How I Talk with my AI Agents (Claude Code)

8 min read · 8 hours ago

Andi Ashari Following

talking with Claude Code

How to Use This Guide

- **If you need a quick refresher**, read the *Quick Reference* and copy the templates from *Prompt Snippets*.
- **If you're designing a new agent or workflow**, read *Core Philosophy*, *The Evolution*, and *Context Engineering* patterns.

- **If results are inconsistent**, jump to *Common Pitfalls*, *Debugging Prompts*, and *Evaluation Rubrics*.

> Mental model: You're briefing a brilliant senior engineer with zero institutional memory. Provide clear goals and constraints; let the model choose the HOW.

My personal Global Guide/Rules for Claude Code (~/.claude/CLAUDE.md):
https://gist.githubusercontent.com/aashari/1c38e8c7766b5ba81c3a0d4d124a2f58/raw/b99a150bfc711c2d2ea6cddeb8b41f8231b33f72/00-core-prompt

. . .

Core Philosophy

Trust Intelligence, Provide Clarity

Two truths for great outcomes:

1. **Trust the model's intelligence.** Don't over-specify the HOW; articulate goals and constraints.
2. **Provide crisp context.** Define audience, use-case, success criteria, and constraints up front.

Golden rule: If a colleague with minimal context would be confused by your prompt, the model will be too.

What to always include:

- **Why** the task matters (motivation, success criteria)
- **Who** the audience is (tone, knowledge level)
- **Where** to look (data sources, repos, tools)
- **Constraints** (policies, security, timeline, cost)
- **Output format** (schema, tags, file path)

. . .

The Evolution

From Prompt Engineering to Context Engineering (2025)

Prompt engineering optimized single-turn phrasing. **Context engineering** orchestrates *what* goes into the context window, *when*, and *how it is structured*. Modern agents run multi-step plans, call tools, and maintain state.

Implications:

- Focus on **information curation** over wordsmithing.
- Use **progressive disclosure** (load guidance only as needed).
- Prefer **structured sections/tags** for separation and extraction.
- Treat prompts as **interfaces** between humans, tools, and memory.

. . .

Fundamental Principles

1) Be Clear and Direct

The model does not know your norms by default. Specify:

- Intended use and downstream consumers
- Audience expertise and tone
- “Definition of done” and acceptance criteria
- Hard constraints (policies, budgets, SLAs)

Template:

```
Goal: <what outcome must exist>
Why: <business or user impact>
Audience: <role, expertise>
Constraints: <security, compliance, latency, cost>
Deliverable: <tags/schema/file path>
Success: <objective checks or tests>
```

2) Explain the WHY

Rules without rationale are brittle. Give the motivation so the model can generalize and handle edge cases. (E.g., “No ellipses because TTS pronounces them poorly → also avoid other unpronounceable glyphs.”)

3) Use Examples (Few-/Multi-shot)

3–5 realistic examples beat long expositions. Cover typical, tricky, and edge cases. Keep examples small but faithful.

Example wrapper:

```
<examples>
  <example>
    <input>…</input>
    <output>…</output>
    <notes>what to notice</notes>
  </example>
  …
</examples>
```

4) Let the Model Think (Safely)

Use a **scratchpad** for complex reasoning (research, multi-step plans, reconciliations). Decide whether to **expose** the reasoning to users, **summarize** it, or **keep it internal** depending on product policy and UX. Don’t force verbose thoughts for trivial tasks.

Pattern:

```
<thinking_instructions>
- Use a private scratchpad to plan steps and checks.
- Keep final output concise and polished.
- Surface only summaries of reasoning unless explicitly asked to show details.
</thinking_instructions>
```

5) Structure with XML-like Tags

Tags reduce ambiguity and enable programmatic extraction.

Common tags: `<context>`, `<data>`, `<instructions>`, `<constraints>`, `<thinking>`, `<solution>`, `<checks>`, `<citations>`, `<next_actions>`.

6) Give a Role (System Prompt)

Roles constrain tone and scope (e.g., "You are a CFO in a board meeting" → brevity, metrics, risk).

Tips: specify seniority, industry, and success criteria; include explicit *don'ts* only when necessary.

. . .

Essential Techniques

Sequential Instructions

Number steps to increase compliance and reduce omission.

Skeleton:

```
Your task is to <goal>.
Follow these steps:
1) …
2) …
3) …
4) Output strictly as <tags/schema>.
```

Positive Framing

Say what to do, not just what to avoid. If you must prohibit something, add the rationale.

Progressive Disclosure (Agent Skills)

Organize guidance into *skills* that load on trigger words or task phases. Prevents context bloat, improves reuse.

Example layout:

```
.claude/
  CLAUDE.md                # core principles
  skills/
```

```
debugging/skill.md      # triggers: debug, trace
deploy/skill.md         # triggers: deploy, release
finance/skill.md        # triggers: runway, margin, CAC
```

Parallel vs Sequential Tool Calls

Run independent calls in parallel for latency; run sequentially when outputs feed later steps. Explicitly state dependencies.

Output Contracts

Define machine-readable schemas so downstream systems can consume outputs safely.

JSON contract example:

```
<output_contract>
{
  "summary": "string",
  "insights": [{"title": "string", "evidence": "string"}],
  "actions": [{"owner": "string", "eta": "YYYY-MM-DD"}]
}
</output_contract>
```

. . .

Context Engineering

> The art of deciding what tokens to include, when, and in what structure.

Sources of Truth (ranked)

1. **Running system / live data**
2. **Code & configuration**
3. **Recent logs & monitoring**
4. **Documentation & notes**

When they disagree, trust higher-ranked sources; update the lower ones after confirming reality.

Packing Strategies

- **Front-load invariants:** policies, constraints, output schema.
- **Scope the task:** narrow file paths, APIs, or domains.
- **Stage data:** put only relevant excerpts in `<data>`.
- **Use indices:** table of contents with anchors for long contexts.
- **Chunk + label:** small, labeled sections beat giant blobs.

Progressive Loading

Start minimal; expand based on detected needs ("skills"). Summarize prior steps to free tokens while preserving state.

Memory & State

- **Ephemeral scratchpads** for reasoning and temporary variables.
- **Durable memory** for user preferences, learned constraints, and policies (governed by privacy rules).

Safety & Privacy

- Keep PII and secrets out of public outputs.
- Redact before sharing.
- Prefer references (IDs) to raw data when possible.

. . .

Advanced Patterns

1) Trust Code Over Docs

Verify claims by inspecting code, configs, and runtime behavior. Update docs *after* confirming reality.

2) Research-First Protocol

Phase 1 — Discovery: read notes (as hints), scan code, map the flow, identify external dependencies.

Phase 2 — Verification: validate understanding; list blockers.

Phase 3 — Execution: act autonomously; update docs when done.

3) Iterative Self-Correction

After each meaningful change: self-check, consider side effects, test now, adjust course early.

4) Context Window Management

Grep before opening big files; summarize and prune aggressively; use IDs and references; prefer deltas over full copies.

5) Quality-First Completion

Don't ship partial chains. If A reveals issue B, fix both. Only mark complete when the full chain is healthy.

. . .

Common Pitfalls & Solutions

1. **Emojis in Professional Output** → Model the tone you want; state rationale (professionalism, log readability).
2. **Absolute Commands Everywhere** → Replace with principles + trade-offs; add the WHY.
3. **Vague Language ("make it nice")** → Replace with measurable criteria (e.g., "lint/type-check passes; zero warnings").
4. **Missing WHY** → Add motivation to generalize correctly.
5. **Buried Critical Info** → Front-load Quick Reference, policies, and output schema.
6. **No Examples** → Provide 1–2 per major rule (good vs bad).
7. **Implicit Intent** → Be explicit; if ambiguous, ask.
8. **Over-Prescription** → Trust intelligence; avoid 300-line decision trees when 20 lines of principles suffice.

. . .

Debugging Prompts

Symptoms → Likely Causes → Fixes

- **Hallucinated specifics** → Under-specified sources of truth → Add `<data>` with canonical excerpts; require citations.

- **Partial outputs** → No output contract or step numbering → Add schema + numbered steps; include a final "verify" step.
- **Policy violations** → Missing constraints or rationale → Front-load policies; add "why" and examples of compliant vs non-compliant.
- **Inconsistent style** → Prompt contains conflicting tone → Provide a Style Guide; remove contradictory examples.
- **Tool misuse** → No tool-selection guidance → Add a short tool policy and common mappings.

Minimal Repro Harness

```
<repro>
Task: <one-line>
Context: <smallest data excerpt>
Expected: <short, concrete>
</repro>
```

Run A/B with 2–3 prompt variants; keep the best parts.

. . .

Evaluation Rubrics & Checklists

Rubric (score 1–5 each)

- **Accuracy** (facts & math)
- **Faithfulness** (uses provided context; no fabrication)
- **Completeness** (covers all requested items)
- **Clarity & Structure** (readable, well-tagged, extractable)
- **Safety & Compliance** (policy adherence, privacy)
- **Actionability** (next steps, owners, dates)
- **Efficiency** (latency/cost appropriate)

Ship-Ready Checklist

- Build/lint/type-check clean (if code)
- Output matches contract/schema exactly
- Tests or verifications run and pass
- Edge cases addressed; failure modes explained
- Config aligned across environments
- Security & privacy reviewed
- Docs updated; artifacts linked
- Temporary notes cleaned up

. . .

Real-World Examples

Ready to use global guide/rules (~/.claude/CLAUDE.md)

I have been using this global guide/rule all the time:

https://gist.githubusercontent.com/aashari/1c38e8c7766b5ba81c3a0d4d124a2f58/raw/b99a150bfc711c2d2ea6cddeb8b41f8231b33f72/00-core-prompt

Example 1 — Senior Software Engineer Agent

```
System: You are a Staff-level Software Engineer with autonomy to
read code, propose changes, and implement safe, incremental fixes.

<context>
- Source of truth order: runtime > code/config > logs > docs.
- Repo: <https://example.com/repo>
- Policy: no secrets in logs; PRs must include tests.
</context>

<instructions>
1) Map the execution path for <feature>.
2) Identify root causes for <symptom>.
3) Propose a minimal, testable change.
4) Implement behind a flag; update tests.
5) Produce PR text in <pr> tags; include risks and rollout plan.
</instructions>

<output_contract>
<pr>
- Summary
```

```
- Root cause
- Changes
- Tests
- Risks & rollback
- Migration/flags
</pr>
</output_contract>
```

Example 2 — Financial Analysis Agent

```
System: You are the CFO of a B2B SaaS company preparing a board
update.

<data>
{{Q2_FINANCIALS_CSV}}
</data>

<instructions>
- Analyze revenue drivers, margin shifts, and runway.
- Weigh growth vs burn; propose actions with owners & dates.
- Keep the board narrative to 8 bullets, each with a number.
</instructions>

<output>
<presentation>
- Key insights (with figures)
- Red flags
- Recommendations (owner, ETA)
</presentation>
</output>
```

Example 3 — Multi-Agent Research Orchestrator

```
System: You coordinate specialized agents (code_search,
web_research, data_analysis).

<research_approach>
1) Decompose the question into sub-questions.
2) Assign agents; run independent work in parallel.
3) Synthesize; highlight contradictions and confidence.
4) Propose follow-ups or experiments.
</research_approach>

Question: {{RESEARCH_QUESTION}}
```

. . .

Prompt Snippets (Copy/Paste)

1) Universal Task Skeleton

```
# Task

Goal: …
Why: …
Audience: …
Constraints: …
Deliverable: …
Success: …

# Data
<data>
…minimal, canonical excerpts…
</data>

# Instructions
<instructions>
1) …
2) …
3) …
4) Output strictly in <format> tags.
</instructions>
```

2) Reasoning Scratchpad (Private by Default)

```
<thinking_instructions>
Use a private scratchpad to:
- plan steps & checks
- enumerate edge cases
- design quick validations
Only include a brief summary of reasoning in the final output unless explicitly
</thinking_instructions>
```

3) Style Guide (Professional)

```
<style>
- Concise, declarative sentences
- Active voice; no emojis
- Cite evidence when making claims
```

```
- Prefer lists over long paragraphs when enumerating
</style>
```

4) Tool Policy

```
<tool_policy>
- Prefer specialized tools (read/edit/write/grep/glob) over generic shell.
- Use absolute paths for file ops.
- Run independent tool calls in parallel; sequence dependent ones.
</tool_policy>
```

5) Quality Gate

```
<checks>
- build/lint/type-check clean
- tests pass; new tests added
- no secrets; logs redacted
- performance unchanged or improved
- docs updated
</checks>
```

. . .

FAQ & Edge Cases

Q: Should I always show chain-of-thought?

A: No. Use private scratchpads to improve reasoning; expose summaries unless your UX explicitly benefits from showing detailed steps.

Q: How many examples are ideal?

A: Usually 1–3. Add more only for high-variance formats or policy-heavy tasks.

Q: How do I avoid context bloat?

A: Progressive disclosure: load skills as needed, trim with summaries, and stage data in `<data>` excerpts.

Q: Can I rely on documentation?

A: Treat docs as hints. Verify against code, configs, and runtime signals.

Q: What if user intent is ambiguous?

A: Default to a conservative summary + options; ask a targeted clarifying question.

. . .

Glossary

- **Context Engineering:** Curating which tokens appear in the context window, in what structure, and at which step.
- **Scratchpad:** Private reasoning space; may be summarized in the final output.
- **Progressive Disclosure:** Loading guidance/data only when triggered by need.
- **Output Contract:** A machine-readable schema that downstream systems rely on.
- **Source-of-Truth Hierarchy:** Ordered list of where reality lives (runtime > code > logs > docs).

. . .

Key Takeaways

1. Trust intelligence; provide clarity.
2. Engineer the context, not just the phrasing.
3. Show the WHY; avoid micromanaging the HOW.
4. Use tags, roles, and examples to reduce ambiguity.
5. Think when it helps; keep reasoning private by default; summarize for users.
6. Evaluate with rubrics; finish the whole task chain before shipping.

Claude Code | Prompting | Ai Agent | Artificial Intelligence

Following

Written by Andi Ashari

192 followers · 36 following

Tech Wanderer, Passionate about Innovation and Deeply Expertised in Technology. Actively Redefining the Digital Landscape Through Cutting-Edge Solutions.

No responses yet

Bgerby

What are your thoughts?

More from Andi Ashari

Andi Ashari

Getting Better Results from Cursor AI with Simple Rules

Hey everyone! I've been playing around with Cursor AI lately, and it's been super helpful for coding. But sometimes, it needs a little...

Feb 28 161 8

Andi Ashari

Easy-to-Follow Guide of How to Install PyENV on Ubuntu

Originally posted at: A Beginner-Friendly Guide on How to Install PyENV on Ubuntu

Jan 27, 2024 186 2

Andi Ashari

Installing Jupyter Notebook on Ubuntu 22.04: A Step-by-Step Guide

This page is originally posted on: https://ashari.me/posts/installing-jupyter-notebook-on-ubuntu-22-04-a-comprehensive-guide

Sep 30, 2023 32 4

Andi Ashari

Tracing Bun and ElysiaJS with OpenTelemetry and Datadog

Bun's speed and ElysiaJS's elegance make them a powerful combination for building performant web APIs. But as your application grows...

Aug 14, 2024 6

See all from Andi Ashari

Recommended from Medium

Reza Rezvani

ChatGPT Atlas: OpenAI's AI Browser That Changes Everything

OpenAI just launched an AI-first browser for macOS that integrates ChatGPT directly into your browsing experience—here's what it means...

3d ago 11 2

In Realworld AI Use Cases by Chris Dunlop

The complete guide to Claude Code's newest feature "skills"

Claude Code released a new feature called Skills and spent hours testing them so you don't have to. Here's why they are helpful

4d ago 61 4

Barnacle Goose

How GPT-5-Codex Compares to Claude Sonnet 4.5

The fall of 2025 was marked by the arrival of two contenders from the industry's leading AI labs. On September 15, OpenAI launched...

Oct 17 3

In Dare To Be Better by Max Petrusenko

Claude Skills: The $3 Automation Secret That’s Making Enterprise Teams Look Like Wizards

How a simple folder is replacing $50K consultants and saving companies literal days of work

Oct 17 283 4

In AI Software Engineer by Joe Njenga

Why Claude Weekly Limits Are Making Everyone Angry (And $100/Month Plan Will Not Save You)

Yesterday, I finally hit my weekly Claude limit, and I wasn't surprised, since I see dozens of other users online going crazy over these...

5d ago 119 20

Alebasterr

How We Achieved 95% Context Savings in Claude Code Without Losing a Single Byte of Information

Solving the context crisis with Claude Code's built-in agents.

Oct 12 1

See more recommendations