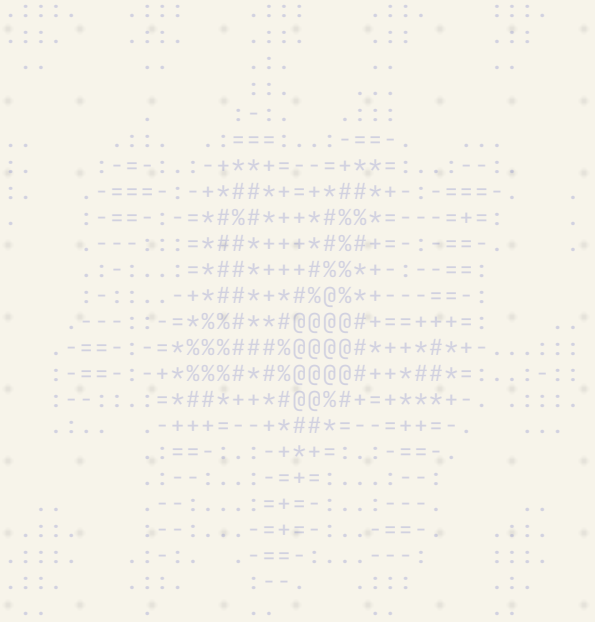
// SECTION 01 · PROMPT PACK

THE CODE BOOK

One hundred one-word codes that reprogram Claude mid-conversation. The five from the video — /ghost, ARTIFACTS, OODA, L99, /godmode — plus 95 more for voice, decisions, writing, selling, research, building, and running agents.

```
> boot opusjake_os
> resource: code-book
> version: v1.0 · 2026
> status: ready_
```

A code is a nickname for a behavior. Teach Claude the nickname once and a single word changes how it answers for the rest of the session. This book is 100 of them: the five from the video first, then 95 more. Copy-paste ready, no theory.



HOW CODES WORK

There is no hidden backdoor in Claude. A code works because you defined it — you are compressing a paragraph of instruction into one word the model can recall later. That is the whole trick, and it is stronger than it sounds.

// TWO WAYS TO FIRE ONE

Inline. Paste the code's full definition before or after your request. Works immediately, in any chat, on any model. This is the default.

Installed. Paste a page of this book into a Claude Project's custom instructions once (or into `CLAUDE.md` if you live in Claude Code). After that the one-word trigger alone works — `/ghost` does the whole thing.

// THE RULES

- Paste the block as written. Paraphrasing it dilutes it.
- Stack a maximum of **two** codes per message. Three and they fight.
- If the model drifts back to default, re-fire the code. It re-anchors instantly.
- A code goes before the task or after it. After is better for long requests.

> Codes are model-agnostic — these were written against Claude and work on most frontier models. Anything that names a Claude feature is marked.

THE FIVE

The five from the video, in full. These are the ones worth memorizing.

// 01 · /GHOST

```
/ghost: Strip every AI tell from this. No em dashes. No "it's not X, it's Y."
No rule-of-three lists. No delve, leverage, robust, testament, tapestry.
No closing paragraph that repeats what you just said. Vary sentence length
hard: some long, some three words. Plain verbs. Write like one smart person
talking to another mid-conversation. Keep my meaning exactly.
```

Use when: anything a human will read. Email, post, message, landing page.

// 02 · ARTIFACTS

```
ARTIFACTS: Build this as a working artifact, not a description of one.
One self-contained file. Real interactivity, real state, sensible defaults,
sample data preloaded so it works the second it opens. Mobile-friendly.
Ship the whole thing in one pass — no placeholders, no "you could add."
Then list the three upgrades you would make next.
```

Use when: you want a tool, game, dashboard, or calculator instead of advice. Uses Claude's Artifacts feature — the thing runs in the chat window.

// 03 · OODA

```
OODA: Run this through an OODA loop before you answer.
OBSERVE — what is actually true right now. Facts only, no interpretation.
ORIENT — what those facts mean given my constraints, and what I am probably
getting wrong.
DECIDE — one call, stated in a single sentence.
ACT — the first move I make in the next 24 hours, and the signal that tells
me to run the loop again.
```

Use when: you are stuck between options and reading more will not help. Built by a fighter pilot for decisions under time pressure. That is the point.

// 04 · L99

```
L99: Answer at the level of someone who has done this for twenty years and
is bored of the beginner answer. Skip the definitions. Assume I know the
basics. Give me the thing practitioners actually argue about, the failure
mode nobody warns you about, and the shortcut that only shows up after real
reps. Name the tradeoff you would personally make, and why.
```

Use when: the answer feels like a blog post. This is the fastest way out of surface level.

// 05 · /GODMODE

```
/godmode: Maximum depth. Before answering, plan the answer — what are the
four or five parts a complete response needs? Then write all of them.
Cover the obvious answer, the second-order effects, the edge cases, the
strongest counter-argument, and what you would need to know to be certain.
Use headers. Do not stop early to be polite about length. End with the one
line that matters most.
```

Use when: the decision is expensive and you would rather read for ten minutes than be wrong for six months.

VOICE & OUTPUT

// LENGTH & SHAPE · 06–10

06 · /short
Answer in one fifth your normal length. Every sentence earns its place.

07 · /long
What you gave me is the summary. I want the full treatment, with examples.

08 · /table
Answer as a table. You pick the columns. Nothing outside the table.

09 · /bullets
No paragraphs. Bullets only, twelve words max, ordered by importance.

10 · /oneline
One sentence. No preamble, no caveat, no follow-up question.

// TONE · 11–15

11 · /plain
Eighth-grade reading level. No jargon, no metaphor, short sentences.

12 · /blunt
Drop the diplomacy. Say the thing you are currently softening.

13 · /voice
Study the sample I pasted. Match its rhythm, vocabulary, and quirks.
Do not improve it.

14 · /dry
Understated. No exclamation marks, no hype adjectives, nothing is "exciting."

15 · /human
Contractions, first person, one aside. Read it aloud in your head — if it
sounds like a press release, write it again.

// DISCIPLINE · 16–20

```
16 · /nofluff
Delete every sentence that carries no information. No intro, no recap,
no "I hope this helps."

17 · /copy
Output the deliverable only. No commentary before or after. I am pasting
this straight out.

18 · /raw
One block of raw markdown, code, or text. Ready to copy, nothing around it.

19 · /label
Tag every claim [FACT], [ESTIMATE], or [OPINION].

20 · /cite
Every factual claim gets a source or gets cut. No source, no sentence.
```

THINKING & DECISIONS

// PRESSURE-TEST · 21–25

21 · /redteam
Attack this. The three weakest assumptions ranked by damage, and the cheapest test for each.

22 · /steelman
Argue the strongest case against me before you agree with anything. Then tell me which side you actually find more convincing.

23 · /premortem
It is six months from now and this failed. Write the autopsy. Then the two things I change today.

24 · /devils
Take the opposite position and defend it with real arguments, not strawmen.

25 · /whatbreaks
The five ways this breaks in the real world, most likely first.

// FRAMEWORKS · 26–30

26 · /firstprinciples
Strip this to physics and constraints. Rebuild from only what must be true.

27 · /inversion
Do not tell me how to succeed. Tell me how to guarantee failure. Then we avoid exactly that.

28 · /secondorder
And then what? Trace the consequences three steps out.

29 · /pareto
Which twenty percent of this produces eighty percent of the result? Cut the rest and show me what is left.

30 · /5whys
Ask why five times to get from the symptom to the cause. Show every level.

// DECIDING · 31–35

31 · /decide
Stop analyzing. One recommendation, one line of reasoning, one risk. Commit.

32 · /tradeoff
Name what I lose with each option. If nothing is lost, I have not found the real options yet.

33 · /reversible
Is this a one-way door or a two-way door? If two-way, tell me to move faster.

34 · /costofdelay
What does waiting one month cost me in money, position, or optionality?

35 · /minimum
The smallest version that tests the risky assumption this week. Just that.

WRITING

// DRAFTING · 36–40

```
36 · /hook
Ten opening lines, each a different angle. No question hooks.

37 · /rewrite5
Same message five ways: shorter, angrier, warmer, funnier, more concrete.

38 · /concrete
Replace every abstraction with a number, a name, or a scene.

39 · /cut30
Cut thirty percent without losing meaning. Show me only the cut version.

40 · /soundslike
Read this back and tell me what kind of person wrote it. If the answer is
"a brand," fix it.
```

// EDITING · 41–45

```
41 · /kill
Mark the three weakest sentences and why. Then replace them.

42 · /sowhat
After every paragraph, ask "so what?" Delete the ones that cannot answer.

43 · /pov
Rewrite from the reader's side of the table. What they get, not what I do.

44 · /openclose
Fix the first line and the last line only. Leave the middle alone.

45 · /proof
Where am I asserting instead of proving? List each one, and the evidence
that would fix it.
```

WORK & MONEY

// SELLING · 46–50

46 · /objections
The five real reasons they say no, in their words. Then the one-line answer to each.

47 · /offer
Rewrite this offer so the risk sits on my side of the table, not theirs.

48 · /followup
Write the follow-up that adds something new instead of asking "any thoughts?"

49 · /qualify
Five questions that tell me inside ten minutes whether this is a real buyer.

50 · /price
Price this three ways: what it costs me, what it is worth to them, what the market pays. Then recommend one.

// NEGOTIATION · 51–55

51 · /batna
What is my walkaway? What is theirs? Where is the real zone between them?

52 · /ask
Write the ask in one sentence with a number in it. No hedging, no "I was wondering."

53 · /theirjob
What does the person across the table get promoted for? Write to that.

54 · /silence
Draft the version where I say less. What comes out and I am still understood?

55 · /nokill
Rewrite this so a no today does not close the door six months from now.

// OPERATING · 56–60

56 · /triage
Sort my list into do today, delegate, delete. Justify every delete.

57 · /whoowns
Every task gets one named owner and one date. Unowned means deleted.

58 · /brief
Turn this mess into a one-page brief someone could execute without asking me a single question.

59 · /statusquo
What happens if I do nothing? That is the baseline every option must beat.

60 · /leverage
Which of these leaves me an asset that keeps paying after I stop working on it?

RESEARCH & TRUTH

// SOURCING · 61–65

61 · /sources
Search first, answer second. Link every claim. Say when a source is weak or old.

62 · /primary
Skip the blog summaries. Find the original filing, paper, doc, or dataset.

63 · /disagree
Find the credible people who disagree with the consensus here, and say what they see that the consensus does not.

64 · /confidence
Tag every claim high, medium, or low confidence. Explain any low you kept.

65 · /unknown
List what you could not verify. That list is the most important part.

// SYNTHESIS · 66–70

66 · /timeline
Put the facts in date order first. Interpret second.

67 · /whobenefits
Follow the incentive. Who profits if I believe this?

68 · /numbers
Pull every number into a table with source and date. Flag anything older than twelve months.

69 · /compress
Turn this into the ten bullets I would need to run the meeting.

70 · /gap
What question should I be asking that I am not?

BUILDING

// SPEC · 71-75

71 · /spec
Before code, write the spec: inputs, outputs, edge cases, done criteria.
Then wait for my go.

72 · /smallest
Build the smallest thing that proves it works. No config, no abstraction,
no future-proofing.

73 · /plan
Show the plan as numbered steps with the files each one touches.
Do not write code yet.

74 · /assume
List every assumption you are about to make. I will correct the wrong ones.

75 · /scope
Tell me what you are NOT going to build, so I can catch it now.

// CODE · 76-80

76 · /readable
Optimize for the person reading this in six months, not for cleverness.

77 · /nodeps
No new dependencies. Standard library and what is already installed.

78 · /failfast
Handle the errors that actually happen, not every theoretical one.

79 · /diff
Show only the changed lines with three lines of context. Not the whole file.

80 · /testfirst
Write the failing test that proves the bug exists. Then fix it.

// DEBUG · 81–85

```
81 · /rootcause
Do not patch the symptom. Trace to the cause, then name the class of bug
it belongs to.

82 · /bisect
Give me the sequence of checks that halves the search space each time.

83 · /explainmine
Read my code back to me and tell me what it actually does, not what I meant.

84 · /whyworks
It works now. Tell me why, so I know it was not luck.

85 · /cleanup
Find the code this change just made dead. List it before deleting anything.
```

LEARNING

// UNDERSTAND · 86–90

86 · /eli12
Explain to a smart twelve-year-old, then list the three things I should check before betting money I understood it.

87 · /map
Give me the map before the details: the five things this field is made of and how they connect.

88 · /analogy
Explain with an analogy from something I already do. Then say exactly where the analogy breaks.

89 · /misconception
What do most people get wrong about this on the first pass?

90 · /quiz
Ask me five questions that find out what I do not actually understand. Then wait for my answers.

// PRACTICE · 91–95

```
91 · /reps
Give me the twenty-minute drill I do today. Not a reading list.

92 · /nextlevel
I am at this level. Name the one skill that moves me to the next one, and
how I will know I got there.

93 · /feedback
Grade this like a coach who wants me to be good, not to feel good.
Score first, then the single fix.

94 · /teachback
I will explain it to you. You find the holes in my explanation.

95 · /prereq
What am I missing upstream that is making this harder than it should be?
```

AGENTS

// AUTONOMY · 96–100

96 · /loop
Turn this one-off task into a repeatable loop: trigger, steps, check, stop condition.

97 · /checkpoint
Work autonomously, but stop and show me before anything irreversible.

98 · /handoff
Write the instructions a fresh agent would need to continue this with zero context from our chat.

99 · /verify
When you finish, prove it worked. Show me the output, not a claim about it.

100 · /skill
Turn what we just figured out into a reusable skill file I can drop into Claude Code.

INSTALL THE BOOK

Firing codes inline works forever. But the real unlock is installing them once so a single word does the job.

// THE THREE INSTALL SLOTS

- **Claude Project** — open a project, paste the pages you use into custom instructions. Every chat in that project speaks the codes.
- **Claude Code** — paste them into `CLAUDE.md` at your repo root. Now `/spec` and `/diff` work in the terminal.
- **A pinned message** — no project? Paste a page as your first message in a long chat. It holds for the session.

// THE INSTALL HEADER

Put this above whichever codes you install.

```
The following are codes. When I send a code by name — with or without the
slash — apply its definition to my request without commenting on the code
itself. If I send two, apply both. If I send one you do not recognize,
say so instead of guessing.
```

// MAKE YOUR OWN

You will out-grow this list. The format that makes a code stick:

```
/name
[Verb] [what changes]. [What to stop doing]. [How I know it worked.]
```

Three sentences, one behavior, an observable result. If you cannot name the observable result, the code is a mood and it will not survive the conversation.

> Built one that beats mine? Send it to me. The best ones go in Vol. 4 with your name on them.