# <PowerShell>
## _ for Total Beginners

Jonathan Hassell

# PowerShell for Total Beginners

Jonathan Hassell

This book is for sale at http://leanpub.com/powershellfortotalbeginners

This version was published on 2015-11-28

# Tweet This Book!

*To Sara, Jacob, and Baby H, because you all are the reason I do everything*

# Contents

# Preface

Welcome to *PowerShell for Total Beginners*! I wrote this content because of one simple truth:

**If you don't know PowerShell, you're getting left behind**.

PowerShell is absolutely everywhere. It's embedded in Microsoft products. You'll find it in Exchange, SharePoint, SQL, Skype for Business and Lync, Windows Server itself, and even third party software has jumped on the bandwagon.

In fact, some Microsoft products use PowerShell as their primary management interface. Sometimes there are features to configure or settings to administer that you *just can't get to via the GUI anymore*. That's how far PowerShell has come.

So it's time to face the bottom line: if you administer or use Windows products from any sort of management perspective, **you're going to use PowerShell at some point**. Or you won't be able to do your job. It's really that simple.

If you don't learn PowerShell now, then when?

Hey, I get it. Learning a new scripting language doesn't pay the bills. You have to put out fires, remove that pesky Cryptolocker malware, increase everyone's mailbox quota by 10 GB a piece because these damn iPhones take such high resolution pictures, move a bunch of data up to Office 365, make sure everyone's Dropbox accounts are locked down–and that's only before lunch.

Except…

**Learning PowerShell will pay the bills**.

Now, I don't like speaking from a fear perspective. I want you to *want* to learn PowerShell. I want you to be convinced it's the best thing since sliced bread. (I don't know that it is, though, because I really like a good sandwich.)

The issue is, plenty of other people already made up their mind. Hiring managers have. Other employee candidates have. PowerShell experience is a requirement for many–dare I say most–job postings in this new economy. If you want a new system administrator position, or if you want to move up in the ranks to management or grow your experience elsewhere, you have to have PowerShell chops in order to fit in.

And if you're a consultant or you manage a lot of systems, **you are actually costing yourself money** by not working in the most efficient way possible. Do you bill by the hour? Do you regularly set up production systems for new clients? Do you have a script that lets you do it in five minutes, or do you sit there manually and take a whole weekend to on-board a new client?

## If you can't do your job without a GUI, you're at the end of the road.

The GUI is dead. We can have a funeral and lament the good ole days of Remote Desktop to 15 different servers at one time, or we can learn how to move forward with PowerShell. I vote for that.

Trouble is, lots of people never take the time to actually *learn* PowerShell. Rather, they just start trying to look up commands one at a time. How do you add a mailbox to Exchange? How do you add new users to your network? How can I run the same command on all ten of my domain controllers?

And so they Google and Bing for answers and they find some that are *sort of* applicable, but don't fully work in their environments. They run the examples, they fail, the error messages are cryptic, and they give up in frustration. It's impossible to learn PowerShell by sound bite, after all. And they think to themselves, "who has time to learn this stuff?"

## Learning PowerShell is hard. And it's OK to admit that.

I took the time to ask some folks what their biggest hang-ups with PowerShell were right then and there. I wanted to understand what was holding people back. I got a lot of answers, and I saw some clear themes develop. Some of the most common answers I got looked a lot like the following:

- “I’m not a developer so these scripts look like greek to me.”
- “I don’t have time to learn PowerShell well.”
- “I’m not sure how to apply PowerShell to my daily work.”
- “It’s so big! How can I remember everything?”
- “I have no idea where to find help and the documentation is poor or intimidating, or both!”

And if you’ve spent only ten or 15 minutes with PowerShell, it’s easy to see how those are your main takeaways from your experience.

But it doesn’t have to be that way.

You don’t have to be a developer. You don’t have to have a lot of time to learn PowerShell. You can apply PowerShell almost immediately. You don’t have to remember every command. And the help you actually need is right around the corner.

I can show you how, and why, all of those things are true in inside this book, *PowerShell for Total Beginners*. This book is my custom built, handcrafted, painstakingly curated book, and along with its accompanying available video series companion, is designed to get you from total PowerShell newbie to confident and open PowerShell user in as little as four weeks.

## What to Expect from This Book

The *PowerShell for Total Beginners* book I can sum up in one way:

> It’s PowerShell as I would like to have learned it.

I’m not a developer. I was a system administrator for a Research I university for a while, I was a SharePoint administrator, and I’ve been covering Microsoft and Windows Server from an IT pro perspective for 16 years now. I saw the importance PowerShell was gaining in the industry and I wanted to learn it.

I wanted to learn it so much, in fact, that I went out and bought basically every beginner resource that I could get my hands on. I bought books that promised I could learn in 24 hours, in a month of hour long sessions, in three weeks, and that I'd know everything at the end. I bought books geared toward idiots because I thought that would be a good place to start.

I'll be honest with you–every single one of them disappointed me in some way. *Every one.*

I didn't learn PowerShell well. I certainly didn't learn it in 24 hours, or even a month. Sure, I picked up bits and pieces, but the overwhelming feeling I got from these books is that they assumed too much knowledge on my part. They thought I would know what an int32 was. They thought I'd be able to figure out what modules and variables were. **And what the hell is a pipe character?**

There were some great books for the advanced beginner, **but for a guy with no development experience and no prior knowledge of PowerShell, I felt lost**.

I know why this is the case. **Most programmers are absolutely terrible at teaching programming, and yet programmers are the ones writing the books**. These guys and gals are so advanced and so in the dirt, so to speak, that they just cannot empathize with or imagine a complete beginner, a total non-programmer, non-developer wanting to learn anything. And so in their writing and their teaching they make giant leaps and bounds without even realizing it, leaving their students lost and frustrated and on the brink of giving up.

It occurred to me that I certainly couldn't be the only one who actually was a newbie with zero development or scripting experience who recognized and acknowledged the need to learn PowerShell, but who needed a serious helping hand. So, given the lack of good–well, *any* as far as I could tell–resources on the market for that demographic, I decided to take action.

I decided to create the book I wanted to learn PowerShell from.

And that's what *PowerShell for Total Beginners* is. It's a whole tome of PowerShell hand holding. If your grandmother cared anything about learning PowerShell, she could learn it from my book.

Writing this book was a labor of love. I wanted to create the authority for total beginners who needed to learn PowerShell. I wanted to create an elegant resource that assumed no prior knowledge. I wanted to create a learning point that covered every topic worth covering with a depth of explanation that is unparalleled by any book of course on the market. I wanted to create a product that would make even the skeptics of PowerShell come away at the end in love with the language and ready to take their skills to the next level. In the end, I did it and I'm proud of it.

## What's Special About This Book

In publishing, after a book had done decently well and had been in print for a year or two, we'd often ask the author to do a second edition. The author would add some material, fix some mistakes, cover some new ground we didn't have time to cover before the deadline for the previous edition, and then my esteemed colleagues at the publishing house would wrap it up, slap a new ISBN on it, and make people pay another $50 or $60 for a brand new book that, let's be honest, might have had 85% identical material to their prior purchase.

**That's not how I'm going to run my business**. First, I have systems in place to make it super easy for me to push out even minor changes to my content. Second, I don't believe in hosing my customers. Third, I want this material to always be the central resource and authority for total beginners, and that means I need to keep it fresh and alive for a long time.

That means that *anyone who purchases any of my content is entitled to all future updates to that content at **no additional charge***. Even if five years from now I'm adding chapters to *PowerShell for Total Beginners* or recording new videos to put into the course, you won't have to pay one more cent to read or watch the new stuff. You'll get an e-mail from me when updates to this material are released, and you can just come back and re-download the book and off you go.

There is no print version of this book available. The economics of printing and shipping books are well beyond my ability as a one-man show to handle. Plus, with a print book, I'd have to do a "second edition" per se in order to make changes and improve the content over time. That's silly, because in this digital age, I can just release updates to the book with a couple of mouse clicks. You'll always be entitled to the most up to date version of this content.

## Don't Wait

I've told you enough about this book. What else are we waiting for? Nothing, as far as I'm concerned. Let's get started.

# Chapter 1: Getting Started and Setting Up

Welcome to PowerShell for Total Beginners! In this chapter, my main goal is to get you set up and ready to start working. To do so, there are a couple of applications you need to learn about and just a couple of notes I'd like to clear up so that you aren't super confused as you start your march to PowerShell mastery.

## Valid Platforms and Versions

PowerShell is of course designed to be run on all kinds of machines. In fact, the broader support that PowerShell has, the more likely more developers and hardware makers will adopt the language and incorporate it into their own offerings. It is sort of like a self-fulfilling prophecy, in that the more systems that support PowerShell, the more people and systems will want to use PowerShell, which will make even more systems use PowerShell, and so on.

The two most important things to note about the systems you will use to learn PowerShell is which version of PowerShell you will be using, and what platform your computer (or computers as we get more advanced, since we will be trying to script actions and fire commands across large swaths of machines simultaneously).

To determine what version of PowerShell you have, all you need to do is open PowerShell and type a single command. Click the Start menu and just starting typing "powershell" and you should come across "Windows PowerShell." Click that to open it up and you should see what looks like a text based prompt. Type the following at that prompt:

```
$PSVersionTable.PSVersion
```

It should spit out a little table with four columns: Major, Minor, Build, and Revision. All you care about is the number in the Major column. That is the column that tells you what version you have. You need to have version 3 or more to get the most from this book. PowerShell updates are free to download, so if you find you are using a version earlier than version 3, go over to Microsoft.com and search for PowerShell and you'll find some pages that will help you download the new version and get it installed on your system. It just so happens that PowerShell 3 was a pretty big release, and a lot of new features were added to make life easier in a lot of ways, so it is a good baseline we can use for teaching. As I write this, Windows 10 ships with PowerShell version 5, so there have definitely been later revisions of the language, but they have mostly added more advanced features and scripting support and almost nothing has changed for the total beginner. So get to version 3 and then you are set on this point.

As far as your system's platform goes, this is only a little bit trickier to figure out. Computing as we know it today is divided into a couple of camps:

- **x64**: 64-bit computing is pretty much the cutting edge of what you can buy today and really enables you to have gobs of RAM. PowerShell generally assumes you are running 64-bit, and pretty much everything you come across in this course or even on the Internet as you start searching for command and scripting ideas and then borrowing them (ahem) for your own personal use will work. If you have bought a machine within the last five years, then you have a 64-bit machine, and correspondingly you have nothing to worry about.
- **x86**: this is the platform of yesteryear. If you are running on machines purchased before 2007, then you probably are still 32 bit. If you have less than 4 GB of RAM, there is a chance that you are running a 32-bit operating system. PowerShell still works on this platform, but some more advanced stuff may run different and some commands actually may not work at all. It is still plenty good enough to learn PowerShell with, so do not think in order to go any further you have to go buy a whole new PC! Yours will work fine. All you need to do is use the x86 version of the Windows PowerShell console or the Windows PowerShell Integrated Scripting Environment—you will know which one because it specifically has "(x86)" in the title. But do not use this version unless you have to.

Again, to keep it simple, use the 64-bit version of PowerShell (the one that does NOT have the (x86) designator in the title) unless you have a really specific reason why you need to use the 32-bit version.

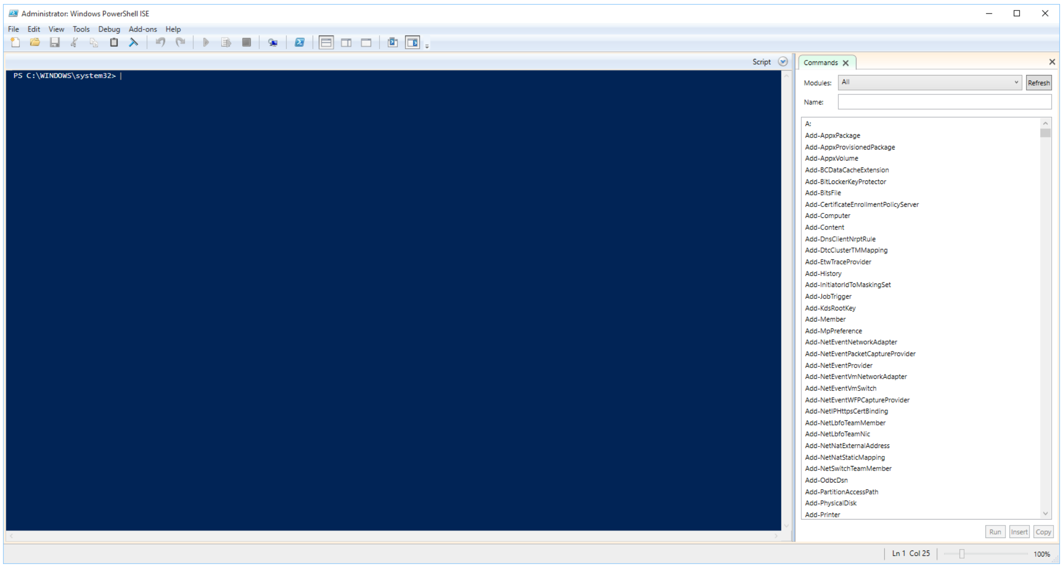
Are you on version 3 and you know which platform version of PowerShell to use? Fantastic! Let's move on.

## Two Important PowerShell Tools

To use PowerShell, there are basically two avenues you can take: you can use a console window which is basically just a place to type in commands and read the responses the system sends back to you, or you can use a graphical tool that lets you type in the same commands but also has a few more features to make it a little easier to explore PowerShell. The two are not mutually exclusive; you can use either or both depending on the task. Let me show you each of them, and while I am taking you through each of the utilities I will mention some sample situations in which each of them would be useful and also when to think about using the other one.

### The PowerShell Integrated Scripting Environment (ISE)

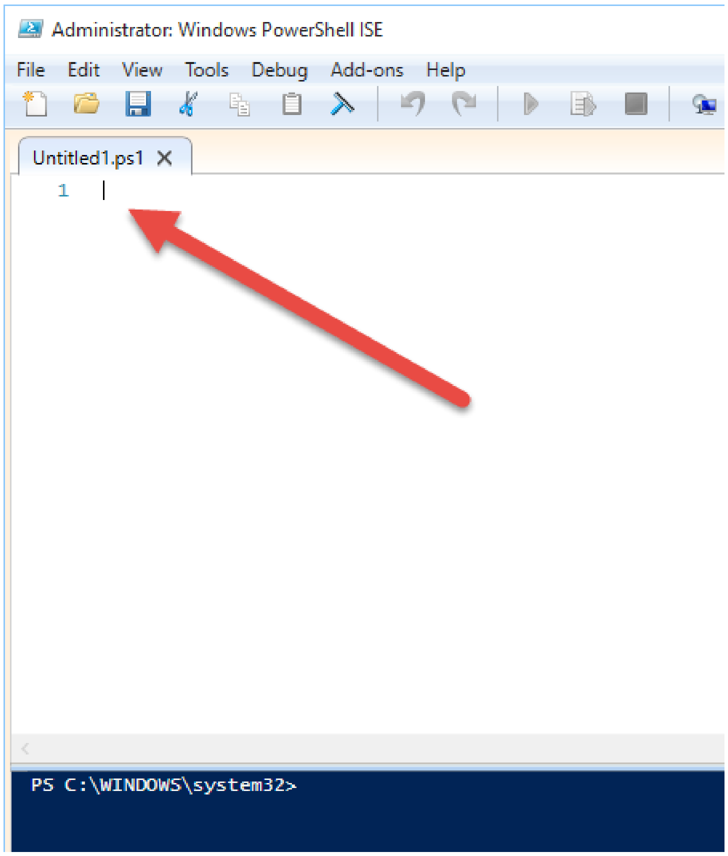
I think the easiest way to get started with PowerShell is to have a little bit of hand holding, and the Windows PowerShell Integrated Scripting Environment, which you will sometimes see labeled as the Windows PowerShell ISE, is the best way to get started without spending any money. Figure 1-1 shows the way the Windows PowerShell ISE looks out of the box.



```
PS C:\WINDOWS\system32> |
```

*Figure 1-1. The default look of the Windows PowerShell ISE.*

The main idea behind the ISE is to help you build scripts. You can do it in a friendlier, more aesthetically pleasing environment with a lot of help both in your face and behind the scenes, too. It gives you the console window—that big blue area that looks like a DOS prompt and kind of has the same look as the window you opened earlier in this chapter to check the version of PowerShell you are running—and also on the right, it gives you some graphical help in finding and selecting the best commands to use for any job. However, there is also the scripting pane, which you can turn on by going to the View menu and selecting Scripting Pane. Here, in Figure 1-2, is what that adds to the equation.


Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Untitled1.ps1

```
PS C:\WINDOWS\system32>
```

*Figure 1-2. Adding the scripting pane to the Windows PowerShell ISE.*

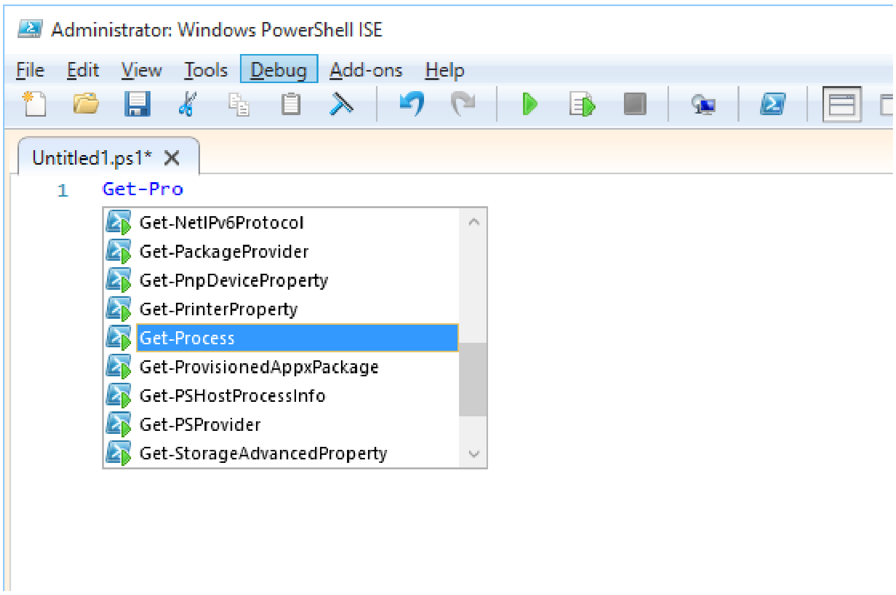
## IntelliSense

It is easy to get started with PowerShell using the ISE because it includes some cool features. The best one for beginners is called IntelliSense, and it is like a PowerShell expert standing behind you, looking over your shoulder helping you pick the right commands for the job. You can try it easily—just type in

```
Get-Process
```

...into the scripting pane. You will see that as you start typing, the ISE will helpfully pop up a menu of choices that you can use. I have put an example of this in Figure 1-3.

If you see one you like, you can simply hit the Tab key and the ISE will automatically type in the rest of the command for you. To scroll up and down in the list, you can use the arrow keys if you do not want your fingers leaving the keyboard, or you can use the mouse to scroll up and down via a center wheel on your mouse or via the up and down buttons.



```
Get-Pro
```

```
Get-NetIPv6Protocol
Get-PackageProvider
Get-PnpDeviceProperty
Get-PrinterProperty
Get-Process
Get-ProvisionedAppxPackage
Get-PSHostProcessInfo
Get-PSProvider
Get-StorageAdvancedProperty
```

*Figure 1-3. IntelliSense in the Windows PowerShell ISE will help you narrow the universe of commands from which you can choose.*

Once you have finished the command you want to run, either hit the F5 key on your keyboard or click the green right arrow up in the toolbar that is a little to the right of the Help menu. What you have written will run in the console window and you will see the results there, too.

The IntelliSense feature also works for the potatoes part of the command (the command itself is the meat, but the potatoes are all the stuff that follows the command; you'll learn about these in the next chapter), too. For instance, and just take my word for this now, there is a command called `Get-EventLog` that will look inside the event logs that Windows keeps to track what happens on your system. Part of that command is specifying which event log you want to look at, and since Windows can generate and log a lot of events, you probably are only ever interested in a pretty small subset of all of the events available in any one event log. So again, just taking my word for it now—I will show you how to puzzle through finding commands and thinking about what command to use when in this book, so fret not—the command we want to use is

```
Get-EventLog -LogName Security -Newest 10
```

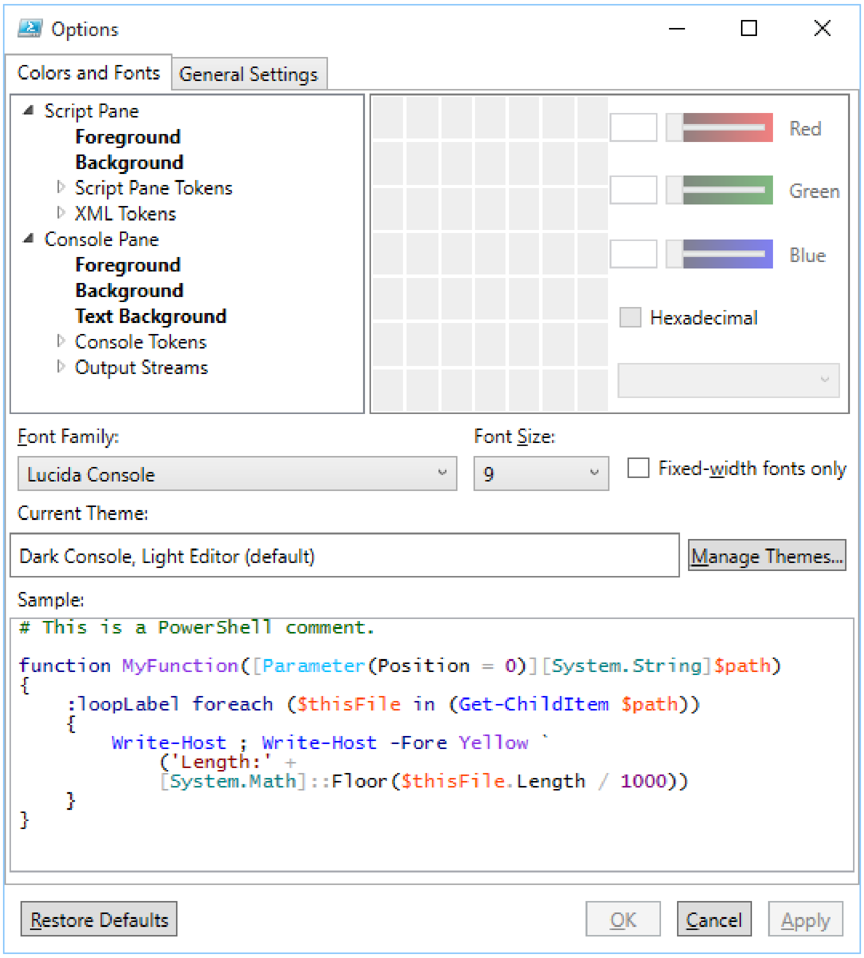
Take that command and start typing it into the scripting pane of the ISE, but as you go, keep hitting tab. As you hit Tab, you will see a list of appropriate choices to type in each place of the command. For instance, as you type `-LogName`, you will see the other parameters you can enter come up. As you type `Security` and hit Tab, you will see a list of event logs from which you can choose. This sort of visual representation of the universe of PowerShell, but put into a really context specific perspective, can be invaluable as you are learning PowerShell and getting your head around it.

The scripting pane is also really useful because you can use it as a kind of notepad or staging area. You can use the scripting pane to write a script or store a couple of commands, and then to test it out, you can select a part of what you have typed into the scripting pane and run only that. You do this by hitting F8 on your keyboard, or choosing the icon in the toolbar that has the green right arrow with the document image behind it. Hover over that icon to make sure you are choosing the right one. That selection will run via the PowerShell engine in the console window below (again, the blue area by default), and it is a great way to interactively build a script over time.

Believe it or not, a unique feature of the ISE is that it supports copy and paste with the same keyboard shortcuts you have been using for years. Plus, you can copy and paste not only from the scripting pane, but also from the console window, too, which unless you are running Windows 10 is kind of a difficult thing to do. You would think since Windows has been around for 30 years that Microsoft would have figured out how to make that work, but you would be wrong if you did think that.

## Getting the ISE Set Up

Frankly, I think the ISE looks pretty good out of the box. There is not a lot I would recommend that you change, but you might want to adjust the font sizes for both the scripting pane and the console pane if your eyes are getting old and tired like mine. This is fairly easy to do: just go to the Tools menu and select Options, and you'll get a window like the one in figure 1-4 where you can adjust the font, dize, and color of just about every aspect of the ISE. Play around until you find something you are comfortable with and that you can read easily. This last point is important because in PowerShell, syntax is absolutely critical and sometimes single and double quotes and tildes mean different things. You must be able to easily see the differences in characters or you will spend hours scratching your head and troubleshooting when things go haywire.



Font Family: Lucida Console

Font Size: 9

☐ Fixed-width fonts only

Current Theme: Dark Console, Light Editor (default)

Sample:

```
# This is a PowerShell comment.

function MyFunction([Parameter(Position = 0)][System.String]$path)
{
    :loopLabel foreach ($thisFile in (Get-ChildItem $path))
    {
        Write-Host ; Write-Host -Fore Yellow `
            ('Length:' +
            [System.Math]::Floor($thisFile.Length / 1000))
    }
}
```

*Figure 1-4. Adjusting the aesthetics of the Windows PowerShell ISE.*

If you don't care about any of the colors but you just want to make things bigger, you can do that with the zoom slider in the lower right corner. This is what I do. 185% works for me. You can see this in Figure 1-5.

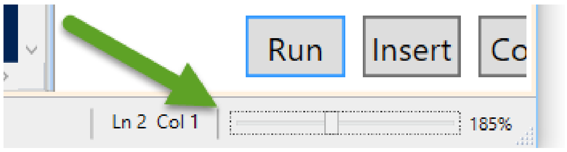


*Figure 1-5. The Zoom slider in the lower right corner of the Windows PowerShell ISE window at 185%*

## The Windows PowerShell Console

If the PowerShell ISE is the friendly neighborhood bar and grill, the PowerShell console is the median of a freeway—you had better know what you are doing and you will get steamrolled by something moving 70 miles per hour. Well, that is not entirely true, because traffic is universally horrible everywhere (OK, so maybe that part is not true, either), but the PowerShell console is definitely where you go to get business done with no frills. It is a place where you should know what you are doing.

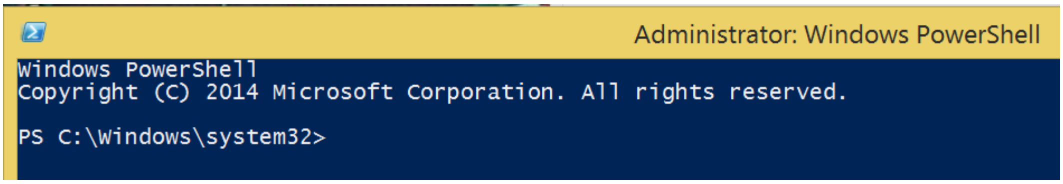
The PowerShell console is also generally installed on all machines that have PowerShell installed. The ISE can sometimes be missing from systems, particularly on servers that do not really have a lot of users log on to their desktops and work on them directly, but the console you can always rely on being present when you need it. It is also the easiest way to simply bang out a bunch of commands and then move on, so if you know what you want to run but you do not have those commands enclosed in a script yet, then the PowerShell console is the place you will execute those.

As I mentioned, the console is totally no frills. You can cut and paste, but only with some weird keystrokes and frankly they do not always work. The Paste most often works via a right mouse button click when you have already cut a command from another place like a text file. The cut and copy from within the console are difficult and frankly do not always work. I don't know why this is. It's frustrating, although Windows 10 seems to have improved on this somewhat. You also do not get IntelliSense, but the console does support using Tab to get some hints and suggestions for commands and the "potatoes" after the command. The difference in the console is that those suggestions do not pop up as a little graphical context menu; instead, they appear already typed out in the command line and you need to hit Tab to keep cycling through until you hit the one you want. You can also use the Shift key and hold down Tab to reverse cycle through, so if you pass a selection because you were cycling too fast, you do not have to go around the whole "wheel" of selections again.

## Running the Console as an Administrator

The first thing we should talk about is security. Cue the collective groaning; security sucks. I get it. But it is important to understand that PowerShell respects your system's security and integrity, in particular because now that almost every administrative function within Windows is accessible via a PowerShell cmdlet, you can really make irreversible changes to your system with just a couple of commands. (If you want to see what I mean, take a system that you do not care about, open the PowerShell command prompt, and then type in `Get-Process | Stop-Process` and count the number of seconds it takes before you see that lovely blue screen of death.) For this reason, PowerShell limits the number of system modifying things you can do as a normal user. To get access to the really powerful stuff, you need to run as an administrator.

This is pretty easy to do. From the Start menu, type in PowerShell, and then on the item that appears, right-click on it and select Run as administrator. You might be prompted to enter a password, depending on how your computer's security is set up and whether you are connected to an Active Directory domain at work or you're using your home computer, or you might not. Either way you will see PowerShell running as an administrator after you satisfy the prompt for credentials. You can tell you have successfully run the PowerShell console as an administrator by looking at the title bar for the application; it will show Administrator to show you are running with what the security professionals call "elevated privilege."



```
Windows PowerShell
Copyright (C) 2014 Microsoft Corporation. All rights reserved.

PS C:\Windows\system32>
```

*Figure 1-6. Running the Windows PowerShell console with administrative privileges.*

I'll be honest with you: I'm of a mixed opinion about running PowerShell as an administrator. For instance, it's more convenient when you are learning PowerShell and its associated syntax because you only have to worry about structuring the command correctly. You don't need to spend brain cycles trying to figure out why something's working only to discover it is because of a system policy or a lack of privileges that is preventing a well structured, syntatically correct command from running. It is also more convenient to run as an administrator on a single machine–you would largely want to avoid running with elevated privileges by default if you were using the PowerShell remoting feature and trying to manage vast swaths of machines at the same time, but in a single machine and especially in a test virtual machine or a throwaway desktop without production sensitive materials on it, you will find PowerShell to be a lot less whiny about permissions errors when you run it as the king per se.

None of that is to say there are not downsides, of course. There are two sides to every coin. First of all, it is patently *insecure,* mainly because you are removing most of the built in protections Windows has to make sure vulnerable parts of the system do not get hacked or torn apart. Secondly, it may be practically impossible for you to actually run as an administrator on your work computer simply because you may not have the credentials on your network to support this. Finally, running as an administrator does dull your awareness to which commands require privilege and which commands do not. This can sometimes lead to bad scripting and poor command structure simply because running as an administrator glosses over permissions issues, as I noted above.

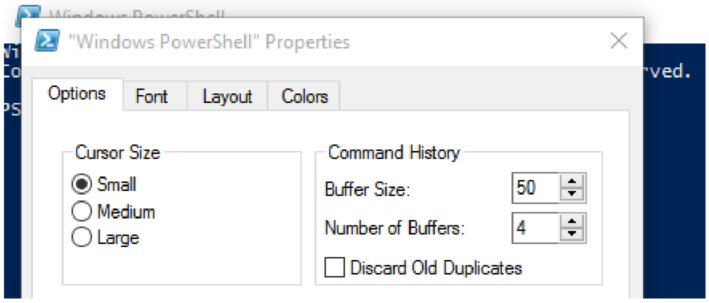
I will leave it to you to decide how you want to proceed on this issue during your training, but let me be **very clear** about using PowerShell in production:

> Never, ever run commands as an administrator in the PowerShell console in production unless you have a very, very good reason to do so. You may break things, inadvertently repave systems, set off a nuclear war, and more.

## Setting Up the Windows PowerShell Console

There are a few tweaks you can make to the console to increase visibility and make it a little more comfortable to use on a daily basis. Here are some of them.

- **Increase your command buffer**. The command buffer is basically a database or a big list held in memory of all of the commands you have entered into the console during the current session; in other words, since the last time you opened the console. To access the command buffer, from the command line, just hit the up arrow and down arrow keys on your keyboard at an empty prompt to cycle through the entries in the buffer. This can be useful if you make an easy to fix typo in a long command, since you can just hit the up arrow to get it back and then change the error and re-run the command without having to type the whole thing in again. What I would recommend is to increase the size of this command buffer. You can do this by clicking the top left corner of the title bar, where the little icon is, selecting Properties from the pop-up context menu, and heading over to the Options tab. Take a look at Figure 1-7. Increase that buffer size to at least 100 to give you more room to maneuver, especially in long sessions where you might be troubleshooting or puzzling out a tough issue. You might need that "breadcrumb" trail.



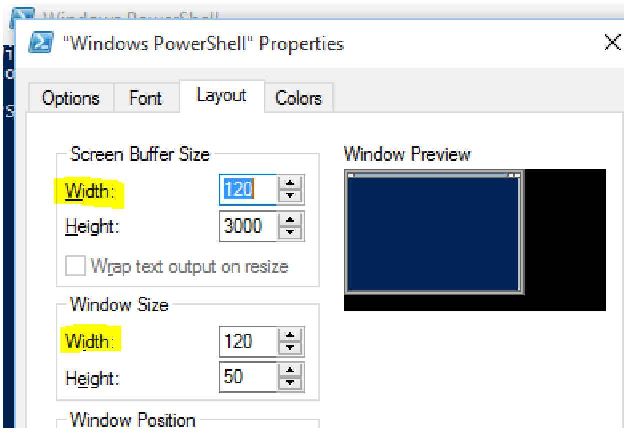
Cursor Size
[✓ Small] Medium Large

Command History
Buffer Size: 50
Number of Buffers: 4
☐ Discard Old Duplicates

*Figure 1-7. Increasing the Windows PowerShell console buffer size.*

- **Change the font in the console**. A nice big font is easier on the eyes and certainly makes shorter work of finding typos. Head over to the Font tab in that same dialog (remember, click the top left corner of the title bar, where the little icon is, select Properties from the pop-up context menu, and click Fonts.) I quite like Lucida Console at size 18, but my eyes are not great. Experiment to find what works for you.
- **Customize the width of the contents of the console window**. Head to the next tab, conveniently labeled Layout. Some things to change here: make the width sizes in both the screen buffer size section and the window size section the same. Figure 1-8 will show you how. This will ensure that all of your text fits on screen to eliminate any scroll bars in the window. Sometimes in long commands or commands with a lot of output, some of that output prints off screen so it's important to eliminate scroll bar if you can. It can be tough to

troubleshoot commands that are not working if you can't even see the output of a command.



Screen Buffer Size
Width: 120
Height: 3000
Wrap text output on resize

Window Size
Width: 120
Height: 50

Window Position

*Figure 1-8. Increasing the Windows PowerShell console widths.*

- **Set up high contrast colors for easier viewing**. Finally, on the Colors tab–which is the next one over–customize the colors, but make sure you choose something that contrats well so that you don't lose track of text on the screen. For most people, the default blue background and white text works well, but for old school folks and storied administrators, a black screen with green text evokes fond memories of VAX and AS/400 machines. You choose what works for you.

## Running and Stopping Commands in the Console

Once you have the console all customized to your liking, you can begin running commands. A quick tour of how this works: you type the command at the prompt, just like you would in a DOS command line environment. Hitting Enter will execute the command, just as you would expect.

Using Ctrl-C will *not* copy text like it would in Word or any other common Windows application. Instead, Ctrl-C in the PowerShell console stops the current command from running. This is most useful when you mistakenly run a command without including the right stuff after the main part of the command; PowerShell will notice the required information is missing and it will prompt you for it, and when you see this you can just hit Ctrl-C to cancel out of the command and start over. This is also useful if you are reading a help page, which for some commands can stretch over 10 or 20 screens. Once you have read to the section you wanted and found the information you were looking for, you can hit Ctrl-C to "exit" the help environment (which is really stopping the `Get-Help` command, but more on that as we get deeper into the book) and get back to the console prompt.

## Windows PowerShell Console Caveats

There is never a perfect solution for any one problem, and that maxim continues to be true with PowerShell. While sometimes a crunchy terminal window and a will to get stuff done is all you need, other times that same window will be a pit of despair, like looking over an abyss in moonlight. (The howling you hear is the simile I have just stretched into oblivion.)

First and foremost, if you are one of my international readers–hello and welcome to you!–the PowerShell console is not well localized. It does not display non-English languages will at all. Secondly, as I discussed in the previous section, copying and pasting work a little different in the console window. Ctrl-C as you know now stops a command from executing, and Ctrl-V works generally as you would expect to paste the contents of the clipboard into the prompt. However, for some reason Ctrl-V doesn't always work, so a shortcut I have found particularly useful is to use the right mouse button–just click anywhere inside the console window with the right mouse button and the contents of the clipboard will be pasted into the current prompt entry. But still, it's not as seamless as you might expect it to be.

One of the biggest differences in actually running commands one by one or in smaller batches is that the console does not include IntelliSense like the ISE does. It only allows for discovery and typing shortcuts via hitting the tab key, known as "tab completion."

## Tab Completion

"What's that about tab completion?" you say. Well, consider it to be the little brother of the IntelliSense feature in the Windows PowerShell ISE I covered in the last main section. The best way to show you is via a hands-on demonstration. Let's take a slightly different version of the command we were using earlier:

```
Get-EventLog -LogName Application -Newest 5
```

Start typing that into a fresh PowerShell console window. After you type `Get-Event`, hit Tab. You'll see PowerShell fills in the `Log` part. Then add the potatoes: type in the hyphen and then hit Tab again, and then again and again. You can see PowerShell cycling through the available choices. You can also use Shift-Tab to go backwards in the cycle in case you're hitting the keys too quick and miss the choice you need.

Some PowerShell users feel like this is tougher to use than the ISE when one is learning PowerShell because you don't get that pop-up window of choices in the console. Some feel like tab completion is helpful mostly only if you already have an idea about what choices are out there, kind of like the old saying about the massive IBM Redbook technical manuals, that they were "clear only if previously known." I'm not sure I agree with that, but I do see where a menu would be easier to visually grasp. But that's why you have the ISE. For those quick one or two commands, especially when you find yourself getting more adept at PowerShell syntax, the tab completion feature in the console can save you a lot of typing and potential carpal tunnel syndrome.

## The Last Word

As I mentioned, my goal for this first chapter of *PowerShell for Total Beginners* was to get you set up and ready to start working. We looked at the PowerShell ISE, the PowerShell console, how to make sure you're on the right platforms to get started with PowerShell, and how to use some of the features of each of these environments to make it easier to get your work done and also to support your learning of PowerShell. In the next chapter, we'll dive into the meat of PowerShell and start learning the basics. There is, after all, no time like the present.