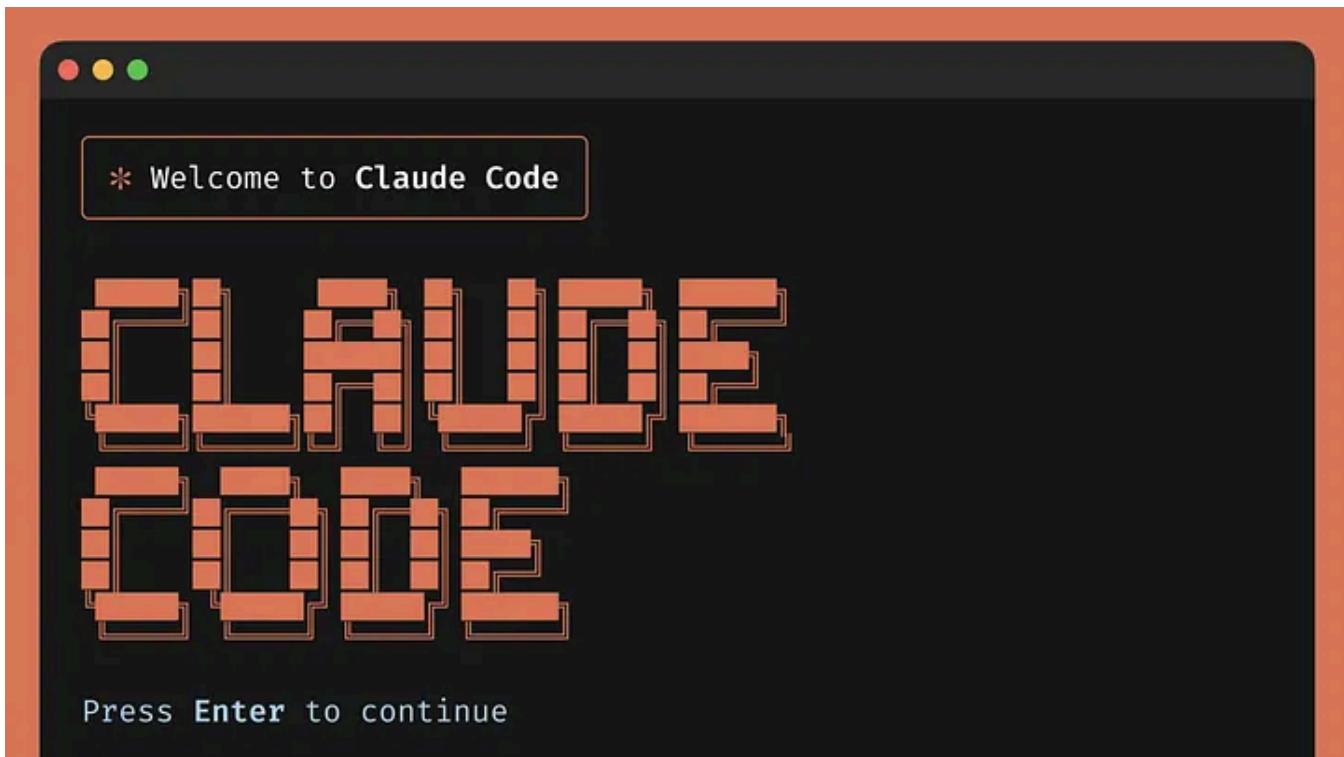


# Claude Code: From Zero to Hero



## 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.

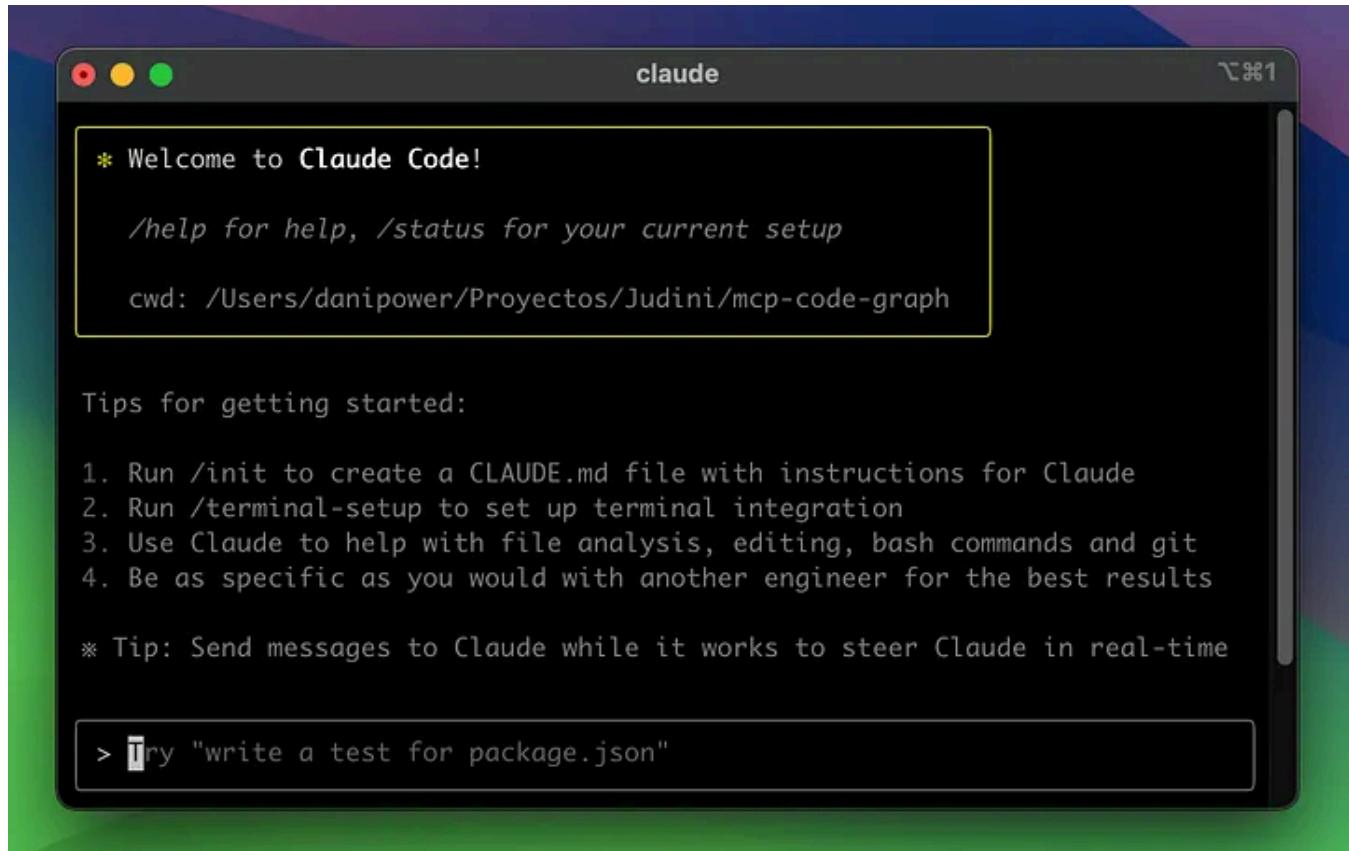
- **MCP Integration**
- **Secure & Private:** Direct API connection to Anthropic, operating within your terminal.

## Installation and Authentication

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

```
npm install -g @anthropic-ai/clause-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

```
> /config
└ (no content)
```

**Settings**  
Configure Claude Code preferences

Auto-compact	true
Use todo list	true
Verbose output	true
Theme	Light mode (ANSI colors only)
Notifications	auto
Editor mode	normal
Model	Default (recommended)
Diff tool	auto

↑/↓ to select · Enter/Tab/Space to change · Esc to close

## 1. Initializing Your Project

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

```
> /init
```

/init	Initialize a new CLAUDE.md file with codebase documentation
/install-github-app	Set up Claude GitHub Actions for a repository
/migrate-installer	Migrate from global npm installation to local installation
/compact	Clear conversation history but keep a summary in context. Optional: /compact [instructions for summarization]
/doctor	Checks the health of your Claude Code installation
/ide	Manage IDE integrations and show status
/terminal-setup	Install Shift+Enter key binding for newlines
/upgrade	Upgrade to Max for higher rate limits and more Opus

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

```
claude
```

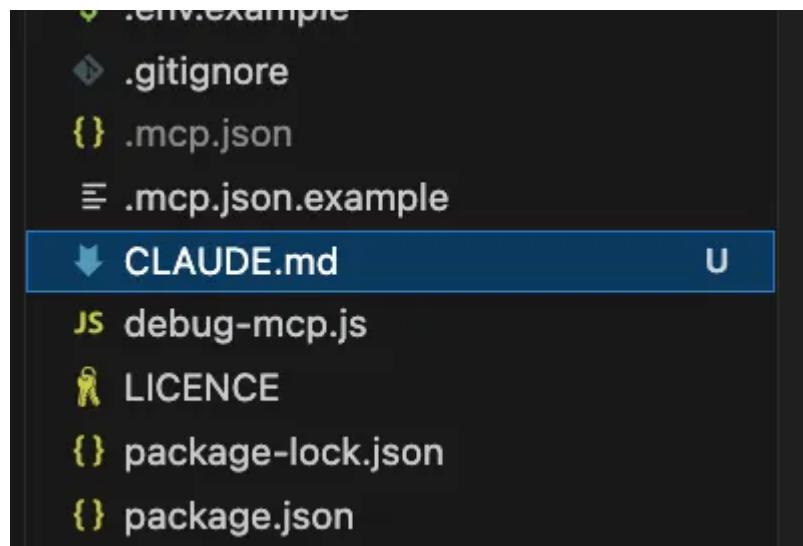
```
- Uses `@modelcontextprotocol/sdk` for MCP protocol implementation
- Implements dynamic schema generation based on environment variables
- Includes extensive error handling and logging
- TypeScript with ES2022 target and NodeNext modules
- Communicates with CodeGPT API at `https://api-mcp.codegpt.co/api/v1`

## Publishing

The package is published to npm as `mcp-code-graph` with binary entry point at `dist/index.js`.

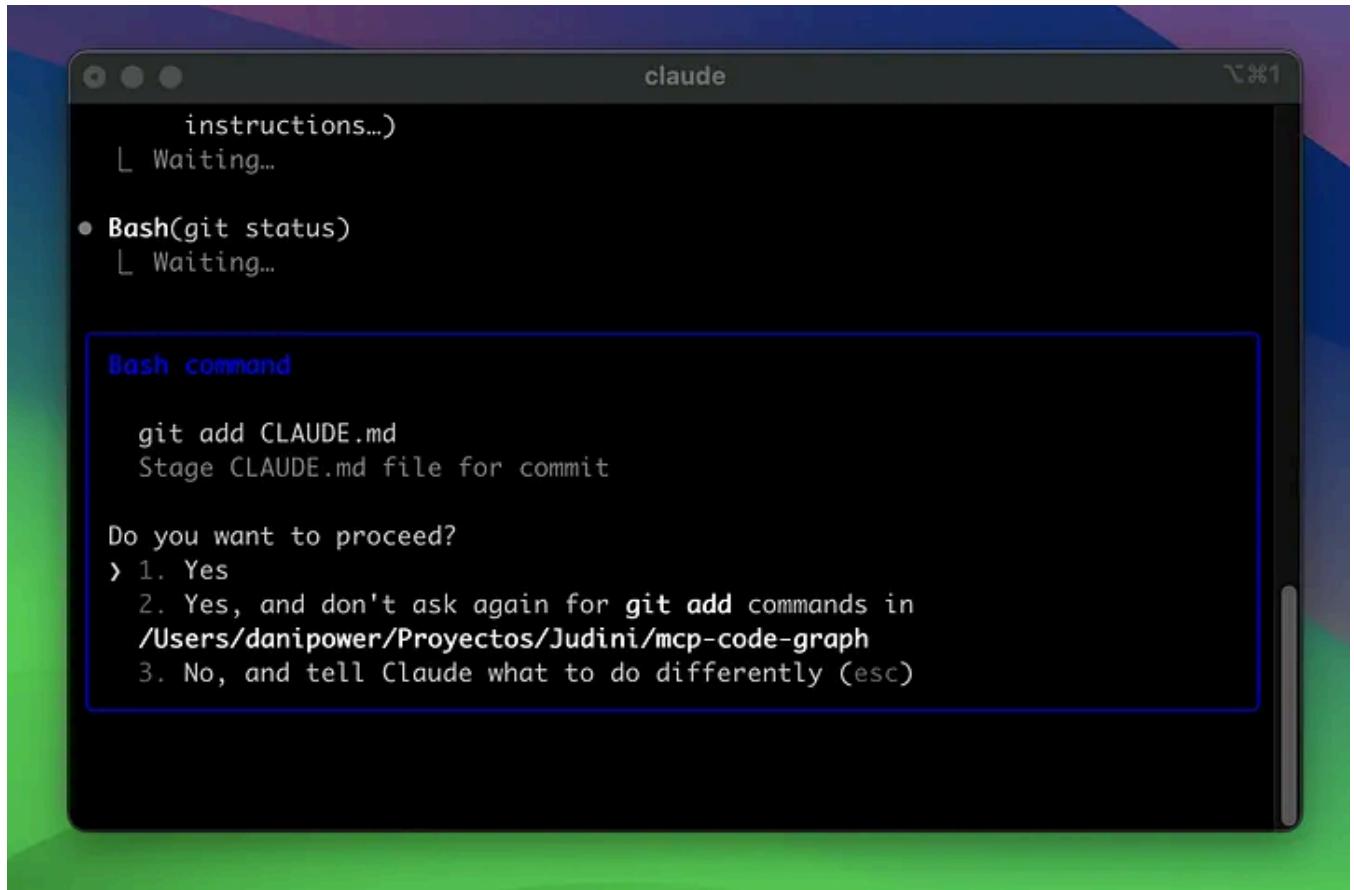
Do you want to create CLAUDE.md?
> 1. Yes
  2. Yes, and don't ask again this session (shift+tab)
  3. No, and tell Claude what to do differently (esc)
```

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.

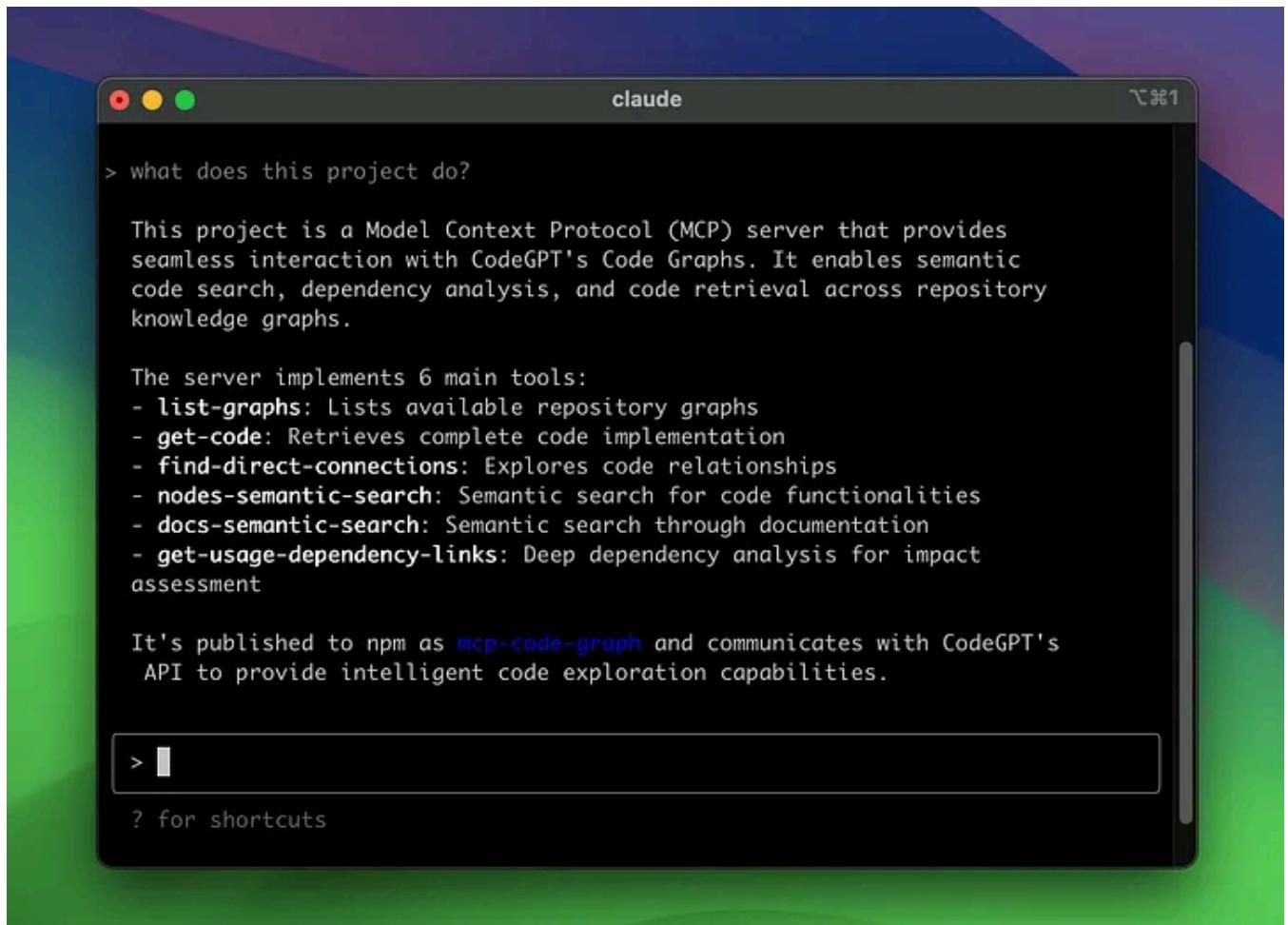


## 2. Basic Usage

### Asking Questions

Start by understanding your codebase:

> ***what does this project do?***



You can try different questions like these:

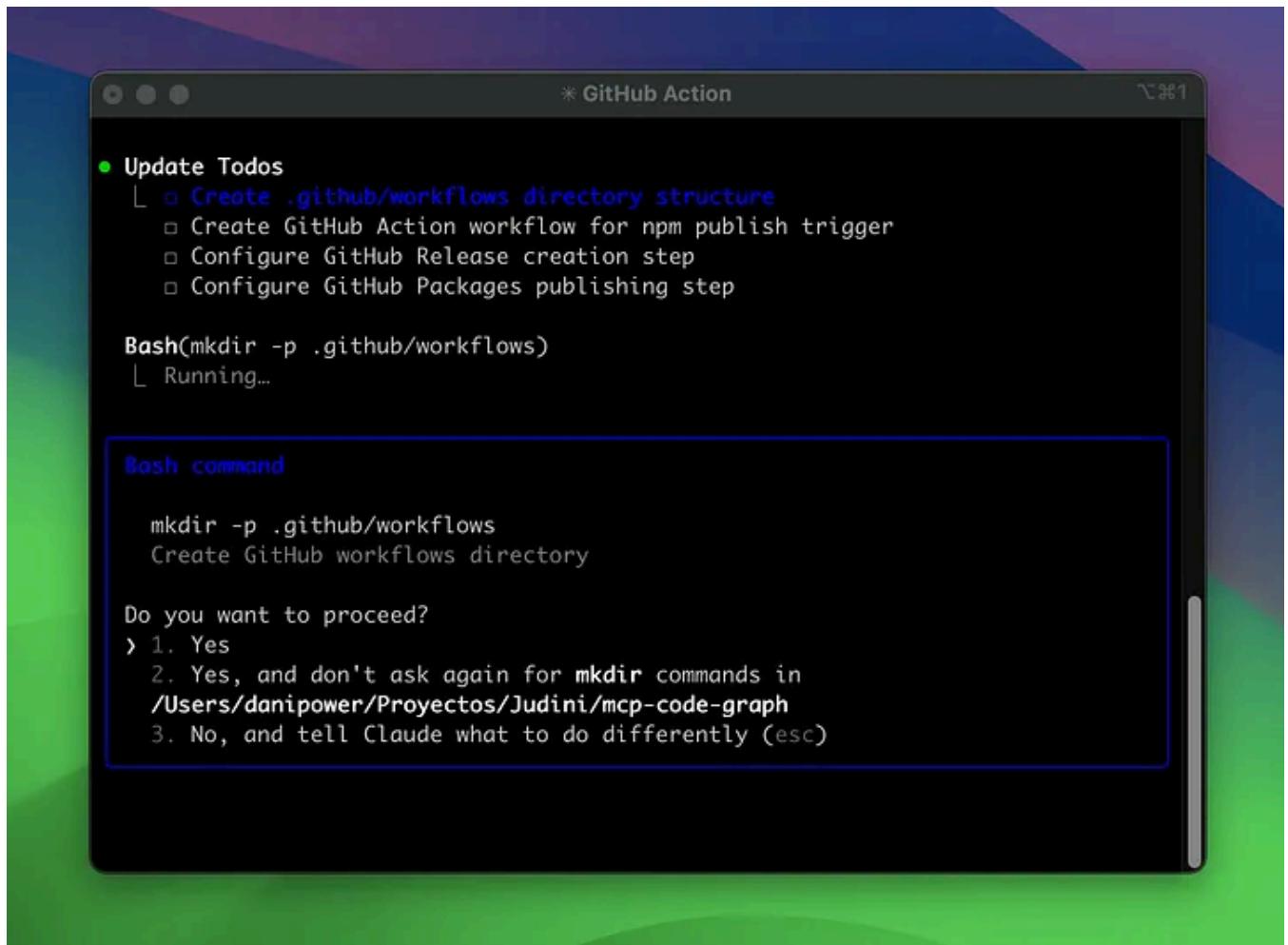
> ***what technologies does this project use?***

> ***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

The screenshot shows a terminal window with the title bar 'GitHub Action'. The terminal displays a todo list and some file operations:

- ☐ Create GitHub Action workflow for npm publish trigger
- ☐ Configure GitHub Release creation step
- ☐ Configure GitHub Packages publishing step

● Bash(mkdir -p .github/workflows)  
└ (No content)

● Update Todos  
└ ☐ [Create .github/workflows directory structure](#)  
☐ Create GitHub Action workflow for npm publish trigger  
☐ Configure GitHub Release creation step  
☐ Configure GitHub Packages publishing step

● Read(package.json)  
└ Read 43 lines (ctrl+r to expand)

\* Reticulating... (17s · 855 tokens · esc to interrupt)

A command-line input field contains a single character '>'. Below it, a help message reads '? for shortcuts'.

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

The terminal window shows the following content:

```
* GitHub Action
... +94 lines (ctrl+r to expand)

● Update Todos
└─ Create .github/workflows directory structure
  └─ Create GitHub Action workflow for npm publish trigger
    └─ Configure GitHub Release creation step
    └─ Configure GitHub Packages publishing step

Created GitHub Action workflow at .github/workflows/publish-release.yml.
The workflow:

Triggers: On npm registry package publish events
Features:
- Creates GitHub Release with auto-generated changelog
- Publishes to GitHub Packages as @judini/mcp-code-graph
- Uploads package tarball as release asset
- Prevents duplicate releases
- Uses proper permissions and authentication

Key capabilities:
- Automatically extracts version from package.json
- Generates changelog from git commits since last release
- Scopes package for GitHub Packages registry
- Handles first release (no previous tags) gracefully
```

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

## Using Claude Code as a Unix-Style Utility

### Command Breakdown: Security Scanning

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

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.

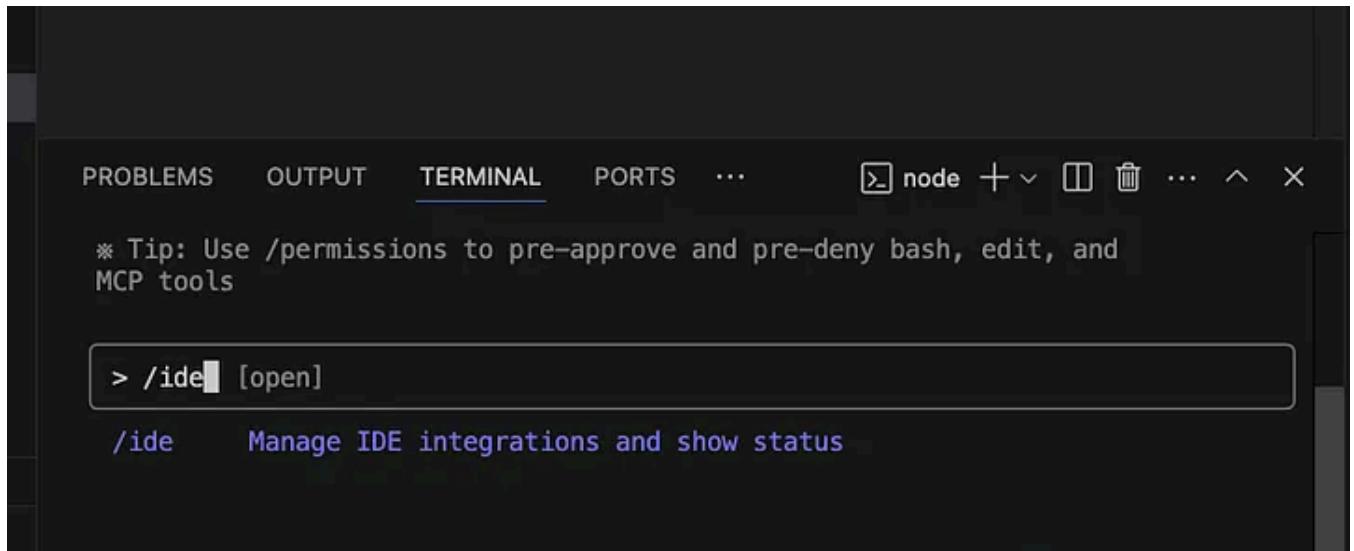
## Result:

```
≡ security_report.txt U X
≡ security_report.txt
1 The package.json is secure with no known vulnerabilities in
  its current dependencies.
2 |
```

# Working with your IDE

The `/ide` command connects Claude Code to your IDE (VS Code, Cursor, Windsor 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.

The screenshot shows the Visual Studio Code interface with the 'TERMINAL' tab selected. In the terminal, the command '/ide' has been entered, followed by a space. A context menu is open, with the option 'In package.json' highlighted and indicated by a red arrow. The menu also includes 'In current file', 'In workspace', and 'In project'. The status bar at the bottom shows the text '? for shortcuts'.

Or you can add a file as context using @

The screenshot shows the Visual Studio Code interface with the 'TERMINAL' tab selected. In the terminal, the command '@' has been entered, followed by a space. A context menu is open, listing files and folders from the current directory: CLAUDE.md, CONTRIBUTING.md, LICENCE, README.md, debug-mcp.js, dist, package-lock.json, package.json, pnpm-lock.yaml, and security\_report.txt.

## 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:

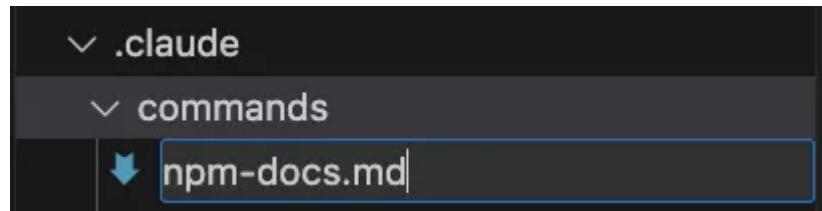
```
.claude/commands/frontend/component.md → /project:frontend:component
```

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):

```
> /npm-c
```

```
/npm-contributing-docs      Create a CONTRIBUTING.md file with: (project)
```

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

- **Write**(CONTRIBUTING.md)
  - └ Wrote 181 lines to **CONTRIBUTING.md**  
# Contributing to Deep Graph MCP Server  
Thank you for your interest in contributing to the Deep Graph MCP Server! This document provides guidelines for development setup, testing, and the pull request process.  
## Development Setup  
### Prerequisites
    - Node.js (version 18 or higher)
    - npm or pnpm (pnpm is preferred, as specified in package.json)... +171 lines (ctrl+r to expand)
- **Update Todos**
  - └ ✅ Analyze existing project structure and documentation
  - ✅ Create CONTRIBUTING.md with development setup instructions
  - ✅ Add testing guidelines to CONTRIBUTING.md
  - ✅ Add pull request process to CONTRIBUTING.md
- Created CONTRIBUTING.md with comprehensive development setup instructions, testing guidelines, and pull request process for the MCP Code Graph project.

```
> █
```

? for shortcuts



## 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-string
```

## 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

The screenshot shows a dark-themed VS Code interface. At the top, there's a tab bar with 'PROBLEMS', 'OUTPUT', 'TERMINAL' (which is underlined), 'PORTS', and '...'. Below the tabs, there's a toolbar with icons for node, plus, minus, and other project-related functions. The main editor area contains the following JSON code:

```
{}
{} .mcp.json U X
{} .mcp.json > {} mcpServers
1  {
2    "mcpServers": {
3      "Deep Graph MCP": {
4        "command": "npx",
5        "args": ["-y" , "mcp-code-graph@latest",
6        "CODEGPT_API_KEY"
7      }
8    }
9  }
10 }
```

Below the editor, a modal dialog box has a yellow border. It displays the message: "New MCP server found in .mcp.json: Deep Graph MCP". It also states: "MCP servers may execute code or access system resources. All tool calls require approval. Learn more in the [MCP documentation](#)". Three numbered options are listed: "1. Use this and all future MCP servers in this project", "2. Use this MCP server", and "3. Continue without using this MCP server". At the bottom of the modal, there are two text inputs: "Enter to confirm" and "Esc to reject".

- 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”

- 
- You have one MCP server configured: "Deep Graph MCP" using the mcp-code-graph package.

```
> █
```

```
? for shortcuts
```

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
```

```
> Deep Graph MCP: list graphs
```

#### Tool use

**Deep Graph MCP:list-graphs()** (MCP)

List all available repository graphs that you have access to. Returns basic information about each graph including the graph ID, repository name with branch, and description. Use this tool when you need to discover available graphs or when CODEGPT\_GRAPH\_ID is not set in the environment.

Do you want to proceed?

- › 1. Yes
- 2. Yes, and don't ask again for **Deep Graph MCP:list-graphs** commands in **/Users/danipower/Proyectos/Judini/mcp-code-graph**
- 3. No, and tell Claude what to do differently (esc)

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
```

- Here are all the API endpoints found in the danielavila.me@main repository:

## API Endpoints

1. **/api/hello** – GET
  - Purpose: Developer introduction endpoint
  - Response: Returns a JSON message introducing Daniel San as a Chilean Developer based in Michigan
2. **/api/contact** – GET
  - Purpose: Contact information retrieval
  - Response: Returns contact details including email (<mailto:dan.avila7@gmail.com>) and social media links (LinkedIn, GitHub, Twitter)
3. **/api/about\_me** – GET
  - Purpose: Personal information endpoint
  - Response: Returns personal details including name (Daniel vila Arias), age (37), location (Grand Rapids, MI, USA), occupation (AI Software Engineer), and education (Bachelor of Science in Computer Science)
4. **/api/technologies** – GET
  - Purpose: Technology skills and proficiency levels
  - Response: Returns an array of 40 technologies with their proficiency levels (Advanced/Intermediate), including technologies like PHP, React, etc.
5. **/api/projects** – GET
  - Purpose: Project portfolio retrieval
  - Response: Returns an array of projects including Medium blog, YouTube

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!