

Member-only story

On Codex CLI Hits GA: A Practical Setup & Safety Playbook for Engineering Teams (2025)

10 min read · Oct 7, 2025



Reza Rezvani

Following ▾



Listen



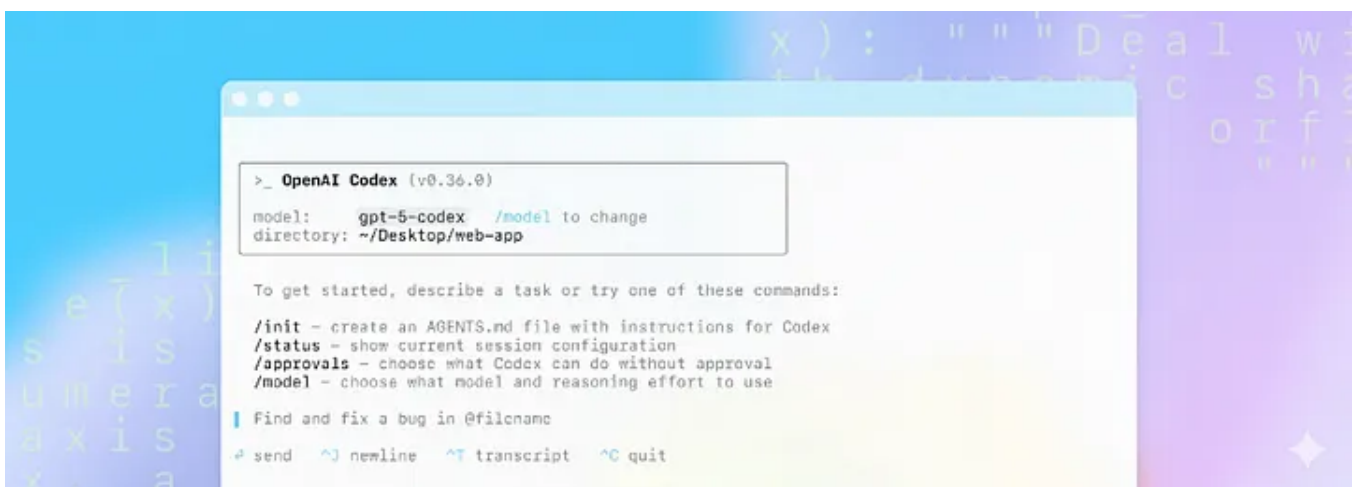
Share

... More

OpenAI's terminal-first coding agent terminal-first coding agent reaches production stability with game-changing features that set it apart from Claude Code and Gemini CLI.

This was my first Month with Codex CLI: A Developer's Honest Journey into Agentic Coding

What I learned about OpenAI's terminal coding agent — the mistakes I made, the surprises I found, and what actually matters for newcomers.



Codex CLI — Agentic Coding Tool

The Terminal Window That Changed My Saturday

It was a Saturday morning. I had coffee, a messy authentication refactor I'd been postponing for weeks, and a terminal window with something new blinking at me:

I'd heard about AI coding assistants. I'd used autocomplete tools. But this felt different. This thing could *read my entire codebase* and make changes across multiple files at once.

My first thought: "This is either brilliant or dangerous."

My second thought: "Let's find out."

That was four weeks ago. This is what I learned.

. . .

What Codex CLI Actually Is (Without the Hype)

Before I share my journey, let me explain what we're talking about — because I was confused at first too.

Codex CLI is a command-line tool that acts like a coding partner sitting in your terminal. You describe what you want, and it:

- Reads through your project files
- Understands your code structure
- Makes edits across multiple files
- Runs tests to check its work
- Explains what it changed and why

Think of it as having a junior developer who works *really* fast but needs your supervision. It's built by OpenAI, recently reached "*general availability*" (*which means it's production-ready*), and it's open-source.

The key word everyone keeps using is "*agentic*" — meaning it doesn't just suggest code, it takes action. That's powerful. That's also why I was nervous.

. . .

Week One: Setting Up and Breaking Things

The Installation (Easier Than Expected)

I expected setup pain. I got none.

```
brew install codex  
codex login
```

Two commands. Three minutes. Done.

It asked me to authenticate with my ChatGPT account (I have the Plus subscription). No API keys to manage, no complex configuration files. Just login and go.

First surprise: It felt suspiciously easy. I kept waiting for the “gotcha.” It never came.

My First Prompt (And First Mistake)

I decided to start small. I had a function with callback hell that needed converting to async/await.

I typed: “Fix the authentication code.”

Codex paused for a moment, then asked me: *“I found three authentication-related files. Which one do you want me to work on?”*

Lesson learned immediately: Be specific. Codex is powerful, but it’s not a mind reader.

I tried again: *“Convert the `authenticateUser` function in `src/auth/middleware.js` to use `async/await` instead of callbacks.”*

This time, it worked. In about 30 seconds, it showed me a diff of the changes. Clean code. Proper error handling. Even updated the function that called it.

I stared at the screen.

That would have taken me 20 minutes, easy. Probably longer if I got distracted.

The Mistake That Taught Me Everything

Emboldened by success, I got cocky.

“Refactor the entire authentication system to use JWT tokens.”

Codex started working. Files changed. New dependencies appeared. Tests were updated. It looked impressive.

Then I ran `npm test`.

Everything broke.

Here’s what I learned: Codex is fast and capable, but big changes need big context. I hadn’t explained our specific security requirements, how sessions worked, or which dependencies we were already using.

I hit `ctrl+c` and rolled back with git. Crisis averted.

The real lesson: Start small. Build trust. Scale gradually.

. . .

Week Two: Finding the Right Workflow

The Approval System (My New Best Friend)

After my refactor disaster, I discovered something I’d initially ignored: approval modes.

Codex has three settings:

- **Auto** — Works freely in your project, asks permission for risky stuff
- **Read Only** — Explores but doesn’t change anything
- **Full Access** — Does whatever it wants (scary)

I’d been using Auto mode, which is actually perfect for beginners. It lets Codex work on your code but pauses before doing anything that could access the network or touch files outside your project.

Why this matters: When Codex wanted to run `npm install` for a new dependency, it asked first. When it wanted to commit to git, it asked. This safety net let me learn without fear.

The Image Trick I Discovered by Accident

One afternoon, I was debugging a UI layout issue. I took a screenshot to send to a colleague and thought: “Wait, can Codex look at this?”

```
codex -i screenshot.png "The login button is cut off on mobile. Fix it."
```

It analyzed the image, found the CSS issue, and fixed it.

I felt like I’d discovered a secret level in a video game.

Practical use: Now when designers send mockups, I just attach them: `codex -i mockup.png "Implement this dashboard"`. It's not perfect, but it gets you 80% there.

. . .

Week Three: The Patterns That Actually Work

Pattern 1: Describe the Outcome, Not the Steps

Bad prompt: “Update the user controller, add error handling, then modify the tests, and make sure the error messages are consistent.”

Good prompt: “The user controller should gracefully handle database connection failures and return meaningful error messages.”

Codex figures out the steps. You define the goal.

Pattern 2: Work in Layers

Instead of “rewrite this entire module,” I learned to work in layers:

1. “Explain how this module works”
2. “Identify potential issues with error handling”
3. “Fix the error handling issues you found”
4. “Add tests for the new error cases”

Each step builds on the last. Each step I can review and approve.

Pattern 3: Use It for the Boring Stuff

Where Codex really shines:

- Converting old patterns to new ones
- Adding consistent error handling
- Updating tests after refactors
- Writing boilerplate code
- Documenting complex functions

Where I still do the work myself:

- Critical security logic
- Complex business rules
- Architecture decisions
- Performance-critical code

The rule I follow: Let Codex handle the mechanical work. I handle the thinking.

. . .

The Features I Actually Use (And The Ones I Ignore)

What I Use Daily

Session resumption — I can close my terminal and pick up exactly where I left off:

```
codex resume
```

This changed everything for me. No more “what was I doing?” moments.

Model switching — For simple tasks, I use the default. For complex refactors, I switch to the specialized coding model:

```
/model gpt-5-codex
```

It's slower but more accurate for architectural changes.

The `exec` command - When I know exactly what I want:

```
codex exec "Add TypeScript types to all function parameters"
```

No back-and-forth. Just do it.

What I Haven't Explored Yet

There's a Slack integration. There's a GitHub PR review bot. There's an SDK for building custom workflows.

Honestly? I'm still mastering the basics. Those feel like "advanced mode" features I'll need when I'm working with a team.

Current philosophy: Master the terminal experience first. Add complexity later.

. . .

The Honest Limitations Nobody Talks About

It Gets Confused in Large Codebases

Once your project has hundreds of files, Codex sometimes needs help finding things. I've learned to point it to specific directories:

"Look in `src/auth/` and update the middleware..."

It Doesn't Understand Your Business Logic

I asked it to "add a discount system to the checkout process."

It added *a* discount system. Not *our* discount system with its specific rules about loyalty points and promotional stacking.

Reality check: Codex writes code. You provide domain knowledge.

The Errors Can Be... Creative

Sometimes when things go wrong, the error messages are cryptic. Or Codex tries to fix an error in a way that creates three new errors.

When this happens, I've learned to:

1. Stop and read carefully
2. Use `/undo` to roll back
3. Break the task into smaller pieces
4. Try again with more context

It's Still Software

I've had sessions crash. I've had it generate code that didn't compile. I've had it misunderstand my intent completely.

That's okay. So do human developers (including me).

The difference: with Codex, I can try again immediately. With a human, I'd wait for them to be free.

. . .

What I Wish I'd Known on Day One

1. The `.codexignore` File Is Your Friend

Create this immediately:

```
.env
.env.*
/secrets/
node_modules/
dist/
build/
```

It tells Codex which files to never touch. Peace of mind.

2. Git Is Your Safety Net

Before every Codex session, I commit my working state:

```
git add .  
git commit -m "checkpoint before codex session"
```

If things go sideways, `git reset --hard` brings me back. No stress.

3. Start With Read-Only Mode

Your first few sessions, just chat:

```
codex  
/approvals readonly
```

Ask it to explain your code. Identify problems. Suggest improvements.

Once you trust its understanding, switch to Auto mode and let it work.

4. The Best Prompts Are Conversations

I used to try writing perfect prompts. Now I have conversations:

Me: “I need to add rate limiting”

Codex: “Where should I add it?”

Me: “To the API endpoints in the user controller”

Codex: “Should I create new middleware or update existing?”

Me: “Create new middleware”

It’s okay to iterate. That’s how you get exactly what you want.

. . .

The Unexpected Benefits

I’m Learning From It

Watching Codex refactor code teaches me patterns I hadn't considered. It's like pair programming with someone who has different approaches.

I've picked up:

- Better error handling patterns
- More consistent code structure
- Testing approaches I hadn't thought of

The Documentation Got Better

When Codex adds a complex function, it often includes clear comments explaining the logic. I've started writing better comments in my own code because of this.

I'm Less Afraid of Tedious Tasks

That 200-line file that needs updating for a new API version? Used to dread it. Now I just describe the changes and review the diff.

The tedious work is still tedious — but I'm supervising, not executing.

. . .

Month One: What Actually Changed

I'm not writing code faster (though sometimes I am). What changed is different.

I'm spending more time thinking. Less time typing boilerplate. More time on architecture.

I'm refactoring more often. Because it's not a huge time investment anymore. I see something that could be cleaner, I fix it.

I'm more willing to experiment. I can try a different approach, see how it feels, and roll back if it doesn't work — all in minutes.

But I'm still the developer. Codex is a tool. A powerful one. But I'm still making the decisions, reviewing the code, and understanding the system.

It hasn't replaced my skills. It's amplified them.

. . .

Should You Try It?

I can't answer that for you. But I can tell you who it's worked for in my first month:

Try it if:

- You're comfortable with the terminal
- You have repetitive coding tasks that drain your energy
- You're curious about AI tools but want something practical
- You're okay with supervising and learning a new workflow

Maybe wait if:

- You're brand new to programming (master the basics first)
- You work in a highly regulated environment (check your company's AI policies)
- You prefer having complete control over every line of code
- You're not interested in learning a new tool right now

For me? I'm still here. Still learning. Still making mistakes and discovering new uses.

It's become part of my workflow — not the center of it, but a reliable tool I reach for daily.

. . .

Your Turn: Start Simple, Stay Curious

If you decide to try Codex CLI, here's my advice from one beginner to another:

Day 1: Just install it and chat. Ask it to explain some of your code. See how it responds.

Day 2: Pick the smallest, lowest-risk task you can find. Let it make one change. Review carefully.

Day 3: If that went well, try something slightly bigger. If it didn't, figure out why and try again.

Week 2: Start developing your own patterns. What prompts work? What doesn't? What's your workflow?

Month 1: Reflect on what changed. Are you solving different problems? Spending your time differently?

And remember: it's okay if it doesn't click immediately. It's okay if it's not for you. It's okay if you use it differently than I do.

We're all figuring this out together.

. . .

What I'm Curious About Next

I haven't tried:

- The PR review automation
- The Slack integration for team collaboration
- Building custom workflows with the SDK
- Using it in a CI/CD pipeline

Each of these feels like its own learning journey. Maybe I'll write about them after another month of exploration.

. . .

Let's Learn Together

This is where I am after four weeks. Your mileage will vary. Your discoveries will be different.

I'm curious:

- What's your experience been if you've tried it?

- What hesitations do you have if you haven't?
- What would you want to know that I didn't cover?

Drop a comment. Share your story. Ask your questions.

The best part about exploring new tools isn't just learning them — it's learning *together* and sharing what we discover along the way.

If this helped you, clap it forward. If you're trying [Codex CLI](#), I'd love to hear how your first week goes.

And if you're interested in more honest explorations of AI tools for developers, follow along. I'm documenting this journey as it happens — mistakes, discoveries, and all.

Now it's your terminal window. What will you discover?

. . .

Or check out my new Master Guide for Spec-Driven Development:

👉 **Step 1:** Read this Master Guide — your evergreen hub for Spec-Driven Development resources.

👉 **Step 2:** [Read Part 1: The Foundation to set up your memory system, constitution, and specification workflow.](#)

👉 **Step 3:** [Continue with Part 2: Execution and Scaling to turn specs into plans, tasks, and tested features.](#)

My new agentic coding guide:

👉 [THE Claude Agent SDK BUILDER'S PLAYBOOK — Part 1](#)

Or read about my experiences using the prompt I have shared with you in this article: [I Gave Claude Code 2.0 Our 3-Week Refactor at 11 PM. At 7 AM, It Was Done](#)

About the Author

Alireza Rezvani is a Chief Technology Officer, Senior Full-stack Architect, and Software Engineer, as well as an AI Technology Specialist, with expertise in modern development frameworks, cloud-native applications, and agent-based AI systems.

With a focus on ReactJS, NextJS, Node.js, and cutting-edge AI technologies and concepts of AI engineering, Alireza helps engineering teams leverage tools like Gemini CLI, and Claude Code or Codex from OpenAI to transform their development workflows.

Connect with Alireza at alirezarezvani.com for more insights on AI-powered development, architectural patterns, and the future of software engineering.

Looking forward to connecting and seeing your contributions — check out my [open source projects on GitHub!](#)

✨ Thanks for reading! If you'd like more practical insights on AI and tech, hit **subscribe** to stay updated.

I'd also love to hear your thoughts — drop a comment with your ideas, questions, or even the kind of topics you'd enjoy seeing here next. Your input really helps shape the direction of this channel.

Openai Codex

Agentic Ai

Codex Cli

Software Development

Artificial Intelligence



Following ▾



Written by Reza Rezvani

1K followers · 76 following

As CTO of a Berlin AI MedTech startup, I tackle daily challenges in healthcare tech. With 2 decades in tech, I drive innovations in human motion analysis.

No responses yet





Bgerby

What are your thoughts?

More from Reza Rezvani



In nginity by Reza Rezvani

How Cursor and Claude Code Plugins Turned Me Into a 20x Developer – The Agentic Coding Setup That...

My Background Agent just submitted a PR that made our senior architect ask, “Who wrote this?”




Oct 14



73




 Reza Rezvani

“7 Steps” How to Stop Claude Code from Building the Wrong Thing (Part 1): The Foundation of...

Learn how to stop Claude Code from rewriting your architecture with vague prompts. This guide introduces Spec-Driven Development...

 Sep 17  44  2


 Reza Rezvani

Gemini CLI: What Happened When I Replaced My IDE With a Free AI Terminal Agent for 30 Days

I gave Google's new Gemini CLI full access to my development workflow and tested it on real production code. Here's what actually worked...

★ Oct 10 🖱 20



 In nginity by Reza Rezvani

I Let Claude Sonnet 4.5


IMAGINE this: It's 6 a.m., the kind of quiet dawn where the world's still wrapped in that soft, hazy light filtering through your blinds...

★ Sep 29 🖱 101 💬 1



See all from Reza Rezvani

Recommended from Medium

 ZIRU

Why Spec-Driven Development Made Me 5x More Productive

And How GitHub Spec Kit Changed Everything

★ Oct 16 🤝 36 💬 3




 Sevak Avakians

Spec Coding Workwheel

Modern software development requires choreographed Human-AI interaction. Here's how to do it.

6d ago 🤝 4



 bbang

ChatGPT Codex vs Claude Code: Strengths, Weaknesses, and How to Choose

Intro: Two Code-Centric AIs, Two Different Philosophies



Oct 1



52



1




Reza Rezvani

“7 Steps” How to Stop Claude Code from Building the Wrong Thing (Part 1): The Foundation of...

Learn how to stop Claude Code from rewriting your architecture with vague prompts. This guide introduces Spec-Driven Development...

★ Sep 17 🖱️ 44 💬 2



 In AI Software Engineer by Joe Njenga

Cursor 2.0 Has Arrived—And Agentic AI Coding Just Got Wild

Cursor has released version 2.0 , bringing the most powerful agentic AI we have seen yet, more autonomous than ever before,here's what's...

★ 5d ago 🖱️ 448 💬 8





Daniel Avila

Claude Code Learning Path: a practical guide to getting started

After spending months diving deep into Claude Code, I wanted to share the learning path that worked for me. This isn't from official...

4d ago



57



See more recommendations