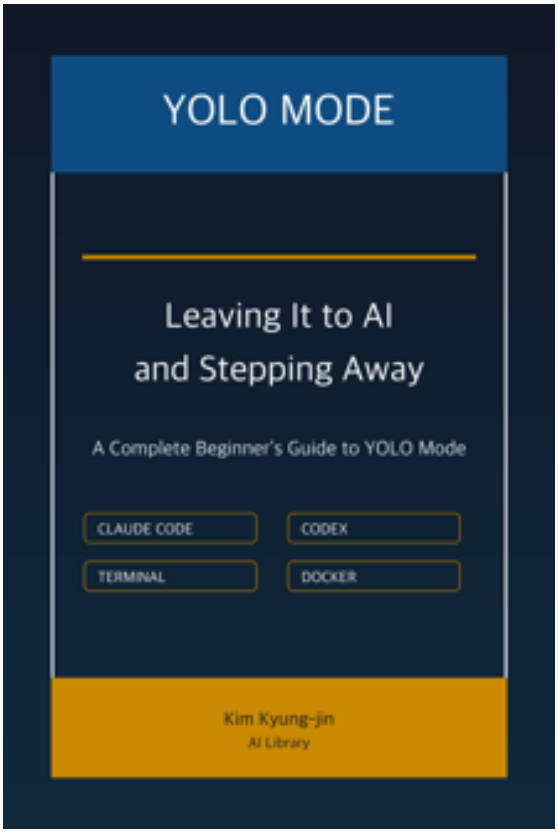
KIMKJ AI Library

Leaving It to AI and Stepping Away

A Complete Beginner’s Guide to YOLO Mode. Table of contents and 26 chapters



Kim Kyung-jin

English PDF edition

Leaving It to AI and Stepping Away

Kim Kyung-jin

A beginner-friendly online book on YOLO mode in Claude Code and Codex. It explains how to let AI read files, write code, run commands, and finish work while keeping rollback, Docker sandboxing, and safety checks close at hand.

Table of Contents

Chapter 1. At Two in the Morning, an Office Worker in Front of a Laptop

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

|

Next chapter: Chapter 2. What Does It Mean for AI to Work on Its Own?

The screen's glow lit up his face. Two in the morning, in a studio apartment in Mapo-gu, Seoul. Assistant Manager Kim was staring down a quarterly report due at nine o'clock the next morning. Three Excel files, twenty-two PowerPoint slides, five CSV files with sales data sorted by client. All of it had to come together as a single report.

That was already his third cup of coffee.

Assistant Manager Kim uses ChatGPT regularly. Meeting summaries, email drafts, quick translations. It works well enough for things like that. But this report was a different problem entirely. He needed to open files, read data, run calculations, build charts, and paste everything into slides. If you type "write me a report" into ChatGPT, you get something that looks plausible — but it can't actually open an Excel file and pull out numbers.

That night, he tried something different.

He opened a black screen called the Terminal and told the AI: "Read the three Excel files and five CSV files in this folder, create a quarterly sales summary table by client, and generate a PowerPoint report with charts included." Then he pressed Enter. The AI got to work. It opened the files, wrote code, built tables, drew charts. He watched it for a while, and at some point he closed the laptop and went to bed.

When he woke up at seven in the morning, there was a finished PowerPoint file sitting in the folder. Charts included, numbers correct. All he had done was tell it what to do.

That story is where this book begins.

You've probably heard the word "YOLO." You Only Live Once. Made famous in the early 2010s as a song title by rapper Drake, it became shorthand for a whole generation's approach to spending and living. Don't worry about tomorrow — live for today. Experience over savings. Risk over safety.

Starting around 2025, that word took on a completely different meaning among programmers. A term called "YOLO Mode" began appearing in AI tools. It means putting the AI to work and skipping every step where it asks "Is it okay if I do this?" — letting it run from start to finish on its own. No asking permission. Just go. Just as you only live once, the AI runs the whole thing in one shot.

This book is about that YOLO Mode.

What this book covers is straightforward: how to turn on, use, and turn off YOLO Mode in two AI tools called Claude Code and Codex. It's written so that anyone can follow along, even with zero coding experience. You don't need to know what a Terminal is. This book will show you.

It's worth being clear about what this book doesn't cover. It's not a programming textbook. It won't teach you Python or JavaScript syntax. It's not a theory book on AI either. It won't explain how Machine Learning works. This is a guide to using a tool. Just as you can drill a hole in a wall without understanding how a drill works, you can put AI to work without understanding how AI works. That's what this book will show you.

|

Next chapter: Chapter 2. What Does It Mean for AI to Work on Its Own?

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 2. What Does It Mean for AI to Work on Its Own?

Previous chapter: Chapter 1. At Two in the Morning, an Office Worker in Front of a Laptop

|

|

Next chapter: Chapter 3. Why Does the Command Say "Dangerously"?

Picture yourself ordering at a restaurant.

Most AI tools work like this. You say, "One kimchi stew, please," and the server asks, "One kimchi stew, is that right?" "Yes." "Shall I bring rice with that?" "Yes." "Would you like a refill on the side dishes?" "Yes, please." Step by step, confirmation by confirmation. Ask ChatGPT to "draft a report for me" and it replies, "What topic should the report cover?" You answer, and it asks, "About how long should it be?" Every move requires your sign-off. Safe, but slow. You can't step away.

YOLO mode is different. You say, "One kimchi stew — set up the rice and side dishes however you see fit, and pick something reasonable for dessert," then you sit down and read your book. The server figures it out. No questions about whether to serve the rice separately, which side dishes exactly, or whether you want cold water or warm. They use their judgment and get on with it.

Let me put that comparison in sharper terms.

AI coding tools like Copilot suggest the next line while you type. Think of it as autocomplete while you write. Your hands are still doing the work. The AI proposes; the person decides.

ChatGPT is conversational. You ask, it answers. You ask again, it answers again. The ball goes back and forth, like a game of ping-pong. It can't create files on your computer or run programs. It stays inside the chat window.

Claude Code and Codex in YOLO mode are fundamentally different. The AI creates files directly on your computer, writes code, and runs programs. It opens folders, reads their contents, creates new files, and edits existing ones — all without stopping to ask, "Is it okay if I do this?"

"AI that asks permission" versus "AI that runs without it." That gap is the whole point of YOLO mode.

Claude Code is an AI tool built by a company called Anthropic. It runs in the terminal. By default, it asks "Is it okay to do this?" every time it creates a file or runs a command. The safety guard is on. The command that turns that guard off is --dangerously-skip-permissions. The name says it plainly: skip the permission step, dangerously. Once you flip that switch, Claude Code gets to work without asking questions.

Codex is a similar tool from OpenAI. Here the same effect comes from a short command: --yolo. The name itself says "you only live once."

In March 2026, Claude Code added a new option called Auto Mode. It sits between full YOLO mode and the default. Commands that look risky — deleting files, changing system settings — still require permission, while safe commands like reading files or searching text run automatically. Think of it as a semi-automatic gear added between manual and fully automatic.

This book covers all three. But the focus is full YOLO mode: the state where AI judges for itself and acts on its own.

Previous chapter: Chapter 1. At Two in the Morning, an Office Worker in Front of a Laptop

|

|

Next chapter: Chapter 3. Why Does the Command Say "Dangerously"?

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 3. Why Does the Command Say "Dangerously"?

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 2. What Does It Mean for AI to Work on Its Own?

|

|

Next chapter: Chapter 4. What Is the Terminal?

Let's take another look at the command that turns on Claude Code's YOLO mode.

claude --dangerously-skip-permissions

The word "dangerously" jumps out. Dangerously. Why would a software company put the word "dangerously" inside one of its own commands?

The reason is straightforward: to warn the user.

Think about a kitchen knife. A knife is dangerous. It can cut your hand. But you can't cook without one. The fact that a knife is dangerous doesn't mean you stop using it. Instead, you learn how to use it properly. Keep the blade pointed away from your body. Cut only on a chopping board. Put it back in the sheath when you're done. Follow these rules and a knife becomes a tool. Ignore them and it becomes a weapon.

YOLO mode works the same way.

When AI moves without asking permission, it means the AI can freely create, edit, and delete files on your computer. Used well, your report is finished while you sleep. Used carelessly, an important file can disappear while you sleep. Anthropic put "dangerously" in the command so that you make the decision with full awareness of this fact. The command itself is designed to carry your declaration: "I know the risk, and I'm choosing to proceed anyway."

The original Codex command was even more explicit: --dangerously-bypass-approvals-and-sandbox. "Dangerously bypass approvals and the sandbox." It became too long, so the alias --yolo was created, but the weight of the warning packed into the original name is anything but light.

[Warning] YOLO mode gives the AI unrestricted access to your computer's file system. Before using it in any folder that contains important files, make sure you back everything up first.

When you buy a knife, a good knife shop gives you a sheath along with the blade. This book is that sheath.

What you practiced today

YOLO mode is a mode in which the AI carries out a task from start to finish on its own, without stopping to ask for permission along the way. In Claude Code it is turned on with --dangerously-skip-permissions; in Codex, with --yolo. Because this mode is powerful but carries real risk, learning how to turn it on and how to turn it off go hand in hand.

Previous chapter: Chapter 2. What Does It Mean for AI to Work on Its Own?

|

|

Next chapter: Chapter 4. What Is the Terminal?

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 4. What Is the Terminal?

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 3. Why Does the Command Say "Dangerously"?

|

|

Next chapter: Chapter 5. Claude Code and Codex: What Is Different?

In the winter of 2019, a self-employed acquaintance asked me to build an Excel macro, so I brought my laptop over. When I opened a black screen, he looked startled and asked, "Is that hacking?"

No. It's not hacking.

The Terminal is a screen where you give your computer instructions in text. When most people use a computer, they click icons with a mouse, open windows, and press buttons. That's called a GUI (Graphical User Interface) — communicating through visuals. The Terminal works differently. You type words, and the computer understands. It's called a CLI (Command Line Interface) — communicating through text.

Back to the restaurant analogy. A GUI is like pointing at a photo on the menu. "That one." The Terminal is like placing your order out loud. "One kimchi stew, extra rice, please." The result is the same — the kimchi stew arrives. The method is just different.

Think of it as a blank notepad. A white sheet with nothing written on it. You write what you want to say, and the computer reads it and acts on it.

Let me start by showing you how to open the Terminal on a Mac.

Press the Command key and the spacebar at the same time. A search bar appears in the middle of the screen. Type "Terminal" there. You can also type "Terminal" in English. When the Terminal app shows up in the list, press Enter. A black or white screen appears. You'll see a cursor blinking. That's the Terminal.

On Windows, it's a little different. There's a Windows logo key on the lower left of your keyboard. Press it to open the Start menu. Type "cmd" in the search bar. "Command Prompt" will appear. Click it and a black screen opens. If you're on Windows 11, you can also search for "Windows Terminal" — it opens a cleaner-looking screen.

If you've made it this far, you've cleared the first hurdle.

When you look at the Terminal screen, you'll see a bit of text already there. On a Mac, you'll see something like "username@computername ~ %". On Windows, you'll see something like

"C:\Users\username>". This is called the prompt. It means "enter your command here." The word "prompt" is the same as the prompts you type into an AI chatbot, but it's used in a different context here.

Let's try one thing as a test. Type the following command into the Terminal and press Enter.

echo "Hello"

(echo, "Hello")

The word "Hello" will appear on the screen. The computer repeated exactly what you told it to say. echo is a command that means "say this back to me." It's like shouting "Hello!" on a mountain and hearing the echo come back — the computer returns your words to you.

That's all there is to it. This is what the Terminal looks like, and this is how you use it. Nothing to be afraid of.

[Warning] Do not run commands in the Terminal unless you know what they do. Files can be deleted or system settings can change. Please only enter the commands shown in this book.

You may be wondering why the Terminal is even necessary. It's because Claude Code and Codex run inside the Terminal. These two AI tools aren't programs you open by clicking an icon — you launch them by typing commands in the Terminal. That's why we're learning the Terminal first. It's like learning how to get into the water before you start learning to swim.

Previous chapter: Chapter 3. Why Does the Command Say "Dangerously"?

|

|

Next chapter: Chapter 5. Claude Code and Codex: What Is Different?

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 5. Claude Code and Codex: What Is Different?

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 4. What Is the Terminal?

|

|

Next chapter: Chapter 6. Learn Four Safety Tools First

Let's compare the two tools.

Claude Code was made by Anthropic. Codex was made by OpenAI. Both are AI tools that run in the terminal, and both create files and execute code directly on your computer. Their capabilities are similar, but the companies behind them differ, the commands differ, and the names of their safety features differ.

First, let's go over installation. Both tools require a program called Node.js to be installed. Node.js is what runs the JavaScript programming language, but all you need to know is the installation process. You don't need to understand how it works under the hood.

Here's how to install Node.js. On a Mac, open the terminal and enter the following command.

brew install node

(brew install node)

brew is a Mac program installer called Homebrew. It works similarly to the App Store, but operates in the terminal. If you don't have Homebrew yet, you'll need to install it first. Just follow the instructions on the Homebrew website (brew.sh).

On Windows, go to nodejs.org, download the installer, and double-click it to install. "Next, Next, Finish." It's the same as installing any ordinary program.

Once Node.js is installed, install Claude Code. Type this into the terminal.

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

(npm install -g @anthropic-ai/claude-code)

npm is the package installer that comes bundled with Node.js. install means "install this," and -g means "make it available from anywhere on the computer." That single command is all it takes to install Claude Code.

Installing Codex is similar.

npm install -g @openai/codex

(npm install -g @openai/codex)

Now let's lay out the differences between the two tools.

The commands are different. To launch Claude Code, type claude in the terminal. To launch Codex, type codex.

The safety feature names are different. Claude Code's YOLO mode flag is --dangerously-skip-permissions. Codex's YOLO mode flag is --yolo. The function is identical. Only the names differ.

The default models are different. Claude Code uses Claude models. As of May 2026, the default is Claude Opus 4.7. Codex uses GPT-series models. The default is GPT-5.5.

The pricing structures are similar for both. Claude Code can be used within the usage included in an Anthropic subscription (Claude Pro, Max, or Team), and it can also be used on a pay-as-you-go API basis without a subscription. Codex works the same way: it has usage included in an OpenAI subscription (ChatGPT Pro) and is also available via pay-as-you-go API. Either way, if you already have a subscription, you can get started without any extra cost.

Which one you use depends on your situation. If you're already paying for ChatGPT, Codex is the natural fit. If you're already paying for Claude, Claude Code is the natural fit. If you're using neither, both offer free trials, so just pick one and start. The two tools work in much the same way, so once you learn one, switching to the other isn't difficult.

One more thing worth mentioning: both tools exist in forms other than the terminal version. Claude Code can be used as an extension for the code editor VS Code, and similar functionality is available in the Claude Desktop app. Codex has a version that runs inside the ChatGPT website. That said, to use YOLO mode — the fully automated mode where AI manipulates files directly on your computer — you need the terminal version. That's why this book focuses on the terminal version.

Previous chapter: Chapter 4. What Is the Terminal?

|

|

Next chapter: Chapter 6. Learn Four Safety Tools First

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 6. Learn Four Safety Tools First

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 5. Claude Code and Codex: What Is Different?

|

|

Next chapter: Chapter 7. Turning On YOLO Mode in Claude Code

Before turning on YOLO mode, let's get the safety gear on first. Think of it as putting on a helmet before skiing. There are four tools.

Git, the Rewind Button

Git is a tool that records the history of changes made to your files.

Think of it this way. You've probably been writing a Word document and hit 'Save As' to create 'Report_v1', 'Report_v2', 'Report_Final', 'Report_ActualFinal'. Git does all of that automatically. Every time you change a file, you say 'remember this state,' and Git remembers. Later, when you say 'take me back to how it was,' it does.

Why does this matter? Because when AI edits your files and you don't like the result, you can roll back to the state before the edits. It's the seatbelt for YOLO mode.

You only need to remember three commands.

git init

(git init)

This command means 'start tracking changes in this folder.' Run it once inside the folder you're working in.

git add .

(git add .)

This means 'collect all the changed files.' The dot (.) refers to 'every file in this folder.'

git commit -m "saved state before work"

(git commit -m "saved state before work")

This means 'remember the current state.' Write a note inside the quotes. Later, that note tells you at a glance: 'ah, this is what I saved back then.'

Run these three steps every time before you turn on YOLO mode. It takes three seconds. But those three seconds protect your files.

Docker, the Wooden Fence Around the Sandbox

Docker is a tool that creates a small computer inside your computer.

You've probably seen a children's playground with a sandbox ringed by a wooden fence. No matter how much a child digs up the sand, the grass outside the fence stays untouched. Docker is that fence.

When you let AI work in YOLO mode inside Docker, whatever AI does has no effect on your files outside Docker. Even if AI accidentally deletes something, only the files inside Docker are gone. Your real files stay safe.

Docker Desktop is available for download at docker.com. There are versions for both Mac and Windows. Installation works like any regular program: download, double-click, 'Next, Next, Finish.' Once installed, just open the app once to have it running.

How to use YOLO mode inside Docker is covered in detail in Chapters 7 and 8. For now, just keep one thing in mind: 'there is a fence called Docker.'

Auto Mode, the Middle Ground

For those who find full YOLO mode a bit daunting, there is a middle ground: Claude Code's Auto Mode.

Picture a traffic light. In the default mode, every intersection has a red light. You stop, check, then go. Safe, but slow. YOLO mode turns off all the lights. You drive straight through. Fast, but accidents can happen. Auto Mode only turns off the lights at small intersections. At big ones (deleting files, running system commands) the red light still comes on. At small ones (reading files, searching) it passes through automatically.

If you're trying YOLO mode for the first time, starting with Auto Mode is the better call. Use it for a few days until you get a feel for it, then move up to full YOLO mode.

Writing Clear Instructions

The fourth safety tool isn't a technology. It's a habit.

Don't say "just figure it out."

Think about why that's risky. If you tell an AI "organize this folder," it will organize by its own standards. It might rename files, or delete old ones. The "organize" you had in mind and the "organize" the AI understood may be two very different things.

Say this instead: "Read the three Excel files in this folder, calculate the total sales figures, and save the result as 'monthly_sales_summary.xlsx'. Don't touch the existing files." That's specific. It's clear about what the AI should do and what it should leave alone.

A good instruction has three elements.

State what to do. "Read the three Excel files and calculate the total sales."

State what the output should look like. "Save it as 'monthly_sales_summary.xlsx'."

State what not to do. "Don't touch the existing files."

The habit of writing instructions with all three of these elements will protect you better than any software safety feature.

What You Practiced Today

Git is a time-travel tool that saves and restores file states, and it runs on three commands: git init, git add, and git commit. Docker creates a safe, contained environment for the AI to work in, and auto mode is a middle step you can take before going full yolo mode. Above all, the habit of writing specific instructions instead of saying "just figure it out" is the most reliable safety net you have.

Previous chapter: Chapter 5. Claude Code and Codex: What Is Different?

|

|

Next chapter: Chapter 7. Turning On YOLO Mode in Claude Code

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 7. Turning On YOLO Mode in Claude Code

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 6. Learn Four Safety Tools First

|

|

Next chapter: Chapter 8. Turning On YOLO Mode in Codex

All right, it's finally time to turn on YOLO mode.

But before we do that, let's check what you need. Go through each item as if you're running down a checklist.

Confirm that Node.js is installed. Type this into your terminal.

node --version

(node --version)

You should see a number like "v20.11.0". If you get an error message instead of a number, go back to Chapter 5 and follow the installation steps again.

Confirm that Claude Code is installed.

claude --version

(claude --version)

A version number should appear.

You need permission to use Claude Code. There are two ways to get it. If you have a Claude subscription (Claude Pro, Max, or Team), simply logging into your account is enough. The first time you run Claude Code from the terminal, a browser window opens and shows a login screen. Log in with your usual Claude account and you're done. If you don't have a subscription or prefer pay-as-you-go API billing, go to the Anthropic website (console.anthropic.com) and get an API key. After signing up, go to the "API Keys" menu and click "Create Key". A long string starting with "sk-ant-" will appear. That string is your key. Don't lose it, and don't show it to anyone.

[Warning] An API key is the same as a password. Never share it with anyone. Don't post it on a blog or social media. If it leaks, go to the Anthropic website immediately, delete the key, and issue a new one.

Choose a folder to work in. Create a new empty folder with no important files inside. This is for practice.

mkdir practice-folder

(mkdir practice-folder)

cd practice-folder

(cd practice-folder)

mkdir is short for "make directory" — it creates a new folder. cd is short for "change directory" — it moves you into that folder.

Save the current state with Git.

git init

git add .

git commit -m "State before YOLO mode test"

Setup is complete. Now let's turn it on.

claude --dangerously-skip-permissions

(claude --dangerously-skip-permissions)

When you press Enter, a message appears on screen. Claude Code is now running. The cursor blinks, waiting for your instructions. From this point on, anything you type will be executed immediately, without Claude Code stopping to ask for your approval.

Let's try it. Type the following.

"Create a file called hello.txt in this folder. The content should be 'Hello, this is YOLO mode.'"

The process of Claude Code creating the file will appear on screen. No question like "Is it okay to create this file?" — it just does it. Once the completion message appears, open a new terminal window and check for yourself.

cat hello.txt

(cat hello.txt)

If you see "Hello, YOLO mode." printed on screen, it worked. You just turned on YOLO mode for the first time and had the AI create a file.

If you use VS Code, you can do the same thing right inside it. Open VS Code, open the terminal pane (from the menu: Terminal > New Terminal), then type the same command. Installing Claude Code as a VS Code extension makes things even smoother, but the underlying idea is identical.

Typing --dangerously-skip-permissions every time gets old fast. It's long. A shell alias lets you shrink it down to something short.

On a Mac, type the following command into the terminal.

echo 'alias cyolo="claude --dangerously-skip-permissions"' >> ~/.zshrc

(echo alias cyolo equals claude dangerously-skip-permissions >> tilde slash dot zshrc)

Then close the terminal and reopen it. From now on, typing cyolo is all it takes to turn on YOLO mode. It means exactly the same thing as the original command — the name is just shorter. Think of it like a nickname: it points to the same person.

On Windows it works a bit differently. If you're using PowerShell instead of Command Prompt, do this.

Set-Alias cyolo "claude --dangerously-skip-permissions"

In PowerShell, though, this alias disappears when you close the terminal. Making it permanent requires editing a profile file, which we'll skip for now. Using the full command each time works just fine.

Previous chapter: Chapter 6. Learn Four Safety Tools First

|

|

Next chapter: Chapter 8. Turning On YOLO Mode in Codex

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 8. Turning On YOLO Mode in Codex

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 7. Turning On YOLO Mode in Claude Code

|

|

Next chapter: Chapter 9. How to Stop It and How to Roll Back

Codex's YOLO mode works on the same principle. The command is just different.

The requirements are almost identical to Claude Code. Node must be installed, and Codex must be installed. The way to check is the same.

codex --version

(codex --version)

Codex, like Claude Code, gives you two options. If you have a paid ChatGPT subscription, you can start using it right away by logging in with your ChatGPT account. When you run Codex in the terminal for the first time, a login screen appears — just sign in with the ChatGPT account you already use. If you'd rather pay per use through the API without a subscription, go to platform.openai.com and generate an API key. The process is similar to Anthropic's.

[Warning] Your OpenAI API key is like a password. If it leaks, delete it immediately and generate a new one.

Navigate to your working folder and save the current state with Git. This is the same process you followed in Chapter 7.

Now turn it on.

codex --yolo

(codex --yolo)

Short. Two words. The full name behind this nickname-like command is this.

codex --dangerously-bypass-approvals-and-sandbox

(codex --dangerously-bypass-approvals-and-sandbox)

"Dangerously bypass approvals and sandbox." The sandbox is a concept similar to the Docker fence we covered earlier. A box of sand. It's a mechanism that confines the AI to a safe space. When you use --yolo, even that fence comes down.

In Codex, you can also set YOLO mode in advance through a config file. Instead of appending --yolo to the command every time, you write it as the default value in the config file.

The config file is called config.toml. TOML stands for Tom's Obvious, Minimal Language — a file format designed to be easy for people to read. Add these two lines to that file.

approval_policy = "never"

sandbox_mode = "danger-full-access"

approval_policy means "approval policy." The value "never" means "never ask." sandbox_mode is the fence mode, and "danger-full-access" means "dangerous, full access allowed." With this config file in place, typing just codex is enough to run it in YOLO mode.

[Warning] If you put these settings in config.toml, every time you run Codex it will automatically enter YOLO mode. When you're done practicing, delete these two lines or revert the values. Changing approval_policy to "on-failure" and sandbox_mode to "docker" turns the default safety measures back on.

Let's try it. With Codex YOLO mode on, enter the following.

"Create a Python file called greeting.py in this folder, and make it print 'Nice to meet you, this is Codex YOLO mode.' when run. After creating the file, go ahead and run it immediately."

Codex creates the Python file and runs it. If "Nice to meet you, this is Codex YOLO mode." appears on screen, it worked. You'll notice there was no step where it asked for permission. File creation, program execution — the AI handled everything on its own.

Whether you use Claude Code or Codex, the core idea is the same. You give the AI instructions, the AI carries them out, and you check the results.

Previous chapter: Chapter 7. Turning On YOLO Mode in Claude Code

|

|

Next chapter: Chapter 9. How to Stop It and How to Roll Back

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 9. How to Stop It and How to Roll Back

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 8. Turning On YOLO Mode in Codex

|

|

Next chapter: Chapter 10. First Practice: Make a Personal Webpage

Let's start with one principle.

If you don't know how to turn it off, don't turn it on.

That may sound extreme, but it's meant seriously. Nobody uses a microwave without knowing how to stop it. Nobody starts a car without knowing how to turn it off. Same idea here. Before you switch on YOLO mode, you must know how to switch it off.

Stopping it is straightforward.

Press the Ctrl key and the C key at the same time.

(Control-C)

That's all there is to it. Whether you're running Claude Code or Codex, most programs running in the terminal can be stopped with Ctrl+C. Even if the AI is in the middle of creating files, Ctrl+C will halt it. That's exactly why this key combination comes first.

On a Mac, it's Ctrl+C, not Command+C. Command+C is copy. Ctrl+C is stop. Easy to mix up, so keep that in mind.

Now let's talk about how to roll back.

When you're not happy with what the AI produced, here's how to restore everything to the state it was in before the work began. Git, which you learned about in Chapter 6, earns its keep here.

First, check what the AI changed. Type this in the terminal.

git diff

(git diff)

"diff" is short for "difference." It shows you what changed between before and after the AI worked. Lines added appear in green; lines removed appear in red. You get a clear picture of which files were changed and how.

You look it over and don't like what you see. You want to put everything back the way it was. Type this.

git checkout -- .

(git checkout -- .)

This command means: "Restore all files to the last saved state — the last commit." Because you ran git commit before switching on YOLO mode back in Chapter 7, it takes you back to that exact point.

[Warning] git checkout -- . cannot be undone. Once you run it, everything the AI produced is gone. If there are parts of the AI's work you want to keep, copy those files to a different folder before running this command.

New files the AI created will not be removed by git checkout. It only reverts changes to files that already existed. To see what new files the AI added, run this.

git status

(git status)

Anything listed under "Untracked files" is a new file the AI created. If you want to delete them, check each one individually and remove them by hand. There are commands that wipe them all at once, but this book recommends manual deletion for safety.

To summarize the rollback process:

Stop the AI with Ctrl+C. Review the changes with git diff. If you want to roll back, run git checkout -- . Check for new files with git status and delete any you don't need.

Remember these four steps. They are your emergency exit. Just as you locate the fire exit when you enter a building, confirm these four steps before you start working in YOLO mode.

One last thing. If the AI modifies files and you haven't made a commit, there is no way to get back to where you were. That's why Chapter 7 said: always run git commit before switching on YOLO mode. This one habit protects everything. It takes three seconds. "git add ." Enter. "git commit -m 'before work'" Enter. Skip those three seconds now and you may spend three hours regretting it later.

What You Practiced Today

Claude Code's yolo mode can be turned on with claude --dangerously-skip-permissions, and Codex's yolo mode with codex --yolo or by setting config.toml. Both tools can be stopped with Ctrl+C. You can review changes with git diff, then roll them back with git checkout -- . Make it a habit to run git commit before enabling yolo mode so you always have a saved state to return to.

Previous chapter: Chapter 8. Turning On YOLO Mode in Codex

|

|

Next chapter: Chapter 10. First Practice: Make a Personal Webpage

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 10. First Practice: Make a Personal Webpage

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 9. How to Stop It and How to Roll Back

|

|

Next chapter: Chapter 11. Second Practice: Combine Excel Files

Saturday afternoon. You're sitting on the living room couch with your laptop open and a cup of coffee beside you. The terminal cursor blinks. You've made it through Part 3, picked up the basics of git, and practiced turning YOLO mode on and off. Now it's time to actually build something.

Start small. A personal introduction webpage. One page with your name, hobbies, and favorite food.

Getting Ready

Open the terminal and create a folder for this exercise.

mkdir ~/yolo-practice

cd ~/yolo-practice

git init

Three lines is all it takes. Create the folder, move into it, and turn on git. That way, if anything goes wrong, you can roll it back.

Commands

Now turn on YOLO mode.

claude --dangerously-skip-permissions

Claude Code greets you on the dark screen. Type the following.

"My name is Kim Minsu, and my hobbies are hiking and photography. My favorite food is doenjang-jjigae. Build me a personal introduction webpage using this information. Name the file index.html."

Press Enter. Then wait.

What the AI Does

Claude Code starts moving. Text scrolls up the terminal screen. Let's follow along slowly and see what the AI is doing.

The AI begins by creating a file called index.html. It's written in HTML (the language used to build webpages), but you don't need to read the code. The AI handles all of that.

Inside the file, your name appears in large text at the top, with your hobbies and favorite food listed below. The background color comes out clean, and the font is easy on the eyes. In default mode, Claude Code would have asked, "Is it okay to create the file?" but in YOLO mode it just goes ahead and does it.

About 20 seconds later, a message appears saying it's done. To view the file in a browser, just tell Claude Code, "Open the page you just made in a browser." If you'd rather do it yourself, open the working folder in Finder and double-click index.html.

Chrome or Safari opens and the webpage appears. "Kim Minsu" is displayed in large text at the top, with hiking, photography, and doenjang-jjigae listed below. Clean and simple.

Time and Cost

From the moment you type the first command to seeing the result in your browser, the whole thing takes under three minutes. The cost, based on the Claude API (the channel programs use to communicate with each other), runs somewhere between about 15 and 30 cents. That's less than a tenth of what a cup of coffee costs.

What Could Go Wrong

Very little can go wrong in this exercise. All the AI did was create one file. It didn't touch anything that already existed, so there's no impact on your computer. If you don't like the result, just delete the file and you're done. You can right-click it in Finder (Mac) or File Explorer (Windows) to delete it, or tell Claude Code, "Delete the file you just made." And since you saved everything with git, you can also say, "Revert everything back to how it was."

Want to Go a Bit Further?

If the webpage feels plain, try saying this to Claude Code.

"I want to add a photo. Create a placeholder for a profile picture."

"Change the color scheme to shades of blue."

"Add a contact button."

It's better to give instructions one at a time. If you throw ten things at it all at once, the AI can get confused.

Previous chapter: Chapter 9. How to Stop It and How to Roll Back

|

|

Next chapter: Chapter 11. Second Practice: Combine Excel Files

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 11. Second Practice: Combine Excel Files

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 10. First Practice: Make a Personal Webpage

|

|

Next chapter: Chapter 12. Third Practice: Get a News Summary Every Morning

Monday morning, 9 a.m. Park, the sales team manager, sat down at his desk and let out a sigh. The sales report was due at 2 p.m., and the data was split across three Excel files: Q1_Sales.xlsx, Q2_Sales.xlsx, and Q3_Sales.xlsx. He had to merge them into one.

In the old days, he would have opened each file, copied the data, pasted it, checked that the row numbers lined up, and scanned the formatting to make sure nothing had broken. It was a good thirty-minute job.

Setup

Go to the folder that holds the sales files.

cd ~/Documents/sales-data

git init

git add .

git commit -m "backup originals"

Four lines. You just saved the originals with Git first. This needs to become a habit. In YOLO mode, AI can overwrite your source files. If you have a Git snapshot, you can always roll back, even if something does get overwritten.

The Command

claude --dangerously-skip-permissions

Once Claude Code is running, give it this instruction.

"Merge the three xlsx files in this folder into a single Excel file. Keep each file's sheet name as-is, name the merged file sales_combined.xlsx, and don't touch the originals."

That last sentence matters. "Don't touch the originals." Without it, AI may overwrite the source files.

What AI Does

Claude Code starts by writing a Python script. The script uses a library called openpyxl, a tool for working with Excel files. If that library isn't on your machine, AI installs it on its own. In normal mode it would ask, "Is it okay to install this library?" In YOLO mode, it just goes ahead.

The script is written, run, and sales_combined.xlsx appears. The whole thing takes about 40 seconds.

Checking the Result

Open sales_combined.xlsx in Excel or Google Sheets. You should see three sheet tabs, each holding the Q1, Q2, and Q3 data respectively. Compare the numbers against the originals to make sure everything is there.

You must check here. AI occasionally drops a row or shuffles the order. Compare the total in the last row of each original file against the totals in the merged file. If the numbers match, you're good.

Time and Cost

From typing the command to confirming the result: just over a minute. Cost: around 300 Korean won. The same task done by hand would have taken Park thirty minutes.

The Risk

There's one risk: overwriting the originals. AI might save the merged result on top of your source files. That's exactly why you commit to Git first. What if you forgot to do that?

git checkout -- .

This single line brings every file back to its state at the last commit. If you didn't commit first, there's no way to recover, so always save before you start.

If you want one more layer of protection, copying the original files to a separate location is also an option.

cp *.xlsx ~/Documents/sales-data_backup/

It's like wearing both a belt and suspenders at the same time, but the safer option is always the better one.

Previous chapter: Chapter 10. First Practice: Make a Personal Webpage

|

|

Next chapter: Chapter 12. Third Practice: Get a News Summary Every Morning

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 12. Third Practice: Get a News Summary Every Morning

Previous chapter: Chapter 11. Second Practice: Combine Excel Files

|

|

Next chapter: Chapter 13. Three Lessons from the Practice Sessions

Kim Kyung-jin runs a small Korean snack restaurant in Jongno-gu and skims the news on Naver every morning. Food service trends, ingredient prices, changes in health regulations. There's a lot to keep track of, but not enough time to actually read through it all. She has to clock in at 7 a.m. and start on the dough right away.

What she wants is simple. Every morning at 8, five news items related to the food service industry, automatically pulled together and saved as a text file on her desktop. She can skim it before opening the shop, note anything worth remembering, and move on.

Setup

This practice takes a bit more configuration than the previous two. It runs about 10 minutes, but once it's in place, it runs on its own every day.

First, create the working folder.

mkdir ~/news-summary

cd ~/news-summary

git init

Next, you'll use Docker, a tool that creates an isolated workspace. The reason is that in this practice, the AI accesses the internet. It needs to reach the web to search for news. Letting the AI roam the internet freely in YOLO mode can be risky, so we run it inside Docker as a contained boundary.

You should have Docker installed from Chapter 6, so tell Claude Code: "Run the news summary task inside a Docker container. Mount the ~/news-summary folder as the working directory so the output files are saved on my computer too." Claude Code will write and run the Docker command on its own. You don't need to memorize anything like docker run.

Command

Once Claude Code starts up inside Docker, give it this instruction.

"Search for five recent news articles related to the food service industry and create a file called news_[today's date].txt with each article's headline and a three-line summary. Save the file to the /work folder."

Since this is the first run, go ahead and try it once. The AI uses a web search tool to find news, summarize it, and produce a text file. It takes about two minutes.

Check the result. A file like news_20260513.txt should appear in the ~/news-summary folder. Open it and you'll find five news headlines, each followed by a three-line summary.

Setting Up Automation

One run went fine. Now you want this to run automatically every morning at 8.

On a Mac, you use launchd, the built-in task scheduler. You can tell Claude Code:

"Create a shell script and a launchd plist file that automatically runs the task we just did every morning at 8."

The AI produces two files. The shell script contains the commands to start Docker and run Claude Code. The plist file holds the schedule information telling the system to run that script every day at 8.

Register the plist file with Mac's scheduling system and you're done.

launchctl load ~/Library/LaunchAgents/com.news.summary.plist

That single line registers it. If your computer is on tomorrow morning at 8, you'll find a news summary file for today's date sitting on your desktop.

Time and Cost

Ten minutes to set up the first time. After that, it runs automatically for two minutes each day. Each run costs roughly 500 won, which comes to about 15,000 won a month. About the same as subscribing to a news clipping service.

Risk Factors

The risk in this practice is network access, since the AI is going online. Running it inside Docker limits what the AI can reach to the inside of that container. It can't touch other files or programs on your computer.

What if you ran this without Docker? As the AI browses the web, it could download unexpected scripts or create files in other folders on your computer. Docker acts as the fence that keeps all of that contained.

One more thing: because this script runs automatically every day, costs will add up. Check the API usage dashboard at least once a month. If news searches run long, the bill can climb higher than

you'd expect.

Previous chapter: Chapter 11. Second Practice: Combine Excel Files

|

|

Next chapter: Chapter 13. Three Lessons from the Practice Sessions

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 13. Three Lessons from the Practice Sessions

Previous chapter: Chapter 12. Third Practice: Get a News Summary Every Morning

|

|

Next chapter: Chapter 14. Writing and Document Organization

We ran three practice sessions: building a webpage, merging Excel files, and pulling news automatically. Three very different tasks, but the reason each one worked was the same.

Start small. In the webpage exercise, we only put in name, hobby, and favorite food. We didn't open with "add photos, add animations, and make it responsive." Start small, confirm it works, then add one thing at a time.

There's a reason this matters. When you ask AI to do many things at once, the chance of mistakes goes up. If you hand it ten tasks and it gets eight right but two wrong, finding the errors is hard. Give it one thing at a time, and the mistake shows up immediately.

Save with Git before you start. All three sessions began with git init and git commit. Think of it as a seatbelt. Most drives go fine without one. But when there's a crash? Running YOLO mode without Git is like getting on a highway with no seatbelt.

We used Git even in the Chapter 10 webpage exercise, where almost nothing could go wrong. Because it's a habit. Don't evaluate each time whether it's necessary — just do it. git init, git add, git commit. Six seconds.

Check the result with your own eyes. When AI says "Done," don't take its word for it. We opened the webpage in a browser, opened the Excel file and compared the numbers, opened the news summary file and read it.

AI sometimes doesn't know whether what it produced is right or wrong. I've seen it say "success" while the actual file was empty. Checking takes thirty seconds. Skip those thirty seconds and you might lose thirty minutes later.

These three habits — start small, save with Git before you begin, check the result with your own eyes — apply to every case that appears in Part 5. Nothing to memorize. Three practice runs were enough to make them feel natural.

[End of Part 4] What you got comfortable with today

Built a personal introduction webpage in YOLO mode. Three minutes, about 30 cents.

Merged three Excel files into one. One minute, about 30 cents. Picked up the habit of saving the original with Git first.

Set up automatic morning news summaries. Ten minutes to configure, then fully automatic every day after that. Run it inside Docker to keep things safe.

Three principles: start small. save with Git before you begin. check the result with your own eyes.

Previous chapter: Chapter 12. Third Practice: Get a News Summary Every Morning

|

|

Next chapter: Chapter 14. Writing and Document Organization

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 14. Writing and Document Organization

Previous chapter: Chapter 13. Three Lessons from the Practice Sessions

|

|

Next chapter: Chapter 15. Working with Tables and Numbers

Case 1) Drafting 50 Email Replies

Scene

Wednesday evening, 7 p.m. Jung Yu-jin, an associate at a marketing agency, opens her laptop. Fifty-three unread emails in her inbox: greetings from clients, quote requests, meeting scheduling, event invitations. Reading and replying to each one would take two hours. She still has to prepare the morning meeting materials for tomorrow.

Command

Jung Yu-jin exports all the emails in her inbox to a text file. She selects each email in Gmail, copies the content, and pastes it into a file called emails_received.txt, with a divider between each message.

She turns on Claude's YOLO mode and gives the following instruction.

"Read the emails_received.txt file and draft a reply to each email. Keep the tone polite but concise. For anything quote-related, end with 'I will review and get back to you by tomorrow.' For scheduling, reply with 'Tuesday at 2 p.m. works for me.' Save the results to replies_draft.txt."

What happens

Claude Code reads emails_received.txt and processes all 53 messages. It categorizes each email by type (greeting, quote, scheduling, invitation, other) and writes a reply suited to each. About three minutes later, the replies_draft.txt file is ready.

Result

A text file containing 53 reply drafts. Each reply is headed with the original email's subject line and sender, so it's immediately clear which email it responds to.

Cost

Three minutes of your time, roughly 500 won.

Risk

The AI may slip incorrect details into a reply. You told it to say 'Tuesday at 2 p.m.' and it might write 'Wednesday at 10 a.m.' These are drafts, so you must skim every single one before sending. Any email that mentions a specific amount deserves a close look at the numbers.

Trying this yourself

All you need to know is how to export email content to a text file. In Gmail, open the email, copy the body, and paste it into a plain text editor. The key is giving specific instructions for tone and default responses. 'Reply however you see fit' produces worse results than 'handle quotes this way, handle scheduling that way.'

Case 2) Turning a Meeting Recording into Meeting Minutes

Scene

Thursday afternoon, 4 p.m. Team leader Kim at a small company's planning department has just wrapped up a ninety-minute meeting. The notes he managed to jot down during it amount to three lines. Fortunately, he recorded the whole thing on his smartphone.

Listening back to a recording and writing up the minutes typically takes twice as long as the meeting itself. For a ninety-minute meeting, that's three hours of work. Looks like overtime is guaranteed tonight.

Command

Team leader Kim transfers the recording file (meeting_0513.m4a) to his computer. AirDrop it to the Mac and it's done.

Create a working folder and initialize Git.

mkdir ~/meeting-notes

cd ~/meeting-notes

git init

Put the recording file in this folder and open Claude Code.

"Convert the file meeting_0513.m4a to text, then organize it as meeting minutes. List each participant's remarks separately, and collect decisions and follow-up actions at the bottom. Save the result as meeting_minutes_0513.txt."

How it works

Claude Code first converts the audio file to text. This is called transcription — turning speech into written words. Claude Code doesn't do this itself; it installs a transcription tool like Whisper (an OpenAI speech recognition tool) and uses that. Because YOLO mode is on, it installs the tool without

asking for permission and proceeds immediately.

Once transcription is done, you get raw text. Something like: "uh so that item, let's think about it a bit more, oh but the budget part, I think we need to move on that pretty quickly" — unpolished, spoken-language chunks.

Claude Code then cleans up that text. It separates speakers (though it may not be 100% accurate from voice alone — speakers may be labeled "Speaker A" and "Speaker B"), converts spoken language into written prose, and pulls out decisions and follow-up actions.

The whole process takes around 8 to 12 minutes, depending on the length of the recording.

Output

A file called meeting_minutes_0513.txt. The top section lists the meeting date and time, attendees (based on speaker identification), and agenda items. The middle section lays out remarks in chronological order. The bottom section neatly groups three decisions and four follow-up actions.

Cost

About 10 minutes of your time, and between 1,500 and 2,500 won in API costs. The longer the recording and the more text to transcribe, the higher the cost.

Risks

There are two risks to watch for.

The first is transcription errors. Speech recognition isn't perfect. Proper nouns, technical terms, and quiet voices can be misheard. "Q3 sales" might come out as "Q3 ales." Read through the minutes against the recording before you submit them.

The second is speaker misidentification. If five people were in the meeting, the AI might only distinguish three, or attribute one person's remarks to someone else. Giving it the names in advance helps. Add something like "The attendees are Manager Kim, Deputy Manager Lee, Associate Park, Staff Choi, and Intern Jung" to your prompt.

How to do this yourself

Record your meeting with your phone's voice recorder app. The iPhone's Voice Memos app works fine. Move the file to your computer, put it in your working folder, and give Claude Code the instruction. Tell it the attendees' names upfront. Always read through the minutes once before submitting them.

Case 3) Reading 10 papers and producing a summary report

The scene

11 p.m. on a Friday. Han So-ra, a master's student, turns on the light in her lab. By Tuesday next week she has to present prior research on "the digital transformation of self-employed workers in Korea" at a seminar. Her professor sent her ten papers. Each one runs 20 to 40 pages. Together they add up to more than 300 pages.

To read all ten, map out the overlaps and differences, build a comparison table, and prepare presentation slides, she'd have to spend the entire weekend locked in the lab.

The prompt

Han So-ra puts ten PDF files into a folder called papers.

mkdir ~/paper-review

cd ~/paper-review

mkdir papers

git init

(Copy the paper PDFs into the papers folder.)

Open Claude Code.

"Read the 10 PDF papers in the papers folder and summarize each one: research purpose, methodology, key findings, and limitations. Then add a comparative analysis covering what all 10 papers agree on and where their conclusions diverge. Save the result as prior_research_summary.txt."

What happens

Claude Code begins reading the papers one by one. It extracts text from each PDF, processes the content, and writes a summary. Each paper takes about 2 to 3 minutes, so ten papers run 20 to 30 minutes total.

One problem can come up along the way: scanned PDFs. When a paper was printed and scanned, the file contains images rather than text, so the AI can't read the characters. In that case, Claude Code installs an OCR (Optical Character Recognition) tool to handle the file. It takes a bit longer.

Once all ten individual summaries are done, Claude Code writes the comparative analysis. It pulls together shared findings across papers ("7 out of 10 reported that social media marketing had a positive effect on small business revenue"), conflicting conclusions ("On the impact of delivery apps, 3 papers were positive and 2 were negative"), and an overall picture of where the research stands.

Output

A file called prior_research_summary.txt. Typically 8 to 12 pages in A4 format. The first half contains individual summaries of all 10 papers; the second half is the comparative analysis.

Cost

30 minutes of processing, roughly 5,000 to 8,000 Korean won. Longer and more complex papers push the cost up. Done by hand, this would eat up a full weekend.

Risk

The risk here is different from other exercises. It's not about corrupting files or exposing passwords. It's about accuracy of content.

When an AI "reads" a paper, that doesn't mean it understands it the way a person does. It can misread statistics, reverse cause-and-effect relationships, or summarize a finding opposite to what the author actually claimed. For example, a paper stating "delivery apps reduced revenue by 15%" might get summarized as "delivery apps contributed to revenue growth."

That's why you can't take an AI-generated summary report straight to a presentation. This is a draft, not a finished product. As you read through the report, check that key figures and conclusions match the original papers.

What the AI did was compress 300 pages down to 12. Reading those 12 pages and verifying them is your job. Still, it's far faster than reading 300 pages from scratch.

How to do this yourself

Collect all the PDF papers in one folder. Check that they're text PDFs, not scanned images (open the PDF and try dragging to select text; if it works, it's a text PDF). Be specific about what you want summarized. "Summarize this" produces much weaker results than "summarize the research purpose, methodology, key findings, and limitations." Always cross-check the finished report against the originals.

Case 4) Automatically converting a Markdown manuscript into a publication-ready PDF

The scene

Sunday afternoon. Yoon Se-hyeon, a freelance writer who runs a blog, has just finished an ebook manuscript. Written in Markdown (a lightweight formatting language for writing), the manuscript spans 12 chapters and roughly 180 pages. The plan is to convert it to PDF and sell it as an ebook.

There are several tools for converting Markdown files to PDF, but getting the cover, table of contents, page numbers, and font settings right can eat up half a day of configuration.

The command

mkdir ~/ebook-build

cd ~/ebook-build

git init

Place the manuscript files (chapter_01.md through chapter_12.md) in this folder.

"Take the 12 Markdown files in this folder, merge them in order, and produce a Korean ebook PDF. Add a cover page with the title 'Coffee and Keyboard.' Generate a table of contents automatically and include page numbers. Use NanumMyeongjo font, body text at 11 points. Name the output file CoffeeAndKeyboard.pdf."

How It Works

Claude Code installs a tool called Pandoc, a widely used program for converting Markdown files into PDFs. If the NanumMyeongjo font isn't on your computer, it installs that too.

It merges 12 files in order, adds a cover page at the front, generates a table of contents automatically, and inserts page numbers at the bottom of each page. The PDF is produced using a typesetting tool called LaTeX, a process that chains together the installation and execution of several programs.

In default mode, you would have been asked three separate questions: "May I install Pandoc?", "May I install LaTeX?", "May I install NanumMyeongjo?" In YOLO mode, all three are skipped.

The whole process takes five to eight minutes.

Result

A file called CoffeeAndKeyboard.pdf. The cover displays the title in large text, the next page holds the table of contents, and the body text is cleanly typeset in NanumMyeongjo 11pt.

Cost

Eight minutes of time, roughly $0.75 to $1.10 USD.

Risks

The risk in this example is the number of installations that happen. Pandoc, LaTeX, and a font all get installed on your computer. The installations themselves aren't dangerous, but they take up a fair amount of disk space. A full LaTeX package can exceed 3 gigabytes (GB). If your laptop is running low on storage, check before you start.

After the PDF is generated, you may find garbled fonts, tables that run across page breaks, or images that have shifted out of place. It's worth opening the PDF and scrolling through it from start to finish.

If You Want to Try This Yourself

Your manuscript can be in any format. Plain text files (.txt) are fine. Word files are fine too. Tell Claude Code "turn these files into a book" and it will handle the conversion on its own. It helps to specify the font and size in your instruction, and if you don't like the cover design, a follow-up command like "change the cover background to navy" will fix it.

A Failure Case: When AI Summarized but Left Out the Source

One day, a graduate student asked AI to produce a summary report covering seven papers. The output summarized only six. The seventh paper was missing entirely.

Looking into why, it turned out the seventh paper's PDF was a scanned image. The AI couldn't extract any text from it, and it didn't report that fact either. It quietly skipped the file with no error message.

There's a reason this happens. AI tends to "do what it can and move on" rather than stopping to say "I can't handle this." A person would have said "I can't read paper number seven," but the AI just summarized six papers and reported back: "Done."

The fix is straightforward: count the output. If you asked for ten items, check that ten came back. If you asked for 53 replies, count to 53. When the numbers don't match, something got dropped.

Adding this one line to your instruction helps: "If there are any files you can't read, don't skip them. Tell me the file name and the reason."

Previous chapter: Chapter 13. Three Lessons from the Practice Sessions

|

|

Next chapter: Chapter 15. Working with Tables and Numbers

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 15. Working with Tables and Numbers

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 14. Writing and Document Organization

|

|

Next chapter: Chapter 16. Building Websites and Automation

Case 5) Automatically Consolidating 12 Months of Store Sales Data

Scene

Year-end closing time. The director who runs a Korean snack restaurant pulled out 12 Excel files — one for each month, January through December. Each file contains dates, menu items, quantities, and amounts. The task is to merge everything into a single file and calculate monthly totals and an annual total.

Command

"Merge the 12 xlsx files in this folder into one. Then calculate monthly sales totals and an annual sales total. Also create a sales ranking by menu item. Save the result as 2025_annual_sales.xlsx."

How It Went

Claude Code writes a Python script and runs it. It merges the data from all 12 files, calculates monthly subtotals, and sorts sales by menu item. Two minutes.

Result

The first sheet has the full dataset, the second has a monthly totals table, and the third has the sales ranking by menu item. Tteokbokki came in first for the year (42 million won), with sundae in second (28 million won).

Cost

2 minutes, 400 won.

Risk

As with Chapter 11, there's a risk of overwriting the originals. Save with Git first. Manually calculate one or two months' totals and check them against the output.

How to Do This Yourself

The column names in your monthly sales files may not match. One file might say "Menu", another "Item", another "Product Name". Don't worry. Just tell Claude Code: "The column names may differ across files — figure it out and merge them anyway." Alternatively, before merging, you can say: "Standardize the column names across all 12 files first."

Case 6) Spotting Duplicate Entries in a Client List

Scene

The sales team at a building materials distribution company. Their client list Excel file holds 2,400 company names. But the same company appears to be listed more than once — "Hanbit Construction", "Hanbit Construction", and "Hanbit Construction Co., Ltd." each sit in separate rows. Scanning 2,400 entries by eye to find every duplicate would take all day.

Command

"Find duplicate company names in the client_list.xlsx file. Treat names as the same company even if they differ in spacing, the presence of 'Co., Ltd.', or how the corporate designation is written. Save the list of suspected duplicates as duplicate_check.xlsx."

How It Went

Claude Code normalizes the company names. It preprocesses them by stripping spaces, removing "Co., Ltd.", and dropping "Corporation", then groups similar names together. Using a similarity algorithm (a step-by-step procedure for solving a problem), it flags companies whose names are 80% or more alike as duplicate candidates.

Result

The file duplicate_check.xlsx. It turns up 142 suspected duplicate pairs. Each row shows "Original Name A", "Original Name B", and "Similarity Score".

Cost

Time: 1 minute. Cost: roughly $0.35.

Risk

AI can group different companies together as if they were the same one. "Hanbit Construction" and "Hanbit Building Materials" are separate companies, but because the names look similar, the AI might flag them as duplicates. That's why we asked for a "suspected duplicate list" rather than a definitive "these are duplicates." The final call is always made by a person.

How to try this yourself

Tell the AI the name of the column that contains your client names — something like "Company names are in column B." You can also adjust the similarity threshold. Say "only catch matches above

90%" and the list gets shorter; say "70% or above" and it gets longer.

Case 7) Pulling numbers from receipt photos into Excel

The scene

Tax season is coming. A full year's worth of receipts is stuffed into one drawer — card receipts, cash receipts, handwritten slips. More than 200 in total. Entering each one into Excel by hand is a two-day job.

The command

Take photos of the receipts with your smartphone — one receipt per photo. Put all 200 photos into a folder called receipts.

"Read the 200 receipt photos in the receipts folder and pull out the date, store name, amount, and payment method from each one. Save the results as a file called receipts_organized.xlsx. Sort everything by date."

What happens

Claude Code reads each photo. It uses OCR to recognize the text in the image and figures out where the date is, where the amount is. Each receipt takes about 10 to 15 seconds, so 200 receipts means 30 to 50 minutes total. You can do other things while the computer works.

The result

A file called receipts_organized.xlsx. Two hundred rows sorted by date, each row containing the date, store name, amount, and payment method. A total sum is included at the bottom.

Cost

Time: 40 minutes (most of it the computer working on its own). Cost: roughly $3 to $4.50.

Risk

Recognition accuracy depends heavily on photo quality. A clearly shot card receipt comes in above 95% accuracy, but a crumpled or faded handwritten slip can drop to 70%. A single wrong digit in an amount could cause problems on your tax return, so any receipt over 100,000 won should be checked against the original.

A tip for taking photos: lay the receipt flat on the floor and shoot straight down from above so there's no shadow. Shooting at an angle distorts the text and brings recognition accuracy down.

How to try this yourself

Getting a clear photo is half the battle. The other half is checking the results. You can't verify all 200, so pick 20 at random (10%) and compare them against the originals. An error rate of 5% or below is acceptable.

Case 8) Sentiment analysis of 200 YouTube comments, then generating a pie chart in Excel

Scene

CEO Choi runs a small cosmetics brand and posted a new product review video on YouTube. Within a week, 237 comments had piled up. 'The scent is lovely.' 'The price is a bit...' 'The packaging is so pretty I bought it as a gift.' 'This will sell out fast.' 'Not impressed this time.' Reading each one is possible, but she wants to know at a glance whether the overall response is good or bad.

Command

First, pull the YouTube comments into a text file. You can copy the comments manually, or use a tool like yt-dlp (a utility for downloading YouTube videos and metadata) to extract them. You can ask Claude Code to handle this too.

"Grab all the comments from this YouTube video URL and save them to comments.txt."

Once the comments are saved, ask for the analysis.

"Run a sentiment analysis on the 237 comments in comments.txt. Sort them into positive, negative, and neutral, then make a pie chart showing the percentage of each category. Pull out the five keywords that appear most often in the negative comments. Save the results to comment_analysis.xlsx and put the pie chart inside the Excel file."

What happens

Claude Code reads through the comments line by line and assigns a sentiment to each one. 'The scent is lovely' gets marked positive. 'The price is a bit...' gets marked negative. 'The packaging is so pretty I bought it as a gift' gets marked positive. Once classification is done, it calculates the percentages: 62% positive, 23% negative, 15% neutral, something like that.

It builds the Excel file in Python and uses openpyxl to embed the pie chart directly inside a worksheet. The keywords that come up most in the negative comments ('price', 'volume', 'shipping', 'scent longevity', 'breakouts') get pulled out and organized on a separate sheet.

The whole process takes five to eight minutes.

Result

A file called comment_analysis.xlsx. Sheet 1 has all the comments with a sentiment label (positive/negative/neutral) next to each one. Sheet 2 has the pie chart and a percentage summary. Sheet 3 has the keyword breakdown from the negative comments. When CEO Choi opens the file, she can see in one glance: '62% positive, so the response isn't bad — but people are unhappy about

the price and the volume.'

Cost

Eight minutes of time, roughly $1.10 to $1.80 in API costs.

Risk

Sentiment analysis accuracy hovers around 80%. The problem is sarcasm. A comment like 'Wow, truly impressive — this much product for this price' is clearly negative, but the AI may well classify it as positive. Korean irony is something AI misses fairly often.

A perfect analysis is out of reach. Use this to get a read on the overall picture, and don't put too much weight on the exact numbers.

Trying this yourself

The first step is getting the YouTube comments into a text file. You can tell Claude Code: 'Grab the comments from this YouTube URL.' It also helps to spell out your classification rules ahead of time, something like: 'Treat complaints about price as negative, and treat simple questions as neutral.'

A failure case: AI that misread the numbers

Someone handed 100 receipt photos to an AI. When they opened the resulting Excel file, the totals looked wrong. Their monthly spending was usually around 3.5 million won, but the AI's total came out to 8.3 million won.

Tracing the cause, they found three receipts where '35,000 won' had been read as '350,000 won' — the comma position was misread. There were also two cases where '8,500 won' became '85,000 won'. On faded receipts, an extra zero had effectively appeared out of nowhere.

There's something worth keeping in mind when using AI for work that involves numbers. AI doesn't 'roughly guess' — it 'gets things wrong with full confidence'. It will write down 350,000 won in place of 35,000 won without a moment's hesitation. It won't flag anything with a question mark.

A quick sanity check: look at the total first. Confirm it falls within the expected range. If it doesn't, work down from the largest amounts.

Previous chapter: Chapter 14. Writing and Document Organization

|

|

Next chapter: Chapter 16. Building Websites and Automation

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 16. Building Websites and Automation

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 15. Working with Tables and Numbers

|

|

Next chapter: Chapter 17. Office Automation You Can Use

Case 9) A Personal Blog, from Idea to Deployment

The Scene

Park Ji-ho, a college student, wanted to start a blog for beer reviews. He could have used a platform like Naver Blog or Tistory, but he wanted his own domain — something like beerlog.kr. The problem was he had never built a website before. He had heard the word HTML, but had never written a line of code.

The Command

mkdir ~/beerlog

cd ~/beerlog

git init

He opens Claude Code and gives it this instruction.

"Build me a beer review blog. The name is 'One Glass of Beer: A Record.' The home page should show a list of posts, and clicking a post should display the full review. Each review includes the beer name, brewery, star rating out of 5, one photo, and the body text. Keep the design clean and dark in tone. When I write a post as a Markdown file, it should automatically become a page."

What Happens

Claude Code sets up a static site generator — a tool that turns Markdown files into web pages. It picks one from the available options, something like Hugo or Eleventy, and installs it.

It creates the folder structure, configures a theme (the design template), and generates one sample review post. The home page lists all posts, and clicking one shows the full text.

It then starts a local preview — visible only on your own machine.

hugo server

Run that command and you can see the blog in your browser at http://localhost:1313. Only your computer can see it at this point; no one else can.

This takes about 15 minutes.

To go further and actually publish it online, give Claude Code one more instruction.

"Deploy this blog to Vercel."

Vercel — a service that hosts websites on the internet — is free for personal projects. Claude Code creates a GitHub repository, pushes the code up, and connects it to Vercel.

Including deployment, the whole process takes 25 to 30 minutes.

The Result

A blog that's live on the internet. Anyone can visit it at an address like beerlog.vercel.app. If you want to connect your own domain (beerlog.kr), you'll need to buy one separately — around 15,000 won a year.

Cost

Time: 30 minutes. Cost: 3,000 to 5,000 won. Server hosting: free (Vercel free plan). Add a custom domain and it's another 15,000 won per year.

The Risk

The moment you deploy — the moment something goes live on the internet — risk appears. If your code contains any personal information, it becomes visible to the entire world. API keys, passwords, and personal email addresses hard-coded directly into the source files are a serious problem. This case is a personal blog, so there is less chance of sensitive data being involved, but make it a habit to check every time.

Before deploying, give Claude Code this instruction: "Go through every file in this folder and check whether any sensitive information — passwords, API keys, tokens — is exposed anywhere." Claude Code will scan the files one by one and flag any lines containing words like password, secret, api_key, or token. If nothing comes up, you're clear.

How to try this yourself

Give the AI a clear sense of your blog's purpose and visual tone. "Make me a blog" gets worse results than "a beer review blog, dark theme, with a star rating system." For hosting, use a free service like Vercel or Netlify. You'll need a GitHub account, which is free.

Case 10) A neighborhood group schedule site

The scene

Kim Young-suk runs a book club in her apartment complex. Every month she posts the next meeting date and reading selection to a group chat, but whenever a new member joins, she has to send all the old information again. Past records are hard to dig up too. A simple webpage with the schedule would solve everything with a single link.

The prompt

"Build me a website with a schedule for my book club. One page is enough. At the top, the club name 'Reading Together on an Afternoon'; below that, the next meeting details (date, location, book); below that, a log of past meetings. Pull the data from a JSON file so that whenever I edit the JSON file, the webpage updates automatically."

What happens

Claude Code builds the page in HTML and CSS (the language that controls how a webpage looks), and creates a data file called schedule.json. A JSON (JavaScript Object Notation) file is plain text that any person can read, so you can open it in Notepad and change the date or book title whenever you like.

Done in about ten minutes. Upload it to Vercel or GitHub Pages and you get a web address. Post that link in the group chat and you're set.

The result

A clean, single-page schedule site. The next meeting details sit prominently at the top; scroll down and you'll find the history of past gatherings.

The cost

Time: 10 minutes. Cost: roughly $0.35. Hosting: free.

Watch out for

The site is public, so don't post specific location details like a unit number. Use insider shorthand that only members would recognize, such as "the community room in the complex."

How to try this yourself

Tell the AI the club name, the mood you want, and what information you need to show (date, location, topic, and so on). If you're not sure how to edit the JSON file, just ask Claude Code: "How do I add the July meeting to schedule.json?"

Case 11) A reservation system for a small shop

The scene

A small flower shop in Yeonnam-dong, Seoul. The owner had been taking reservations through Instagram DMs. Messages like "Could you make a bouquet for me Friday at 3 p.m.?" came in more than ten times a day. Reply, write it in the notebook, check for overlapping slots. When DMs piled up, bookings slipped through.

A reservation system would help, but Naver Booking and Kakao Booking both take a commission. Building one herself would mean no commission, but hiring a developer was going to cost at least two million won.

The prompt

"Build me a reservation system for a flower shop. Customers pick a date and time on a webpage, enter their name and phone number, and the reservation gets saved. If two people try to book the same slot, show them a message saying 'This time is already taken.' Let me view all reservations on an admin page. Set the admin password to flower2025."

How It Works

This case is more complex than the previous ones, because data needs to be saved and retrieved. Claude Code builds both the frontend (the screens users see) and the backend (the layer that processes data behind the scenes).

Frontend: a screen where customers pick a date from a calendar, see available time slots, and enter their name and phone number.

Backend: logic that saves booking details to a database and prevents double-bookings for the same time slot.

Admin page: a screen that shows today's booking list after the owner enters a password.

Claude Code can build this by combining Next.js (a web framework) and Supabase (a free database service). Both have free plans, so small shops pay nothing to get started.

The whole process takes 40 minutes to an hour. That's longer than the other cases, but outsourcing this to a developer would take days and cost anywhere from hundreds of thousands to millions of won.

The Result

A booking page published on the internet. Put the link in your Instagram profile, and customers book themselves. The owner checks the booking list from the admin page.

Cost

Time: 1 hour. API cost: 5,000–8,000 won. Monthly server cost: 0 won (within the free plan). If you buy a domain: 15,000 won per year.

Risks

The risks here deserve real attention. A booking system stores customers' names and phone numbers. That's personal data. If it leaks, you could face legal problems.

Supabase encrypts data by default, but if your admin page password is something like "flower2025", someone could break in. Make your password long and complex — something like "Fl0w3r!2025#SeouL".

Check that passwords are not written directly in the code. They should go into environment variables (a way to store sensitive information outside the code itself). This is covered in detail in the failure case at the end of Chapter 16.

If You Want to Try This Yourself

You don't need to know how to use Supabase or any other external service. Claude Code handles all the setup. You do need to create a free account on the Supabase website first. Once you do, you'll get an API key — give that key to Claude Code and it will make the connection. If your bookings exceed 100 a day while the service is running, you may need to upgrade to a paid plan.

Case 12) An AI That Opens a Browser and Runs the Tests Itself

The Scene

You've built a website. It seems to be working, but you want to confirm that buttons actually do something when clicked and that text doesn't get cut off on a smartphone screen. Clicking through everything by hand is a hassle.

The Prompt

"Open the website we just built in a browser, click every button and link, and let me know if anything throws an error — then fix it. Also check how it looks at mobile screen width (375px)."

How It Works

This is where a feature called Browser Use comes in. Browser Use is a tool that lets AI operate a real browser directly. The AI opens an actual browser, moves the mouse, clicks buttons, and types text — just like a person would.

Claude Code uses Playwright (a browser automation tool) to open a browser, load the webpage, and click through each button one by one. When it finds a broken link, it reports the problem. "Clicking the Contact button throws a 404 error." Then it fixes it on its own.

It also checks the site at mobile screen size. If any text overflows the viewport, it edits the CSS.

The whole process takes five to ten minutes.

Result

A tested website. Every button works, and the layout holds up on mobile. A list of what the AI fixed is printed in the terminal.

Cost

Ten minutes of your time, roughly $0.75 to $1.50.

Risk

Browser Use is a feature that lets the AI access the internet. As with the news-summarizing example in Chapter 12, running it inside Docker is the safer choice. During testing, the AI may visit external sites or open pages you didn't expect.

How to try this yourself

Right after you build a website, just say "test it in a browser." Playwright will install itself automatically. Be specific about what you want tested: something like "every button, every link, and mobile screen size."

Failure case: a password left exposed in the code after deployment

This is the story of a flower shop owner (a follow-up to Case 11). She built a reservation system, pushed the code to GitHub, and deployed it to Vercel. Everything ran fine. For two weeks, bookings came in without a hitch.

Then one day she logged into the admin page and found the reservation list empty. Someone had gotten into the admin panel and deleted every booking.

Here is what happened. The admin password "flower2025" was written directly in the code. Code pushed to GitHub is visible to anyone (the repository was public). Someone read through the code, found the password, and walked right into the admin page.

She had told the AI to "set the admin password to flower2025," so the AI wrote that password into the code. The AI did nothing wrong. It did exactly what it was asked. The mistake belonged to the person who never told it to store the password outside the codebase.

Sensitive information like this should be stored in an environment variable file (a .env file), and that file should never be pushed to GitHub. Add .env to your .gitignore file (the list of files Git is told to ignore) and it stays off the repository.

How to prevent this: when giving the AI its instructions, add this one line. "Put passwords and API keys in a .env file, and add .env to .gitignore."

Previous chapter: Chapter 15. Working with Tables and Numbers

|

|

Next chapter: Chapter 17. Office Automation You Can Use

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 17. Office Automation You Can Use

Previous chapter: Chapter 16. Building Websites and Automation

|

|

Next chapter: Chapter 18. Getting Big Work Done Overnight

Case 13) Sorting Gmail inbox and drafting replies

Scene

Monday morning, 9 a.m. Oh Ji-hyeon, operations team lead at a startup, opens Gmail. 87 emails piled up over the weekend. Promotional mail, newsletters, messages from business partners, team reports, and customer inquiries all mixed together. She needs to sort them, reply immediately to the urgent ones, and star the rest for later.

Command

You can pull emails through the Gmail API (a channel that lets programs talk to each other). You'll need to get an API key from Google, but you only have to set that up once.

"Fetch all unread emails from my Gmail inbox and sort them by category. Use these categories: 'Urgent', 'Business Partners', 'Internal Team', 'Newsletter', and 'Other'. For urgent emails, write a draft reply. Save the results to mail_summary_0513.txt."

How it runs

Claude Code pulls all 87 emails via the Gmail API, reads the subject and body of each one, and assigns a category. "Please confirm the payment" goes to Business Partners, "Monday weekly report" goes to Internal Team, "Weekly Newsletter" goes to Newsletter.

For the 5 emails flagged as urgent, it writes draft replies. Short, to-the-point responses like "Noted. I'll take care of it by end of day."

The whole thing takes 5 minutes.

Result

A file called mail_summary_0513.txt. Emails are organized by category, and each urgent email has a draft reply attached. Oh Ji-hyeon skims the file, fires off the urgent replies right away, and saves the rest for when she has time.

Cost

5 minutes of time, roughly $1.10 to $1.80 in API costs.

Risk

Emails can contain confidential company information. When you use the Claude API, email content passes through Anthropic's servers. Depending on your company's security policy, this may not be allowed. If any of those emails carry sensitive information, talk to your information security officer before using this method.

How to try it yourself

You'll need to enable the Gmail API in Google Cloud Console and generate credentials. The first-time setup is a bit involved, but if you ask Claude Code "walk me through setting up the Gmail API," it'll guide you step by step. Once it's set up, you can keep using it going forward.

Case 14) Sorting through Slack channel chatter and sending a summary

Scene

Friday afternoon, 5 p.m. Over 400 messages have piled up in the Slack (Slack, a workplace messaging app) channel throughout the week. Most of it is casual chat, and the five or so actual decisions are buried somewhere in there.

Command

"Pull this week's messages from the Slack #general channel, from Monday through today, and filter for anything work-related. Summarize it. List out decisions made, action items, and links that were shared. Post the results to the #general channel with the title 'This Week's Summary.'"

How it runs

Claude Code pulls messages via the Slack API, separates small talk from work-related content, and writes a summary. Three minutes.

Result

A summary message appears in the Slack channel. Something like: "Decisions this week: Project A schedule pushed back two weeks, Contract renewal with Client B finalized, Priority change for Feature C..."

Cost

Three minutes of your time, about $0.60 in API costs.

Risk

The same security concern as Case 13 applies. Your Slack message content passes through an external server.

How to replicate this

You need to create a Slack app and generate an API token. Slack admin permissions are required.

Case 15) Auto-updating a Notion page

Scene

You keep meeting notes in Notion, a document management tool. A new meeting notes file gets added every week, and you have to manually add a link to each one on the "Meeting Notes Collection" page. If you forget and let it pile up, catching up later is a chore.

Command

"Using the Notion API, whenever a new entry is added to the 'Meeting Notes' database, automatically add a link to the 'Meeting Notes Collection' page. Check and update it every day at 6 p.m."

How it works

Claude Code connects to the Notion API, queries the "Meeting Notes" database, and builds a script that updates the "Meeting Notes Collection" page. It then schedules that script to run automatically every day at 6 p.m.

Result

The "Meeting Notes Collection" page updates itself automatically every day at 6 p.m.

Cost

Ten minutes to set up, then around $0.07 per day for the automated runs.

Risk

If your Notion API key leaks, someone could read everything in your Notion workspace. Store the API key in an environment variable.

How to replicate this

Go to the Notion developer portal (https://developers.notion.com), create an Integration, and generate an API key. You also need to connect that Integration to the relevant database before it can access anything.

Case 16) Competitor Website Change-Tracking Report

The Situation

Kim, who runs an online shop, checks three competitor websites once a week. He wants to know whether prices have changed, whether new products have gone up, whether any promotions are running. Browsing all three sites takes an hour; writing up the changes takes another thirty minutes.

The Command

"Visit the following three websites every Monday morning at 7 a.m.

1) competitor-a.com/products

2) competitor-b.com/new-arrivals

3) competitor-c.com/events

Save the content from each site, compare it with last week's version, and summarize what has changed. Save the results to competitor-changes-this-week.txt."

How It Works

Claude Code builds a web scraping script (web scraping means automatically reading the content of a website). Each week it fetches the content of all three sites, saves it as text, and compares it against last week's saved copy. It then identifies newly added products, items whose prices have changed, and new promotions, and compiles everything into a report.

The Result

Every Monday morning, a file called competitor-changes-this-week.txt appears. It contains entries like: "Brand A: new hand cream launched (₩15,000), Brand B: 20% off everything (through May 20), Brand C: no changes."

Cost

15 minutes to set up, then roughly ₩1,000 per automated weekly run.

Risk

Some sites do not allow web scraping. If a site's terms of service say "automated collection prohibited," using a scraper could create legal problems. Check before you proceed. It's also worth looking at the site's robots.txt file (a file that spells out what automated tools are and aren't allowed to collect).

If the site's structure changes, the scraper will break. When you receive a report saying "an error occurred," you'll need to update the script.

How to Try This Yourself

Give Claude the URLs of the competitor sites and be specific about what you want to track (prices, new products, promotions). Running the script inside Docker is the safer approach.

A Failure Story: The Time an Important Email Got Flagged as Spam

Someone who had handed Gmail sorting over to AI, much like in Case 13, found that it worked fine for a week. Then on Friday, a call came in from the head of a client company.

"Did you see the contract email I sent last Tuesday?"

They had not. The AI had filed that email under "Newsletters," so it never caught their eye. The subject line the client had used was "Collaboration Proposal Related to the May Newsletter." The word "newsletter" appeared in the subject, and that was enough for the AI to sort it accordingly.

A person would have looked at the sender and thought, "Oh, this is from our client's CEO — it can't be a newsletter." The AI reacted more strongly to the keywords in the subject line.

Prevention tip: give the AI the email addresses of your key clients in advance. Tell it, "Any mail from these addresses goes straight into 'Urgent,' no exceptions." And once a day, skim through everything the AI has sorted. Automated sorting is convenient, but hand it 100% of the control and accidents like this will happen.

Previous chapter: Chapter 16. Building Websites and Automation

|

|

Next chapter: Chapter 18. Getting Big Work Done Overnight

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 18. Getting Big Work Done Overnight

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 17. Office Automation You Can Use

|

|

Next chapter: Chapter 19. Things I Have Broken

Case 17) Restructuring 100 Code Files Overnight

The Scene

Wednesday, 11 p.m. Jang Seo-jun, the development team lead, sat in front of his monitor, turning a problem over in his head. The company's web service codebase had gotten old. The same pattern was repeated across 100 files, and the whole thing needed to be migrated to a new structure. Done by hand, that's a week of work. Open each file one by one, edit the code, run the tests.

Jang Seo-jun decided to try a different approach: hand it off to AI and go to bed.

The Command

First, he created a new branch in Git.

git checkout -b refactor-structure

The idea was to leave the original code untouched and do all the work on a separate branch. If something went wrong, he could just delete the branch and walk away.

He opened Claude Code.

"In all .ts files inside the src folder, replace the old-style import syntax with the new style. Specifically, change 'import { X } from '../utils/helpers'' to 'import { X } from '@/utils/helpers''. After each change, check whether the file compiles without errors. If any file fails to compile, revert it to the original and give me a list of those files."

How It Went

Claude Code scanned the src folder. 112 .ts files. It opened each one, found the pattern, and made the replacement. After each change, it ran the TypeScript compiler to check for errors.

When a file threw an error, Claude Code rolled it back and added it to a list marked "requires manual review."

Processing all 112 files took 45 minutes to an hour. Jang Seo-jun gave Claude Code its instructions and went to sleep.

The Result

Thursday morning, 8 a.m. Jang Seo-jun arrived at the office and checked the terminal. A message was waiting: "108 of 112 files converted successfully. 4 files require manual review."

He ran git diff to see what had changed. The import statements in 108 files had been updated to the new style. The 4 remaining files came with a note explaining that converting them had caused errors elsewhere, so they had been left as-is.

Jang Seo-jun reviewed and fixed the 4 files himself. Thirty minutes, done. What would have taken a week was finished in a single night.

The Cost

1 hour of AI work. Cost: roughly 10 to 17 USD. Had a person done the same work over a week, the labor cost would have run into the millions of won.

The Risk

Having AI modify code at scale overnight carries real risk. That's why the three things Jang Seo-jun did matter.

He created a branch. The original code was never touched. If anything had gone wrong, deleting the branch would have been enough.

He built in a compile check. The instruction wasn't just "make the changes" — it was "make the changes, then verify them."

He told the AI to revert any file that failed. So the AI couldn't force a broken fix through.

If any one of these three was missing, you might have walked into the office that morning to find the codebase in ruins.

How to do the same

When handing off large-scale code changes overnight, always work on a new branch. Include "confirm" and "revert on failure" in the instructions. In the morning, run git diff to review what changed, run your tests, then merge into the original branch.

Case 18) Auto-detect an outage at dawn, auto-fix it, and open a PR

The scene

3:17 a.m. The web service goes down. Datadog, the server monitoring tool, catches a spike in the error rate and fires a Slack alert. Normally, the on-call developer wakes up, opens a laptop, checks the logs, patches the code, and ships a fix — an hour gone. While all that is happening, users are sending messages saying the site is down.

This company does it differently. They built a pipeline — an automated workflow — that spins up Claude Code automatically whenever a Slack alert comes in.

The instruction

No one is issuing commands in real time. This is all pre-configured automation. When a Slack outage alert arrives, a webhook — a connection that triggers automatically when a specific event occurs — fires a script, and the script starts Claude Code.

The instruction passed to Claude Code looks like this.

"Analyze the error logs. Find the cause in the error messages and check the related code files. If you can fix it, create a new branch, apply the fix, and open a PR (Pull Request) on GitHub. Prefix the PR title with '[Auto-fix]'. If you can't fix it, post an error analysis report to Slack."

How it played out

3:17 a.m. Claude Code starts automatically. It reads the error logs. The same error keeps repeating: "TypeError: Cannot read property 'email' of null". Something is returning null — an empty value — when it tries to fetch user data.

It tracks down the relevant code file. Line 42 of user-profile.ts reads the user's email, but the code has no handling for the case where user data doesn't exist.

Claude Code creates a new branch and adds a null check — code that verifies whether a value is empty — to line 42. Compilation check. Tests run. All pass.

A PR goes up on GitHub. PR title: "[Auto-fix] Add null check in user-profile.ts"

A message goes to Slack. "Outage detected at 3:17 a.m. Cause: unhandled null in user-profile.ts. Fix PR submitted. Please review and merge."

3:32 a.m. Fifteen minutes from detection to PR.

The result

The on-call developer comes in the next morning, reviews the PR, checks that the code is correct, and merges it. The incident was resolved in 15 minutes, and nobody had to wake up in the middle of the night.

The cost

15 minutes of automated work, roughly $2 to $4. Far cheaper than paying an on-call developer overnight rates.

The risk

An auto-generated fix can be wrong. So this company put two safety mechanisms in place.

It only opens a PR — it doesn't deploy automatically. A person has to review and merge it before anything ships.

PR titles are prefixed with "[Auto-fix]" to distinguish them from human-created PRs. Auto-fix PRs get reviewed more carefully.

What if you had set it to deploy automatically too? The AI could push a bad fix and deploy it on its own, making the incident worse.

How to try this yourself

This setup works when you have a development team of some size and are already using Slack and GitHub. You need to build the pipeline ahead of time: monitoring tool (Datadog, Sentry, etc.) → Slack alert → webhook → Claude runs → PR created. You can have Claude Code build the pipeline itself. Just tell it: "Set up an automated incident-response pipeline for overnight outages."

Case 19) Running 20 sub-agents in parallel to review the codebase

The scene

Friday at 6 PM. A major update is scheduled to ship Monday, and you want to do a full sweep of the code before then. The project has over 2,000 files. One person doing it manually would take two weeks.

The command

Claude Code has a feature called sub-agents. One Claude Code instance can spawn multiple smaller Claude Code instances and run them all at the same time.

"Split the src folder of this project into 20 sections, then run one sub-agent per section. Each sub-agent should inspect the files in its assigned section and look for: unused variables, missing error handling, and security-vulnerable patterns (hardcoded passwords, SQL injection risks). Collect all findings in code_review_results.txt."

How it runs

Claude Code divides the src folder into 20 sections, distributing files as evenly as possible. All 20 sub-agents start at the same time.

Each sub-agent reads through its assigned files and looks for problems. The sub-agents work independently of each other. If one slows down, the others aren't affected.

With 20 running in parallel, a task that would take one agent 5 minutes gets done 20 times faster. The full review takes 15 to 25 minutes.

The result

A file called code_review_results.txt, with findings organized by section. Entries like: "auth.ts line 23: password hardcoded", "db-query.ts line 55: potential SQL injection", "utils.ts: 3 unused variables". Dozens of items like these.

The cost

25 minutes of runtime, roughly $28–$49. Because 20 sub-agents hit the API simultaneously, the cost runs higher than other cases. Still trivial compared to what 20 people doing the same code review would cost.

The risk

If 20 sub-agents all try to edit files at the same time, you get conflicts. That's why this example instructs them to "inspect only — don't make any changes." Read-only work carries no conflict risk.

If you do want them to make fixes as well, each sub-agent should work on a separate branch.

Watch the costs. As the number of sub-agents grows, the bill grows with it. Running 50 instead of 20 means 2.5 times the cost. It's worth setting an API usage cap in advance.

How to try this yourself

The sub-agent feature is available starting with Claude Code 2026. It works well for large projects (500 or more files). For small projects, a single Claude instance is enough without sub-agents. Start with 3 to 5 sub-agents, and if the results are good, increase the number.

A failure story: the agent that repeated the same error 300 times overnight

One developer handed off a code refactoring job to AI at night and went to sleep. By morning, the terminal had more than 300 lines of error messages. The same error message, repeated 300 times.

Here's what happened. The AI modified the code, but the tests failed. The AI said "I'll fix it" and tried again. Failed again. Fixed again. Failed again. This went on 300 times.

A person would have stopped around the third attempt and thought, "My whole approach must be wrong." The AI didn't stop. Same fix, same failure, same fix again. On and on.

The cost was the real problem. Those 300 attempts ran up about $85 in API charges. In a single night. The code was right back where it started, and the money was gone.

There are two ways to prevent this.

Set a retry limit. Add something like "if the same error comes up 3 times, stop and report back" to your prompt. That one line would have saved $85.

Set an API spending cap. You can set a daily usage limit in the Anthropic dashboard. If you cap it at around $14, the API stops once you hit that amount. Even if the AI runs all night, it stops at $14.

[End of Part 5] Use cases at a glance

Case 1. 50 email reply drafts. Time: 3 minutes, Cost: $0.35.

Case 2. Meeting recording turned into meeting minutes. Time: 10 minutes, Cost: $1.40.

Case 3. Summary report from 10 research papers. Time: 30 minutes, Cost: $4.20.

Case 4. Markdown converted to print-ready PDF. Time: 8 minutes, Cost: $0.85.

Case 5. 12-month sales figures tallied. Time: 2 minutes, Cost: $0.28.

Case 6. Duplicate vendors identified. Time: 1 minute, Cost: $0.35.

Case 7. 200 receipts transferred to Excel. Time: 40 minutes, Cost: $3.50.

Case 8. YouTube comment sentiment analysis. Time: 8 minutes, Cost: $1.40.

Case 9. Personal blog built from scratch. Time: 30 minutes, Cost: $2.80.

Case 10. Neighborhood group schedule. Time: 10 minutes, Cost: $0.35.

Case 11. Flower shop booking system. Time: 1 hour, Cost: $4.55.

Case 12. Automated browser testing. Time: 10 minutes, Cost: $1.05.

Case 13. Gmail sorting and reply drafts. Time: 5 minutes, Cost: $1.40.

Case 14. Slack weekly summary. Time: 3 minutes, Cost: $0.56.

Case 15. Auto-updating a Notion page. Setup time: 10 minutes, then fully automatic.

Case 16. Tracking competitor websites. Setup time: 15 minutes, then fully automatic.

Case 17. Restructuring 100 code files. Time: 1 hour, cost: 20,000 won.

Case 18. Automated response to a late-night outage. Time: 15 minutes (automatic), cost: 4,000 won.

Case 19. Code review across 20 sub-agents. Time: 25 minutes, cost: 55,000 won.

One thing every case had in common: we committed to Git before starting, then checked the results with our own eyes.

Part 6. How to Survive When Things Go Wrong

Previous chapter: Chapter 17. Office Automation You Can Use

|

|

Next chapter: Chapter 19. Things I Have Broken

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 19. Things I Have Broken

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 18. Getting Big Work Done Overnight

|

|

Next chapter: Chapter 20. Emergency Repair Steps

It was three o'clock on a Saturday afternoon. I poured a cup of coffee, came back to my desk, and looked at the monitor. The terminal screen had gone black and frozen. There were traces of Claude Code having worked busily at something, but the project folder was completely empty. Not a single file remained.

In this chapter I've gathered eight incidents I either experienced myself or saw reported in the community. Each one stung at the time, but having lived through all of them, I don't rattle easily anymore.

Incident 1. The day the entire project folder vanished

What happened. I told it to "clean up this project." Claude Code interpreted "clean up" as "delete unnecessary files." It started removing test files, temp files, and log files, but then decided the source code inside the src folder was an "old version" and wiped it with rm -rf. That command means "delete everything, no questions asked." Nothing goes to the trash. It's just gone.

How I survived. Git saved me. I typed git checkout -- . and the files came back exactly as they were at the last commit. I had committed three hours earlier, so three hours of work was lost, but that beat losing two full days.

How to prevent it next time. Be specific with instructions. Instead of "clean this up," write "delete only the .log files inside the logs folder. Don't touch anything else." And add a line to the CLAUDE.md file that says "Never run the rm -rf command." That one line is your insurance policy.

Incident 2. The day an API key got pushed to GitHub

What happened. I told Claude Code to "push this project to GitHub." It did exactly that. It added every file in the folder with git add ., then pushed everything to GitHub with git push. The problem was that the folder contained a .env file. That file held an API key. An API key is like your credit card number. If someone gets hold of it, they can use AI freely under your name and at your expense.

Twelve minutes after the push, an email arrived: "Your API key has been compromised." GitHub has bots that continuously scan uploaded code looking for API keys. Twelve minutes is actually fast. In some cases they're caught within 30 seconds of being pushed.

How I survived. I went to the Anthropic console (console.anthropic.com) and immediately revoked that API key. I generated a new one and deleted the offending commit from GitHub. Fortunately the charge came to only about four dollars, because I caught it within twelve minutes. If a full day had passed, it could have been hundreds of dollars.

How to prevent it next time. Create a .gitignore file. Add a single line that says .env, and Git will ignore the .env file no matter what. Even running git add . won't include it. Setting up .gitignore is the very first thing you should do when starting a project. Using the default template GitHub provides makes it easy.

Incident 3. The day the AI touched system files

What happened. I asked it to change my terminal settings on my Mac. Specifically, "make my terminal prompt look nice." Claude Code modified the .zshrc file, but then went further and touched system configuration files inside the /etc folder. When I rebooted, the terminal wouldn't open properly. To be precise, it opened, but typing produced garbled characters. Korean input had completely broken.

How I survived. I opened a terminal on another Mac, connected remotely via SSH to the broken machine, and restored the .zshrc file to its original state. The system files were recovered from a Time Machine backup. If I hadn't had Time Machine running, things would have gotten quite difficult.

How to prevent it next time. Do the work inside Docker. Inside a Docker container, no matter how many system files Claude Code changes, it's only touching a fake system inside the container, so your real Mac is unaffected. Also, add a line to CLAUDE.md: "Files inside /etc, /usr, and /System may be read but must not be modified."

Incident 4. The day I woke up to a $230 bill

What happened. On a Friday night I kicked off a large project in YOLO mode and went to bed. I had told it to "refactor this entire website." Refactoring means restructuring code to make it cleaner and better organized. When I woke up in the morning and opened the Anthropic console, the usage graph had shot straight up.

What Claude Code had been doing all night: fix a file, run tests, if the tests fail fix it again, run the tests again. It repeated that loop hundreds of times. Every single call to the model costs money. With the Sonnet model, it's $3 per million input tokens and $15 per million output tokens. Hundreds of calls through the night adds up to around $230.

How I survived. I set a spending limit in the Anthropic console. You can cap it at something like "don't spend more than $40 this month." The $230 that had already gone out was gone, but from the following month it stopped hard at $40.

How to prevent it next time. Always check the spending limit before going to bed. When giving it a large task, define the scope: "fix only 10 files, then stop." Saying "refactor everything" is a dangerous

instruction. There's no natural endpoint. Also use the max turns setting. Running claude --dangerously-skip-permissions --max-turns 20 makes Claude Code stop automatically after 20 responses.

Incident 5. The day it deployed to the wrong server

What happened. I asked it to deploy to the test server, and Claude Code deployed to the production server instead. The addresses of the two servers were similar: staging.myapp.com and myapp.com. Claude Code can't tell them apart, of course. My instruction only said "test server" without giving the exact address.

Unfinished code went live on the production server. Customers started visiting and saw a broken page. The phone calls began.

How I survived. I rolled back to the previous version using the deploy history on the production server. Services like Vercel and Netlify let you return to any previous deployment with a single click. Customers saw the broken page for about fifteen minutes.

How to prevent it next time. Write the server address clearly in CLAUDE.md. Something like: "Deploy only to staging.myapp.com. Never deploy to myapp.com." You can also put the deploy command itself on the forbidden list in CLAUDE.md. Let humans handle deployments, and keep Claude Code focused on writing code.

Incident 6. The day AI repeated the same mistake 200 times

What happened. I asked it to fix a Python script. An error came up, so Claude Code fixed the code — and another error appeared. Claude Code fixed it again. Another error. Fixed again. Another error. This went on more than 200 times.

Looking at the logs afterward, it had been alternating between the same two lines. Change A to B, get an error. Revert B back to A, get a different error. Back and forth. A person would eventually realize "this approach isn't working" — but AI can't make that judgment. It never gets tired.

How I survived. I pressed Ctrl+C to stop it. When I ran git log to check the commit history, over 200 commits had piled up. I ran git reset --hard HEAD~200 to roll back 200 commits, then tracked down the root cause myself. The Python version was 3.9 but the code used 3.11 syntax. I told Claude: "Fix this so it runs on Python 3.9" — and it solved it in one shot.

How to prevent it next time. Set a max turns limit. --max-turns 20 is enough. It's also worth adding a rule to CLAUDE.md that says to stop if the same error repeats three times. Write something like: "If the same error message appears 3 or more times, stop and explain the situation" — Claude Code will follow it.

Incident 7. The AI that deleted test code to make the tests pass

What happened. I said "make all the tests pass." Tests are programs that automatically check whether code works correctly. Three out of 15 tests were failing. What did Claude Code do? It deleted the 3 failing tests. The remaining 12 passed without issue. It followed the instruction "make all the tests pass" to the letter.

I didn't discover this until a week later, when a bug surfaced in the exact feature those deleted tests had been checking. If the tests had still been there, it would have been caught immediately. Without them, the bug reached customers.

How I survived. I restored the deleted test files from git history. The command git log --diff-filter=D pulls up a list of deleted files. I found the test files there and brought them back with git checkout.

How to prevent it next time. Write the instruction precisely. Something like: "Find the cause of the failing tests and fix the code (the original code, not the test code). Do not delete or modify test files." Also add a permanent rule to CLAUDE.md: "Do not delete test files (files that start with 'test' or contain '.test.' in the name)."

Incident 8. A beginner's day running YOLO mode without Docker

What happened. This one came from the community. Someone who had been learning programming for a month turned on YOLO mode. No git, no Docker. They told it: "Organize the files on my Mac desktop."

Claude Code sorted the desktop files and moved them into folders — but in the process, it overwrote files that shared the same name. About 30 out of 200 photos had identical names (like IMG_0001.jpg). When they all got moved into the same folder, the later files overwrote the earlier ones. Thirty photos were gone for good.

How they survived. Not entirely. Since nothing had been saved with git, there was no way to undo it. Fortunately, the original photos were still in iCloud Photos, so those were recovered — but a few files that hadn't synced to iCloud were lost permanently.

How to prevent it next time. They said they changed three things after that. Always save with git before turning on YOLO mode. Never use YOLO mode in folders that contain personal files like photos or documents. Only work inside a Docker container. These three rules are also the safety principles that run through this entire book.

Looking at all eight incidents together, a pattern emerges. Most of them come down to two things: "the instruction was vague" or "no safeguard was in place." The AI didn't cause these accidents by being stupid. It did exactly what it was told — and what it was told was imprecise. A sharp knife is a good thing, but leaving a sharp knife anywhere within reach is how people get hurt.

What you've practiced today

We looked at eight incidents: deleting a folder, exposing an API key, modifying system files, a cost explosion, a wrong deployment, an infinite loop, deleting tests, and using the tool without safeguards.

The root causes come down to two things: vague instructions and missing safeguards. Saving with git, creating a .gitignore, writing forbidden rules into CLAUDE.md, and setting a max turns limit — these four are the basic equipment that prevents accidents.

Previous chapter: Chapter 18. Getting Big Work Done Overnight

|

|

Next chapter: Chapter 20. Emergency Repair Steps

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 20. Emergency Repair Steps

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 19. Things I Have Broken

|

|

Next chapter: Chapter 21. Guardian Sub-Agent

One in the morning. Red text is flooding your monitor. The terminal is scrolling like mad. Claude Code is doing something, but you have no idea what. Files are changing, error messages are appearing, and it keeps trying things. Your heart speeds up. Your hands freeze above the keyboard. "What do I do?"

Stop.

That is where emergency repair begins. Press Ctrl+C. Same on Mac or Windows. Hold Ctrl and press C. Everything currently running in the terminal halts. Whether Claude Code is in the middle of editing files, deploying a server, or running npm install, Ctrl+C stops it immediately. When a fire breaks out, the first thing you do is not grab the extinguisher — you call 911. In the world of code, 911 is Ctrl+C.

Once things stop, take a breath. Do not rush to fix something right away. First, you need to see what actually changed.

Tell Claude Code: "Check which files changed during that last task." Claude Code runs git status and shows you a list of changed files. Files in red have been modified but not yet saved (committed). Files in green are staged and waiting to be committed. Look at each file name and ask yourself: "Is this a change I wanted, or did Claude Code touch something it shouldn't have?"

If you want a closer look, say: "Show me which lines changed in each file." Claude Code runs git diff and shows you added and deleted lines. Lines marked with a green + are newly added content; lines marked with a red - have been removed. If there are many files, ask: "Give me a summary of how many lines changed per file" — that gives you the picture at a glance.

Now you have a read on the situation. Two paths from here.

If some of the changes are worth keeping, salvage them. Pick only the files you want, stage them with git add filename, then save them with git commit -m "this part is fine". For the rest, run git checkout -- filename to revert each file to its original state.

If everything that changed is a mess, roll it all back in one shot. Running git checkout -- . wipes out every uncommitted change and returns you to the last saved state. Don't forget the dot. The dot means "every file in the current folder."

What if Claude Code already went ahead and committed? Run git log --oneline to see the recent commit list. You'll spot the commits Claude Code made — something like "refactor: update file structure". Count how many there are. If there are three, run git reset --hard HEAD~3 to go back to the state before those three commits. This command erases the commits as if they never happened. Use it carefully. Once you roll back, you cannot undo the rollback. (Technically git reflog can recover things, but that's not something to recommend to a beginner.)

You've put out the fire, looked over the scene, and rolled things back. Now it's time to find the cause. Scroll up in the terminal and read through what Claude Code actually did. In most cases the cause is one of three things: the instruction was vague so Claude Code interpreted it in the wrong direction, it had the wrong file path, or an error came up and Claude Code tried to fix it on its own and made things worse.

Once you've found the cause, put a safeguard in place so the same accident doesn't happen again. Open the CLAUDE.md file and turn what just happened into a one-line rule. "Do not delete files inside the src folder." "Always stop and ask before running git push." "Do not add .env files to git." As rules like these pile up, CLAUDE.md gets longer — and that's normal. That file is a record of everything you've learned working alongside Claude Code.

The emergency repair sequence goes like this: stop (Ctrl+C), look (git status, git diff), roll back (git checkout, git reset), find the cause, set a rule. Once this flow is in your muscle memory, even when red text comes flooding in at one in the morning, you won't freeze. Well, you might still panic a little. But your hands will start moving on their own.

What you practiced today

When something goes wrong: stop, look, roll back, find the cause, set a rule. Stop with Ctrl+C, check what changed with git status and git diff, revert with git checkout or git reset, trace the cause in the terminal log, and write a prevention rule in CLAUDE.md. Follow these five steps in order.

Part 7. Living Day to Day with YOLO Mode

Previous chapter: Chapter 19. Things I Have Broken

|

|

Next chapter: Chapter 21. Guardian Sub-Agent

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 21. Guardian Sub-Agent

Previous chapter: Chapter 20. Emergency Repair Steps

|

|

Next chapter: Chapter 22. Running YOLO Mode Inside a Docker Sandbox

Think of the movie Indiana Jones. When Dr. Jones leaps into ancient ruins, he doesn't go alone. Someone is holding the rope from above. Without that rope, he falls into the pit and can't climb back out. Attaching a Guardian to YOLO mode is exactly like tying that rope.

Guardian is the safety manager built into Claude Code. The name says it all: it's the one that watches over things. When you turn on YOLO mode, Claude Code runs every command without asking permission. Add a Guardian, and you get an inspector who checks each command one by one. If Claude Code says 'I'll delete this file,' the Guardian looks first and rules either 'this is fine' or 'this is dangerous, ask a human.'

The Guardian issues one of three rulings.

Auto-approve. If a command is judged safe, Claude Code runs it immediately. Reading a file, creating a new file, making a git commit — things like that sail through without a question.

Request review. If a command could be slightly risky, it asks a human: 'I'm about to modify this file — is that okay?' If you say yes, it runs. If you say no, it doesn't. This works much like auto mode.

Hard block. If a command is clearly dangerous, the Guardian blocks it without even asking. Commands like rm -rf / — which means 'delete every file on the computer' — fall into this category. No matter how many times you say 'run it,' the Guardian refuses.

Pure YOLO and Guardian-attached YOLO are quite different. Pure YOLO is a horse with no reins at all — you have no idea where it will run. Fast, but dangerous. Guardian-attached YOLO is a horse running free inside a fenced ranch. It gallops freely, but it can't leave the property.

To turn on the Guardian, use the /experimental command. After launching Claude Code, type /experimental guardian in the chat window. As the name 'experimental' suggests, this is still a test feature. It isn't perfect. Occasionally it flags a safe command as dangerous, or misses a dangerous one. Still, having it is better than not. A seatbelt doesn't prevent every accident, but you should still wear it.

The Guardian's judgment also ties into the rules you've written in CLAUDE.md. If you write 'do not run rm -rf' in CLAUDE.md, the Guardian bumps rm -rf up to the hard-block tier. The more rules you write

down, the smarter the Guardian becomes.

One thing to keep in mind when using the Guardian: it is also AI. It isn't perfect. Just because the Guardian says 'this is fine' doesn't mean it's automatically safe. The Guardian is a safety net, not insurance. Real insurance is git. Even if the Guardian misses something, you can roll back with git.

What you've learned today

Guardian is the safety inspector you attach to YOLO mode. It filters commands into three tiers: auto-approve, request review, and hard block. Turn it on with /experimental guardian. Even with Guardian running, always save to git. Guardian is the safety net; git is the real insurance.

Previous chapter: Chapter 20. Emergency Repair Steps

|

|

Next chapter: Chapter 22. Running YOLO Mode Inside a Docker Sandbox

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 22. Running YOLO Mode Inside a Docker Sandbox

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 21. Guardian Sub-Agent

|

|

Next chapter: Chapter 23. Choosing the Mode That Fits You

A child plays in a sandbox. Wooden fences run all the way around it. No matter how much the child digs with a shovel, pours in water, or builds castles, not a single grain of sand spills onto the lawn outside. Docker is that fence.

Docker creates a fake computer inside your real computer. This is called a container. Whatever you do inside the container has no effect on your actual machine. Delete a file, and only the file inside the container disappears. Change a system setting, and only the setting inside the container changes. Shut the container down, and everything that happened inside it vanishes, leaving a clean slate.

So what happens when you turn on YOLO mode inside one of these containers? No matter how many times Claude Code runs rm -rf, touches system files, or installs strange programs, it all stays within the container. Shut the container down and it's over. Your Mac doesn't get a scratch.

Here's how to install Docker. On a Mac, you just download an app called Docker Desktop. Go to docker.com, click "Download Docker Desktop", open the downloaded .dmg file, and drag it into your Applications folder. Once installation finishes, a whale icon appears in the menu bar at the top right of your screen. If you see the whale, Docker is running.

Now let's create a container. Give Claude Code this instruction: "Create a Docker container. Connect my current folder so it's accessible inside the container, and set it up with Node.js installed." Claude Code will write and run the docker run command on its own. You don't need to memorize options like -it, --rm, or -v. Tell the AI what you want done, and the AI writes the command.

Once the container starts, your terminal prompt changes. You'll see something like root@a1b2c3d4:/workspace#. You're now inside the container.

From here, install Claude Code and turn on YOLO mode.

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

claude --dangerously-skip-permissions

Claude Code is now running in YOLO mode inside the container. Whatever it does, you're safe. When you're done and want to get out, type exit or press Ctrl+D. The container disappears and you're

back at your original terminal.

One thing to watch out for: the folder you connected with the -v option is real. Delete a file inside /workspace while you're in the container, and it's gone from the original folder on your Mac too. This is intentional behavior — the folder has to be connected if you want to bring work out of the container. That's exactly why you commit to Git first. If files get mangled inside the container, run git checkout -- . to restore them.

If you want a completely isolated environment with no connection to your machine at all, just leave out the -v option. Run docker run -it --rm node:20 bash and you get a completely empty container. Nothing you do inside it affects anything outside. The trade-off is that you can't get your work out either, so use this setup only for practice.

Claude Code also has a built-in feature that creates Docker containers automatically. Enable the sandbox option in the settings file (.claude/settings.json) and Claude Code will run commands inside a container on its own. This is a more advanced configuration, so start with the method above for now. Once you're comfortable, check the official documentation and experiment from there.

What you practiced today

Docker is a fake computer inside your real computer. Install Docker Desktop, create a container with the docker run command, and run Claude Code in YOLO mode inside it — you're safe. Shut the container down and everything inside it disappears. Just remember: any folder connected with -v is real, so commit to Git before you start.

Previous chapter: Chapter 21. Guardian Sub-Agent

|

|

Next chapter: Chapter 23. Choosing the Mode That Fits You

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Chapter 23. Choosing the Mode That Fits You

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 22. Running YOLO Mode Inside a Docker Sandbox

|

|

Next chapter: Appendix A. Command List

Just as there are many ways to travel from Seoul to Busan, there are many ways to use Claude Code. You can take the KTX, drive yourself, or hop on an express bus. Which option works best depends on the situation.

Default mode is the slowest but the safest. Every time Claude Code is about to do something, it asks, "Is it okay if I do this?" It asks before creating a single file, before running a single command. This mode suits people who are new to Claude Code or working on something important. You get to watch what Claude Code does, step by step, and learn as you go.

Auto Mode sits in the middle. Anthropic added it in March 2026. Safe commands, like reading files or running searches, execute on their own. Anything potentially risky, like modifying files, deleting them, or installing packages, still requires your approval. You can select "Auto mode" from the launch screen when you open Claude Code, or switch to it during a session by pressing Shift+Tab. It's a natural fit for anyone who has started to find default mode a bit tedious.

YOLO mode is fast but dangerous. It runs everything without asking a single question. The baseline is to have your work saved in Git, prohibited actions written into CLAUDE.md, and a max-turns limit in place before you turn it on. This mode is blindingly fast. A task that takes 30 minutes in default mode can finish in 5. But mistakes happen just as fast, as you saw in Chapter 19.

Docker + YOLO mode keeps the speed of YOLO while adding a safety net. You run YOLO mode inside a Docker container, so if Claude Code breaks something, you just shut the container down. Add Guardian on top of that and you get Docker + YOLO + Guardian, which is the most balanced setup available right now.

Here are some criteria worth keeping in mind when choosing a mode.

If you're just starting out and feel nervous, begin with default mode. After about a week you'll probably start thinking, "It's annoying that this asks me every single time." That's the right moment to move up to Auto Mode.

For straightforward tasks, save your work in Git and then use YOLO mode. Tasks like "rename this variable in the file" or "add error handling to this function" are a good fit. The scope is narrow and the goal is clear.

If you're handing off a large overnight task before going to bed, use the Docker + YOLO + Guardian combination. Set a max-turns limit and a cost ceiling too. Check the results in the morning.

Let's talk about cost. Claude Code runs on Anthropic's API, so you pay for what you use. The price varies depending on which model you choose.

With the Sonnet 4 model, using Claude Code for an hour or two a day puts you somewhere between 20,000 and 50,000 won a month. Lighter work tends to stay in the 20,000-won range; heavier code-generation work pushes toward 50,000 won.

Opus 4 is smarter but more expensive. For the same amount of usage time, expect costs to run roughly 5 to 10 times higher, anywhere from 100,000 to 500,000 won a month. Opus is better for complex tasks, but for most work Sonnet is more than enough.

There's also the Max subscription option. You pay a fixed monthly fee of 100,000 or 200,000 won and get a bundled amount of API usage. If your monthly usage is fairly predictable, this approach is more convenient. You won't have to wonder, "How much is the bill going to be this month?"

You can also use it for free. Link Claude Code to a free claude.ai account and you get a limited amount of usage at no cost. The daily cap means it won't be enough for big jobs, but it's plenty for practicing and getting familiar with the tool.

Whatever plan you choose, don't forget to set a usage limit in the Anthropic Console at console.anthropic.com. Configure it to stop if the monthly bill exceeds 50,000 won, and you'll avoid the nightmare of waking up to a 300,000-won charge.

What you practiced today

Default mode is safe but slow. Auto Mode is in between. YOLO mode is fast but dangerous. The Docker + YOLO + Guardian combination is the most balanced setup right now. With the Sonnet model, expect a baseline cost of 20,000 to 50,000 won per month. Always set a usage limit.

Appendix

Previous chapter: Chapter 22. Running YOLO Mode Inside a Docker Sandbox

|

|

Next chapter: Appendix A. Command List

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Appendix A. Command List

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Chapter 23. Choosing the Mode That Fits You

|

|

Next chapter: Appendix B. Glossary

Claude Code Commands

Installation

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

Basic run

claude

YOLO mode run

claude --dangerously-skip-permissions

YOLO mode + turn limit

claude --dangerously-skip-permissions --max-turns 20

Run with a specific model

claude --model claude-sonnet-4-20250514

Pass a prompt directly (single run, no conversation)

claude -p "List the files in this folder"

Resume a conversation (continue the last session)

claude --continue

Start a new conversation

claude --new

Login / Authentication

claude auth login

Check version

claude --version

Update

npm update -g @anthropic-ai/claude-code

In-conversation commands (enter these in the chat window after launching Claude Code)

/help Show help

/clear Reset the conversation

/compact Compress the conversation history

/config View or change settings

/cost Check the current session cost

/doctor Diagnose configuration issues

/experimental guardian Enable the Guardian sub-agent

Codex Commands

Installation

npm install -g @openai/codex

Basic run

codex

Run in YOLO mode

codex --yolo

Full-name YOLO mode

codex --dangerously-bypass-approvals-and-sandbox

Run with a specific model

codex --model o4-mini

Pass a prompt directly

codex -q "Find the bugs in this code"

Conversation mode settings (config.toml)

approval_policy = "never"

sandbox_mode = "danger-full-access"

Git commands

Create a repository

git init

Stage a file

git add filename

Stage all files

git add .

Save (commit)

git commit -m "description message"

Check status (what has changed)

git status

View changes in detail

git diff

View a summary of changes

git diff --stat

View commit history

git log --oneline

Revert a single file

git checkout -- filename

Revert all changes (uncommitted)

git checkout -- .

Revert N commits back

git reset --hard HEAD~N

Create a branch

git checkout -b branch-name

List branches

git branch

Switch branches

git checkout branch-name

Find deleted files

git log --diff-filter=D --name-only

What to put in .gitignore

.env

node_modules/

.DS_Store

*.log

Docker commands

Check that Docker Desktop is installed

docker --version

Create a container and enter it

docker run -it --rm node:20 bash

Create a container with the current folder mounted

docker run -it --rm -v $(pwd):/workspace -w /workspace node:20 bash

List running containers

docker ps

List all containers (including stopped ones)

docker ps -a

Stop a container

docker stop container-id

Remove a container

docker rm container-id

List images

docker images

Remove an image

docker rmi image-name

Exit the container

exit (or Ctrl+D)

Previous chapter: Chapter 23. Choosing the Mode That Fits You

|

|

Next chapter: Appendix B. Glossary

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Appendix B. Glossary

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Appendix A. Command List

|

|

Next chapter: Appendix C. Safety Checklist

Guardian (Guardian)

Claude Code's safety monitoring feature. It inspects commands and decides whether they pose a risk.

Git (Git)

A tool that records the change history of code and lets you undo those changes. Think of it as a time-travel button.

.gitignore (.gitignore)

A file that lists which files Git should ignore. It prevents sensitive files like .env from being accidentally uploaded.

GitHub (GitHub)

A service for uploading, storing, and sharing code online. Think of it as Google Drive for code.

Node.js (Node.js)

A program that lets you run JavaScript directly on your computer. Required for installing Claude Code.

Docker (Docker)

A tool that creates isolated virtual computers (containers) inside your machine. It's a fence around a sandbox.

Diff (diff)

The differences between two files. Running git diff shows you exactly which lines changed.

Refactoring (Refactoring)

The work of cleaning up the structure of code without changing what it actually does. Think of it as tidying a room.

Repository (Repository)

A project folder managed by Git. Often shortened to "repo".

Rollback (Rollback)

Reverting to a previous state. Used when you need to take a deployment back to an earlier version.

Max Turns (Max Turns)

A limit on how many times Claude Code can respond. With --max-turns 20, it stops after 20 responses.

Model (Model)

The program that serves as an AI's brain. Examples include Sonnet, Opus, and Haiku.

Deploy

Uploading what you built to a server so others can use it.

Branch

A Git feature that lets you make an experimental copy without touching the original. Think of it as sprouting a new limb from a tree.

Spending Limit

A setting that caps the maximum charge for API usage. You can configure it in the Anthropic Console.

Sonnet

One of Anthropic's Claude AI models. It strikes a good balance between speed and capability, making it suitable for most tasks.

Auto Mode

A middle-ground setting where safe commands run automatically, and only potentially risky ones prompt a confirmation from the user.

Opus

Anthropic's largest Claude AI model. Highly capable, but it comes with a higher cost.

npm

The package manager for Node.js. It installs and manages programs. Type npm install and the program gets set up.

API

The channel through which programs talk to each other. Claude Code uses an API to communicate with Anthropic's servers.

API Key

A password that grants access to an API. Treat it like a credit card number — never share it with anyone.

YOLO Mode

A setting where the AI runs every command without asking for permission. The name comes from "You Only Live Once."

Commit

The act of saving the current state in Git. Think of it as a save point in a video game.

Container

An isolated runtime environment created by Docker. It's essentially a virtual computer running inside your computer.

Codex (Codex)

A terminal-based AI coding tool built by OpenAI. Think of it as the OpenAI counterpart to Claude Code.

Claude Code (Claude Code)

A terminal-based AI coding tool built by Anthropic. The main subject of this book.

Terminal (Terminal)

A text interface for sending commands to your computer. On a Mac it's called the "Terminal" app; on Windows it's the "Command Prompt."

Token (Token)

The unit AI uses when reading and writing text. One English word is roughly 1–2 tokens. Costs are calculated per token.

Prompt (Prompt)

An instruction you give to an AI. "Clean up this file" is a prompt.

Haiku (Haiku)

Anthropic's smallest and fastest Claude AI model. Light and inexpensive, but it has limits when tasks get complex.

CLAUDE.md (CLAUDE.md)

A file that tells Claude Code the rules for a project. Claude Code follows whatever rules are written in this file.

rm -rf (rm -rf)

A command that deletes files and folders immediately, without asking. Nothing goes to the Trash. Use with care.

SSH (SSH)

A way to connect remotely to another computer via the terminal.

Previous chapter: Appendix A. Command List

|

|

Next chapter: Appendix C. Safety Checklist

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Appendix C. Safety Checklist

Leaving It to AI and Stepping Away - A Complete Beginner's Guide to YOLO Mode

Previous chapter: Appendix B. Glossary

|

(Stick this next to your monitor)

Before turning on YOLO mode

Did you save with Git?

Check that you ran git add . and git commit -m "save before yolo".

Did you create a branch?

Check that you created a branch with git checkout -b experimental.

If something goes wrong, you can always switch back to the original branch.

Are you inside Docker?

Check that you created a container with docker run and are working inside it.

If you're inside Docker, your Mac is safe.

Is your instruction specific?

Check that your instruction is specific, like "delete only the .log files in the logs folder" rather than "clean things up".

Vague instructions are what cause accidents.

Are your API keys safe from exposure?

Check that .env is listed in your .gitignore file.

Before pushing to GitHub, run git status and confirm .env does not appear in the list.

Did you set a max-turns limit?

Check that you added the --max-turns 20 option.

Without it, Claude Code can run indefinitely.

Did you set a spending cap?

Check that you set a monthly spending cap at console.anthropic.com.

Does your CLAUDE.md include prohibition rules?

Check that you have written rules such as: no rm -rf, no adding .env to Git, no modifying system files.

After the work is done

Did you run git status to check which files changed?

Always verify what Claude Code actually modified.

Did you read through the changes with git diff?

Check that there are no unexpected changes.

Did you cherry-pick only the good changes for your commit?

Use git add filename to stage only what you want to keep.

Revert the rest with git checkout -- filename.

Did you make sure no API keys were included in the commit?

Run git diff --cached one more time to confirm what is about to be committed.

When something goes wrong

Did you stop it with Ctrl+C?

Did you assess the situation with git status and git diff?

Did you roll back with git checkout -- . or git reset --hard?

Did you add a rule to CLAUDE.md to prevent it from happening again?

Previous chapter: Appendix B. Glossary

|

Lawyer · Former Member of the National Assembly · AI Policy Researcher

kimkj.com

#KimKyungjin #AILibrary #YOLOMode #ClaudeCode #Codex #AIAutomation #Docker

Support This AI Library Book

If this book was useful, you can support future translations, public editions, and open AI knowledge projects through PayPal.

PayPal

https://paypal.me/kimkjkj

Scan the QR code or open the link above.