The Playbook on Two Pages

From typing code to directing it. How to steer Claude Code on real software, from your first session to your whole team.

v2.0 · November 2026

Daniel Wanja

The core workflow: **Explore** → **Plan** → **Build in small steps** → **Verify each one** → **Commit**

A check that can fail
The typecheck, the tests that cover the change, and for UI the running app.

Evidence, not claims
Claude shows the command and its output. "The tests pass" doesn't count.

A check that fits the risk
A shared rule needs unit and API tests; an access rule needs denial tests.

You read the diff
Claude writes the code. A human owns the merge.

LOOP Understand the loop you're steering

Ch 1

Give Claude a clear goal, the right context, and a check it must pass before it's done.

- **Your first session edits nothing.** Ask how the code works before you change it.
- **One task per context.** `/clear` between tasks; the old one stays in `/resume`. `/context` shows what's using space.
- **Start on the Opus tier.** Turn `/effort` up or down before you change models; `/usage` shows where the spend went.

Permission mode	Runs without asking	Good for
Manual (`default`)	Reads only	New repos, risky work
`acceptEdits`	Reads, edits, common file commands	Edits you'll review
`plan`	Reads; doesn't edit source files	Exploring, planning
`auto`	Most actions; a classifier checks risky ones	Daily work you verify
`dontAsk`	Pre-approved tools; the rest is denied	Locked-down scripts, CI
`bypassPermissions`	All but deny and ask rules	Disposable containers only

Symptom	Likely cause	Do this
Big, wandering diff	Vague goal	Name files and a finish line
Plausible but wrong rule	Missing context	Put the rule in `CLAUDE.md`
"Done," but it isn't	No feedback signal	Give it a check to run
Same mistake after fixes	Polluted context	`/clear`, rewrite the prompt

WORKFLOW Explore, plan, build, verify, commit

Ch 2

Explore, plan, then build in small steps that each end with a check you've seen pass.

- **Be specific:** name the files, the limits, the edge cases and the command that proves it's done. Hand over screenshots and errors.
- **Plan mode** for unfamiliar code (`Shift+Tab`). Press `Ctrl+G` to edit the plan before you approve it.
- **Fuzzy ask?** Let Claude interview you and write the spec. **One-sentence diff?** Skip the plan.
- **Watch the first minute.** `Esc` interrupts; `Esc Esc` or `/rewind` restores code, conversation or both.
- **Two failed corrections,** then `/clear` and a better prompt.
- **Commit every green step;** git is the real undo. Let Claude write the commit and PR, then read both.

CONTEXT Tell Claude what it can't see

Ch 3

Give Claude the few things it can't learn from the code, in the place where each one belongs.

Kind of context	Where it goes	When it loads
Facts every task needs	`CLAUDE.md`	Every session
Rules for one area	`.claude/rules/` + `paths`	When Claude touches a match
Your own preferences	`CLAUDE.local.md`	Every session, just you
A repeatable procedure	A Skill	Description always; body on use
Parallel or noisy side work	A subagent	Own window; a summary returns
Live system state	An MCP server	Names up front; output on use
Must happen every time	A hook	Runs as code, not context

- **`CLAUDE.md`:** commands (including one test file), rules, house libraries, gotchas. Under 200 lines. `/init`, then cut what Claude can read in the code.
- **Long work:** write progress to a file so a fresh session can pick it up.

CHECK Give Claude a check it can run

Ch 4

Lock the tests, then let Claude make them pass.

- **A `Stop` hook** runs your fastest check, so "the checks pass" is a gate, not a request.
- **Bug?** Reproduce it with a failing test first. A test you haven't seen fail can't be trusted.
- **Test at the lowest layer that can catch it:** unit → API → access rules → browser.
- **`/verify`** on app changes: Claude runs the app and looks at the result.
- **Smells:** asserts nothing · mocks the subject · flaky waits · coverage theater · a quiet rewrite of an expected value. Read test diffs first.

ENFORCE Enforce what can't have exceptions

Ch 5

If a rule can't have exceptions, make Claude Code enforce it.

- **`CLAUDE.md` asks;** deny rules, hooks and managed settings enforce.
- **Committed settings:** allow the loop, ask before push, deny secrets and deploys. Deny beats ask beats allow.
- **Hooks** for non-negotiables: a `PreToolUse` script that exits 2 blocks the action.
- **A deny rule is a guardrail; the sandbox is the wall.** `/sandbox`; strict mode sets `allowUnsandboxedCommands: false`.
- **Keep the enforced set small,** and let Claude weigh the rest.

Independent; not affiliated with Anthropic.

The Playbooks and the Team

Five playbooks for the work developers bring to Claude Code, then review and rollout. Each chapter has a full checklist; Appendix A collects them.

v2.0 · November 2026

Daniel Wanja

Ship a feature as one slice — Ch 6

Use when one user outcome crosses layers: data, access rules, shared logic, API, client, screen, tests.

1. Write the slice spec before you open a session.
2. Turn it into an ordered plan, and check the plan four ways.
3. Build bottom-up, one layer per step, with a check at each.
4. Changing a rule? Map every place it lives; let the compiler find the rest.
5. Ship a rule change dark; backfill with a human at the controls.
6. Finish with a fresh review and a PR that tells the story.

The UI explains; the server enforces.

Integrate an API you don't own — Ch 7

Use when your code calls an outside API: payments, rates, maps, a partner.

1. Put the provider's contract in the repo.
2. Have a subagent research the library; record the choice in `CLAUDE.md`.
3. Tell Claude what the runtime hides.
4. Test against recorded reality, then break the configuration on purpose.
5. Enforce the choice in review and lint.

Let a library do the plumbing; hand-write only your rules.

Design the contract, then build to it — Ch 8

Use when someone else calls your endpoint: another team, a mobile app, a partner or an agent.

1. Pick one source of truth, and generate the rest.
2. Design the contract in plan mode; commit it before any route.
3. Review it as an outsider would.
4. Build each endpoint with a Skill; a drift test holds the line.
5. Test it with an agent; expose only the smallest tool.

Agents read your descriptions literally: write them for a new hire.

Modernize legacy code without changing what it does — Ch 9

Use when the old app's behavior is the spec: a framework, language or stack migration.

1. Map the old app; spend your own hours on the map.
2. Build the judge (parity cases) before you translate a line.
3. Decide the target once; show Claude one finished screen.
4. Convert in slices, not layers.
5. Prove parity on behavior, screens and data, then cut over.

Reconcile counts and money to the cent before you switch.

Ship the whole pie — Ch 10

Use when an oracle can say "done" (the old app, or the designs) and you could throw the result away.

1. Write a brief: end state, validation, scope, guardrails, a cap, a sign-off.
2. Seal the run in a worktree, with permissions for unattended work.
3. Read its first turn before the long part starts.
4. Set the `/goal`, and cap the run from outside.
5. Grade it with a judge that didn't build it.
6. Land the result as slices.

The run is a draft: rerun rather than patch a bad foundation.

REVIEW Review what Claude writes — Ch 11

- **The writer never grades its own work.** `/code-review` hunts bugs; a read-only reviewer subagent checks the spec.
- **Ask for correctness gaps, skip style,** and judge each finding yourself.
- **Look for** files out of scope, invented APIs, swallowed errors, duplicated logic, new dependencies, looser access checks.
- **Every input can carry instructions.** A human merges; branch protection has the last word.

TEAM Scale it to the team and CI — Ch 12

- **Commit the setup:** `CLAUDE.md`, settings, rules, Skills, agents, hooks, `.mcp.json`. Review it like code.
- **In CI:** `claude --bare -p`, narrow `--allowedTools`, `dontAsk`, `--max-turns`. Never pair untrusted input with a token that can push.
- **Parallel work** in worktrees: `claude --worktree name`.
- **Measure** cycle time, rework and escaped defects, not lines of code.

COMMANDS USED MOST

`/clear` new task, fresh context	`/context` what's loaded
`/rewind` restore code or chat	`/effort` how hard it thinks
`/usage` where the spend went	`/code-review` fresh bug hunt

SET UP ONCE

- Commands in `CLAUDE.md`, incl. one test file
- Area rules in `.claude/rules/`; procedures in Skills
- Committed allow / ask / deny settings
- A `Stop` hook with your fastest check
- Sandbox on for anything sensitive

EVERY TASK

- `/clear` between tasks
- Plan for unfamiliar code
- Name files, limits, the proof of done
- Bug? Failing test first
- Small steps; commit each green one

BEFORE MERGE

- Fresh-context review against the spec
- Read test diffs first
- Evidence, not claims
- A human merges

UNATTENDED

- Done defined, a self-check, limited reach
- A turn or budget cap on `-p`
- A person signs off

Independent; not affiliated with Anthropic.