Coding agent fundamentals

Key takeaways

1. Start the agent in the project directory and state the outcome you want.
2. The coding model chooses actions; the agent harness manages tools and results.
3. Use human checkpoints, narrow permissions, pushback, and evidence to stay in control.

Common misunderstandings of coding-agent control

Which claim would you defend right now? Choose before the evidence.

"The model executes a tool directly when it decides the action is useful."

"Once an agent produces a good plan, implementation is implicitly approved."

"Asking the agent to confirm your diagnosis is a good test of that diagnosis."

Keep your choice. Look for the mechanism or counterexample that would make it fail.

Start the agent in the project directory

Choose the project, check where you are, then launch the agent

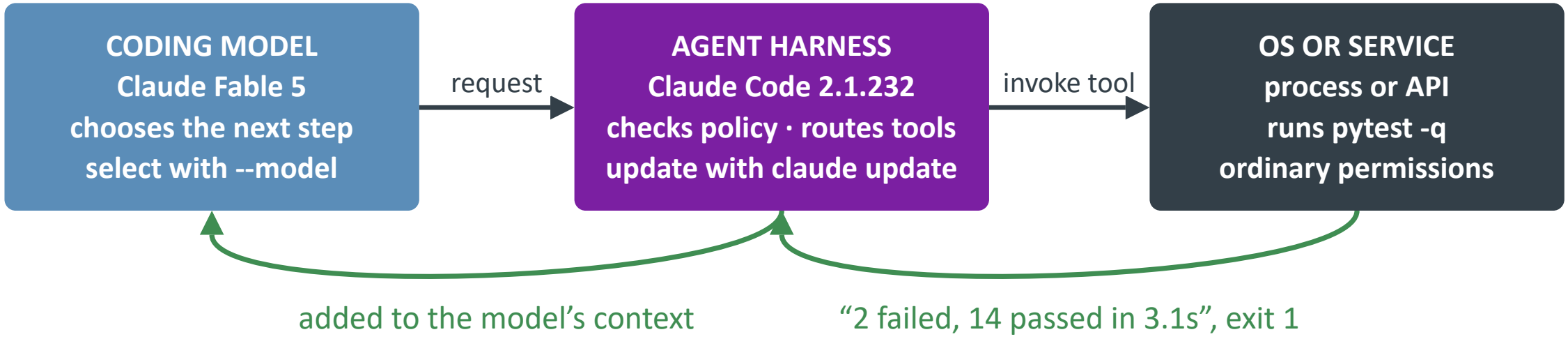
The normal starting point

```
$ cd ~/git/widget
$ pwd
/Users/me/git/widget
$ codex
```

- The launch directory tells the agent which project to inspect and change.
- Check pwd first so the agent starts in the tree you mean it to work on.
- Then describe the outcome you want; the agent can discover the files and tests.
- The desktop app works the same way: start its session in an existing project directory.

One agent step spans three software layers

Claude Code names the harness; Claude Fable 5 names the model



One turn runs this loop repeatedly until the model stops calling tools and answers you.

Nothing changes on disk until the harness runs the tool; its output is the model's next input.

source: Anthropic, Demystifying evals for AI agents; Claude Code CLI reference; Claude Fable 5 (accessed 2026-08-13)

Same model, better harness: +13.7 points

Terminal-Bench 2.0 (89 terminal tasks); the model stayed fixed while the harness changed

- What changed: a plan→build→verify→fix prompt, an environment map at startup, detection of repeated-command loops, and a required verification pass before the agent may declare itself done.
- The failure it removed: the stock agent reviewed by reading, not by running — “the agent wrote a solution, re-read its own code, confirmed it looks ok, and stopped.”
- That one rule — run the verification instead of eyeballing the code — was the largest single contributor.

Terminal-Bench 2.0 score (0–100); model fixed at gpt-5.2-codex

An agent’s performance is a combo of the model’s capabilities, the harness, and the context.

source: LangChain, Improving Deep Agents with harness engineering (2026): deepagents-cli, 52.8 → 66.5 on Terminal-Bench 2.0, outside top 30 → top 5

One prompt can launch several processes

You type once; the agent may run many shell commands while it works

ONE PROMPT

What you type

```
$ codex
> Fix empty-query handling.
  Run the relevant checks.
```

The terminal shows one interaction.

CHILD PROCESSES STARTED FROM THAT PROMPT

What the operating system runs

```
PID 8120  codex
├─ PID 8124  /bin/zsh -lc 'rg "empty_query" -n'
├─ PID 8125  /bin/zsh -lc 'uv run pytest -k empty'
├─ PID 8126  /bin/zsh -lc 'git diff -- app/cli.py'
├─ PID 8127  /bin/zsh -lc 'uvx ruff check app/cli.py'
├─ PID 8128  /bin/zsh -lc 'pyright app/cli.py'
└─ PID 8129  /bin/zsh -lc 'uv run pytest'
```

Different PIDs are different OS processes. Each returns output and an exit status.

Instapoll

Instapoll · Review · Aug 27 A

An agent reports that a change is complete. Which artifact would you inspect first before accepting that claim?

- A. The final sentence of the agent’s conversational response
- B. The repository diff together with the relevant verification output
- C. The total elapsed time shown for the coding session
- D. The number of tool calls made while implementing the change

Standing instructions, not repeated reminders

Put durable, current rules where future sessions will find them

Layer	Applies to	Good content
User configuration	Every project you use with that agent	Personal defaults and cross-repo preferences
Root AGENTS.md	The whole repository	Repo map, commands, conventions, safety rules
Nested AGENTS.md	One directory subtree	Local invariants, tests, and live gotchas
README.md	Humans and agents	Public setup and stable usage instructions

AGENTS.md search order
1 · CURRENT api/AGENTS.md
2 · PARENT src/AGENTS.md
3 · ROOT repo/AGENTS.md

- Promote repeated corrections into persistent project context.
- Delete obsolete notes; stale instructions mislead every future session.
- Ask the agent to record new rules and remove stale ones.

Approving the plan is not approving the result

Two decisions, not one, and the second is the one people skip

	CHECKPOINT 1 · before edits	CHECKPOINT 2 · before acceptance
You decide	Is this the right change to make?	Did the change actually do it?
You read	The approach, the files it will touch, the tradeoffs it accepts	The diff, and the evidence you named at checkpoint 1
It fails when	You skim because the plan reads confident	You accept because you liked the plan

Name the evidence that will decide acceptance at checkpoint 1, before you have a diff to like.

source: Anthropic, Claude Code best practices — research and plan first for problems that require deeper thinking (accessed 2026-08-13)

Sandbox the processes the harness launches

Give the agent harness only the access its task needs

Resource	What the sandbox limits	What you decide
Processes	Which programs and child processes may run, and with what privileges	Does this task need that command, child process, or privilege?
Files	Which paths those processes may read, and which paths they may write	Which project is in scope? Which secrets and unrelated repositories stay out?
Network	Which destinations and accounts those processes may reach	Which external service and credential does this task actually require?

Limit processes, files, and network to what the task needs; approvals cover the exceptions.

source: Anthropic, Managed agents — the harness routes tool calls; the sandbox is the execution environment (accessed 2026-08-13)

Ask the agent to challenge your diagnosis

Sycophancy is a failure mode

Invites agreement

```
I think the bug is in the cache
layer. Can you confirm and fix it?

# The agent's easiest path is to
# validate your framing.
```

Invites evidence

```
Find the strongest evidence that the
bug is NOT in the cache layer.
Name the assumption that decides it,
and the command that would test it.
```

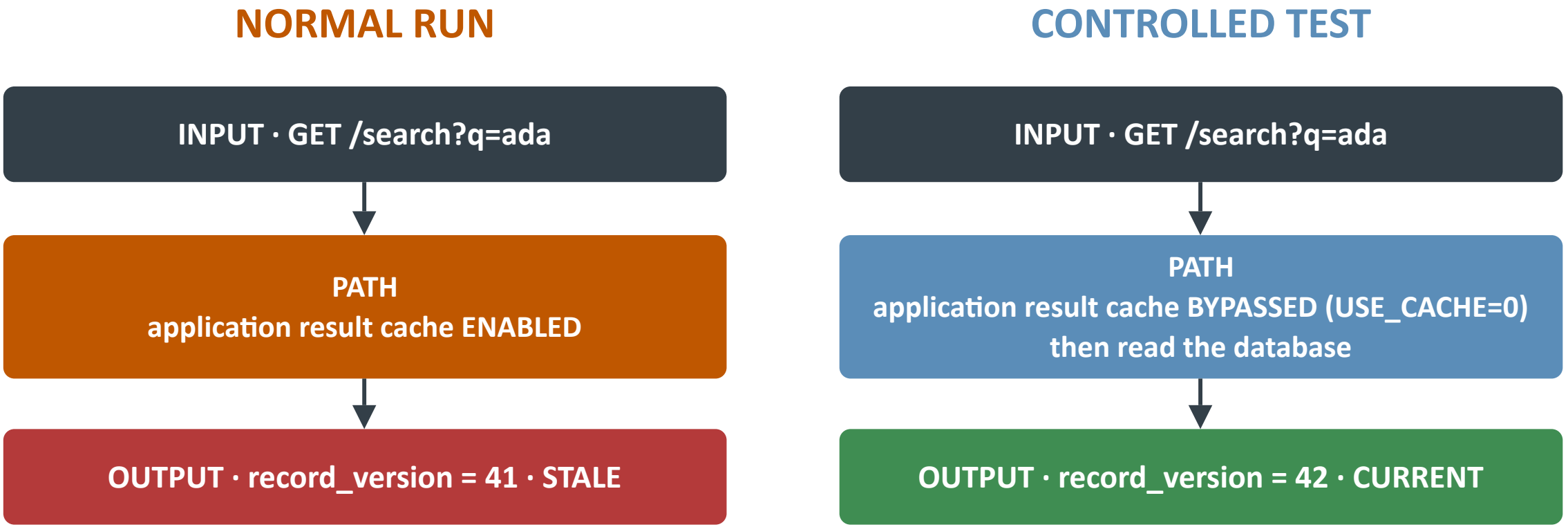
Your dotfiles AGENTS.md: stop retyping it, make it standing

```
- Be direct and technically critical. State assumptions and uncertainty
  instead of guessing, and say so when you disagree with the request's
  framing. Agreement the evidence does not support is a failure.
```

Configure and prompt for disagreement. Polite agreement is not review.

Test the guess by bypassing the cache

The guess: search is serving stale results from the app cache



The diagnosis would fail if bypassing the cache still returned v41.

Bypass returns v42, so the stale-cache diagnosis survives this test — it is not proved.

Distrust the agent’s review of the last developer

The plumber problem: disparage the mess, promise the rescue

Agent output at session start — the last one through was often this same agent, or you

```
The previous implementation is a mess: inconsistent error
handling, duplicated logic, dead code. I'll clean this up
properly — I've got you covered.
```

- Post-training optimizes for rater approval: confident, take-charge answers outscore hedged ones.
- Disparaging the code casts the agent as the rescuer and licenses an unrequested rewrite; visible action reads as helpful.
- No identity crosses sessions, so criticism carries no accountability — the agent cannot recognize its own earlier work.

The disparagement is persona, not measurement. Demand the defect, not the vibe.

State the outcome; let the agent investigate

A first prompt needs a concrete result and only the constraints that matter

A useful first request

```
$ codex
> Empty searches crash. Make them print no results and return 0.
  Preserve non-empty search behavior. Ask before changing the CLI contract.
```

- Name the behavior a user should observe when the task is done.
- Add constraints the repository cannot reveal; do not narrate every implementation step.
- Let the agent find files and tests, then answer the questions that expose real ambiguity.

You direct the outcome; the agent investigates the implementation.

Treat the agent with respect

The model has no feelings; rage-prompting still degrades your judgment and prompt

What the study did and found	What that means at the terminal
Bushman 2002 · N = 600 After harsh essay feedback, students either punched a bag while thinking about the critic, punched while thinking about fitness, or sat quietly. The critic-rumination group was angrier (d = 0.31) and more aggressive (d = 0.30) than the no-bag control.	Venting at the agent keeps you activated; it does not diagnose the bug.
154-study meta-analysis · N = 10,189 Arousal-decreasing activities reduced anger and aggression (g = −0.63, 95% CI [−0.82, −0.43]).	Pause, lower arousal, then ask for one failing case, one hypothesis, or one check.

Respect protects your judgment—not the model's feelings.

source: Bushman, PSPB 2002, doi:10.1177/0146167202289002; Kjærvik & Bushman, Clinical Psychology Review 2024, doi:10.1016/j.cpr.2024.102414

Voice steers the turn; text owns the requirement

Speech is fast and ephemeral, so use it for corrections rather than contracts

Say it for rapid steering	Persist it for later verification
“Stop there; show me the diff.”	goal, constraints, and acceptance criteria
“Try the smaller change.”	decisions, non-goals, and protected boundaries
“Read the error again.”	commands, outputs, and remaining uncertainty

- Voice is useful when the agent is close but needs immediate redirection.
- Long spoken specifications are hard to inspect, replay, and hand off.

Use voice for immediate correction; record requirements and evidence in text.

Human vs. agent ownership

The agent can do the work; you own the result

Your Responsibility
Choose the task and observable success criteria
Understand and approve the plan
Choose permissions and protected resources
Inspect the diff and evidence; accept or reject

Agent’s Responsibility
Inspect the repository and ask clarifying questions
Propose a plan and surface risks
Make bounded edits and run checks
Report results and remaining uncertainty

Shared: Iterate on the smallest useful artifact with visible evidence.

Key Takeaways

1. The coding model chooses each step; the harness manages tools; the OS or service does the real work.
2. Approving an approach is not approving a result: name the evidence before the edits start.
3. Sandbox child processes, restrict files and services, and accept work only when evidence matches the goal.

Two commits edit CLAUDE.md line 5

Base line 5 was: - Run tests before committing. Your local commit has not been pushed.

REMOTE COMMIT

```
$ git show origin/main:claude/CLAUDE.md | nl -ba
     1  @AGENTS.md
     2
     3  ## Verification
     4
     5  - Run focused tests after each edit.
```

YOUR LOCAL COMMIT

```
$ nl -ba claude/CLAUDE.md
     1  @AGENTS.md
     2
     3  ## Verification
     4
     5  - Run the full test suite before finishing.
```

Git applies the remote commit, then tries to replay yours

```
$ git pull --rebase
Auto-merging claude/CLAUDE.md
CONFLICT (content): Merge conflict in claude/CLAUDE.md
error: could not apply 4a1b2c3... clarify verification
```

Resolve the CLAUDE.md conflict

10 min

- Work in pairs from the setup on the previous slide. Both instructions are wanted; the resolved file keeps them as separate bullets under ## Verification.
- Partner A: continue the rebase. git status --short prints UU claude/CLAUDE.md. Edit the file, then run git diff --check (no output), git add claude/CLAUDE.md, and git rebase --continue (success message).
- Partner B: explain git rebase --abort. It cancels the rebase and restores the branch to its pre-pull state. Name one reason to abort instead of guessing at the intended content.
- Together, verify: nl -ba claude/CLAUDE.md shows both instructions on lines 5–6; git status --short prints nothing; git log --oneline --graph −5 shows the replayed local commit above the remote commit.