ITPro Today™

Tech Insights

A Beginner's Guide to Microsoft PowerShell

Want to become a PowerShell expert? This guide explains the essential concepts and skills for getting started. Learn about PowerShell commands, scripts, and more.

By Brien Posey

Brought to you by informa tech

NEXT

TABLE OF CONTENTS

INTRODUCTION

Ask any PowerShell user whether Microsoft's automated task and configuration system is worth learning today, more than likely you'll receive an enthusiastic, "YES." Debuted in 2006, PowerShell remains an indispensable tool for automating critical tasks, creating scripts and applications, and much more.

This guide is designed to get you familiar with the features of PowerShell, starting with its basic command structure. From there, you'll pick up foundational PowerShell commands (or "cmdlets") to get you moving, as well as how to run PowerShell scripts. Finally, you'll learn how to create your own PowerShell functions. ITPro Today's PowerShell expert Brien Posey provides easy-to-follow instructions, script examples, and visual demos along the way.

5 Tips for Learning PowerShell

PowerShell endures as an automated task system used by countless IT professionals. Here are tips on how to learn PowerShell — plus, resources to jumpstart your education.

Learning Microsoft's PowerShell system can be a daunting process, particularly for those who tend to avoid command line environments. Even so, there are tricks you can use to make the learning process easier.

In this article, I'll provide tips on how to learn PowerShell so you can get up and running quickly.

Check Out Video Tutorials

Arguably, the easiest way to learn PowerShell is through videos. I recently finished recording a 12-hour PowerShell video course that will launch soon. However, there are plenty of other courses available.

If you prefer the free option, check out YouTube. YouTube offers lots of videos on the subject.

Understand the Basic Syntax

Putting aside video-based tutorials, I would recommend starting out with PowerShell's basic command structure. While there are thousands of PowerShell commands, nearly all of them adhere to the same basic syntax rules. Once you learn how those syntax rules work, you can figure out nearly any command.

Incidentally, Microsoft refers to the native PowerShell commands as cmdlets (pronounced "command-lets"). Cmdlets consist of two words: a verb and a noun. The verb and noun have a dash in between them (for example: Copy-Item).

PowerShell cmdlets tend to use familiar, everyday language because Microsoft wanted to make it relatively easy to figure out which cmdlet to use in any situation.

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

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Brien> Get-Process

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)      Id  SI ProcessName
-------  ------    -----      -----     ------      --  -- -----------
    454      28    32280      42708     130.59   1204   1 ApplicationFrameHost
    152       9     1580       5012              11936   0 AppVShNotify
    153       8     1656       5076       0.00   11964   1 AppVShNotify
    314      17     4064      18908               3192   0 armsvc
    419      25     8896      25988       0.69    8036   1 ASHelper
    702      46    93320      49640       4.38    2224   1 BlueJeans.Detector
    503      32    40044      25132       5.33   12824   1 BlueJeans.Detector
    159      11     1788       5152               4496   0 BtwRSupportService
    566      30    45860      51036       1.59    8588   1 Calculator
    269      26     6516       9828     101.55   10948   1 CardLauncher
    181      14     2580       7988      66.88    9392   1 CMFNSS6
    284      15     4544      17796       0.20    1848   1 conhost
    110       8     6340       9052               7084   0 conhost
    125       9     6688       9984               9912   1 conhost
    900      28     5116       5112                864   1 csrss
    888      28     2492       4036                992   0 csrss
    582      18     7376      18428     166.14    9652   1 ctfmon
    172      11     2404       6016               5428   0 dasHost
```

Figure 1: The Get-Process cmdlet retrieves a list of system processes.

Here's an example of how to use a PowerShell cmdlet. Suppose that you wanted to see a list of the processes running on your computer. In this case, you would use the Get-Process cmdlet, shown in Figure 1. Here, ***Get*** is the verb and ***Process*** is the noun.

So, with that in mind, what command might you use to display a list of the services running on your computer? The cmdlet that would be used is Get-Service.

Notice that both the Get-Process and

Get-Service cmdlets use the word "Get." PowerShell cmdlets that include the Get verb are always used to retrieve some piece of information.

Just as verbs such as Get play a consistent role across a variety of cmdlets, so too do nouns. I showed that Get-Service retrieves a list of services, but what cmdlet might you use to ***shut down*** a service? The cmdlet would be Stop-Service (you would also have to supply the name of the service that you want to stop). In other words, all the cmdlets that are designed for actions related to system services will use the Service noun. Similarly, cmdlets that interact with system processes use the Process noun.

Get Familiar With Parameters

Most PowerShell cmdlets also accept various parameters. The required and accepted parameters of each cmdlet is a little bit different because the parameter must be based on the cmdlet's purpose. Even so, you will find that while some parameters are unique to a particular cmdlet, others are used again and again.

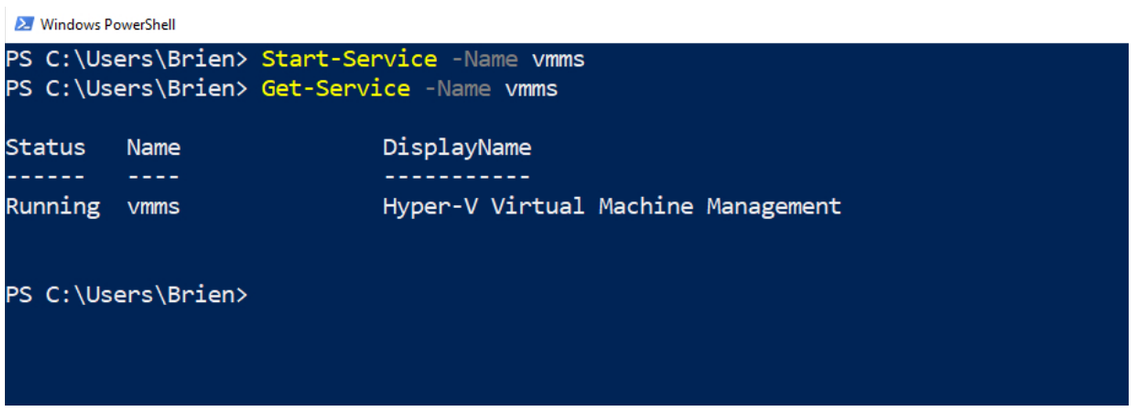


Figure 2: Many PowerShell cmdlets use the Name parameter.

I noted above that if you want to stop a service, you must supply the name of the service that you want to stop. The name is entered with the Name parameter. Generally speaking (and there are exceptions), anytime that you must supply a named value to a PowerShell cmdlet, you can usually use the Name parameter to do so. That holds true whether you tell PowerShell the name of a service, process, virtual machine, or just about anything else.

Of course, this raises the question: How can you know which parameter goes with which cmdlet?

One option is to consult the documentation. Microsoft has a dedicated webpage for each cmdlet. The page lists all the cmdlet's parameters and provides examples of how to use the cmdlet.

PowerShell Integrated Scripting Environment (ISE) can also help. As you

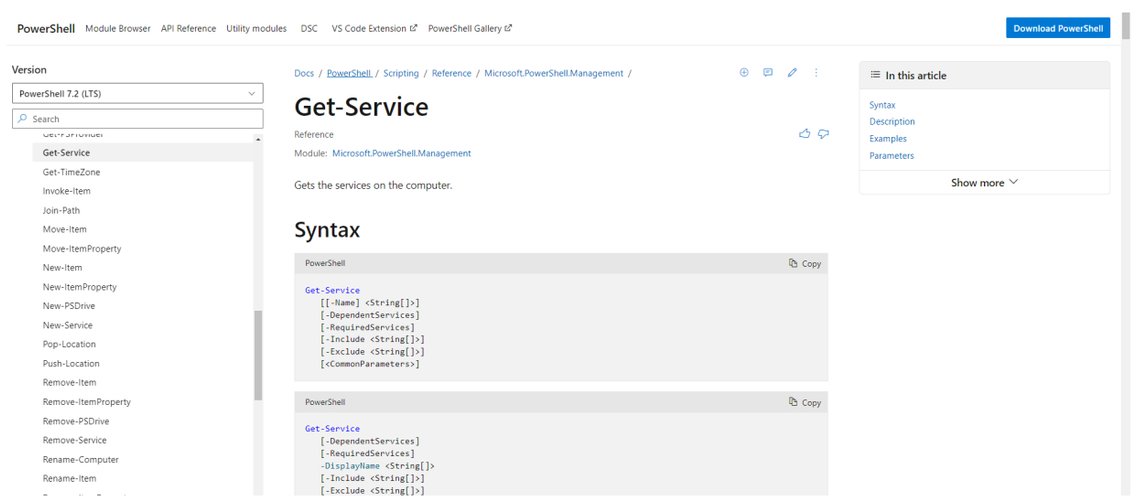


Figure 3: Microsoft provides documentation like this one for every PowerShell cmdlet.

type the name of a cmdlet, PowerShell ISE supplies you with a list of the parameters that you can use, as shown in Figure 4.

You can always type Get-Help, followed by the name of the cmdlet you need help with (notice that the Get-Help cmdlet uses the Get verb, shown in Figure 5).

Keep It Safe

When you learn PowerShell, it involves more than just memorizing a few cmdlets and understanding what parameters to use. The only way to truly learn PowerShell is by experimenting with the various cmdlets. However, before doing so, I recommend setting up a dedicated physical or virtual machine to use solely for practicing PowerShell.

The reason you should use a practice environment is because PowerShell is both a programming language and a Windows management tool. As such, some of the cmdlets can be destructive if used improperly. Creating an isolated environment lets you experiment with PowerShell without jeopardizing your production environment.

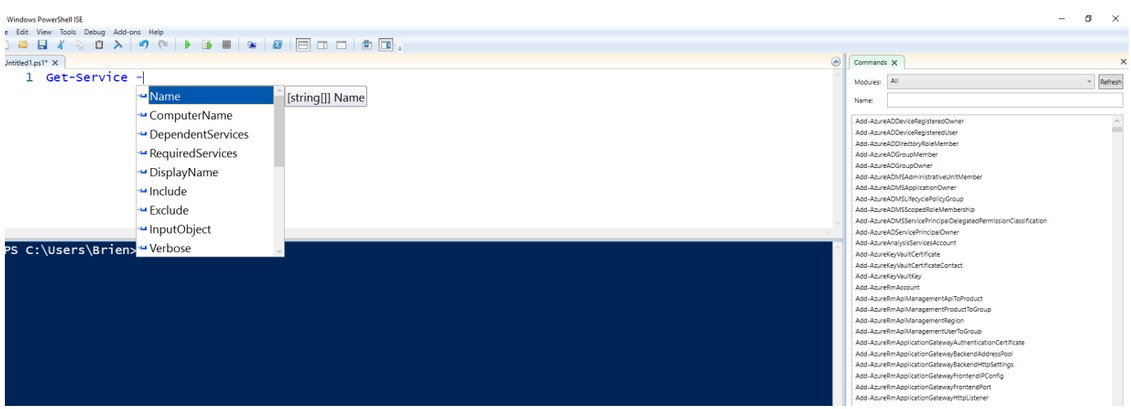


Figure 4: PowerShell ISE provides help as you type a cmdlet.

Learn From PowerShell Users

Finally, perhaps the best thing you can do to learn PowerShell is to learn from others.

Suppose that you want to know how to perform a particular task, such as displaying system processes. You could Google terms such as "PowerShell cmdlet for listing processes" or "how do I list processes in PowerShell." Google will usually serve up advice from other PowerShell users. Depending on the complexity of the task, you might not always totally understand the solution. However, you can always look up any of the individual cmdlets that are used in the solution.

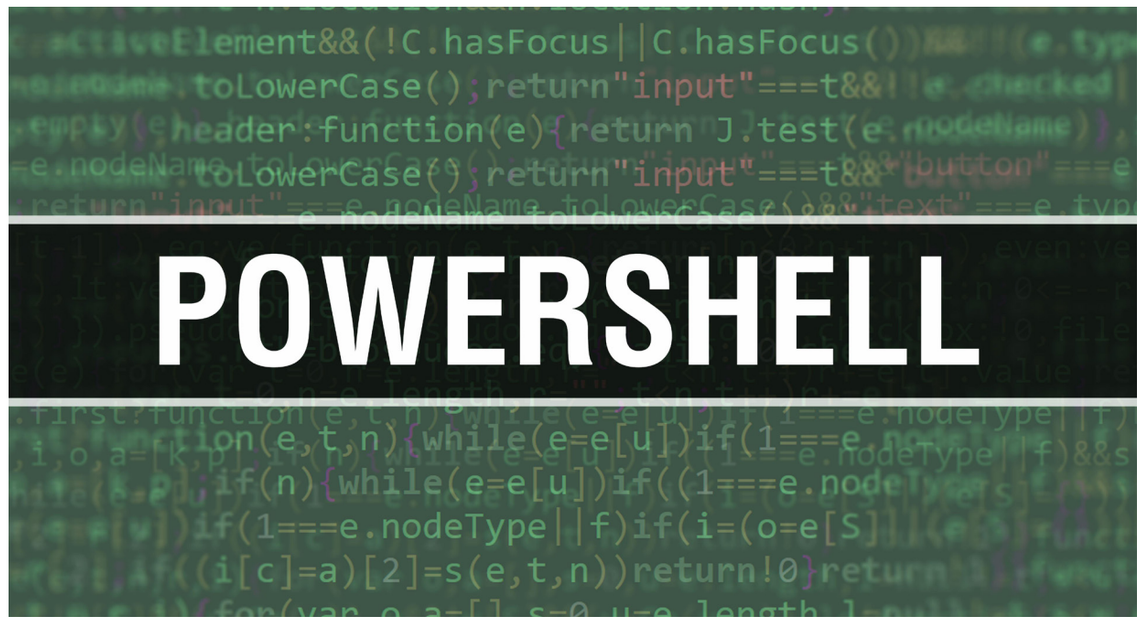
You can also experiment with changing the values assigned to the various parameters used. By doing so, you will be able to get a better feel for how the cmdlets work. ■

```
PS C:\Users\Brien> Get-Help Get-Service

NAME
    Get-Service

SYNTAX
    Get-Service [[-Name] <string[]>] [-ComputerName <string[]>] [-DependentServices] [-RequiredServices] [-Include
    <string[]>] [-Exclude <string[]>]  [<CommonParameters>]

    Get-Service -DisplayName <string[]> [-ComputerName <string[]>] [-DependentServices] [-RequiredServices] [-Include
    <string[]>] [-Exclude <string[]>]  [<CommonParameters>]

    Get-Service [-ComputerName <string[]>] [-DependentServices] [-RequiredServices] [-Include <string[]>] [-Exclude
    <string[]>] [-InputObject <ServiceController[]>]  [<CommonParameters>]


ALIASES
    gsv


REMARKS
    Get-Help cannot find the Help files for this cmdlet on this computer. It is displaying only partial help.
        -- To download and install Help files for the module that includes this cmdlet, use Update-Help.
        -- To view the Help topic for this cmdlet online, type: "Get-Help Get-Service -Online" or
           go to https://go.microsoft.com/fwlink/?LinkID=113332.
```

Figure 5: Use Get-Help for help using any PowerShell cmdlet.

What Are the Basic PowerShell Cmdlets

Newbies to PowerShell can benefit from learning a handful of basic commands. Here are six PowerShell cmdlets you should know.



When getting started with PowerShell, many people can be intimidated by the sheer number of cmdlets available. The total number of cmdlets varies widely based on the Windows version you run and on how Windows is configured. Even so, it is not uncommon for a Windows deployment to support over 12,000 cmdlets!

The good news is that you don't have to learn all these thousands of cmdlets. You can start out by learning just the essential ones.

There is no official list of "basic" cmdlets. Even so, there are a handful of cmdlets that most experts would agree are foundational. These cmdlets are very commonly used in conjunction with other cmdlets.

PowerShell cmdlets that you should know include the following:

Get-Command
Get-Help
Clear-Host
Set-Location
Get-ChildItem
Get-Alias

Get-Command

When used by itself, the Get-Command cmdlet will display a list of every known PowerShell cmdlet. Typically, this list is in the thousands.

While Get-Command might not seem particularly useful, it's important that you know it. That's because it can help you to find the cmdlet you need in any situation.

Most PowerShell cmdlets are made up of two words, a verb and a noun, which are separated by a dash. In the case of Get-Command, ***Get*** is a verb and ***Command*** is a noun. A verb or a noun that is used in one cmdlet is often supported by other cmdlets, as well. The verb ***Get***, for example, is used in a huge number of cmdlets. ***Get*** usually instructs the cmdlet to return a list of information (examples: Get-Users, Get-Process, etc.).

If you want to know all the cmdlets that support ***Get***, just type:

Get-Command Get-*

Get-Help

Whereas Get-Command can help you to find the name of a specific cmdlet, Get-Help can show you how to use a cmdlet.

To use Get-Help, just type the command followed by the name of the cmdlet that you need help with. For example, if you want to know how to use Get-Command, you would type this:

Get-Help Get-Command

You can see what this looks like in Figure 1.

```
PS C:\Users\Brien> Get-Help Get-Command

NAME
    Get-Command

SYNTAX
    Get-Command [[-ArgumentList] <Object[]>] [-Verb <string[]>] [-Noun <string[]>] [-Module
    <string[]>] [-FullyQualifiedModule <ModuleSpecification[]>] [-TotalCount <int>] [-Syntax]
    [-ShowCommandInfo] [-All] [-ListImported] [-ParameterName <string[]>] [-ParameterType
    <PSTypeName[]>]  [<CommonParameters>]

    Get-Command [[-Name] <string[]>] [[-ArgumentList] <Object[]>] [-Module <string[]>]
    [-FullyQualifiedModule <ModuleSpecification[]>] [-CommandType {Alias | Function | Filter |
    Cmdlet | ExternalScript | Application | Script | Workflow | Configuration | All}]
    [-TotalCount <int>] [-Syntax] [-ShowCommandInfo] [-All] [-ListImported] [-ParameterName
    <string[]>] [-ParameterType <PSTypeName[]>]  [<CommonParameters>]


ALIASES
    gcm


REMARKS
    Get-Help cannot find the Help files for this cmdlet on this computer. It is displaying
    only partial help.
        -- To download and install Help files for the module that includes this cmdlet, use
    Update-Help.
        -- To view the Help topic for this cmdlet online, type: "Get-Help Get-Command -Online"
    or
           go to https://go.microsoft.com/fwlink/?LinkID=113309.
```

Figure 1: The Get-Help cmdlet is used to display the syntax of any other PowerShell cmdlet.

Clear-Host

Clear-Host is one of the easiest cmdlets to use. It's used to clear the screen. Hence, if you need to clear the screen, just type Clear-Host.

As a shortcut, you can also use the CLS command to clear the screen. CLS is a leftover from the days of DOS and is an alias to the Clear-Host cmdlet (more on aliases in the Get-Alias section below).

Set-Location

The Set-Location cmdlet is used to navigate the file system. It can be used as an alternative to the CD command (CD is a DOS leftover and stands for Change Directory).

To use Set-Location, just append the -Path parameter, then provide the path to go to. For example, if I wanted to navigate to C:\Users\Brien, I would type this:

Set-Location -Path "C:\Users\Brien"

The next figure provides an example of Set-Location. Incidentally, Set-Location can also be used to navigate the Windows registry.

Get-ChildItem

When you type the Get-ChildItem cmdlet by itself, PowerShell will display the contents of the current folder (see Figure 3).

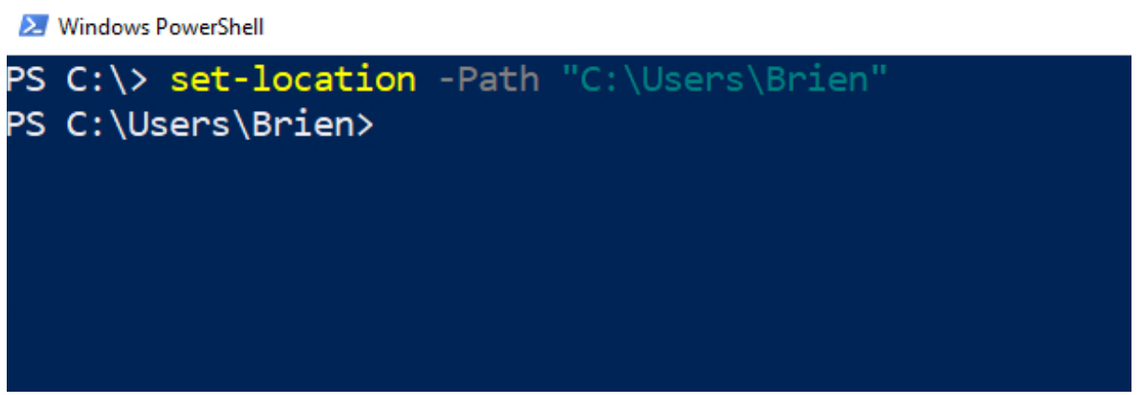


Figure 2: You can use the Set-Location cmdlet to navigate the file system.

```
PS C:\Users\Brien\3d objects> Get-ChildItem


    Directory: C:\Users\Brien\3d objects


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----        10/30/2015   5:12 AM         192654 Bulldozer.3mf
-a----        10/30/2015   5:12 AM         227638 Caboose car.3mf
-a----        10/30/2015   5:12 AM         158247 Chess Set.3mf
-a----        12/13/2015   6:55 AM          40924 Cone Shape.3mf
-a----        12/13/2015   6:55 AM          17403 Cube Shape.3mf
-a----        12/13/2015   6:55 AM          31205 Cylinder Shape.3mf
-a----        10/30/2015   5:12 AM         189582 Gift Box.3mf
-a----        10/30/2015   5:12 AM         305981 Gimble Keychain.3mf
-a----        12/13/2015   6:55 AM          26638 Hexagon Shape.3mf
-a----        10/30/2015   5:12 AM          53141 Keychain.3mf
-a----        10/30/2015   5:12 AM         176493 Left Curve Track.3mf
-a----        12/13/2015   6:55 AM          13298 Pyramid Shape.3mf
-a----        10/30/2015   5:12 AM         147058 Right Curve Track.3mf
-a----        10/30/2015   5:12 AM         119010 Ship in a Bottle.3mf
-a----        10/30/2015   5:12 AM         211318 Space Shuttle.3mf
-a----        12/13/2015   6:55 AM         154842 Sphere Shape.3mf
-a----        10/30/2015   5:12 AM         236858 Split Track.3mf
-a----        10/30/2015   5:12 AM         295552 Star Trophy.3mf
-a----        10/30/2015   5:12 AM         128970 Straight Track.3mf
-a----        12/13/2015   6:55 AM          17897 Tetrahedron Shape.3mf
-a----        12/13/2015   6:55 AM         283532 Torus Shape.3mf
-a----        10/30/2015   5:12 AM         169424 Track Connector.3mf
-a----        10/30/2015   5:12 AM         306363 Train Engine.3mf
-a----        10/30/2015   5:12 AM         232957 Trophy Cylinder.3mf
```

Figure 3: The Get-ChildItem cmdlet is used to display a folder's contents.

Get-Alias

As you become more familiar with PowerShell, you will find that some commands do not adhere to the previously mentioned verb-noun naming convention. Such commands are typically aliases.

An alias is just an alternative name for a cmdlet. In some cases, an alias is simply used as a shortcut. For example, you can type Select instead of Select-Object because the word Select is an alias for Select-Object.

In other cases, aliases are used for backward compatibility purposes. For example, in the days of DOS, the DIR command was used to view the files within a folder. The DIR command is still used within the Windows Command Prompt window. Even though PowerShell uses a different command (Get-ChildItem), DIR is used as an alias.

The Get-Alias cmdlet can help you to figure out which cmdlet an alias points to. For example, to find out which cmdlet was in fact being used when you typed DIR, you could type:

Get-Alias DIR

This command would show you that DIR is an alias for Get-ChildItem, as shown in Figure 4.

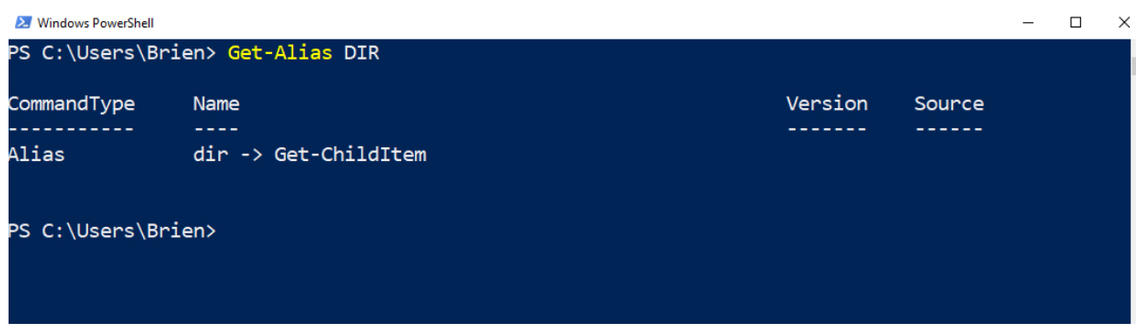


Figure 4: The DIR command is an alias for Get-ChildItem.

How to Use PowerShell to Navigate the Windows Folder Structure

Learn these PowerShell commands to move quickly around Windows folder structures.

PowerShell uses several commands to navigate the Windows folder structure.

When you open a new PowerShell window, PowerShell usually starts you in your user profile directory. In Figure 1, for example, you can see that the PowerShell prompt is pointing to C:\Users\Brien. Although the user profile has its place, PowerShell operations often require you to navigate to a different location within the folder hierarchy. In my case, most of my PowerShell scripts are located in a folder called C:\Scripts and are inaccessible from the user profile folder.

In this article, I will explain how to use PowerShell to navigate the Windows folder structure.

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

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Brien>
```

Figure 1: PowerShell often opens to your user profile folder.

Switching Drives

In PowerShell-based navigation, the first thing to know is how to switch to different drives.

To change drives, just enter the drive letter you want to go to, followed by a colon. In Figure 2, for example, I switched from the C: drive to the Q: drive by typing Q:. I then switched back to the C: drive by typing C:.

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

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Brien> q:
PS Q:\> c:
PS C:\Users\Brien>
```

Figure 2: You can switch drives by entering the drive letter and a colon.

CD.. and CD\ Commands

Most file system navigation involves traversing the directory structure. If you want to drop down one level within the directory hierarchy, you can do so by typing CD.. (note that there are two periods).

If you look back at the previous two figures, you can see that PowerShell initially placed me in the C:\Users\Brien folder. By entering CD.., PowerShell would drop me down to the C:\Users folder.

Although you can use the CD.. command to move through the folder hierarchy one level at a time, it's not always the most efficient method. For example, if I were in the C:\Users\Brien folder and needed to drop down to the root folder, I could enter the CD.. command twice, as I have done in Figure 3.

```
PS C:\Users\Brien> cd..
PS C:\Users> cd..
PS C:\>
```

Figure 3: Typing CD.. moves you down a level in the folder hierarchy.

However, as a shortcut, I could enter the CD\ command. That would immediately drop me down to the root folder. Figure 4 provides an example.

```
PS C:\users\Brien> cd\
PS C:\>
```

Figure 4: Typing CD\ causes PowerShell to move to the root directory.

As you can see, you can use the CD.. or CD\ command to move to a lower level within the folder hierarchy. You can also use the CD command to enter a folder. Just type CD, followed by the folder name. If I were in the C:\Users folder and wanted to navigate to the Brien subfolder, I could type CD Brien.

PowerShell Aliases

Note that even though this article discusses the CD command, CD is not a "real" PowerShell command. The CD command is what is known as an alias (aliases are discussed in the previous article). In the days of DOS, CD was the command used to traverse the directory structure. Microsoft has included support for the CD command in PowerShell both as a shortcut and way of making PowerShell a bit more like DOS. The longer cmdlet for which CD is an alias is Set-Location.

The Set-Location cmdlet works identically to the CD command, with one minor caveat. Unlike the CD command, you must include a space after the Set-Location cmdlet. Whereas CD.. is a valid command, Set-Location.. is not. Instead, to avoid receiving an error, you would have to type:

Set-Location ..

You can see this in Figure 5.

```
PS C:\Users\Brien> Set-Location..
Set-Location.. : The term 'Set-Location..' is not recognized as the name of a cmdlet,
function, script file, or operable program. Check the spelling of the name, or if a path was
included, verify that the path is correct and try again.
At line:1 char:1
+ Set-Location..
+ ~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (Set-Location..:String) [], CommandNotFoundExcep
   tion
    + FullyQualifiedErrorId : CommandNotFoundException
```

Figure 5: The CD command is an alias for Set-Location.

Get-ChildItem and DIR Commands

There is one more PowerShell cmdlet that is useful when navigating the Windows folder structure: Get-ChildItem.

The Get-ChildItem cmdlet displays the contents of the current folder. It's helpful if you want to navigate to a subfolder but aren't sure of the exact folder name.

Incidentally, the Get-ChildItem cmdlet also has an alias. The alias is DIR, which also originates from the days of DOS. Back then, DIR was the DOS Directory command. You can see how the DIR command works in Figure 6.

Figure 6: You can use the DIR command to see the current folder's contents.

```
PS C:\> CD Users
PS C:\Users> CD Brien
PS C:\Users\Brien> DIR


    Directory: C:\Users\Brien


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        10/23/2018  11:13 AM                .aws
d-----         3/19/2021  11:50 AM                .cache
d-----        12/29/2015  11:17 AM                .dnx
d-----         5/24/2018   8:13 PM                .dotnet
d-----         1/14/2019   2:30 PM                .elasticbeanstalk
d-----        12/31/2016  10:36 AM                .gimp-2.8
d-----         1/8/2017   12:15 PM                .jssc
d-----         6/25/2019  11:02 AM                .kube
d-----          7/5/2022   9:40 AM                .ms-ad
d-----         7/27/2016  11:13 AM                .nuget
d-----        10/15/2016  11:45 AM                .oracle_jre_usage
d-----        10/30/2020   8:12 PM                .vscode
d-r---        12/23/2019   1:17 AM                3D Objects
d-----         1/13/2017   8:51 PM                ADXL345_library
d-----         10/4/2019  10:03 AM                Apple
d-r---        12/23/2019   1:17 AM                Contacts
d-r---         7/29/2022   9:05 AM                Desktop
d-r---         7/31/2022   8:16 PM                Documents
d-r---         3/30/2022  11:12 PM                Downloads
```

So, with that in mind, suppose that I needed to navigate to the C:\Users\Brien\Desktop folder but couldn't remember the name of the Desktop folder. I could navigate to C:\Users\Brien, then use either the DIR or the Get-ChildItem cmdlet to see a list of all the files and folders in that location. I could then use that information to get the name of the folder that I need (in this case, Desktop) and navigate to it using the CD command. ■

How To Run a PowerShell Script

Here are the steps for running PowerShell scripts.

PowerShell scripts have a wide variety of purposes. Microsoft, for example, sometimes provides PowerShell scripts to automate difficult configuration tasks. Similarly, organizations often run PowerShell scripts to automate the process of onboarding new users. Such a script might create a user's Active Directory account and add the user to a group. In my own organization, I use PowerShell scripts to monitor disk health.

So, seeing that PowerShell scripts can be unquestionably useful, how do you run them?

But First – A Word of Caution

Before I explain how to run a script, I must give a word of caution.

When you execute a script, that script runs with the same permissions as the account you are currently logged in with. Depending on what a script is designed to do, you may sometimes need to run the script with administrative privileges. However, it's best to avoid using a privileged account if possible. This is especially true if you downloaded the script from the internet. If you happened to download a malicious script, its ability to inflict damage on your system will be limited by your account's permissions.

Use PowerShell in Administrative Mode

Even if you are logged in as an administrator, PowerShell does not automatically allow you to exercise your administrative authority. If you need to run a PowerShell script as an administrator, you will need to open PowerShell in administrative mode.

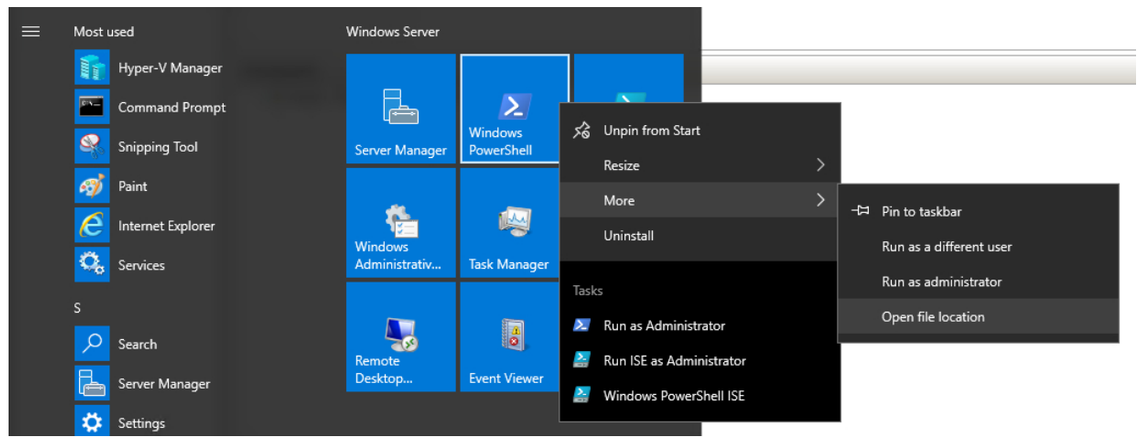


Figure 1: Right-click on the PowerShell icon and choose the More | Run as Administrator commands from the shortcut menus.

To do so, find PowerShell on the Start menu, right-click on the PowerShell icon, and then select More | Run as Administrator from the shortcut menu (See Figure 1).

Note that using the Run as Administrator option is not a requirement for running all PowerShell scripts. Using the Run as Administrator option is only necessary if the script requires administrative privileges.

Disable the Execution Policy

Here's the next thing to know about running PowerShell scripts: Depending on what version of Windows you have, where you got the script, and how your system is configured, the system's execution policy may prevent you from running the

PowerShell script. Thankfully, it's relatively easy to temporarily disable the execution policy so that your script can run.

Execution policies are designed to prevent you from accidentally running a potentially untrustworthy PowerShell script. If you need to disable an execution policy to run a script, the easiest way to do that is to type this command:

Set-ExecutionPolicy Unrestricted

When you're done running the script, it's a good idea to reenable the execution policy. There are several different types of execution policies, but the most secure one is called ***restricted***. You can enable it by running this command:

Set-ExecutionPolicy Restricted

Running the Script

With the execution policy out of the way, you can now run your script.

First, navigate to the folder containing the script. In PowerShell, the CD command is used for navigating the directory structure. Suppose that right now the PowerShell prompt is pointing to the C:\Data folder and I need to run a script located in C:\Scripts. I could navigate to that folder by typing these commands:

CD..

CD Scripts

The first command drops down a level in the directory hierarchy. The second command goes to the specified folder name. However, if the folder name consists of more than a single word, then you will need to put the folder name in quotation marks to avoid receiving an error. For example, if the script is in a folder named My Scripts (instead of just Scripts), then you would use this command:

CD "My Scripts"

Once you have navigated to the folder containing the script, you can run the script in question. To do so, just type a period, followed by a slash and the filename. Don't forget to include the .PS1 extension.

If you look at Figure 2, you can see that the first command that I typed was CD\. This is a shortcut. The CD.. command drops down one level in the folder hierarchy. Since I needed to go all the way to the root directory, I used CD\ rather than typing CD.. twice. The next thing that I did was to run a script called HelloWorld.ps1 by typing ./HelloWorld.ps1. ■

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

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Brien> cd\
PS C:\> cd scripts
PS C:\scripts> ./HelloWorld.ps1
PS C:\scripts>
```

Figure 2: Place ./ in front of the file name to run the script.

How to Create Functions in PowerShell Scripts

PowerShell functions provide a slew of benefits. Here are the basics for designing functions, including functions that use parameters.

If you write a lot of PowerShell scripts, sooner or later you will find that you need to create a function. PowerShell functions are essentially a block of code that you can run again and again. You can call a PowerShell function as often as you need to.

Benefits of PowerShell Functions

Even if you don't need to repeatedly run a block of code within a script, PowerShell functions can serve other purposes. Imagine a situation in which you might need to handle an error within a script, but there is no guarantee that an error will occur. A function is perfect in this case because functions provide a way of writing code that will not execute unless called on.

Another benefit of using PowerShell functions is that they can help to make long and complex scripts easier to read and debug. Rather than having the script consist of one huge block of code, you can break major areas of functionality out into functions.

There are countless other benefits to using PowerShell functions. You can even use functions (inside of modules) as a tool for creating custom PowerShell cmdlets!

PowerShell Function Basics

With that said, I want to show you the basics of creating a PowerShell function.

Name your function

The first thing to know about PowerShell functions is that every function must have a name. Technically, you can give a function any name you want, so long as you don't use any reserved words. However, as a best practice, I recommend giving your functions names that adhere to the basic PowerShell cmdlet structure (i.e., in a verb-noun format, discussed in

"5 Tips for Learning PowerShell"). It works well to use the same naming convention for your functions because the verb-noun combination can describe what the function is designed to do.

Additionally, if you later decide to turn the function into a custom PowerShell cmdlet, it will already be in a format that matches the general PowerShell naming convention.

Use 'Function'

When you create a PowerShell function, another thing to do is use the word "Function." ***Function*** appears just before the function name. If you try to declare a function name without using ***Function***, PowerShell will think you have typed an invalid command and give you an error.

Enclose body in brackets

The function body can contain anything that you want. The only real requirement is that the function body must be enclosed in brackets.

Example of a PowerShell Function

Now that I have explained how to declare a function, let's take a look at a very simple function provided by Microsoft.

The command for finding out which version of PowerShell you are running is $PSVersionTable.PSVersion. This command is not very intuitive, so let's create a function called Get-Version with the $PSVersionTable.PSVersion command in the function body. That way, you simply type Get-Version to see the PowerShell version.

Here is what the code looks like:

```
Function Get-Version {
   $PSVersionTable.PSVersion
}
```

As you can see, we have the word "Function," followed by a function name and an opening bracket. This is followed by the function body and a closing bracket. Once a function has been declared, you can execute it by entering the function name. You can see what this looks like in Figure 1.

PowerShell Functions That Use Parameters

You should also learn how to design a PowerShell function that accepts a parameter (although a PowerShell function can in fact accept multiple parameters).

There are different ways to handle parameters in PowerShell, so let's look at one of the simpler methods.

The easiest way I know to pass a parameter to a function is to include a variable in parentheses just after the function name. This variable then inherits whatever value is passed to the function. There are far more sophisticated ways of handling parameters, but this method works and is easy to use.

In the following example, I'm going to create a really simple function in which I pass a text

```
PS C:\Users\Brien> function Get-Version {
>>     $PSVersionTable.PSVersion
>> }
PS C:\Users\Brien>
PS C:\Users\Brien>
PS C:\Users\Brien>
PS C:\Users\Brien> Get-Version

Major  Minor  Build  Revision
-----  -----  -----  --------
5      1      19041  1682
```

Figure 1: This is an example of a simple PowerShell function.

string to the function, then use the function to display the contents of that texturing.

Here is what the function looks like:

```
Function Display-Text($MyText){
        Write-Host $MyText
}
```

This function, which you can see in Figure 2, accepts a string of text as input and then uses the Write-Host cmdlet to display that text. I supply the text immediately after entering the function name when I call the function.

You will notice that I have called the function twice. The first time, I just passed a raw text string to the function. The second time, I added the text string to a variable and passed the variable to the function. Both techniques produced the same output. ■

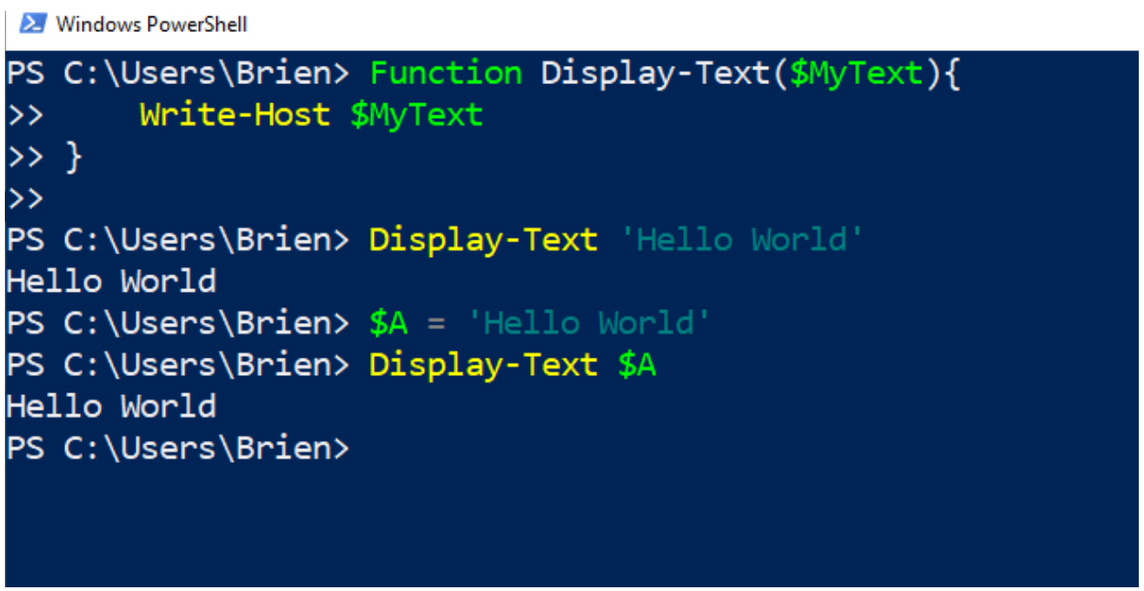


Figure 2: I have passed a value to a function.