Windows PowerShell Networking Guide

>

PowerShell.org

PowerShell Networking Guide

PowerShell.org

This project can be followed at:

https://www.penflip.com/powershellorg/powershell-networking-guide

Contents

1 Windows PowerShell Networking Guide

by Ed Wilson, Microsoft's "The Scripting Guy"

cover design by Nathan Vonnahme

2 Foreword

Visit Penflip.com/powershellorg to check for newer editions of this ebook.

PowerShell.org ebooks are works-in-progress, and many are curated by members of the community. We encourage you to check back for new editions at least twice a year, by visiting Penflip.com/powershellorg.

You can download this book in a number of different formats (including Epub, Pdf, Microsoft Word and Plain Text) by clicking on the 'Download' link on the right.

PDF Users: Penflip's PDF export often doesn't include the entire ebook content. We've reported this problem to them; in the meantime, please consider using a different format, such as EPUB, when you're downloading the book.

You may register to make corrections, contributions, and other changes to the text - we welcome your contributions! check out our contributor tips and notes before jumping in.

You may also subscribe to our monthly e-mail TechLetter for notifications of updated ebook editions. Visit PowerShell.org for more information on the newsletter.

3 Windows PowerShell Basics - Introduction

Windows PowerShell is not new technology. Windows PowerShell 4.0 ships in Windows 8.1 and in Windows Server 2012 R2, it has therefore been around for a while. Windows PowerShell is an essential admin tool designed specifically for Windows administration. By learning to use Windows PowerShell, network administrators quickly gain access to information from Windows Management Instrumentation, Active Directory and other essential sources of information. Additionally, Microsoft added Windows PowerShell support to the Common Criteria requirements for shipping enterprise applications. Therefore, to manage Microsoft Exchange, Azure, SQL Server, and others one needs to know and to understand how to use Windows PowerShell. In the networking world, this knowledge is also a requirement for managing DNS, DHCP, Network Adapters, and other components.

So what are the basics of Windows PowerShell that I need to know?

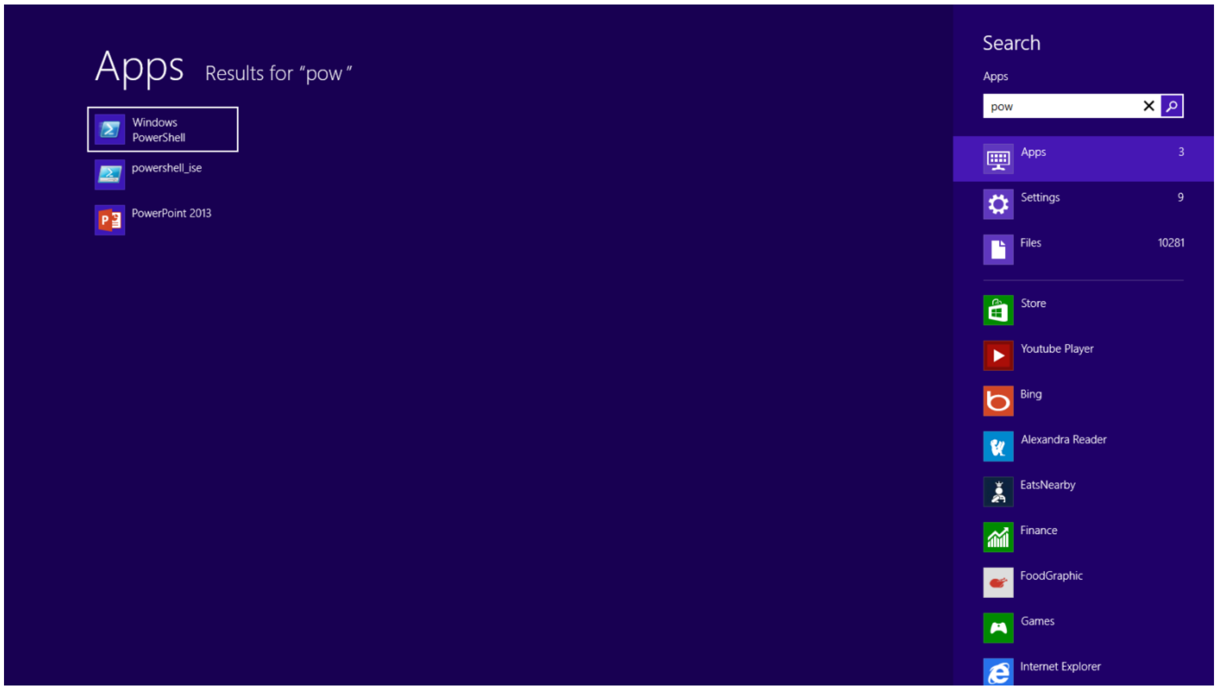
Windows PowerShell comes in two flavors - the first is an interactive console (sort of like a KORN or a BASH console in the UNIX world) built into the Windows command prompt. The Windows PowerShell console makes it simple to type short commands and to receive sorted, filtered, formatted results. These results easily display to the console, but can redirect to XML, CSV, or text files. The Windows PowerShell console offers several advantages such as speed, low memory overhead, and a comprehensive transcription service that records all commands and command output.

There is also the Windows PowerShell ISE. The Windows PowerShell ISE is an Integrated Scripting Environment, but this does not mean you must use it to write scripts. In fact, many Windows PowerShell users like to write their code in the Windows PowerShell ISE to take advantage of the color syntax-highlighting, drop down lists, and automatic parameter revelation features. In addition, the Windows PowerShell ISE has a feature, called the _Show Command Add-On _that permits using a mouse to create Windows PowerShell commands from a graphical environment. Once created, the command either runs directly, or adds to the script pane (the choice is up to you).

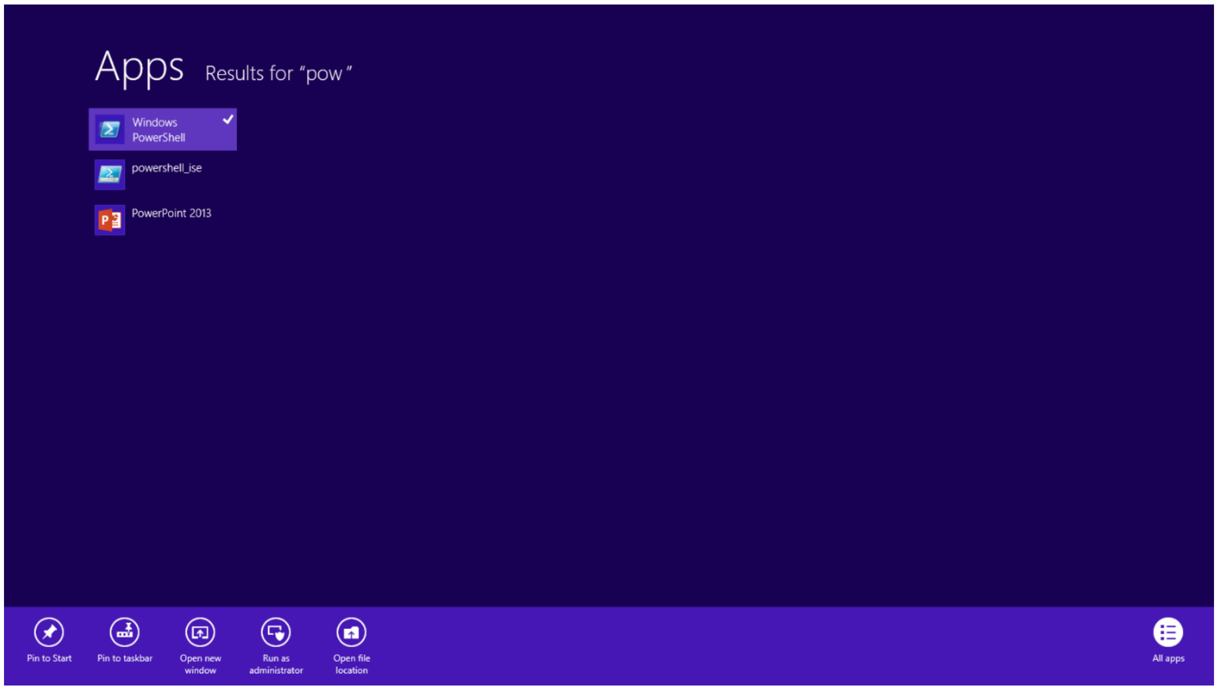
Working with Windows PowerShell

On Windows 8 or on Windows Server 2012 Windows PowerShell 3.0 already exists. On Windows 8.1 Windows PowerShell 4.0 is installed, as it is on Windows Server 2012 R2. Windows 8 (and 8.1) you only need to type the first few letters of the word PowerShell on the Start screen before Windows

PowerShell appears as an option. The figure appearing here illustrates this point. I only typed _pow _before the Start screen search box changes to offer Windows PowerShell and an option.



Because navigating to the Start screen and typing _pow _each time I want to launch Windows PowerShell is a bit cumbersome, I prefer to Pin the Windows PowerShell console (and the Windows PowerShell ISE) to both the Start page and to the Windows desktop taskbar. This technique of pinning shortcuts to the applications provides single click access to Windows PowerShell from wherever I may be working.



On Windows Server 2012 (and on Windows Server 2012 R2), it is not necessary to go through the Start screen / Search routine because an icon for the Windows PowerShell console exists by default on the taskbar of the desktop.

NOTE : The Windows PowerShell ISE (the script editor) does not exist by default on Windows Server 2012 and Windows Server 2012 R2. You add the Windows PowerShell ISE as a feature.

4 Windows PowerShell Basics - Security issues with Windows PowerShell

There are two ways of launching Windows PowerShell - as an administrator and as a normal user. It is a best practice when starting Windows PowerShell to start it with minimum rights. On Windows 8 (and on Windows 7) this means simply clicking on the Windows PowerShell icon. It opens as a non-elevated user (even if you are logged on with Administrative rights). On Windows Server 2012, Windows PowerShell automatically launches with the rights of the current user and therefore if you are logged on as a Domain Administrator, the Windows PowerShell console launches with Domain Administrator rights.

Running as a normal (non-elevated) user

Because Windows PowerShell adheres to Windows security constraints, a user of Windows PowerShell cannot do anything that the user account does not have permission to do. Therefore, if you are a non-elevated normal user, you will not have rights to do things like install printer drivers, read from the Security log, or change system time.

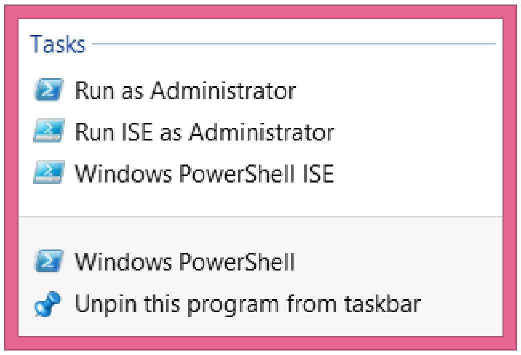
Even if you are an administrator on the local Windows 8 (or Windows 7) desktop machine and you do not launch Windows PowerShell with admin rights, you will get errors when attempting to do things like see the configuration of your disk drives. This command and associated error appears here.

```
PS C:\> get-disk get-disk : Access to a CIM resource was not available to the client. At line:1
char:1 + get-disk + ~~~~~~~~ + CategoryInfo : PermissionDenied: (MSFT\_Disk:ROOT/Microsoft/Windows/S
torage/MSFT\_Disk)[Get-Disk], CimException + FullyQualifiedErrorId : MI RESULT 2,Get-Disk
```

TIP : There is an inconsistency with errors arising when attempting to run cmdlets that require elevated rights. For example, when inside a non-elevated Windows PowerShell console, the error from Get-Disk is _Access to a CIM resource was not available to the client. _The error from Stop-Service is _Cannot open xxx service on computer. _While the Get-VM cmdlets simply returns no information (an no error). Therefore, as a first step in troubleshooting, check for console rights.

Launching PowerShell with Admin rights

When you need to perform tasks that require Admin rights, you need to start the Windows PowerShell console with admin rights. To do this, right click on the Windows PowerShell icon (from either the one pinned to the task bar, the start page, or even from the one found from the Start / Search page) and select the _Run As Administrator _option from the action menu. The great thing about this technique is that it permits launching either the Windows PowerShell console (the first item on the menu) as an Administrator, or from the same screen you can launch the Windows PowerShell ISE as an Administrator. This appears in the figure that follows.



Once you launch the Windows PowerShell console with admin rights, the User Account Control dialog box appears seeking permission to allow Windows PowerShell to make changes to the computer. In reality, Windows PowerShell is not making changes to the computer - not yet. But using Windows PowerShell you can certainly make changes to the computer - if you have the rights, and this is what the dialog is prompting you for.

NOTE : It is possible to avoid this prompt by turning off User Account Control (UAC). However, UAC is a significant security feature, and therefore I do not recommend disabling UAC. We have fine-tuned it in Windows 7 and continuing through Windows 8.1 and greatly reduced the number of UAC prompts (from the number that used to exist in the introduction of UAC on Windows Vista. This is not “your grandma’s UAC”.)

Now that you are running Windows PowerShell with admin rights, you can do anything your account has permission to do. Therefore, if you were to, for example, run the Get-Disk cmdlets, you would see information similar to the following appear.
“ “

PS C:> get-disk

Number Friendly Name OperationalS Total Size Partition
tatus Style
—— ————– ———— ———– ———

0 INTEL SSDSA2BW160G3L Online 149.05 GB MBR
“ “

5 Windows PowerShell Basics - Using PowerShell cmdlets

PowerShell cmdlets all work in a similar fashion. This simplifies their use. All Windows PowerShell cmdlets have a two-part name. The first part is a verb (not always strictly a grammatical verb however). The *verb* indicates the action for the command to take. Examples of verbs include Get, Set, Add, Remove, or Format. The *noun* is the thing to which the action will apply. Examples of nouns include Process, Service, Disk, or NetAdapter. A dash combines the verb with the noun to complete the Windows PowerShell command. Windows PowerShell commands, named cmdlets (pronounced command let), because they behave like small commands or programs are used standalone, or pieced together via a mechanism called the *pipeline* (refer to chapter two for the use of the *pipeline*).

The most common verb - `Get`

Out of nearly 2,000 cmdlets (and functions) on Windows 8, over 25 percent of them use the verb *Get.* The verb *Get* retrieves information. The Noun portion of the cmdlet specifies the information retrieved. To obtain information about the processes on your system, open the Windows PowerShell console by either clicking on the Windows PowerShell icon on the task bar (or typing PowerShell on the start screen of Windows 8 to bring up the search results for Windows PowerShell (as illustrated earlier)). Once the Windows PowerShell console appears, run the Get-Process cmdlet. To do this, use the Windows PowerShell Tab Completion feature to complete the cmdlet name. One the cmdlet name appears, press the key to cause the command to execute.

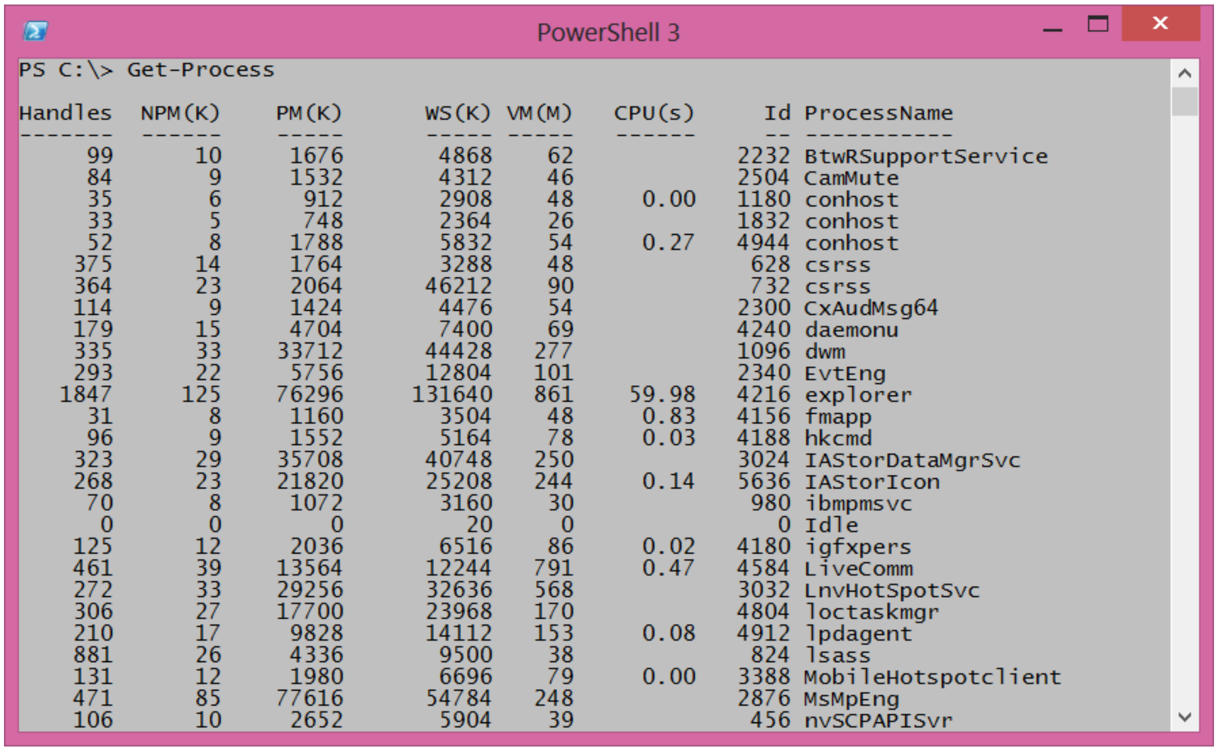
NOTE : The Windows PowerShell Tab Completion feature is a great time saver. It not only saves time (by reducing the need for typing) but it also helps to ensure accuracy, because Tab Completion accurately resolves cmdlet names - it is sort of like a spell checker for cmdlet names. For example, attempting to type a cmdlet name such as Get-NetAdapterEncapsulatedPacketTaskOffload accurately (for me anyway) could be an exercise in frustration. But using tab completion, I only have to type Get-Net and I hit the key about six times and the correctly spelled cmdlet name appears in the Windows PowerShell console. Learning how to quickly, and efficiently use the tab completion is one of the keys to success in using Windows PowerShell.

Finding process information

To use the Windows PowerShell Tab Completion feature to enter the Get-Process cmdlet name onto the Windows PowerShell console command line, type the following on the first line of the Windows PowerShell console:

```
Get-Pro + <tab> + <ENTER>
```

The Get-Process command and the associated output from the cmdlet appear in the figure that follows.



```
PS C:\> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     99      10     1676       4868    62            2232 BtwRSupportService
     84       9     1532       4312    46            2504 CamMute
     35       6      912       2908    48     0.00   1180 conhost
     33       5      748       2364    26            1832 conhost
     52       8     1788       5832    54     0.27   4944 conhost
    375      14     1764       3288    48             628 csrss
    364      23     2064      46212    90             732 csrss
    114       9     1424       4476    54            2300 CxAudMsg64
    179      15     4704       7400    69            4240 daemonu
    335      33    33712      44428   277            1096 dwm
    293      22     5756      12804   101            2340 EvtEng
   1847     125    76296     131640   861    59.98   4216 explorer
     31       8     1160       3504    48     0.83   4156 fmapp
     96       9     1552       5164    78     0.03   4188 hkcmd
    323      29    35708      40748   250            3024 IAStorDataMgrSvc
    268      23    21820      25208   244     0.14   5636 IAStorIcon
     70       8     1072       3160    30             980 ibmpmsvc
      0       0        0         20     0               0 Idle
    125      12     2036       6516    86     0.02   4180 igfxpers
    461      39    13564      12244   791     0.47   4584 LiveComm
    272      33    29256      32636   568            3032 LnvHotSpotSvc
    306      27    17700      23968   170            4804 loctaskmgr
    210      17     9828      14112   153     0.08   4912 lpdagent
    881      26     4336       9500    38             824 lsass
    131      12     1980       6696    79     0.00   3388 MobileHotspotclient
    471      85    77616      54784   248            2876 MsMpEng
    106      10     2652       5904    39             456 nvSCPAPISvr
```

To find information about Windows services, use the verb **Get** and the **noun** service. To type the cmdlet name, type the following:

`Get-Servi + <TAB> + <ENTER>`

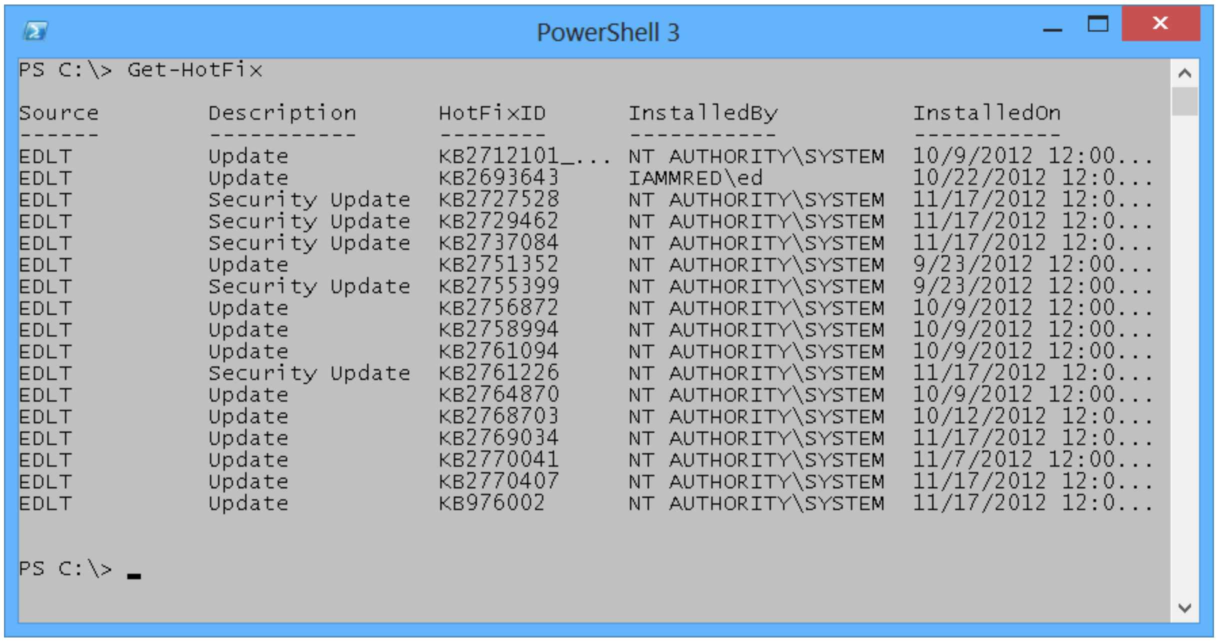
NOTE : It is a Windows PowerShell convention to use singular nouns. While not universally applied (my computer has around 50 plural nouns) it is a good place to start. So if you are not sure if a noun (or parameter) is singular or plural, choose the singular - most of the time you will be correct.

Identifying installed Windows Hotfixes

To find a listing of Windows Hotfixes applied to the current Windows installation, use the Get-Hotfix cmdlet (the **verb** is Get and the **noun** is Hotfix). Inside the Windows PowerShell console, type the following:

`Get-Hotf + <TAB> + <ENTER>`

The command, and the output associated with the command appear here.



```
PS C:\> Get-HotFix

Source        Description      HotFixID      InstalledBy          InstalledOn
------        -----------      --------      -----------          -----------
EDLT          Update           KB2712101_... NT AUTHORITY\SYSTEM  10/9/2012 12:00...
EDLT          Update           KB2693643     IAMMRED\ed           10/22/2012 12:0...
EDLT          Security Update  KB2727528     NT AUTHORITY\SYSTEM  11/17/2012 12:0...
EDLT          Security Update  KB2729462     NT AUTHORITY\SYSTEM  11/17/2012 12:0...
EDLT          Security Update  KB2737084     NT AUTHORITY\SYSTEM  11/17/2012 12:0...
EDLT          Update           KB2751352     NT AUTHORITY\SYSTEM  9/23/2012 12:00...
EDLT          Security Update  KB2755399     NT AUTHORITY\SYSTEM  9/23/2012 12:00...
EDLT          Update           KB2756872     NT AUTHORITY\SYSTEM  10/9/2012 12:00...
EDLT          Update           KB2758994     NT AUTHORITY\SYSTEM  10/9/2012 12:00...
EDLT          Update           KB2761094     NT AUTHORITY\SYSTEM  10/9/2012 12:00...
EDLT          Security Update  KB2761226     NT AUTHORITY\SYSTEM  11/17/2012 12:0...
EDLT          Update           KB2764870     NT AUTHORITY\SYSTEM  10/9/2012 12:00...
EDLT          Update           KB2768703     NT AUTHORITY\SYSTEM  10/12/2012 12:0...
EDLT          Update           KB2769034     NT AUTHORITY\SYSTEM  11/17/2012 12:0...
EDLT          Update           KB2770041     NT AUTHORITY\SYSTEM  11/7/2012 12:00...
EDLT          Update           KB2770407     NT AUTHORITY\SYSTEM  11/17/2012 12:0...
EDLT          Update           KB976002      NT AUTHORITY\SYSTEM  11/17/2012 12:0...

PS C:\> _
```

Get detailed service information

To find information about services on the system, use the Get-Service cmdlet. Once again, it is not necessary to type the entire command. The following command uses Tab Expansion to complete the Get-Service command and to execute it.

`Get-Servi + <TAB> + <ENTER>`

NOTE : The efficiency of Tab Expansion depends upon the number of cmdlets, functions, or modules installed on the computer. As more commands become available, the efficiency of Tab Expansion reduces correspondingly.

The following (truncated) output appears following the Get-Service cmdlet.
""

PS C:> Get-Service

Status Name DisplayName
—— —- ————

Running AdobeActiveFile... Adobe Active File Monitor V6
Stopped AeLookupSvc Application Experience
Stopped ALG Application Layer Gateway Service
Stopped AllUserInstallA... Windows All-User Install Agent

""

Identifying installed network adapters

To find information about network adapters on your Windows 8 (or Windows Server 2012) machine, use the Get-NetAdapter cmdlet. Using Tab Expansion, type the following:

`Get-NetA + <TAB> + <ENTER>`

The command and associated output appear here.
““
```
PS C:> Get-NetAdapter

Name InterfaceDescription ifIndex Status
—– ——————— ——– ——
Network Bridge Microsoft Network Adapter Multiplexo… 29 Up
Ethernet Intel(R) 82579LM Gigabit Network Con… 13 Not Pre…
vEthernet (WirelessSwi… Hyper-V Virtual Ethernet Adapter #4 31 Up
vEthernet (External Sw… Hyper-V Virtual Ethernet Adapter #3 23 Not Pre…
vEthernet (InternalSwi… Hyper-V Virtual Ethernet Adapter #2 19 Up
Bluetooth Network Conn… Bluetooth Device (Personal Area Netw… 15 Disconn…
Wi-Fi Intel(R) Centrino(R) Ultimate-N 6300… 12 Up
```
““

Retrieving detected network connection profiles

If you want to see the network connection profile that Windows 8 (or Windows Server 2012) detected for each interface, use the Get-NetConnectionProfile cmdlet. To run this command, use the following command with Tab Expansion.
```
Get-NetC + <TAB> + <ENTER>
```
The command and associated output appear here.
““
```
PS C:> Get-NetConnectionProfile

Name : Unidentified network
InterfaceAlias : vEthernet (InternalSwitch)
InterfaceIndex : 19
NetworkCategory : Public
IPv4Connectivity : NoTraffic
IPv6Connectivity : NoTraffic

Name : Network 10
InterfaceAlias : vEthernet (WirelessSwitch)
InterfaceIndex : 31
NetworkCategory : Public
IPv4Connectivity : Internet
IPv6Connectivity : NoTraffic
```
““

NOTE : Windows PowerShell is not case sensitive. There are a few instances where case sensitivity is an issue (for example when using Regular Expressions) but cmdlet names, parameters and values are not case sensitive. Windows PowerShell convention uses a combination of upper case and lower case letters (generally at syllable breaks in long noun names such as NetConnectionProfile) but this is not a requirement for Windows PowerShell to interpret accurately the command. This combination of upper case and lowercase letters are for readability. If you use Tab Expansion, Windows PowerShell automatically converts the commands to this fashion.

Getting the current culture settings

There are two types of culture settings on a typical Windows computer. There are the culture settings for the current culture settings. This includes information about the keyboard layout, and the display format of items such as numbers, currency, and dates. To find the value of these cultural settings, use the Get-Culture cmdlet. To call the Get-Culture cmdlet using Tab Expansion to complete the command, type the following on the current line of the Windows PowerShell console:

```
Get-Cu + <TAB> + <ENTER>
```

When the command runs basic information such as the Language Code ID number (LCID), the name of the culture settings, as well as the display name of the culture settings return to the Windows PowerShell console. The command and associated output appears here.
““

```
PS C:> Get-Culture

LCID Name DisplayName
—- —- ———
1033 en-US English (United States)
```

The second culture related grouping of information is the current user interface (UI)settings for Windows. The UI culture settings determine which text strings appear in user interface elements such as menus and error messages. To determine the current UI culture settings that are active use the Get-UICulture cmdlet. Using Tab Expansion to call the Get-UICulture cmdlet, type the following:

```
Get-Ui + +
```

The command and output associated from the command appears here.

```
PS C:> Get-UICulture

LCID Name DisplayName
—- —- ———
1033 en-US English (United States)
```

““

NOTE : On my laptop, both the current culture and the current UI culture are the same. This is not always the case, and at times, I have seen machines become rather confused when the user interface is set for a localized language, and yet the computer itself was still set for US English (this is especially problematic when using virtual machines created in other countries. In these cases, even a simple task like typing in a password becomes very frustrating. To fix these types of situations you can use the Set-Culture cmdlet.

Finding the current date and time

To find the current date or time on the local computer, use the Get-Date cmdlet. When typing the Get-Date cmdlet name in the Windows PowerShell console tab expansion does not help too much. This is because there are 15 cmdlets (on my laptop) that have a cmdlet name that begins with the letters Get-Da (this includes all of the Direct Access cmdlets as well as the Remote Access cmdlets). Therefore using Tab Expansion (on my laptop anyway) to get the date requires me to type the following:

```
Get-Dat + <TAB> + <Enter>
```

The above command syntax is just the same number of letters to type as doing the following:

```
Get-Date + <ENTER>
```

The following illustrates the command and the output associated with the command.

```
PS C:\> Get-Date
```

Tuesday, November 20, 2012 9:54:21 AM

Generating a random number

Windows 2.0 introduced the Get-Random cmdlet, and when I saw it I was not too impressed. The reason was that I already knew how to generate a random number. Using the .NET Framework System.Random class, all I needed to do was create a new instance of the System.Random object, and call the _next _method. This appears here.

```
PS C:\> (New-Object system.random).next()225513766
```

Needless to say, I did not create all that many random numbers. I mean, who wants to do all that typing. But once I had the Get-Random cmdlet, I actually began using random numbers for all sorts of things. Some of the things I have used the Get-Random cmdlet to do appear in the following list.

- Pick prize winners for the Scripting Games
- Pick prize winners for Windows PowerShell user group meetings
- To connect to remote servers in a random fashion for load balancing purposes
- To create random folder names
- To create temporary users in active directory with random names
- To wait a random amount of time prior to starting or stopping processes and services (great for performance testing)

The Get-Random cmdlet has turned out to be one of the more useful cmdlets. To generate a random number in the Windows PowerShell console using Tab Expansion type the following on the first line in the console:

```
Get-R +<TAB>+<ENTER>
```

The command, and output associated with the command appears here.

```
PS C:\> Get-Random 248797593
```

6 Windows PowerShell Basics - Supplying options for cmdlets

The easiest Windows PowerShell cmdlets to use require no options. But unfortunately, that is only a fraction of the total number of cmdlets (and functions) available in Windows PowerShell 4.0 as it exists on either Windows 8.1 or Windows Server 2012 R2. Fortunately, the same Tab Expansion technique used to create the cmdlet names on the Windows PowerShell console, works with **parameters** as well.

Using single parameters

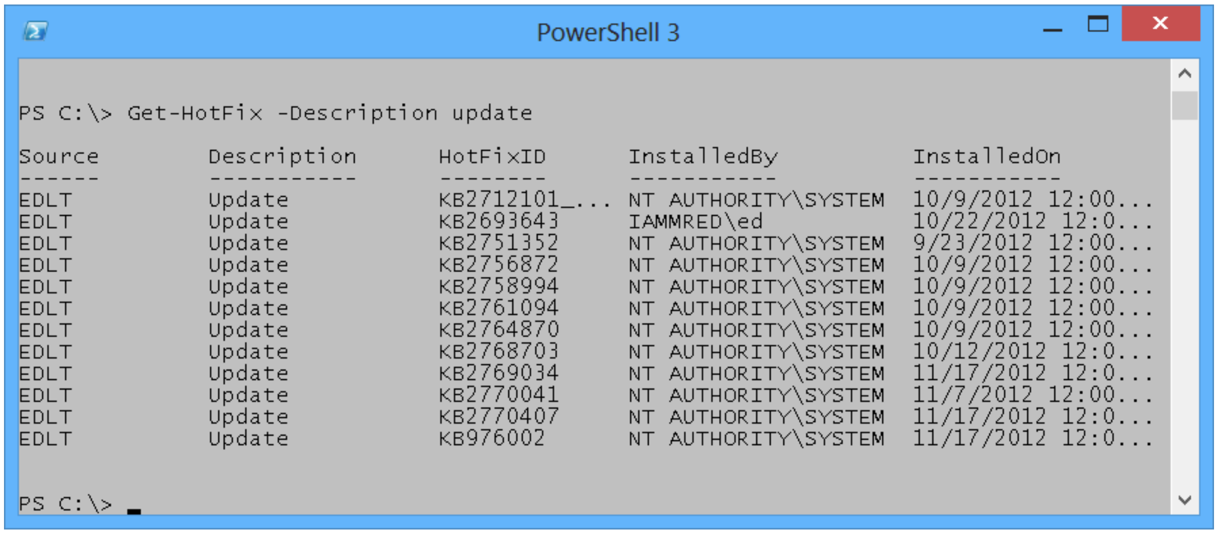
When working with Windows PowerShell cmdlets, often the cmdlet only requires a single parameter to filter out the results. If a parameter is the default parameter, you do not have to specify the parameter name - you can use the parameter positionally. This means that the first value appearing after the cmdlet name, is assumed to be a value for the default (or position 1) parameter. On the other hand, if a parameter is a **named parameter** the parameter name (or parameter alias or partial parameter name) is always required when using the parameter.

Finding specific types of hotfixes

For example to find all of the _update _hotfixes, use the Get-HotFix cmdlet with the -Description parameter and supply a value of _update _to the -Description parameter. This is actually easier than it sounds. Once you type Get-Hot and press the key you have the Get-Hotfix portion of the command. Then a space and -D completes the Get-HotFix -Description portion of the command. Now you need to type Update and press . With a little practice, using Tab Expansion becomes second nature. You only need to type the following:

```
Get-Hot + <TAB> + -D + <TAB> + Update + <ENTER>
```

The completed command and the output associated with the command appear in the figure that follows.



```
PS C:\> Get-HotFix -Description update

Source        Description     HotFixID      InstalledBy          InstalledOn
------        -----------     --------      -----------          -----------
EDLT          Update          KB2712101_... NT AUTHORITY\SYSTEM  10/9/2012 12:00...
EDLT          Update          KB2693643     IAMMRED\ed           10/22/2012 12:0...
EDLT          Update          KB2751352     NT AUTHORITY\SYSTEM  9/23/2012 12:00...
EDLT          Update          KB2756872     NT AUTHORITY\SYSTEM  10/9/2012 12:00...
EDLT          Update          KB2758994     NT AUTHORITY\SYSTEM  10/9/2012 12:00...
EDLT          Update          KB2761094     NT AUTHORITY\SYSTEM  10/9/2012 12:00...
EDLT          Update          KB2764870     NT AUTHORITY\SYSTEM  10/9/2012 12:00...
EDLT          Update          KB2768703     NT AUTHORITY\SYSTEM  10/12/2012 12:0...
EDLT          Update          KB2769034     NT AUTHORITY\SYSTEM  11/17/2012 12:0...
EDLT          Update          KB2770041     NT AUTHORITY\SYSTEM  11/7/2012 12:00...
EDLT          Update          KB2770407     NT AUTHORITY\SYSTEM  11/17/2012 12:0...
EDLT          Update          KB976002      NT AUTHORITY\SYSTEM  11/17/2012 12:0...

PS C:\> _
```

If you attempt to find only update types of hotfixes by supplying the value _update _in the first position, an error raises. The offending command, and associated error, appears here.

`PS C:\> Get-HotFix update Get-HotFix : Cannot find the requested hotfix on the 'localhost' computer. Verify the input and run the command again. At line:1 char:1 + Get-HotFix update + ~~~~~~~~~~~~~~~~~ + CategoryInfo : ObjectNotFound: (:)[Get-HotFix], ArgumentException + FullyQualifiedErrorId : GetHotFixNoEntriesFound,Microsoft.PowerShell.Commands .GetHotFixCommand`

The error, while not really clear, seems to indicate that the Get-HotFix cmdlet attempts to find a hotfix named _update. _This is, in fact, the attempted behavior. The help file information for the Get-HotFix cmdlet reveals that -ID is position 1. This appears here.
""

-Id
Gets only hotfixes with the specified hotfix IDs. The default is all
hotfixes on the computer.

```
Required?              false
Position?              1
Default value            All hotfixes
Accept pipeline input?     false
Accept wildcard characters? False
```

```
Well, what about using the -Description parameter, you may ask? The help file tells
    that the -Description parameter is a named parameter. This means you can only
    use the -Description parameter if you specify the parameter name as was
    accomplished earlier in this section. Here is the applicable portion of the
    help file for the -Description parameter.
```

-Description
Gets only hotfixes with the specified descriptions. Wildcards are
permitted. The default is all hotfixes on the computer.

```
Required?              false
Position?              named
Default value            All hotfixes
Accept pipeline input?     false
Accept wildcard characters? True
```

```
### Finding specific processes

To find process information about a single process, I use the -Name parameter.
    Because the -Name parameter is the default (position 1) parameter for the
    Get-Process cmdlet, you do not have to specify the -Name parameter when calling
    Get-Process if you do not wish to do so. For example, to find information about
    the PowerShell process by using the Get-Process cmdlet type the following
    command in the first line of the Windows PowerShell console by using Tab
    Expansion:
```

Get-Pro + + + Po + +

The completed command and associated output appears here.

PS C:> Get-Process powershell

Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id ProcessName
——- —— —– —– —– —— – ———–

607 39 144552 164652 718 5.58 4860 powershell

You can tell that the Get-Process cmdlet accepts the -Name parameter in a positional manner because the Help file states it is in position 1. This appears here.

-Name
Specifies one or more processes by process name. You can type multiple process names (separated by commas) and use wildcard characters. The parameter name ("Name") is optional.

```
Required?          false
Position?          1
Default value
Accept pipeline input?    true (ByPropertyName)
Accept wildcard characters? True
```

```
**NOTE** : Be careful using positional parameters. This is because they can be
    confusing. For example, the first parameter for the Get-Process cmdlet is the
    -Name parameter, but the first position parameter for the Stop-Parameter is the
    -ID parameter. As a best practice always refer to the Help files to see what
    the parameters actually are called, and the position in which they are
    expected. This is even more important when using cmdlet with multiple
    parameters - such as the Get-Random cmdlet discussed next.

### Generating random numbers in a range

When used without any parameters, the Get-Random cmdlet returns a number that is in
    the range of 0 to 2,147,483,647. We have never had a Windows PowerShell user
    group meeting in which there were either 0 people in attendance, nor have we
    had a Windows PowerShell user group meeting with 2,147,483,647 people in
    attendance. Therefore when handing out prizes at the end of the day, it is
    important to set a different minimum and maximum number.

**NOTE** : When using the -Maximum parameter for the Get-Random cmdlet keep in mind
    that the maximum number never appears. Therefore, if you have 15 people
    attending your Windows PowerShell user group meeting, you would want to set the
    -Maximum parameter to 16 (unless you do not like the 15 person and do not want
    them to win any prizes).

```

```
The default parameter for the Get-Random cmdlet is the -Maximum parameter. This
    means that you can use the Get-Random cmdlet to generate a random number in the
    range of 0 to 20 by using Tab Expansion on the first line of the Windows
    PowerShell console. Type the following (remember Get-Random never reaches the
    maximum number, therefore always use a number 1 greater than the desired upper
    number):
```

Get-R + + + 21
`If you want to generate a random number between 1 and 20, you might think you could use Get-Random 1 21, but that generates an error. The command and the error appear here.`
PS C:> Get-Random 1 21
Get-Random : A positional parameter cannot be found that accepts argument '21'.
At line:1 char:1
+ Get-Random 1 21
+ ~~~~~~~~~~~~~~~~
+ CategoryInfo : InvalidArgument: (:) [Get-Random], ParameterBindingEx
ception
+ FullyQualifiedErrorId : PositionalParameterNotFound,Microsoft.PowerShell.Comm
ands.GetRandomCommand
““

The error states that *a positional parameter cannot be found that accepts argument '21'*. This is because the Get-Random only has one positional parameter - the -Maximum parameter. The -Minimum parameter is a named parameter (this appears in the Help file for the Get-Random cmdlet. Use of the Help files appears in Chapter two).

To generate a random number in the range of 1 to 20, use named parameters. To assist in creating the command use Tab Expansion for the cmdlet name as well as for the parameter names. Type the following to create the command using Tab Expansion.

`Get-R + <TAB> + -M + <TAB> + <SPACE> + 21 + -M + <TAB> + <SPACE> + 1 + <ENTER>`

The command and the output associated with the command appears here.

`PS C:\> Get-Random -Maximum 21 -Minimum 1 19`

An introduction to parameter sets

One of the things that quickly becomes confusing with Windows PowerShell cmdlets is that there are often different ways of using the same cmdlet. For example, you can specify the -Minimum and the -Maximum parameters, but you cannot also specify the -Count parameter. This is a bit unfortunate, because it would seem that using the -Minimum and the -Maximum parameters to specify the minimum and the maximum numbers for the random numbers makes sense. When the Windows PowerShell user group has five prizes to give away it is inefficient to have to either write a script to generate the five random numbers. It is also inefficient to have to run the same command five times.

This is where command sets come into play. The -Minimum and the -Maximum parameters specify the range within which to pick a single random number. To generate more than one random number use the -Count parameter. Here are the two parameter sets.
““

Get-Random [[-Maximum]][-Minimum <Object>] [-SetSeed]
[]

Get-Random [-InputObject] [-Count][-SetSeed <Int32>]
[]
""

The first parameter set accepts -Maximum, -Minimum and -SetSeed. The second parameter set accepts -InputObject, -Count and -SetSeed. Therefore you cannot use -Count with -Minimum or -Maximum - they are in two different groups of parameters (called parameter sets).

NOTE : It is quite common for Windows PowerShell cmdlets to have multiple parameter sets. Tab Expansion only offers parameters from one parameter set - therefore when you choose a parameter (such as -Count from Get-Random) the non-compatable parameters do not appear in tab Expansion. This feature keeps you from creating invalid commands. For an overview of a cmdlets parameter sets, use the Get-Help cmdlet.

Generating a certain number of random numbers

The Get-Random cmdlet, when used with the -Count parameter accepts an -InputObject parameter. The -InputObject parameter is quite powerful. The help file, appearing here, states that it accepts a collection of objects.
""

-InputObject
Specifies a collection of objects. Get-Random gets randomly selected
objects in random order from the collection. Enter the objects, a variabl
that contains the objects, or a command or expression that gets the
objects. You can also pipe a collection of objects to Get-Random.

Required? true
Position? 1
Default value
Accept pipeline input? true (ByValue)
Accept wildcard characters? False

```
An array (or a range)of numbers just happens to also be a collection of objects. The easiest
way to generate a range (or an array)of numbers is to use the range operator. The **range
operator** is two dots (periods)between two numbers. The **range operator** does not require
spaces between the numbers, and dots. This appears here.
```

PS C:> 1..5
1
2
3
4
5

```
Now to pick five random numbers from the range of 1 to 10, only requires the command appearing
here. (The parentheses are required around the range of 1 to 10 numbers to ensure the range of
numbers creates prior to attempting to select five from the collection.
```

Get-Random -InputObject (1..10) -Count 5

```
The command and output associated with the command appear here.
```

```
PS C:> Get-Random -InputObject (1..10) -Count 5
7
5
10
1
8
```
“ “

7 Windows PowerShell Basics - Using command line utilities

As easy as Windows PowerShell is to use, there are times when it is easier to find information by using a command line utility. For example, to find IP configuration information you only need to use the _Ipconfig.exe _utility. You can type this directly into the Windows PowerShell console and read the output in the Windows PowerShell console. This command and associated output appears here in truncated form.
““

```
PS C:> ipconfig

Windows IP Configuration

Wireless LAN adapter Local Area Connection* 14:

Media State . . . . . . . . . . . : Media disconnected
Connection-specific DNS Suffix . :

Ethernet adapter vEthernet (WirelessSwitch):

Connection-specific DNS Suffix . : quadriga.com
Link-local IPv6 Address . . . . . : fe80::915e:d324:aa0f:a54b%31
IPv4 Address. . . . . . . . . . . : 192.168.13.220
Subnet Mask . . . . . . . . . . . : 255.255.248.0
Default Gateway . . . . . . . . . : 192.168.15.254

Wireless LAN adapter Local Area Connection* 12:

Media State . . . . . . . . . . . : Media disconnected
Connection-specific DNS Suffix . :

Ethernet adapter vEthernet (InternalSwitch):

Connection-specific DNS Suffix . :
Link-local IPv6 Address . . . . . : fe80::bd2d:5283:5572:5e77%19
IPv4 Address. . . . . . . . . . . : 192.168.3.228
Subnet Mask . . . . . . . . . . . : 255.255.255.0
Default Gateway . . . . . . . . . : 192.168.3.100
```

```
To obtain the same information using Windows PowerShell would require a more
    complex command. The command to obtain IP information is Get-NetIPAddress, But
    there are several advantages. For one thing, the output from the _IpConfig.exe
    _command is text, whereas the output from Windows PowerShell is an object. This
    means you can group, sort, filter, and format the output in an easy fashion.
```

```

The cool thing is that with Windows PowerShell console, you have not only the
    simplicity of the command prompt, but you also have the powerful Windows
    PowerShell language built in. Therefore, if you need to refresh Group Policy
    three times and wait for five minutes between refreshes, you can use the
    command appearing here (looping is covered in chapter eleven).
```

1..3 | % {gpupdate ; sleep 300}
““