The "Fleet Commander" Strategy: How Boris Cherny Uses Claude Code

Based on the January 2026 Thread by the Creator of Claude Code

In early 2026, Boris Cherny (Staff Engineer at Anthropic and creator of Claude Code) revealed his personal workflow for the tool. Despite building the agent, his usage isn't defined by complex, hidden scripts, but rather by a rigorous operational philosophy.

Boris treats Claude Code not as a chat bot, but as **reliable infrastructure**. By shifting from "writing code" to "managing parallel agents," he achieves 100% AI-written production commits.

Here are the five pillars of his strategy.

1. The Fleet Commander Approach (Parallelism)

The biggest bottleneck in AI coding isn't the AI's speed—it's the human waiting for the output. Boris solves this by running wide.

- **5+ Terminal Sessions:** He keeps 5 active tabs open in his terminal (numbered 1–5), each running a separate instance of Claude Code.
- **Browser & Mobile Extensions:** He supplements this with 5–10 sessions in the browser (`claude.ai/code`) and manages quick checks via the iOS app.
- **Context Switching:** He uses iTerm2 system notifications to alert him when a specific tab finishes a task.
- **The Workflow:** While Tab 1 runs a test suite and Tab 2 refactors a legacy module, he is actively planning a feature in Tab 3. He essentially acts as a localized Engineering Manager for 5-10 "junior" developers.

2. Plan Mode First (Shift+Tab)

Boris rarely jumps straight into coding for complex tasks. He utilizes **Plan Mode** as a strict prerequisite.

- **The "One-Shot" Goal:** The objective is to refine the plan so thoroughly that the execution phase is a single, successful pass.
- **Iterative Planning:** He converses with Claude to refine the architecture and requirements *before* a single line of code is written.
- **Auto-Accept:** Once the plan is solid, he switches to execution mode (often with auto-accept enabled), allowing Claude to implement the specification rapidly.

3. `CLAUDE.md`: The Shared Memory

To prevent the AI from making the same mistake twice, Boris utilizes a `CLAUDE.md` file in the root of his repositories. This acts as a persistent memory and style guide.

- **The "Do Not Repeat" Ledger:** Every time Claude makes a stylistic error or uses a forbidden pattern, Boris updates this file.
- **Examples of Rules:**
  - "Prefer `type` over `interface`."
  - "Never use `enum`; use union types."
  - "Use `vitest` instead of `jest`."
- **Compound Interest:** This file is shared team-wide. Every correction makes the entire "fleet" of agents smarter for every engineer on the project permanently.

4. Verification Loops (The Critical Step)

Boris cites this as the most important part of the workflow. He never assumes the code works; he forces the agent to prove it.

- **Self-Correction:** He instructs Claude to write a test, run the test, fail, fix the code, and pass the test—all without human intervention.
- **Frontend Validation:** For UI work, he forces the agent to open a browser or run end-to-end tests to verify visual components.
- **Impact:** This loop reportedly improves code quality by **2–3x**.

5. Tooling & Infrastructure (Opus 4.5)

He keeps the setup surprisingly "vanilla" but leverages the highest-end compute available.

- **Model:** He exclusively uses **Claude 3.5 Opus (or the 2026 equivalent, Opus 4.5)** with "Thinking Mode" enabled. He prioritizes reasoning capability over raw speed, relying on parallelism to offset the latency.
- **Slash Commands:** He builds custom commands for inner-loop workflows (e.g., `/verify` to run a standard test suite, `/simplify` to refactor complexity).
- **Post-Tool Hooks:** He uses automated hooks to run formatters (Prettier/ESLint) *after* the agent generates code, ensuring the AI doesn't waste tokens on formatting nuances.

Summary: How to Adopt This Today

If you want to replicate this productivity boost, start with these three steps:

1. **Create a `CLAUDE.md` file** in your repo immediately. Write down your top 5 non-negotiable coding standards.
2. **Stop watching the cursor.** Open a second terminal tab. When the first agent is working, start planning the next task in the second tab.
3. **Refine the Plan.** Don't let the agent code until you have agreed on *how* it will code.