Learn 90% Of Pi Agent in Under 17 Minutes

16 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=8DT0HM8HIQ4
https://www.youtube.com/watch?v=8Dt0HM8HIq4

SUMMARY

Pi is a minimal terminal coding harness that emphasizes adaptability and simplicity, garnering significant popularity with over 45,500 stars on GitHub and 2.5 million weekly downloads on NPM. Unlike other coding tools, Pi focuses on a core set of functionalities, allowing users to customize and extend its capabilities according to their specific workflows.

- *Pi's core features include file reading, bash execution, file editing, writing, and session management.*
- *Users can enhance Pi's functionality through extensions, skills, and prompts, promoting a personalized coding experience.*
- *Installation is straightforward via a shell command, compatible with Linux, Mac, and Windows (with bash).*
- *Pi supports various LLM providers, allowing users to connect APIs like OpenAI, Anthropic, and others.*
- *Key commands for navigation and functionality include Control L for model selection, Control P for cycling models, and Control G for opening external editors.*
- *Sessions in Pi allow users to create branches in conversations, enabling easy navigation and recovery of previous states.*
- *Users can extend Pi's capabilities by modifying context, creating skills, and utilizing prompt templates for reusable commands.*
- *Pi lacks built-in features like plan mode and subagents, encouraging users to create their own workflows and templates as needed.*

This is Pi, the minimal terminal coding harness. While every other company is trying to release as many new features as possible, Pi has decided to go the opposite direction and give you the bare minimum. And this has been the result, getting over 45.5 thousand stars on GitHub and getting right now over 2 and 1/2 million weekly downloads on NPM. And even one of the most popular coding agents in the world right now, Open Claw, has decided to integrate with Pi to power its own AI agent capabilities. And the main reason for this hype comes down to Pi's core philosophy. Pi expects you to adapt it to your workflows and not the other way around. In this video, I'm going to go over everything that you need to know about Pi so you can decide for yourself if this is a harness that you can truly make your own. And we're going to split it up into five sections. Section one, the philosophy. Pi is a small core with programmable edges. Inside of the core of Pi, you have these five things and this make up what Pi is. You have the ability to read files, it can perform things with bash, it can edit files, it can write, and it also has sessions.

Now, automatically, when you see this list, you'll realize that there's one thing that Pi doesn't have that almost every other AI coding tool does have. There's no plan mode inside of Pi built in. That allows you to move a lot faster for by coding, which I think is what most people tend to do anyways. But you can download a plan mode extension and I'll show you how to do that later on in this video. But everything else outside of Pi is the models,

the agents.md, the skills, the prompts, and the extensions.

These are all the things that you can do in order to modify the core and you can modify the behavior. And this is all built around the idea that if you need something beside this, you're going to build it. You're going to shape the tool how you need it and you're going to share it with others. And that's what allows it to become a very personalized agent, an agent that only includes the the that you need and nothing that you don't want. Section two, setup. In order to install pie, you can just head over to pie.dev and copy this shell command.

And this will work if you're running Linux or Mac or WSL, which is the Windows subsystem for Linux on Windows. But if you want to use it inside of Windows, there's a few things that you need to know. Just make sure that you have a bash shell installed. You can install your get bash. You can find the information at pie.dev docs latest Windows, and it even gives you the link here for where you can install get for Windows. And it tells you the paths where it checks. I run WSL personally, I'll just copy that script and I'll hit enter on that, and now it starts to download. It tells us what it will run, which is just the npm install script.

We'll put yes for that. But now we have it installed, we'll just open up a new terminal window. Now we can just run it with pie, and there we go. We have it set up. Now that we have it installed, there are several ways that we can connect a model to it. We can use a subscription such as Claude Pro Max, which I believe that if Claude detects that it's a different provider other than them, they're going to bill it separately. But you can also use the chat GPT plus or pro plan, and you can use your GitHub co-pilot plan as well.

You can also connect an API key from providers like Anthropic, OpenAI, Grok, Mistral, xAI, Azure, Bedrock, or you can use something custom as well. Pie also supports the ability to run certain prompts in the command line like this using the dash provider and then what provider you want and the model, and then putting your prompt afterwards. You can see some other examples of using the models in the command line right here under the examples in the doc under usage, and you can see the list of providers that they support here as well. Anthropic, OpenAI, Deep Seek, really they have a lot of different options for you. Even the ability to run your open code zen here or your open code go plan. Or I also know open router is very popular, too. We can connect an LLM provider by running the {slash} login command and then selecting which one that we want to use. We'll use the subscription option and you can see here I already have it connected to my ChatGPT subscription. I'm going to rerun that just to show you the process. It'll take you to the OpenAI login and you'll just go ahead and login here. You may have to verify your identity and then after that you can just hit continue and there you go, authentication successful.

We get this message here saying logged in and the credentials were saved. We'll run a test by just typing the word hello. Hello, what is Pi? And looks like there we go, we have it working. For section three we're talking all about navigation. Here's some really helpful commands that are really worth memorizing and that you'll probably end up using about every day when you're working with Pi. Control L is going to allow you to choose the model. Control P will let you cycle the model. A shift tab would also change the thinking level and I won't go through each and every one of these but you can consider taking a screenshot or clicking the link in the description, joining our school community and I'm going to have this deck made for this video on a post inside

of the school community and you can feel free to download it and have this as a reference for you. I will spend a little bit of time of just showing you what each of those looks like though. For control L you can see now we're able to switch to our different models here. We can switch to a 5.1 if we wanted to do that or we can hit control P and now we're cycling through each one of these as you can see up here. We can bring it back up by doing control L again and going back to our GPT 5.5 model. We can do shift tab to change our thinking level as we can see right here. We'll leave that on medium for now. We can open up an external editor by doing control G which we don't have that configured but we'll go ahead and actually ask Pi to configure that for us. Now it tells us if we wanted to apply this to our current terminal we need to run this.

But how do we do that? Let's look at the docs. On the GitHub read me there's an editor section and if we look here it says for bash commands we can run a command prefixed with one next point and that's going to run it and send the output to the LLM, but if we do two exclamation points before the command, that's going to run it without sending the output to the LLM. In our case, we don't need this to go to the LLM, we'll do two exclamation points and hit enter.

And that's it. So, now we can go ahead and try our control G again, and it looks like that didn't work. We'll just go ahead and open up a new terminal and try it again. And now, if we go ahead and try it out by doing control G, Visual Studio Code opens up and we can go ahead and hit manage trust this window. Then we can type the rest of our prompt, such as "This is a test. I am thinking." And this would be the main reason that I would think that you would want to open up a separate editor because if we go ahead and save that and close it, then it puts whatever we're writing inside of here. So, if we wanted to think a little bit more about our prompt, or if we wanted to look at some stuff a little bit of a different way or something like that, like if we have some prompt library, that could be a bit easier than just pasting it all in here or typing it all out. I'll also show you the settings. If we do {slash} settings, then inside here, we can change whatever sort of settings that we'd like. They have auto compact, auto resize images, block images, scale commands, and they have about 21 of them. There's a lot of really helpful things in here. Even you can change the theme inside of here if there's something else that you'd like, which I don't know why you would choose light mode over dark mode on this, but it's there for you if you're interested.

There are four different modes that live inside of Pi. We have interactive, which is the full TUI experience, which I've been showing you. There's also the print {slash} JSON mode, which you could use for scripts, and you can add the {dash} JSON mode for event streams, and there's also [music] the RPC, which is for non-node integration, and the SDK, which allows you to embed Pi inside of your apps, such as in the open claw example. To show you how that looks, here's a readme that basically just covers Python best practices. So, here inside of the terminal, without running Pi first, we can run something like this, like the cat command, which would print out whatever this file is, but we're going to send that over to this command and ask it to summarize this text. What it did here is it sent this readme over to the model inside of Pi, and this is what we get back from it. And if we were to run that same command with the mode flag followed by JSON, we would get the JSON stream of it processing this. Down here at the bottom, it shows you what folder you're currently working in. It shows you how much money that you're spending and what percentage of the context window that you've currently filled up.

And over here on the right, it tells you the model, the thinking level, and also what skills you currently have available. This find skills is from my open code, and this doesn't ship with Pi. All right, section four. This is all about branching and recovering. Really, we're going to be talking about how sessions work, how conversations work, how to pick up where we left off, how to undo things. So, let's get into that. Sessions are just trees that we can go back and create branches. So, you think this would be like our first message that we sent. This is our second. This would be our third. Maybe we get to D, and then we think, "Oh, I want to go back to B, but instead, I want to ask a different question from there." That's what we can do whenever we run the slash tree command. That allows us to jump to an earlier message, edit it, and then resubmit it. And Pi keeps the alternate path in the same session file. We can also resume a previous session with a dash C flag, or we can do the R flag. We can also create a new file from an old prompt using the slash fork command or the clone. Let's hop over into Pi and I'll show you how all this looks. Here, I've asked Pi to make a basic Flask server. Here, we can see it's thinking as well. And what steps it ran, what did it make. Then I asked it to make a new path called hello, which is just a page that says hello. So, we got that to happen as well. And if we inspect the code, we can see all those paths currently exist right here. But now, let's go ahead and stop this session by running control C twice. We'll clear it.

We can run Pi again, but this time just hit {dash} C, and we can see we get all the same session loaded back up. And if we do pie {dash} R, we get all of our sessions back up. We can sort through them either by current folder or all by doing the tab key. We can sort them with control S. You can see over here the sorting either threaded, recent, or fuzzy. And we can even delete one by coming down to it and then hitting the control D and hitting enter to confirm.

And now that session is deleted. Let's go back to our previous conversation that we were just working on. And then inside of here, we're going to run {slash} fork. Now we can select which message that we want to go back to. Let's say we want to go back to our second message. And now instead of asking it to make a new path called hello, let's say we want it to make a path called goodbye. It never got rid of our hello path. It just added the goodbye path. So what we're doing whenever we run the fork is we're not undoing the code changes. We're just changing the conversation history. So if instead of doing a fork, we wanted to instead just pick up right from where we are, just duplicate the current session at the current point, we would use the clone command for that. With that example in mind, we now understand that undo means two different things. There are prompt undos, and there are file undos. The {slash} tree command lets you branch off from an earlier message, right? But if you want a file undo, if it changes something and you want to undo it, you should use Git or an extension for checkpoints or something like that. Because pie doesn't support undoing file changes out of the box. But before a run, you definitely want to go ahead and make sure that you have everything saved inside of Git if you want to go back. The first way that you can extend pie's capability is modifying the context, right? You can create an append system.md, which is going to add on to the default prompt. You can create an agents.md or cloud.md, and this is going to add instructions to how you want it to behave. You can also add skills to your agent, and then finally, whenever the default prompt is loading, it's also going to put the current date and whatever current directory that you're working at. And if there are certain things that you want to modify globally, you can go to the dot pi folder, and then under agent, then you can put your agents.md. For instance, if you wanted to do the project tree, then you just do it in that folder. And you can even replace the default system prompt by putting a system.md or append to it with an append system. md. And you can read more about how all this works inside

of the documentation under usage.

It goes into more detail on that. And if you're really curious, you can head over to the system prompt.ts on GitHub, and you can even look at how the default system prompt is made. Let's talk briefly a little bit about how skills and prompt templates help us to extend our agent and how they solve different problems. Whenever we think about skills, this is going to be used for reusable capability packaging. When you think about workflows, setup, scripts, reference, etc. There are a lot of really popular skills out there. It would be a lot easier to understand how a skill works if I show you a real example of one. So, this is an example of skill planning with files. I recently made a video on it. It teaches your agent how it can work similar to how Manis works. And they all usually come with some kind of an install script. I don't want to install this globally, so I'm just going to copy everything before the G flag. Then inside cursor, we can run that. Then it asks us what agent that we're installing this for. We'll select pi, and we'll select the project.

Now we have this agents folder here, and I'll make that a bit bigger. But inside of here, this is what a skill looks like. Each skill is a folder that must have a skill.md, and that tells it how it should work. Now, this is a very complicated skill, and there's a lot of fields here. But really, when it comes down to it, the only things that are required for a skill are just the name and description. And everything up here, in between these three dashes, we call the front matter, which is just a key and a value. And everything underneath that little section is just a prompt. It might be referencing these different files, it might be referencing these folders, and it's just telling you how to perform a certain task. And a simplified skill is really just a prompt template, and they're used from saved prompts that expand from slash commands.

So, you run a slash command and the model just sees the full prompt. So, in this instance, you'll have in your .pi folder a prompts folder, and then you can put whatever sorts of commands that you want. So, we end up with something that looks like this. I made a sample prompt here that just says hello there, and I reloaded Pi, and now we can see here at the top what prompts are available to us. And we can run them with that slash hello, and it just puts the entire prompt there for us. There are really a lot of ways that you can extend Pi. You can make a different tools, which the LLM can then call. You can make commands, which we talked about. You can even ask Pi to make things that will intercept the events, which will allow you to intercept turns and tools. You can also modify state, which is all about session persistence and how it handles that. You can add all these different providers, which really just changes the model. A different model means different behavior, and you can even ask it to change different things about the UI. Really, Pi is all up to your imagination and what you would like to make it to be. Or if you're lazy and you don't want to build anything and you just want to see what other people have built, you can check out the official examples on the GitHub repo for the project, or you can head over to their package library at pi.dev/packages.

pi.dev/packages is really extensive, and you can find a lot of really cool stuff on here, like context mode. This is an MCP plugin that saves 98% of your context [music] window, Pi subagents, Pi MCP adapter, Pi web search. Yeah, a lot of really cool things on here. Here are some of the things that Pi leaves out that are kind of common in other AI coding tools. There's no built-in MCP, there's no subagents, there's no permission pop-ups, there's no background bash, there's no to-do's, there's no plan mode. The main rule is if you need a workflow every

day, make it a template, a skill, an extension, or a package. And you have these seven layers that you can play with. You can change the defaults by adjusting your settings.json.

You can adjust the project rules by adjusting the agents.md. You can replace the agent identity by changing the system.md, which is going to adjust the default system prompt. You can repeat a prompt with a prompt template like we talked about, or add a capability with a skill, change behavior with an extension, and you can share what you made by making it into a package, which I didn't touch much on in this video, the settings.json, but you can just head over to the settings a section in the documentation to check out more of that if you're interested in it. And you can also check out the documentation on how to build your own packages there as well. So, that should give you a very in-depth understanding of how to use Pi.

If you want to get this deck that I used in the video, which you can use for your notes and later reference, click the link in the description and join my school community. It's completely free to join, and I'll put a post on here and I'll have my deck included in it. You can see here we just hit 272 members, and we have people from literally all over the world who are a part of our community. So, go ahead and click the link in the description and check that out if you'd like. Other than that, I just want to thank you so much for watching this video. If you liked it, please give it a like. Please subscribe.

And if you enjoyed this video, you may enjoy checking out some of my other tech demos right over here.