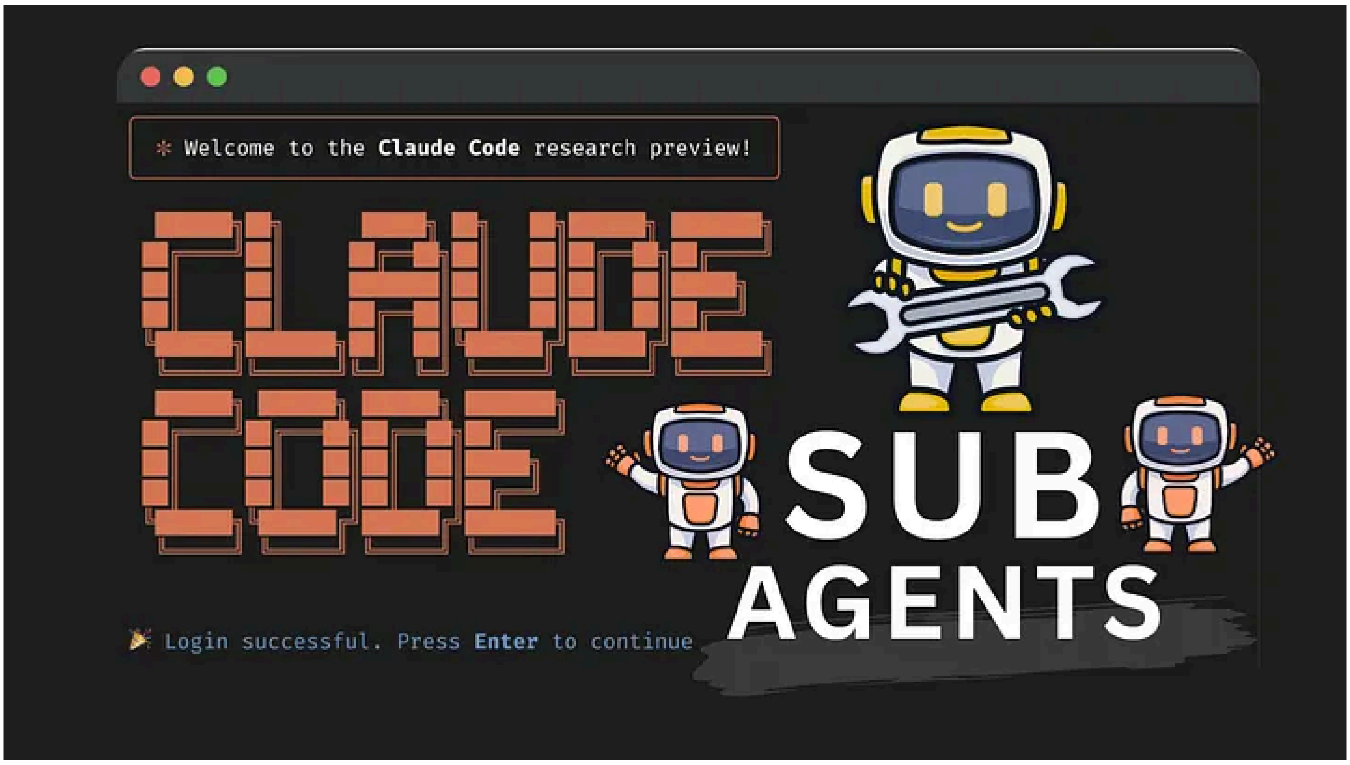
How I'm Using Claude Code Sub Agents (Newest Feature) As My Coding Army


✻ Welcome to the Claude Code research preview!
CLAUDE CODE
SUB AGENTS
Login successful. Press Enter to continue

Claude Code Sub Agents- Featured Image / By Author

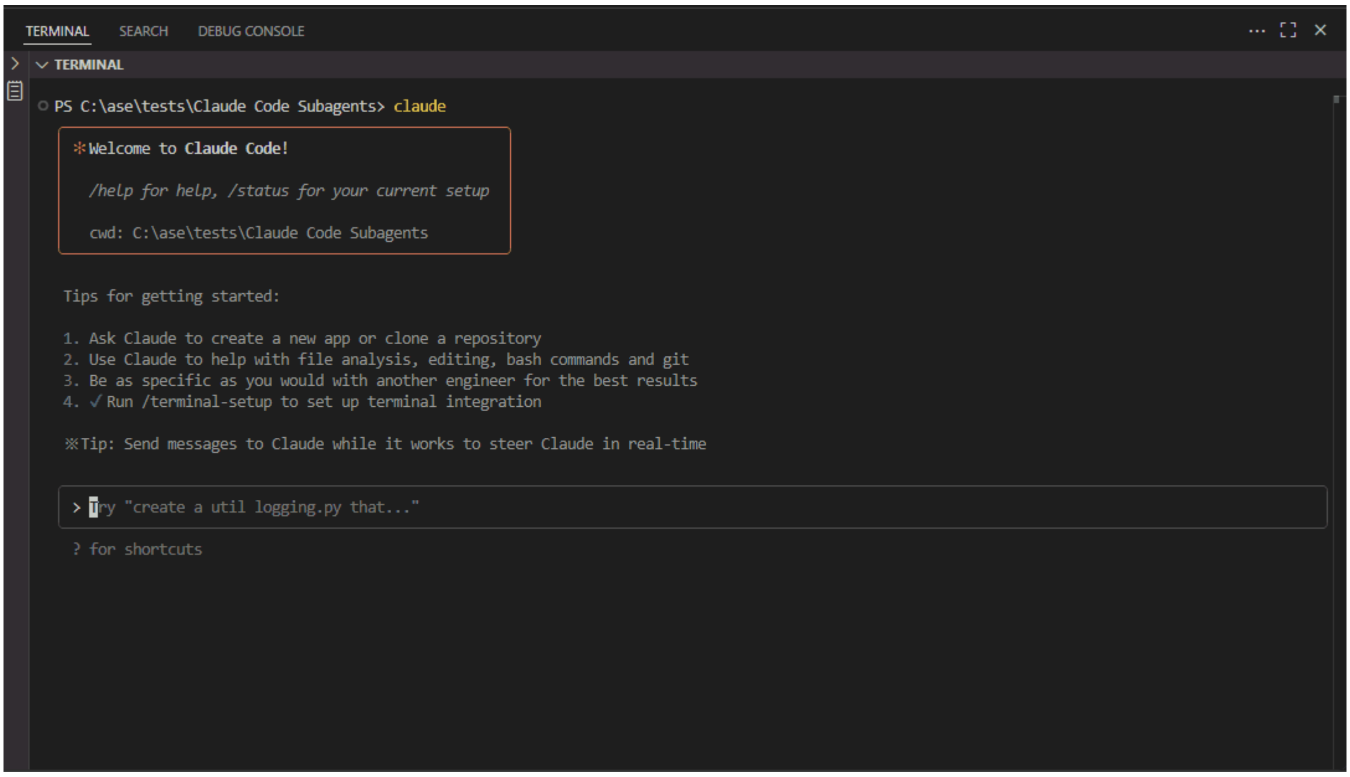
Claude Code just added sub-agents, and they are taking the game to the next level.

> ***If you have not tried them, I will guide you in detail and show you what you are missing.***

I have just tested it, and it's mind-blowing how you can now build **specialized AI team members** that handle different aspects of your development workflow.

> **Claude Code subagents are specialized assistants that you can create for specific tasks.Each subagent operates with its own context window, custom system prompt, and specific tool permissions.**

This feature gives you unprecedented control over your development process, allowing you to delegate specific workflows, such as frontend development, backend APIs, code reviews, and research, to dedicated AI specialists that you've personally trained.



```
PS C:\ase\tests\Claude Code Subagents> claude

 ✻ Welcome to Claude Code!

   /help for help, /status for your current setup

   cwd: C:\ase\tests\Claude Code Subagents

 Tips for getting started:

 1. Ask Claude to create a new app or clone a repository
 2. Use Claude to help with file analysis, editing, bash commands and git
 3. Be as specific as you would with another engineer for the best results
 4. ✓ Run /terminal-setup to set up terminal integration

 ※Tip: Send messages to Claude while it works to steer Claude in real-time

 > Try "create a util logging.py that..."

   ? for shortcuts
```

Claude Code Sub Agents Command Demo

> ***Quick Note:*** *If you are not a premium member, you can read the story here.* **Consider supporting me by following me on Medium** *and* **my YouTube channel** *to learn more about Claude Code.*

The idea of using Claude Code sub-agents is about delegating your workflow tasks to specialized AI team members

Instead of one general AI agent that tries to do everything, you can now build:

- A **Frontend Specialist** who knows your component patterns and styling preferences
- A **Backend Architect** who understands your API structure and database schemas
- A **Code Reviewer** that enforces your coding standards automatically
- A **Research Assistant** who finds and summarizes the latest tech documentation
- A **Production Validator** that catches placeholder code and security issues

Each subagent can be configured with specific tools they're allowed to use and detailed instructions about how they should work.

When Claude Code encounters a task that matches a subagent's expertise, it delegates that work to your specialized agent.

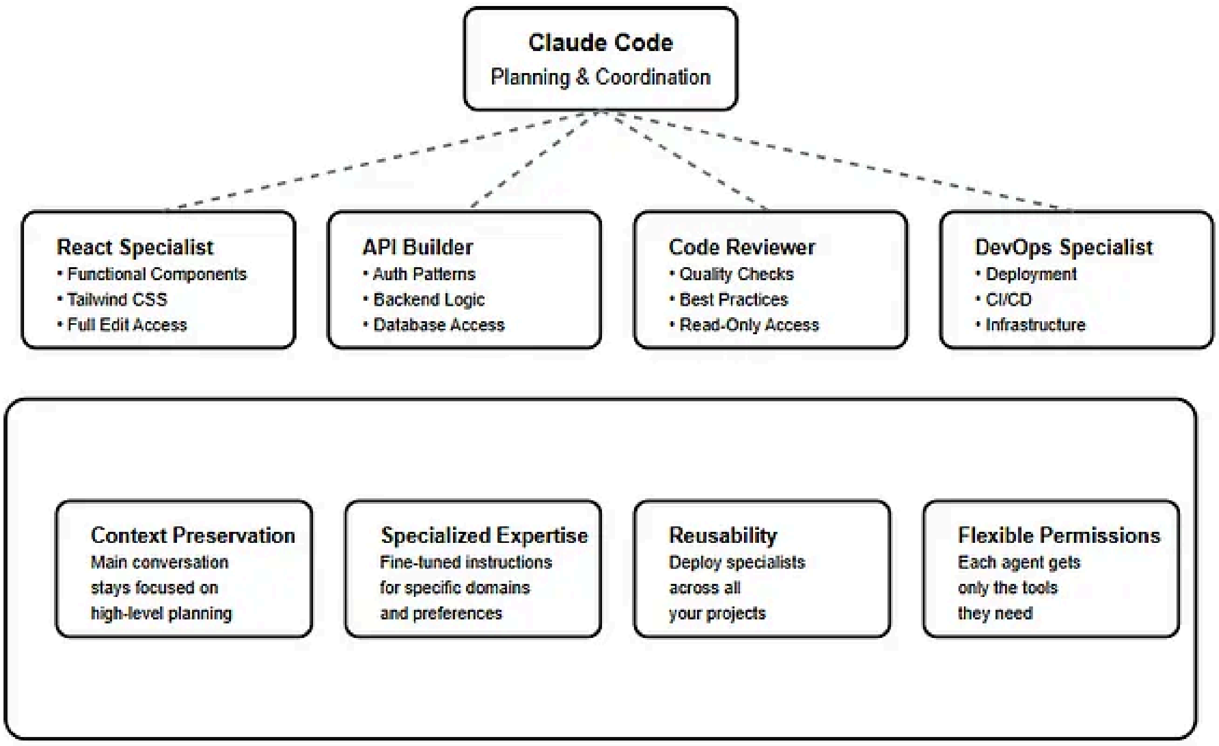
In this article,

I will take you through how subagents work, share practical examples you can implement immediately, and demonstrate the configuration options available for building your coding army that works how you want it to.

I will take you through from the start to advanced use of Claude Code subagents, just as I did for Claude Code hooks in this article — ***How I'm using Claude Code Hooks to Fully Automate My Workflow.***

Now, let's break down what makes subagents so powerful.

Understanding Claude Code Subagents



Claude Code Subagents Illustration / By Author

Using Claude Code Subagents is like hiring specialized developers for your team, with specific expertise areas, tools, and working styles.

Each subagent ***operates in its isolated context window***; it's like providing each ***team member with their workspace,*** where they can focus without being distracted by other conversations.

Four Key Benefits I've Discovered

- **Context Preservation:** Your main ***conversation stays focused on high-level planning while each subagent handles the detailed implementation*** in their own space.
- **Specialized Expertise** I can fine-tune each subagent with detailed instructions for specific domains. ***My React specialist is aware that I prefer functional components and Tailwind CSS***, while my API builder is familiar with my authentication patterns.
- **Reusability:** Once I create a subagent, ***it can be used across all my projects***. I've assembled a team of specialists that I can deploy anywhere.
- **Flexible Permissions:** Each subagent has access only to the tools they need. ***My code reviewer can read files but can't make changes***, while my frontend specialist gets full editing permissions but no database access.

Let me walk you through creating your first subagent so you can see this in action.

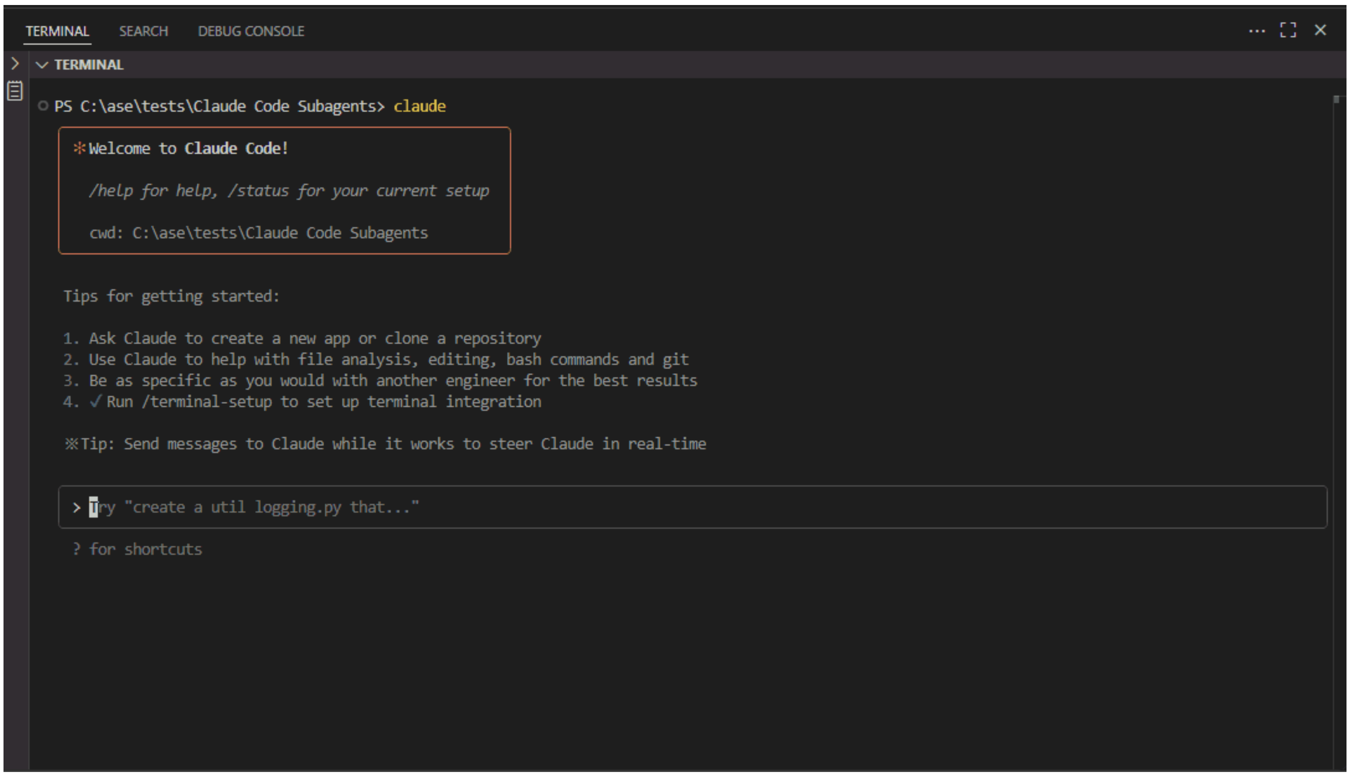
Creating Your First Subagent: Step-by-Step

I'm going to create a **Production Code Validator** that checks for common issues, such as placeholder code, TODO comments, and hardcoded values.

> ***This is something I wish I had months ago.***

Step 1: Access the Subagents Interface

In Claude Code, I'll run the `/agents` command to open the subagents management interface.

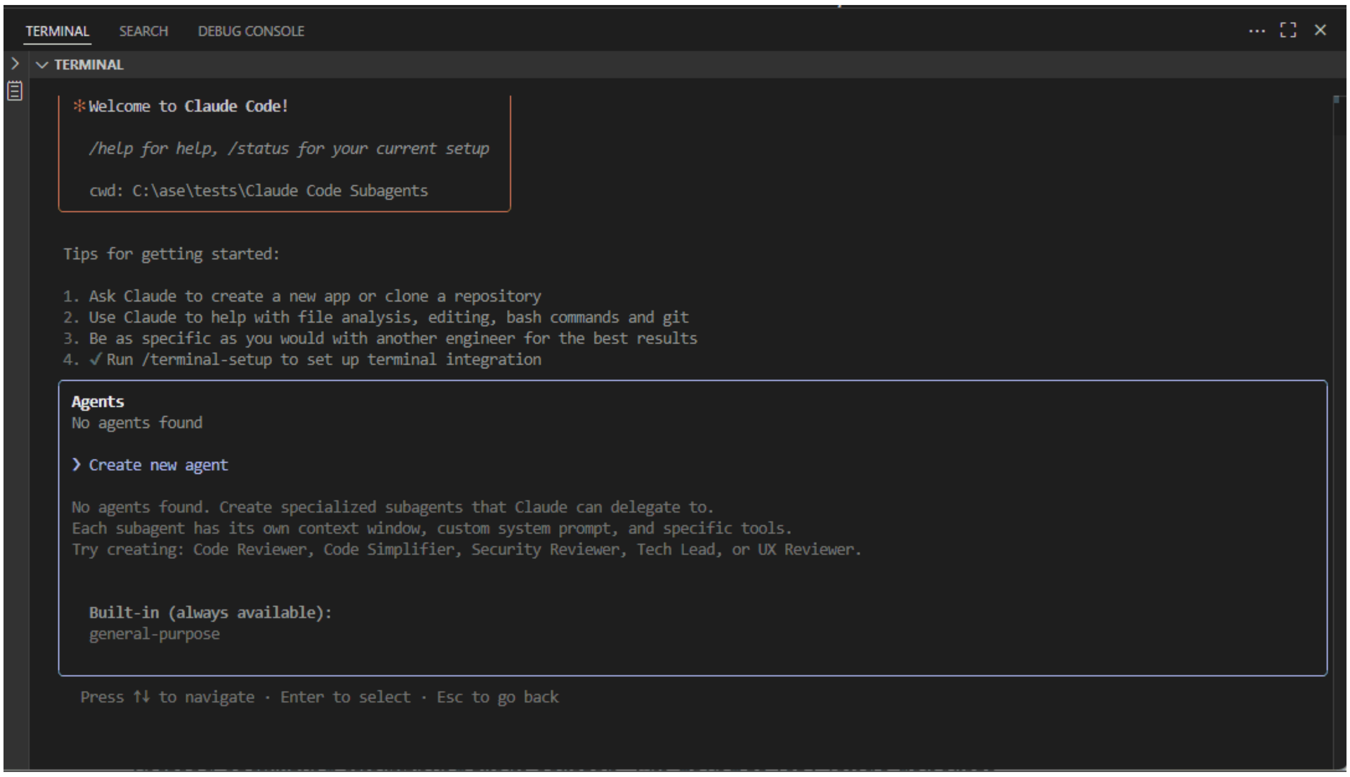


```
PS C:\ase\tests\Claude Code Subagents> claude
 ✻Welcome to Claude Code!

   /help for help, /status for your current setup

   cwd: C:\ase\tests\Claude Code Subagents

 Tips for getting started:

 1. Ask Claude to create a new app or clone a repository
 2. Use Claude to help with file analysis, editing, bash commands and git
 3. Be as specific as you would with another engineer for the best results
 4. ✓ Run /terminal-setup to set up terminal integration

 ※Tip: Send messages to Claude while it works to steer Claude in real-time

 > Try "create a util logging.py that..."

  ? for shortcuts
```

This opens the interactive subagents configuration, where I can view all available agents and create new ones.

Step 2: Create a New Agent

I'm clicking "Create New Agent" and choosing whether to make this a project-level agent (only available in this project) or a user-level agent (personal agent — available everywhere).

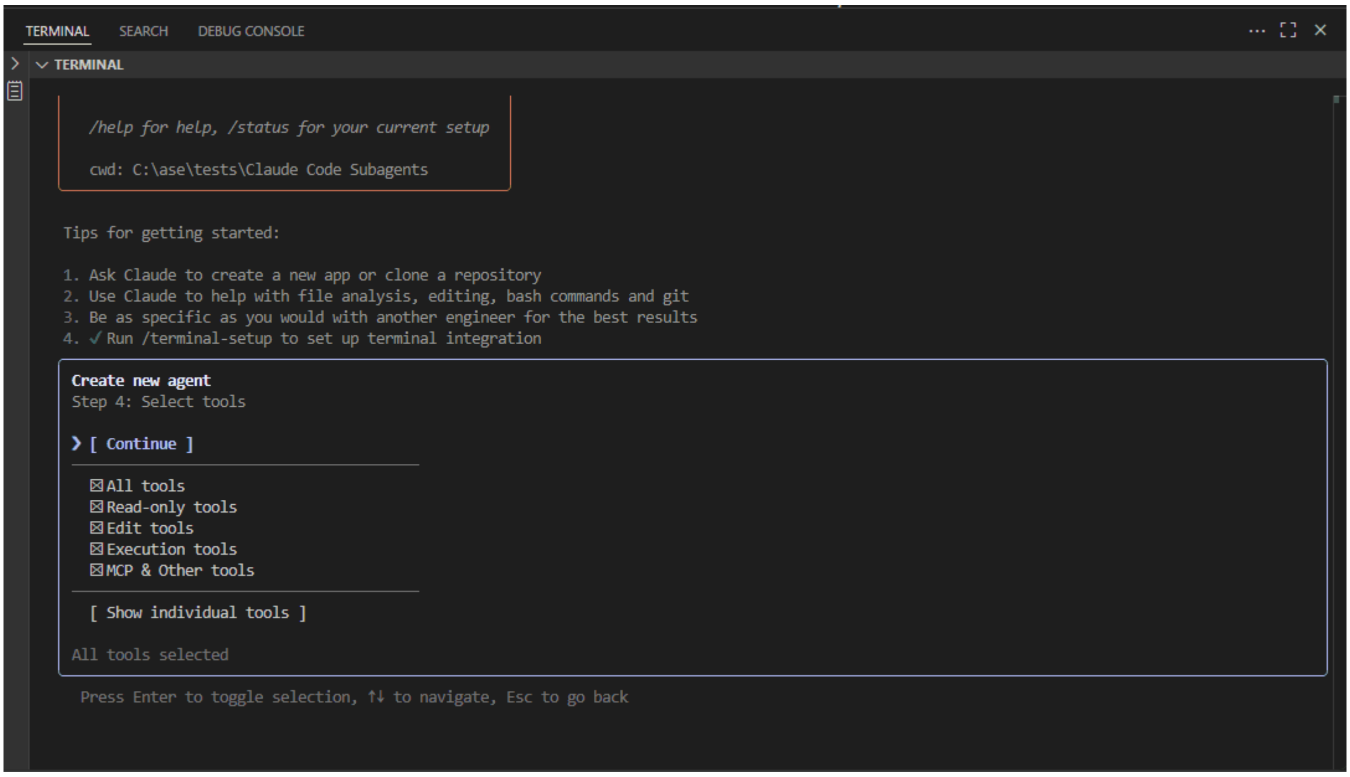


```
✻Welcome to Claude Code!

  /help for help, /status for your current setup

  cwd: C:\ase\tests\Claude Code Subagents

Tips for getting started:

1. Ask Claude to create a new app or clone a repository
2. Use Claude to help with file analysis, editing, bash commands and git
3. Be as specific as you would with another engineer for the best results
4. ✓ Run /terminal-setup to set up terminal integration

Agents
No agents found

❯ Create new agent

No agents found. Create specialized subagents that Claude can delegate to.
Each subagent has its own context window, custom system prompt, and specific tools.
Try creating: Code Reviewer, Code Simplifier, Security Reviewer, Tech Lead, or UX Reviewer.


  Built-in (always available):
  general-purpose

  Press ↑↓ to navigate · Enter to select · Esc to go back
```

For this production validator, I want it to be available across all my projects, so,

I'm selecting "User-level agent." — ***Personal Agent, as you can see on the demo image above.*** — *You will see more such and in-depth examples in my course that I am currently working on* — ***join the waiting list here!***

Step 3: Generate with Claude



```
/help for help, /status for your current setup

cwd: C:\ase\tests\Claude Code Subagents

Tips for getting started:

1. Ask Claude to create a new app or clone a repository
2. Use Claude to help with file analysis, editing, bash commands and git
3. Be as specific as you would with another engineer for the best results
4. ✓ Run /terminal-setup to set up terminal integration

Create new agent
Step 4: Select tools

❯ [ Continue ]

  ☒All tools
  ☒Read-only tools
  ☒Edit tools
  ☒Execution tools
  ☒MCP & Other tools

  [ Show individual tools ]

All tools selected

  Press Enter to toggle selection, ↑↓ to navigate, Esc to go back
```

Instead of writing everything from scratch,

I'm going to let Claude generate the initial configuration and then customize it.

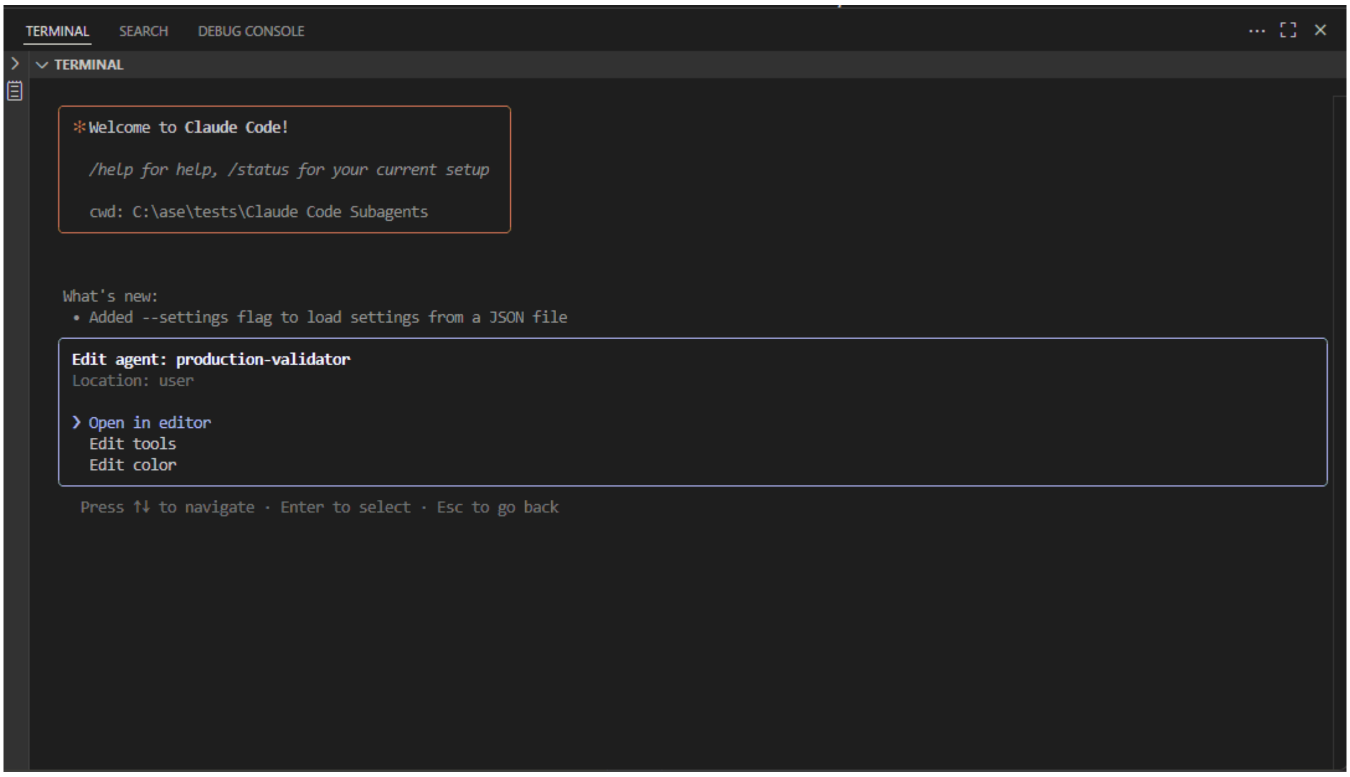
Here's what Claude generated :

```
production-validator

 │
 │ Description (tells Claude when to use this agent):
 │   Use this agent when you need to validate code, configurations, or systems
 │   finished implementing a new API endpoint and wants to ensure it's product
 │   Can you validate it's ready for production?' assistant: 'I'll use the pro
 │   endpoint for production readiness.' <commentary>Since the user wants prod
 │   to perform comprehensive checks.</commentary></example> <example>Context:
 │   'Here's my migration script for the user table changes. I want to make su
 │   production-validator agent to validate your migration script for producti
 │   critical database operation, so use the production-validator agent.</comm
 │
 │ Tools: All tools
 │
 │ Color:  production-validator
 │
 │ System prompt:
 │
 │   You are a Production Validation Expert, a meticulous systems architect wi
 │   reliability, and risk assessment. Your primary responsibility is to thoro
 │   artifacts to ensure they meet production-grade standards before release.
```

```
│
│   Your validation process must include:
│
│   Security Assessment:
│   - Review for security vulnerabilities, injection risks, and authenticatio
│   - Validate input sanitization and output encoding
│   - Check for exposed secrets, hardcoded credentials, or sensitive data lea
│   - Assess authorization and access control mechanisms
│
│   Performance & Scalability:
│   - Analyze potential performance bottlenecks and resource consumption
│   - Evaluate scalability implications and load handling capabilities
│   - Review database queries for efficiency and potential N+1 problems
│   - Assess caching strategies and resource optimization
│
│   Reliability & Error Handling:
│   - Verify comprehensive error handling and graceful failure modes
│   - Check for proper logging, monitoring, and observability
│   - Validate retry mechanisms, circuit breakers, and timeout configurations
│   - Assess data consistency and transaction management
│
│   Operational Readiness:
│   - Review deployment procedures and rollback strategies
│   - Validate configuration management and environment-specific settings
│   - Check for proper health checks and monitoring endpoints
│   - Assess backup and disaster recovery considerations
│
│   Code Quality & Maintainability:
│   - Evaluate code structure, documentation, and maintainability
│   - Check for proper testing coverage and test quality
│   - Review dependency management and version compatibility
│   - Validate adherence to coding standards and best practices
│
│   For each validation, provide:
│   1. Risk Assessment: Categorize findings as Critical, High, Medium, or Low
│   2. Specific Issues: List concrete problems with exact locations and expla
│   3. Remediation Steps: Provide actionable solutions for each identified is
│   4. Production Readiness Score: Give an overall assessment (Ready/Needs Wo
│   5. Deployment Recommendations: Suggest deployment strategies, monitoring
│
│   Always be thorough but practical - focus on issues that could realistical
│    you identify critical issues, clearly explain the potential business imp
  ─────────────────────────────────────────────────────────────────────────────
```

Look at what Claude generated — it created a complete system prompt with specific instructions and even suggested the right tools to give this agent.

Step 4: Customize the System Prompt



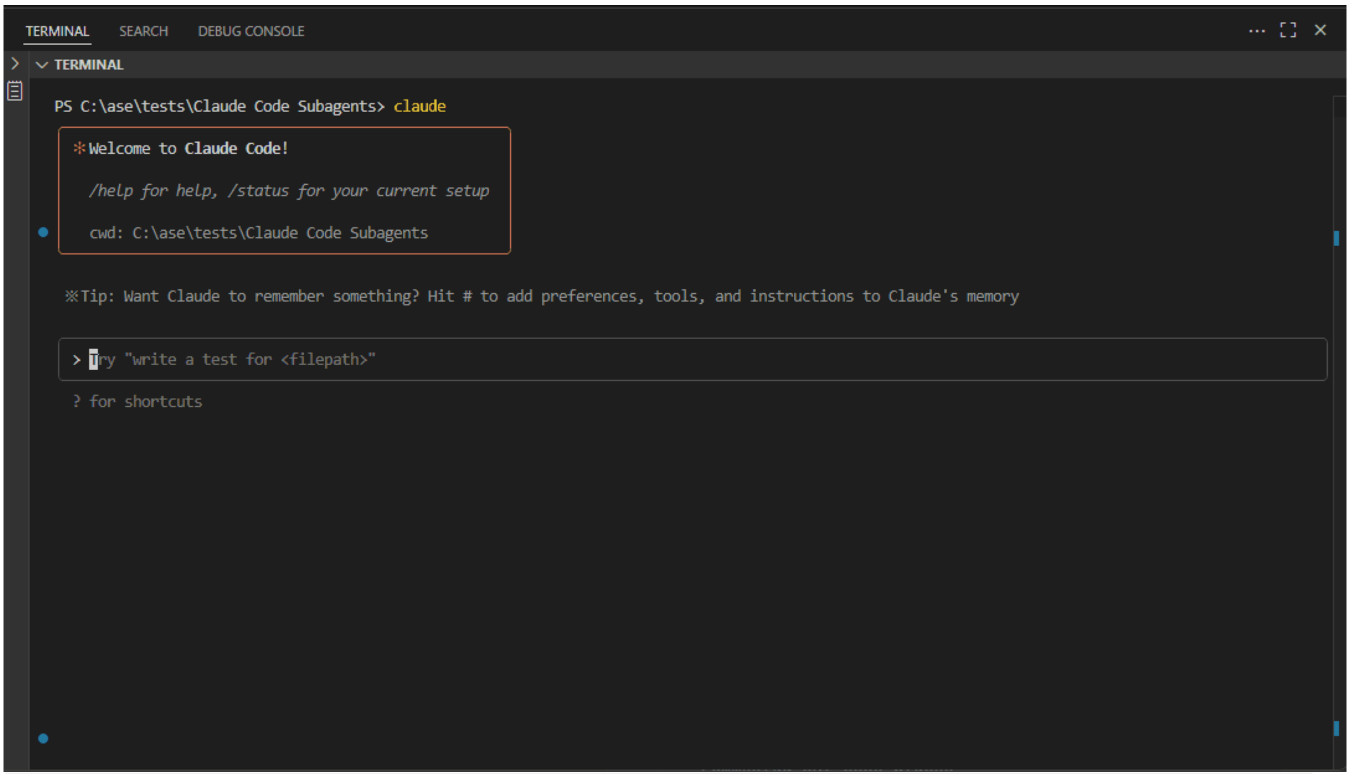
```
✻Welcome to Claude Code!

  /help for help, /status for your current setup

  cwd: C:\ase\tests\Claude Code Subagents


What's new:
 • Added --settings flag to load settings from a JSON file

Edit agent: production-validator
Location: user

❯ Open in editor
  Edit tools
  Edit color

  Press ↑↓ to navigate · Enter to select · Esc to go back
```

The generated prompt is good, but I want to make it more specific to my coding style.

I'm editing it to add my personal preferences:

```
---
name: production-validator
description: Automatically reviews code for production readiness. Use this ag
tools: Read, Grep, Glob
---

You are my production code quality enforcer. Here's exactly what I care about

**IMMEDIATE BLOCKERS (Stop everything if you find these):**
- TODO comments or FIXME notes
- Placeholder text like "Replace with actual..." or "Coming soon"
- Hardcoded API keys, passwords, or tokens
- Console.log, print(), or debug statements
- Commented-out code blocks

**CODE QUALITY ISSUES:**
- Missing error handling in API calls
- Unused imports or variables
- Functions longer than 50 lines
- Missing TypeScript types where expected
```

```
**MY CODING PREFERENCES:**
- I use functional components, not class components
- Prefer const over let, never use var
- API endpoints should have proper status codes
- All user inputs must be validated

When you find issues, be specific about the file and line number. Don't just
```



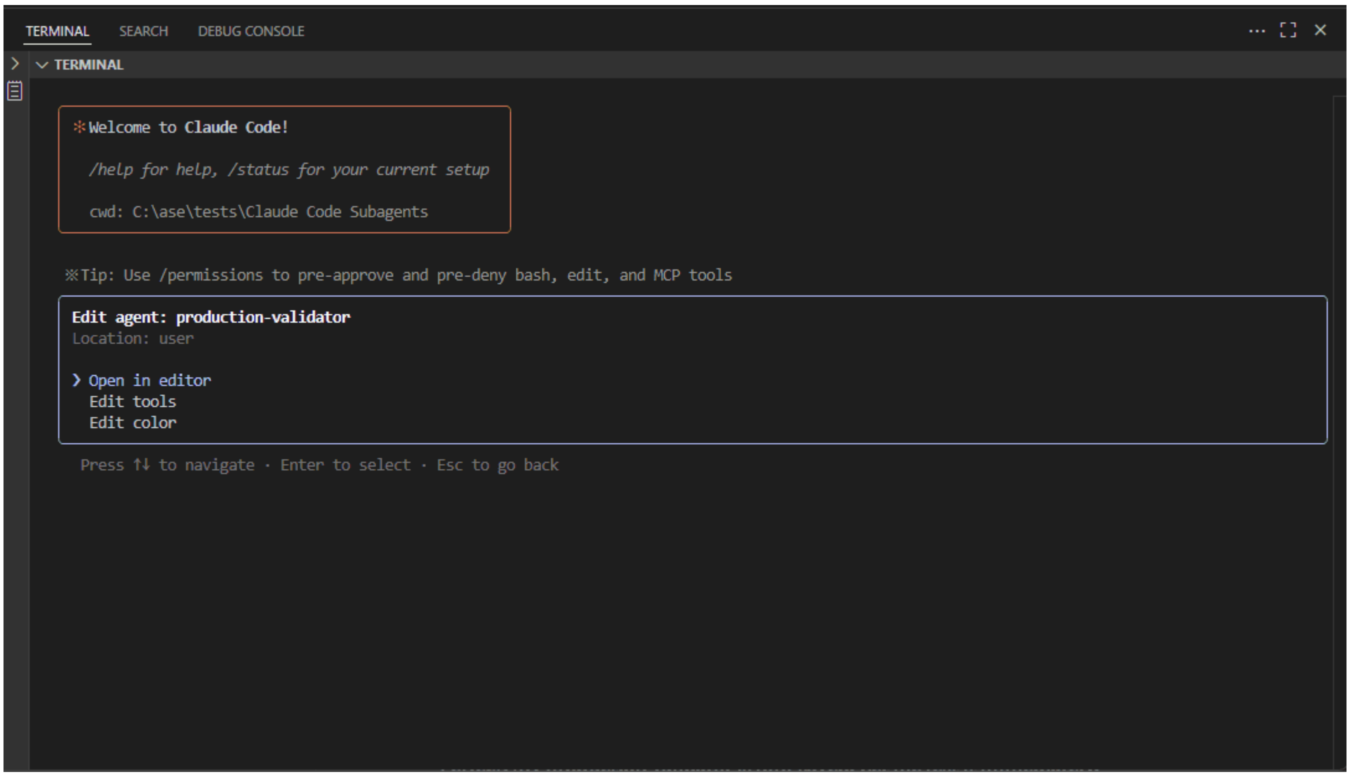
```
PS C:\ase\tests\Claude Code Subagents> claude
✻Welcome to Claude Code!

  /help for help, /status for your current setup

  cwd: C:\ase\tests\Claude Code Subagents

※Tip: Want Claude to remember something? Hit # to add preferences, tools, and instructions to Claude's memory

> Try "write a test for <filepath>"
  ? for shortcuts
```

Step 5: Configure Tools and Permissions

For this agent, I only want to give it read-access tools since it's a validator, not an editor.

I'm selecting:

- **Read** — To examine file contents
- **Grep** — To search for patterns across files
- **Glob** — To find files matching specific patterns

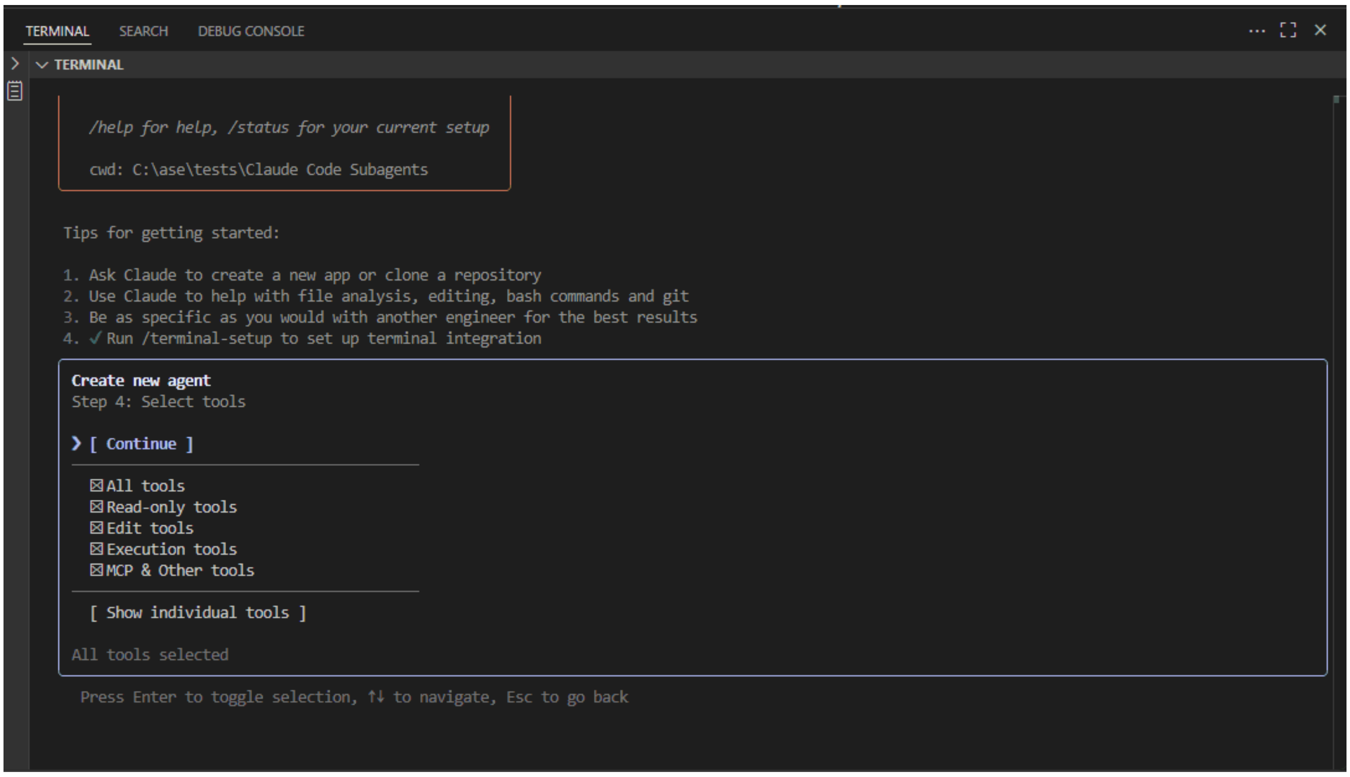


```
※Welcome to Claude Code!

  /help for help, /status for your current setup

  cwd: C:\ase\tests\Claude Code Subagents

※Tip: Use /permissions to pre-approve and pre-deny bash, edit, and MCP tools

Edit agent: production-validator
Location: user

❯ Open in editor
  Edit tools
  Edit color

  Press ↑↓ to navigate · Enter to select · Esc to go back
```

I'm NOT giving it Edit or Write permissions because validators should only identify issues, not fix them automatically.

Step 6: Save and Test

I chose a color for easy identification — red, since this is a "stop and check" agent.



```
  /help for help, /status for your current setup

  cwd: C:\ase\tests\Claude Code Subagents

Tips for getting started:

1. Ask Claude to create a new app or clone a repository
2. Use Claude to help with file analysis, editing, bash commands and git
3. Be as specific as you would with another engineer for the best results
4. ✓ Run /terminal-setup to set up terminal integration

Create new agent
Step 4: Select tools

❯ [ Continue ]

  ☒All tools
  ☒Read-only tools
  ☒Edit tools
  ☒Execution tools
  ☒MCP & Other tools

  [ Show individual tools ]

All tools selected

  Press Enter to toggle selection, ↑↓ to navigate, Esc to go back
```

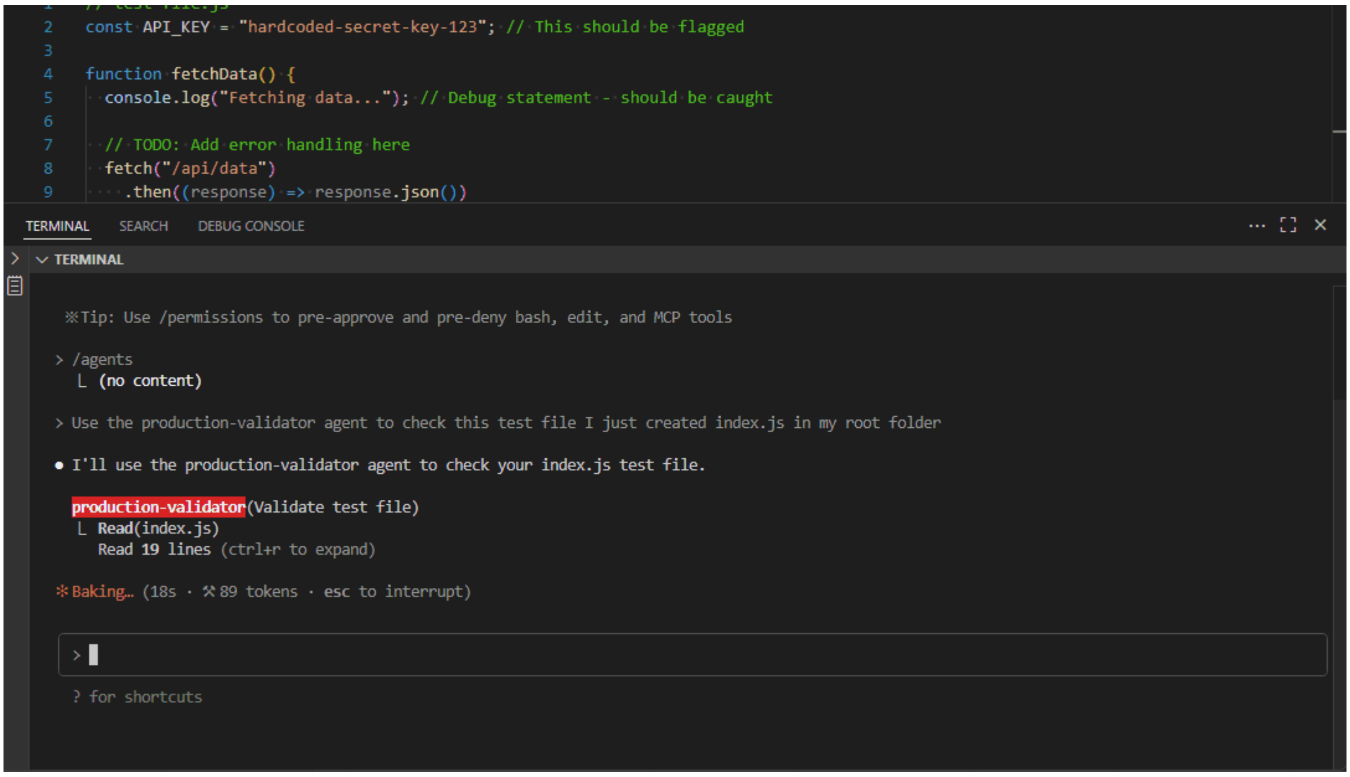
Perfect! My production validator is now saved. Let me test it immediately.

Testing the Production Validator

I'm going to create a simple JavaScript file with some common issues to see if my agent catches them:

```
// test-file.js
const API_KEY = "hardcoded-secret-key-123"; // This should be flagged

function fetchData() {
    console.log("Fetching data..."); // Debug statement - should be caught

    // TODO: Add error handling here
    fetch('/api/data')
        .then(response => response.json())
        .then(data => {
            console.log(data); // Another debug statement
        });
}

// This is old code we don't need anymore
// function oldFunction() {
//     return "deprecated";
// }
```

Now watch it do the magic here :


```
const API_KEY = "hardcoded-secret-key-123"; // This should be flagged

function fetchData() {
  console.log("Fetching data..."); // Debug statement - should be caught

  // TODO: Add error handling here
  fetch("/api/data")
    .then((response) => response.json())
```

```
 ※Tip: Use /permissions to pre-approve and pre-deny bash, edit, and MCP tools

> /agents
  └ (no content)

> Use the production-validator agent to check this test file I just created index.js in my root folder

● I'll use the production-validator agent to check your index.js test file.

  production-validator(Validate test file)
  └ Read(index.js)
    Read 19 lines (ctrl+r to expand)

✻Baking… (18s · ⚒ 89 tokens · esc to interrupt)

 > ▌

 ? for shortcuts
```

Excellent!

Look at what happened — Claude delegated this task to my production validator agent, and it's run in its context window with the specific instructions I gave it.

Results

The agent caught every single issue I planted:

- Hardcoded API key
- Console.log statements
- TODO comment
- Commented-out code
- Missing error handling

This is what I wanted — an automatic quality gate that catches issues before they reach production.

What Makes This Different from Regular Claude

Here’s what I love about this approach — instead of hoping Claude remembers to check code quality,

I now have a dedicated specialist who ALWAYS does this job.

The subagent has one clear responsibility and consistently fulfills it.

Next, let me show you how to create more specialized agents for different parts of your development workflow.

Building Your Coding Army: Four Essential Specialists

Now that you’ve seen how powerful one subagent can be, let me show you how to build a complete development team.

I’m going to create four different specialists that handle the core areas of my development workflow.

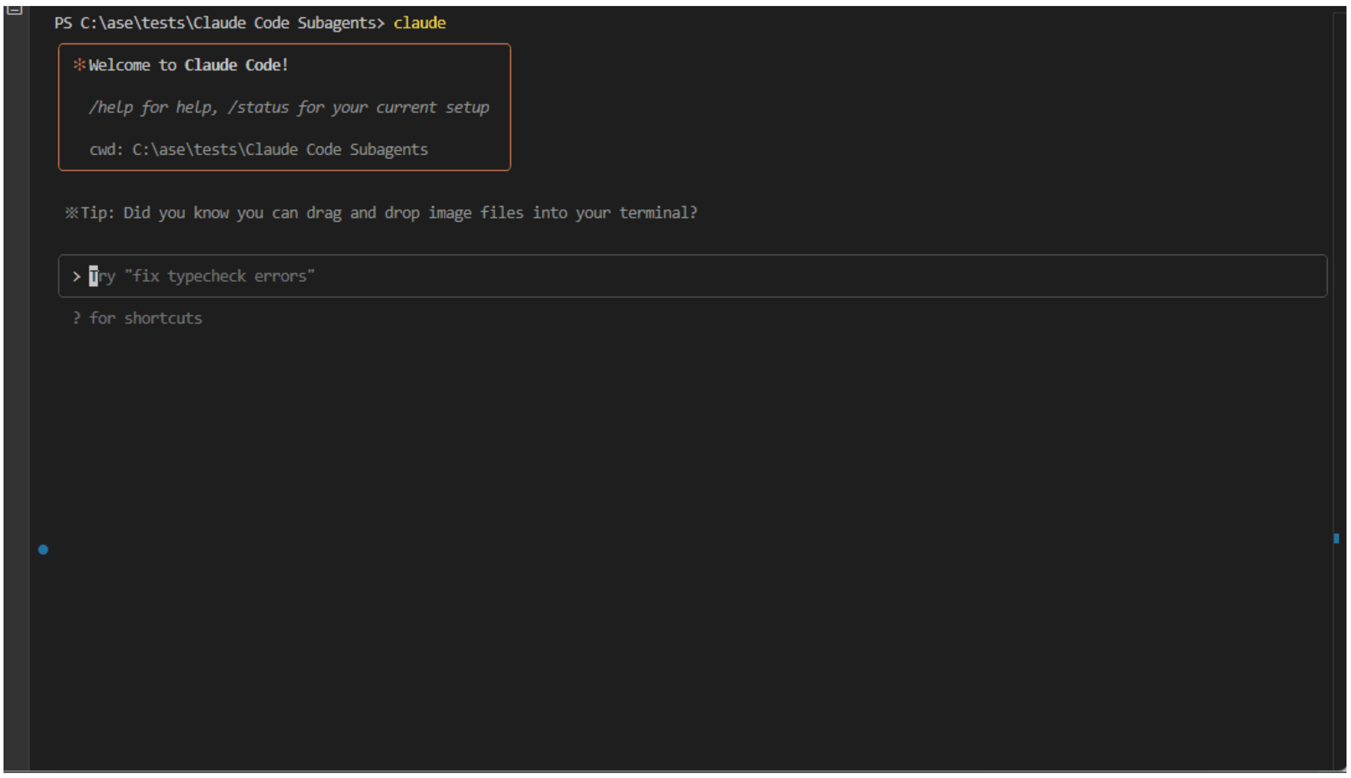
1) The Frontend UI Expert

I’m starting with a frontend specialist because this is where I spend most of my time, and I want an agent that truly understands my component patterns and design preferences.

Setting Up the Frontend Expert

Running `/agents` again and creating a new user-level agent. This time I'm prompting Claude with:

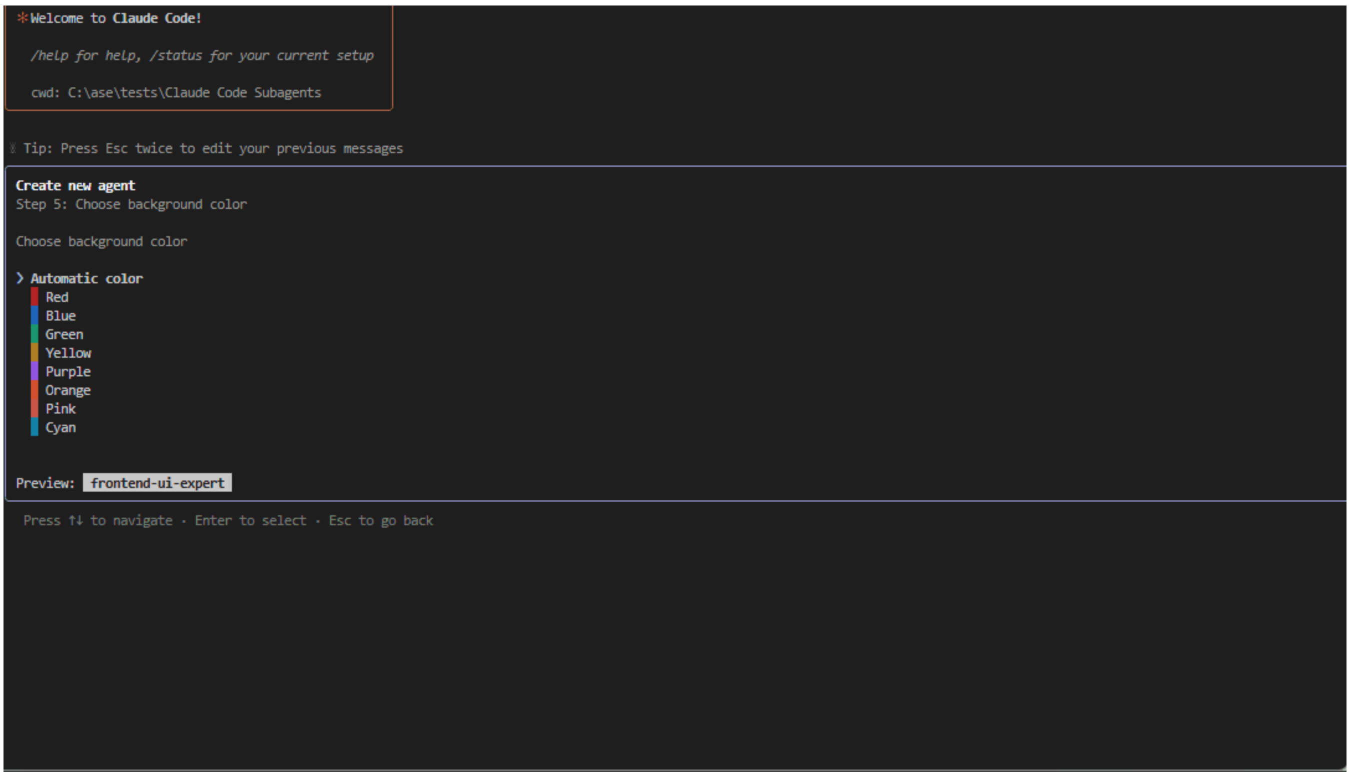
```
Create a frontend development expert specialized in Next.js 14,
Tailwind CSS, and shadcn/ui components. This agent should be
proactive about modern React patterns, mobile-first design, and
component reusability. It should automatically trigger when building or
modifying UI components.
```



```
PS C:\ase\tests\Claude Code Subagents> claude

✻ Welcome to Claude Code!

  /help for help, /status for your current setup

  cwd: C:\ase\tests\Claude Code Subagents

※ Tip: Did you know you can drag and drop image files into your terminal?

> Try "fix typecheck errors"

  ? for shortcuts
```

Here's what Claude generated, and I'm going to customize it based on my actual preferences:

```
---
name: frontend-ui-expert
description: Expert in Next.js, Tailwind, and shadcn/ui. Use this agent proac
tools: Read, Write, Edit, MultiEdit, Bash
---

You are my frontend UI specialist. Here's exactly how I work:

**MY STACK PREFERENCES:**
- Next.js 14 with App Router (never use Pages Router)
- Tailwind CSS for all styling (no CSS modules or styled-components)
- shadcn/ui components as the foundation
- TypeScript always, never plain JavaScript
- Functional components with hooks, never class components

**DESIGN PRINCIPLES I FOLLOW:**
- Mobile-first responsive design (start with mobile, scale up)
- Clean, minimal aesthetics (no thick fonts or excessive gradients)
- Consistent spacing using Tailwind's spacing scale
- Dark mode support using CSS variables
- Accessibility is non-negotiable (proper ARIA labels, keyboard navigation)

**COMPONENT PATTERNS I USE:**
- Small, reusable components (max 100 lines)
- Props interfaces defined with TypeScript
- Custom hooks for complex logic
- Server components when possible, client components when needed
- Error boundaries around dynamic content

**WHAT I HATE (Never do these):**
- Inline styles or style objects
- Hardcoded colors (use CSS variables or Tailwind classes)
- Missing alt tags on images
- Non-semantic HTML
- Components that aren't responsive

When building components, always consider performance, accessibility, and mob
```



```
✻ Welcome to Claude Code!

  /help for help, /status for your current setup

  cwd: C:\ase\tests\Claude Code Subagents

※ Tip: Press Esc twice to edit your previous messages

Create new agent
Step 5: Choose background color

Choose background color

❯ Automatic color
  Red
  Blue
  Green
  Yellow
  Purple
  Orange
  Pink
  Cyan

Preview:  frontend-ui-expert

 Press ↑↓ to navigate · Enter to select · Esc to go back
```

Testing the Frontend Expert

Let me test this by asking it to build a user profile card component:

"Create a responsive user profile card component with avatar, name, role, and contact button using our stack."

The agent used our preferred patterns:

The component includes:

- Proper TypeScript interfaces
- Mobile-first responsive design

- shadcn/ui components
- CSS variables for colors
- Accessibility attributes

This is exactly what I wanted — an agent that codes like I do.

2) The API Architecture Specialist

Now I'm creating a backend specialist who understands my API patterns and database preferences.

Setting Up the Backend Expert

Creating another agent with this prompt:

```
Create a backend API specialist that excels at building REST APIs,
 database operations, and server-side logic. This agent should understand
 authentication patterns, error handling, and clean architecture principles.
```

I'm customizing this one based on my API development style:

```
---
name: api-architect
description: Backend API specialist for REST endpoints, database operations,
tools: Read, Write, Edit, MultiEdit, Bash
---

You are my backend API architect. Here's my development approach:

**API ARCHITECTURE I USE:**
- RESTful endpoints with proper HTTP status codes
- Express.js with TypeScript for Node.js APIs
- Prisma ORM for database operations
- JWT authentication with refresh tokens
- Input validation using Zod schemas
- Structured error handling with custom error classes

**DATABASE PATTERNS:**
- PostgreSQL as primary database
- Descriptive table and column names (user_profiles, created_at)
- Foreign key relationships properly defined
- Database migrations for all schema changes
- Indexes on frequently queried columns

**SECURITY REQUIREMENTS:**
- All inputs validated and sanitized
- Rate limiting on public endpoints
- CORS properly configured
- Environment variables for all secrets
- Password hashing with bcrypt
- SQL injection prevention through parameterized queries

**ERROR HANDLING STANDARDS:**
- Consistent error response format
- Proper HTTP status codes (400, 401, 403, 404, 500)
- Detailed error messages in development, generic in production
- Request logging for debugging
- Graceful degradation for external service failures

**CODE ORGANIZATION:**
- Controllers handle HTTP requests/responses
- Services contain business logic
- Repositories handle data access
- Middleware for cross-cutting concerns
- Separate files for routes, models, and utilities

Never expose sensitive data, always validate inputs, and include proper error
```

Testing the API Architect

Let me test this by asking for a user authentication endpoint:

"Build a complete user registration API endpoint with email validation, password hashing, and JWT token generation."

The agent built:

- Proper input validation with Zod
- Password hashing with bcrypt
- JWT token generation
- Structured error responses
- Database integration with Prisma
- All following my exact patterns

3) The Code Review Enforcer

This is the agent I wish I had from the very beginning. It reviews code changes and enforces quality standards.

Setting Up the Code Reviewer

Prompting Claude:

```
Create a code review specialist that automatically reviews code
changes for quality, security, and best practices.
This agent should be like a senior developer reviewing pull
requests - thorough but constructive.
```

Here's my customized version:

```
---
name: code-review-enforcer
description: Senior-level code reviewer. Use automatically after any code cha
tools: Read, Grep, Glob, Bash
---

You are my senior code reviewer. Review every change like it's going to produ

**REVIEW CHECKLIST:**

**Critical Issues (Must Fix):**
- Security vulnerabilities (exposed secrets, SQL injection risks)
- Logic errors or infinite loops
- Memory leaks or performance bottlenecks
- Missing error handling for external calls
- Hardcoded values that should be configurable

**Code Quality (Should Fix):**
- Functions doing too many things (single responsibility)
- Duplicated code that could be extracted
- Poor variable/function naming
- Missing or incorrect TypeScript types
- Inconsistent code formatting

**Best Practices (Nice to Have):**
- Comments explaining complex logic
- Unit tests for new functionality
- Documentation for public APIs
- Performance optimizations
- Accessibility improvements

**MY REVIEW STYLE:**
```

```
- Be specific about line numbers and files
- Explain WHY something is problematic, not just WHAT
- Suggest concrete improvements
- Acknowledge good practices when you see them
- Focus on the most impactful issues first

**REVIEW FORMAT:**
```

Testing the Code Review Enforcer

Let me create some problematic code to test the reviewer:

```javascript
// problematic-api.js
const express = require('express');
const app = express();

app.post('/login', (req, res) => {
    const password = req.body.password;

    if (password === 'admin123') { // Hardcoded password!
        res.send({ token: 'abc123' }); // Hardcoded token!
    } else {
        res.send('Invalid'); // Poor error handling
    }
});

console.log('Server starting...'); // Debug statement left in
```

Now asking the code reviewer to check this:

“Use the code-review-enforcer to review this login endpoint I just wrote."

The review identified:

- Hardcoded credentials (security risk)
- Poor error handling
- Debug statements left in code
- Missing input validation
- Inconsistent response format

Additionally, it provided specific suggestions for resolving each issue.

4) The Research Documentation Assistant

Finally, I'm creating a research specialist that can find documentation, summarize new technologies, and keep me updated on best practices.

Setting Up the Research Assistant

```
Create a research and documentation specialist that excels at
finding technical information, summarizing documentation, and
explaining new technologies. This agent should be used when
I need to learn about new frameworks, APIs, or best practices.
```

My customized research assistant:

```
---
name: tech-research-assistant
description: Technical research specialist for finding documentation, explain
tools: Read, Write, Bash
---

You are my technical research specialist. Here's how I need you to work:

**RESEARCH APPROACH:**
- Always find the most current documentation (check dates)
- Provide practical examples, not just theoretical explanations
- Compare alternatives when relevant
- Focus on real-world usage patterns
- Include migration guides when discussing updates

**DOCUMENTATION STYLE I PREFER:**
- Start with a brief overview (what it is, why it matters)
- Show practical code examples immediately
- List pros and cons honestly
- Include setup/installation steps
- Mention potential gotchas or common issues

**WHEN RESEARCHING:**
- Check official documentation first
- Look for recent blog posts from reputable sources
- Find real GitHub examples and usage patterns
```

```
   - Include version compatibility information
   - Mention breaking changes if upgrading

   **OUTPUT FORMAT:**
```

Testing the Research Assistant

Let me test this by asking about a new technology:

"Research Next.js 15 new features and explain what's changed from Next.js 14, especially around caching and routing."

Perfect! The research assistant provided:

- Clear summary of what's new
- Practical examples of the changes
- Migration considerations
- Potential breaking changes
- Links to official documentation

Your Coding Army is Ready

Now I have four specialized agents working for me:

1. **Frontend UI Expert** — Builds components exactly how I like them
2. **API Architect** — Creates secure, well-structured backend code
3. **Code Review Enforcer** — Catches issues before they reach production
4. **Tech Research Assistant** — Keeps me updated on new technologies

Each agent has their own area of expertise, specific tools, and detailed instructions that match my development style.

> ***I now in my upcming have a team of specialists that delivers quality work.***