How to *Work* with Claude

This handbook guides you through Claude Code, Anthropic's AI coding assistant, from basics to advanced techniques.

Anthropic | Claude Code

Introduction: Beyond the Hype

Coding assistants are everywhere. It's easy to feel like there's a new one launched almost every week, each promising the ultimate boost to your productivity. Amidst this constant stream of tools, Claude Code stands out for its deep integration with the terminal and its focus on agentic capabilities.

But that's the catch: using it effectively requires a bit of a rethink.

This guide isn't about boilerplate prompts or magic bullets. It's about understanding the core philosophy, Claude Code's specific strengths and how to adapt your workflow to leverage its unique capabilities. We'll cover foundational concepts, practical techniques, and crucial considerations for collaborating with this powerful assistant.

Getting Started

Project Initialization

After starting a new project, always run the `/init` command. This command creates a `CLAUDE.md` file containing project-specific information and conventions Claude needs to know.

```
/init
```

Working Effectively

After initializing your project, begin a session with Claude in `plan mode` (shortcut `shift+tab`). Request details about each component, their functionality, and how they interconnect with other parts of your codebase.

5 Surprising Truths About Claude Code

These insights will change how you work with Claude Code.

1. Doesn't Index Your Codebase (And That's a Good Thing)

This is a deliberate choice for security and simplicity. Instead of indexing, Claude Code understands your project by intelligently reading files as needed, leveraging its agentic capabilities to navigate your codebase on demand.

Why This Matters: This approach ensures maximum security, as your code isn't stored or processed by external indexing systems. It also avoids the complexity of managing an index, making the tool more lightweight and adaptable to different project structures.

Pro Tip: To help Claude Code understand your codebase more quickly, you can manually guide its exploration by asking it to execute specific commands (like `ls -R` or `tree`) or to read certain files relevant to the task at hand.

2. It's Not Just a Coder, It's a Terminal Wiz

While Claude Code is adept at coding, its strength lies in its ability to execute terminal commands. This allows it to interact with your system, install dependencies, run tests, and even perform more complex operations like managing files or interacting with version control.

Example Workflow: You can instruct Claude Code to download a GitHub pull request, run the test suite, and then fix any failing tests. This demonstrates its capability to handle a complete development workflow beyond just writing code.

3. The Best Time to Use It Might Be Before You Write Code

Since the core value of Claude Code lies in its ability to act as a "**through-worker**" for your terminal, it can be particularly useful for tasks that don't involve immediate coding.

Consider Using It For: Tasks like analyzing logs, managing files, or performing system administration. Its ability to understand context and execute commands makes it a powerful assistant for these non-coding activities.

4. A Simple Markdown File Acts as Its Shared Memory

The `CLAUDE.md` file serves as a persistent memory for Claude Code, storing important information about your project, coding conventions, and commands you frequently use. **Minimize the noise in this file.**

Best Practices: Keep the file concise and focused on the most relevant information. Regularly review and prune outdated entries. The file can be part of version control, allowing you to share it with your team.

What to Include (or Exclude):

- Dependency install commands
- Project-specific aliases and shortcuts
- Coding style preferences (e.g., "use ES6 syntax")
- Common commands and workflows
- Architectural patterns and decisions

```
# Common commands
npm run build
npm run test:unit
npm run test:e2e
npm run dev:server
# Code style preferences
Use ES6 syntax.
Prefer functional programming patterns where appropriate.
Run linter before committing changes.
```

5. Sites Jitter Trick Are Long Conversations

Anthropic offers two ways to access Claude Code: through its subscription plan or via the **`anthropic-sdk-cli`** for API access.

Longer tasks sometimes hit **session timeouts**. Use the `--resume` flag to pick up where you left off.

Accelerated Consideration: Remember that Claude Code's strengths lie in its ability to act as a "through-worker" for your terminal, handling tasks beyond just coding. Consider how this capability might be leveraged for tasks like log analysis, system administration, or managing complex workflows.

```
# Start a new session
claude "Implement feature X"
# Resume the last session if it timed out
claude --resume
```

Working with *Code*

Set Complexity Changes

Prefix prompts with **`think`** keywords to trigger extended reasoning, allowing Claude more time to consider complex problems. More detailed requests like `think hard` or `think deeply` will trigger even more thorough analysis.

```
think
think hard
think deeply
```

Pro Tip: Use these keywords for tasks requiring careful planning, debugging complex issues, or evaluating different approaches.

For Smaller Changes

For quick, small changes, you can use specific shortcuts for straightforward fixes. Learn more about the modes here.

Example: To add a comment, you might type `# Add a comment to the file` instead of a full prompt.

```
# Add a comment to the file
# Create a function that...
```

Other Best Practices

- **Provide Clear Context:** The more context you provide, the more accurate and relevant Claude's responses will be.
- **Break Down Complex Tasks:** For large or complex tasks, consider breaking them into smaller, more manageable steps.
- **Iterate and Refine:** Don't expect perfection on the first attempt. Be prepared to refine your prompts based on Claude's initial responses.
- **Specify Output Formats:** If you need Claude to generate code in a specific format, clearly specify the desired output format in your prompt.

Advanced *Prompting* Techniques

1. Be Positively Specific

Focus on what you want Claude to do, rather than what you don't want it to do.

Example:

> Instead of "Don't use jQuery", try "Use vanilla JavaScript for DOM manipulation."

2. Tell It Why You Want It

Provide context by explaining the purpose behind your request.

Example:

> "Refactor this function to improve readability. The current implementation is difficult to understand."

3. Your Examples Are The Tools

Provide concrete examples to guide Claude's behavior.

Example:

> "Convert these date formats to ISO 8601: 'MM/DD/YYYY' → 'YYYY-MM-DD'"

4. Big Projects? Work in Small Checkpoints

Break tasks into manageable chunks for better results.

Example:

> "Let's first create the basic structure. Next, we'll add error handling. Finally, we'll optimize performance."

5. It Sees Code as Input Workflows, Say So

Clarify that Claude should treat code as input for analysis.

Example:

> "Analyze this code for potential security vulnerabilities. Do not execute it."

6. Wrong Actions, Now Instructed Say "Do It"

Explicitly instruct Claude when you want it to take action.

Example:

> "Fix the bug in this function." vs. "Explain why this function is causing the bug."

7. Control Formatting by Saying What to Do

Specify the desired output format in your prompt.

Example:

> "Output the results as a JSON object with keys 'errors' and 'warnings'."

8. Can Simple "Lazier" LLMS Lead to Smarter Behavior

Minimal prompts may sometimes lead to unexpected "smarter" behavior.

Example:

> "Fix the bug." vs. "Fix the bug in the file `utils.py` by adding a try-except block around the file I/O."

9. Encourage "Chains of Thought"

Ask Claude to think step-by-step for complex tasks.

Example:

> "Debug this step-by-step, before you fix the bug mentioned in the issue."

Real-World Use Cases

Content Writing
Generating blog posts, articles, or marketing copy.

Learning Understanding
Asking Claude to explain complex concepts, code snippets, or documentation.

Code Development
Using Claude for tasks like refactoring, debugging, or generating boilerplate code.

Data Analysis
Leveraging Claude to analyze data, extract insights, or generate reports.

Research & Synthesis
Utilizing Claude to gather information from various sources, synthesize findings, and present a coherent summary.

Conversation
Engaging in open-ended conversations for brainstorming, exploring ideas, or simply learning.

Essential *Claude* Features

Basic Software
Claude Code's core functionality, including file reading, editing, and command execution, forms the foundation for its more advanced features.

Plan Mode
Enabling Claude to create a detailed plan before taking any actions, allowing you to review its proposed approach and ensure it aligns with your expectations.

Thinking Modes
Different levels of "thinking" (`think`, `think hard`, `think deeply`) allow you to control the depth of Claude's reasoning process, with more complex tasks benefiting from more thorough analysis.

Slash Commands
Custom shortcuts (`/init`, `/resume`, `/clear`) for frequently used prompts or actions, streamlining your workflow.

Tools
Extensions that allow Claude to interact with external systems and perform actions beyond simple text generation, such as accessing APIs or databases.

Model Context Protocol (MCP)
A standardized protocol for connecting Claude to external data sources and tools, enabling more complex and context-aware interactions.

Plug-in Ecosystem
Leveraging community-developed plugins to extend Claude Code's capabilities with pre-built integrations and functionalities.

Hooks
Automated scripts that can be triggered by specific events, such as file modifications or commands, enabling automated workflows and customizations.

GitHub Code Review
Integration with GitHub pull requests, allowing Claude to analyze code changes, suggest improvements, and identify potential issues.

Claude Code SDK
Software development kits (SDKs) for integrating Claude Code functionality into your own applications and workflows.

Headless Mode
Running Claude Code without a graphical interface, useful for automated tasks and integrations with other tools.

Engineering Pro-Tips

Advanced techniques leaning on Anthropic's own engineering teams.

Multi-Claude Approach
Don't just use one agent. Anthropic's engineers frequently run multiple Claude instances simultaneously:

- **Code Writer:** Focuses solely on generating code.
- **Code Reviewer:** Analyzes the generated code for potential issues and suggests improvements.
- **QA Tester:** Runs tests and identifies bugs.

This mirrors a human team, and it's surprisingly effective at leveraging Claude's strengths in specialized, independent tasks.

Slash Commands
Custom shortcuts (`/commands`) for frequently used prompts or actions, streamlining your workflow.

Elegant Command with Hooks
For complex custom setups, GitHub Actions offers a powerful way to automate Claude's interaction with your codebase.

```
# .github/workflows/claude-code.yaml
name: Claude Code Assistant
on:
  issue_comment:
    types: [created]
  pull_request_review_comment:
    types: [created]
jobs:
  claude:
    runs-on: ubuntu-latest
    steps:
      - uses: anthropic-ai/claude-code-action@v1
        with:
          anthropic_api_key: ${{ secrets.ANTHROPIC_API_KEY }}
          timeout_minutes: "60"
```

This setup allows Claude Code to respond to issue comments and pull request reviews, making it a valuable tool for collaborative development.

Scale Yourself: Subagents & MCP

Stop trying to do everything in one context window.

Use Subagents

The Mental Model
Think of Claude Code's subagents like a team of specialists. Instead of one generalist, you have:

- **Subagents** as your specialists.
- **Main Agent** as the project manager, coordinating between specialists.

Each subagent has its own context window, tools, and specialized prompt.

Why This Matters
Context is a finite resource. Subagents let you offload specific tasks without polluting your main conversation with their intermediate work.

Create Subagents

```
/agents
```

Example Subagents

- **Code Reviewer:** Reviews code for quality and security.
- **Test Runner:** Runs tests and fixes failures.
- **Debugger:** Investigates bugs and root causes.

Claude will delegate to these specialists when appropriate.

Use MCP

Connect Claude to external tools and data sources.

```
# Add a server
claude mcp add --transport http notion https://mcp.notion.com/mcp
# Authenticate
/mcp
```

You can ask Claude to use specific tools: "Use the Notion MCP to create a page from this PRD."

Popular MCPs

- Notion
- Stripe
- Sentry
- Your own custom tools

Learning Resources & Next Steps

Official Training Courses

Get to the crew with comprehensive Anthropic-backed courses. They offer development guidance on prompt engineering, tool integration, and building production-ready AI applications.

- **Claude Code: A Highly Agentic Coding Assistant**
  Learn to code with Claude through collaboration, exploration, and iteration.
  Anthropic | 1 hour | Intermediate
  Enroll for Free
- **Prompt Engineering with Claude**
  Learn prompt engineering techniques to optimize Claude's performance for your specific tasks.
  Anthropic | 10 hours | Intermediate
  Enroll for Free
- **MCP: Introduction to Model Context Protocol**
  Learn how to connect Claude to external tools and data sources using MCP.
  Anthropic | 1 hour | Intermediate
  Enroll for Free

Total: 24 hours of content

Three Learning Paths

- **Beginner Journey**
  Start developing Claude-powered applications with comprehensive MCP guides and best practices for tool integration.
- **Advanced Mastery**
  Master production-ready AI agents with advanced MCP implementations and enterprise-level security and performance optimization.
- **Quick Wins**
  Explore ready-to-deploy MCP servers and templates for immediate productivity with enterprise tools.

Beyond the Fundamentals

Explore how to leverage Claude's capabilities in more complex and specialized ways.

Getting the Best *Output*

For Research & Analysis

- Use web search to stay current with information and research trends.
- Use **`think`** modes for complex analysis and strategic planning.
- Use **`/resume`** to maintain context across multi-session research projects.
- Combine Claude's reasoning capabilities with its tool integrations.

Conclusion: A New Kind of Collaboration

Claude Code represents a shift from traditional AI coding assistants. It's designed to be more of a collaborative partner, leveraging its agentic capabilities and deep integration with your development environment to tackle complex tasks and boost your productivity. Remember, the key to effective collaboration with Claude Code lies in understanding its strengths and adapting your workflow to leverage its unique capabilities.

> "We strongly encourage teams that are serious about adopting agentic coding to invest in prompt engineering and tool integration. The ROI can be significant."
>
> — Anthropic Engineering Team

Explore Claude Code's capabilities and start boosting your coding efficiency today!

Visit Anthropic's Claude Code Documentation