THE ULTIMATE

CLAUDE CODE

MASTERY GUIDE

9 Multiplier Hacks • Multimodal Mastery • Replicate Anything Framework

Your Blueprint to Building an AI Factory

By Mark Kashef

Founder, Prompt Advisers

ACT 1: THE 9 MULTIPLIER HACKS

Intermediate and advanced techniques to 10x your output without 10x effort.

Hack #1: The Global Brain

Use the hashtag command to persist instructions to your CLAUDE.md file—your project's command center.

- Type #your instruction to add rules to project memory
- Rules persist across ALL sessions in the project
- Perfect for style guidelines, workflow preferences, and recurring patterns

> 💡 **Pro Tip**
> Use this for instructions like "always create 3 drafts" or "never use periods in casual emails" so you don't repeat yourself.

Hack #2: Plugin Marketplace

Plugins bundle agent instructions + MCP servers into shareable packages.

- Use /plugins to browse and install from marketplaces
- Add Anthropic's official repo: github.com/anthropic/claudecode
- Key plugins: feature-dev, front-end-design, code-review
- Share entire environments with your team via plugin bundles

> 💡 **Verification Trick**
> Check settings.local.json to confirm plugins are ACTUALLY enabled. If enabledPlugins doesn't show true, the plugin isn't really installed.

Hack #3: The MCP Shortcut

Skip the /mcp command frustrations—create an MCP.json file directly.

- Create MCP.json in your project root
- Paste the MCP server configuration (find docs via Perplexity or docs)
- Save and restart session—MCP is instantly available
- Example: Chrome MCP for visual UI inspection

Hack #4: The CORE Agent Rule

Don't create agents as separate roles—create agents as different parts of ONE task.

- Traditional approach: UI Designer, Code Reviewer, Security Auditor (often misaligned)
- Better approach: Break a single goal into sub-tasks handled by specialized prompts
- Example: "Improve front-end" becomes Color Optimizer + UX Optimizer + Feature Expander

- All agents share aligned incentives = better, cohesive results

> 💡 **Key Insight**
> Agents working toward the same goal (one unified army) massively outperform agents fighting separate battles.

Hack #5: Rewind (/re)

Claude Code's version of Git commits—hop back to any checkpoint.

- Use /re to see all available checkpoints
- Roll back to before a broken feature implementation
- Perfect for: 99% right execution done 100% wrong
- Redo with newfound knowledge without losing progress

Hack #6: Phased Planning

Use Plan Mode (/plan) to create milestone-based phases with a tracking file.

- Request phases with logical cutoff points
- Create a markdown tracker file with checkboxes for each phase
- Clear context window between phases (/clear)
- Check /context to monitor remaining capacity (aim for 80%+ free before new phases)

> 💡 **Anti-Slop Strategy**
> Clearing context between phases prevents AI hallucinations that occur when context windows near capacity. Extra step = massive time savings.

Hack #7: The "Are You Sure" Hack

Add a global instruction forcing Claude to push back on unclear commands.

- Add to CLAUDE.md: "If I don't use plan mode for non-trivial changes, ask clarifying questions"
- Prevents destructive actions from vague instructions
- Acts as plan mode insurance when you forget
- Test it: "nuke the whole app" should trigger clarification, not destruction

Hack #8: Context & Compact

Monitor and manage your context window proactively.

- Use /context to check remaining capacity
- /compact creates a TLDR summary and clears history
- Works best WITH a tracker file (artifact to reference)
- Remove unused MCP servers to free up space
- Keep CLAUDE.md lean—bloated files eat context every session

Hack #9: The Project Brain (Codebase Inventory)

For large codebases, create an inventory file mapping files to functionalities.

- Ask Claude to create a codebase-inventory.md file
- Document what each folder/file does in plain English
- Include dependencies and references

- Use instead of /init for massive codebases—saves context
- Point Claude to specific sections: "refer to inventory, only change X domain"

ACT 2: MULTIMODAL MASTERY

Leverage Claude Code Web and mobile for parallel, async workflows.

Claude Code Web

- Access at claude.ai → Code section
- Import existing repos or create new ones
- Run multiple agents in parallel
- Use "Teleport" to port work back to CLI

Best Uses for Claude Code Web

- Planning and brainstorming
- Surface-level aesthetic/UI edits
- Security vulnerability scans
- Code review and analysis
- PRD generation

The Cost Hierarchy

Be strategic with your tokens:

- Claude.ai Chat → Free conceptual Q&A, brainstorming
- Claude Code Web → Low-cost planning, parallel async tasks
- CLI → Reserved for building and heavy iteration

> 💡 **Mobile Power**
> Use downtime on mobile to queue planning tasks—have agents ideate while your computer does the heavy building.

ACT 3: REPLICATE ANYTHING FRAMEWORK

Clone Claude.ai's front-end capabilities and build from API docs in one shot.

Blueprint #1: Clone Claude's Skills

Replicate Claude.ai's file creation abilities (DOCX, PPTX, XLSX) in Claude Code.

- Use Claude.ai to generate a file (e.g., PowerPoint)
- Ask it to document EVERYTHING it did: code, mistakes, lessons
- Export the script and example output
- Feed to Claude Code via /init
- One-shot identical outputs with full control

> 💡 **Why This Matters**
> Claude.ai times out and restarts on large tasks. In Claude Code, you clear context and continue. Same skill, more reliability.

Blueprint #2: API Doc → Working App

Turn any API documentation into a functional app instantly.

- Grab API docs (many sites have a "View as Markdown" option)
- Save to a .md file in your project
- Run /init to familiarize Claude
- Request your app—Claude knows the API syntax

The AI Studio Cheat Code

For Google/Gemini APIs, use AI Studio to scaffold:

- Go to AI Studio → Build
- Select API capabilities (e.g., Video Understanding)
- Prompt a basic version in the browser
- Download as code (zip file)
- Import to Claude Code, /init, and enhance

Blueprint #3: Adapt to New Use Cases

Transform generic apps into specialized tools.

- Example: Video analyzer → Sales call analyzer
- Add domain-specific system prompts behind the scenes
- Customize the UI for the new use case
- Result: Tailored analysis (rapport building, objection handling, etc.)

ACT 4: REAL-WORLD APPLICATION

Core principles from teams who shipped real products with Claude Code.

The Power of Diverse Teams

- Technical + non-technical collaboration is a STRENGTH
- Different perspectives prevent blind spots
- Less overlap = more efficiency (everyone has a lane)
- Marketing minds + builders + testers = complete product

Planning Before Building

- Spend days on whiteboarding and wireframes FIRST
- Define brand, feel, and user experience upfront
- Use tools like Canva for visual planning
- Grounded planning = faster, cleaner building
- Think about user types: Who uses this? What do they feel?

Solve Real Problems

- Choose problems that resonate with your team personally
- Authenticity drives motivation through tough debugging
- Real pain points = real products people want

Embrace the Grind

- Things WILL break—expect it, don't fight it
- Give yourself grace when frustrated
- Read your screen—understand what Claude is doing
- The emotional rollercoaster is normal (every 30 minutes!)
- If you can install Claude Code, you can build with it

Strategic Integration Decisions

- Meet users where they are (e.g., WhatsApp for elderly users)
- Don't force new app downloads if existing tools work
- Think about redundancy (3 notification channels, not 1)
- Prioritize intuition over documentation for end users

> 💡 **The Bottom Line**
>
> Anyone can do this. If you can install Claude Code, that's step one—you're already there. Learn by doing, not by watching from the sidelines.

QUICK REFERENCE

Essential Commands

/init — Initialize repo, create CLAUDE.md
/plan — Enter plan mode before execution
/context — Check remaining context window capacity
/clear — Wipe context, start fresh session
/compact — Summarize and clear history
/re — Rewind to previous checkpoint
/plugins — Manage plugin marketplace
#instruction — Add to project memory (CLAUDE.md)

Key Files

CLAUDE.md — Command center, project rules, memory
settings.local.json — True source for settings, plugins, permissions
MCP.json — Direct MCP server configuration
codebase-inventory.md — Map of files to functionalities

Ready to go deeper?

Join the Early AI-dopters community for advanced modules, hackathons, and hundreds of AI builders leveling up together.

— Mark Kashef