How to *Work* with **Claude**

The complete guide to maximizing productivity with Claude Code & Claude AI

Claude Sonnet 4.5 | December 2025

🚀 Getting Started

Project Initialization

When starting a new project, always use the **"/init"** command. This command creates a "Claude.md" document containing your project documentation for easy reference.

```
→ /init
```

Working Effectively

After initializing your project, always ask Claude Code to *explain the project thoroughly*. Request details about each component, their functionalities, and how they interconnect with other parts of your project.

🎯 Working with *Code*

For Complex Changes

If you're planning a **major change**, such as large-scale refactoring or adding complex new functionality, always instruct Claude Code to outline a plan before implementing any changes. Encourage deeper thinking by using phrases such as:

Think hard | Think deep | Think longer

After reviewing Claude's proposed plan, if you agree, proceed directly with the changes. If not, request revisions to the plan.

For Smaller Changes

For smaller changes, such as *simple, single-file updates* or straightforward features, clearly specify the exact file you want to modify. For example:

```
→ Claude, can you increase the size of the chat input in the file "user/page.tsx"?
```

```
→ Claude, can you add error handling to "data_processing.py"?
```

```
→ Claude, please refactor the `calculate_totals` function in "invoice.py"
```

BEST PRACTICES

Other Best Practices

- Regularly use the **/compact** command during development. This helps Claude Code remain efficient and reduces the risk of confusion in long-term projects.
- Always be **explicit about which files** you intend to change and clearly state their relationships to other components.
- Frequently run tests. Claude Code excels at validating new functionality through thorough testing, ensuring reliability and accuracy.
- Periodically ask Claude Code to *review the entire project*, assessing logic consistency and identifying any redundant or inefficient code.

💡 Advanced *Prompting* Techniques

1. Be Painfully Specific

Say exactly what you want (format, tone, length, audience).

✓ **Good Prompt:**
"Write a 300-word blog post for tech-savvy millennials in a casual, humorous tone about the benefits of mechanical keyboards, using bullet points for key features."

✕ **Bad Prompt:**
"Write something about keyboards."

2. Tell It Why You Want It

A one-line reason improves results fast.

✓ **Good Prompt:**
"Summarize this legal document. This is for a non-lawyer executive to understand the key risks."

✕ **Bad Prompt:**
"Summarize this legal document."

3. Your Examples Are The Truth

If you show a sample, Claude copies it.

✓ **Good Example:**
"Here is a well-formatted customer service response. [Sample] Now write a response to this new query in the same style."

✕ **Bad Example:**
"Write a response like this one: [A poorly written, rude email]."

4. Big Projects? Work in Small Checkpoints

Force "step-by-step progress," not "do everything at once."

✓ **Good Strategy:**
"First, outline the main sections of the report. [Claude provides outline]. Now, write the introduction based on that outline."

✕ **Bad Strategy:**
"Write a complete 50-page report on climate change."

5. If You Use an Agent Workflow, Say So

Tell it whether context will be compacted / saved to files.

✓ **Good Prompt:**
"You are an agent in a multi-turn workflow. At the end of each turn, summarize your progress and save the current state to a file named 'state.json'."

✕ **Bad Prompt:**
"Just do the task."

6. Want Action, Not Advice? Say "Do It"

If you want edits/implementation, explicitly request that behavior.

✓ **Good Prompt:**
"Here is some code. Refactor it to improve performance and readability. Return only the refactored code."

✕ **Bad Prompt:**
"What do you think of this code?"

7. Control Formatting by Saying What to Do

Instead of "don't use markdown," say "write 3 short paragraphs" or "return JSON."

✓ **Good Prompt:**
"Output the final answer as a JSON object with keys 'summary' and 'key_points'."

✕ **Bad Prompt:**
"Don't use any special formatting or markdown."

8. Use Simple "Labels" (XML tags) to Steer Behavior

Recommend lightweight tags to set defaults like "be proactive" vs "be cautious."

✓ **Good Prompt:**
"You are a helpful assistant. <persona>Be proactive and offer suggestions beyond the user's direct request.</persona>"

✕ **Bad Prompt:**
"Be proactive."

9. If 'Thinking' Is Off, Avoid the Word 'Think'

Claude models can be weirdly sensitive to "think"—use "consider / evaluate" instead.

✓ **Good Prompt:**
"Before answering, please consider the user's request and evaluate the best approach."

✕ **Bad Prompt:**
"Think about the user's request before answering."

⚡ Essential *Claude* Features

📚 Skills System

Specialized knowledge folders (docx, pptx, xlsx, pdf, frontend-design) that dramatically improve output quality. Let Claude read relevant SKILL.md files before creating documents for professional-grade results.

🔍 Web Search & Tools

Claude can search the web, fetch URLs, access Google Drive, and use integrated tools for research and current information. Essential for staying up-to-date and accessing your company resources.

🧠 Memory System

Claude remembers context from past conversations, making every interaction feel informed by shared history. Your ongoing projects benefit from continuity across sessions.

📄 File Creation

Create actual files—presentations, documents, spreadsheets, PDFs—not just content previews. Claude generates professional deliverables you can immediately download or share.

🎯 Strategic Planning

For complex multi-tool tasks, Claude develops comprehensive research plans before executing. Particularly valuable for transformation work requiring multiple data sources.

💾 200K Context Window

With a 200K token context window, Claude can handle entire codebases, multiple documents, or extensive conversations without losing context.

✨ Best *Practices*

LEVERAGE SKILLS

Read Skills Before Creating

Before creating documents, presentations, or complex interfaces, **explicitly ask Claude to read the relevant skill documentation**. This ensures you get the highest quality, most professional outputs.

```
→ Please read the pptx skill, then create a presentation about our Q3 results
```

```
→ Read the docx skill and create a project proposal
```

ORGANIZATION

Use Projects for Context

For ongoing work, use *Claude Projects* to maintain context, custom instructions, and relevant files. This creates a dedicated workspace with persistent memory and knowledge.

🎨 Getting the Best *Output*

For Documents & Presentations

When creating professional documents, **always reference the skills system:**

```
→ "Read the docx skill and create a project proposal"
```

```
→ "Use the pptx skill to make a 10-slide investor deck"
```

This ensures Claude uses battle-tested best practices for formatting, structure, and visual design.

For Research & Analysis

Combine tools strategically:

- Use **web_search** for current information and industry trends
- Use **web_fetch** to read complete articles and reports
- Use **Google Drive** or internal tools for company-specific data
- Ask Claude to synthesize findings from multiple sources

This guide reflects Claude's capabilities as of December 2025

Made with Claude Sonnet 4.5 by Simone Leonelli