How to Start with Claude

Claude Tutorial

This tutorial is prepared for anyone interacting with AI with Claude, Cursor & Google AI.

Tutorial

1. Introduction: Beyond the Chat

AI Models Evolve from Chatbots to Agents. Our journey begins with the foundational transformation of Large Language Models (LLMs) into Agentic AI. We will trace the evolution from simple chatbots to sophisticated, tool-using agents capable of complex reasoning and action.

Coding Tutorial

Project Introduction

Welcome to the Claude Tutorial! This project introduces the Claude ecosystem, including Claude Code, Cowork, and Claude Desktop, and explores how Anthropic's models can be used for coding, automation, and AI-assisted workflows.

- Claude

Working Effectively

Learn how to work effectively with Claude Code using the Explore, Plan, Code, Commit workflow, plan mode, extended thinking, and how to provide clear context and verify Claude's output.

1.1 Navigating Tools: Direct Chat & Code

Three ways to get Claude to do your work: chat in the web UI, coding assistance

1. Direct Chat: Text Collaboration (Anti-Thesis to Cursor/Tabs)

Chat with Claude directly in the web interface. Ideal for brainstorming, drafting, and analyzing documents. Unlike Cursor's tab completion, this is a conversational partner for ideation and planning.

Why Use Chat?

- Brainstorming: Explore ideas and concepts before implementation.
- Document Analysis: Upload PDFs, images, or text files for Claude to analyze and summarize.
- Learning: Ask questions about code, technologies, or concepts.
- Quick Prototyping: Generate small scripts or snippets to test ideas.

Claude Projects: Create a project to upload relevant documents, set custom instructions, and maintain context across multiple conversations. Perfect for ongoing research or complex tasks.

2. Claude Code: Terminal Agent

Claude Code is a command-line tool that runs in your terminal. It can read files, write code, run tests, and execute commands. Ideal for developers who prefer terminal workflows.

- Installation: npm install -g @anthropic-ai/claude-code
- Usage: Navigate to your project directory and run claude to start an interactive session.
- Capabilities: Can read and write files, run shell commands, search codebases, and iterate on errors.

3. Cowork: Desktop Agent

Cowork is a desktop application that allows Claude to work on your computer. It can access files, browse the web, and execute tasks in a sandbox environment.

- Access: Available in the Claude Desktop app.
- Capabilities: Can organize files, create documents, and automate repetitive tasks.
- Safety: Runs in a sandbox with limited permissions.

1.2 Plan Ahead: Claude Code's Thinking Mode

Claude Code can think deeply before writing code. Use "plan mode" or "extended thinking" to allow Claude to analyze the problem, consider multiple approaches, and create a plan before implementation.

- Plan Mode: Claude will outline its approach before making changes.
- Extended Thinking: Enable deeper reasoning for complex problems.
- Review: You can review and approve the plan before execution.

1.3 Prompt Engineering: Context & Clarity

Claude's performance depends on the quality of the context you provide. Learn how to write clear prompts, provide relevant context, and iterate on results.

- Be Specific: Clear, detailed prompts yield better results.
- Provide Context: Share relevant files, documentation, and examples.
- Iterate: Refine your prompts based on Claude's responses.

1.4 Model Selection: Balance Cost, Speed & Intelligence

Choose the right model for your task. Claude offers models optimized for different trade-offs between cost, speed, and intelligence.

- Claude Opus: Most capable, for complex reasoning and analysis.
- Claude Sonnet: Balanced performance, for most coding tasks.
- Claude Haiku: Fastest and most cost-effective, for simple tasks.

Learn more

1.5 Advanced Prompting Techniques

1. Be Explicit: Specify Intent

Instead of "fix this," specify the desired outcome: "Fix the bug causing the login form to reject valid emails."

- Bad Example: "Make it better."
- Good Example: "Refactor the function to improve readability by extracting the validation logic into a separate function."

2. Add Context: Why Does It Matter?

- Bad Example: "Add a button."
- Good Example: "Add a button to the dashboard. This is an internal tool used by customer support to refund orders."

3. Iterate: Refine Based on Results

- Bad Example: "Make it better."
- Good Example: "The button is too small on mobile. Increase the size and ensure it meets the minimum tap target size."

4. Use Examples: Show, Don't Tell

- Bad Example: "Format the output."
- Good Example: "Format the output like this: `{"name": "John", "age": 30}`"

5. Set Constraints: Define Boundaries

- Bad Example: "Add a feature."
- Good Example: "Add a feature. Do not modify the database schema."

6. Ask Questions: Clarify Before Acting

- Bad Example: "Do it."
- Good Example: "Before you start, what is your understanding of the problem?"

7. Break Down Complex Tasks

- Bad Example: "Build the entire app."
- Good Example: "Let's start with the user authentication module. Once that is complete, we'll move to the dashboard."

8. Provide Feedback: Guide Learning

- Bad Example: "Do it again."
- Good Example: "Use a more functional approach with pure functions and avoid mutating state."

9. Verify Output: Trust But Check

- Bad Example: "Looks good, ship it."
- Good Example: "Run the tests and show me the output before we commit."

Learn more

1.6 Plan of Action: Prompt Engineering Exercises

Practice applying the principles of clear communication with Claude. Transform vague requests into specific, actionable prompts. Each exercise focuses on a different principle: specificity, context, constraints, or iteration.

Exercise 1: Specificity & Context

Transform a vague request into a specific, actionable prompt by adding relevant details.

- Vague Prompt: "Make the dashboard load faster."
- Your Task: Identify which dashboard, what is slow, and what "faster" means (time, resources, perceived performance).
- Example Improvement: "The analytics dashboard at /admin/analytics takes 5 seconds to load. Optimize the database queries and add loading skeletons."

Exercise 2: Constraints & Boundaries

Add constraints to prevent unwanted changes or scope creep.

- Vague Prompt: "Add a feature."
- Your Task: Specify what not to do, what files to avoid, and what must remain unchanged.
- Example Improvement: "Add a feature. Do not modify the database schema. Do not add new dependencies. Keep the existing API unchanged."

Exercise 3: Iterative Refinement

Practice providing feedback to guide Claude toward better results.

- Vague Prompt: "Do it again."
- Your Task: Specify what to keep and what to change in the next attempt.
- Example Improvement: "Use a more functional approach with pure functions and avoid mutating state."

Exercise 4: Verification & Testing

Ensure Claude verifies its work before you review it.

- Vague Prompt: "Fix the bug."
- Your Task: Add requirements for tests, logs, or proof that the fix works.
- Example Improvement: "Fix the bug. Run the tests and show me the output before we commit."

Learn more

1.7 Working with Code

Core Coding Concepts

Understanding how Claude interacts with your codebase is key to effective collaboration. Learn how Claude reads, writes, and runs code, how it manages context, and how to guide its decisions.

- Context Window: Claude's memory for your conversation and codebase.
- Tool Use: Claude can read files, write files, run commands, and search the web.
- Iteration: Refine Claude's output through feedback and follow-up prompts.

Pro Coding Workflows

- Provide Clear Context: Share relevant files, documentation, and examples.
- Review Changes: Always review Claude's proposed changes before accepting them.
- Test Frequently: Run tests after each change to catch issues early.
- Use Version Control: Commit frequently so you can revert if needed.

Learn more

Claude Memory

Claude remembers your preferences and conversations across sessions. This makes Claude more efficient and personalized over time.

- Preferences: Claude remembers your coding style, publication info, and other preferences you share.
- Conversations: Claude can reference previous conversations when relevant.
- Control: You can view, edit, and delete what Claude remembers in Settings.

Learn more

1.8 Advanced Prompting Techniques

1. Be Explicit: Specify Intent

Instead of "fix this," specify the desired outcome: "Fix the bug causing the login form to reject valid emails."

- Bad Example: "Make it better."
- Good Example: "Refactor the function to improve readability by extracting the validation logic into a separate function."

2. Add Context: Why Does It Matter?

- Bad Example: "Add a button."
- Good Example: "Add a button to the dashboard. This is an internal tool used by customer support to refund orders."

3. Iterate: Refine Based on Results

- Bad Example: "Make it better."
- Good Example: "The button is too small on mobile. Increase the size and ensure it meets the minimum tap target size."

4. Use Examples: Show, Don't Tell

- Bad Example: "Format the output."
- Good Example: "Format the output like this: `{"name": "John", "age": 30}`"

5. Set Constraints: Define Boundaries

- Bad Example: "Add a feature."
- Good Example: "Add a feature. Do not modify the database schema. Do not add new dependencies. Keep the existing API unchanged."

6. Ask Questions: Clarify Before Acting

- Bad Example: "Do it."
- Good Example: "Before you start, what is your understanding of the problem?"

7. Break Down Complex Tasks

- Bad Example: "Build the entire app."
- Good Example: "Let's start with the user authentication module. Once that is complete, we'll move to the dashboard."

8. Provide Feedback: Guide Learning

- Bad Example: "Do it again."
- Good Example: "Use a more functional approach with pure functions and avoid mutating state."

9. Verify Output: Trust But Check

- Bad Example: "Looks good, ship it."
- Good Example: "Run the tests and show me the output before we commit."

1.9 Plan of Action: Prompt Engineering Exercises

Practice applying the principles of clear communication with Claude. Transform vague requests into specific, actionable prompts. Each exercise focuses on a different principle: specificity, context, constraints, or iteration.

Exercise 1: Specificity & Context

Transform a vague request into a specific, actionable prompt by adding relevant details.

- Vague Prompt: "Make the dashboard load faster."
- Your Task: Identify which dashboard, what is slow, and what "faster" means (time, resources, perceived performance).
- Example Improvement: "The analytics dashboard at /admin/analytics takes 5 seconds to load. Optimize the database queries and add loading skeletons."

Exercise 2: Constraints & Boundaries

Add constraints to prevent unwanted changes or scope creep.

- Vague Prompt: "Add a feature."
- Your Task: Specify what not to do, what files to avoid, and what must remain unchanged.
- Example Improvement: "Add a feature. Do not modify the database schema. Do not add new dependencies. Keep the existing API unchanged."

Exercise 3: Iterative Refinement

Practice providing feedback to guide Claude toward better results.

- Vague Prompt: "Do it again."
- Your Task: Specify what to keep and what to change in the next attempt.
- Example Improvement: "Use a more functional approach with pure functions and avoid mutating state."

Exercise 4: Verification & Testing

Ensure Claude verifies its work before you review it.

- Vague Prompt: "Fix the bug."
- Your Task: Add requirements for tests, logs, or proof that the fix works.
- Example Improvement: "Fix the bug. Run the tests and show me the output before we commit."

Learn more

1.10 Working with Code

Core Coding Concepts

Understanding how Claude interacts with your codebase is key to effective collaboration. Learn how Claude reads, writes, and runs code, how it manages context, and how to guide its decisions.

- Context Window: Claude's memory for your conversation and codebase.
- Tool Use: Claude can read files, write files, run commands, and search the web.
- Iteration: Refine Claude's output through feedback and follow-up prompts.

Pro Coding Workflows

- Provide Clear Context: Share relevant files, documentation, and examples.
- Review Changes: Always review Claude's proposed changes before accepting them.
- Test Frequently: Run tests after each change to catch issues early.
- Use Version Control: Commit frequently so you can revert if needed.

Learn more

Claude Memory

Claude remembers your preferences and conversations across sessions. This makes Claude more efficient and personalized over time.

- Preferences: Claude remembers your coding style, publication info, and other preferences you share.
- Conversations: Claude can reference previous conversations when relevant.
- Control: You can view, edit, and delete what Claude remembers in Settings.

Learn more

1.11 Advanced Prompting Techniques

1. Be Explicit: Specify Intent

Instead of "fix this," specify the desired outcome: "Fix the bug causing the login form to reject valid emails."

- Bad Example: "Make it better."
- Good Example: "Refactor the function to improve readability by extracting the validation logic into a separate function."

2. Add Context: Why Does It Matter?

- Bad Example: "Add a button."
- Good Example: "Add a button to the dashboard. This is an internal tool used by customer support to refund orders."

3. Iterate: Refine Based on Results

- Bad Example: "Make it better."
- Good Example: "The button is too small on mobile. Increase the size and ensure it meets the minimum tap target size."

4. Use Examples: Show, Don't Tell

- Bad Example: "Format the output."
- Good Example: "Format the output like this: `{"name": "John", "age": 30}`"

5. Set Constraints: Define Boundaries

- Bad Example: "Add a feature."
- Good Example: "Add a feature. Do not modify the database schema. Do not add new dependencies. Keep the existing API unchanged."

6. Ask Questions: Clarify Before Acting

- Bad Example: "Do it."
- Good Example: "Before you start, what is your understanding of the problem?"

7. Break Down Complex Tasks

- Bad Example: "Build the entire app."
- Good Example: "Let's start with the user authentication module. Once that is complete, we'll move to the dashboard."

8. Provide Feedback: Guide Learning

- Bad Example: "Do it again."
- Good Example: "Use a more functional approach with pure functions and avoid mutating state."

9. Verify Output: Trust But Check

- Bad Example: "Looks good, ship it."
- Good Example: "Run the tests and show me the output before we commit."

1.12 Plan of Action: Prompt Engineering Exercises

Practice applying the principles of clear communication with Claude. Transform vague requests into specific, actionable prompts. Each exercise focuses on a different principle: specificity, context, constraints, or iteration.

Exercise 1: Specificity & Context

Transform a vague request into a specific, actionable prompt by adding relevant details.

- Vague Prompt: "Make the dashboard load faster."
- Your Task: Identify which dashboard, what is slow, and what "faster" means (time, resources, perceived performance).
- Example Improvement: "The analytics dashboard at /admin/analytics takes 5 seconds to load. Optimize the database queries and add loading skeletons."

Exercise 2: Constraints & Boundaries

Add constraints to prevent unwanted changes or scope creep.

- Vague Prompt: "Add a feature."
- Your Task: Specify what not to do, what files to avoid, and what must remain unchanged.
- Example Improvement: "Add a feature. Do not modify the database schema. Do not add new dependencies. Keep the existing API unchanged."

Exercise 3: Iterative Refinement

Practice providing feedback to guide Claude toward better results.

- Vague Prompt: "Do it again."
- Your Task: Specify what to keep and what to change in the next attempt.
- Example Improvement: "Use a more functional approach with pure functions and avoid mutating state."

Exercise 4: Verification & Testing

Ensure Claude verifies its work before you review it.

- Vague Prompt: "Fix the bug."
- Your Task: Add requirements for tests, logs, or proof that the fix works.
- Example Improvement: "Fix the bug. Run the tests and show me the output before we commit."

Learn more

2. Conclusion: A New Kind of Collaboration

Through this tutorial, you've learned that Claude Code is a fundamental shift from chat to a tool-using, plan-based, and iterative AI collaboration partner. The key takeaways are: Context is Key, Plan Before Coding, Iterate Based on Results, and Verify Output. The goal is not to replace your skills, but to amplify them, allowing you to focus on high-level architecture and creative problem-solving.

Ready to Accelerate Your Workflow? Start with a simple task in your current project. Apply the principles you've learned. The more you practice, the more intuitive this collaboration will become.

Start Your Claude Journey