A 90-MINUTE CRASH COURSE · 7 PRINCIPLES · 4 PHASES

Problem Solving with General Agents

The operating discipline that turns any agent — Claude Code, OpenCode, Cowork, OpenWork — from a clever toy into a tool you can ship work on.

Two people open the same agent on Monday morning.

Same task: review a folder of vendor contracts, flag the non-standard clauses, produce a comparison memo.

PERSON A

22 *minutes*

A clean, verified output. Done.

PERSON B

90 *minutes*

Correction loops, a polluted context — then starts over.

SO WHAT WAS DIFFERENT?

Same agent. Same capability. *A different operator.*

Person A knew seven things Person B didn't. Not seven features — seven habits of mind for handing real work to a machine with hands. This course teaches those seven things.

THE ONE IDEA UNDERNEATH EVERYTHING

The tool is a choice. The mental model is the *work.*

Anyone can name the agents. The thing that actually matters is the model in your head that connects a general agent to a real business problem. Naming them is trivia; **using them is the skill** — and it's the same skill in every tool.

DEFINITION

A general agent is a co-worker with hands.

NOT THIS

A chatbot that answers questions.

It explains. You still do the work.

THIS

An AI that takes actions on your behalf.

It runs commands, reads files, writes files, calls services — a tool with hands, not a voice with opinions.

*This whole course lives in **Mode 1** : the problem-solving engagement. You open a tool, you solve a thing, the session ends, the outcome ships.*

WHO THIS IS FOR

The domain changes. The discipline doesn't.

STUDENTS & NEWCOMERS

Learning to direct a machine that acts

The first tool you'll trust with real stakes.

KNOWLEDGE WORKERS

Law, finance, marketing, HR, consulting

Contracts, models, briefs, hiring loops, research.

ENGINEERS

Reaching for Claude Code or OpenCode

Reviews, refactors, migrations, merges.

Every example ahead is cross-domain. The one that matches your work will land hardest — but the principle is identical in all of them.

THE FOUR TOOLS

Four interfaces to the same session.

ENGINEERING · TERMINAL-NATIVE

Claude Code
Anthropic's terminal tool

OpenCode
Open-source, model-agnostic

KNOWLEDGE-WORK · DESKTOP

Claude Cowork
Anthropic's desktop agent

OpenWork
Open-source desktop agent

Where they differ is surface — a typed command vs. a step card. **Everything that matters works the same in all four.** Learn the principles and your skill transfers whichever tool you're in.

PART ONE

I Why these principles look old.

The terminal, files, Git, SQL. They're all from another era — and they still do the real work.

Age, in the right category, is evidence of resilience.

A tool useful for *40 years* is probably more durable than one useful for *4*.

4 flash trend · likely already fading

40 proven survivor · the agent's foundation

Longer past survival is evidence of likely continued survival. These interfaces survived because they *work*.

THE STACK

The agent reasons in language. It *acts* through proven interfaces.

BASH · EXECUTE

GIT · TRACK & REVERSE

SQL · QUERY TRUTH

FILES · PERSISTENT MEMORY

LOGS · OBSERVE

TESTS · VERIFY

SCHEMAS · CONSTRAIN

The agentic era doesn't replace the old stack — it **activates** it. The Lindy stack *is* the agent's stack.

WHAT AN AGENT NEEDS FROM ITS TOOLS

Five properties an action surface must have.

STABLE	TYPED	REVERSIBLE	INSPECTABLE	GOVERNABLE
Files	**Schemas**	**Git**	**Logs**	**Permissions**
P5 · persists	P2 · constrains	P4 · undoes	P7 · reveals	P6 · bounds

Each long-lived technology already has the property an agent needs. The seven principles are the operator's discipline for using them.

IF YOU REMEMBER ONLY THIS

The essentials in five — 60% of the value.

1. **Action over talk.** Every prompt should produce an action or an artifact, not a paragraph.
2. **Code over prose.** When precision matters, ask for a schema, a table, a checklist — not a paragraph.
3. **Verify, don't trust.** Every meaningful output needs a check. "Looks right" is the failure mode.
4. **Small steps, atomic checkpoints.** Reversible units. Commit after each one lands.
5. **Files are memory.** The conversation is volatile; the filesystem is durable. Decisions belong in a file.

PART TWO

The Seven Principles.

Ordered by building dependency. Each one rests on the ones above it. Read them in sequence at least once.

P1 Bash Is the Key

P2 Code as Universal Interface

P3 Verification as a Core Step

P4 Small, Reversible Decomposition

P5 Persisting State in Files

P6 Constraints & Safety

P7 Observability

FAILURE · "IT ONLY TALKS"

Bash is the key.

The agent has full keyboard access to your machine — through commands instead of clicks. The first principle is to treat it that way.

> **The mental model:** the agent has hands. *Brief the hands, not the brain.*

THE SAME NEED, TWO PROMPTS

Rephrase the question as an action with an artifact.

CHATBOT PROMPT → AN ESSAY

"How should I summarize last week's customer interviews?"

The agent had hands. You asked it for advice.

AGENT PROMPT → AN ARTIFACT

```
Read every transcript in /interviews/week-12 .
Extract name, top 3 pain points, pricing
objections. Save to week-12-themes.md , sorted by
frequency.
```

A usable artifact, keyed to your 53 actual files.

Cut every verb that describes *how*. Keep only the verbs that describe what you want at the end.

FAILURE · "PROSE GETS MISREAD"

Code as universal interface.

3,000 vacation photos across three devices. Three apps each did part of it. One paragraph to an agent — organized by city, renamed by date, duplicates removed by image content — done in fifteen minutes. She wrote no code.

> The shape of what you ask for *is itself an interface.* A schema is not ambiguous. The clearer the contract, the less the agent has to guess.

The five powers code unlocks.

01

Precise thinking

Computes; doesn't approximate.

02

Orchestration

Writes the whole branch tree once.

03

Organized memory

Files become working memory.

04

Universal compat

Reads formats that never talked.

05

Tool creation

No app does it? It builds one.

The big wins live where two or three powers *compose*. The app's features were pre-built. Your needs aren't.

THE TWO SKILLS THAT SURVIVE AUTOMATION

The agent writes the code. You do the two bookends.

BOOKEND ONE

Define the problem

Frame it as a spec, a schema, a typed signature, a template. The shape is the spec; the agent fills it.

BOOKEND TWO

Read to verify

Read, not write. Enough to catch a wrong WHERE clause, or a number that doesn't tie back to a row.

Agent prose is fluent — that's the trap. Read for what's **true,** not what *sounds* right.

FAILURE · "LOOKS RIGHT, BREAKS LATER"

Verification as a core step.

Models produce outputs that are plausible — which is not the same as correct. They miscount lists, mis-cite paragraphs that don't exist, and ship code that compiles cleanly while failing on the third edge case.

> Verification must be a **step in the workflow** — not an afterthought. A finished-looking output is not a verified output.

THE KEY RULE

The agent that produced the output is the worst possible verifier of it.

It has the same blind spots that produced the original. Asking it to check its own work is two coats of the same paint. Verification needs an **independent path** — your own reading, a different model, a test, a type-checker, a database constraint.

VERIFICATION PASS

```
# before saving the final version: List every factual claim in the draft. For each, quote the source text that supports it. Flag any claim you cannot ground. Refuse to save until every flag is resolved.
```

FAILURE · "ONE CHANGE NUKED AN AFTERNOON"

Small, reversible decomposition.

Models are good at small, specified moves and progressively worse at large, vague ones. The same 12 steps as 12 prompts — each verified before the next starts — keeps the agent on track throughout.

> The rule of thumb: *if reversing the change would take more than two minutes, it was too big.*

• 1998 · TOY STORY 2

90% of two years of work, gone in seconds.

Someone ran the wrong command. The backup system had silently failed weeks earlier. The film was saved only because one employee had a personal copy at home.

git commit

Reversibility has to be a system property — not a daily discipline you might forget. A commit after every step turns disasters into nuisances.

PRINCIPLE FIVE

FAILURE · "IT FORGOT WHAT WE DECIDED"

Persisting state in files.

A conversation is volatile. The filesystem is durable. The solution isn't longer context windows — it's external memory.

VOLATILE

Conversation

DURABLE

Files in the folder

ON-DEMAND

Referenced docs

A table of contents — not an encyclopedia.

CLAUDE.MD

```
# Matter: Smith v. Acme ## Where things live /pleadings — filed (do not edit) /depositions — YYYY-MM-DD-NAME.pdf ## Critical rules - Never finalize a brief citing a passage we haven't quoted in full. - Flag anything that may waive privilege.
```

A short markdown file at the folder root that the agent reads automatically at session start. You stop re-explaining; it stops forgetting.

Earned, line by line.

Starter: under 250 words. Mature: under 60 lines. Every line earned by a specific past mistake — not speculative documentation.

FAILURE · "IT TOUCHED FILES I DIDN'T AUTHORIZE"

Constraints & safety.

An agent that can do anything is one you must watch every second. Constraints don't slow the work down — they raise the autonomy ceiling.

LEVER ONE

Scope

What files & data it can see.

LEVER TWO

Connections

What services it can reach.

LEVER THREE

Approvals

When it pauses for your OK.

Constraints set in config are durable. Constraints set in your prompt are aspirational.

Climb deliberately. Step back down when the task changes.

1. **Watching closely.** Default for any novel task. Approve every action.
2. **Ambient supervision.** Done it 3–4 times. Approve the plan, glance at execution.
3. **Walk away.** You trust the pattern. Come back to a finished deliverable.
4. **Act without asking.** No pauses, but still watching. Pre-approved inputs only.
5. **Scheduled.** Hands-off & recurring. Only for tasks already trusted at "walk away."

If you wouldn't trust it at "walk away," don't schedule it. Automation amplifies whatever calibration you've built — including the gaps.

PRINCIPLE SEVEN

FAILURE · "I DON'T KNOW WHAT IT DID"

Observability.

Every meaningful action should be visible to you in close to real time. It's how you debug a drifted session, build the track record to climb the ladder, and trust the output enough to use it.

> You can only direct what you can *see.*

The single biggest source of "the agent did something I didn't expect" is the user not having looked. Watch the execution view at least once on every novel task.

WHEN A SESSION GOES OFF THE RAILS

Five symptoms — then stop typing.

1. It references earlier chat that has nothing to do with the task.
2. Responses get longer and vaguer, with more hedging.
3. It contradicts a constraint you stated several turns ago.
4. It apologizes repeatedly without making progress.
5. It proposes touching files or connectors you didn't mention.

Don't fix it with another prompt. Reset, paste the one or two facts that matter, continue from a file. The reset is almost always faster than the rescue.

PART THREE

The four-phase workflow.

II

In production, the seven principles collapse into one loop. Once it's in your hands, the principles fire automatically inside the phases.

ONE LOOP, ANY DOMAIN

Explore → Plan → Implement → Commit.

PHASE 1

Explore

Read the files, surface unknowns. Read-only — no writes yet.

P1 · P7

PHASE 2

Plan

A written, structured plan. Save it. Review it. Edit it.

P2 · P5

PHASE 3

Implement

Small atomic steps. Verify each. Commit each.

P4 · P3

PHASE 4

Commit

Final verification. Persist decisions to the rules file.

P6 · P7

Constraints (P6) wrap the whole loop. They aren't a phase — they're the boundary every phase runs inside of.

WHERE THE WORK PAYS OFF

Almost all the leverage is in the *Plan.*

A written plan is a structured artifact you can save, read, edit, and approve — before a single change touches your files. The same task in one prompt gives you a block of plausible text with no checkpoint where you could have intervened.

Slower in clock time on the first run. Faster in trust-time forever after.

DIAGNOSTIC SHORTHAND

When something breaks, name the pattern.

#	PATTERN	SYMPTOM	REACH FOR
1	**The Drift**	Agent wanders from the brief	**P5** · Persistence
2	**The Confident Wrong**	Plausible output that's quietly incorrect	**P3** · Verification
3	**The Big Bang**	One huge change nukes hours of work	**P4** · Decomposition
4	**The Scope Creep**	Touches things you didn't authorize	**P6** · Constraints
5	**The Black Box**	Ran 20 minutes; you've no idea what it did	**P7** · Observability

"That was a Confident Wrong" tells a teammate exactly which step was missing — without anyone relitigating the run.

THE WORKED EXAMPLE

Review a complex input → produce a verified, structured response.

ENGINEER TRACK

A contractor's pull request

Review the diff, flag risks, write a response — grounded in file:line.

DOMAIN-EXPERT TRACK

A vendor's master services agreement

Flag deviations from your redline standard — grounded in section cites.

Different domains. **Identical workflow shape.** The phases don't change; only the inputs and outputs do. That's what makes the loop portable.

Friction is the curriculum.

THE FRICTION YOU FEEL	THE PRINCIPLE TO BUILD
"Why is it just chatting?"	**P1** — rewrite as an action with an artifact
"Why is the output subtly wrong?"	**P2** — constrain the format
"Why did this confident answer fail?"	**P3** — add a check step
"Why did one prompt nuke my work?"	**P4** — break it up
"Why does it keep asking the same thing?"	**P5** — put it in the rules file
"Why did it touch a folder I didn't name?"	**P6** — tighten scope

Build the response to each friction when you hit it — not before. A rules file grown line by line is memory. Written speculatively, it's just documentation.

THE PAYOFF

You start manual. The friction *is* the training.

Each piece of friction maps to a principle. Build the muscle on real inputs, and the discipline becomes invisible — you cross from *learning* the principles to *using* them.

The second run of a task is usually 40–60% faster. The third is where the discipline disappears into habit.

THE CAPABILITY TEST

You've completed this course if you can do all five — with real work.

1. Reframe a chatbot prompt as an agent task with an explicit artifact. P1 · P2
2. Write the output shape — schema, table, template — before asking for content. P2
3. Name two independent verification paths and invoke one before shipping. P3
4. Decompose non-trivial work into atomic units with a checkpoint after each. P4
5. Maintain a rules file earned line-by-line, and explain a session from its trace. P5 · P7

The seven principles, one line each.

P1 **Bash Is the Key.** Brief the hands, not the brain.

P2 **Code as Interface.** Specify the shape; eliminate prose ambiguity.

P3 **Verification as a Step.** "Looks right" is the failure mode. Force a check.

P4 **Reversible Decomposition.** Atomic units. Verify each. Commit each.

P5 **Persisting State in Files.** Conversation is volatile. Files are memory.

P6 **Constraints & Safety.** Constraints enable autonomy; they don't limit it.

P7 **Observability.** You can only direct what you can see.

→ *Five make the work happen. Two make the discipline survive real projects.*

THE WHOLE COURSE IN ONE LINE

Anyone can name them. Now you can *use* them.

The tools will keep changing. The principles govern the session; the tools are interfaces to it. Learn to think with the seven, and your skill transfers whichever tool you happen to open.