readable logs, periodic human review of the pipelines, and always someone able to do the work by hand.

Source: code.claude.com/docs (Agent SDK, headless) · blog.tranie.org, parts 05-06

Your job changes: verifying becomes the core of the work

Give Claude a way to check its own work: tests, build, lint, screenshot comparison, run in the loop. It then catches its own errors before you do. Then read the diff like the PR of a very fast junior: confident, including when it's wrong.

Review posture	Typical code	What makes it safe
Full delegation: outputs reviewed, code never read	Scrapers, throwaway scripts, internal dashboards, one-off migrations	Schema-validated at the boundary, limited credentials, regenerable from spec rather than maintained
Boundary review: interface and tests read, body skimmed	Pipelines, CLIs, glue, test suites	The tests are the review: read them harder than anything else in the diff
Full review: line by line, as if you wrote it	Auth, payments, critical writes, concurrency, prod secrets	What checks can't see: no test proves the absence of an authorization flaw

Review level is a property of your verification, never of the model. A component only moves up the table when its controls improve (schema at the seam, restricted credentials, regenerable), never because "the last three deliveries were fine". The most profitable refactor of the agent era: moving code up this table.

Source: blog.tranie.org, part 05 (table 01) (July 2026) · support.claude.com

Beginner vs power user: seven habits that make the difference

The beginner's habit	The power user's habit
One catch-all session for the whole day	One task per session; /clear between topics
Patching five times in a row when it derails	Two failures → /rewind, rewrite the spec
"Build me an app": vague prompt, straight to code	Spec + acceptance criteria, plan mode first
No project memory: repeat it all every session	CLAUDE.md reworked via /init, committed to git
All in series, one window at a time	Worktrees + subagents on independent tasks
Opus everywhere, whatever the job	Sonnet/Haiku for volume, Opus for judgment
Accepting the diff unread	Tests in the loop + every diff reviewed

The seventh habit is the only non-negotiable.

Source: synthesis of support.claude.com & blog.tranie.org (July 2026)

The "lethal trifecta": when the agent becomes an attack surface

Access to private data	+	Untrusted content (web, email, docs)	+	Ability to act or exfiltrate

Prompt injection risk peaks here: text the agent reads becomes an instruction.

THE DEFENSES ARE ARCHITECTURAL, NOT PROMPTS

- **Least privilege**: tool permissions and allowlists, narrow-scope tokens.
- **Sandboxed execution**: container or VM as the agent gains autonomy.
- **Human approval** on any irreversible action (push, deploy, delete).
- **All fetched content is hostile by default**: web pages, emails, tickets, third-party docs.
- **Community skills and MCP servers = supply chain**: audit before you install.

"Prompt injection is still unsolved in 2026. Design your agents accordingly."

(after Simon Willison, Prompt Injection series)

Source: Simon Willison, "The lethal trifecta" (via blog.tranie.org, part 08) · OWASP Top 10 LLM

Your first seven days with Claude Code

Day	Action	You're done when…
D1	Install, log in (/login), ask a question about your own codebase	Claude explained a flow you barely knew
D2	/init, rework CLAUDE.md, commit it to git	Under 200 lines, your real conventions, your bans
D3	A real bug in plan mode, with tests as the criterion	Plan reviewed and fixed, one-shot run, tests green
D4	One task per session; /clear between; /context watched	No more catch-all sessions in your day
D5	Build a subagent (/agents) or a skill for a recurring task	It runs on a real task, not a toy example
D6	Two worktrees in parallel on two independent tasks	Two PRs shipped without the threads mixing
D7	A headless job (claude -p) in a sandboxed environment	The JSON output feeds a script, and you logged it

After seven days, you're no longer a Claude Code user: you have a workflow.

Plan from the "30-day ramp" on blog.tranie.org (fig. 8), trimmed for Claude Code

THE HABIT TO KEEP

Specify → Delegate → Verify

The loop that separates pros from tourists, with any model, in any tool.

GOING FURTHER

Official documentation
code.claude.com/docs

Power user tips, Claude Code team
support.claude.com

Simon Willison's Weblog, the field's daily read
simonwillison.net

Claude Code: Best Practices, Anthropic
anthropic.com/engineering

AI for Software Engineers: Field Guide
blog.tranie.org (July 2026)

Artificial Analysis, independent benchmarks
artificialanalysis.ai

Good first session. Read every diff, and retest your models monthly.