Claude Code Won using Simplicity: Here's How You can too

33 MIN · YOUTUBE · HTTPS://WWW.YOUTUBE.COM/WATCH?V=BVLRXAPCD-O
https://www.youtube.com/watch?v=bvlRxaPCd-o

SUMMARY

The video addresses common misconceptions about setting up context operating systems for go-to-market strategies, emphasizing the importance of understanding basic computer science principles rather than focusing solely on specific tools. By framing the discussion around first principles, the speaker aims to clarify how file systems and cloud code function, ultimately empowering users to make better decisions regarding their tools and organization.

- *Cloud code is effective due to its simplicity and minimal moving parts.*
- *Understanding file systems is crucial as applications often hide their complexities from users.*
- *The shift from app-focused to file-focused thinking is essential for leveraging cloud code effectively.*
- *Emergent properties arise when simple components work together, creating capabilities greater than their individual parts.*
- *Directory navigation, keyword search, and file reading are foundational tools that enhance the functionality of cloud code.*
- *Clear organization of files is critical for optimizing interactions with cloud code and ensuring clarity in operations.*
- *The speaker encourages users to prioritize file organization over the choice of specific tools like Cursor, Obsidian, or VS Code.*
- *The video aims to provide a broader understanding of how to effectively utilize cloud code for various applications.*

I'm going to go over uh and address the common misconceptions and like FAQs uh I run into consistently setting up uh in building out these context operating systems for go to market. Now instead of addressing these like one by one around like what tool should I use? Should I use obsidian or cursor or uh stuff like that? I'm going to attack it from like more of a first principles perspective on just like a quick reminder on like how computers work. And the reason I'm doing that is the my core belief is that cloud code is so good. It's so effective. Whatever you want to say because it's simple, right?

Genius has the least moving parts. Uh and I'll get more into that in a second. But this space is just a reminder on like how file systems work, why they work the way they do, how cloud code works with them, and why this actually creates the like emergent dynamics we see. So, we're going to go over that and by the end of this video, you should have like a much better understanding of basic computer science in a way that is hopefully like applicable and valuable to you using cloud code for your own needs and go to market work. And it should be more intuitive to you because rather than doing like point by point question by question, I want to give you like the broad understanding and empower you that way. Um, I think that's a better solution or better approach

overall. So, let's get into it. What I'm seeing over and over again is there's three things.

There is confusion around what tool to pick, why to pick it, how in regard to do you use cursor, do you use uh VS Code, do you use Obsidian? Um, how do you structure your context operating system? Why does it work the way it does? And these are good questions. Uh, and they're fair questions. The thing I think is what the the thing I am sorry phrasing what's actually going on is I think there are just broad first principle understanding or that would alleviate all of those questions and concerns. Um, and the core thing I'm getting at is our apps. So, your LinkedIn Clay, iMessage, um, every single app you use pretty much on a day-to-day basis hides how computers actually work from you. They hide the file system. They hide um, that's pretty much the main one. They hide the file system from you.

um you don't see why something why why uh a file is saved one one place versus another um and how code is run and all these other pieces of complexity that when in the preAI phase or pre AI era that hiding that that abstraction was a net benefit to the user because you only needed to know how the interface worked um and what to put in there, you didn't need to know anything of what it was doing on the back back end. But um the the opposite case is not true. So because plain English is now code, you actually really need to know how a file system works. You need to know um how to execute that code, right? That program, which is now plain English, mind you. It's not like Python or some specific syntax that is um you have to learn. Plain English is code and it's now more important than ever to understand the basics of how an operating system um works because you are now directly interfacing with it.

That is all the value that is not all value that is where a large portion of the value from cloud code is derived from. Um, and it's more simple than you think. And I know it's like, oh great, another engineer telling us um telling us the thing they spend hours and hours and hours a day on is easier than uh easier than we think. It's not the case. But like it, let's start with this core axiom.

Uh, genius has the least moving parts. Simple is most effective. So those two things right the it's really easy to over complicate a system um it's very hard to make a elegant simple and effective system that solves through the real uh functionality or real functionality that creates value. Let's say that so more often than not software engineering is an iterative process of adding a bunch of stuff and then removing the things that you don't need and honing down. Like a good analogy that I like is like you know how they they they sand a surfboard down. The surfboard is inside that block of wood or fiberglass I guess whatever and you're moving pieces iteratively.

Um and the point I say that the reason I say that is cloud code is so effective because it is very very simple. You look at the kind of trend of AI development or AI engineering because we're not talking about machine learning here. talking about large language model driven agents and other stuff like that. When it first came on, when the thing the scene first exploded, it was vector databases and lang chain and like all this shitty shitty abstraction. Um, and abstraction I mean you're adding new things to the system and it's getting more and more complex um to try and solve for your outcome. But we haven't realized that those things actually added far more work than the value they brought more often than not.

Um, cloud code is so effective because there's like three big portions of it. It can just read and write from your file system. It can run commands and it operates uh it operates locally. So, and the thing the next word that creates what's called emergent properties. So, emergent properties is 2 plus 2 equals 6. You get something that's greater than the sum of its parts. The like visual analogy of this is you see a flock of birds flying around. That is an immersion property of you having more than one bird, right? Birds flock together and it creates something. It creates a flock of birds. Um having all of those tools and that functionality locally for cloud code creates greater than sandwich parts capabilities which is by and large that planning which markdown flowers are now code. So to make that tangible and specific and understandable and to give you that actual first principle understanding today I'm going to just break down bisque file system how everything fits together and talk about the different layers of abstraction and why why it actually doesn't matter if your terminal is here inside of um cursor which I know we're looking at a lot right now or if it's floating off in space in a different window. We're going to go through all of that and work through like the basics computer science which we all learned um at least if you're not like someone who grew up totally in the um app era where that that all that was hidden from you.

You all learned how file systems worked when you had to go save a Word document. I'm talking about my use case now. um or my use case, Jesus Christ, my experience of being a kid in the early 2000s or in the 2000s of I want to save this file on uh the C drive. Where's the C drive? Oh, I have to go here and here's my folder for me. I'm going to save it there. Right? That it's like that level stuff.

So, we're going to start there. And uh I hopefully that makes it a little more clear how cloud code works and how to tool it and point it in the right direction to actually solve your go to market or your development or any use case you're actually using it for right now. Okay, so the first thing is what is an what is a file system? We're going to close code and now we're getting into the terminal. Scary, I know, super scary being facicious. We're going to hit clear. And now we are in a file system, right? We are in a terminal that has a file system. So I'll actually bump this out or actually we'll do PowerShell. We'll pull this up. And this will be Windows specific, but the principles are the true true across all all modern operating systems. Windows, Linux, Mac OS are all file system based systems. Um so you see here this is at the root layer of my uh user in this case. So when I open a ter a powershell at my root layer it's going to say only two things and this is called a path.

So it's the path to where you currently are in your system. Now if I do directory there's gonna be a bunch of stuff in here. Um, a lot of this is actually junk that I should remove. Claude has been throwing around much files. But you see here it is by and large um just like configuration files and other stuff like that. Um, in fact, it's not even like where you would think where documents are or anything else like that because Windows gives you another layer of traction in your file explorer. Um, so you can navigate around without having to see this. So this looks flat, right? I'm going to exit out of this because there's too many things to list out properly. But if we go into this context operating system example, so I a while ago I put out a context OS quick start guide and I used like Jurassic Park as the fake business um to work through. If we do director here and that just stands for directory, you're going to see.cloud 00 foundation knowledge base. So the context OSI design you have your operational docs right with like your 00 foundation then you'd have like a 00 customer or 01 customer 02 u messaging whatever your uh setup is that's kind of dependent on what you need and how often you access specific files.

Um and then a knowledge base which like a deep knowledge graph like an actual imaginary graph. Um that's what the knowledge base is for. So you see here we're looking at this directory like this. If I hit tree, you see now we see this like branching tree system. So this is what cloud code is navigating up and down, right? That's going to go to, hey, Jacob told me to go to check the messaging. So it's going to use something called uh there's multiple different ways to get there, but let's say it's going to do uh list. It's going to get see basically this exact thing.

It goes, "Oh, okay. The positioning is in the foundation. I'm going to go in there and I'm going to do I'm going to mimic this right now." So, CD is change directory the 00 foundation, right? And now I'm in there and I'm going to do d I'm going to do tree actually so you can see and I see tree. Okay. Messaging right here. So, I'm going to do CD positioning or I'm going to go messaging my own example. Uh, CD messaging and there's nothing in there.

So, if we go into here, right? Nothing in there. Nothing in there. Nothing in there. Uh we'll go actually to our knowledge base. So we'll go up. That is how you move up a directory. CD cd knowledge base. And then we're going to do dur. And then we do cd business. I think there's examples in there, right? Yeah. And then we're going to do list. Oh, we'll do react. What is dare? Um list is a different command that does the same thing as dare. Um so this is how cloud code is navigating around your directory at the basic level. So if we go to here uh where is my Figma right we're going to do first one is um directory navigation.

So that was super manual, right? That took me like a good amount of time relative to what I was trying to do just to move around three files or three directory levels in a terminal. Because clog code can generate all of this all these commands on the fly extraordinarily quickly. it suddenly becomes much more valuable um because it can move up and down and go yep uh across these a thousand files and go one two three four five oh and I should use uh what we're going to show next like a keyword search or use GP and actually I can just skip all the rest of these 950 files and just go right to this next section here that is like this emergent agentic um agentic lexical search is what it's called um where it's combining ing the navigation directory or excuse me yeah uh no directory navigation I flipped my words there um that capability along with uh keyword search so GP that's the next one and reading a file which glob those are the tools that cloud code has out of the box that create this emergent dynamic so now we're going to we're going to get into GP Next.

Okay, so I put in this very leading prompt um or I'm basically telling it the answer. This is like a bad prompt um when you're actually trying to find like net novel information, but I know what I'm looking for here. So, I'm just going to give it this leading prompt. Um and the reason I'm doing this in cloud code is I don't want to have to write all my grip commands myself because uh that takes too much time. And that's I think one of the things that is an underrated um explanation of why cloud code is so valuable. Um it makes it so if you had infinite time to search across your entire knowledge base uh your transcripts, internal working documents, CRM data sets, everything like that.

What questions would you now be able to ask that you weren't able to ask before? That's basically what cloud code is giving you because of this emergent dynamic. Um, so you can find files by pattern names, right? Oh

yeah, excuse me. I I put I think I said globus read files, but um it's a different type of search. So hey, I'm correcting myself. There you go. Um, prep content discovery immersion patterns. Exactly. So you can do like the way I visualize it or I yeah the way I personally visualize it is imagine um drilling for oil, right? They drill like a bunch of pilot holes. So they're like no oil here onto next grid. No oil here. Oh, there's a little more oil here. And you can like iterate down to eventually getting to like striking oil. And that's what this capability these tools all come together to create in it's like machine learning in uh or it's like gradient in in machine learning if you want like a more complex um analogy but I'm trying to simplify here. Yeah, triangulation triangulation that's exactly the thing.

So now I'm gonna say to explain uh to make uh understandable Um show how simple functionality superior to uh overengineered complexity.

Again, leading prompt, not how it actually interface with this is not how I actually interface with my context OS if I'm trying to find um net new insight. So, this is a good example here. It knows this is a Jurassic Park context OS because if we go down to this right here, this MD file, this is always injected into cloud code when it starts up. So this is like your governing cananical rule set. It tells it where is and how to work together or how to interface with that um given app or context OS, whatever you're telling it to look against. This is always given to cloud. So you should make sure in your own system that this is really really indepth and well thought out because an error in here compounds across multiple sessions and you either you either have clarity in this uh you either have clarity in this file which guides all future sessions or you have errors in this like you have basically ambiguity we'll call it clarity versus ambiguity and you either compound positively or compound negatively.

So, going back to letting it run its um the search. I'm doing this live, by the way. I'm just reading this for the first time myself. Here we go. So, that's how new to search against um Raptor. Do control O. Now, we're seeing the um searches. So, this hides it from you, right? Pattern raptor. And it's looking, it's telling it, all right, I want to find this keyword in this area and tells it, hey, this is where I found this line. And it gets the area around it. In this case, it was telling it like, when you find Raptor, give me the lines before it and the lines after it. At least that's what I think it's doing right now. Yeah, exactly. It's saying, hey, in this MD file, Raptor is I don't know where it is. Somewhere in here.

Yeah, right there. So all of these are then being injected into the context window and it's far more token efficient and they can iterate much faster across the search giving it that emerging capability to almost to to almost act like it's thinking like a human being even though it's not right. It's doing this sophisticated information lookup. Um exactly. Yeah. sort of built from simple parts that combine this and then bash is even more valuable um because previously if you wanted to run a Python script or some sort of uh let's say let's talk about Python script let's say you have a you want to do data analysis on a data set uh and you need to use pandas you would often doing that in Jupyter notebooks which is basically you have Python and you can do like stuff down here uh like a Python runtime and then you can do other stuff separately so you have like multiple different areas to run separate apps um or doing it locally. The point being like there's a lot more overhead yourself now because it's inside of the context window in here. Um the agent can write and run that command

for you and then push out that information to your local file system.

So it's just persisted outside of the context window. And that is what allows any context operating system to become this compounding intelligence asset because once you solve a problem once you can say hey claude I want to save how we solved this uh persona uh analysis let's save this as a skill and they then get pushed up into skills right here which is again our markdown file because plan English is now code. So do NY and Y and I think this would be the last thing we cover. Um so let's dive into this.

You see here O right ls raw context we're pulling in it's listing the directory and it's being able to see everything in here. So an important distinction is in the syntax here. This is basically what I just showed you with commit change directory. That's CD and tree. Except because cloud code is like its own separate application, it's writing and running stuff using bash which is using um Unix syntax.

So same fundamental principles just a little bit different. I had to let someone in. Um, look at this. So, it's now actually kind of messing up because I didn't give it tight enough instructions to say, "Hey, don't get don't try and go and do git, which git is version control software." Um it's basically how we take a snapshot of an application at a point in time and then you can have multiple snapshots to make sure that if you make a change to something you can just roll it back. Um or you can branch applications into different versions and that allows um asynchronous that is what enables asynchronous collaboration.

So I'm going say no not relevant uh say on task. Um so this is what I mean by either ambiguity or clarity compounds. Uh this is not or because I gave it a pretty actually ambiguous yeah ambiguous maybe. um prompt that because I gave it a prompt without like very clear constraints uh it went off and it got sidetracked.

When you are like engineering these context architectures you have to be like far more specific than you think you do otherwise you get stuff like this. Um, but I think I think I've gotten my point across. At least I hope I have. Let me check my notes. Yeah, let me go let me go back to this. Basically, um, directory navigation and we do prep glob. Oops.

And finally we'll say like read file. Basically it's not fully accurate but so these are like different layers of tooling that all build on one another to create something bigger than the sum of its parts. All of these fit together to create something called agentic search. And we'll say that's bigger. Whatever. I know this is bad visual. Um, and this all depends on if we go to here. Oh, see. Look at this. Now we're getting an advanced um search pattern. So, it was able to do enough pull in enough information, pull enough context into this window from this cloud. MD and all the other files it searched across here to say, hey, this would be a good demo. And what is it demoing? It's going to use something.

It's going to GP. And then this pattern right here is basically a pattern to search for emails. I know it doesn't look like it, but this is a this is what I mean by you get into syntax. So, I could have said, "Hey, find me emails." And it would translate that into this syntax pattern to find um to find those emails and answer that question. Who's answered the mo who is mentioned the most in emails and and and I can just do similar commands. So then it

can go search on its own.

And there you go. See, it's actually printed out and an an answer here, right? By name Smoldon is the most Hammond Nedri. Now, this doesn't look right to me. So, this is where it also comes down to you fundamentally. Comes down to someone who like has an understanding of the system. There's no way this guy's name will do is mentioned 711 times. Um, and it's actually caught that itself. Uh but this is just to show like the systems are not perfect. It's going to make mistakes. But that's why we have guard rails and attribution systems as a part of this as a part of this.

And then it ran another one and fixed it. Oh, that was actually uh artifacting error like a a UI error. So 71 times. That makes more sense. Um I think it's funny that's actually like cosplay like yeah that's a red flag. Um here we go. This is a good breakdown. Read file filter order count rank. So this is how it's navigating across this entire context operating system and creates greater than the sum of parts capabilities.

So going back to my crappy uh diagram here, right? This these function out these tools read GP glob direct navigation creating this emergent emergent agentic search right it depends on this fundamental core functionality of computers that has been around since we created digital computers. Um, it's actually more simple than all of the crazy you need a vector database, you need lang chain, all all of that stuff. It's more simple and thereby more effective because there's less moving parts for things to break, for things not to work. Um, it is just that simple genius.

So then finally getting back to the original um thing I said I was going to cover around should I use cursor or obsidian or VS code or anything like that. I want to reframe it. It's not a it's not a tool question fundamentally doesn't matter um because instead of apps you now the primary working unit that is now important is files. It's files over apps. So, if I take this, I'm going to copy this and steal from myself. It's going to be this is our file system.

And by the way, this is how it's always been. It's just that, you know, we've had, you know, our operating system, uh, like, you know, like a Mac OS or Windows that hides all of this from us. It hides this like fundamental, you know, terminal from us because it's easier for us to click around with UI um and anything like that. You look at old computers, it was all terminal. The huge innovation was the operating system which made the barrier to entry much lower. It made it so more people could use computers without actually having to understand all of the complexity of a file system.

But now we are moving away from so that's the next layer would be like your applications which had even more layers on top of them or had even uh what's the word had uh which was another layer abstraction. So we think we're used to working at the application layer. So we're like oh what app should I use? The most important layer now is actually the file system because now our application cloud code is going to get rid of let's say I'm going to say this is cursor obsidian uh vs code whatever by the way so cursor and obsidian are both called not cursor obsidian cursor and vs code are called integrated development environments that's what those that like class of application is called they are specialized or they're they're different than a traditional like consumer application

because they're meant to kind of go work through you know all of the stuff here around I'm waving I guess um this like unnecessary abstraction when you're doing development work but now because plain English is now code this direct connection right to our file system we're just using cloud code agentic let's go search to make doing all of this stuff far more easy. Um we're not actually bypassing the operating system. This is maybe not the best analogy. Um you want to get pedantic. It's like this and then the kernel and then kernel is like what you know firmware and then finally like the point I'm trying to make is like there's layers all the way from the most software that you interact with today and all the way down to like the actual computer CPU running zeros and ones.

There's like layers of abstraction and each layer hides the stuff below it. So you only have to work with that new layer. But we are removing unnecessary layers of abstraction now because we've like moved that complexity of like running and writing those scripts to cloud code and other agents that can read and write on our files on our file systems. So the fundamental question of why do I what tool to pick, don't think about that. Think about the the question of like how do I want to organize my files so it's optimized for my agents and for myself. It has to be intuitive for your own understanding um to work in this file system. That's how all of this works. Um, yeah. I think that is pretty much everything I wanted to cover. This is a like more impromptu video because I was seeing these questions come up over and over again. And they're good questions.

I understand where they're coming from. I just wanted to uh almost invert the question. And I think that's a much more valuable and useful place to come from. And it's also more intuitive once you remember like, hey, like we were all taught this stuff as kids or at least I hope you were. um if you're not like hope there's a good crash course. Um and we're just removing stuff that is no longer valable or useful to our actual needs. So yeah, I hope you have I hope you learned something from this. I hope it was valuable and I hope you have a uh a great rest of your day. Bye.