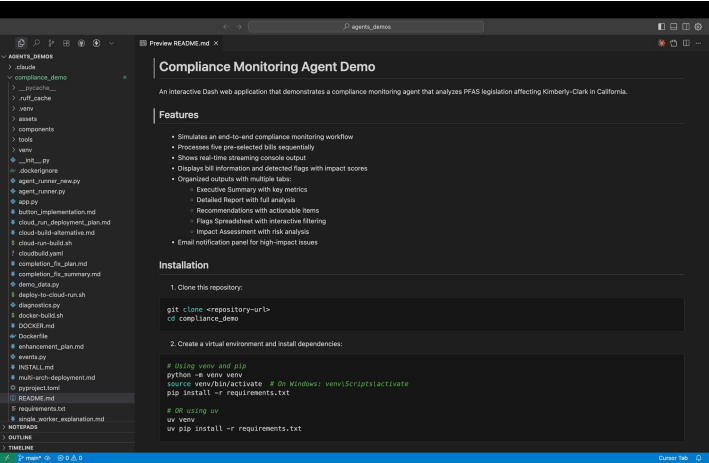
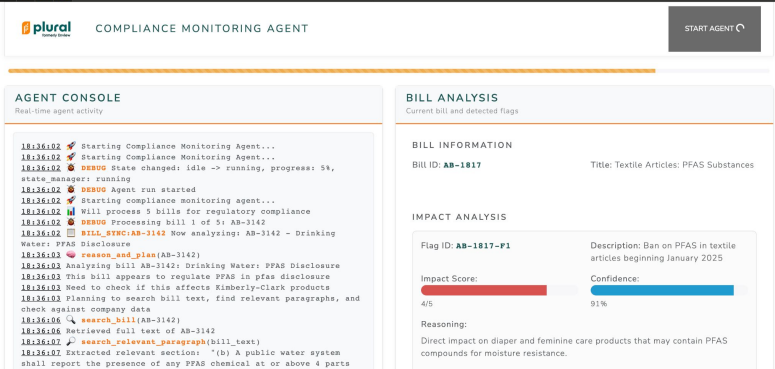
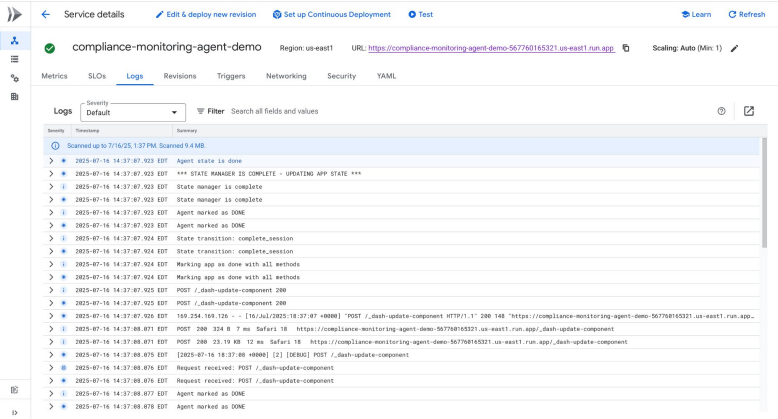


Claude Code

Why it matters, and how to get started

Why it matters

Claude Code is able to semi autonomously build software solutions near the level of an average junior engineer.



Claude Code

Terminal application that uses
Anthropic models to write and run
code

Is:

The closest thing to an
autonomous SWE agent.

Isn't:

IDE code autocomplete

Human level SWE (yet)

**Similar
Agents:**



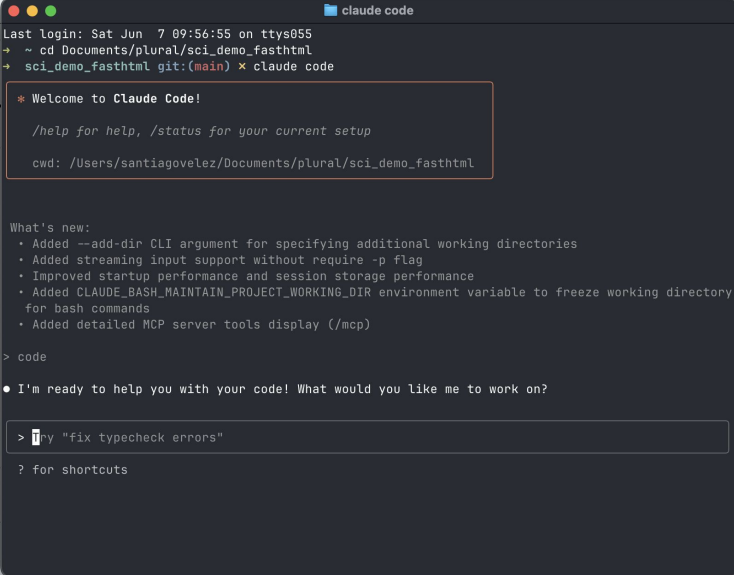
How to

Install:

- Open your terminal
- Install [node.js](#) 18+
- Run `npm install -g @anthropic-ai/claude-code`

Run:

1. Navigate to your repo – ``cd <path>``
2. Write ``claude code`` and press enter
 - `--continue`: flag to load conversation history
 - `--model`: `opus` is best, `sonnet` is cheaper
3. Log to your account using ``/login`` command
4. Write your prompt and press enter



```
claude code
Last login: Sat Jun  7 09:56:55 on ttys055
→ ~ cd Documents/plural/sci_demo_fasthtml
→ sci_demo_fasthtml git:(main) × claude code

* Welcome to Claude Code!

/help for help, /status for your current setup

cwd: /Users/santiagovelez/Documents/plural/sci_demo_fasthtml

What's new:
• Added --add-dir CLI argument for specifying additional working directories
• Added streaming input support without require -p flag
• Improved startup performance and session storage performance
• Added CLAUDE_BASH_MAINTAIN_PROJECT_WORKING_DIR environment variable to freeze working directory for bash commands
• Added detailed MCP server tools display (/mcp)

> code

• I'm ready to help you with your code! What would you like me to work on?

> fix "fix typecheck errors"

? for shortcuts
```

Usage Patterns

Small Production Tasks

When:

- Small changes
- Simple bug fixes
- Ask the codebase
- Ask about documentation

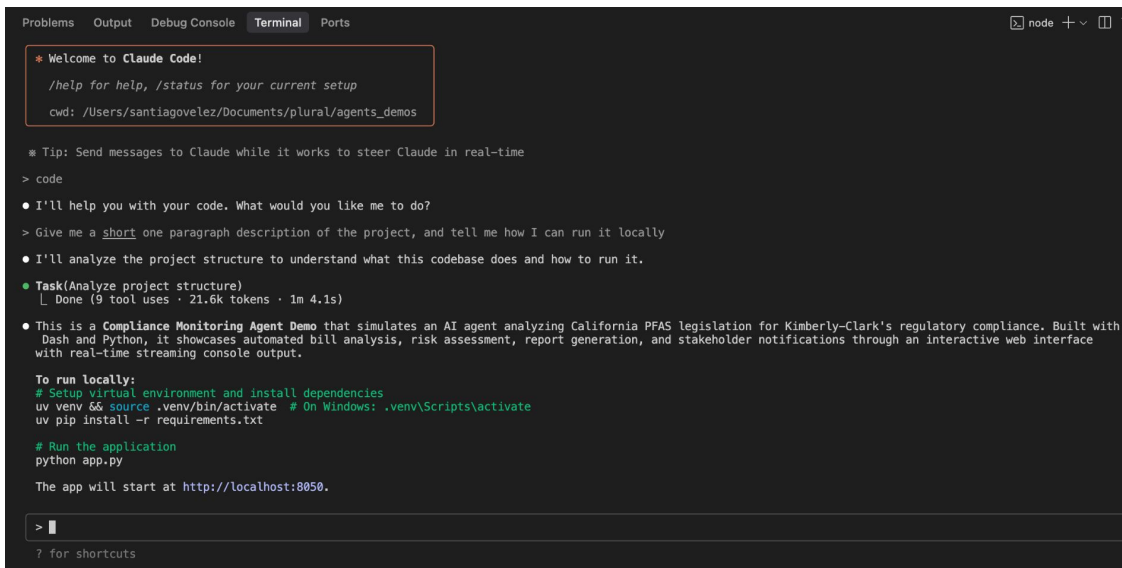
How:

- Short prompt to the point
- Model or Reasoning barely matter.
- Auto-accept code is optional
- Always use version control

Small Production Tasks

Prompt example:

> Give me a short one paragraph description of the project, and tell me how I can run it locally



```
Problems Output Debug Console Terminal Ports
* Welcome to Claude Code!
/help for help, /status for your current setup
cwd: /Users/santiagoavelez/Documents/plural/agents_demos

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

> code
• I'll help you with your code. What would you like me to do?
> Give me a short one paragraph description of the project, and tell me how I can run it locally
• I'll analyze the project structure to understand what this codebase does and how to run it.
• Task(Analyze project structure)
  | Done (9 tool uses · 21.6k tokens · 1m 4.1s)
• This is a Compliance Monitoring Agent Demo that simulates an AI agent analyzing California PFAS legislation for Kimberly-Clark's regulatory compliance. Built with Dash and Python, it showcases automated bill analysis, risk assessment, report generation, and stakeholder notifications through an interactive web interface with real-time streaming console output.

To run locally:
# Setup virtual environment and install dependencies
uv venv && source .venv/bin/activate # On Windows: .venv\Scripts\activate
uv pip install -r requirements.txt

# Run the application
python app.py

The app will start at http://localhost:8050.

> |
? for shortcuts
```

Medium Production Tasks

When:

- Basic research
- Write documentation
- Code new features
- Write simple unit tests

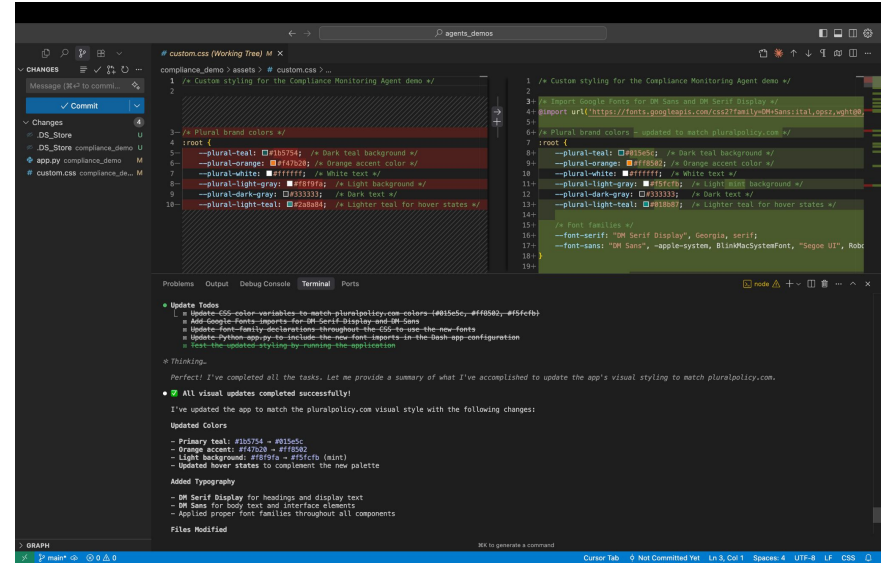
How:

- Detailed prompt (~1 paragraph)
- Usually good to plan first
- Auto-accept code usually works, but always review the results
- Sonnet may be good enough with reasoning prompts
- Always use version control

Medium Production Tasks

Prompt example:

> Update the app visuals by changing the main colors of the dashboard. The new colors should be similar to the ones present in this reference <https://pluralpolicy.com> for background, buttons, words, etc. Use the same font too. Make sure to update all the visual elements of the app in the python and css files. The main hex colors are #015e5c, #ff8502 and #f5fcfb. think hard .



Large Production Tasks

When:

- Complete applications
- Complex new features
- Large technical challenges
- Architecture recommendations

How:

- Write a large and complete prompt (+1 page)
- Include references, visual guides, and other additional resources.
- Always plan first. Review the plan multiple times with your manual edits.
- Write the plan for future reference. Split into todo list elements
- Follow the implementation closely, disable auto-accept code
- Always use Opus 4 Model
- Always use the strongest reasoning
- Always use version control

Vibecoding

When:

- Demos
- Independent experiments
- One use solutions

Notes:

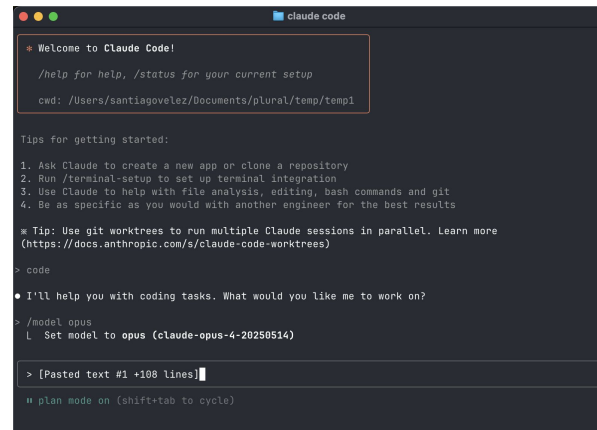
- This is a bad idea in my experience if you need to maintain the software
- Results are usually not production ready
- Small changes to vibe coded software may be more complex than expected

How:

- Write a large and complete prompt (+1 page)
- Include important details like language, deployment strategy, etc.
- Include references, visual guides, and other additional resources.
- Use Claude to write a Plan, and ask to modify as needed.
- If the project is too complex, ask Claude to split the development in different stages
- Let Claude loose and wait until results are done
- Test the results, usually they have errors or inconsistencies
- Copy error logs and screenshot visuals into the prompt, ask for a plan to fix the errors.
- Approve the plan and let Claude loose again.
- Always use version control
- Limit the use of tools outside the project

Prompt structure example:

- Role + task instruction
- Business context (goal, outcome, timeline)
- Software use cases w/ inputs & outputs
- Web pages spec (pages + behaviors)
- Mocked data requirement (external files)
- Simulated behavior
- Branding / visual assets
- Tests coverage areas
- Sample log format (style guide)
- Deliverables & repo structure guidance



```
claude code

Welcome to Claude Code!

/help for help, /status for your current setup
cwd: /Users/santiagovelez/Documents/plural/temp/temp1

Tips for getting started:

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

Tip: Use git worktrees to run multiple Claude sessions in parallel. Learn more
(https://docs.anthropic.com/s/claude-code-worktrees)

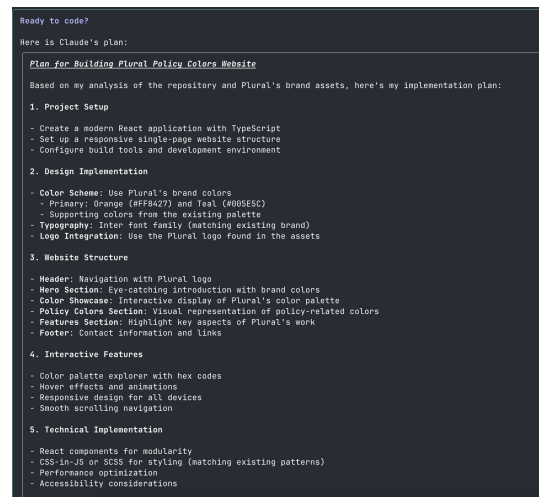
> code

• I'll help you with coding tasks. What would you like me to work on?

> /model opus
  L Set model to opus (claude-opus-4-20250514)

> [Pasted text #1 +108 Lines]

plan mode on (shift+tab to cycle)
```



```
Ready to code?

Here is Claude's plan:

Plan for Building Plural Policy Colors Website

Based on my analysis of the repository and Plural's brand assets, here's my implementation plan:

1. Project Setup
- Create a modern React application with TypeScript
- Set up a responsive single-page website structure
- Configure build tools and development environment

2. Design Implementation
- Color Scheme: Use Plural's brand colors
  - Primary: Orange (#FF8422) and Teal (#00858C)
  - Supporting colors from the existing palette
- Typography: Inter font family (matching existing brand)
- Logo Integration: Use the Plural logo found in the assets

3. Website Structure
- Header: Navigation with Plural logo
- Hero Section: Eye-catching introduction with brand colors
- Color Showcase: Interactive display of Plural's color palette
- Policy Colors Section: Visual representation of policy-related colors
- Features Section: Highlight key aspects of Plural's work
- Footer: Contact information and links

4. Interactive Features
- Color palette explorer with hex codes
- Hover effects and animations
- Responsive design for all devices
- Smooth scrolling navigation

5. Technical Implementation
- React components for modularity
- CSS-in-JS or SCSS for styling (matching existing patterns)
- Performance optimization
- Accessibility considerations
```

Low Level Details

Planning

Claude includes a planning mode where it will explore the code base but won't write any code until the plan is approved. This step is especially important for complex tasks, where it can split the plan into milestones, and can use the reasoning keywords. It's useful to ask Claude to write the plan for future reference.

```
> Change the port to 8060

• I'll help you change the port to 8060. Let me first examine the current port configuration in the app.py file to understand how it's set up.

• Search(pattern: "PORT|8050", path: "compliance_demo/app.py", output_mode: "content")
  | Found 60 lines (ctrl+P to expand)

• Perfect! I can see that the port configuration is on line 1539 in the app.py file. Currently it's set to use port 8050 as the default when no PORT environment variable is set.

Ready to code?
Here is Claude's plan:

Plan to Change Port to 8060

I found that the port configuration is in /Users/santiagovelez/Documents/plural/agents_demos/compliance_demo/app.py on line 1539.

Current code:
port = int(os.environ.get("PORT", 8050))

Required change:
- Change the default port from 8050 to 8060 in the os.environ.get("PORT", 8050) call

Steps:
1. Update line 1539 in app.py to change the default port from 8050 to 8060
2. The app will then run on port 8060 by default (unless overridden by the PORT environment variable)

This is a single one-line change that will make the application start on port 8060 instead of 8050.

Would you like to proceed?
> 1. Yes, and auto-accept edits
2. Yes, and manually approve edits
3. No, keep planning
```



Claude Opus 4

Our most powerful and capable model

📄 Text and image input

📄 Text output

📄 200k context window

🏆 Superior reasoning capabilities



Claude Sonnet 4

High-performance model with exceptional reasoning capabilities

📄 Text and image input

📄 Text output

📄 200k context window

Models

Sonnet 3.7 was like an eager junior engineer, able to make simple changes but overcomplicating code, making unnecessary changes and simple errors.

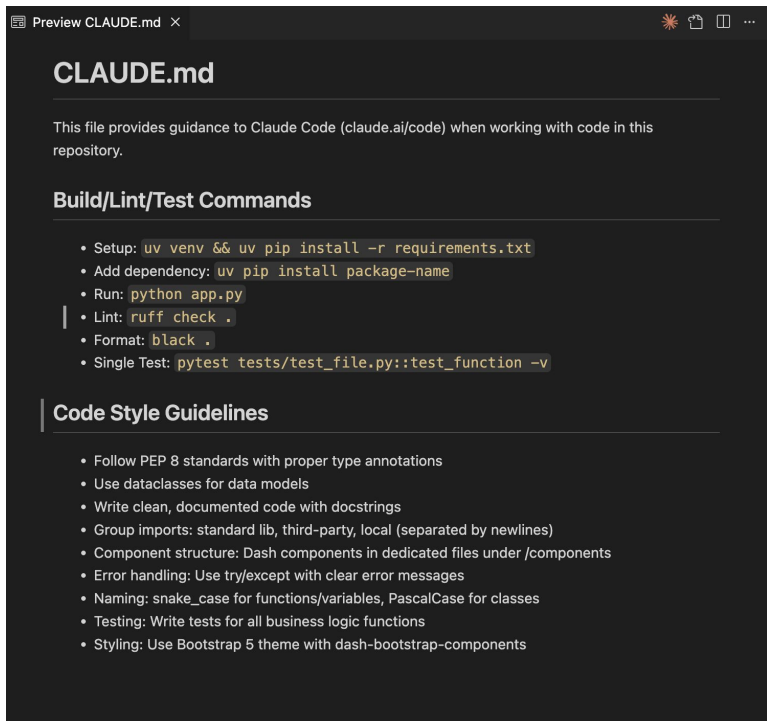
Sonnet 4 is pretty good, but the code is qualitatively similar to Sonnet 3.7.

Opus 4 is different, with better planning, smaller changes and sometimes sparks of interesting results.

Prompting

The task complexity should define the prompt length and level of detail. It's useful to include images and additional references. The more specific your instructions are the best. You can reference files with “@”, though not all file types can be used. For long prompts writing a markdown file can be useful.

Poor	Good
add tests for foo.py	write a new test case for foo.py, covering the edge case where the user is logged out. avoid mocks
why does ExecutionFactory have such a weird api?	look through ExecutionFactory's git history and summarize how its api came to be
add a calendar widget	look at how existing widgets are implemented on the home page to understand the patterns and specifically how code and interfaces are separated out. HotDogWidget.php is a good example to start with. then, follow the pattern to implement a new calendar widget that lets the user select a month and paginate forwards/backwards to pick a year. Build from scratch without libraries other than the ones already used in the rest of the codebase.



claude.md

Special file that is always included inside Claude context. Specially useful for instructions that Claude should always follow like useful commands and best practices.

Reasoning

Claude Code includes special keywords that relay to the model how much it should reason about the prompt. They are "think" < "think hard" < "think harder" < "ultrathink", each allocating more thinking budget than the last. This may improve performance but also increases times and costs.



Max 5x

\$100

Per person billed monthly

Claude Code is included in your Max plan.
Great value for everyday use in larger
codebases with access to both Claude Sonnet
4 & Claude Opus 4.

Sign up

Max 20x

\$200

Per person billed monthly

Even more Claude Code included in your Max
plan. Great value for power users with the
most access to Claude Opus 4.

Sign up

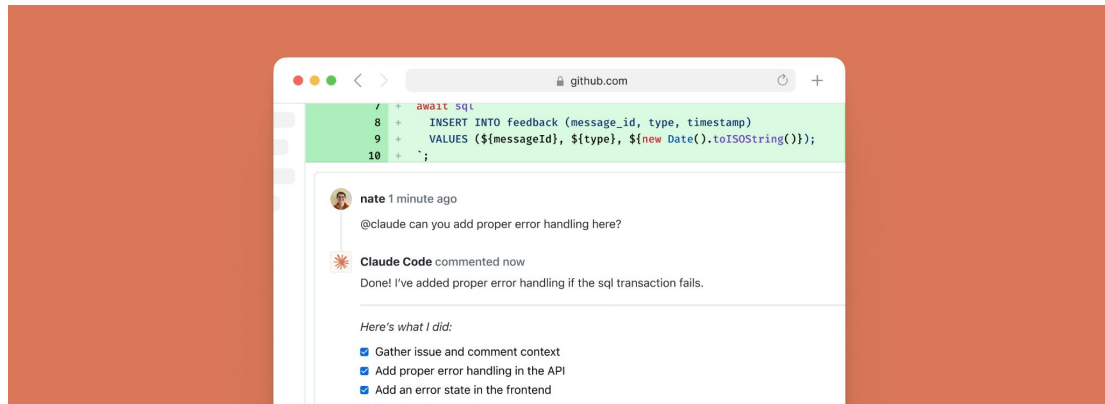
Pricing

Pay-per-use using the API is a great way to start, but you can easily spend thousands with medium use.

Claude Max \$100 or \$200 plans are a better idea for significant use, though you may run out of Opus from time to time.

Advanced use cases

- MCP Connection
- Multi-branch pattern
- Github integration
- Headless mode
- Etc.



Source: <https://www.anthropic.com/engineering/claude-code-best-practices>

Caveat: <https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/>