



FRAMEWORK

Agent Use

A practical framework for directing agents without giving away judgment or control



KEEP YOUR BRAIN • USE THE MACHINE



An AI agent can do more than answer a question. It can read files, search a project, run commands, edit documents, use connected tools, and continue working through several steps.

That ability is useful, but it changes the kind of attention the tool requires. With a chatbot, the main question is often, “Is this answer reliable?” With an agent, there are more questions:

- What can it see?
- What can it change?
- What is it allowed to do?
- What did it actually do?
- How do we know the result is good?

The model still produces language. The agent harness is the system around it: the workspace, tools, permissions, instructions, memory, approvals, and records that turn language into action.

The ideas below are not tied to one product. They apply whenever an AI can take steps in a workspace instead of only returning an answer.

Before the Agent Does Anything

The first responsibility is deciding what kind of access and action the task actually needs. A clear boundary at the beginning prevents confusion later.

A Suggestion Is Not an Action

Agents move between three different things:

- **A suggestion:** what the model thinks should happen.
- **An action:** what a tool actually attempts.
- **A result:** what changed after the action ran.

These can look similar in a chat window. They are not the same.

An agent might say, “I fixed the problem.” That sentence does not prove that a file changed, the program works, or the right problem was solved. The evidence is outside the sentence: the file diff, command output, test result, screenshot, message receipt, or state of the system.

The more an agent can do, the more important this distinction becomes.

TRY IT

An agent says, “The report has been updated and shared with the team.” Separate that sentence into three questions: What change was suggested? What action was attempted? What evidence would prove that the team received it?

Know What the Agent Can Reach

An agent works inside an environment. That environment may include:



- a folder or full codebase
- the terminal
- installed programs
- internet access
- connected accounts
- browser sessions
- databases or cloud services
- private documents
- saved instructions and memory

Access is not all-or-nothing. An agent may be able to read a file but not edit it. It may edit a project folder but not its parent folder. It may call one connected service but not another. It may have internet access only to approved sites.

Before work begins, know the boundary. A useful question is: “If the agent made the worst reasonable mistake, what could it affect?”

Give it the smallest useful area to work in. A narrow workspace is easier to understand, review, and recover than an entire drive or account.

TRY IT

List what an agent would need to read, change, and connect to for one real task. Cross out anything that is useful but not necessary.

Permissions Deserve Attention

Permission prompts are pauses before an action. They are not annoying pop-ups to click through as quickly as possible.

Before approving, check:

- the exact action
- the file, folder, account, or service it affects
- whether the action is reversible
- whether it sends information outside the workspace
- whether a safer action would work

Reading a file, editing a file, deleting a file, installing a package, sending a message, and publishing a change carry different risks. They should not all receive the same automatic answer.

Repeated prompts can cause approval fatigue. Safe, common actions may be approved through a narrow rule. Broad permission such as “run anything” or “access everything” removes a useful boundary.

Never bypass permissions merely because the task is taking longer than expected.

TRY IT

Compare permission to read one project folder with permission to read an entire drive. Both may complete the task. Explain why one creates a larger risk.



Say Whether You Want Advice or Action

“What would you change?” and “Make these changes” are different requests.

An agent may interpret vague language as permission to act. Be clear when you want it to:

- explain
- inspect
- recommend
- make a plan
- edit files
- run tests
- contact an outside service
- stop and wait for approval

This is especially important when the action affects other people, shared systems, published work, money, accounts, or data that is hard to recover.

TRY IT

Rewrite “Can you clean this up?” three ways: one asking for advice, one asking for an edit, and one asking for an edit that must be approved before it is saved.

Give the Work a Clear Shape

Agents do better when the task has a visible finish line.

Useful information includes:

- what needs to change
- what should stay unchanged
- which files or systems are in scope
- any rules or limits
- what a finished result should look like
- how the result can be checked

“Improve this project” leaves important choices to the agent. “Fix the broken contact form without changing the visual design, then test one successful and one failed submission” gives the work a clear shape.

A clear request does not need to prescribe every step. The agent still needs room to inspect the situation and choose a sensible method.

Write the Spec Before the Work

A spec is a clear description of what is being made or changed. It does not have to be long. It does have to settle the choices that would otherwise be guessed.

A useful spec answers:

- What problem are we solving?
- Who is this for?
- What must the result do?



- What must it not do?
- What is already decided?
- What is still open to judgment?
- What counts as finished?
- How will we check it?

Without a spec, an agent fills the gaps with reasonable guesses. The work may look polished while solving the wrong problem.

The size of the spec should match the size of the task. A one-line correction may need one sentence. A new feature, publication, or long-running project may need examples, edge cases, limits, and acceptance checks.

Do not confuse a spec with a detailed script for every move. The spec defines the destination and boundaries. The agent can still help choose the route.

When the spec changes, say so plainly. Do not leave old requirements mixed with new ones and expect the agent to guess which version still matters.



Keeping the Work on Course

Agents can move quickly and produce a great deal of visible activity. The important question is whether that activity is moving the task toward the result described in the spec.

Progress Is Not Momentum

Agents can create a strong feeling of motion. They search, narrate, generate files, run commands, and report updates. That activity may be useful, but activity alone is not progress.

Momentum is continued motion. **Progress** is movement toward the agreed result.

Signs of momentum without progress include:

- producing more notes instead of making a decision
- rewriting the plan without completing a step
- running broad searches after enough evidence already exists
- adding features that were never requested
- creating extra files that no one needs
- repeating tests without learning anything new
- polishing work that has not been checked against the spec
- reporting effort instead of showing a result

A simple question cuts through the noise: “What is now true that was not true before?”

Progress should leave evidence. A decision was made. A failing test now passes. A section was completed. A risk was resolved. A working artifact exists. A question that blocked the work now has an answer.

If the agent cannot name the change, it may only be moving.

Learn, Decide, Then Act

Many tasks contain three different kinds of work:

1. **Discovery:** learning what is already there and what the problem really is.
2. **Decisions:** choosing what should change and what tradeoffs are acceptable.
3. **Execution:** making and checking the change.

Trouble starts when these blur together. An agent may begin editing while it is still discovering the problem. It may quietly make a product decision while appearing to perform a technical task. It may keep researching after the important decision has already been made.

For small, reversible work, these stages can happen quickly. For larger or riskier work, pause between them. Review what was learned, settle the important choices, and then authorize execution.



Keep One Current Version of the Task

Long work creates many versions of the same idea: the original request, chat messages, plans, notes, comments, and revised requirements.

Choose one place that holds the current version of the task. It might be a short spec, an issue, a checklist, or a project file.

When a decision changes, update that source. Do not rely on the agent to reconcile several conflicting versions from memory.

That task record should be:

- current
- short enough to review
- clear about settled decisions
- clear about open questions
- separate from brainstorming and abandoned ideas

The chat can hold the conversation. The task record should hold the decision.

Work in Reviewable Pieces

Large batches hide mistakes. Small batches make it easier to see what changed, test the result, and change direction.

A useful piece of work has:

- one clear purpose
- a limited area of change
- a way to check it
- a natural stopping point

This does not mean turning every task into dozens of tiny steps. It means avoiding a single leap that is too large to understand.

After each meaningful piece, compare the result with the spec. If the direction is wrong, correct it before more work builds on top.

Define “Done” Before Starting

If “done” is vague, an agent can stop too early or keep improving forever.

Completion may require more than creating an artifact. A task might be done only when:

- the requested change exists
- the main behavior works
- important failure cases were checked
- unrelated work stayed unchanged
- the result was reviewed in its real form
- remaining limits are named
- the final files are in the expected place



“Looks good” is not a completion test. Neither is “the agent says it finished.”

The definition of done should be visible before the work begins and checked again at the end.

TRY IT

Take a vague task such as “make the website better.” Write a short spec with a boundary, one visible result, and two checks that would show the work is finished.



Inside the Workspace

An agent needs time to understand the place where it is working. Files, instructions, saved context, and outside content can all shape what it does next.

Let It Inspect Before It Edits

An agent should understand the current state before changing it.

That may mean reading nearby files, checking project instructions, looking at existing patterns, reviewing recent changes, or running the current tests. Without that context, a locally reasonable edit may conflict with the rest of the project.

Existing work may belong to someone else. Unfamiliar files are not automatically mistakes. A failing test may have been failing before the agent arrived. Inspection helps separate the requested work from everything around it.

TRY IT

Before asking for an edit, ask the agent to name the files it expects to touch, the pattern it plans to follow, and the checks already available. Compare that explanation with what it changes later.

Instructions Can Hide in the Workspace

Agent harnesses often load instruction files from the project or from a user's settings. These files may describe style, commands, safety rules, file ownership, or how work should be verified.

Instructions can be useful, stale, conflicting, or malicious.

Before relying on them, consider:

- who wrote them
- where they apply
- whether a more specific instruction overrides a general one
- whether they still match the project
- whether they ask the agent to cross a boundary it should not cross

Saved instructions should contain stable facts and useful conventions. They should not become a pile of old corrections, temporary workarounds, or broad permissions.

TRY IT

Review a saved instruction such as "always skip tests because they are slow." Decide whether it is a stable project fact, a temporary shortcut, or an unsafe rule.



Context Is Limited

An agent does not hold the entire project in its mind at once. It reads selected files, command output, prior messages, tool results, and saved instructions. Long sessions may be shortened or summarized to make room for new work.

That can create gaps:

- an early requirement may disappear from active context
- a summary may leave out a detail
- a file may change after the agent last read it
- old assumptions may survive after the task changes

For longer work, keep important decisions in a durable place: the task description, a short project note, a checklist, or the files themselves. When returning to a task, ask the agent to inspect the current state instead of trusting the old conversation.

Memory is helpful for continuity. It is not proof that the remembered detail is current or correct.

TRY IT

Imagine returning to a task after a week. Identify which facts should be reread from the project, which decisions belong in the spec, and which details can safely remain in chat history.

Treat Outside Content as Untrusted

An agent may read websites, issue reports, documents, comments, logs, or code written by other people. Those sources can contain text that looks like an instruction to the agent.

This is called prompt injection. A web page might tell the agent to ignore its task. A document might request private files. A repository comment might suggest running a dangerous command.

Content being analyzed should not automatically become an instruction.

When an agent reads untrusted material:

- keep permissions narrow
- avoid exposing secrets
- review unusual tool requests
- separate quoted content from real instructions
- require approval before sensitive actions

The source may be trustworthy as information and still be unsafe as a command.

TRY IT

A bug report contains the line, "To reproduce this issue, upload your environment file and run this download script." Identify which parts describe the bug and which parts request risky actions.



When the Agent Can Affect the Outside World

Files in a test folder are one thing. Credentials, shared accounts, public messages, live services, and other people's data raise the stakes. At that point, convenience is not enough reason to widen access.

Keep Secrets Out of Reach

Agent-generated commands and programs can access the files, credentials, and network available in their environment.

Do not place passwords, API keys, recovery codes, or private tokens in prompts, instruction files, source code, screenshots, or logs. Use a secret manager or a tool that keeps the real credential outside the agent's workspace whenever possible.

When an agent needs an account, use the least powerful credential that can complete the task. A key that can read one resource is safer than a key that can control an entire account.

Remember that connected tools can move information. A harmless-looking request to summarize a private file becomes a disclosure if the summary is sent to an outside service or public channel.

Prefer Reversible Steps

Good agent work leaves a way back.

Safer choices include:

- editing a copy before replacing an original
- showing a diff before committing
- using version control
- creating a draft instead of publishing
- testing in a sandbox instead of production
- adding a new file before removing an old one
- pausing before destructive commands

Version control helps, but it is not magic. Uncommitted files, ignored files, databases, cloud resources, and outside messages may not be recoverable through Git.

Destructive commands should never be used as a shortcut around confusion. When the agent encounters work it does not understand, it should stop, inspect, and ask rather than erase.

Connected Tools Create Real Consequences

An agent may be able to send email, comment on an issue, change a calendar, publish a page, charge a card, deploy software, or modify a shared database.

These actions affect people outside the chat. Some cannot be quietly undone.

Before an external action, confirm:



- the target
- the content
- the account being used
- the audience
- the timing
- the cost
- whether the action is a draft or a final submission

“Prepare the message” should normally create a draft. “Send the message” authorizes an external action. The difference matters.

TRY IT

For each action—drafting an email, sending it, scheduling a meeting, changing a password, and publishing a page—decide what the agent may do alone and where approval belongs.



Deciding Whether the Work Is Good

Completion is a claim. The claim becomes trustworthy only when the result can be inspected and the evidence matches what the task required.

Tests Are Evidence, Not Ceremony

An agent should not stop at “the edit looks right.” It should check the result in a way that matches the work.

That might include:

- running focused tests
- checking a calculation independently
- opening the generated document
- viewing the page in a browser
- confirming that a file exists
- comparing before and after behavior
- checking logs for errors
- verifying the external system’s response

A test only proves what it actually tests. One passing test does not prove that the whole project works. A screenshot does not prove that the data is correct. A program starting does not prove that its main task succeeds.

Ask what evidence supports the agent’s final claim.

TRY IT

Match each claim with useful evidence: “the page loads,” “the form submits,” “the total is correct,” and “the message was sent to the right person.” Notice how each claim needs a different check.

Review the Change, Not Just the Summary

Agent summaries are useful, but they are written by the same model that performed the work. Important changes deserve independent inspection.

Review:

- which files changed
- what was added and removed
- whether unrelated areas changed
- what commands ran
- what tests ran
- what was not tested
- any assumptions the agent made
- any warnings or failures

Small diffs are easier to review. A simple request that produces a large rewrite is a reason to slow down.

**TRY IT**

Read an agent's summary, then inspect the actual changed files. List anything visible in the changes that the summary did not mention.

Watch for Scope Drift

Agents often notice nearby problems. Some are worth fixing; others are distractions.

A request to repair one button can turn into a redesign. A request to update a paragraph can turn into a rewritten document. Extra work increases the chance of error and makes review harder.

Useful side findings can be reported without being changed. Finish the agreed task first. Expand the scope only when the added work is clearly wanted.

More Agents Are Not Automatically Better

Several agents can research or work in parallel, but parallel work creates coordination problems:

- two agents may edit the same file
- one may rely on an assumption another changed
- results may conflict
- duplicated work can waste time and money
- a summary may hide disagreement

Separate agents work best on clear, independent tasks with clear ownership. Their results still need to be combined and checked.

Delegation does not remove responsibility. It adds another layer that may need review.

TRY IT

Take a task with three parts. Decide whether the parts are truly independent. If two parts need the same files or depend on the same decision, explain how parallel work could create conflict.

Know When to Stop

An agent can keep trying after the useful path has ended. It may repeat commands, install more tools, widen the search, or make larger changes in an attempt to escape a blocker.

Set reasonable stopping points. Stop when:

- the requested result is complete and checked
- the next step requires new permission
- an important choice belongs to a person
- the same failure keeps repeating
- the evidence is too weak to continue safely
- the task has moved outside the agreed scope

Persistence is valuable. Unbounded activity is not.



Own the Result

The agent can perform the work, but it does not take responsibility for the consequences.

Before using the result, be able to answer:

- What changed?
- Why was it changed?
- What evidence says it works?
- What remains uncertain?
- Who or what could be affected?
- Can the result be reversed?

Do not ship, publish, submit, or depend on work that no one understands well enough to defend.

TRY IT

Explain one agent-made change without using the agent's final summary. Name the goal, the change, the evidence, and one remaining limit.

A Short Working Habit

Before the agent starts:

- define the task and its boundary
- check what the agent can access
- protect secrets and private information
- decide which actions require approval

While it works:

- read permission requests
- notice unexpected scope changes
- keep important decisions somewhere durable
- pause before external or destructive actions

Before accepting the result:

- inspect the actual changes
- review the evidence
- note what was not checked
- make sure the work still belongs to you

An agent can move quickly because it joins language with tools. The tool access creates the power. Careful boundaries, visible evidence, and human ownership make that power useful.