**EPISODE 1153**

[INTRODUCTION]

**[00:00:00] JM:** For all the advances in software development over the years, one area that has been minimally improved is the terminal. Typing commands into a black text interface seems antiquated compared to the dynamic flashy interfaces available in web browsers and modern desktop applications like Slack. Fig is a visual terminal assistant with the goal of changing that. Fig sits next to the developers normal terminal and enhances the terminal experience. The founders of fig Brendan Falk and Matt Schrage join the show today to discuss how Fig works and why it's useful to having an enhanced terminal.

[INTERVIEW]

**[00:00:42] JM:** Guys, welcome to the show.

**[00:00:44] MS:** Thanks for having us.

**[00:00:45] BF:** Great to be here.

**[00:00:47] JM:** Every developer is familiar with a terminal. What is a terminal used for?

**[00:00:52] MS:** I mean, the terminal is one of the few tools that is useful for developers regardless of what your stack is, what language you're working on. It's kind of the lowest common denominator. And so, again, when Brendan and I have been of coding, we use it for everything from running tests, from deploying code, managing orchestration. Really, the full gamut can be done in the terminal.

**[00:01:16] JM:** And how much is the terminal advanced in terms of sophistication over the last, let's say, 20 years?

**[00:01:23] BF:** Not much at all. Matt definitely knows the history, but you look at the terminal in the movies in the 1970s, it hasn't changed very much.

 

**[00:01:35] MS:** Yeah. I mean, I think the last big kind of update to it was the BT100, which is like what gave you kind of the ANSI escape codes and lets you move the cursor around. And people have innovated a little bit with newer shells. But fundamentally, you're using the same technology that the earliest computers were running.

**[00:01:53] JM:** And you guys work on Fig. Explain what Fig is.

**[00:01:57] BF:** So Fig makes developers more productive in the terminal. General idea is we help developers build apps and shortcuts that streamline their workflows in the terminal. And then we let them share these apps with their team or the community.

**[00:02:13] JM:** What's an example of an app?

**[00:02:16] BF:** So curl is a command that people use all the time when you want to make an HTTP request. You always forget the flags for curl. You always forget how to escape certain characters, and you end up by the defaulting to a tool like Postman, which is really heavy application on your desktop. Or you Google like what are the exact flags for curl. And it sort of takes you out of the terminal. And all you really want to do is make a quick HTTP request.

So what we have done is we sort of took the UI of an app like Postman. We brought it into your terminal and now you can just type Fig curl, and all old keyboard-driven, you can build an entire code request and then use a keyboard shortcut to run it in your terminal and you don't have to worry about memorizing the flags, memorizing the escape codes, that sort of stuff. So that's just one example. But we have apps from navigating your file directory all the way up to configuring sort of Kubernetes deployments and orchestration or – Well, we've got so much.

**[00:03:12] MS:** Like running Postgres servers, everything in between. FFmpeg commands.

**[00:03:16] BF:** Yeah, and anyone can build one of these.

**[00:03:20] JM:** Give a specific example of a Fig app that people might build.

**[00:03:25] MS:** So again, going back, we started with this idea assuming that novices would be

the one to get most value out of it, because they're the ones who may be haven't been using the shells since they were five-years-old. They don't know all the kind of more arcane UNIX commands. But when we launched over the summer, we discover that there's a huge class of developers who find themselves in the terminal for work. Again, maybe they know Kubernetes slightly or they know Docker and they had some of the commands memorized. But there is a ton of overhead and there is all these processes you don't just do every day.

And so those are the kinds of apps that people are building for themselves. It's taking a process that they may be running the terminal a couple times a day or a couple times a week and then wrapping it in a UI so that they can share with their team and share it with members who are maybe less comfortable in the kind of text-only environment.

**[00:04:17] BF:** So more specifically, an example would be something like – Is it DBT?

**[00:04:22] MS:** Yeah, DBT.

**[00:04:22] BF:** So we're working with a team. We won't say to the company name, but they're big analytics team and they use DBT. We've never used DBT before. But they just had this really long like string of commands that need to put in every single time they use this thing. And they're data scientists, they don't want to be doing all of this complicated stuff. And so what we've been enabled to build is just a really short sharp UI that lets them put in some parameters. It actually constructs the command sequence that they need to run for them.

Almost like an easy way of building a shell script essentially, and they can click a button and it runs it. And then they're back to doing what they do best, which is analyzing data not messing around in the terminal.

**[00:05:05] JM:** And for those who have not seen Fig, can you describe in more detail what the UI looks like?

**[00:05:12] MS:** Yeah. Basically, Fig isn't trying to replace your terminal. We want to be a layer on top of it that you can use when you want to when it makes sense, but then you can drop down to a lower level of abstraction when that's the right choice. So what Fig does is it connects

to your existing terminal as a sidebar or kind of a window next to the terminal. And when you run a Fig command, it'll open up a UI there, and then that fills in the information and lets you have this visual window into a terminal command.

**[00:05:43] BF:** A crazy analogy is, if you remember, Clippy in Microsoft Word, that sort of you could drag it around and it was always there. It was always popping up suggestions. Like we hesitate to use that example, because everyone is sort of – Clippy was there, but it wasn't actually that useful. So we have that sort of idea of you have your core application, but then wear this productivity layer on top that actually lets you enrich and extend what you're already doing and just make you much faster.

**[00:06:10] JM:** So there're different plug-ins that fit into the Fig module. Can you talk about who writes the plug-ins and what they look like?

**[00:06:18] BF:** Sure. We had this evolution. As Matt said, we launched over the summer. When we first started, what we had built was the ability for you to build a web app that can also read and write to your local shell. And so it was literally anything you could build at HTLM, CSS, JavaScript, plus the power of running shell command, Bash commands, whatever it was on your computer. That is what you could build. So anything you can sort of build on the Internet, you could build with Fig.

What we realized is that's great, but it takes time to build a good web app. And so what we did was build our own sort of structured UI, the idea being we want you to hear about Fig and sign up to it and then have built an app for yourself over your team in two minutes. And so we built these things called interactive runbooks, which are essentially you can build a document in Markdown, like the simplest programming language, if you even want to call it that. And then we wrote a tiny little scripting language that lets you – That sort of adds form components, like dropdowns, imports, checkboxes, those sorts of things, plus the ability to read and write to the shell.

We still sort of call them apps. We'd call the interactive runbooks just because we make that separation. But the idea is you can build anything that you can sort of build in a shell script and anything that you would build and you'd write down in Notion or Confluence or one of these

other documentation tools. But have it actually live in your scripts folder. And that means you can just streamline all of these tasks that you're often context switching out of the terminal for.

**[00:07:46] MS:** An the deeper integration means we can do smart things. So say you're forking a database. We can run a shell command in the background to pre-populate a dropdown with the list of all your local databases so you don't need to find that data by running the command yourself.

**[00:08:02] JM:** What's been the hardest engineering problem in building Fig so far?

**[00:08:06] MS:** That a really interesting question. I think probably, with the terminal, there are just these layers of standards that build up over time, because it's such an old and kind of fundamental application. And so we didn't want to rewrite the entire terminal from scratch. So what we've had to do is kind of retrofit to existing terminals. And that's kind of required doing some hacky things on the shell side and then also using the accessibility API for Mac to connect to the various Windows that you're using.

**[00:08:32] JM:** Can you go a little bit deeper on that?

**[00:08:35] MS:** The accessibility API is just one of these is Mac low-level frameworks that gives you more access to certain window components and stuff. So like, normally, you would use this to build a screen reader or something. But what we're using it for is to connect this window to your terminal.

And then the other kind of interesting set of challenges we're run into. And so our newest product is called Auto Complete, and what we're trying to do with Auto Complete is allow you to have the kind of experience of a modern IDE in the shell. So as you're typing CD, we can start popping up and pre-populating the list of folders in your current directory and so you can drill down through your file structure without needing to run LS every time if you don't remember the exact path there. So that's been kind of than the newest thing we're working on. And that's required building out our own shell parser, our Bash parser, and then building a full kind of auto completion engine that allows us to kind of predict what you're – Not predict what you're going to type. But like give you smart contextual suggestions.

**[00:09:37] JM:** And there's also a form of documentation for terminal workflows with Fig. Can you talk about the terminal workflows and the documentation for that?

**[00:09:48] BF:** I think – are you referring to the interactive runbooks?

**[00:09:51] JM:** Yes.

**[00:09:52] BF:** Okay. Yeah. That is literally markdown plus a little bit of our own scripting language. We call these apps or these documents run down documents. And the idea is you can type in Fig runbooks and in the same way, GitHub. You have multiple repose, which store your applications. You can sort of store your processes and workflows centered around the terminal in this mini-Fib runbooks app that we built, and you can just build them locally inside the application. And then just with a click, you can share them with the world in the same way you can sort of make anything public and open source on GitHub. Or if you want to just share them privately with yourself or your team, you can publish them to the cloud. We'll host them, but anyone in your team can now go type in Fig, the name of your sort of company identifier, and then the name of one of these runbooks.

And so for instance, Matt and I use Fig to build Fig, and whenever we need to – Let's say we're publishing a video or we're a deployment or something like that, we just open up the relevant runbook, which not only has the documentation, but actually lets us quickly type in, "What is the name of this file? What is this location?" those sorts of things. And then we click a button and it does that entire process for us. And we don't have to leave the terminal.

**[00:11:12] JM:** How does Fig help developer teams?

**[00:11:15] BF:** Our current focus is actually more on the individual developer. We really, really want to nail that. like you download Fig and you don't need your team on it. You don't need the rest of the world on it. You are immediately getting used out of Fig from day one. Our plan is obviously to go into teams. Our plan is same with sort of GitHub adds value individually, but at the end of the day, when you open up to teams, you can add a lot more value.

So for teams, things like these runbooks, these processes that everyone on the team is doing. And a lot of the user interviews we had, and even when we signed up, when we had our product hub launch, we had thousands of people sign up. One of the interesting things was a lot of engineers who'd been an engineer for 10 years or more signed up. And as Matt said at the start, we thought that this was going to be a tool for novices, like beginner. I've never used the terminal. I don't know how to do anything at all. And what we realized is your problems when you're an advance engineer are not CD and LS and navigating the terminal. It is Kubernetes, and Docker, Compose, and like all of these other crazy CLI tools, these monolithic tools. And what happens is a CTO might write tons of scripts that automate all of these processes. But the people who end up using them are the people who've been at the company for the past five years or more, or the CTO, him or herself. And then everyone else sort of gets left behind and they end up bugging their senior engineers saying, "How do I do this? What are the input parameters for that? There's an error with this." And documentation goes stale. No one ends up using the tools which are designed to make you faster. And what's the point of having all of these tools if no one's actually using them?

And so Fig for teams is – Well, it's the best of both worlds. It's the best of having sort of something in notion, and that it's discoverable. There's some sort of hierarchy to it. Actually, you can read through it and go, "Oh! This makes sense." There's that discoverability layer. But at the same time, there's the speed layout of I can run this like a shell script essentially. It is as fast as me doing ./build or something like that. And that enables the advanced engineer is to be quick, but then the newer engineers to be just as quick as the advanced engineers from day one.

**[00:13:29] MS:** An what's cool is we're seeing some organic adoption. So if somebody in an organization writes a runbook, the natural thing to do is to share it with their teammates. And we're still in beta at the moment, but that's been a large way we onboard new users to the platform, is in the form of like their team asking for access.

**[00:13:48] JM:** So the vision here is actually to have an app store where people can build different GUIs for different tasks. Tell me about the app store vision.

**[00:13:58] BF:** So we sit a lot like GitHub, that these apps that you build, they can be useful for yourself and useful for your team. But in the same way, you can just open source something

and make it public on GitHub. We want to take that whole world, but hyper focus it on the terminal. So I may build – We're talking earlier about DBT. I may have these processes in the terminal that are really long and complicated, but I have to do them for myself or for my business. But chances are someone else in the world has that same problem. And so why can't you just say just let anyone do this just like GitHub?

And then what we imagine is we say app store, and maybe a better analogy is sort of like a browser, which is why should I – you often search things in Google. You get an error in your terminal. You look something like that and you end up on the – Really far down, some GitHub issues page or on the fourth answer and stack overflow because the correct answer was marked correct five years ago and things have really, really changed.

Google is fantastic for general search. But when it comes to actually developer search, it isn't actually as good as you would hope it could be. So we imagine this world of, "Well, I'm in the terminal. And, hey, why can't I just type in Fig?" It pops up this little browser for developer search for terminal-specific processes. And I search how to speed up a video in FFM peg? And someone in the world has faced that problem before, has built a little runbook or app for it, and it just pops up on the side of your screen. You do that exact process, and then you're done.

And the same with UNIX, you can chain commands together. Well, why can't you do that with Fig? Why can't you speed up a video and then reduce the file size and then something else? So we imagine this world of these tiny little apps essentially that and components that are hyper focus on one specific task and it's all – You never have to leave your terminal.

**[00:15:59] JM:** The GUI itself for Fig. So it like hooks on to the terminal. How is that actually built?

**[00:16:06] MS:** Like I mentioned earlier, that's built using the accessibility API on Mac.

**[00:16:12] JM:** Is that kind of a hack?

**[00:16:14] MS:** It is a little bit of a hack.

**[00:16:15] BF:** Oh, yeah.

**[00:16:16] MS:** Because, again, going back to something we said earlier. It's like it's very hard to build a terminal from scratch. There is a lot of moving parts. There are old protocols, and everybody uses the terminal in kind of their own way. And so we realize kind of when we're getting started, that if we wanted to validate this idea quickly, it would take way too long for us to build a terminal from scratch. And then also, we had no guarantees that people would switchover and use it.

So we wanted to make something that was useful that allowed developers to keep using the tools that they were already comfortable with and get a chance to try out Fig without kind of replacing everything they already are doing. But going forward, at some point, we would love to kind of reimagine what a modern terminal looks like and sort of basically take the hack and turn it into a fully-fledged terminal of its own.

**[00:17:03] BF:** We just went through YC, and YC's big thing is move quickly, move fast, break things, all of that. And they're one of the reason why we built it like this. But we had a sort of 10-week spring. And what we really wanted to prove was that terminal augmentation is something that people actually want. It really hasn't changed since 1970. There must be something else here. There must be a thought, like more productive ways to do certain things. And so rather than spending literal years building the perfect terminal and spending all of our time making sure the basics actually work, what we did was exactly as Matt said, we sort of attached on and add this extra layer. Yes, it is a hack, but if people are using this, then we have a very clear signal that, "Okay, terminal augmentation is something people want. Now, let's go back and build the best terminal having spent all of this time working out," one of the core things that people really, really need." And so just, one, it's a good way of us helping users quickly. And two, its' a good way of us like prioritizing what we build when we built the new terminal.

**[00:18:09] JM:** So you must've built some kind of protocol between the accessibility APIs. Like you must've layered over the accessibility APIs to have some consistent interface that is at least much easier to interact for the average developer. What's the shim that you built over those accessibility APIs?

**[00:18:30] MS:** Yeah. So the way we think about Fig – again, Brendan mentioned this earlier, is sort of as a browser. And again, under the hood, we are using web technologies, but we're not an Electron app. To offset any worries. We're built in Swift. But what we like about the web is that it makes it really easy for people to build something quickly, approve in kind of rendering engine that is used by all these incredible sites and there's lots of mature tooling around it. And it gives you kind of cross-platform capabilities out-of-the-box.

So what we've done is we've let you run shell commands and have kind of deeper access to the native system through a JavaScript bridge. That you can run code through Fig and it can run a shell command on the user's behalf. It can read from the file system so they can access their local files. And yeah, that's basically the sort of bridge that we built out between the local system and the app store.

**[00:19:26] JM:** So let's say I'm a developer working on my newest Fig app. Let's say I want to build a Fig app that allows you to listen to a Software Engineering Daily episode based on what terminal commands you've entered in recently, for example. Something dumb like that. What am I doing?

**[00:19:44] BF:** Sounds great. Okay. The idea is you want to build based on the terminal command you've entered recently, you want to match that up with the Software Engineering Daily. So you could – Well, you could run a shell command on your computer, which gets your history. You can use JavaScript to parse your history rather than using like shell or Bash or whatever shell you're using. And then you can just use web technologies to hit some API end point that you've set up and essentially return the suggestions and then just web technologies to render the output. So that's one version, is you build a web app, and the way you get a lot of the data is by running shell commands on the user's local computer.

And then another version is you can build one of these interactive runbook. So rather than doing all of the UI and whatnot yourself, like we handle all of that, and you can just build this sort of underlying technology. So you could have a little app that built entirely in Markdown. It would run a shell command. You maybe would have to do some the parsing, more in Bash in this case. But can we do webhooks? We can't do HTTP yet in these runbooks, but you could type in a specific command, for instance, and use like a call request to hit an endpoint, get the output,

and then we can just display that in Markdown for you just like a list. It's very, very simple. And that would take you like – If you had the right API endpoint set up, that would take you probably 2-1/2 minutes to build.

**[00:21:16] JM:** I guess this is definitely something you cannot open source, right?

**[00:21:19] MS:** So I think the long-term goal actually is to open source a lot of these stuff. And all of the apps that we've built on top of kind of the core Fig browser are open source. And then once we start developing our new terminal, I think that that's probably going to be open source from a very early stage. But yeah, these are questions that we've been thinking about, and we really – We use a lot of open source technology and want to give back to the community.

**[00:21:45] JM:** Give me a little bit more detail on the lifecycle of the execution of a Fig command.

**[00:21:52] MS:** So if you run Fig command through the JavaScript bridge that we've set up, it kind of goes through to the Swift app itself, which then runs the shell command on the user's behalf and then returns whatever error codes or texts that's passed back to the JavaScript.

**[00:22:10] BF:** And then if you are asking about a CLI command, like I type in Fig curl and it opens up the curl app, then that is – Matt, you're obviously the one who should be answering this, but I'll give it a go. It hits a CLI. Like we have an executable installed on your computer. Then that is hitting the Swift app. And then that is doing all of the parsing that we need to work out. What do you actually want to do? And then that's – Is it opening a window? Is it giving a specific output? Is working out of Fig? Whatever it is. And performs the action based on that. So just like any other CLI tool.

**[00:22:46] JM:** So the code that interacts with the accessibility APIs in the terminal, that's Swift code?

**[00:22:51] MS:** Yeah.

**[00:22:52] JM:** Cool. Did you guys know Swift before you started working on this?

**[00:22:56] MS:** So I've been like an iOS Mac developer for ages, which is one of the reasons why we went with Swift, rather than trying to do this in Electron. And yes, it is a great language. It is much more pleasant than the JavaScript in my opinion. The Typescript is maybe changing a little bit. But yeah, I've been writing Swift for a while.

**[00:23:14] JM:** So how does the experience of a developer that uses Fig compare to one without Fig?

**[00:23:20] BF:** And this is not building an app in Fig. This is just I'm a first-time –

**[00:23:23] JM:** I'm a developer using Fig and I've never tried it before. What am I going to be surprised by? What am I going to like?

**[00:23:30] MS:** So out-of-the-box, one of the things that Fig does that's really cool is as you're typing in the terminal, say, you're typing out a Git commit or a Git push, it can start generating suggestions for you so you don't need to type everything out. And it can sort of auto completing the branches that you're on or the remote servers that you're trying to push something to. Similarly, when you're kind of seeding through your directory, your file directory, you can pre-populate the different folders that are there. That's kind of out-of-the-box what you're going to see as you install to Fig. But then there is this sort of app ecosystem that's there as well when you kind of explicitly callout to Fig. If you run Fig dirt to open the file navigation, or if you run Fig curl to get this sort of Postman-like UI.

**[00:24:13] BF:** It is sort of what the – when we first started, it was really the value you were getting out of Fig is that it's a super extensible application that it's very hackable. You can build on top of it. You can build whatever the hell you want on top of it. And we build a bunch of proof of concept. So okay, curl was the sort of obvious one. Everyone forgets the flags. You can use the Fig curl app that Matt and I have built. But really, you know your problems way better than we do.

And so our focus has really ben, for at least the early stages, on let's make this as easy as possible for you to build your own applications and streamline your own workflows. Then we

sort of started building our own apps. We have the curl app. We have a node's app, where, okay, I just learned a new command. I'll probably going to forget this in a week. So where do I write these nodes? And you end up defaulting the Mac node's app, or to some other place that is just not the right place for them.

So you can install your nodes in Fig. It's an obvious place to do that. And then the last thing, what matches that, is we want – When you download Fig for the very, very first time, how can we give you that aha moment that, "Okay, fig is awesome. I'm going to stick with this for a long time." And that is auto complete. It's such an obvious thing that IDEs give you. It's weird going to an IDE that doesn't have auto complete, or Google searching something without auto complete. And it's crazy that the terminal doesn't have that. And so we've sort of built the infrastructure that lets CLI tools build auto complete much cost of the building it using the old sort of completion specs in Bash or the new Z-Shell completion specs.

And now rather than hitting tab, it can actually just pop up and like, "Oh, crap. What's that AWS command that lets me create a new S3 bucket?" Or, "Oh! What does this command mean again? I vaguely remember seeing it. But I'd like to learn it." And the same way you could do this with IDE or auto complete. You'd never have to leave your terminal. It all just pops up. And importantly, it's not annoying. That was one of the big things we wanted, is the worst thing would be auto complete, but when I want to hit enter, it adds a new command or something like that.

So we've really built it with stay with your existing workflow, and we augment that existing workflow. And if you want to buy into the Fig ecosystem even more, you type in Fig and it pops up our app store. Or you type in Fig curl, then it pops up the app. But you can keep your exact same workflow, and Fig is adding value straightaway.

**[00:26:29] JM:** What would you like to see people be able to build with Fig that they can't build today?

**[00:26:35] MS:** I think that for a lot of people, the terminal can be a little bit scary especially when you're getting started. And so anything that can just take away the pain and kind of confusion of setting up a development environment, of installing the right dependencies. I think that that's a huge opportunity. As well as a lot a lot of kind of the existing CLI commands. Again,

Heroku has an incredible CLI tool. It has great documentation. But how much better could that experience be if you didn't even need to go to the browser to look up how to do something? It was auto completed or auto suggested as you were typing it in the terminal. Or there was the runbook for getting set up with a new node Heroku app. So those are the type of things that I think really would add a lot of value to the community and are things that we're actively trying to build up.

**[00:27:20] JM:** The AWS examples. So AWS has a huge surface area. So what's the design process for somebody building AWS-related fig apps? How do you cover that surface area?

**[00:27:35] BF:** Yes. So two things. One, because we have a bunch of little products – Not little. We have the auto complete product. We have the runbooks. We have the Fig apps. We thought when we started that we could build the Fig AWS app and it would do everything. And that already exists, and that is called the AWS Console, and that is – You can speak to any developer. Not the most ergonomic place to go when you want to get stuff done unless you've like lived and breathed the console for a while. And so what we realized it's actually very modular. When I'm working with AWS, often, I want to do something with credentials. Often, I want to set up an S3 bucket. I'm working with EZ2. I'm doing something with containers, whatever it is. You have that workflow, which is often pretty – There is a workflow that is actually standard across. There's definitely a workflow that's standard across teams and likely standard across other companies and individuals.

And so what we ended up doing is what I was talking about earlier of the modularity. What if we could build the little runbook that makes it easy to set up an S3 bucket? And then the little runbook that makes it easy to delete another S3 bucket? What if can then let you add files to it really? Like all of these little things that rather than having one application that does absolutely everything, it is just one thing that is like very, very specific, and then you move to the next thing that's very specific, and you can chain them together.

And so our dream is you type in Fig, opens up our app store, and then you can search how to set up AWS S3 bucket. And maybe 3, 4, 5 million people have built these runbooks, and because the community is sort of ranking them, you get very quickly at the top. Maybe AWS has built the official runbook that does it, or maybe someone else has built the runbook that lets you

do it.

**[00:29:13] MS:** You can think about it a little bit like NPM, but for tutorials or workflows. So just like you would kind of go Google something when you're getting started setting up a Postpres server or instance or whatever, you could do that now in your terminal and it will give you the canonical way to set it up and then can pre-fill in certain values and stuff.

**[00:29:33] JM:** How else do you see Fig evolving in the future?

**[00:29:37] BF:** We want to build our own terminal. That's the main goal. Yeah, our key thesis is terminal has been updated since 1970. Here's a tool that is used by software engineers, hardware engineers, data scientists, sales engineers. This is the one tool that binds any sort of engineer together, and it hasn't been updated in so long. It's crazy. So that's really where we want to go here. And we want to build it with the community. And that's why rather than trying to impose what we think the best thing is, we want to see how this Fig app store flourishes and see what people really want, and then build that into this new terminal that we start building. Whether that's six months from now, a year from now, tomorrow. We don't know the exact answer, but we just want to be iterating quickly and probably like launching this, getting people using it, getting their feedback and iterating.

**[00:30:27] JM:** What would be hard about building your own terminal?

**[00:30:31] MS:** It's sort of similar to building your own browser in a lot of ways, because there's a lot of backwards compatibility that you need to be aware of. You don't want to break anyone's existing workflows. So there're a lot of just edge cases that maybe only 1% of developers end up needing. But you don't want to annoy people. You don't want to take something away from people when they think they're upgrading. And so that's I think probably one of the biggest challenges, is finding ways of blending this old technology with some of the new approaches to affordances and accessibility.

**[00:31:04] BF:** Another thing I would certainly say is speed. That one of the big reasons people use the terminal is you don't have to go – AWS, example again, you don't have to go to the AWS console and click all these buttons and that's really, really slow. When if you just know the exact

sequence of characters, you can perform that workflow in 10 seconds, 5 seconds, however long it takes you to type it out.

And so making sure that everything we do is super, super fast and there is no – We spoke to a ton of developers as we built Fig out. And they say I can notice if there's a 50 millisecond lag in anything in the terminal. There is a reason why people don't use – Not don't use tools. We're not say anything bad about it all. But hyper – Just takes a little bit longer to set up, to load up when you open it up for the first time. When you're used to opening up the terminal and it's instant and you type things in and there's no lag, whatsoever. Even anything that's 50 milliseconds more is weirdly noticeable. Not weirdly. It's understandably noticeable. And so we got to make sure that it is super, super fast, as fast as before. You can do everything that you can do before in the same way, and you can do way more is the goal.

**[00:32:19] JM:** The last time a new terminal was built – I know there are these different terminals. There's like Z-shell, and Bash and there are things that I never paid much attention to why there are different terminals. How have they diverged in the past?

**[00:32:33] MS:** So there are two things there. There's kind of new terminal. So like Hyper, Alacrity, Kitty. I mean, these are terminal emulators. So they're basically giving you the same experience that you would've had when you were typing at the teletype 70 years ago. But some of these newer ones are rendered by a GPU or rendered with web technology. So it's kind of giving a new interface for this old tool.

And then there are shells. So that's like Bash, you have Z-shell, you have Fish, you have a new shell called Oil that is interesting that hits Hacker News some of the time. And for a longtime, these have been sort parallel efforts. So you have the terminals, which kind of are the browsers here. They provide sort of the base functionality, but they don't do a ton of smart stuff. They leave the smart stuff to the shells or to whatever is happening inside the terminal process.

And the shells will do smart things like auto complete or kind of giving you suggestions. But we think that there's a cool possibility of combining the two of having a shell that can present stuff in a UI and basically breaking down this delineation between the shell and the terminal and creating an integrated experience that feels smoother and feels more modern.

**[00:33:46] JM:** How is the auto complete feature built?

**[00:33:50] BF:** So a lot of moving parts. The first part is we – It's all local. I will say that first-off that we know there stories of things hitting servers, and the last thing you want to do is send any confidential or private information to any server. We're always upfront about all of our privacies up. So everything to do with auto complete is local.

The first thing you do is you download a completion spec. A completion spec is literally, "Okay, Git has this number of sub-commands, and then this number of sub-sub-command, this number of flags," whatever it is. And it's really just a big JSON object that you downloaded into a specific folder and we have a Fig command that lest you sort of Fig installed, Git Fig install, whatever that the completion spec name is. And all of these is open source and built by the community it in our public GitHub repo.

So you download it, and then what happens is you're typing something in. We are parsing what you're typing in. So if you are – I don't know. You have some expansion. You have used a semicolon, the start of a new command. You have used an and operator, like all of that stuff. We need to sort of workout what is the command sequence that you typing at any given time. Then we need to sort of work out, "All right. Are you typing in –" Of that, are you now typing in the arguments to an option? Are you typing in subcommand? Sort of building this in the same way VS code has built it. And so predicting what are the possible things that you can type. And then using the completion spec that you've downloaded to build those completions out.

And so we can do completions for things like Heroku is an example here. I want to add a new add-on to my Heroku app. When you do the -a flag or --app and you select which app you want to do, usually you would have to either know the app's name exactly or you'd have to run a previous command to work out the exact name or you'd end up going to the browser to work out the name. Whereas we can actually auto complete the 3, or 4, or 10 app names you have. And then you start typing them in. And this is all local. We're just sort of running the Heroku apps, JSON commands to work out what the apps are.

So under the hood, the quick summary is we are parsing your command. We are then mapping

the parsing of it to the completion spec, and then giving you suggestions based on what the completion spec has said.

**[00:36:09] JM:** What parts of fig have we not explored?

**[00:36:12] BF:** So one of the things we built pretty early on was building your own CLI tool for either yourself and for your team. So teams would have a CLI, but a big problem was they wouldn't know – A couple of problems. One was building the CLI. There's usually sort of key man risk. There's one person who's built it. And it's hard to update. Things go stale. It's sort of this black box, and there is usually one person. If they leave the company, someone has to learn how to do it. It's not really – You can't contribute to it together. Sometimes you can, but it's not as nice as having a shared notion document or something like that.

And so we wanted to make it easy to sort of collaborate on building this CLI. And then the second thing was, well, people, the top 10%, 20% of engineers at a company know a big CLI pretty well. But then the other 80% – And these numbers aren't exactly fixed, but they may know how to do a couple of things. But the other commands just aren't discoverable. So you wouldn't know what the inputs are or the parameters for anything, the subcommands, all of that. You'd end up bugging your boss, "How do I do this?" How do I do that?" And that sort of defeats the purpose of a CLI if it's meant to make you faster and it ends up adding two or three more problems.

So what we wanted to do was make it way more discoverable and much easier to collaborate. And so we have a cool tool, Fig build, and you can build your entire CLI just visually. We have a nice little nested hierarchy on the left side so you can add new sub commands. And then you can just define which scripts you want to run. You could define the input parameters and options and whatnot to the sort of CLI tool. And you can just build this thing really, really quickly. And then we're actually building out the feature where we can host it. So when you make an update, it's pushed instantly to everyone's computer, or you can also just have this in your Git repo and use it amongst your team as soon as everyone does a Git pull.

**[00:38:09] JM:** It's a really interesting property, and it makes me think about the fact that when I was first looking at Fig, it's like, "Oh, this is entirely client-side application." But there are some

benefits to the network, like kind of over the wire updates and stuff, obviously. Can you tell me a little bit more about what aspects of Fig are not inclusively client-side?

**[00:38:32] MS:** Yeah. So we definitely were thinking about this upfront, because anytime you're giving an app access to you shell, like that's a lot of power potentially. And so security and what's going over the wire and what's local was top of mind for us. So all the runbooks can be hosted locally, or you can share them in the clouds, but definitely if you have a local version, like that's run first. You can install fig apps locally so that it doesn't hit the cloud. Or when you're kind of just discovering apps, that's basically the time when it might go download something from a remote server as if you are on the app store and you find an app that you want, then you haven't installed it already, it will prompt you with the set of permissions and what command this app is enabled to run on your behalf. So before you have remote code running on the machine, we make sure to give you the chance to that and opt in o opt out.

**[00:39:27] JM:** The terminal is something about the developer experience that you guys have reimagined. Are there any other elements of the developer experience that you think should be reimagined?

**[00:39:38] MS:** I think like a lot of the pain points in the terminal have to do with deployment. Have to do with dependencies. Like this is why there needs to be a reimagining, it's because like the processes and the workflows in the terminal are the things that are hard and not necessarily the most discoverable. And so we think that having abstractions over some of these lower-level things will be useful and make the terminal more accessible. But in general, I think that for things like deployment, Heroku abstracts over a bunch of AWS commands and infrastructure expertise that you need in order to set up a similar thing. And so we'd like to kind of apply that model to the terminals well and potentially even kind of build out our own sort of abstractions in a similar way.

**[00:40:23] BF:** I think there are always things that you can improve with the developer experience. It's just things always getting faster. Three are always going to be new ways of doing things. And so things need to be simpler. The more we do this, the more we notice problems in other areas. And I think the cool thing with the terminal is it is this – The root of all of these problems, you can start with the terminal and you can have an easier way to do things

from the terminal because it connects to – Does web requests. It does cloud. All of these stuff can start at the terminal, which is why we like it as the space.

**[00:41:00] JM:** Okay, guys. Well, that sounds like good place to stop. Thanks for coming on the show.

**[00:41:04] BF:** Thanks so much for having us.

[END]