BECOME HARDCORE EXTREME BLACK BELT POWERSHELL NINJA ROCKSTAR

BY DON JONES

Become Hardcore Extreme Black Belt PowerShell Ninja Rockstar

Don Jones

This book is for sale at http://leanpub.com/become-powershell

This version was published on 2018-04-04

Tweet This Book!

Please help Don Jones by spreading the word about this book on Twitter!

The suggested tweet for this book is:

I'm becoming extreme black belt PowerShell ninja rockstar!
http://leanpub.com/become-powershell

The suggested hashtag for this book is ##powershellninjabook.

Find out what other people are saying about the book by clicking on this link to search for this hashtag on Twitter:

##powershellninjabook

Also By Don Jones

The DSC Book

The PowerShell Scripting and Toolmaking Book

Be the Master

Don Jones' PowerShell 4N00bs

Don Jones' The Cloud 4N00bs

Instructional Design for Mortals

As with everything, this one's for Chris.

Contents

About This Book

The 'Forever Edition' of this book is published on LeanPub[1], an "Agile" online publishing platform. That means the book is published as I write it, and *that* means although you may not have a complete book right now, you eventually will - and I'll also be able to revise it as needed. I also appreciate your patience with any typographical errors, and I appreciate you pointing them out to us - in order to keep the book as "agile" as possible, I'm forgoing a traditional copyedit. My hope is that you'll appreciate getting the technical content quickly, and won't mind helping me catch any errors I may have made.

If you purchased this book, thank you. I know it's available for free on LeanPub.com[2], but I appreciate your purchase. Know that most of the money from this title actually goes to local charities where I live, including ThreeSquare.org. If you *didn't* purchase the book, but later decide it's worth a buck or two, hop on ThreeSquare.org and make a donation in whatever size you can. I really thank you.

---

[1] http://leanpub.com

[2] http://leanpub.com/becoming-powershell

About the Author

Don Jones has been a Microsoft MVP Award recipient since 2003 for his work with Windows PowerShell and administrative automation. He has written dozens of books on information technology, and today helps design the IT Ops curriculum for Pluralsight.com. Don is also President, CEO, and co-founder of The DevOps Collective (devopscollective.org), which offers IT education programs, scholarships, and which runs PowerShell.org and PowerShell + DevOps Global Summit (powershellsummit.org).

Don's other recent works include: * Learn Windows PowerShell in a Month of Lunches (Manning.com) * The DSC Book (LeanPub.com) * The Pester Book (LeanPub.com) * The PowerShell Scripting & Toolmaking Book (LeanPub.com) * Learn PowerShell Toolmaking in a Month of Lunches (Manning.com) * Learn SQL Server Administration in a Month of Lunches (Manning.com)

Follow Don on Twitter @concentratedDon, on Facebook at facebook.com/concentrateddon, or on LinkedIn at LinkedIn.com/in/concentrateddon. He blogs at DonJones.com.

Feedback

I'd love your feedback. Found a typo? Discovered a code bug? Have a content suggestion? Wish I'd answered a particular question? Let us know.

First, please have a chapter name, heading reference, and a brief snippet of text for me to refer to. I can't easily use page numbers, because my source documents don't have any.

Second, understand that due to time constraints like having full-time jobs, I can't personally answer technical questions and so forth. If you have a question, please hop on the forums at PowerShell.org[3], where myself and a big community of enthusiasts will do our best to help.

Then, head to the LeanPub website and use their email link[4] to email me. I can't always reply personally to every email, but know that I'm doing our best to incorporate feedback into the book.

Finally, accept my thanks!

[3] http://powershell.org

[4] https://leanpub.com/the-dsc-book/email_author/new

Intro

This book started out, as many of my books do, as a discussion over drinks. The original end-game of that discussion was the "VERIFIED EFFECTIVE" tests that PowerShell.org ran for a while, but those had a kind of low success rate. A lot of people *think* they want a certification exam, but in fact what many people want is a personal *diagnostic* to direct their future learning. Yeah, certifications are important as a way of saying, "look what someone else said I can do," but folks, *PowerShell is code*. Unlike a network infrastructure that you built, you can carry PowerShell scripts around with you. That's called a *portfolio*, and if you don't have one, start working on one. Your portfolio is better than any acronymic exam.

So this book is what happened next.

Basically, this book is about the journey to becoming really awesome in PowerShell. That is, incidentally, an entirely useless thing as far as jobs go. Nobody wants people who are merely awesome in PowerShell, any more than construction managers want people who are awesome in hammer. Contractors want people who are *awesome at building walls* and stuff; IT managers want people who are awesome at getting stuff done - like automating business processes. Being awesome in PowerShell is fine, but only if you can put it to beneficial use someplace. So you also need to be awesome in Windows, or Active Directory, or Azure, or whatever, right?

Anyway.

The first part of this book is presented as a quiz. Each "chapter" in the quiz basically covers one of the major steps to becoming Hardcore Black Belt PowerShell Ninja Craftsman (tip o' the hat to Jim Christopher, there, who shares a disdain for stupid and overwrought job titles). If you can complete the second quiz, you can probably complete the first one pretty easily. They build on each other.

Here's the part you're probably not going to like: there are no answers to the quizzes. Well, I mean, there are obviously *answers*, but they're not printed here. That's because I feel there are really three mental conditions you can find yourself in after reading one of these questions:

1. **You flat-out know the answer**. In which case, you don't need me to confirm it for you. If you know it, you'll know you know it, and that's all that matters.
2. **You maybe might know the answer, but you're not sure**. Meaning, you don't *know* the answer, and me giving it to you isn't going to make you more smarter or more usefuler. You need to go *learn it* until you *know it*. This book isn't intended as some shortcut to common job interview questions. This isn't the CliffsNotes of PowerShell. Go *actually learn something*.
3. **You dunno**. In which case, again, me merely giving you the answer isn't going to make you know. Go learn.

Part of the point here, too, is that many PowerShell questions have many answers. None are inherently better than the other, and whenever I do some kind of "quiz" with written-down answers, everyone wants to argue about why *their* answer is better than mine. I'm not into it, this time. If you *know*, then you know, and we're good. If you don't, well, then you know where to go *learn* because I'll be telling you that much, at least.

You might also just consider trying some of these things. Figure 'em out. That experience alone will make you smarter than some appendix full of "answers." Face it: *I can't give you answers*. I can recite facts at you, but that's not learning.

And don't take this too awfully serious, OK? I mean, look at the title of thing thing. This is meant to be a kinda fun thing, maybe a way to figure out where you could be learning something new and fun, and a way to kill some time that doesn't involve reading a lot of code. You're definitely *not* going to become a ninja, there are no belts, and there's no core, hard or otherwise. Have fun.

Part 1: The Quiz

In each of the following phases, I'll provide some learning and reference material suggestions. If you can breeze through that phase's questions, move on to the next. If you miss a few, consider reading the materials referenced. If you miss a lot, definitely don't move on - go grab the materials referenced and learn-up. Then come back and try again.

Phase A

PowerShell v1.0 was introduced in 2006 at TechEd EMEA in Barcelona, Spain. Inventor Jeffrey Snover, at the time a Partner Program Manager for Microsoft, demonstrated what had been called "Monad" and announced that it was now "PowerShell." PowerShell v2.0 came out a little bit later, and essentially "finished off" all the stuff that we all wished 1.0 had contained (1.0 was a bit rushed, due to it being a pre-requisite for Exchange Server 2007). Most of what you'll be asked in this Phase has existed since 1.0 and 2.0.

If these questions stump you, consider picking up a copy of *Learn Windows PowerShell in a Month of Lunches* (presently in its third edition) from your favorite bookseller, or from the publisher's website[5]. Note that, no matter where you buy it, every physical copy comes with a voucher for PDF/MOBI/EPUB versions. You can also buy just the ebook versions directly from the publisher.

1. How would you make PowerShell display a list of all commands that have a `-ComputerName` parameter?
2. How would you get PowerShell to run an external command, like `ping.exe`?
3. How would you display a list of installed digital certificates?
4. How would you map a new drive letter to a UNC path, from PowerShell, and have that drive also show up in Explorer?
5. Why can you pipe the output of `Get-ADUser` to `Set-ADUser` and successfully make changes to whatever users you got?
6. Assuming you had a CSV file containing a column named samAccountName, why would `Import-Csv filename.csv | New-ADUser` work, or why would it not work?
7. How could you display a list of modules from PowerShell Gallery that contain "HTML" in their titles?
8. How could you display a list of all installed modules on your computer?
9. What's the difference between a snap-in and a module?
10. Why does `Get-Service | Format-Table | ConvertTo-HTML` result in garbage output?
11. In *PowerShell v5 or later*, how could you redirect `Write-Host` to a text file?
12. What is the difference between `-like` and `-contains`?
13. What does `Get-Member` display?
14. What underlying protocol do `Invoke-Command` and `Get-CimInstance` both use?
15. What authentication protocol does `Enter-PSSession` default to in a domain environment?
16. Do `Get-WmiObject` and `Get-CimInstance` draw their information from the same source, or from different sources?

[5] http://manning.com

17. How do the RemoteSigned and Bypass execution policies differ?
18. List the 5 (PowerShell v4 and earlier) or 6 (v5 and later) pipelines. You don't need to list them in order.
19. Running `Invoke-Command` with the `-AsJob` parameter will always start how many jobs at minimum?
20. Why is object-based command output better than text output?

OK, how'd you do? If you're thinking, "I'm not sure," that's the same as "not 100%," so it might be time to research some of these. I know, I know… *why can't he just give me the damned answer?* Because the answer isn't the point. Knowing the details, the *why*, the *context*… that's what's important. And I've already written plenty of coverage on these topics, as have many other folks. The act of learning, and of *trying*, is what makes you smarter, not just some nut job on the Internet handing out trivia answers.

Phase B

If you're on a good learning trajectory with PowerShell, then picking up the basics will lead inevitably to full-on scripting, and ideally to full-on Toolmaking, the cornerstone of being an Amazing Evangelizing PowerShell Team Leader. [*The PowerShell Scripting & Toolmaking Book*][http://leanpub.com/powersh scripting-toolmaking) has your back covered, and is a good reference if any of the below are stumpers.

1. Why might a well-designed function accept computer names via a `-ComputerName` parameter, but not *also* accept computer names from a `-FilePath` parameter?
2. What's the difference between a "basic" function and an "advanced" function?
3. What does `[CmdletBinding()]` turn on in a function?
4. What's at least one type accelerator that can replace `New-Object -TypeName PSObject`?
5. Why do hash tables ordinarily not maintain the order of the keys you add to them?
6. How would you suppress the output of `Write-Warning` for a single command that was emitting too many warnings?
7. What does the `.SYNOPSIS` keyword pertain to?
8. What construct is used to handle exceptions?
9. How do you force a command to throw a terminating exception for a normally non-terminating error?
10. How do you activate the output of `Write-Debug` in an advanced function?
11. How do you designate a parameter that must belong to all available parameter sets?
12. What's the name of the tool that can create MAML documents from Markdown source?
13. Without writing a line of XML, how can you designate a set of properties to be displayed by default, for a custom object with a lot of properties?
14. Why does Script Analyzer toss warnings for commands that have a `-Password` parameter?
15. What are some of the mandatory pieces of information a manifest can contain in order to publish a module to PowerShell Gallery?
16. How does the `return` keyword behave differently in a function vs. a class?
17. When you run a Workflow in PowerShell, what is it translated to? What actually executes that?
18. What's the .NET Framework class that creates a connection to SQL Server?
19. What's the type accelerator that forces text to be parsed as XML?
20. What is JSON and why might you care?
21. How can you measure the time it takes a function to execute?
22. How can you check the parameter binding decisions that PowerShell makes when executing a series of commands? Like, what command would do this?

23. What feature of PowerShell enables you to keep text strings (like prompts or error messages) in a separate location from your code, for purposes of globalization?
24. What are at least three ways you can get code running in parallel, when running against multiple targets?
25. How many parameters of type `[string]` can accept `PipelineInputByValue` in a single parameter set of a function?
26. Do parameter aliases "count" for pipeline binding ByPropertyName?
27. What is a proxy function?
28. What does the `-Action` parameter of `New-PSBreakpoint` do, and why might you use it?
29. What value must a ValidateScript attribute return?
30. Why is there air?

Phase C

Coming off of your "scripting and toolmaking high," you might wonder what's next. The answer is, Becoming Truly Professional PowerShell Magnate. And "going pro" means being able to answer some trickier questions across a broader topical domain. Some references include *The Pester Book*[6], *The DSC Book*[7], and *Learn Git in a Month of Lunches*[8]. What's that? Not using Git for your source control? Well, that doesn't make you a bad person. Not *knowing* Git, however, makes you a bad person. Well, not a bad person. Just a less-than-stellar one, maybe. Learn Git.

1. What does a `Describe` block in a Pester test denote?
2. What is the function of a Pester `Context` block, and why might you use one?
3. What are the three main functions a module-based DSC resource must implement?
4. What are the dangers of using an HTTP: based DSC pull server?
5. What is the function of a *branch* in a Git repo?
6. In Git, who sends a Pull Request, who receives it, and what can the recipient do with it?
7. How do you ask Pester to show code coverage statistics? What do these mean?
8. What parameter of a Pester `It` block is always used, and almost always used positionally?
9. What are the different operational modes of the LCM?
10. What is a disadvantage of using partial configurations in DSC?

[6] http://leanpub.com/the-pester-book
[7] http://leanpub.com/the-dsc-book
[8] https://www.manning.com/books/learn-git-in-a-month-of-lunches

Phase D

Make it past this final quiz, and those ninja toe boots are as good as yours. Also that rockin' Fender. Sadly, there are no references for you should you get one of these wrong. At this level, you should dang well be able to look it up on your own. Jump in. Note that this section in particular requires you to have some historical context for PowerShell; if you jumped on the bandwagon in v3, v4, or later, you may need to seriously research some of these.

1. What's the purpose of a dynamic parameter?
2. How would you, in v5 or later, remove all DSC configuration MOFs from a given node?
3. What protocol does `Get-CimInstance` use (and the answer is NOT "remoting")?
4. How could you rename an existing variable without creating a new variable in the process?
5. What is a `filter` construct, how does it differ from a function, and should you use them or avoid them?
6. What is a `trap` construct, how does it differ from other constructs used for the same purpose, and should you use traps or avoid them?
7. In a DSC resource, what are the valid return values from the Test element?
8. Why doesn't `Get-Service` include the logon account that the service uses?
9. How can you permanently add a path to the PSModulePath environment variable?
10. What cmdlet can be used to interact with REST services?
11. What cmdlet can generate a local class that represents a remote SOAP service?
12. What is the wing speed velocity of an unladen swallow?

Got 'em all? Claim your prize.[9]

[9] https://www.amazon.com/Ninja-High-Top-Tabi-Boots/dp/B0000C4JXX

Part 2: The Journey

Nobody becomes Master PowerShell Guru overnight. It's a journey, often one without milestones that we actively acknowledge. But it's definitely a journey anyone can take, and it's one that, for most people, will always wind up in a slightly different place.

Let's Take a Walk

I tried really hard, back in 2006, to convince people to learn PowerShell *right then*. "Do it now," I said, "because there's only like 150 commands. If you wait, it'll get harder with every successive version." Of course, *some* people just weren't courteous enough to have been out of primary school yet, and so here we are - folks are stuck learning ten-plus years of development and evolution. And it can seem like a big hill to climb. Because… well, it's a lot.

But, as the saying goes, "you can't drink all day if you don't start in the morning." If you've not already gotten started with PowerShell, well, you've gotta start someplace. If you start in the right place, you'll line yourself up to naturally proceed to the *next* place, and so on, and so on, like a big technological bar crawl. Before you know it, you'll be passed out in an alley, covered in your own… hmm. I forget where that was going. Anyway.

Rather than tell you what to *learn*, and in what order, I think it'd do us all a lot more good for you to focus on what you want to *learn to do*. You see, I firmly believe that, in the world of IT, simply knowing stuff isn't useful. Fun at cocktail parties (not good ones), of course, but not useful. PowerShell is a tool. A means to an end. So rather than focusing on what to learn in PowerShell, focus on what to *do* with that tool. Make *tasks* your milestones, your accomplishments.

Start Here

Your first milestone should simply be to take several basic tasks that you perform every day, ideally in a GUI, and learn to do those from the PowerShell command-line. I'm not thinking of giant meta-processes like provisioning new users, either - we'll get there, but make this first milestone all about simple, basic stuff. Resetting passwords. Checking on a computer's OS version or free disk space. Little stuff.

However.

You can't accomplish this milestone using Google. I know, *freaking everyone* turns to the Goog first. "Someone else has problem done this," they think, "so why don't I just copy what they did?" Yeah, for the same reason your English teacher didn't let you copy that essay from someone else. *There's value in learning*. We're on a PowerShell *journey*. It's a scenic journey. There are many things to see along the way, and they are mostly awesome (this guy named Jason Helmick is a little scary-looking, but once you hug him it's all good). We're absolutely not in a rush to get to the end of the journey - we're enjoying the trip.

And so, if you're thinking to yourself, "gosh, I can already do bunches of simple things in PowerShell, but I still flunked most of the Phase A questions from earlier, and I really just copied a bunch of commands from Goog to get as far as I have," then make no mistake: **You done yourself wrong**. You

cheated yourself. You practically *lied* to yourself. Had you done that to yourself in church, they'd toss you out.

And trust me, many of us in the PowerShell world *see this all the time*. "I'm running this script I found on (website) and every time I run it, it deletes my entire AD domain. How do I make it not do that?" By *learning*. Everyone wants a shortcut. Everyone just wants to get the job done. Those everyones are wrong. **Wrong**, I say.

Here's the deal: a great teacher can make learning easier and faster. But you cannot skip the learning. If you think learning PowerShell will be hard, or has been hard, you have had a bad teacher. Destroy them and find a new one. **You do not want to skip the learning** because it is the learning that makes you a better human bean.

So. Sit yourself down and *force yourself* to *learn* to do something simple in PowerShell. Use `Help` and wildcards to find things that might help get the job done. Run `Update-Help` first. Once you find something (`Help *network*` maybe, to learn to set a NIC to use DHCP or not), *read the help file* in its entirety to *learn* that command. Don't understand something you've read? Go to forums[10] that are friendly and ask *why*. Man, do we love answering the *why* a lot more than the *how*.

Milestone 1

I'm comfortable navigating the PowerShell console, finding commands, and learning to run those commands. I don't mind experimenting a bit, and I don't mind getting something wrong. I'm much happier to ask "why" on the Forums than "how," because I want to know what makes this thing tick. Syntax, I know I can look up on my own. I never copy someone else's crap from Google. I will make my own crap.

Get Bored

Stick with your first milestone for a while. Keep learning new commands, learning new variations of using them, and using the shell interactively. Do this until you get bored of running the same damn command over and over - this may take you a few minutes or a few days to reach, depending on how often you're doing it. Because once you're bored of running the SDCs (Same Damn Commands) over and over, you're almost ready for scripting.

Almost.

You see, before you're ready for PowerShell scripting, you really need to meet a few specific criteria:

1. You are tired of manually running the SDCs over and over again.

[10] http://powershell.org/forums

2. You are tired of talking to the humans who result in you running the SDCs over and over again.
3. You are tired of talking to humans.

You see, success as a PowerShell Scripter really depends on you becoming something of a misanthrope. The *best* PowerShell scripters can perhaps *emulate* human empathy and emotion, but *most* of them are a lot like Sheldon in *The Big Bang Theory*. If you merely hit criterion #1 up there, you're likely to start writing *scripts*. If you wait until you're firmly into #1, #2, and #3, then you'll be ready to instead write *tools*. Here's an excerpt from *The PowerShell Scripting & Toolmaking Book*[11] (and I swear, we only used "scripting" in the title for SEO reasons):

> My first job out of high school was as an apprentice for the US Navy. In our first six weeks, we rotated through various shops - electronics, mechanical, and so on - to find a trade that we thought we'd want to apprentice for. For a couple of weeks, I was in a machine shop. Imagine a big, not-climate-controlled warehouse full of giant machines, each carving away at a piece of metal. There's lubrication and metal chips flying everywhere, and you wash shavings out of yourself every evening when you go home. It was disgusting, and I hated it. It was also boring - you set a block of metal into the machine, which might take hours to get precisely set up, and then you just sat back and kind of watched it happen. Ugh. Needless to say, I went into the aircraft mechanic trade instead. Anyway, in the machine shop, all the drill bits and stuff in the machine were called *tools* and *dies*. Back in the corner of the shop, in an enclosed, climate-controlled room, sat a small number of nicely-dressed guys in front of computers. They were using CAD software to *design new tools and dies* for specific machining purposes. These were the *tool makers*, and I vowed that if I was ever going to be in this hell of a workplace, I wanted to be a *toolmaker* and not a *tool user*. And that's really the genesis of this book's title. All of us - including the organizations we work for - will have happier, healthier, more comfortable lives as high-end, air-conditioned *toolmakers* rather than the sweaty, soaked, shavings-filled tool users out on the shop floor.

True story. You see, someone who's *merely bored* will end up writing batch files. That is, a bunch of the SDCs strung together in a text file, simply so they don't have to manually type those commands any more. Then, those people will start trying to get fancy, without realizing that all they've done is move their automation problem up a level. They haven't *eliminated* the problem. If you can get to the mindset of, "I am tired of solving this problem, and I do not ever want it brought to my attention again," then you're ready for *Toolmaking*, my friend. You are ready to make *tools that you give to other people* who are less inspired and powerful than you. Those other people can deal with the people who have problems; you simply make tools to solve problems.

[11] http://leanpub.com/powershell-scripting-toolmaking

Milestone 2

I don't solve problems, I write *tools* that solve problems. I do not run those tools, because I am too busy making other tools. Do not call me with your problems; either I have a tool which has already fixed it, or someone else needs to be notified to run said tool. Lose my phone number, please.

Control, Doc, and Test Your Code

After you've gotten some solid tools under your belt, it's time to start acting like a professional. *Nobody likes script kiddies.* Man, that's why VBScript got so little respect, back in the day. Well, that and the fact that it always let you down in the end. But mostly script kiddies.

So what's it take to be a pro? Well, for starters, you're not keeping your code on your laptop or on some file server. You're treating your code like the *mission critical component* that you know it is. You're putting it in source control. If you don't know how to use TFS, or Git, or whatever, then you are not a professional. You also cannot be my friend. Cannot. Be. My. Friend. Even if you buy me bourbon. Even if it's really old bourbon. No source control, no friend.

Pros also document their help, and I do mean writing help files, here. Not comment-based help, either. Amateur stuff. I mean source documents in Markdown, translated by Platyps to MAML, distributed as XML files inside a proper, manifest-driven module. Drool. And put *all* the help stuff in. Examples. Parameters. Reference links. Stuff. All the stuff. Or June Blender will track you down and nag the snot out of you. In front of people.

Finally, *test* your code, which does not mean "run it and see if it explodes." I mean Pester. Learn Pester[12]. Use Pester. The guy who invented PowerShell thinks you should learn Pester. He's right, and you can't be *his* friend unless you do. Not even if you buy him a… well, he drinks crap beer and pays for it himself, but he won't be your friend unless you use Pester to set up *automated testing* of your code. Oh yeah. Automated testing. It's like meta-toolmaking.

Milestone 3

My code is awesome. It is so awesome that I put it in source control without even thinking. I do not write help for my tools, *because I already wrote it* and it's awesome also. And my tools have no problems, and I know that, because I have completely automated test suites that *tell* me my tools have no problems. Also, I refuse to be friends with other PowerShell people who do not also do these things, unless they're just learning how.

[12] http://leanpub.com/the-pester-book

Get Fancy

Your next milestone will be when you start building truly detailed, fine-grained, automation units in PowerShell. Don't even *fuss* with this stuff before you get the first three milestones underneath you, though, because without their *solid* foundation under you, you will not do well. But once you've got the toolmaking thing conquered, start looking at tools that…

- Surface a GUI for less-fortunate users
- Run in parallel against multiple targets
- Fearlessly use .NET classes, which you have wrapped into functions to make them more usable and consistent

Stuff like that. "Advanced" stuff. But, perhaps most importantly, at this point you start to look at things from a very DevOps-y point of view. Your tools are running through a continuous integration pipeline - you make a new version of your tool, you check it into source control, your Pester tests run, and the code is deployed to a repository - all automagically.

You also start to look at your tools as a way to manage infrastructure without touching it. Your tools don't just "do" something, you also have tools that check the same something to see if it's right or not. Your tools lend themselves well to being DSC resources[13] that can provide a hands-off approach to infrastructure management. You have almost *risen above PowerShell* and started to view your entire IT Estate (love that phrase, go Gartner!) as a set of code-based elements that you can manipulate almost by thought.

You are there.

Milestone 4

PowerShell flows from me effortlessly. Advanced topics are one of the few remaining joys in my life, as they give me something new to figure out. Merely automating tasks is something my medulla oblongata handles autonomously now. My code enables declarative infrastructure management. My poo smells of rainbows. I am there.

[13] http://leanpub.com/the-dsc-book

Keeping a Diary

Being an Executive PowerShell Alchemist is no good if it don't pay. One of my biggest frustration points with most tech people is their complete lack of connectivity to business concerns. Actually, that's *most* people, because nobody *teaches us business concerns*, now do they? No, they do not. But not understanding business concerns means you can't communicate your value. Oh, sure, you can say things like, "I'm an MCSE" or "I'm a CCIE" or whatever, but you're taking advantage of a shortcut, there. In those cases, the certification vendor has already done the marketing legwork to translate the acronym into "business value" for you. You're just riding along like an alphabet soup leech. No, what you *need* to do is learn to communicate *your unique value* all on your own. And you don't do that with acronyms. You do it with currency symbols and numbers.

Business is money.

First: Log Your Successes

Never start writing a tool (or script, ugh) unless you can state exactly how much time or money the *current, non automated process requires* on an annual basis. Help desk ticketing systems can be gold for this. "Hey, we spend 8 minutes resetting a user password, and we do that 9 times a day, five days a week, for an entire year. That's (math) 18,648 minutes a year! The people who do that are paid around $40k a year, which is over $20/hour (more on that in a bit), so all those resets cost us over $6,100 a year! Or something. Check my math.

Anyway, don't bother automating anything that you can't do that with. There's no point, because there's no "win." Once you've automated that, though, you can say, "oh, look, that took me 8 hours to do, and now we spend only 1,000 minutes a year on manual resets, which is a savings of (whatever, math, ugh)!!"

And write that in your diary.

At worst, your new resume should look something like:

1. Your name and contact info
2. How many hours of manual labor you saved your last company
3. Your contact info, again

Even if all your'e doing is jockeying for a promotion, this stuff is pure awesome sauce. If you can show that you *saved more than your salary*, surely you deserve a raise. And if not, maybe you deserve a visit to Dice.com or something, no?

Second: Talk in Money

Let's get back to the money thing. It's all well and good to track your automation "wins" in terms of hours saved. Do that, for sure. But you need to know how to translate that to *money*. And to do that, you need to know how to translate a salary (e.g., the salary of a human doing the task manually) into per-minute costs.

1. Start with the gross salary in question.
2. Multiply by 1.14. This gives you a *fully loaded salary* that is inclusive of benefits, employer taxes, and such. Now, that's for the USA - but other countries have a similar back-of-envelop multiplier.
3. Divide by 2000. This is the number of working hours in a year. This obviously varies a bit from place to place, but it's a good round number, and it's what most analysts will use. The result your per-hour cost.
4. If you need per-minute, divide by 60.

This should tell you how much a manual operation *costs the company*. Also measure how long it takes you to automate the task; that's the *investment*, and you use your salary in the calculation since you're the one who did the automating. The *return on investment* is the difference between the cost and the investment.

"This task used to cost us $6,100 a year to do manually. We spent $273.60 to automate it, reducing costs to $1,000 a year, for an annual savings of $5,100. Our first-year return on investment is $4,826.40. We win."

Expressed in those terms, automation is nearly *always* an immediate ROI. The problem IT people have in convincing management to invest in automation is that IT people never express it in those terms.

So start keeping that diary of "wins."

Not Being Selfish

Once you have become Dream PowerShell Demi-God Jedi, you will find that you have taken on a responsibility. You did not obtain Supreme PowerShell Knowledge Wizard status on your own - at the very least, you owe a lot to the product team at Microsoft and to all those people whose code you ripped off via Google.

It is time to give back. A true Shell of Power Wizard Genius is not someone who can run tasks in parallel, whose code practically *writes* itself, let alone tests itself, or who can replicate the output of `Trace-Command` in their mind. No, the true PowerShell Hero is someone who stops looking ahead in their career - for just a moment, now and then - and instead looks behind them, to see who might need a hand up.

Open-Source It

Why are you keeping your tools to themselves? *Do they suck that much?* "No," I can hear you thinking (I hear thoughts, keep that in mind), "my tools just have so much proprietary stuff in them."

Fail. Go back to milestone 1. I am disappointed and we cannot be friends until you fix this.

Tools should *almost never* have proprietary information in them, unless they're dealing purely with some internal application which nobody else will ever care about anyway. Tools should accept proprietary information *by means of input parameters*, not via some hardcoded internal whatever. Sure, your *controller scripts* (I write about those often[14]) may have some hardcoded proprietary stuff, because that's how you feed it to your tools in an automation pattern, but your *tools* should be pretty generic if you're doing it right. And if they are, you should be sharing them. Sure, okay, if you wrote them on company time there may be some question of ownership. Fine. I get it, we can be friends - **provided** you demonstrate to your company the non-secret nature of the tool's code, remind them that *you* benefited from other people's generosity, and press them to let your open-source the tool on GitHub or something. They don't have to agree. So long as you try, and bring it up again around Thanksgiving (or other appropriate regional or cultural observance) every year, fine. We can be friends.

Just Answer the Question

You could also answer questions for other newcomers. You see, there's this weird thing in the world called a *birth rate*, which means, by definition, *there are always other people less knowledgeable*

[14] http://leanpub.com/powershell-scripting-toolmaking

than you in the world. Figure out where they're hanging out, and help them. It's why I started ScriptingAnswers.com back in the day, and it's why I helped start PowerShellCommunity.org, and why I later helped start PowerShell.org. To provide a friendly, respectful place for people to ask for help. It's a *huge deal* for most adults to ask for help, especially now that ubiquitous availability of GPS apps mean we no longer have to stop at gas stations to ask for directions. When someone finally gets up the gumption to ask for help, you need to be there to help give a great, polite, professional answer.

Write All About It

You can also blog. Or write books. You don't need to be a *writer* - I mean, have you seem some of the goofy grammar going on out there? Doesn't matter - so long as you can get the words in basic enough order that people can follow, then the words are doing the job of communicating. And you may think that everything worth blogging has been blogged. Well, you are wrong. Because *your perspective* is important.

Consider how hard it is for me to teach "beginning PowerShell," for example. Not because I'm too good for it, oh no. I actually *love* teaching entry-level. I do. My problem is that it's been over a decade since I was a PowerShell newcomer, and I forget the challenges a newcomer faces. And I never experienced the challenges a *current* newcomer experiences, because in my day (ugh) the IT landscape was different. We didn't have Azure. Or iPhones, for that matter, which let me go get a cocktail, thanks for bringing that up.

So *you*, at your current point in life, at your current skill level no matter what that is… *you* have a unique and valuable perspective that *will be useful* to someone else. Offer that perspective. Help someone out.

Part 3: Faking It

OK, OK. You don't have time to learn PowerShell the right way, but you need to get through a job interview, like, tomorrow. Fine. So long as your interviewer hasn't read this book, I suppose we can do this. It'll be fun. Of course, if they've read this, they'll probably have you arrested.

All the Cool Words

First of all, because sometimes it doesn't come across well in writing, know that this chapter is *entirely* meant to be a tongue-in-cheek, poke-fun, smile-a-lot effort. Please take it in the spirit in which it's offered.

So you've got that big job interview coming up, and you know PowerShell is going to be a topic of discussion. You're completely unprepared - you haven't even finished the introduction of *Learn Windows PowerShell in a Month of Lunches*[15] and you're sweating it. No problem. Here's your guide to faking it, by using all the cool words and phrases all the other cool PowerShell kids are using these days.

Dead Puppies

INTERVIEWER: "So, how much do you know about PowerShell?"

YOU: "Well," (chuckling) "I know enough not to kill any puppies!"

INTERVIEWER: (Confused look)

YOU: "Oh, sorry, that's a big in-joke with us in the PowerShell community. It means not writing output to inappropriate places. I don't know where we come up with these sayings!"

You've now established yourself as a "member" of this "PowerShell community" (which is *not* a commune, don't make *that* mistake). +5 credibility points, because the interviewer can actually Google "powershell puppies" and verify that this is, indeed, a thing.

Parameter Sets

Get the conversation turned toward function design. "I try to avoid multiple parameter sets," you can say, "so I barely even know the syntax. You ever try to write automated tests for multiple parameter sets? It's insane. *So* hard to maintain. What was the next question?"

You've now implied that (a) you know about parameter sets, (b) you have rejected them as a design concept, (c) anyone who does use them is on meth, (d) you don't know the syntax because, like, who needs to? and (e) you know how to write automated unit tests. You've created a logical infinite loop that practically begs someone to argue with you about it, because you can just smile condescendingly at them and nod knowingly.

[15] https://www.manning.com/books/learn-windows-powershell-in-a-month-of-lunches-third-edition

The Pipeline

The big trick interview question is, "explain how pipeline parameter binding works." If someone asks you this, *then you definitely want the job*, because they likely appreciate PowerShell for all it is and does. It will also be hard to fool them.

Start with, "You mean in a non-DSC context, right?" which will confuse them, since DSC basically doesn't use the pipeline in any way that relates to what they asked. This'll give you a minute to remember the correct answer, which is covered in, "The Pipeline, Deeper" in *Learn Windows PowerShell in a Month of Lunches*. If you have not read that, you are going to need to try and move to a different topic quickly. Try saying, "well, ByValue means it's going to bind the data type, and ByPropertyName means it's going to bind properties to parameters based on them being spelled the same. By the way, that whole process is one of the things Microsoft has a patent on, isn't it? What're your thoughts on software patents in these times?" This should help move the discussion to a useless tangent on the ridiculousness of software patents, and you can throw in some thoughts on patent trolls, the US 3rd Circuit, and so on.

Continuous Integration

This is a good fallback topic. For example:

INTERVIEWER: "Can you describe to me how the formatting subsystem works?"

YOU: "Well, probably - honestly, though, I tend to really approach things from a Continuous Integration perspective, and what does the formatting subsystem really have to contribute to that? You guys *do* implement CI or CD, right?"

This can usefully turn the interview around, so that *you're* interviewing *them*. Taking control of the conversation is a great way to conceal the fact that you weren't following along in the first place.

Got suggestions for this section? Use the "Email the Author" link on this book's landing page at http://leanpub.com/become-powershell. Or, you know, read the "Feedback" section in the front of this book. Don't use Twitter. I love Twitter, but notice how everything here is longer than 140 characters? Yeah.