Claude Code: From Zero to Hero

What is Claude Code?

Claude Code is an AI-powered coding assistant that lives in your terminal, understanding your codebase and accelerating development through natural language. It integrates directly into your workflow, providing a flexible and safe way to leverage AI for coding tasks.

Why Use Claude Code?

Claude Code offers several key benefits:

- **Codebase Understanding:** Quickly understand project architecture and logic.
- **Code Editing & Bug Fixing:** Edit files and fix bugs with natural language.
- **Testing & Linting:** Execute and fix tests and linting errors.
- **Git Integration:** Simplify Git operations like commits, PRs, and conflict resolution.
- **Web Search:** Access documentation and online resources.

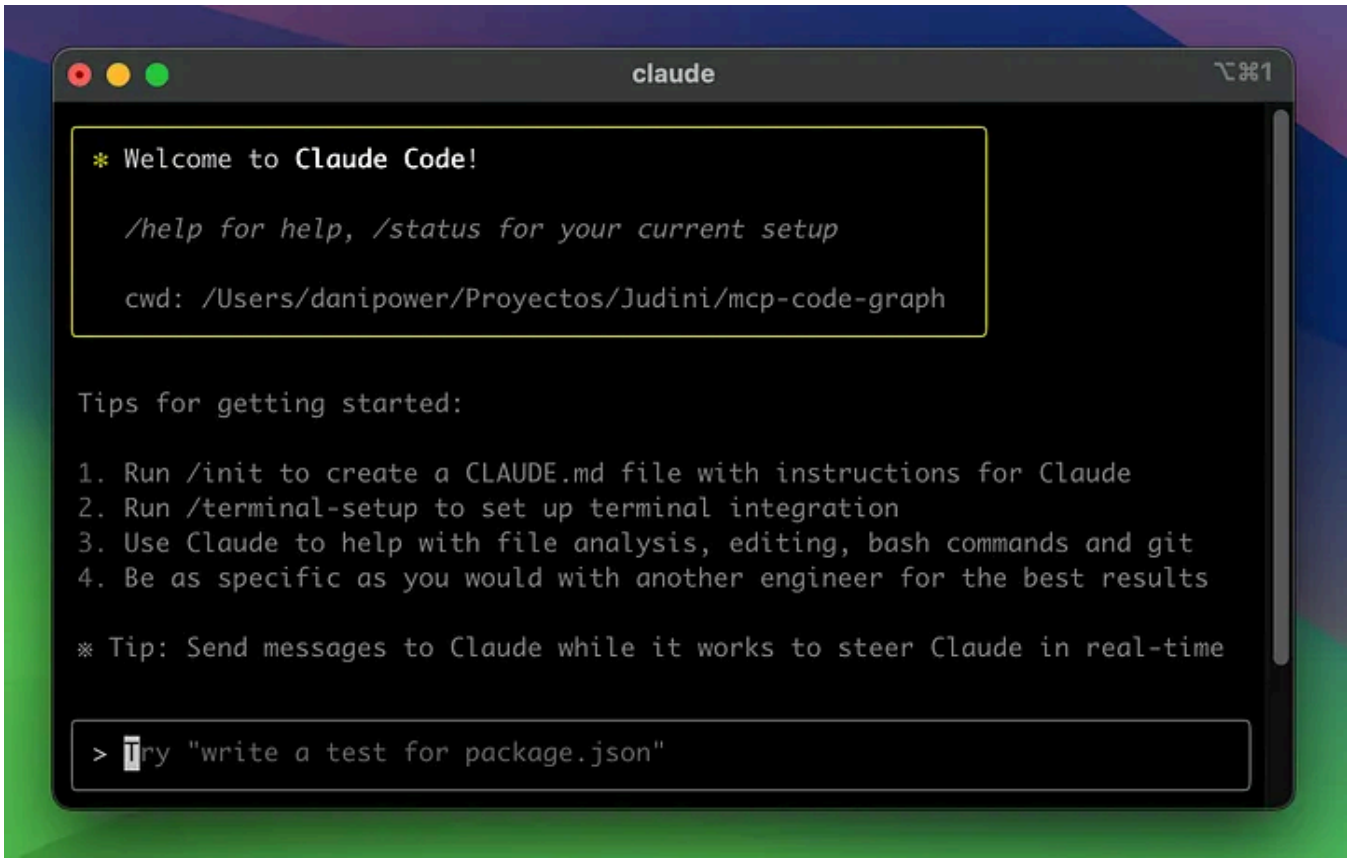
- **MCP Integration**
- **Secure & Private:** Direct API connection to Anthropic, operating within your terminal.

Installation and Authentication

Install Claude Code: Ensure you have Node.js 18+ installed, then run:

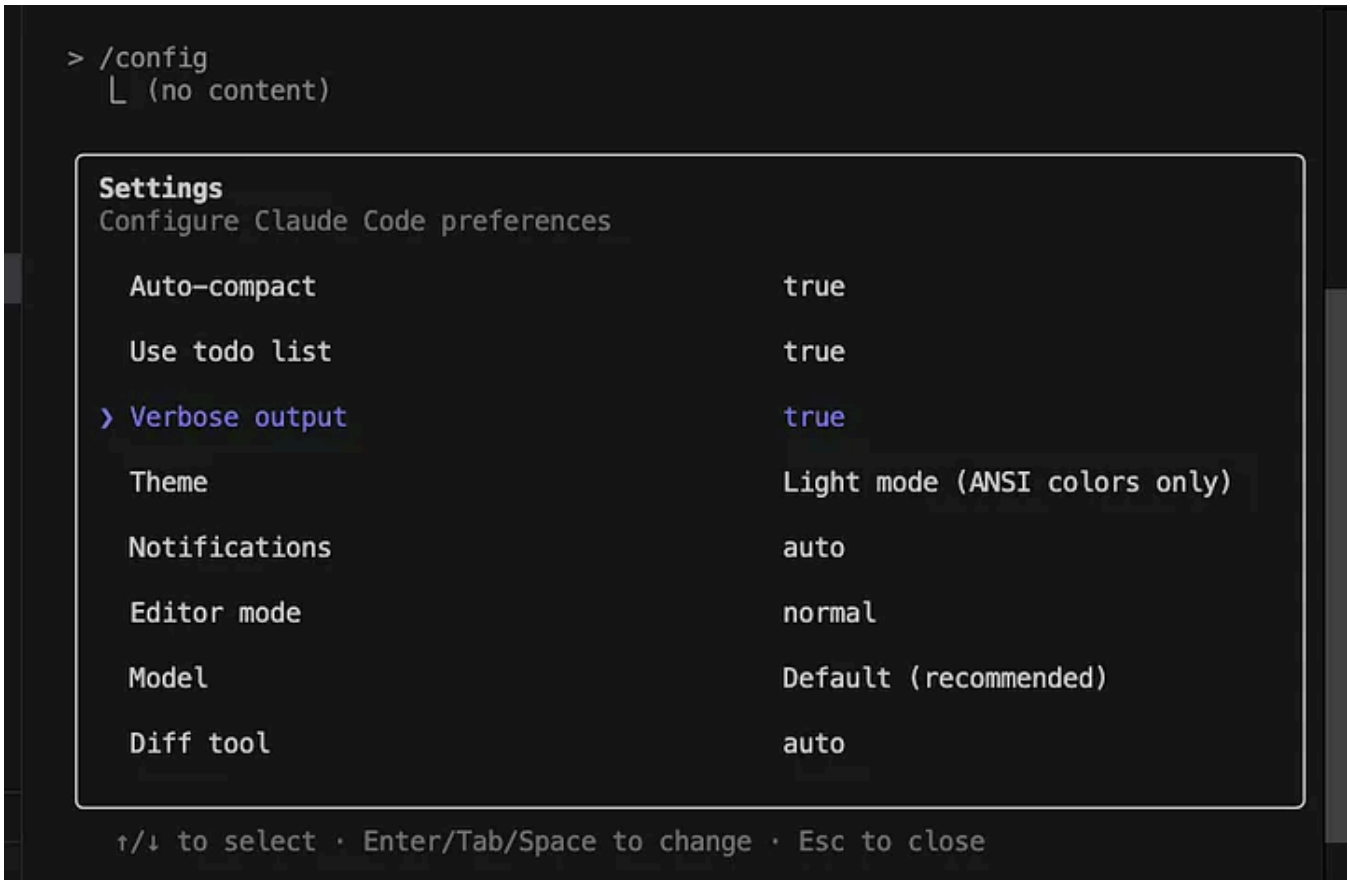
```
npm install -g @anthropic-ai/claude-code
```

Authenticate: Run ***claude*** in your terminal and follow the authentication prompts.



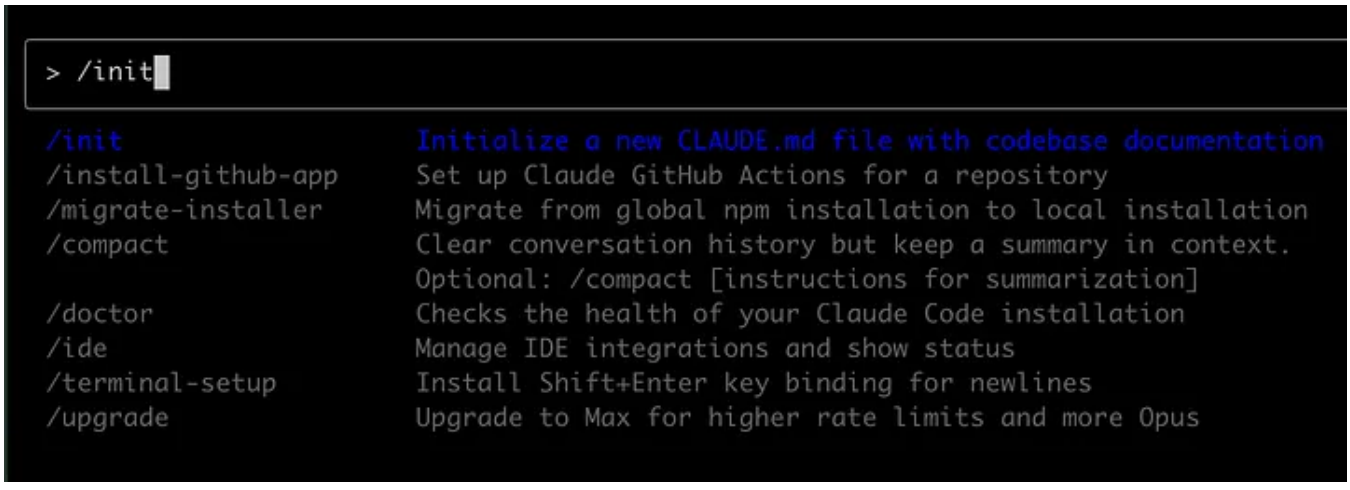
> *Pro tips: You can authenticate via the Anthropic Console, Claude App (Pro/Max plan), or enterprise platforms like Amazon Bedrock or Google Vertex AI.*

If you want to make any changes to Claude Code's initial configuration, just run the /config command

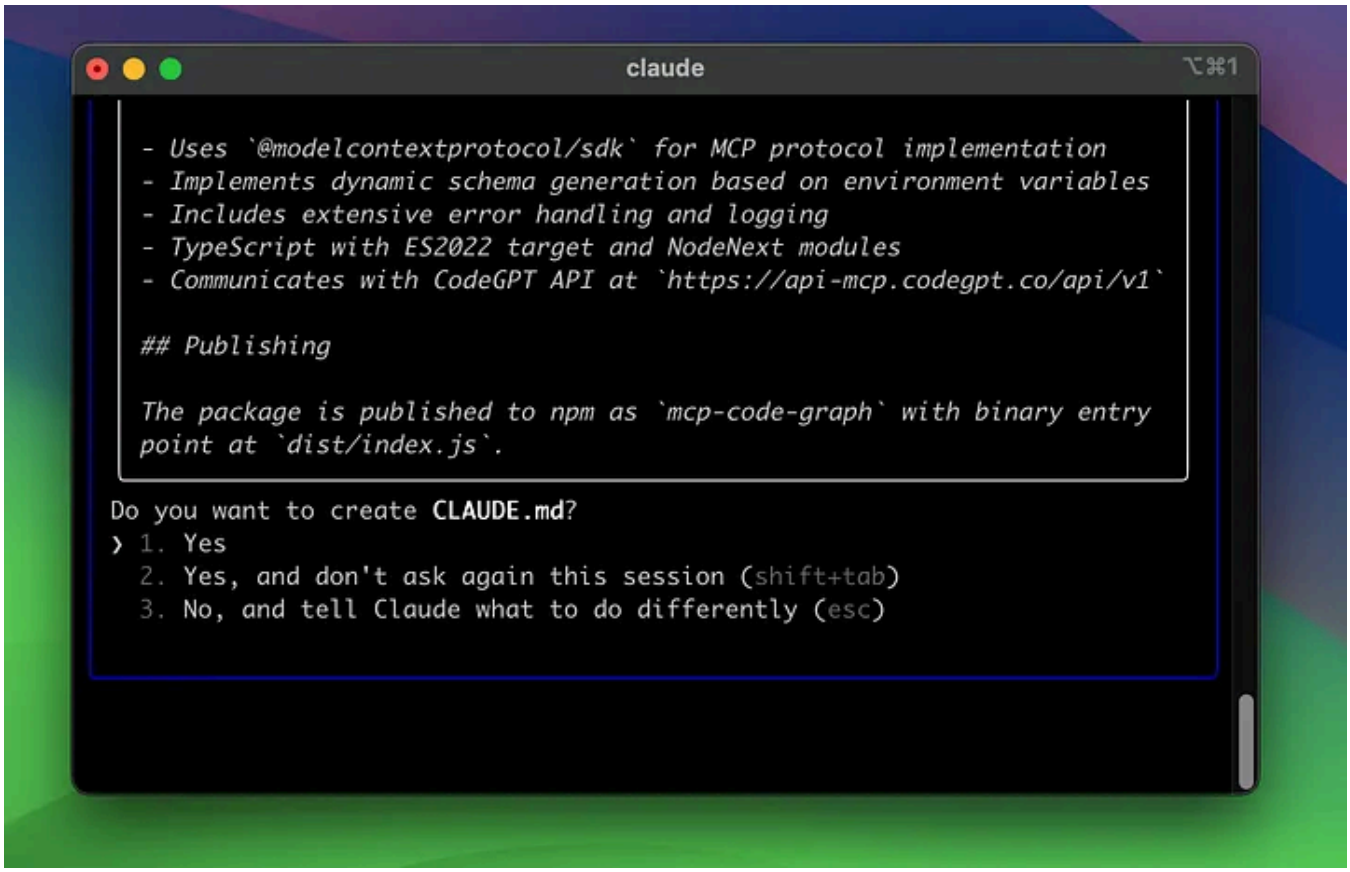


1. Initializing Your Project

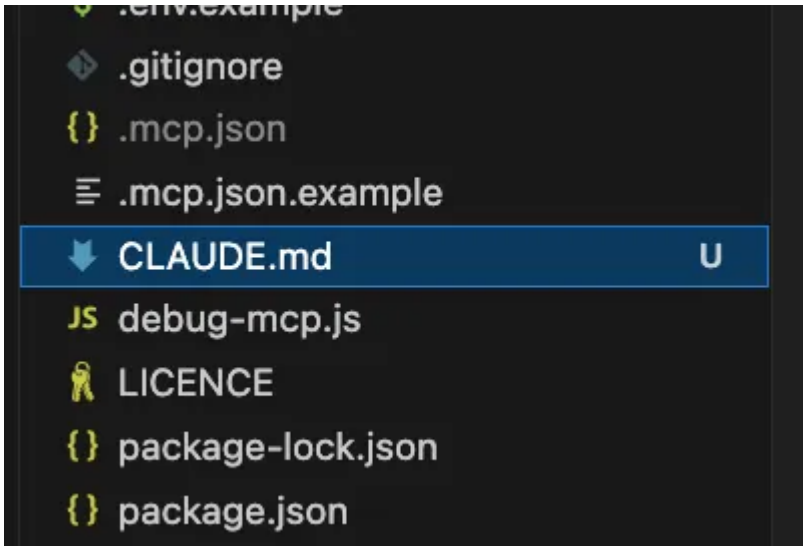
Initialize: Use the /init command to generate a CLAUDE.md project guide.



Running this command, Claude Code will create the CLAUDE.md file with all the necessary information to work properly with this project.

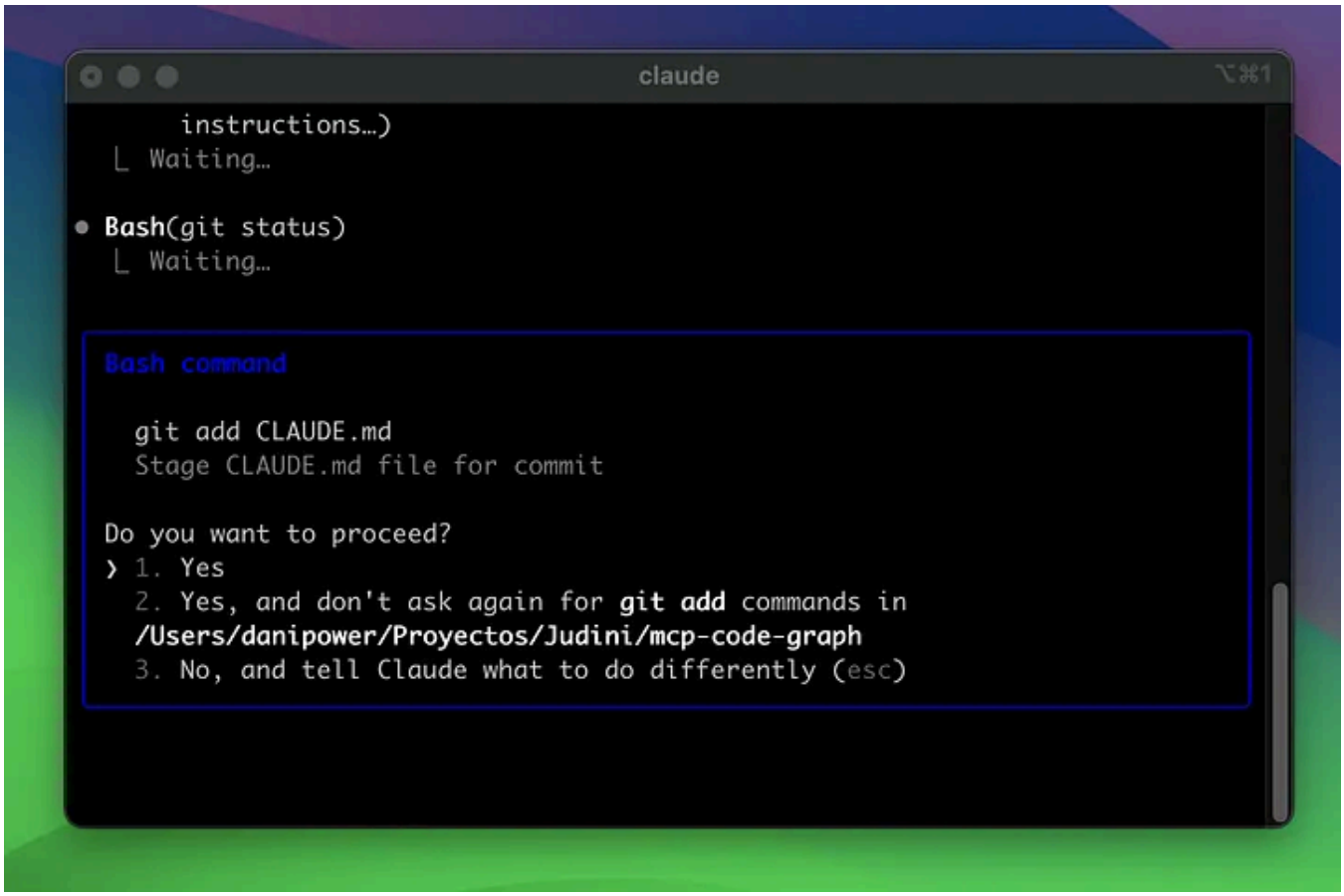


Click Yes and you'll see the file created in your project.



You can ask Claude Code to commit the CLAUDE.md file it just generated: "commit the CLAUDE.md file".

Claude Code will execute git and add this file to staging.

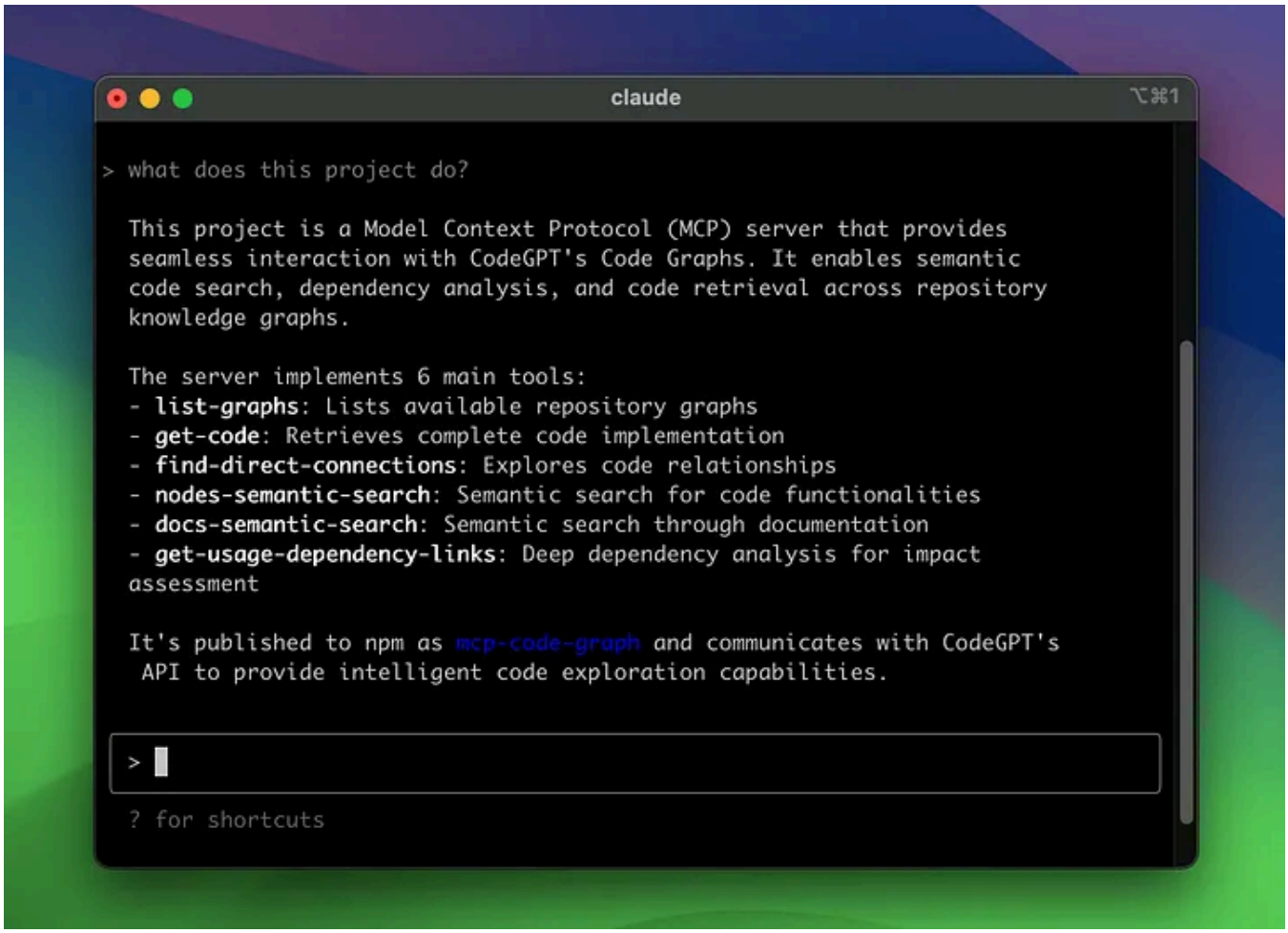


2. Basic Usage

Asking Questions

Start by understanding your codebase:

> what does this project do?



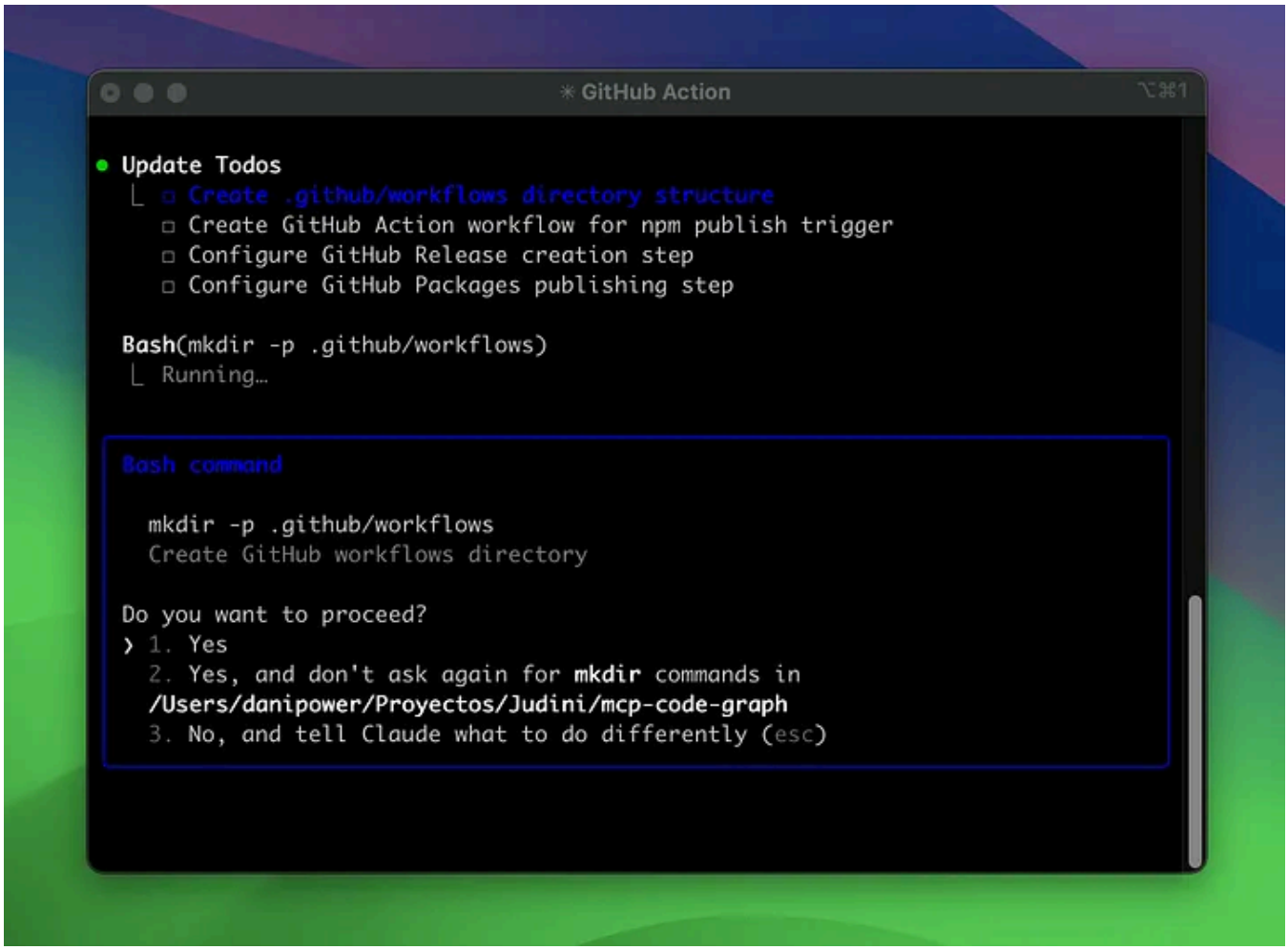
You can try different questions like these:

> what technologies does this project use?
> explain the folder structure

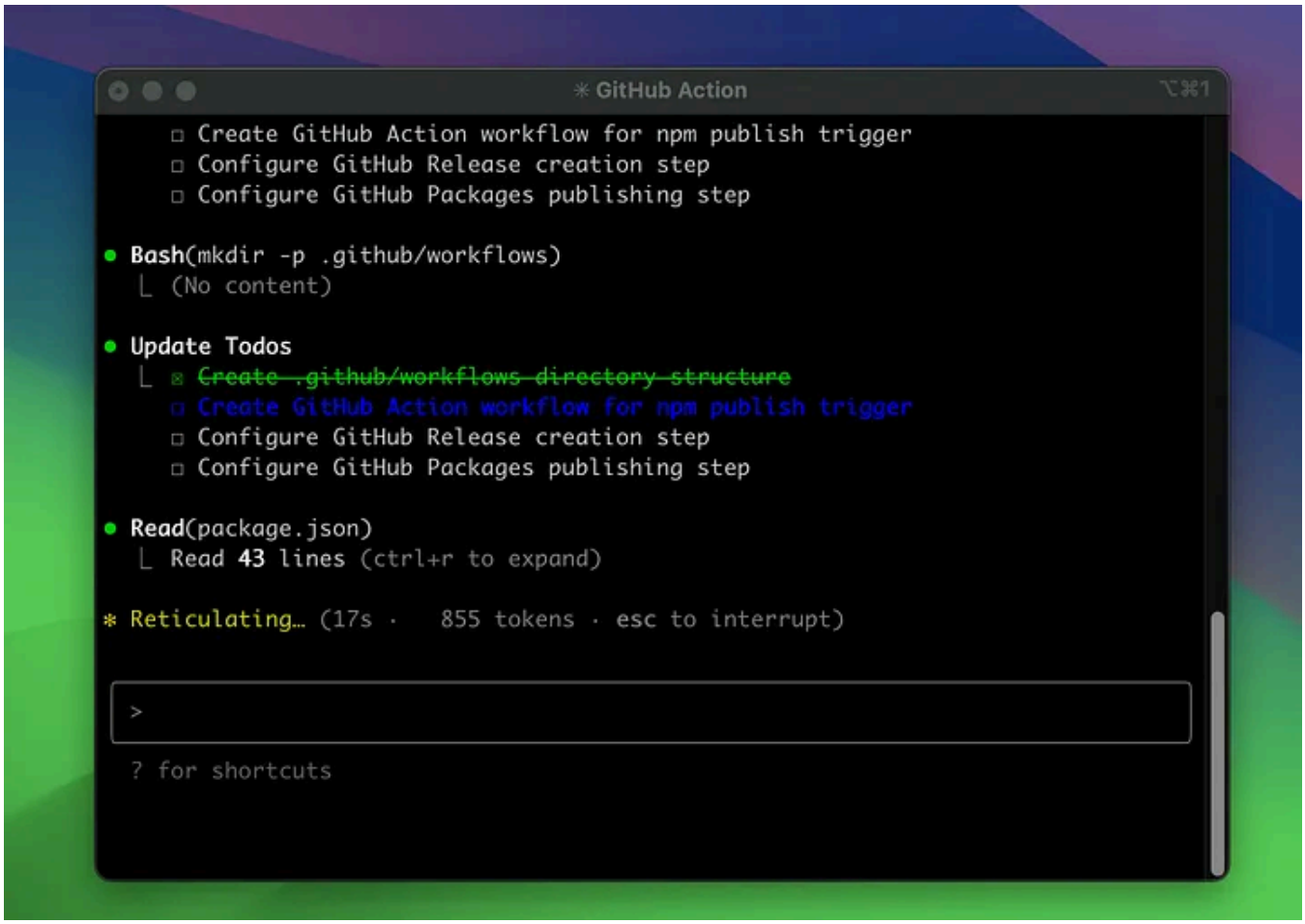
Make Code Changes

Instruct Claude to make edits:

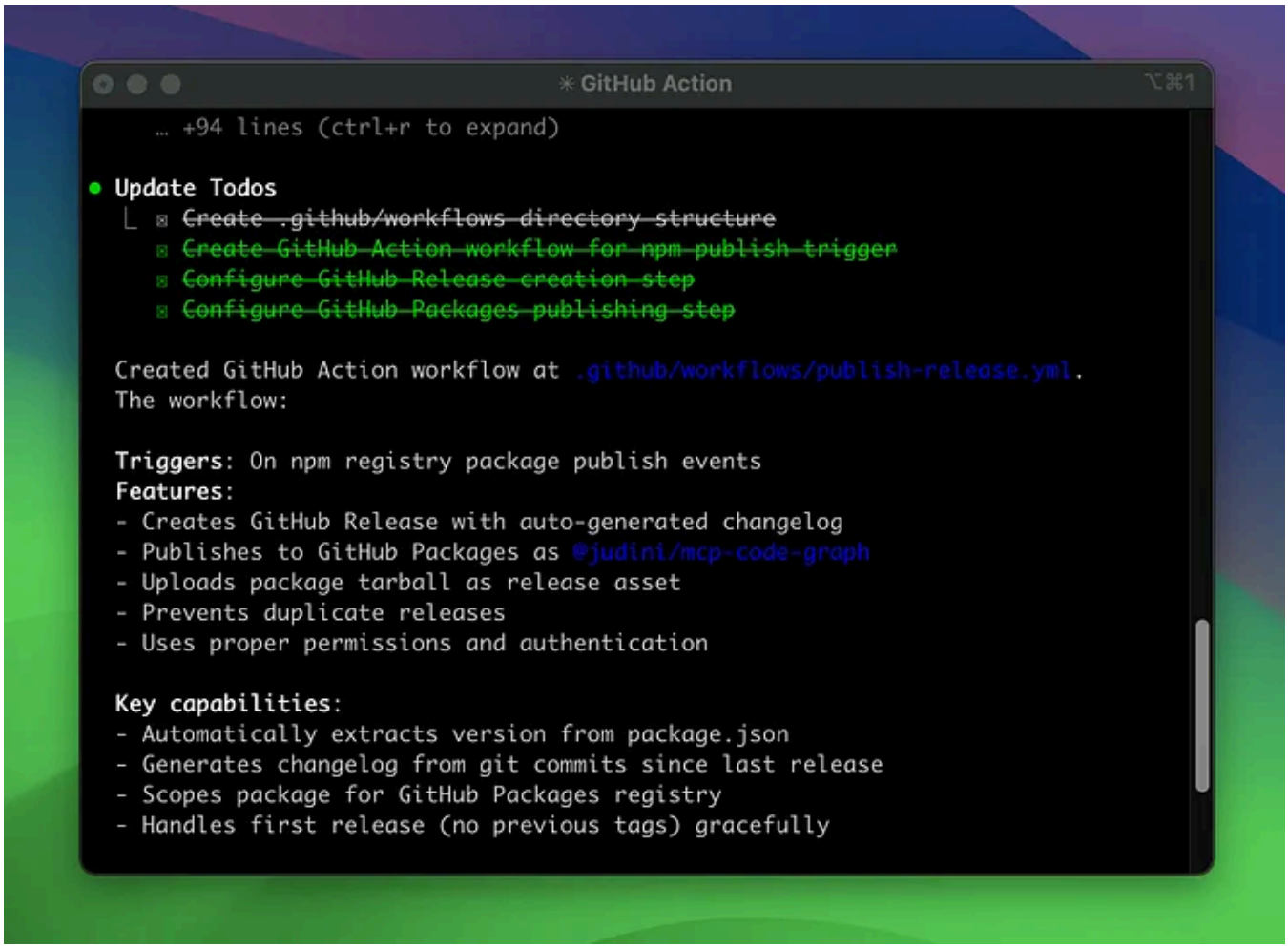
> Create a GitHub Action that, on every npm publish, automatically creates a GitHub Release and publishes the package to GitHub Packages.



Claude Code will show you how it’s performing the tasks one by one



When finished, it will give you a summary of the changes it just made



Claude will show you the proposed changes and ask for your approval before modifying any files.

3. Essential Commands and Workflows

Navigating the Command Line Interface (CLI)

Claude Code offers a simple yet powerful command-line interface with tab completion for files and commands. Use /help to see all available commands and /clear to reset the conversation context.

CLI commands

Command	Description	Example
`claude`	Start interactive REPL	`claude`
`claude "query"`	Start REPL with initial prompt	`claude "explain this project"`
`claude -p "query"`	Query via SDK, then exit	`claude -p "explain this function"`
`cat file | claude -p "query"`	Process piped content	`cat logs.txt | claude -p "explain"`
`claude -c`	Continue most recent conversation	`claude -c`
`claude -c -p "query"`	Continue via SDK	`claude -c -p "Check for type errors"`
`claude -r "<session-id>" "query"`	Resume session by ID	`claude -r "abc123" "Finish this PR"`
`claude update`	Update to latest version	`claude update`
`claude mcp`	Configure Model Context Protocol (MCP) servers	See the Claude Code MCP documentation.

Using Claude Code as a Unix-Style Utility

Command Breakdown: Security Scanning

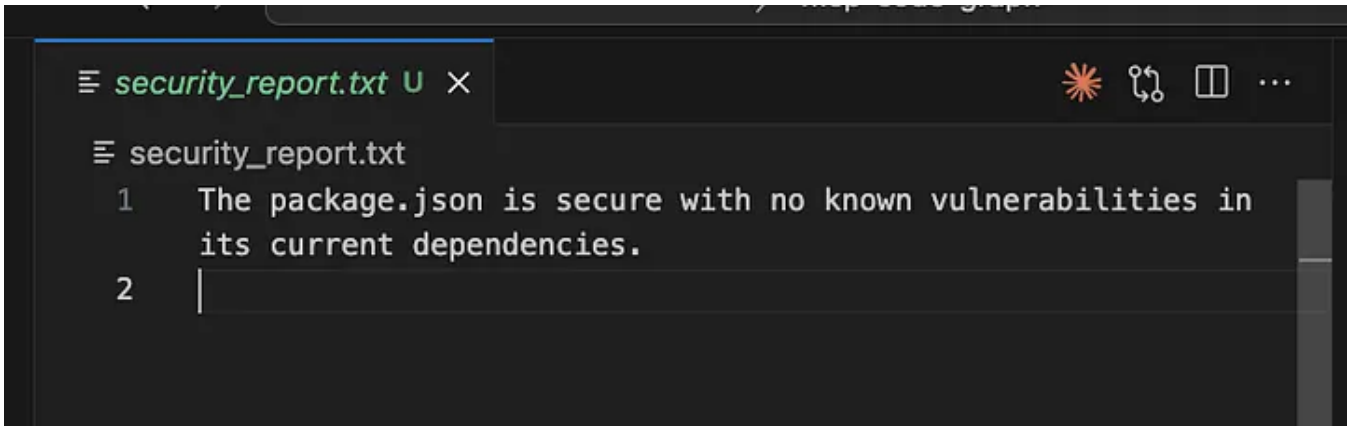
```
cat package.json | claude -p "review this file for security vulnerabilities a
```

This command demonstrates Claude Code's versatility as a Unix-style utility that can be integrated into your existing shell scripts and workflows. Let's break it down:

1. **Input Piping:** cat package.json | reads the contents of your package.json file and pipes it directly to Claude Code.
2. **Headless Mode:** The -p flag runs Claude in headless (non-interactive) mode with the specified prompt.
3. **Specialized Analysis:** The prompt instructs Claude to perform a security-focused code review, specifically looking for vulnerabilities and dependency issues.

4. **Output Redirection:** > security_report.txt captures Claude's analysis in a text file for documentation or further processing.

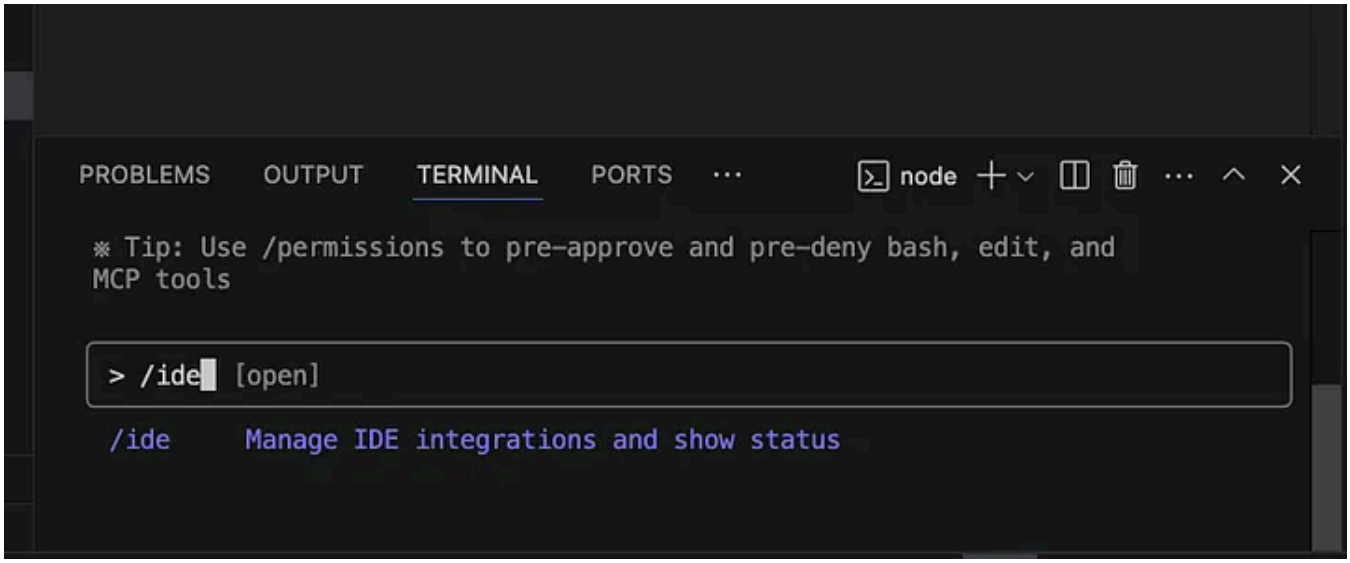
Result:



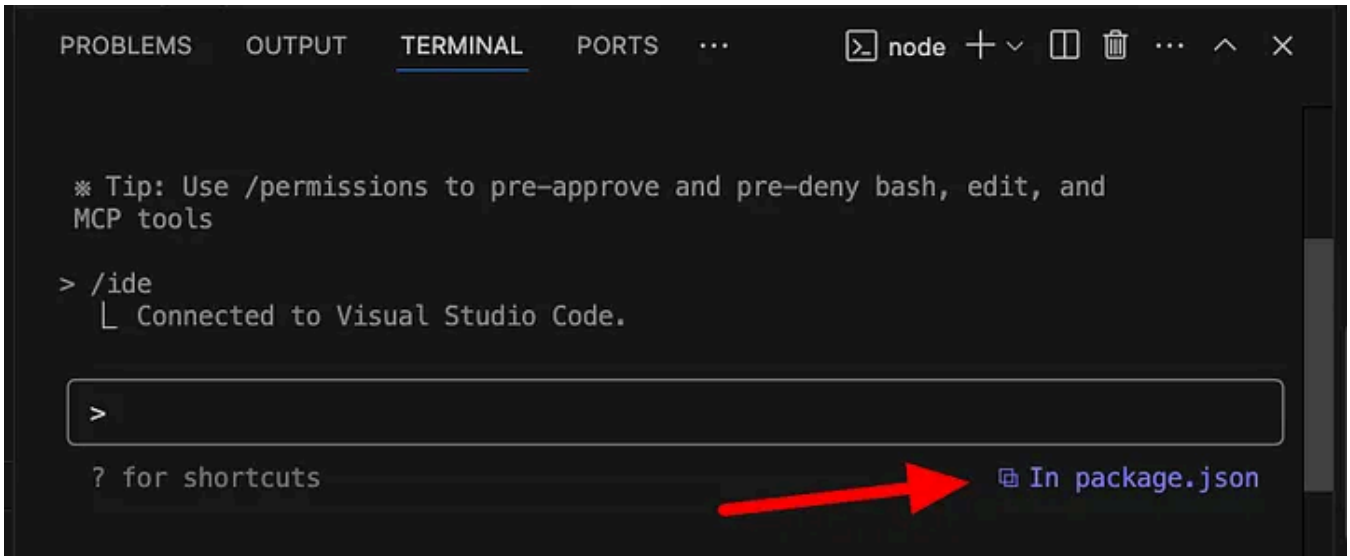
Working with your IDE

The /ide command connects Claude Code to your IDE (VS Code, Cursor, Windsorf or JetBrains), enabling powerful integrations:

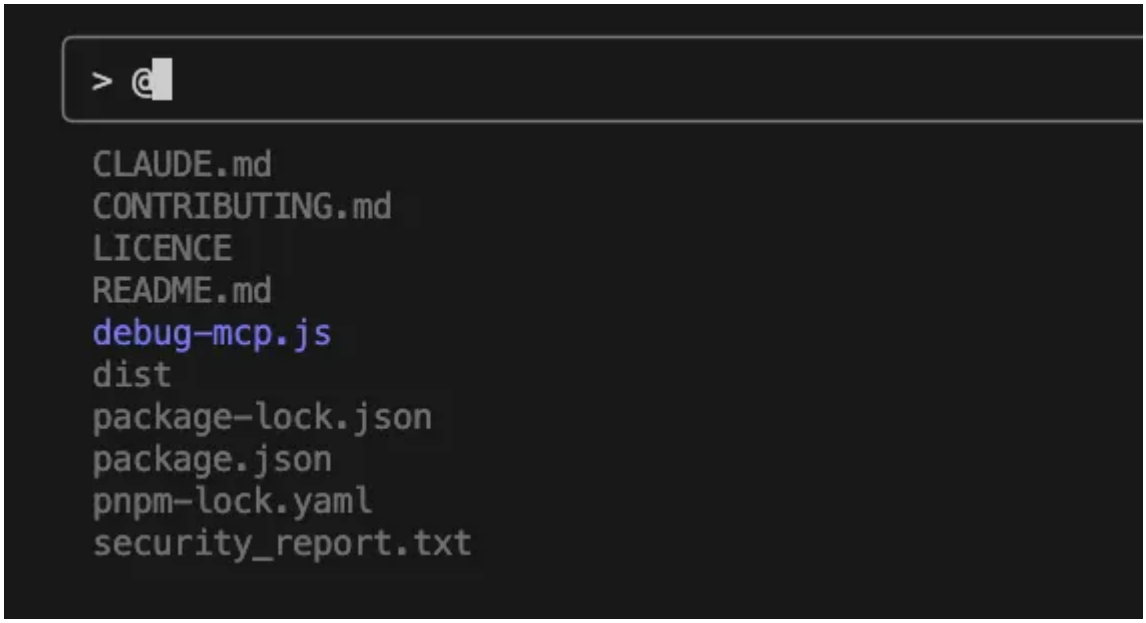
```
# Connect Claude to your IDE
> /ide
```



Automatic Context Sharing When you select files or code in your IDE, Claude automatically receives this context.



Or you can add a file as context using @



Creating Custom Slash Commands

The custom slash commands feature in Claude Code lets you create reusable prompts for common tasks:

```
# Create a project-specific command
mkdir -p .claude/commands
echo "Analyze this code for security vulnerabilities and suggest fixes:" > .
```

Project Commands vs. Personal Commands

Project Commands (shared with your team):

```
> /project:security-review
```

- Stored in .claude/commands/ directory
- Available to everyone who clones the repo
- Great for standardizing team workflows

Personal Commands (just for you):

```
> /user:optimize
```

- Stored in ~/.claude/commands/directory
- Available across all your projects
- Perfect for your individual preferences

Adding Command Arguments

Make commands flexible with the $ARGUMENTS placeholder:

```
# Create a command with arguments
echo "Find and fix issue #$ARGUMENTS. Follow these steps:
1. Understand the issue described in the ticket
2. Locate the relevant code
3. Implement a solution
4. Add appropriate tests" > .claude/commands/fix-issue.md
```

Then use it with:

```
> /project:fix-issue 123
```

Organizing Commands

You can create subdirectories for better organization:

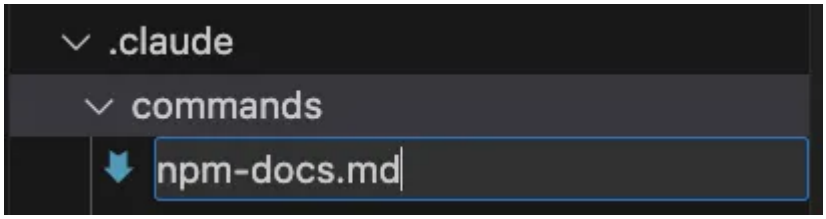
```
.claude/commands/frontend/component.md → /project:frontend:component
```

Custom commands transform Claude Code into a powerful, personalized coding assistant that adapts to your specific workflows and team standards.

Creating Custom Slash Commands for npm Packages

Here's a practical example of creating a custom slash command for npm package development:

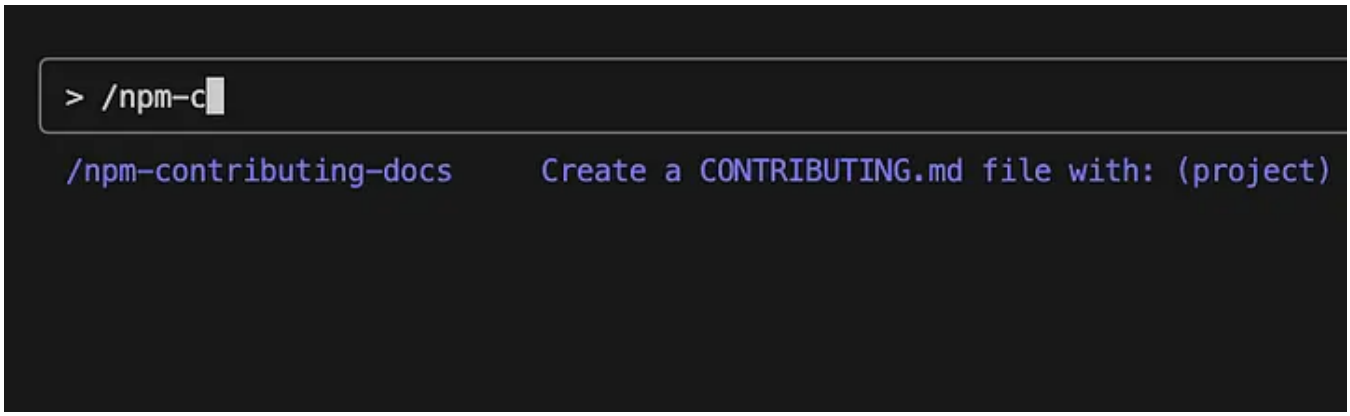
Create the file .claude/commands/npm-contributing-docs.md



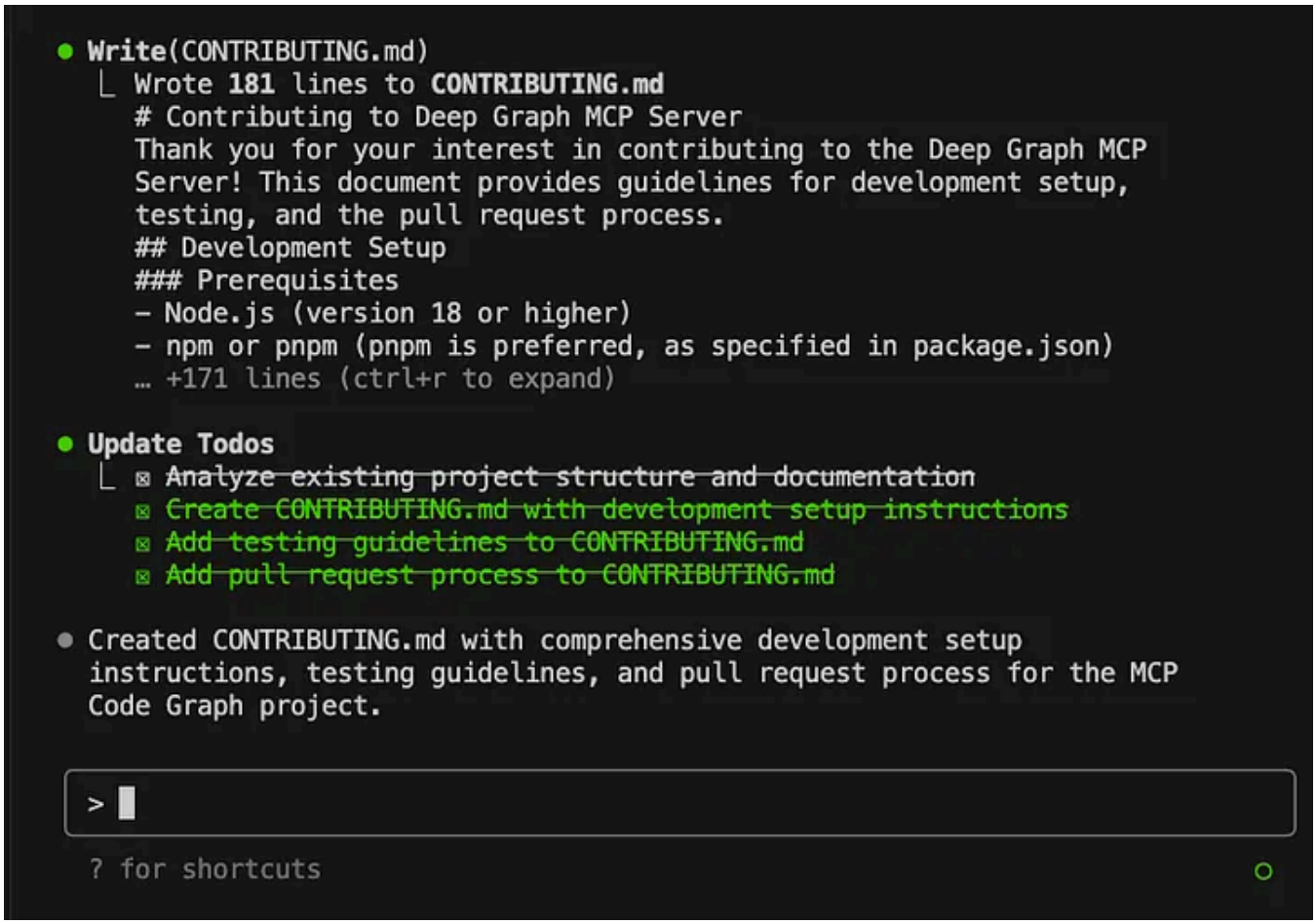
With the following content:

```
Create a CONTRIBUTING.md file with:
   - Development setup instructions
   - Testing guidelines
   - Pull request process
```

Now you can use this command in your npm package project (you must restart claude to see the command):



The command will be executed and in this case will create the CONTRIBUTING.md file as we requested in the created command file.



Working with Model Context Protocol (MCP)

MCP allows Claude Code to connect with external tools and data sources, extending its capabilities beyond your local environment.

```
# Basic MCP server addition
claude mcp add postgres-db -- /path/to/postgres-mcp-server --connection-strin
```

Managing MCP Servers

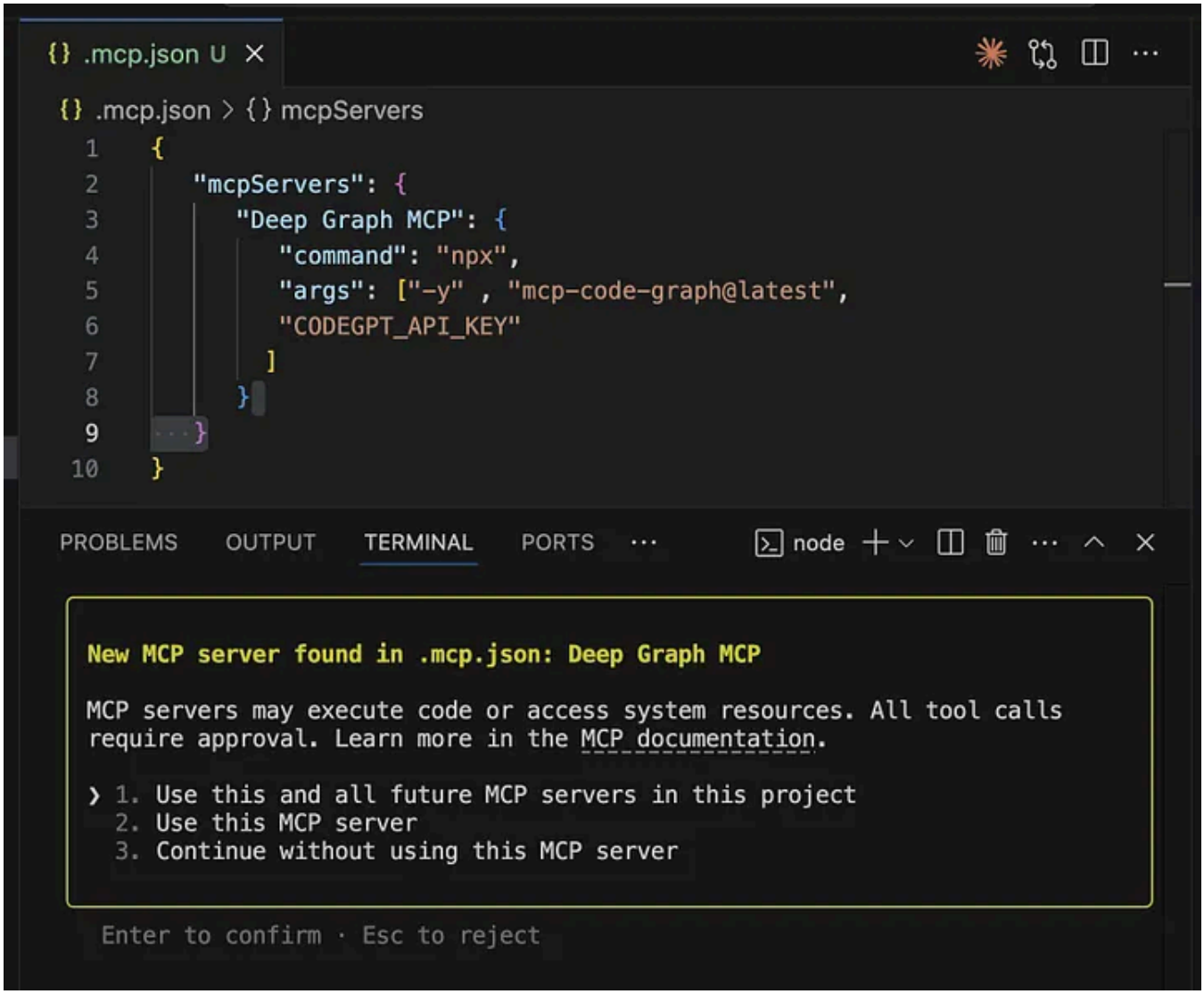
```
# List all configured servers
claude mcp list

# Get details for a specific server
claude mcp get postgres-db

# Remove a server
claude mcp remove postgres-db
```

Scopes for MCP Servers

- Local (-s local): Available only to you in the current project
- Project (-s project): Shared with everyone via .mcp.json file



- User (-s user): Available to you across all projects

Example: Using CodeGPT's Deep Graph MCP

Let's see how to implement a powerful code analysis MCP in practice:

Let's integrate the Deep Graph MCP with Claude Code. This integration transforms Claude Code into a powerful code comprehension tool that understands your entire codebase at a semantic level, making it ideal for large, complex projects.

Here you can see all the documentation for this CodeGPT MCP and all the information to use it in Claude Code: https://github.com/JudiniLabs/mcp-code-graph

Add the MCP with the following command and don't forget to add your CodeGPT API Key

```
# Add the Deep Graph MCP server
claude mcp add "Deep Graph MCP" npx -- -y mcp-code-graph@latest YOUR_CODEGPT_
```

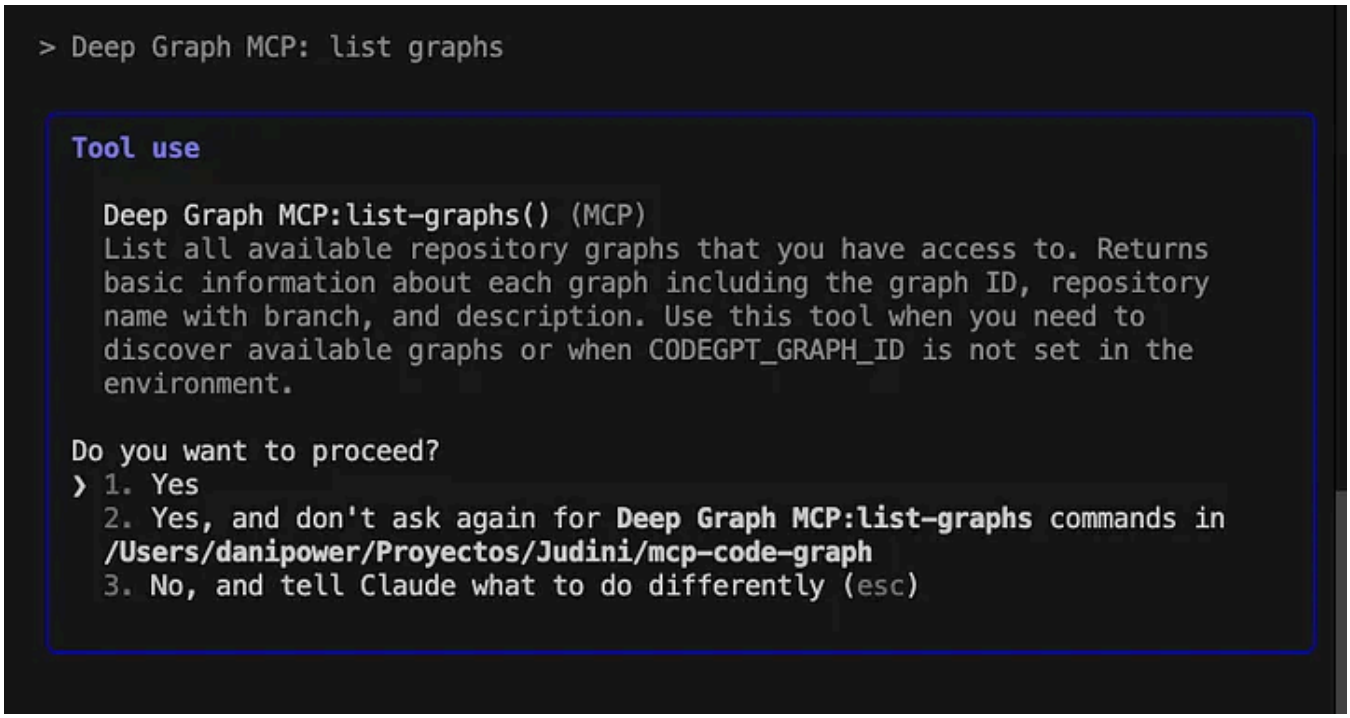
Interacting with Deep Graph MCP

Once configured, you can use it in your Claude Code sessions:

Check if you have Deep Graph MCP installed by running "claude mcp list"

To call the MCP and use its tools, you can mention it and then request something directly. Claude Code will detect the tools it needs to execute and show you the confirmation message:

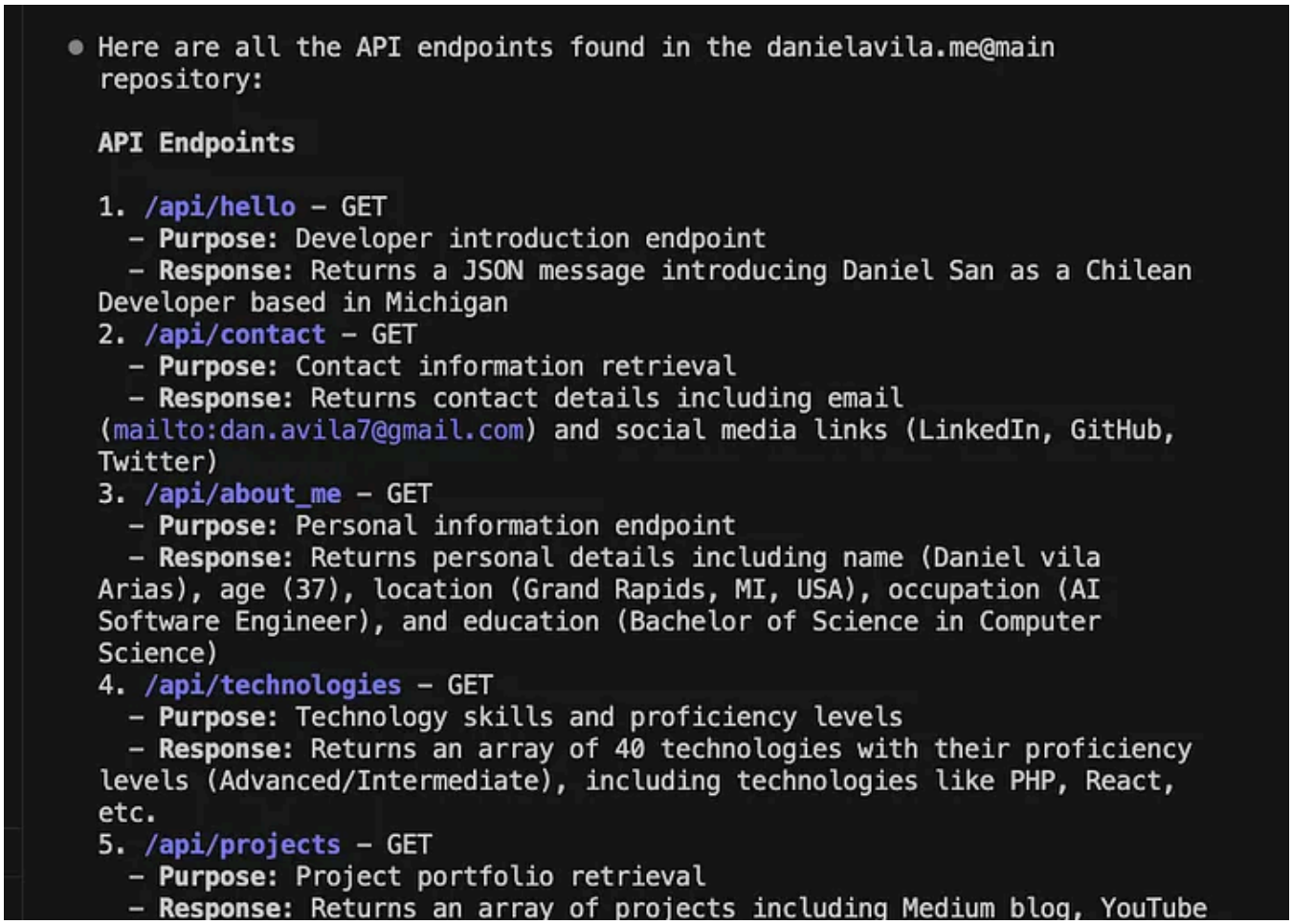
>Deep Graph MCP: list graphs



With the Deep Graph MCP you can work with all the repositories you have converted into knowledge graphs.

Once your repositories are listed, you could ask Claude Code to work directly with a graph.

> Deep Graph MCP: show all the endpoints from danielavila.me@main



Claude Code will execute the corresponding tools with the selected graph and finally you'll have a direct response in the console.

Amazing! Now you can work with any repository within your project using Claude Code, special commands, Deep Graph MCP and all the tools that Claude has available!