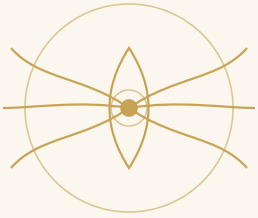

MONOLITH

The Prompt System Behind the Build

The actual operating patterns I use to run an AI enterprise — generalized so you can lift them straight into your own stack. Not prompt tricks. The system underneath.

By **Hunter Grant** · huntergrant.dev

Built with Claude — Fable 5
Day 2 of building in public

START HERE

You don’t write code. You write context.

Most people prompt for an *output* — a paragraph, a snippet, a quick answer. You ask, you get a thing, you move on. That’s a tool.

An enterprise is different. You stop prompting for outputs and start prompting for a **system** that produces them. You give the model a role, a memory, a set of constraints, and a cadence — and it generates the work on its own, the way you would, while you’re doing something else.

> *The model isn’t the thing you built.*
> *The system around the model is the thing you built.*

Everything in here is one of those system pieces. Five patterns I lean on every day, plus one bonus for when you want it running without you. The prompts are written so you can paste them in and adapt the brackets to your own project — today.

THE FIVE — AND ONE MORE

01	**The Cold Start**	load context before you ask for anything
02	**The Operator Frame**	give it a role and a loop, not a task
03	**The Constraint Stack**	front-load the rules before a big build
04	**Negative Space**	define what must *not* change
05	**The Mirror**	make the model improve its own prompts
+	**The Night Shift**	scheduled prompts that run while you sleep

01 The Cold Start

Load context before you ask for anything.

USE IT WHEN

You open a fresh session and you're about to ask the model to do real work on an ongoing project.

PROMPT

```
Before we do anything, load context.

Read these three files in order and hold them as ground
truth for this entire session:
  1. STATE.md          where the project is right now
  2. ARCHITECTURE.md   how the system is built, and why
  3. STANDARDS.md      the bar everything ships at

Then, before writing a single line:
  - Tell me which phase we're in and what "done"
    looks like for it.
  - Flag anything in STATE that looks stale or
    contradicts the architecture.
  - Confirm you'll operate at the level in STANDARDS.

Do not start building until you've done this.
```

> **Why it works.** A blank model defaults to generic. A loaded model defaults to *your* project. You're not saving context for later — you're making it the first thing the model reads every single time, so it argues from your reality instead of its training data. Three files is enough: where you are, how it's built, and how good it has to be.

02 The Operator Frame

Give it a role and a loop, not a task.

USE IT WHEN

You want the model to handle something on its own — not answer once, but run a process you can trust.

PROMPT

```
You are not a chat assistant. You are the operator of
this system.

Everything you do falls into one of three modes:
  OBSERVE  pull the current state — data, logs,
           files, metrics. Never act on memory;
           always check what's true right now.
  THINK    reason out loud. What changed, what it
           means, the options, the tradeoffs of each.
  ACT      execute exactly one decision, then
           report what you did and what it changed.

Start in OBSERVE. Show me what you see before you think,
and think before you act. If you're about to act without
observing first — stop, and observe.
```

Why it works. Most failures aren't bad reasoning — they're the model acting on a stale assumption. Forcing *observe* → *think* → *act* turns a one-shot answer into a loop you can actually hand off. It's the same shape whether the model is managing data, watching a signal, or running an agent.

03 The Constraint Stack

Front-load the rules before a big build.

USE IT WHEN

You're about to change a lot of files at once and you can't afford the model to drift halfway through.

PROMPT

```
We're about to change a lot at once. Before you touch
anything, here's the full frame. Hold all of it the
whole time.

  ARCHITECTURE   the stack, the data flow, the shape
                 of the system that can't change.
  REFERENCES     the target feel — "match the
                 restraint of X, the density of Y."
  HARD RULES     the things that have burned me
                 before, stated as laws. (e.g. "X always
                 comes last in the pipeline.")
  OUT OF SCOPE   what you are NOT changing this pass.

Now restate the plan back in your own words, list the
files you expect to touch, and wait for my go before
you start.
```

Why it works. A big build drifts when the model fills gaps with guesses. Front-loading the rules — especially the mistakes that already cost you — turns "creative interpretation" into execution. The restate-and-wait step catches the misunderstanding while it's still free, instead of forty files later.

04 Negative Space

Define what must not change.

USE IT WHEN

Anything generative — an image, a video motion, a redesign, an edit — where the model tends to "improve" the parts you already loved.

PROMPT

```
I'm going to describe what I want generated. Half of
that description is what must NOT change.

  WANT      make this happen — the one or two things
            I actually want to move or add.
  PROTECT   keep these exactly as they are —
            everything that should not move, shift, glow,
            recolor, or drift. List it explicitly.

The protected list is not a suggestion. If a change
would touch anything on it, don't make the change.
```

> **Why it works.** Generative models add. Left unconstrained, they polish the exact details you fell in love with. Naming what to *protect* is how you get one clean change instead of a brand-new image. It works the same for video motion, a UI reskin, or a line edit — the negative space is where the control lives.

05 The Mirror

Make the model improve its own prompts.

USE IT WHEN

A prompt you rely on is producing "fine" output — not wrong, just generic. Run this on the ones that matter, on a schedule.

PROMPT

```
Here's a prompt I've been using: [paste the prompt]

Don't run it. Critique it.
  - Where will it produce vague or generic output,
    and why?
  - What context is it assuming that it never
    actually states?
  - What would make the result sharper and more
    repeatable?

Then rewrite it. Give me the improved version, plus a
one-line note on what you changed and why.
```

Why it works. Your prompts are the highest-leverage thing you own, and they quietly decay as the project grows. Running the important ones back through the model — weekly is plenty — compounds. The system that writes your work gets a little sharper every week, and you never start from a blank page again.

+ BONUS · FOR WHEN YOU'RE READY TO LEAVE THE ROOM

The Night Shift

Scheduled prompts that run while you sleep.

USE IT WHEN

You've got a model wired to your tools and you want it working on its own cadence — not waiting for you to ask.

PROMPT

```
While I'm offline, run on this cadence:

  2:00 AM   REFLECT   review today. What worked, what
                      stalled, what you'd do differently.
  3:00 AM   EXPLORE   surface one idea worth my
                      attention — with the reasoning,
                      not just the headline.
  7:00 AM   BRIEF     three lines: what happened
                      overnight, what needs me today, what
                      you already handled yourself.

Keep the briefing short enough to read before coffee.
```

> **Why it works.** The line between a tool and an enterprise is whether it runs when no one's watching. Scheduled prompts turn the model into a teammate that shows up before you do — reflecting, exploring, and handing you a briefing instead of a blank screen. This is the one that makes it feel like a company.

THE WHOLE GAME

Five patterns, one idea underneath: you're not asking a model to do your work. You're building a *system* that does.

Context instead of instructions. A role instead of a task. Constraints instead of hope. A cadence instead of you. The code is downstream of all of it — which is why I haven't had to write a line.

— Hunter

FOLLOW THE BUILD

@huntergrant.dev

Building something real and want a second set of eyes on the system behind it? That's the work. The door's open.