PATTERNS, NOT FEATURES — COPY, PASTE, SHIP

The Playbook.

Claude Code, twenty real moves.

FOUR CHAPTERS · ONE PRINCIPLE EACH

I • The terminal IS the IDE.
5 patterns · #1–#5

II • Tests are the new prompt.
3 patterns · #6–#8

III • Context is a craft.
4 patterns · #9–#12

IV • Leverage compounds.
1 pattern + flywheel · #13–#20

Every slide has a command you can run before this talk ends.

Sister deck to "From Zero to Hero"
That one made the argument. This one shows the work.

Real syntax, current as of 2026
Cited inline, every claim

Each pattern slide answers four questions — in this order.

01 TOP OF SLIDE

When to reach for it.

One sentence. The trigger that makes this pattern the right answer.

02 CENTER · THE MOVE

The exact command.

Copy-paste-ready — slash command, hook config, or shell pipeline.

03 SIDE PANEL

Why it works.

The mechanism, in two lines. Not a sales pitch — the reason.

04 BOTTOM RAIL

The trap to avoid.

The mistake everyone makes the first time. Skip the tax by reading it now.

If you have 5 minutes, read the orange box on each slide. That's the entire deck, distilled.

I

THE ARGUMENT OF THIS CHAPTER

The terminal IS the IDE.

Not nostalgia — architecture. Pipes, sessions, and headless mode make the terminal the only surface where AI can compose with the rest of your tools.

WHAT YOU'LL LEARN TO DO, IN FIVE MOVES

#1 bootstrap · #2 plan-first · #3 /clear discipline · #4 /rewind · #5 pipe

#01 The 60-second project bootstrap.

WHEN —

Opening Claude Code in an unfamiliar repo for the first time.

~/work/unknown-repo · first session — THE MOVE

```
$ claude

> /init

# Claude reads the repo, then writes CLAUDE.md for you.
# Read it. Edit the wrong parts. Commit it.

> read CLAUDE.md and tell me what's missing
```

WHY IT WORKS

/init reads the repo end-to-end — package files, READMEs, top-level dirs — and drafts the memory file. Faster than you typing it.

THE TRAP

Don't commit the auto-generated CLAUDE.md as-is. It always invents two rules your team doesn't follow. **Read it.**

> *First 60 seconds in a new repo: /init → read → commit. Skip this and you'll re-explain the codebase every session for a week.*

Source: Anthropic, code.claude.com/docs/commands — `/init` built-in command (2026); Anthropic Engineering, "Best practices for Claude Code," Apr 2025.

#02 Plan first. Always.

WHEN —

Any change that touches more than one file — which is most changes.

your repo · inside a session — THE MOVE

```
# press Shift+Tab to enter plan-only mode
# Claude can read, search, reason — but can't edit, run, or commit

> Add a rate-limit middleware to the /payments endpoint.
  Don't write code yet. Give me the file list,
  the function signatures, and the test cases.

# review the plan in chat. cheaper than reviewing the code.
> approve ↵
```

WHY IT WORKS

Plan-mode is **read-only** by design. You edit prose, not 600 lines of code. The cost of correcting course drops 50×.

THE TRAP

Approving a plan that names files but no *tests*. If the plan can't say "this test will pass," send it back.

> *A plan you approved in 60 seconds is the cheapest insurance policy in software.*

Source: code.claude.com/docs (Plan Mode); ClaudeDirectory, "Claude Code Plan Mode (2026): When to Use It"; Hidekazu Konishi, Features Reference 2026.

#03 The /clear discipline.

WHEN —

Switching tasks. The agent starts confusing the new task with the previous one.

inside a long session — TWO MOVES

```
# switching to an unrelated task → wipe everything
> /clear

# same task, but context is bloated → compress with intent
> /compact keep the API contract decisions and the failing test

# Anthropic's own rule of thumb: /clear between tickets,
# /compact within one.
```

WHY IT WORKS

Context isn't free attention. Old log lines, abandoned plans, dead-end greps — they all bias the next answer.

THE TRAP

Bare **/compact** with no instructions. It throws away the wrong half. Always tell it what to keep.

> *Most "the agent got dumber" complaints are old context, not a worse model.* **/clear** *fixes it 80% of the time.*

Source: Anthropic, code.claude.com/docs/commands (built-in slash commands, 2026); Gigi Sayfan, "Claude Code Deep Dive — Slash Commands," Medium.

#04 Double-tap Esc to rewind.

WHEN —

Three turns ago the agent went the wrong way and you didn't catch it.

keyboard shortcut · built-in

THE MOVE

```
# press Esc twice → pick a previous checkpoint
[Esc][Esc]

  ◯ checkpoint — restore conversation only
  ◯ checkpoint — restore code only
  • checkpoint — restore both     ← usually what you want

# or: type /rewind to pick by message
> /rewind
```

WHY IT WORKS

Checkpoints are **auto-saved before every edit**. You always have a clean point to fall back to — no git stash needed.

THE TRAP

Rewinding without saying *why* in the next message. The agent repeats the same wrong turn.

> **[Esc][Esc]** *is the cheapest undo in software. Use it instead of fighting the agent for ten more turns.*

Source: Anthropic, "Enabling Claude Code to work more autonomously," Sept 29, 2025 (checkpoints); MindStudio, "Claude Code /rewind Command Rollback," 2026.

#05 Pipe stdin → Claude → stdout.

WHEN —

One-off transforms, log triage, CI steps. No interactive session needed.

headless mode · claude -p

UNIX-COMPOSABLE

```
# pipe a stack trace straight in
$ kubectl logs api-pod-7f | claude -p "what crashed and why?"

# transform CSV → markdown table
$ cat data.csv | claude -p "to markdown table" > out.md

# a CI step that summarises a failing run for Slack
$ pytest -v | claude -p "failures only, one line each" \
    | slackcat #builds
```

WHY IT WORKS

-p = print mode. One prompt, exit. Unix philosophy: small programs composed via pipes — Claude is now one of them.

THE TRAP

Piping more than ~7K characters of stdin can return empty output. For long logs, write to a temp file and pass the path.

> *The moment Claude works with | and >, it stops being an app and starts being a tool.*

Source: Anthropic, code.claude.com/docs (headless mode + -p flag, 2026); Repovive, "Claude as a CLI Tool"; GitHub anthropics/claude-code issue #7263 (stdin size).

II

THE ARGUMENT OF THIS CHAPTER

Tests are the new prompt.

A failing test is the most precise specification you can hand an agent. Write the test, then say "make this pass." The code becomes a byproduct.

THREE PATTERNS THAT TURN TDD INTO A GUARD RAIL

#6 red→green→Claude · #7 property tests · #8 make-it-crash

#06 Red → Claude → Green.

WHEN —

Any new function. The spec exists in your head — turn it into an executable test.

tests/test_billing.py · new feature — THE LOOP

```
# 1. YOU write the test (the spec)
def test_discount_caps_at_50_percent():
    assert apply_discount(100, 0.8) == 50

# 2. CLAUDE writes the code
> implement apply_discount() to pass tests/test_billing.py
> don't change the test. iterate until it passes.

# 3. CLAUDE reports back
✓ 1 passed in 0.04s
```

WHY IT WORKS

The test is an **oracle**. Claude can iterate against it without you in the loop — right or wrong is decidable.

THE TRAP

Forgetting to say **"don't change the test."** Otherwise Claude will sometimes "fix" the spec instead.

> *You wrote the easy part — the assertion. The agent writes the boring part — the implementation that satisfies it.*

Source: Anthropic Engineering, "Best practices for Claude Code" (TDD section, Apr 2025); RanTheBuilder, "Claude Code Best Practices: Lessons From Real Projects."

#07 Property tests for AI-written code.

WHEN —

Parsers, encoders, anything with a "round-trip" invariant. The agent loves to fake these.

tests/test_roundtrip.py · hypothesis — THE MOVE

```
from hypothesis import given, strategies as st

# YOU write the invariant — not 50 specific examples
@given(st.dictionaries(st.text(), st.integers()))
def test_json_roundtrip(d):
    assert json.loads(json.dumps(d)) == d

# Hypothesis generates 100s of inputs. Claude can't cheat by
# hard-coding "return the example value" — there is no example.
```

test_roundtrip.py

WHY IT WORKS

Hypothesis generates adversarial inputs. The agent's "looks-right" code dies on the first weird unicode string.

THE TRAP

Property tests are slow. Run them in CI, not in the inner Claude loop — or the agent will time out.

> *Example-based tests check what you remembered. Property tests check what you forgot — which is what the agent missed.*

Source: Hypothesis docs (hypothesis.readthedocs.io); MacIver, "Property-based testing in Python," 2024 — pattern unchanged in 2026.

#08 The "make it crash" hook.

WHEN —

You want "all tests pass" to actually mean it — before the agent declares the task done.

.claude/settings.json · Stop hook — BLOCKS "DONE"

```
// Fires when Claude says "I'm done." If pytest fails, hook
// exits 2 → Claude is forced to keep working.
"hooks": {
  "Stop": [{
    "hooks": [{
      "type": "command",
      "command": "pytest -q || exit 2"
    }]
  }]
}
```

WHY IT WORKS

A **Stop hook** with exit code 2 is the one signal Claude must obey. "All tests pass" becomes deterministic, not promised.

THE TRAP

A 10-minute test suite as a Stop hook. Use a fast smoke subset (under 30s) — full suite in CI.

> *"Done" used to be a vibe. Now it's an exit code.*

Source: Claudefa.st, "Hooks Guide" (Stop hook + exit code 2 behaviour, 2026); Anthropic, code.claude.com/docs (Hooks reference).

III

THE ARGUMENT OF THIS CHAPTER

Context engineering is a craft.

CLAUDE.md, subagents, slash commands, MCP — none of these are tricks. They're the four primitives. Pick the right one for each problem.

FOUR PRIMITIVES, ONE PER PATTERN

#9 CLAUDE.md · #10 subagent · #11 /commands · #12 MCP

#09 The 80/20 CLAUDE.md.

WHEN —

Every project that's outlived one feature. The auto-generated CLAUDE.md isn't enough.

./CLAUDE.md · under 80 lines — THE 4 SECTIONS

```
# Stack      — one line per axis. languages, frameworks, deploy target.

## Commands — make targets. Each line: command, runtime, what it does.
- make test    # ~90s, full suite, requires Postgres up

## House rules — ≤5 bullets. The conventions that diff from the language default.
- Money uses decimal.Decimal, never float.

## Ask first    — file paths that touch prod. The dragons.
- core/billing.py, infra/deploy.sh
```

WHY IT WORKS

Loaded into **every** session. 80 lines you write once save 30 lines of re-prompting per session, forever.

THE TRAP

The 800-line CLAUDE.md. Past ~150 lines, recall drops. Move details to `.claude/rules/*.md`.

> *Four headings, eighty lines. CLAUDE.md is a router, not an encyclopedia.*

Source: r/ClaudeAI, "How I structure Claude Code projects (CLAUDE.md, Skills, MCP)," 2026; Bijit Mishra, "The Complete Guide to CLAUDE.md," Medium 2026.

#10 Dispatch a subagent for noisy reads.

WHEN —

The next task needs noisy reading — grep 40 files, audit a 600-line diff. Don't pollute the main session.

.claude/agents/code-reviewer.md — DEFINE ONCE

```
---
name: code-reviewer
description: Audits a diff for security & missing tests.
tools: Read, Grep, Bash(git diff:*)
---

You are a senior reviewer. For each changed file:
  1. Read the diff context.
  2. Flag untyped user input.
  3. Confirm tests cover new branches
```

WHY IT WORKS

The subagent runs in its **own context window**. Returns a one-paragraph summary. Main session stays clean.

THE TRAP

Treating subagents as a *speed* trick. It's not. It's a memory trick — one isolated context per noisy job.

> *Your main context is a budget. Subagents are how you stop spending it on log-grepping.*

Source: Anthropic, code.claude.com/docs (Subagents reference, 2026); r/ClaudeAI, "Subagents aren't a speed trick — they're a memory trick," 2026.

#11 Bottle a workflow into a slash command.

WHEN —

You've typed the same 4-line prompt three times. Stop typing it. Make it a command.

.claude/commands/pr-review.md — NOW: /PR-REVIEW

```
# filename = command name. Body = the prompt.

Review the staged diff for PR #$ARGUMENTS:

  1. Run git diff origin/main...HEAD.
  2. Flag: untyped inputs, missing tests, console.log left in.
  3. Output a markdown checklist I can paste into GitHub.

# Then in the session:
> /pr-review 1284
```

WHY IT WORKS

Commands are **versioned**. The team gets the same workflow you do, by checking out the repo.

THE TRAP

A folder of 40 commands. You'll forget which exists. Cap at 8—10 per project; rename if needed.

> *The third time you type a prompt by hand, you owe yourself a file.*

Source: Anthropic, code.claude.com/docs/agent-sdk/slash-commands (custom commands, 2026); CodeSignal, "Creating Reusable Shortcuts with Custom Slash Commands."

#12 The four-MCP starter stack.

WHEN —

Day one of any new project. These four cover 90% of what teams need.

01 · GITHUB

PRs, issues, code search.

"Open issue #42 and find a related PR" works in one turn.

02 · CONTEXT7

Current library docs.

Kills "hallucinated API" errors. Pulls the real signature, today.

03 · PLAYWRIGHT

Drive a real browser.

The agent can click, screenshot, and verify the UI it just changed.

04 · POSTGRES

Read-only SQL.

Queries dev DB to debug "why is row 7 wrong." Read-only by design.

install all four in one shell session — ONE-TIME SETUP

```
$ claude mcp add github     -- npx -y @modelcontextprotocol/server-github
$ claude mcp add context7   -- npx -y @upstash/context7-mcp
$ claude mcp add playwright -- npx -y @playwright/mcp
$ claude mcp add postgres   -- npx -y @modelcontextprotocol/server-postgres $READ_ONLY_URL
```

> **Trap:** *writable database MCP on a prod connection string. The agent will run DELETE. Read-only URLs, always.*

Source: EvoMap, "Best MCP Servers for Claude Code 2026" (top picks); modelcontextprotocol/servers GitHub registry.

IV

THE ARGUMENT OF THIS CHAPTER

Leverage compounds.

Every hook, command, and CLAUDE.md line you write today silently accelerates every session for the next year. The flywheel is real.

THE LAST MOVE + THE SEVEN SLIDES THAT PROVE IT

#13 hooks — the rule code enforces · #14–#20 the flywheel + closing

#13 Hooks — rules in code, not prose.

WHEN —

Any rule that's non-negotiable. "Always format." "Never read .env." "Always run tests before done."

.claude/settings.json · three hooks worth shipping — NON-OPTIONAL

```
"hooks": {
  "PreToolUse": [{                    // block reading secrets
    "matcher": "Read",
    "hooks": [{ "command": "grep -q '\\.env' <<< $FILE && exit 2" }]
  }],
  "PostToolUse": [{                   // auto-format on edit
    "matcher": "Edit|Write",
    "hooks": [{ "command": "ruff format $FILE" }]
  }],
  "Stop": [{                           // "done" means tests pass
    "hooks": [{ "command": "pytest -q || exit 2" }]
```

WHY IT WORKS

Exit code 2 is the one signal Claude has to obey. Prose in CLAUDE.md is advice; a hook is a contract.

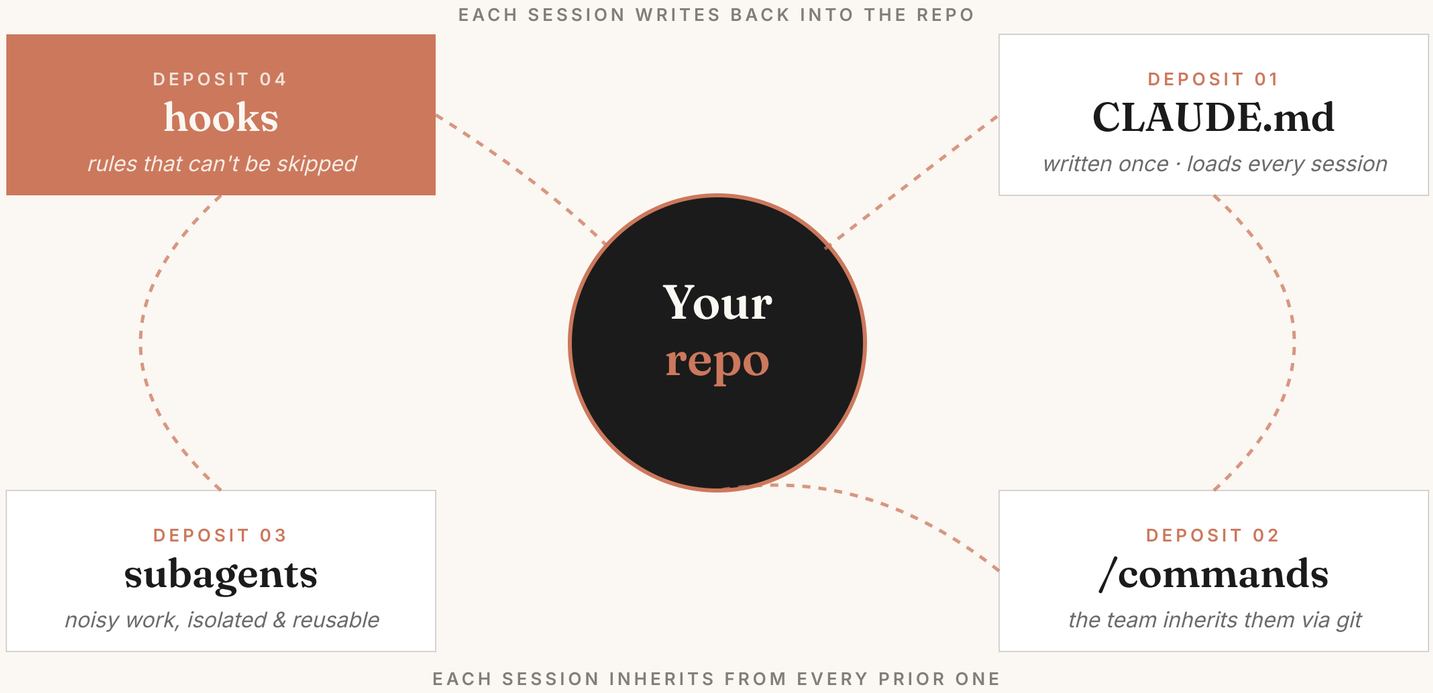
THE TRAP

A hook that's noisy in stdout. Claude reads it. Keep them *silent* unless they fail.

> *A rule the model might forget is a wish. A hook is a contract.*

Source: Anthropic, code.claude.com/docs (Hooks reference, 2026); Claudefa.st, "Complete Guide to All 12 Lifecycle Events"; MorphLlm, "Claude Code Hooks (2026): Block Reading .env."

Each pattern is a deposit. The flywheel pays interest.



THE CLOSING ARGUMENT

Patterns are deposits.

The four primitives compound. A hook you write tonight runs in every session you have for the next twelve months — for you, and for whoever clones the repo.

Question to take home: *which of the 20 patterns are you committing to your repo before next Monday?*