HUMAN.md

A practical guide for humans working with AI agents.

This file is not written for the AI. It is written for you.

AI agents can work fast, generate code, analyse problems, review designs and automate repetitive tasks. But they only work well when you give them clear direction. Vague requests produce vague results. Missing context produces wrong answers. Skipped reviews produce hidden damage.

This guide helps you give better instructions, catch mistakes early, and stay in control of the work.

AI agents are tools. You are the decision-maker.

---

1. What this file is for

Use this file whenever you ask an AI agent to help with work — whether that is writing code, reviewing a design, finding a bug, improving tests, checking security, or preparing a release.

The AI agent reads project files, code and configuration. That gives it useful context. But it does not replace your responsibility to say what you want, what matters, and what must not break.

Why this matters: when things go wrong with AI-assisted work, the cause is usually not the AI model. It is unclear instructions, missing context, or a human who approved something without reading it. This guide exists to prevent those failures.

The agent is not accountable. You are.

---

2. The basic rule

Before you ask the AI to do anything, answer two questions:

> *What outcome do I want? What must not change?*

If you cannot answer both, you are not ready to ask for changes. Start by exploring instead.

Why this matters: AI agents act on what you say, not what you mean. If you say "fix this" without saying what "fixed" looks like, the agent will guess. Its guess may be wrong, and it may change things you did not want changed.

Good starting prompts:

```
Explore the codebase and explain how this feature currently works. Do not change files.
```

```
Review this design and identify missing decisions. Do not rewrite it yet.
```

```
Create a plan first. Wait for approval before editing files.
```

Bad starting prompts:

```
Fix this.
```

```
Make it better.
```

```
Implement the obvious solution.
```

The AI does not know what "better" means unless you define it.

---

3. Choose a work mode first

Every task fits one of five modes. Choosing the right mode before you start prevents the agent from jumping ahead — for example, editing code when you only wanted to understand the problem.

Why this matters: most wasted work happens because the human and the agent are in different modes. You wanted exploration; the agent started implementing. You wanted a plan; the agent started editing files. Naming the mode makes expectations clear.

Explore mode

Use when you do not yet understand the problem or the code.

```
Explore only. Read the relevant files, map the flow, identify likely causes, and report findings. Do not modify files.
```

Plan mode

Use before making meaningful changes. The agent produces a plan; you review and approve it before any files change.

```
Create an implementation plan. Include affected files, risks, tests, and rollback considerations. Do not edit files until I approve the plan.
```

Execute mode

Use after the plan is clear and approved.

```
Implement the approved plan. Keep the change small. Do not modify unrelated files. Run or describe the relevant tests.
```

Review mode

Use after work is done, to check it before you approve it.

```
Review the change critically. Look for skipped requirements, weak tests, security/privacy issues, performance risks, and unintended side effects.
```

Repair mode

Use when something went wrong and you need to stop, understand the damage, and recover.

```
Stop. Summarise what changed, what assumption was wrong, and how to revert or correct it. Do not continue editing until I approve.
```

4. Give the agent a proper brief

When the task is more than a quick question, give the agent structured context. This does not need to be long — but it needs to be clear.

Why this matters: AI agents fill in gaps with assumptions. If you leave out context, constraints or goals, the agent will invent them. Its invented constraints may be wrong, and you will not know until the work is already done.

```
Goal:
[What outcome do we need?]

Context:
[Why are we doing this? What already exists?]

Scope:
[What may be changed? What must not be changed?]

Constraints:
[Any rules: language, framework, security, privacy, performance, deadlines,
compatibility.]

Evidence:
[Attach the actual files, errors, logs, screenshots, reports or test results.]

Definition of done:
[How do we know the work is complete?]

Verification:
[Which tests, checks or manual steps confirm success?]
```

If you skip this, expect the agent to drift from what you actually needed.

---

5. Diagnose before you fix

Humans naturally want to jump straight to "fix it". That instinct often makes things worse, because the root cause is not yet understood.

Why this matters: if you ask the agent to fix a problem it does not understand, it will guess at the cause and apply a plausible-looking fix. If the guess is wrong, you now have the original problem plus a new change to undo.

Use diagnosis first when:

- the root cause is unclear
- multiple parts of the system are involved
- something is slow but you have not measured it
- a security issue is suspected
- the agent already made the problem worse on a previous attempt

Better prompt:

```
Diagnose the problem first. Identify the most likely causes, the evidence for each, and
the smallest safe next step. Do not change code yet.
```

---

6. Make the agent show its confidence level

AI agents sound confident even when they are guessing. They do not naturally separate facts from assumptions. You need to ask for this explicitly.

Why this matters: if you treat a guess as a fact, you may approve a change based on a wrong assumption. Requiring the agent to label its confidence lets you spot where it is uncertain and where you need to verify.

Ask for:

```
Separate confirmed facts, assumptions, hypotheses, and unknowns.
```

For reviews, require evidence labels:

- **Confirmed**: directly verified from the code, logs or documents
- **Inferred**: a reasonable conclusion based on the evidence
- **Hypothesised**: a plausible explanation that has not been confirmed
- **Not verifiable**: cannot be determined from the available information
- **Needs human decision**: the agent cannot make this call for you

Never accept a confident answer that does not say what it is based on.

---

7. Keep changes small

AI agents produce better results with small, focused tasks than with large, open-ended ones.

Why this matters: a large change is harder to review, harder to test, and harder to undo if something goes wrong. Small changes give you checkpoints. If one step is wrong, you lose one step — not twenty.

Prefer:

```
Change only the validation layer. Do not refactor the service layer.
```

```
Add tests for this failure mode before changing the implementation.
```

```
Make the smallest change that fixes the bug.
```

Avoid:

```
Refactor the whole module while you are there.
```

```
Clean this up.
```

```
Make it production ready.
```

Large vague requests create large vague damage.

8. Check what the agent touched

Before you approve any change, check whether the agent modified anything sensitive. AI agents sometimes change more than you asked for, especially when they see related code they think should be "improved".

Why this matters: a small code change that accidentally modifies a login flow, a database structure, or an API response can break things for real users. These areas need careful review, not quick approval.

Watch for changes to:

- login or access control
- database structure
- data encryption or security settings
- what gets written to logs
- deployment or build configuration
- connections to external services
- file formats or data exports
- anything that other systems depend on

For engineering teams:

Also check for changes to public APIs, authorisation models, dependency versions, CI/CD pipelines, infrastructure configuration, database migrations, backwards compatibility contracts, and performance-sensitive code paths.

If the agent touched any of these, slow down. Ask:

```
List all externally visible behaviour changes and compatibility risks.
```

---

9. Never approve what you did not read

It is tempting to approve a change because it "looks reasonable" or because the agent explained it confidently. Do not do this.

Why this matters: AI agents can produce work that looks correct at a glance but contains subtle mistakes — wrong logic, missing edge cases, weakened security, or deleted code that was actually needed. The only way to catch this is to read what changed.

At minimum, review:

- which files changed and why
- what tests were added or changed

- whether security and privacy were considered
- whether the change can be undone if it causes problems

Use this prompt:

```
Before I approve, summarise the change as a reviewer: what changed, why it changed, what risks remain, what tests prove it, and what I should inspect manually.
```

---

10. Treat tests as proof, not paperwork

Tests exist to prove that the code does what it should. Ask the agent to write tests that would actually catch a real problem.

Why this matters: AI agents often write tests that pass but prove nothing — they check that the code runs without crashing, but not that it produces the right results. Weak tests give false confidence. When a bug appears later, you discover that the "passing tests" never actually checked the behaviour that broke.

Good:

```
Add tests that fail before the fix and pass after the fix.
```

```
Add tests for the normal case, invalid input, failure case, and boundary condition.
```

Bad:

```
Add some tests.
```

Ask:

```
Explain why these tests would catch a real regression.
```

For engineering teams:

Watch for these weak test patterns: no assertions, `assert true`, only checks non-null, mocks everything so the test cannot fail meaningfully, duplicates the implementation logic instead of

testing behaviour, only tests constructors, avoids failure paths, or changes existing tests to match broken behaviour instead of fixing the code.

11. Base performance work on measurements

Do not ask the agent to "make it faster" without knowing what is actually slow.

Why this matters: optimising without measuring is guessing. The agent may speed up something that was already fast and ignore the real bottleneck. Worse, some "optimisations" add complexity without measurable improvement. Always measure first, then fix what the measurements reveal.

Use:

```
Profile first. Identify what is actually slow and why. Do not recommend changes without evidence or a plan to verify the improvement.
```

For engineering teams:

Require specific measurements: normal-load and peak-load profiles, 2x peak-load stress test, p95 and p99 latency, memory and CPU ceilings, queue depth or processing lag, database connection usage, first bottleneck identified, and scaling curve. Do not accept generic recommendations like "use caching" or "add parallelism" unless the profiling evidence supports them.

12. Make security and privacy part of the conversation

If the task involves users, personal data, passwords, files, logs, external services, or anything that could be misused, ask for a security and privacy review before writing code.

Why this matters: security and privacy mistakes are expensive to fix after release and can damage trust permanently. AI agents do not automatically consider these risks. If you do not ask, they will not check. A few seconds of prompting can prevent serious incidents.

Use:

```
Before implementing, identify security and privacy risks. What sensitive data is involved? What could go wrong? What protections are needed?
```

For engineering teams:

Specifically check that the agent does not: log secrets or personal data, weaken authentication or authorisation, store tokens in unsafe locations, send sensitive data to external services without disclosure, remove input validation for convenience, hardcode credentials, or hide errors that should be auditable. Require explicit identification of trust boundaries, personal data flows, secret management, and abuse scenarios.

13. Question AI-generated designs carefully

AI-generated designs often look polished and complete, but contain gaps that only become visible during implementation.

Why this matters: a design that reads well but has missing decisions, unverified assumptions, or generic placeholder sections will cause expensive rework later. The more professional a design looks, the more carefully you should check whether the substance matches the presentation.

Common warning signs:

- security or privacy sections that contain only general statements, not specific decisions for this project
- goals stated without a concrete plan to achieve them
- technology choices made without explaining why they fit this problem
- assumptions treated as confirmed decisions
- only the success scenario described — no mention of what happens when things fail

Ask:

```
Review this design for missing decisions, unsupported assumptions, contradictions, and
implementation blockers. Do not rewrite it yet.
```

For engineering teams:

Also watch for: vague non-functional requirements (NFRs) without mechanisms, data models without defined relationships or constraints, APIs without error contracts, queues without backpressure handling, retry logic without idempotency, fashionable architecture patterns (microservices, event sourcing, CQRS) applied without justification, and open questions silently resolved with optimistic assumptions.

14. Interrupt early when things go wrong

If the agent is heading in the wrong direction, stop it immediately. Do not wait to see if it "figures it out".

Why this matters: AI agents build on their own previous steps. A wrong assumption in step 2 gets reinforced in steps 3, 4 and 5. The longer you wait, the more work needs to be undone. Interrupting after one wrong step costs seconds. Interrupting after ten wrong steps costs hours.

Use:

```
Stop. You are solving the wrong problem. Restate the goal, list what you assumed, and
wait.
```

```
Stop. Do not edit more files. Summarise what you changed and why.
```

```
Stop. Revert the last change only and explain the impact.
```

Interrupting early is not rude. It is the single most valuable habit in AI-assisted work.

---

15. One task at a time

Do not ask the agent to redesign, implement, test, secure, optimise and document in a single request.

Why this matters: when you give the agent too many things at once, it rushes through all of them instead of doing any of them well. Quality drops. Details get skipped. You end up reviewing a large messy change instead of several small clean ones.

Break work into phases:

1. Explore — understand the problem
2. Diagnose — find the root cause
3. Plan — design the solution
4. Implement — make a small change
5. Test — prove the change works

6. Review — check for problems
7. Document — record what was decided
8. Repeat

16. Give the agent real evidence, not summaries

Whenever possible, give the agent the actual error message, log file, test output, design document, or screenshot — not your summary of it.

Why this matters: your summary might leave out the detail the agent needs most. A stack trace tells the agent exactly where the error occurred. Your summary of "it crashed on the login page" tells it almost nothing. Real artefacts lead to accurate answers. Summaries lead to guesses.

Examples of useful evidence:

- exact error messages and stack traces
- log output from the actual failure
- screenshots of the problem
- failing test output
- design documents or specifications
- example input and expected output
- previous decisions or constraints

17. State what must not change

Every important task should include rules about what must stay the same. These are your guardrails.

Why this matters: AI agents optimise for the goal you stated. If you do not say "keep the existing API responses unchanged", the agent may happily change them to make the code cleaner. Stating your guardrails prevents the agent from breaking things you care about.

Examples:

```
Existing API responses must remain backwards compatible.
```

```
Do not change the database schema.
```

```
Do not change login behaviour.
```

```
Do not log request bodies.
```

```
The app must work offline after first sync.
```

The AI works better when it knows the boundaries.

---

18. Say clearly when you change direction

Humans change their minds. That is normal and allowed. But the agent does not automatically know your intent changed.

Why this matters: if you give new instructions without saying that the old ones no longer apply, the agent may try to combine both — producing confused, contradictory work. A clear signal saves time.

Use:

```
Change of direction: ignore the previous implementation plan. New goal is...
```

```
Correction: my earlier constraint was wrong. Replace it with...
```

```
Pause the current task. First answer this design question...
```

```
We are switching from exploration to implementation. Use the findings above as context.
```

---

19. Give useful corrections

When the agent gets something wrong, do not just say "wrong". Say why, and give it something to work with.

Why this matters: "wrong" tells the agent nothing about what to do differently. A specific correction gives the agent exactly what it needs to produce a better answer on the next attempt.

Use:

```
I disagree because [reason]. Re-evaluate using this constraint: [constraint].
```

```
Your answer assumes [assumption]. That assumption is false. Rework the plan.
```

```
You optimised for [X], but the priority is [Y]. Try again.
```

The quality of your correction determines the quality of the next answer.

---

20. Answer questions directly

When the agent asks you a question, give a clear answer. Do not tell it to decide for you.

Why this matters: when you say "just do what you think is best", the agent picks a default that may not match your needs. Then you spend time undoing work that was based on a choice you could have made in five seconds.

Bad:

```
Just do what you think is best.
```

Better:

```
Use option B. Constraint: keep the public API unchanged. Trade-off accepted: more
internal complexity is acceptable.
```

If you genuinely do not know:

```
I do not know. Propose the safest default and list what we should verify.
```

21. Know when not to use an AI agent

AI agents are powerful, but they should not operate unsupervised in high-risk situations.

Why this matters: AI agents cannot take responsibility. If the work involves real user data, real money, legal compliance, or irreversible actions, a mistake could cause serious harm that the agent cannot undo or be held accountable for.

Do not use an AI agent for unsupervised work when:

- real user data or production systems are involved
- passwords, keys or secrets are exposed
- legal or compliance judgement is needed
- the action cannot be undone
- infrastructure or databases could be permanently changed
- financial transactions are involved
- user safety is at risk
- nobody can review the output before it takes effect

Use AI to prepare the work. A human must review and approve before it reaches production.

22. Review before you merge

Before any AI-assisted work is merged into the main codebase, require a clear summary of what happened.

Why this matters: once work is merged, it becomes part of the system that everyone depends on. A clear summary helps you (and your team) understand what changed, verify it was done correctly, and know how to undo it if needed.

```
Produce a final review note with:
- summary of the change
- files changed
- requirements addressed
- tests added and run
- risks remaining
- security and privacy impact
- performance impact
- how to undo this change if needed
- anything that was not verified
```

If the agent cannot explain the change clearly, the change is not ready.

---

23. End each session cleanly

At the end of meaningful work, ask the agent to summarise what happened. Then decide what to do with that information.

Why this matters: important decisions and context get lost when they live only in chat history. Chat history disappears or becomes impossible to search. If a decision, assumption, or risk was identified during the session, it should be recorded somewhere permanent.

```
Summarise what was done, what remains open, what decisions were made, what assumptions
were introduced, and what should be added to project documentation.
```

For engineering teams:

Decide what belongs in: `CLAUDE.md` (stable project instructions for the agent), `HUMAN.md` (collaboration rules for humans), design docs, the issue tracker, Architecture Decision Records (ADRs), or tests.

---

24. The human checklist

Before starting:

- Do I know the goal?
- Do I know what must not change?
- Have I provided the relevant evidence?

- Have I chosen the right mode: explore, plan, execute, review, or repair?
- Have I stated what "done" looks like?

Before approving:

- Did I actually read the change?
- Do the tests prove the right behaviour?
- Were security and privacy considered?
- Are assumptions clearly stated?
- Can this change be undone?
- Would I be comfortable explaining this change to a colleague?

Before release:

- Has a human reviewed the work?
- Has it been tested properly?
- Are operational risks understood?
- Are user-facing changes documented?
- Is rollback ready?

---

25. Copy-paste starter prompts

Explore a codebase

```
Explore this codebase for [topic]. Identify relevant files, explain the current flow,
and list risks or unknowns. Do not modify files.
```

Plan a change

```
Create a small implementation plan for [goal]. Include affected files, design decisions,
risks, tests, and rollback considerations. Do not edit files yet.
```

Implement a bounded change

```
Implement the approved plan. Keep the change minimal. Do not modify unrelated files. Add meaningful tests. Preserve existing public behaviour unless explicitly required.
```

Review AI-generated work

```
Review this AI-assisted change critically. Look for skipped requirements, weak tests, security/privacy issues, performance risks, over-engineering, and unintended side effects. Separate confirmed findings from hypotheses.
```

Diagnose a bug

```
Diagnose this bug from the supplied evidence. List likely causes, evidence for each, what to inspect next, and the smallest safe fix. Do not change code yet.
```

Review a design

```
Review this design for missing decisions, unsupported assumptions, contradictions, unclear concepts, security/privacy gaps, operational risks, and implementation blockers. Do not rewrite it yet.
```

Performance review

```
Profile before optimising. Identify the bottleneck using evidence. Include normal load, peak load, and 2x peak load. Recommend only changes tied to measured findings or explicit hypotheses.
```

Stop and recover

```
Stop. Do not continue. Summarise what you changed, what assumption led you there, what risk exists, and how to revert or correct it safely.
```

26. Final principle

The best AI users are not the people who write the longest prompts.

They are the people who:

- give clear intent
- provide evidence
- set boundaries
- interrupt early
- check the output
- record decisions
- take responsibility

AI agents are accelerators. They accelerate good direction and bad direction equally.

Your job is direction.