

Master PowerShell tricks

Volume 1

Dave Kawula - MVP

Thomas Rayner - MVP

Sean Kearney - MVP

Allan Rafuse - MVP

Ed Wilson – The Scripting Guy

PUBLISHED BY
MVPDays Publishing
<http://www.mvpdays.com>

Copyright © 2016 by MVPDays Publishing

All rights reserved. No part of this lab manual may be reproduced or transmitted in any form or by any means without the prior written permission of the publisher.

ISBN: 978-1541092839

Warning and Disclaimer

Every effort has been made to make this manual as complete and as accurate as possible, but no warranty or fitness is implied. The information provided is on an “as is” basis. The authors and the publisher shall have neither liability nor responsibility to any person or entity with respect to any loss or damages arising from the information contained in this book.

Feedback Information

We’d like to hear from you! If you have any comments about how we could improve the quality of this book, please don’t hesitate to contact us by visiting www.mvpdays.com or sending an email to feedback@mvpdays.com.

Foreword Ed Wilson “The Scripting Guy”

There are many cool things about Windows PowerShell. For me three of the most awesome things are the following:

1. If you don't like the way Windows PowerShell does things, you can change it.
2. If Windows PowerShell can't do something you need, you can add it.
3. The Windows PowerShell community is super dedicated, and will help you do both one and two.

Now, to be honest at times, it is necessary to write hundreds and hundreds of lines of arcane code, to dive into the deepest and darkest mysteries of programing, and even to learn about things like API's, Constructors, Events, threading, ACL's, DACL's, CACL's and maybe even Tetradactylies. Then again, most of the time it is not. In fact, it has been years and years since I wrote a hundreds and hundreds of lines of arcane code.

It is almost as if the Windows PowerShell team deliberately tried to make Windows PowerShell easy to use and easy to learn. Hmm ... I wonder if that approach would ever catch on? Anyway, there used to be an old saw: “Ease of use is directly opposed to program capability.” Or, in other words, if it is easy to use, it probably is not all that powerful. Well, PowerShell changes that ... dramatically.

And yet, Windows PowerShell is also deceptively easy to use. One can go from Get-Service and Get-Process or even Get-Date to some pretty complicated stuff in like one line of code.

This is why item number three is so important. The authors of this book: Dave, Sean, Thomas and Allan are all Windows PowerShell experts, and have even been recognized by Microsoft as community leaders. So, this means not only do they know their stuff, but they are also great at sharing that knowledge with the community. Sean Kearney is even an Honorary Scripting Guy – a very elite group indeed!

One of the great way that MVP’s share their knowledge and experience is via MVP Days a traveling road show that was started by Dave and Cristal Kawula. This is a very well run event, and I have had the opportunity to speak at two of the events ... it is cool, and it is fun.

So grab this book, get it autographed, and learn how to master some awesome PowerShell tricks. It is cool.

Ed Wilson

Microsoft Scripting Guy

@ScriptingGuys

Acknowledgements

From Dave

Cristal, you are my rock and my source of inspiration. For the past 20 + years you have been there with me every step of the way. Not only are you the “BEST Wife” in the world you are my partner in crime. Christian, Trinity, Keira, Serena, Mickaila and Mackenzie, you kids are so patient with your dear old dad when he locks himself away in the office for yet another book. Taking the time to watch you grow in life, sports, and become little leaders of this new world is incredible to watch.

Thank you, Mom and Dad (Frank and Audry) and my brother Joe. You got me started in this crazy IT world when I was so young. Brother, you mentored me along the way both coaching me in hockey and helping me learn what you knew about PC's and Servers. I'll never forget us as teenage kids working the IT Support contract for the local municipal government. Remember dad had to drive us to site because you weren't old enough to drive ourselves yet. A great career starts with the support of your family and I'm so lucky because I have all the support one could ever want.

A book like this filled with amazing Canadian MVP's would not be possible without the support from the #1 Microsoft Community Program Manager – Simran Chaudry. You have guided us along the path and helped us to get better at what we do every day. Your job is tireless and your passion and commitment make us want to do what we do even more.

Last but not least, the MVPDays volunteers, you have donated your time and expertise and helped us run the event in over 20 cities across North America. Our latest journey has us expanding the conference worldwide as a virtual conference. For those of you that will read this book your potential is limitless just expand your horizons and you never know where life will take you.

About the Authors

Dave Kawula - MVP

Dave is a Microsoft Most Valuable Professional (MVP) with over 20 years of experience in the IT industry. His background includes data communications networks within multi-server environments, and he has led architecture teams for virtualization, System Center, Exchange, Active Directory, and Internet gateways. Very active within the Microsoft technical and consulting teams, Dave has provided deep-dive technical knowledge and subject matter expertise on various System Center and operating system topics.

Dave is well-known in the community as an evangelist for Microsoft, IE, and Veeam technologies. Locating Dave is easy as he speaks at several conferences and sessions each year, including TechEd, Ignite, MVP Days Community Roadshow, and VeeamOn.

Recently Dave has been honored to take on the role of Conference Co-Chair of TechMentor with fellow MVP Sami Laiho. The lineup of speakers and attendees that have been to this conference over the past 20 years is really amazing. Come down to Redmond or Orlando in 2018 and you can meet him in person.

As the founder and Managing Principal Consultant at TriCon Elite Consulting, Dave is a leading technology expert for both local customers and large international enterprises, providing optimal guidance and methodologies to achieve and maintain an efficient infrastructure.

BLOG: www.checkyourlogs.net

Twitter: @DaveKawula



Sean Kearney - MVP

\$PowerShellMVPBio=@'

"A long time ago in a Cmdlet far far away... there was this guy, who sang about PowerShell"

Here is a person who genuinely loves his job and smiles he gets paid to do what he loves, It's PowerShell MVP Sean Kearney

Presently working for a Microsoft Gold Partner in Ottawa as a Senior Solutions Architect, he lives each and every day for an opportunity to show someone and easier and more consistent way to do their job with Windows PowerShell.

BLOG: <http://www.energizedtech.com/>

Twitter: @energizedtech

Thomas Rayner - MVP

Thomas Rayner is an information technology, entrepreneurship and leadership enthusiast with a penchant for Microsoft tools and products. Thomas is a proud graduate of several programs at NAIT, an institution that he remains actively connected to. He works on the DevOps and Automation team at PCL Construction.

BLOG: <http://workingsysadmin.com>

Twitter: @mrthomasrayner



Allan Rafuse – MVP

Allan has worked as a senior member of the Windows and VMWare Platform Department at Swedbank. He took part in the architecture and implementation of multiple datacenters in several countries. He is responsible for the roadmap and lifecycle of the Windows Server Environment,

including the development of ITIL processes of global server OSD, configuration, and performance.

He is an expert at scripting solutions and has an uncanny ability to reduce complexity and maximize the functionality of PowerShell. Allan has recently rejoined the TriCon Elite Consulting team again as a Principal Consultant.

BLOG: <http://www.checkyourlogs.net>

Twitter: @allanrafuse



Ed Wilson – The Scripting Guy

Ed Wilson is a well-known scripting expert. He is a Microsoft Certified Trainer who has delivered a popular Windows PowerShell workshop to Microsoft Premier customers worldwide. Ed has written six books on Microsoft Windows scripting for Microsoft Press. His three Windows PowerShell books are Windows PowerShell 2.0 Best Practices, Windows PowerShell Scripting Guide, and Microsoft Windows PowerShell Step by Step. He has also written or contributed to almost a dozen other books. Ed holds more than 20 industry certifications, including Microsoft Certified Systems Engineer (MCSE) and Certified Information Systems Security Professional (CISSP). Before coming to work for Microsoft, Ed was a senior consultant for a Microsoft Gold Certified Partner where he specialized in Active Directory design and Exchange Server implementation. Ed and the “Scripting Wife” Teresa live in South Carolina. In his spare time, he enjoys woodworking, underwater photography, and scuba diving. And tea.

BLOG: <https://blogs.technet.microsoft.com/heyscriptingguy/>

Twitter: @scriptingguys

Technical Editors

Cristal Kawula – MVP

Cristal Kawula is the co-founder of MVPDays Community Roadshow and #MVPHour live Twitter Chat. She was also a member of the Gridstore Technical Advisory board and is the President of TriCon Elite Consulting. Cristal is also only the 2nd Woman in the world to receive the prestigious Veeam Vanguard award.

Cristal can be found speaking at Microsoft Ignite, MVPDays, and other local user groups. She is extremely active in the community and has recently helped publish a book for other Women MVP's called Voices from the Data Platform.

BLOG: <http://www.checkyourlogs.net>

Twitter: @supercrystal1



Emile Cabot - MVP

Emile started in the industry during the mid-90s working at an ISP and designing celebrity web sites. He has a strong operational background specializing in Systems Management and collaboration solutions, and has spent many years performing infrastructure analyses and solution implementations for organizations ranging from 20 to over 200,000 employees. Coupling his wealth of experience with a small partner network, Emile works very closely with TriCon Elite, 1E, and Veeam to deliver low-cost solutions with minimal infrastructure requirements.

He actively volunteers as a member of the Canadian Ski Patrol, providing over 250 hours each year for first aid services and public education at Castle Mountain Resort and in the community.

BLOG: <http://www.checkyourlogs.net>

Twitter: @ecabot

Contents

Foreword Ed Wilson “The Scripting Guy”	iii
Acknowledgements.....	v
From Dave	v
About the Authors	vi
Dave Kawula - MVP	vi
Sean Kearney - MVP	vii
Thomas Rayner - MVP	vii
Allan Rafuse – MVP	vii
Ed Wilson – The Scripting Guy	viii
Technical Editors	ix
Cristal Kawula – MVP	ix
Emile Cabot - MVP	ix
Contents.....	xi
Introduction	1
North American MVPDays Community Roadshow	1
Structure of the Book	1
Sample Files	2
Additional Resources	3
Chapter 1.....	5
Simulating a Ransomware Attack with PowerShell	5
But how does one test for Ransomware detection?	5
The Code.....	6
Chapter 2.....	8
Retrieving the Distribution Lists a User is a Member Of	8
The Code.....	9
Chapter 3.....	10

Identifying Large Exchange Mailbox Folders via PowerShell	10
The Code	11
Chapter 4.....	14
Deploying a Domain Joined Nano Server via PowerShell	14
PowerShell Cmdlets for the Network Stack.....	15
Chapter 5.....	19
Enabling Auto Notification of Specific File Changes	19
The Code	19
Chapter 6.....	21
Copying the Output of the last PowerShell command to the clipboard	21
The Code.....	21
Chapter 7.....	23
Validate the Length of An Integer.....	23
The Code	23
Chapter 8.....	24
Allow a Null Value for An Object That Doesn't Normally Allow it	24
Chapter 9.....	25
Getting Started with Pester.....	25
Chapter 10.....	29
Find missing Subnets in AD Sites and Services	29
Cloud Environments	29
Solution	29
Things to thing about.....	30
The Code	30
Chapter 11.....	32
Copy Directories/Files to a list of Remote Computers.....	32
Here-Strings to the Rescue.....	33
Chapter 12.....	36

Making the Configuration Manager PowerShell Module Discoverable.....	36
The Code.....	37
Chapter 13.....	39
Resolving issues with slow Outbound mail in Office 365 in a Hybrid connection using a SmartHost or Internal Appliance.....	39
Chapter 14.....	43
Use PowerShell to Work with Data from OMS.....	43
Parsing MS OMS data with PowerShell	43
Importing the CSV file using PowerShell.....	44
Chapter 15.....	50
Use PowerShell to Explore Office 365 Installation.....	50
Chapter 16.....	53
PowerShell 5 Classes: Constructor Overloading.....	53
What's an overload anyway?	53
Creating overloaded constructors	55
Chapter 17.....	58
Filtering Event Logs with PowerShell	58
Seven parameter sets.....	58
Chapter 18.....	64
Use the PowerShell 5 Convert-String Cmdlet.....	64
Convert-String.....	64
The old flipy dipty.....	64
Expanding the string idea	65
Chapter 19.....	67
Use PowerShell to Detect if a Location is a Directory or a Symlink	67
Chapter 20.....	68
Bypassing PowerShell Execution Policy.....	68

Chapter 20	70
Starting up and Shutting down a list of VM's in a specific order	70
Chapter 21	72
Join us at MVPDays and meet great MVP's like this in person	72
Live Presentations	72
Video Training.....	72
Live Instructor-led Classes.....	72
Consulting Services	73
Twitter.....	74

Introduction

North American MVPDays Community Roadshow

The purpose of this book is to showcase the amazing expertise of our guest speakers at the North American MVPDays Community Roadshow. They have so much passion, expertise, and expert knowledge that it only seemed fitting to write it down in a book.

MVPDays was founded by Cristal and Dave Kawula back in 2013. It started as a simple idea; “There’s got to be a good way for Microsoft MVPs to reach the IT community and share their vast knowledge and experience in a fun and engaging way” I mean, what is the point in recognizing these bright and inspiring individuals, and not leveraging them to inspire the community that they are a part of.

We often get asked the question “Who should attend MVPDays”?

Anyone that has an interest in technology, is eager to learn, and wants to meet other like-minded individuals. This Roadshow is not just for Microsoft MVP’s it is for anyone in the IT Community.

Make sure you check out the MVPDays website at: www.mvppdays.com. You never know maybe the roadshow will be coming to a city near you.

The goal of this particular book is to give you some amazing Master PowerShell tips from the experts you come to see in person at the MVPDays Roadshow. Each chapter is broken down into a unique tip and we really hope you find some immense value in what we have written.

Structure of the Book

The first chapter of this book introduces you to the concepts of HyperConverged infrastructure.

Chapter 1 in this chapter Thomas Rayner shows how to simulate a Ransomware attack using PowerShell.

Chapter 2 Thomas Rayner shows how to retrieve the distribution list members that a user is a member of with PowerShell.

Chapter 3 Thomas Rayner demonstrates how to identify large exchange mailbox folders with PowerShell

Chapters 4 Sean Kearney shows us how to Domain Join a Nano Server with PowerShell.

Chapter 5 Thomas Rayner shows us how to setup auto notifications for a file that has changed. This could easily be turned into an email auto notification for a file that gets dropped into a folder or that is changed.

Chapter 6 Thomas Rayner shows how to copy the last PowerShell command to the clipboard via PowerShell.

Chapter 7 Thomas Rayner validates the length of an integer with PowerShell.

Chapter 8 Thomas Rayner is back again showing us how to allow a null value for an object that doesn't normally allow it.

Chapter 9 Thomas Rayner is back for his final trick showing us how to get starter with Pester.

Chapter 10 Allan Rafuse show us an extremely handy trick to figure out the missing subnets in Active Directory Sites and Services using PowerShell.

Chapter 11 Allan Rafuse shows us how to Copy Directories or files to a list of remote computers using PowerShell.

Chapter 12 Sean Kearney is back to show us how to make the Configuration Manager PowerShell Module Discoverable with this golden master trick.

Chapter 13 Sean Kearney helps us resolve slow outbound mail in Office 365 Hybrid with PowerShell.

Chapter 14 Ed Wilson shows how to work with Data from OMS using PowerShell

Chapter 15 Ed Wilson helps us to explore our Office 365 installation using PowerShell

Chapter 16 Ed Wilson teaches us about Constructor Overloading

Chapter 17 Ed Wilson shows us a master trick to filter event logs using PowerShell

Chapter 18 Ed Wilson helps us to learn about Convert-String and some cool functionality

Chapter 19 Thomas Rayner show us how to detect if a location is a Directory or a Symbolic Link using PowerShell

Chapter 20 Thomas Rayner walks us through some better ways to setup Execution Policies with PowerShell

Chapter 21 Want to meet us in person all the info you need is here.

Sample Files

All sample files for this book can be downloaded from <http://www.checkyourlogs.net>

Additional Resources

In addition to all tips and tricks provided in this book, you can find extra resources like articles and video recordings on our blog <http://www.checkyourlogs.net>.

Chapter 1

Simulating a Ransomware Attack with PowerShell

By: Thomas Rayner – MVP

Ransomware issues have escalated as of late. While there is a common belief that there is no sure-fire way of guaranteeing your organization will never be hit by a ransomware attack, IT administrators should be prepared to detect, stop, and recover from it when it strikes.

But how does one test for Ransomware detection?

While it is ill advised to purposely install ransomware, there are ways to emulate its effects. Conditions that detection software look for include:

- A user that renames more than 100 files
- A user that modifies more than 100 files

1 and 2 happen in under 60 seconds

Once the above happens, ransomware will usually encrypt, modify and append the file extension very quickly.



Figure 1 – Ransomware is a real threat

Note: Many ransomware variants behave in many different ways. The conditions listed above are the more common behaviors documented.

The Code

The following PowerShell script can be used to emulate the above conditions within your lab environment:

```
$strDir = "C:\temp\test1\  
GCI $strDir | Remove-Item -Force  
  
1..200 | % { $strPath = $strDir + $_ + ".txt"; "something" | Out-File $strPath | Out-Null }  
Measure-Command { 1..101 | % { $strPath = $strDir + $_ + ".txt";  
$strNewPath = $strPath + ".chng"; "changed" | Out-File -Append $strPath; Rename-Item -Path $strPath -  
NewName $strNewPath } }
```

The breakdown of this script is as follows:

Lines 1, 2 and 3 setup the environment.

Line 1 assigns \$strDir with the test directory to be monitored for ransomware attacks

Line 2 empties the test directory which you probably don't want to do indiscriminately in a production area but I want to do in my test area

Line 3 creates 200 txt files in \$strDir. 1..200 is a slick way of writing all the numbers between 1 and 200 inclusive. Try it yourself in a PowerShell console. Then, for each of those numbers, we're creating a file and suppressing the output.

Line 4 simulates the ransomware condition. For 101 files, we're making a variable \$strPath which is an individual file we created in line 3. We're also crafting a new path stored in \$strNewPath which is the same file but with an extension. Then I'm changing the contents of the file by writing "changed" inside it. Finally, I rename the file. The whole thing is wrapped in a Measure-Command block so I can see how long it takes.

During my previous test the ransomware part took 688 milliseconds.

```
Days          : 0
Hours         : 0
Minutes      : 0
Seconds      : 0
Milliseconds  : 688
Ticks        : 6887630
TotalDays    : 7.97179398148148E -06
TotalHours   : 0.000191323055555556
TotalMinutes : 0.0114793833333333
TotalSeconds : 0.688763
TotalMilliseconds : 688.763
```

Test this in lab for yourself and see if you can detect this simulated ransomware attack.

Chapter 2

Retrieving the Distribution Lists a User is a Member Of

By: Thomas Rayner – MVP

Let's say you have a user and want to get all the distribution lists this user is a member of. Well the following PowerShell script can help you with that. This can be very handy because sometimes a user can mistakenly get added to the wrong Distribution List.



Figure 2 – Exchange Distribution List members

The Code

The following PowerShell script can be used to emulate the above conditions within your lab environment:

```
$DN = 'CN=ThmsRynr,OU=BestUsers,DC=lab,DC=workingsysadmin,DC=com'  
Get-DistributionGroup -filter "members -eq '$DN'" | Select-Object Name,@ {  
l='Members';e={ ( Get-DistributionGroupMember $_.SamAccountName -ResultSize Unlimited | % { $_.Name } )  
-join ';' } }
```

Line 1 declares the variable to hold the DistinguishedName attribute for the user of interest. Line 2 competes the grunt work of searching. This gets all the distribution groups which have a member equal to the DN of the user of interest.

The above script positions the Distribution Groups return into a SelectObject cmdlet to report the Name property and then a custom column. The label is Members and the content is going to just be a string of all the Distribution Group members' names separated by semicolons. The expression for my custom column is a Get-DistributionGroupMember command for the Distribution Group piped into a Foreach-Object (alias is "%") which returns an array of all the names of the members in the Distribution Group. I use the -join command to take the array and convert it into a string separated by semicolons.

When Get-DistributionGroup is utilized, you do not get the members of that group back with it. Here are the properties that come back that contain the string mem in the name.

```
PS C:\> Get-DistributionGroup -filter "members -eq '$DN'" | Select-Object -First 1 | GetMember |  
Where-Object Name -match 'mem' | Select-Object Name
```

Name

—

AcceptMessagesOnlyFromDLMembers

AcceptMessagesOnlyFromSendersOrMembers

AddressListMembership

BypassModerationFromSendersOrMembers

MemberDepartRestriction

MemberJoinRestriction

RejectMessagesFromDLMembers

RejectMessagesFromSendersOrMembers

Nothing in there contains the members and so the initial script provided in this post is utilized.

Chapter 3

Identifying Large Exchange Mailbox Folders via PowerShell

By: Thomas Rayner - MVP

Identifying users with large Exchange mailboxes is a task undertaken by most system administrators who are in need of freeing up space on their mail servers. While most search for mailboxes approaching a certain size, I wanted to take this a step further and identify the large folders within user mailboxes. An example of this would be to find all the users who have a large Deleted Items folder or Sent Items or Calendar that would be eligible to be cleaned out. It's made to be run from a Remote Exchange Management Shell connection instead of by logging into an Exchange server via remote desktop and running such a shell.



Figure 3 – Large Mailbox Folders

The Code

Let's get started by defining the function and parameters

```
{
    [CmdletBinding()]
    param (
        [Parameter(Mandatory = $False)]
        [ValidateSet('All', 'Calendar', 'Contacts', 'ConversationHistory',
'DeletedItems', 'Drafts', 'Inbox', 'JunkEmail', 'Journal',
'LegacyArchiveJournals', 'ManagedCustomFolder', 'NonIpMRoot', 'Notes', 'Outbox',
'Personal', 'RecoverableItems', 'RssSubscriptions', 'SentItems', 'SyncIssues',
'Tasks')]
        [string]$FolderScope = 'All',
        [Parameter(Mandatory = $False)]
        [int]$Top = 1,
        [Parameter(Mandatory = $False,
            Position = 1,
            ValueFromPipeline = $True)]
        [string]$Identity = '*'
    )
}
```

This function is to be named Get-LargeFolder and takes three parameters.

\$FolderScope is used in the Get-MailboxFolderStatistics cmdlet and must belong to the set of values specified.

\$Top is an integer used to define how many results we're going to return

\$Identity can be specified as an individual username to examine a specific mailbox, or left blank (defaulted to *) to examine the entire organization.

```
function Get-LargeFolder
{
    [CmdletBinding()]
    param (
        [Parameter(Mandatory = $False)]
        [ValidateSet('All', 'Calendar', 'Contacts', 'ConversationHistory',
'DeletedItems', 'Drafts', 'Inbox', 'JunkEmail', 'Journal',
'LegacyArchiveJournals', 'ManagedCustomFolder', 'NonIpMRoot', 'Notes', 'Outbox',
'Personal', 'RecoverableItems', 'RssSubscriptions', 'SentItems', 'SyncIssues',
'Tasks')]
        [string]$FolderScope = 'All',
        [Parameter(Mandatory = $False)]
        [int]$Top = 1,
        [Parameter(Mandatory = $False,
            Position = 1,
            ValueFromPipeline = $True)]
        [string]$Identity = '*'
    )

    Get-Mailbox -Identity $Identity -ResultSize Unlimited |
    Get-MailboxFolderStatistics -FolderScope $FolderScope
}
```

Next I've added a lines to retrieve all of my organizations mailboxes which I direct the data stream into a **Get-MailboxFolderStatistics** command with the FolderScope parameter set to the same value we passed to our function. Now we need to sort the results.

```
function Get-LargeFolder
{
    [CmdletBinding()]
    param (
        [Parameter(Mandatory = $False)]
        [ValidateSet('All', 'Calendar', 'Contacts', 'ConversationHistory',
        'DeletedItems', 'Drafts', 'Inbox', 'JunkEmail', 'Journal',
        'LegacyArchiveJournals', 'ManagedCustomFolder', 'NonIpMRoot', 'Notes', 'Outbox',
        'Personal', 'RecoverableItems', 'RssSubscriptions', 'SentItems', 'SyncIssues',
        'Tasks')]
        [string]$FolderScope = 'All',
        [Parameter(Mandatory = $False)]
        [int]$Top = 1,
        [Parameter(Mandatory = $False,
        Position = 1,
        ValueFromPipeline = $True)]
        [string]$Identity = '*'
    )

    Get-Mailbox -Identity $Identity -ResultSize Unlimited |
    Get-MailboxFolderStatistics -FolderScope $FolderScope |
    Sort-Object -Property @{
        e = {
            $_.FolderSize.split('(').split(' ')[-2].replace(',','') -as [double]
        }
    } -Descending
}
```

The FolderSize parameter returns with a **Get-MailboxFolderStatistics** cmdlet which is a string to be split up to provide only the value in bytes which I am casting to a double. Once the stats have been gathered and set in order, they need to be selected so they can be returned.

Here is the complete script:

```
function Get-LargeFolder
{
    [CmdletBinding()]
    param (
        [Parameter(Mandatory = $False)]
        [ValidateSet('All', 'Calendar', 'Contacts', 'ConversationHistory',
        'DeletedItems', 'Drafts', 'Inbox', 'JunkEmail', 'Journal',
        'LegacyArchiveJournals', 'ManagedCustomFolder', 'NonIpMRoot', 'Notes', 'Outbox',
        'Personal', 'RecoverableItems', 'RssSubscriptions', 'SentItems', 'SyncIssues',
        'Tasks')]
        [string]$FolderScope = 'All',
        [Parameter(Mandatory = $False)]
        [int]$Top = 1,
```

```

        [Parameter(Mandatory = $False,
                   Position = 1,
                   ValueFromPipeline = $True)]
        [string]$Identity = '*'
    )

    Get-Mailbox -Identity $Identity -ResultSize Unlimited |
    Get-MailboxFolderStatistics -FolderScope $FolderScope |
    Sort-Object -Property @{
        e = {
            $_.Foldersize.split('(').split(' ')[-2].replace(',','') -as [double]
        }
    } -Descending |
    Select-Object -Property @{
        l = 'NameFolder'
        e = {
            $_.Identity.Split('/')[-1]
        }
    },
    @{
        l = 'FolderSize'
        e = {
            $_.Foldersize.split('(').split(' ')[-2].replace(',','') -as
[double]
        }
    } -First $Top
}

```

You can now figure out your large mailbox folders by executing the following:

```

#Get 25 largest Deleted Items folders in your organization
Get-LargeFolder -FolderScope 'DeletedItems' -Top 25

#Get my largest 10 folders
Get-LargeFolder -Identity ThmsRynr -Top 10

#Get the top 25 largest Deleted Items folder for users in a specific group
$arrLargeDelFolders = @()
(Get-ADGroupMember 'GroupName' -Recursive).SamAccountName | ForEach-Object -
Process {
    $arrLargeDelFolders += Get-LargeFolder -FolderScope 'DeletedItems' -Identity
    $_
}
$arrLargeDelFolders |
Sort-Object -Property FolderSize -Descending |
Select-Object -Property NameFolder, @{
    l = 'FolderSize (Deleted Items)'
    e = {
        '{0:N0}' -f $_.FolderSize
    }
} -First 25 |
Format-Table -AutoSize

```

Chapter 4

Deploying a Domain Joined Nano Server via PowerShell

By: Sean Kearney - MVP

Recently I've put together a PowerShell module called DeployImage with the intent to simplify the deployment of a WindowsImage file. In this case, my goal was to make NanoServer an easily deployable option for the average system administrator. So began my experimentation with Windows Server 2016 to get a fully deployed Nano server online.

Deploying Nano Server is no different than deploying any other WIM file except for the fact that you must compensate for is it is headless environment.



Figure 4 – Nano Server

This requires a plan to have certain tasks completed within the server without actually touching it. These tasks include:

Assigning a Static IP address

Naming the workstation

Joining it to a Domain

Most of this could be completed through PowerShell remotely via WinRM and adding it to TrustedHosts. However, the preference here is to have the system up and running and completed in a more fully automated fashion.

The following cmdlet to obfuscate the creation of the XML within the DeployImage module was added. The following Cmdlet can be used to create an unattend.xml:

```
New-UnattendXMLContent -Computersname Contoso-Nano1 -Timezone  
'Eastern Standard Time' -Owner 'Contoso' -Organization 'Contoso' -  
AdminPassword 'P@ssw0rd'
```

This will generate the XML content for a Computer with the following specs

```
Name       : Contoso-Nano1  
TimeZone   : Eastern Standard Time  
Owner      : Contoso  
Organization : Contoso  
Password   : P@ssw0rd
```

(The Password referred to is the Default Administrator account)

Once completed, the Unattend.xml file will need to be copied into the Destination file structure under C:\Windows\system32\sysprep.

PowerShell Cmdlets for the Network Stack

Next, we'll need to assign a static IP address for said server.

PowerShell Cmdlets for the Network Stack

Configuring the IP address using Unattend.xml

Nano Server can be accessed directly through the text console and can be configured with an IP address post install.

At present NetSH.exe can still be utilized to configure the required settings.

The default network adapter name in Nano Server is called Ethernet. In this scenario the following will be assigned to said Nano Server:

IPv4 Address : 192.168.1.10
Subnet : 255.255.255.0
Gateway : 192.168.1.1
DNS Server : 192.168.1.5

These settings can be assigned with two lines from NetSh.exe

```
netsh interface ipv4 set address Name="Ethernet" static 192.168.1.10  
255.255.255.0 192.168.1.1  
netsh dns set dnsservers name="Ethernet" source=static  
address=192.168.1.5
```

To configure this once the Nano server boots up requires a script called SetupComplete.cmd which exists at C:\Windows\Setup\Scripts.

Upon initial startup after processing Unattend.xml and before the login screen the script will execute. So, we can build this script to auto configure our network with a little PowerShell and a HereString.

```
$IPAddress='192.168.1.10'  
$Subnet='255.255.255.0'  
$Gateway='192.168.1.1'  
$DNS='192.168.1.5'  
$SetupCompleteCMD="@  
netsh interface ipv4 set address Name="Ethernet" static $IPAddress $Subnet  
$Gateway  
netsh dns set dnsservers name="Ethernet" source=static address=$DNS  
"@  
New-Item -ItemType File -Name SetupComplete.cmd -Force | Out-Null  
Add-content SetupComplete.cmd -value $SetupCompleteCMD
```

At the time of writing this Nano Server is STILL in technical preview and so unfortunately some options can't use at this time. First there is no directly way through the Emergency console to add this to a Domain. In fact, there is no command that creates the account in Active Directory. Nano Server does however support an Offline Domain join.

Establishing an offline Domain join requires the following three steps:

1. Create the offline Join file
2. Copy the file to the workstation/server
3. Execute an offline Join with the provided file

In this scenario we would like to join a workstation to a Domain with the following settings.

Domain : Contoso
Computer : Contoso-Nano1
Filename : domainjoin.djoin

Djoin.exe command is needed to create the file on the computer with the RSAT tools for Active Directory. This can be run manually for the stated configuration in the following manner:

```
Djoin.exe /Provision /Domain Contoso /Machine Contoso-Nano1 /Savefile domainjoin.djoin
```

This can also be run in PowerShell by providing objects to store the information.

```
$Domain='Contoso'  
$Computersname='Contoso-Nano1'  
$Filename='domainjoin.djoin'  
Djoin.exe /Provision /Domain $Domain /Machine $Computersname /Savefile $Filename
```

Next the file needs to be copied to the destination file system. Ideally the file needs to be in the same folder as the Setup\Scripts folder.

The following command needs to be run on the destination system directly to join this system to a Domain and does not require to be on the network to make this work once the file is the destination system as this is an offline domain join

```
Djoin.exe /RequestODJ /loadfile C:\Windows\setup\scripts\domainjoin.djoin  
/windowspath c:\windows /localos
```

Now to make this work we'll be using the same process as detailed with SetupComplete.cmd and a HereString only we'll be appending it to the NetSh.exe content.

```
$Filename='C:\windows\setup\scripts\domainjoin.djoin'  
$SetupCompleteCMD=@"  
netsh interface ipv4 set address Name="Ethernet" static $IPAddress $Subnet  
$Gateway  
netsh dns set dnsservers name="Ethernet" source=static address=$DNS djoin /requestodj  
/loadfile $Filename /windowspath c:\windows /localos shutdown -f -r -t 0  
"@  
# Create the new one  
#  
Remove-Item -Path SetupComplete.cmd -Force -ErrorAction SilentlyContinue  
New-Item -ItemType File -Name SetupComplete.cmd -Force | Out-Null  
Add-content SetupComplete.cmd -Value $SetupCompleteCMD
```

With the Unattend.xml and the Setupcomplete.cmd in the appropriate locations the boot code needs to be injected into the Nano Server to boot up, be named as it should be, have an IP address assigned and joined to the appropriate Domain.

If you'd wish to take a deeper look at the script performing this in action just access: DeployImage from www.powershellgallery.com.

Once you install the module, which has been Windows 10 tested with the Windows 10 ADK installed, you can execute the following Cmdlet to get the sample scripts.

Copy-DeployImageSample

Just open up the script called DeployNanoServerVHDDomain.ps1 and run it.

Chapter 5

Enabling Auto Notification of Specific File Changes

By: Thomas Rayner – MVP

Recently I was asked...

“How do I send an email automatically whenever a change is made to a specific file?”

There are many 3rd party solutions to achieve this. I however wanted to take the opportunity to learn of a way to complete this task via PowerShell. Setting up a file watcher seemed to address the issue and its was actually simple to script.

The Code

The following was created to watch the intended file for changes:

```
$watcher = New-Object System.IO.FileSystemWatcher
$watcher.Path = 'C:\temp\'
$watcher.Filter = 'test1.txt'
$watcher.EnableRaisingEvents = $true

$changed = Register-ObjectEvent
    $watcher 'Changed' -Action {
write-output "Changed: $($EventArgs.FullPath)"
}
```

Creating the file watcher took the utilization of the FileSystemWatcher object. This script specifically watches the entire directory and path for changes. An added filter is added to specify the required file.

Next an ObjectEvent is registered to perform an action the watcher detects a change event. This script highlights a simple output however it could easily be modified to enable sending of an email or performing some other task.

Removing the ObjectEvent is just as easy and can be performed using the following script:

`Unregister-Event $changed.Id`

Chapter 6

Copying the Output of the last PowerShell command to the clipboard

By: Thomas Rayner – MVP

I recently needed to copy and paste a PowerShell script output. While poking around in PowerShell, I discovered that both trying to copy and paste it out of PowerShell or hitting the up arrow and piping whatever the last command was into Set-Clipboard was such a hassle!



Figure 5 – Microsoft Clippy

The Code

So to address this, I threw the following small function into my profile:

```
function cc { r | scb }
```

Now PowerShell 5.0 is needed to evoke this. Specifically for use of Set-Clipboard. Lets now break down the workings of the small function.

First I'm defining a function named cc which is not a properly named PowerShell function but for now it will do the trick. Its assigned `r | scb` in which `r` is an alias for `Invoke-History` which re-runs the last command typed. Try it yourself:

```
PS G:\> write-output "hah!"  
hah!
```

```
PS G:\> r  
write-output "hah!"  
hah!
```

```
PS G:\> r  
write-output "hah!"  
hah!
```

Last but not least `scb` is an alias for `Set-Clipboard` which means whatever came out of the last command will be the new contents of your clipboard.

Chapter 7

Validate the Length of An Integer

By: Thomas Rayner – MVP

A little while ago, I fielded a question in the PowerShell Slack channel which was “How do I make sure a variable, which is an int, is of a certain length?”

Turns out it’s not too hard. You just need to use a little regex. Consider the following example.

The Code

```
[int] $v6 = 849032
[int] $v2 = 23
$v6 -match '^d{6}$'
$v2 -match '^d{6}$'
```

\$v6 is an int that is six digits long. \$v2 is an int that is only two inches long. On lines three and four, we’re testing to see if each variables match the pattern ‘^d{6}\$’ which is regex speak for “start of the line, any digit, and six of them, end of the line”. The first one will be true, because it’s six digits, and the second one will be false. You could also use something like ‘^d{4,6}\$’ to validate that the int is between four and six digits long.

Chapter 8

Allow a Null Value for An Object That Doesn't Normally Allow it

By: Thomas Rayner – MVP

In the PowerShell Slack channel (powershell.slack.com) a question came up along the lines of “I have a script that needs to pass a datetime object, but sometimes I’d like that datetime object to be null”. Never mind that maybe the script could be re-architected. Let’s solve this problem.

The issue is, if you try to assign a null value to a datetime object, you get an error.

```
[datetime]$null  
Cannot convert null to type "System.DateTime".
```

The solution is super easy. Just make the thing nullable.

```
[nullable[datetime]]$null
```

This will return no output. So when you’re declaring the variable that will hold your datetime object, just make sure you make it nullable.

```
[nullable[datetime]]$date = $MaybeNullMaybeNot
```

Just for more proof this works as advertised, try this.

```
[nullable[datetime]]$date = $MaybeNullMaybeNot  
try { [datetime]$null; write-output 'worked!' } catch { write-output 'no  
worked!' }  
no worked!  
try { [nullable[datetime]]$null; write-output 'worked!' } catch { write-output  
'no worked!' }  
worked!
```

Cool!

Chapter 9

Getting Started with Pester

By: Thomas Rayner - MVP

If you don't know what Pester is, it's a framework for running unit tests and validating PowerShell code. Also, it's awesome. In May I finally dipped my toe in the water with a pretty simple test for a REALLY simple function. I'm not going to go into a world of detail on how exactly all my Pester code works because there are tons of guides for that. What I'm going to do instead is provide a quick run down of what I came up with.

First things first, I need a function to validate.

```
function Write-SomeMath
{
    param(
        [int]$First,
        [int]$Second
    )
    return $First + $Second
}
```

I guess that will work. Write-SomeMath takes two integers and returns their sum. Hardly a breathtaking display of complexity and function but it will do just fine for this example.

Now I need to install Pester. The easiest way to do this is using the PSGet module in PowerShell 5.0 to get it from PowerShellGallery.com.

```
Install-Module Pester -Scope CurrentUser -Force
Import-Module Pester
```

This Describe block will contain and – you guessed it – describe the tests (I just used my filename) and provide a unique TestDrive (check out the getting started link).

Now I need a Context block.

```
Describe 'GoofingWithPester.ps1' {
    Context 'Write-SomeMath' {
    }
}
```

I'm further grouping my tests by creating a Context here for my Write-SomeMath function. This could have been named anything.

Now, I could start with a bunch of tests, but I want to show off a particular feature of Pester that allows you to pass an array of different test cases.

```
Describe 'GoofingwithPester.ps1' {
    Context 'write-SomeMath' {
    }
}

Describe 'GoofingwithPester.ps1' {
    Context 'write-SomeMath' {
        $testcases = @(
            @{
                fir = 1
                sec = 2
                exp = 3
                test = '1 and 2'
            },
            @{
                fir = 3
                sec = 6
                exp = 91 #wrong on purpose
                test = '3 and 6 (wrong on purpose)'
            },
            @{
                fir = 4
                sec = 6
                exp = 10
                test = '4 and 6'
            }
        )
    }
}
```

All I did was define an array called \$testcases which holds an array of hash tables. It's got the first number, second number, expected result and a name of what we're testing. Now I can pass this entire array to a test rather than crafting different tests for all of them individually.

```
Describe 'GoofingwithPester.ps1' {
    Context 'write-SomeMath' {
        $testcases = @(
            @{
                fir = 1
                sec = 2
                exp = 3
                test = '1 and 2'
            },
            @{
                fir = 3
                sec = 6
                exp = 91 #wrong on purpose
                test = '3 and 6 (wrong on purpose)'
            },
            @{
                fir = 4
                sec = 6
                exp = 10
                test = '4 and 6'
            }
        )
    }
}
```



```

    It 'Can add <test>' -TestCases $testcases {
        param($fir,$sec,$exp)
        Write-SomeMath -First $fir -Second $sec | Should Be $exp
    }
}

```

This is an It block which is what Pester calls a test. I’ve named it “Can add <test>” and it will pull the “test” value from the hashtable and fill it in. Cool! I’m using the -TestCases parameter to pass my array of test cases to the It block. Then I’ve got parameters inside the test for my first value, second value and expected outcome. I execute Write-SomeMath with the values pulled from my test cases and pipe the result to “Should Be” to compare the outcome to my expected outcome.

Now, just one more test for fun. What if I don’t pass an integer to my function?

```

Describe 'GoofingwithPester.ps1' {
    Context 'write-SomeMath' {
        $testcases = @(
            @{
                fir = 1
                sec = 2
                exp = 3
                test = '1 and 2'
            }
            @{
                fir = 3
                sec = 6
                exp = 91 #wrong on purpose
                test = '3 and 6 (wrong on purpose)'
            }
            @{
                fir = 4
                sec = 6
                exp = 10
                test = '4 and 6'
            }
        )
    It 'Can add <test>' -TestCases $testcases {
        param($fir,$sec,$exp)
        Write-SomeMath -First $fir -Second $sec | Should Be $exp
    }
    It 'Detects wrong datatypes' {
        {Write-SomeMath -First 9 -Second 'cat'} | Should throw
    }
}
}

```

Another It block for detecting wrong datatypes. I pipe the result into Should throw because my function should throw an error. For this to work properly, the code I’m testing has to be wrapped in a scriptblock, otherwise the thrown error will occur and be trapped in my function.

Here's the outcome when I run this file!

```
Describing GoofingWithPester.ps1
Context Write-SomeMath
  [+] Can add 1 and 2 183ms
  [-] Can add 3 and 6 (wrong on purpose) 74ms
    Expected: (9)
    But was:  (9)
    35:         Write-SomeMath -First $fir -Second $sec | Should Be $exp
    at <ScriptBlock>, \\j63p7v1\es\PowerShell\repos\tr-area-51\Puzzles\GoofingWithPester.ps1: line 35
  [+] Can add 4 and 6 107ms
  [+] Detects wrong datatypes 79ms
```

Figure 6 – Pester Output

Pretty cool. My first test passes, the second one fails and tells me why, the third and fourth tests pass. The fourth one is especially interesting. The function FAILED but because the test said it SHOULD FAIL, the test itself passed.

So, that's my “dip my toes in the water” intro Pester test. Stay tuned for more complicated examples.

Chapter 10

Find missing Subnets in AD Sites and Services

By: Allan Rafuse – Future MVP

This has been a very common pain point for Active Directory administrators. AD is perfectly planned according to Microsoft's best practices and successfully deployed. But as time goes on, network admins change the network topology, devices are added here and there and if there is no formal process of adding new networks, AD Sites and Services will mostly likely not be updated to reflect these changes.

Cloud Environments

Where I've seen this problem popping up again and again is in Azure or other virtual environments. Why? This is due to the blurring of traditional job responsibilities between network engineer and systems engineer/administrators. Admins can now log into their management consoles and easily create new networks, assign VLANS and deploy a VM to them. Life is grand, what took a few minutes used to take hours or days.

If AD Sites and Subnets are not correctly defined, a device will attempt to authenticate with a domain controller, randomly somewhere on your network. If the DC see's a request from a device and it cannot locate it's subnet, it reports this into a simple text file, C:\Windows\Debug\netlogon.log. Viewing this text file can be tedious to go through and has to be done on each domain controller.

Solution

The following PowerShell code is quite straight forward but was built to run from a single server and remotely connect to each DC. It will remotely connect to a given DC and show any missing IPs from that file.

The basic idea is to

1. Remotely read the C:\Windows\Debug\Netlogon.log text file into a variable
2. Read from the bottom of the file contents until we hit \$FromDate (which is 30 days in the example)
3. Build an array of a simple PowerShell object
4. Once you have the results, you can then display it to the screen, Out-File it to a file server for central reporting, or even email it to yourself

The downside to the log file is that it displays the raw IP. This is because the AD server does not know the network topology and subnet mask of the devices failing to find their proper DC. This holds true for the PowerShell code below. It will display all IPs that have attempted, but you will still have to make the final decision on what type of network and netmask you assign in Active Directory Sites and Services.

Things to think about

After running this script you'll hopefully find and fix everything. Perhaps you may want to think about creating a script, run it as a scheduled task and centralize the results for further auditing.

The Code

```
$FromDate = (Get-Date).AddDays(-30)
$content = Get-Content "\\$DomainController\c$\windows\Debug\netlogon.log"

# Run through the netlogon.log (in reverse order, think about speed/performance) while the
# dates are greater than $FromDate
$MissingEntry = @{}
For ($counter = $content.Count; $counter -ge 0; $counter--) {
    If ($content[$counter] -match "(\\d\\d)/(\\d\\d) (\\d\\d):(\\d\\d):(\\d\\d)") {
        $EntryDate = Get-Date -Month $Matches[1] -Day $Matches[2] -Hour $Matches[3] -
Minute $Matches[4] -Second $Matches[5]
        if ($EntryDate -lt $FromDate) {
            break
        }
        # within the timeframe, let's save the IP and Date attempted in a hashtable. Only
        # keep the first hit, which is the latest failed site attempt
        $ip = $content[$counter] -Replace ".* (.*)$", '$1'
        If ($MissingEntry[$ip] -eq $null) {
            $MissingEntry[$ip] = $EntryDate
        }
    }
}

# Sort the missing IPs
$MissingEntry = $MissingEntry.GetEnumerator() | Sort-Object -Property Name

# Output the missing IPs and failed date attempt
$MissingEntry | Select-Object @{name="DC"; expression={$DomainController}}, @{name="IP";
expression={$_.Name}}, @{name="Last Failed Site Attempt"; expression={$_.Value}}
```

Here are the results from \$MissingEntry

DC	IP	Last Failed Site Attempt
--	--	-----
DemoDC01	192.168.100.11	10/26 21:35:08
DemoDC01	192.168.31.100	10/26 22:01:35

DemoDC01	192.168.31.110	10/26 21:53:16
DemoDC01	172.16.2.100	10/26 17:07:02
DemoDC01	172.16.2.101	10/26 22:05:55
DemoDC01	172.16.2.103	10/26 21:49:23
DemoDC01	172.16.2.104	10/26 21:38:14
DemoDC01	172.16.2.11	10/26 22:00:28
DemoDC01	172.16.2.12	10/26 21:55:08
DemoDC01	172.16.2.13	10/26 21:39:09
DemoDC01	172.16.2.14	10/26 22:01:40
DemoDC01	172.16.2.15	10/26 21:46:33
DemoDC01	172.16.2.16	10/26 22:03:43
DemoDC01	172.16.2.17	10/26 22:02:22
DemoDC01	172.16.2.18	10/26 21:34:52
DemoDC01	172.16.2.19	10/26 21:53:05
DemoDC01	172.16.2.20	10/26 14:10:05
DemoDC01	172.16.2.21	10/26 22:06:03
DemoDC01	172.16.2.22	10/26 22:06:25
DemoDC01	172.16.2.23	10/26 21:53:03
DemoDC01	172.16.2.24	10/26 16:32:10
DemoDC01	172.16.2.25	10/26 21:59:13
DemoDC01	172.16.4.10	10/26 22:04:26
DemoDC01	172.18.244.58	11/02 18:23:53
DemoDC01	172.18.65.232	11/02 18:23:55

Chapter 11

Copy Directories/Files to a list of Remote Computers

By: Allan Rafuse – Future MVP

Here is a quick code block that I use quite often to copy directories from a management server to a ton of other servers.

Copy-Item [-Recurse]



Figure 7 – Copy and Paste like the pro's

For the pros or even a beginner, you're done, you've got the command you wanted... but did you think about the rest of the information in this blog? Keep reading!

The problem is a request comes across your desk, more often than we want to admit, saying "We" need to update this file or files on these completely random machines. Then you given a list of servers and you need to quickly copy files to them.

1. Save the list of servers to a text file, read it in with Get-Content. This works cumbersome when in RDP and PS Remitting Environments

```
content = Get-Content C:\Temp\SomeRandomFileCopy176.txt
```

2. Reformat them into a PowerShell array by putting quotes around each of them and commas. Example:

```
$Computers = @(
    "Server01",
    "Server02",
    "Server03.mydomain.local",
    "Server06"
)
```

3. Using/modifying the code at the bottom of this blog post

Myself, I love scripts, I love modules, but what I love more than anything is the flexibility of PowerShell. I even love copying and pasting blocks or one liners directly into PowerShell. So for the copy task presented above here is the magic that my mind goes through:

1. Which machines do I need to copy files to
2. Are all the machines online
3. Test that the destination share path is reachable on all machines
4. Fix any errors with the remote destinations or simply remove them from the list
5. Copy the files

Here-Strings to the Rescue

Normally when you create a string in PowerShell, or an array it has some form of structure it has to follow. With a Here-String,

```
$MyHereString = @"
    This can
be some random te    text    that
    and it's formatting will be preserved when the Here-String variable
    is used"
"@
$MyHereString
```

So instead of creating some random temp file on the management server every time, I simply open my Microsoft OneNote and copy and paste the code below. But the technique I use to manage the list of machines is called a PowerShell “Here-String”. Then further down in the code will

“convert” the Here-String into an array so that we can easily loop through it. In my opinion when using reusable code blocks, PowerShell Here-Strings are the most forgiving and easiest/fastest to work with as they maintain formatting. To use a Here-String, the unformatted text, in this case a list of machines, are encapsulated between a starting “@” tag and a close “@”.

Note The closing “@” MUST BE on its own line and in column 1.

Again, this is code I copy and paste directly into a PS Window. It allows flexibility when logging into a locked down Remote Desktop environment or when using PowerShell remoting.

Here is the sample script to copy a directory from a source server to the servers listed in the \$Computers Here-String below.

```
$Computers =
@"
    Server01
    Server02
Server03.mydomain.local
    Server06
    Server07
    Server08
    Server09
    Server26
    Server27
        Server28
    Server29
        Server30.mydomain.local
    Server35
    Server36
    Server45
    Server46
    Server112
Server113
    Server114
"@

# Here we will set the source and remote share location and path
$SourceDir = "D:\Software\Datadog\v5.8.5-x64-privatefix"
$RemoteSharePath = "D$\Software\Datadog"

# Loop through the Here-String, trim each line and build the ComputersArray
$ComputersArray = @()
$Computers.Split("`n") | % { if ($_.Trim()) { $ComputersArray += $_.Trim() } }
$AllRemoteSharePathsExist = $True

# Do a precheck to see if all the remote directories are reachable and exist
$ComputersArray | % {
    If ((Test-Path "\\$_\$RemoteSharePath") -eq $False) {
        Write-Host -ForegroundColor Red "[PreCheck Error] \\$_\$RemoteSharePath
does not exist"
        $AllRemoteSharePathsExist = $False
    }
}

# If the precheck passes, then copy and overwrite existing files
If ($AllRemoteSharePathsExist) {
    $ComputersArray | % { Write-Host "[$_] Copying $SourceDir to
\\$_\$RemoteSharePath"; Copy-Item -Path "$SourceDir" -Destination
"\\$_\$RemoteSharePath" -Recurse -Force }
} Else {
```



```
} write-Host "Errors Exist, Exiting"
```

Happy Copying!

Chapter 12

Making the Configuration Manager PowerShell Module Discoverable

By: Sean Kearney – MVP

I'm not going to ENABLE-RANT today. Anybody that uses System Center Configuration Manager knows one thing, you need to MANUALLY add in the Cmdlets to use them using this line

```
Import-Module 'C:\Program Files (x86)\Microsoft Configuration Manager\AdminConsole\bin\ConfigurationManager.psd1'
```

True, it's irritating that I can just do something like this in a modern server

```
Import-Module ConfigurationManager
```

If this were there not only would it be EASIER to access the module locally, it would also allow me to create and establish Remote PowerShell sessions to EASILY work with Configuration Manager and make Implicit remoting far smoother.

But rather than get upset, I decided to sit down and find out why. Why if I do THIS

```
Get-Module -listavailable ConfigurationManager
```

Does it NOT find the module?

The answer was surprisingly easy (and a bit silly it got missed by somebody)

For your module to be discoverable it needs to follow two rules

The PSD1 file must be in a folder of the same name

This Folder must be added to the System Environment variable "PSModulePath"

So examining my default PSModulePath on a Configuration Manager 1606 server It was DEFINITELY not there (yes, fully patched, rebooted multiple times)

```
$ENV:PSModulePath
```

```
C:\Users\Administrator.SYSTEMCTR\Documents\WindowsPowerShell\Modules;C:\Program Files\WindowsPowerShell\Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules;C:\Program Files (x86)\Microsoft SQL Server\110\Tools\PowerShell\Modules\
```

So initially I thought I would just add it to the PSModulePath in the following fashion while in a PowerShell prompt as Administrator.

```
$ENV:PSModulePath=($ENV:PSModulePath) + ';C:\Program Files (x86)\Microsoft Configuration Manager\AdminConsole\bin\'
```

Tried that and looked and still nothing.... Oh, right forgot the FIRST rule

The PSD1 file must be in a folder of the same name

Our problem is that all of the needed files and content exist in the BIN folder and it needs to be renamed to ConfigurationManager. But obviously, we can't do THAT or things will break.

We COULD just make a copy of that folder called "ConfigurationManager" and add THAT to the path. That might work too. But that offers up other scary words.

"Supportability" (Microsoft probably won't support that)

"Patching" (If it DID work, we'd have to dupe that folder EACH and EVERY TIME we patched)

"Works?" (Moving supporting Binaries in ANY major application is just Ripe with Risk)

The it dawned on me. Since VISTA (Stop cringing) we've had Junctions. A Junction is a "Pseudo Folder" which ACTS like a Directory but points off to another Directory.

We COULD create a Junction called "ConfigurationManager" and point it RIGHT BACK to the folder with all of the goodies needed in the Configuration Manager module.

The Code

Well guess what... it works! Here's the Script in PowerShell to make this useful for you as well.

```
# Create a Junction to act as a folder called "Configuration Manager" pointing to the
# 'Bin' folder which has the Actual Configuration Manager module
# This provides a "Pseudo Folder" to match the Module Name needed

$TargetFolder='C:\Program Files (x86)\Microsoft Configuration Manager\AdminConsole\bin\'
$TargetPath='C:\Program Files (x86)\Microsoft Configuration Manager\AdminConsole\PSModule\'

# Verify this has not been done yet, if it has go ahead
If (!(Test-Path "$($TargetPath)ConfigurationManager\"))
{
New-Item -ItemType Directory -Path $TargetPath -Force
CMD.EXE /C MKLINK /J "$($TargetPath)ConfigurationManager" $TargetFolder
# You can use this line INSTEAD of MKLINK if you are running
# PowerShell 5
```

```
#New-Item -ItemType Junction -Name ConfigurationManager -value $TargetFolder -
Path $TargetPath
}
# Permanently update the PSMODULEPATH Environment variable
# With the parent path holding the Configuration Manager folder
# This will make it fully discoverable
$OldPSPATH=(Get-ItemProperty -Path
'Registry::HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session
Manager\Environment' -Name PSMODULEPATH).PSMODULEPATH
# Verify this has not been done yet, if it has go ahead
IF (!(($OldPSPATH | Select-String -SimpleMatch
"$($TargetPath)ConfigurationManager"))
{
$NewPSPATH=$OldPSPATH+';'+'$TargetPath+ConfigurationManager'
Set-ItemProperty -Path
'Registry::HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\Session
Manager\Environment' -Name PSMODULEPATH -Value $NewPSPATH
}
$ENV:PSMODULEPATH=$NewPSPATH
```

Now I can just run `Import-Module ConfigurationManager` and Woohoo! I'm off to races!

Chapter 13

Resolving issues with slow Outbound mail in Office 365 in a Hybrid connection using a SmartHost or Internal Appliance

By: Sean Kearney – MVP

You may find that flow to Office 365 hosted users is slow after initial Hybrid Configuration. But the answer is closer than you think and is only one Cmdlet away with Windows PowerShell.

The actual problem is the mail flow to Office 365 is slow because the connector is defaulting to using DNS for mail sending. In this scenario, the server is either sending directly to an internal appliance (such as a Barracuda) or uses a Smarthost on the internet (Such as on SBS 2011 / SBS 2008) and in many SMB configurations.

Normally you would change the connector properties to point the exist SmartHost and the problem would be resolved.

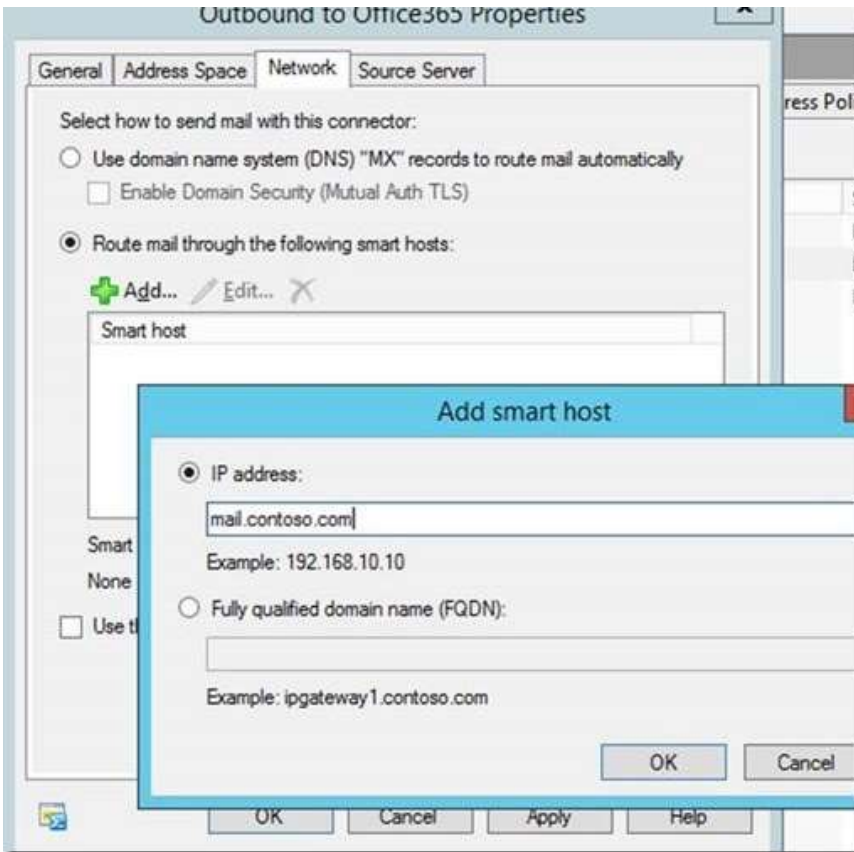
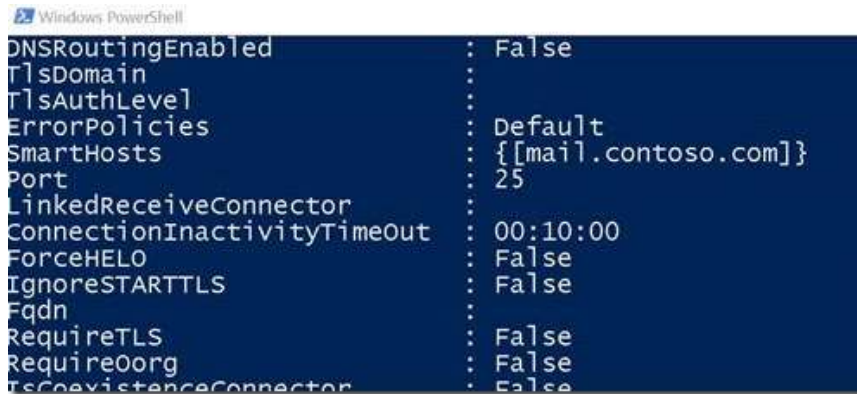


Figure 8 – SmartHost Connection

What you will find in the connector created by DEFAULT with the Hybrid wizard is this does NOT solve the problem. You can see the actual problem when you view the connectors with the Get-SendConnector Cmdlet and view a full list of the properties.

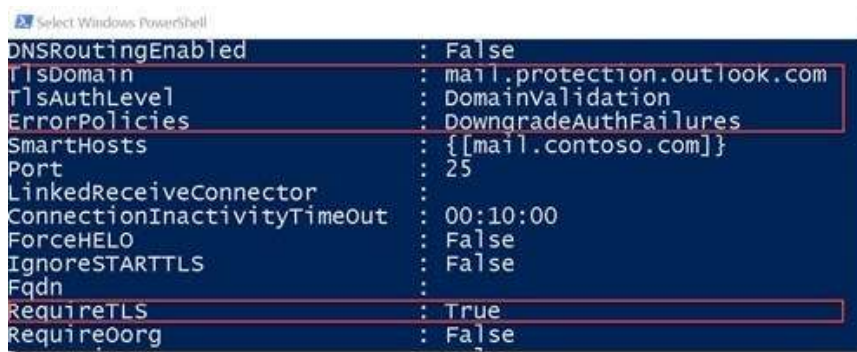
```
Get-SendConnector -identity 'SmartHost Connector' | Format-List *
```



```
Windows PowerShell
DNSRoutingEnabled : False
TlsDomain         :
TlsAuthLevel     :
ErrorPolicies    : Default
SmartHosts       : {[mail.contoso.com]}
Port             : 25
LinkedReceiveConnector :
ConnectionInactivityTimeout : 00:10:00
ForceHELO        : False
IgnoreSTARTTLS   : False
Fqdn             :
RequireTLS       : False
RequireOrg       : False
IsCoexistenceConnector : False
```

Figure 9 – Get-SendConnector output

```
Get-SendConnector -identity 'Outbound to Office365' | Format-List
```



```
Select Windows PowerShell
DNSRoutingEnabled : False
TlsDomain         : mail.protection.outlook.com
TlsAuthLevel     : DomainValidation
ErrorPolicies    : DowngradeAuthFailures
SmartHosts       : {[mail.contoso.com]}
Port             : 25
LinkedReceiveConnector :
ConnectionInactivityTimeout : 00:10:00
ForceHELO        : False
IgnoreSTARTTLS   : False
Fqdn             :
RequireTLS       : True
RequireOrg       : False
```

Figure 10 – Get-SendConnector Output for Office 365

Note the highlighted areas of the Office 365 connector. By default it's insistent on a TLS connection and a few additional properties (which are fine if you're hosting a live Exchnage/DNS configuration)

However in the case of a SmartHost, it may not like this (Bad Smarthost on the internet with lousy Authentication?) or you're running an internal box to handle outbound spam which is not configured to accept TLS internally.

Three options you have. (as Yoda would say)

1. One, get a better SmartHost provider or improve your internal Device security to accept TLS.
2. The Second option is to recreate the Office 365 Connector from scratch with the same settings and point straight to the SmartHost.
3. The third (and probably quite a bit easier) is to adjust the connector through PowerShell and disable the requirement for TLS and remove the additional properties left behind by Office 365's Hybrid wizard (which should be fine for INTERNAL devices)

First make a copy of the Connector should you need to Rollback the process

```
Get-SendConnector -identity 'Outbound to Office365' | Export-Clixml  
Office365Connector.xml
```

Then adjust the particular connector name for Office 365 outbound mailflow on your On Premise Exchange environment

```
Set-SendConnector -identity 'Outbound to Office365' -RequireTLS $False -  
RequireTLS $False -TlsAuthLevel $NULL -TlsDomain $NULL -ErrorPolicies Default
```

No restart of any services are required and the effect should be immediate. Note as always, any mail stuck in the queue under the old configuration is just good for one thing... NDR's.

Chapter 14

Use PowerShell to Work with Data from OMS

By: Ed Wilson – The Scripting Guy

Microsoft Scripting Guy, Ed Wilson, is here. One of the things I have been working on a lot the past month is the new Microsoft Operations Management Suite Blog. For one thing, it is on the new blogging platform we using at Microsoft, so the blog is like a new toy with lots of fun things for a geek to play with. In addition, Microsoft Operations Management Suite (MS OMS) is a tremendously powerful platform that uses Windows PowerShell in the background, so it is also way cool to play with. With these two sets of givens, it was only a matter of time before I begin to merge the two.

The other day I wrote a blog post for the MS OMS blog titled Perform system update assessment with #MSOMS. I have written such articles before for the Hey, Scripting Guy! Blog, but they were always a bit annoying to write because update information has a tendency to be reported in different ways, in different places, depending on the level of the operating system and so on. It can be done, but I am never too sure of the results, for instance, as compared to what the Security Update tool reports in Control Panel. The System Update Assessment tool in MS OMS is reliable, produces cool reports, is quick, and it can export a CSV file.

Parsing MS OMS data with PowerShell

The first thing I do is in MS OMS. I click the Export button at the bottom of the System Update Assessment tool:

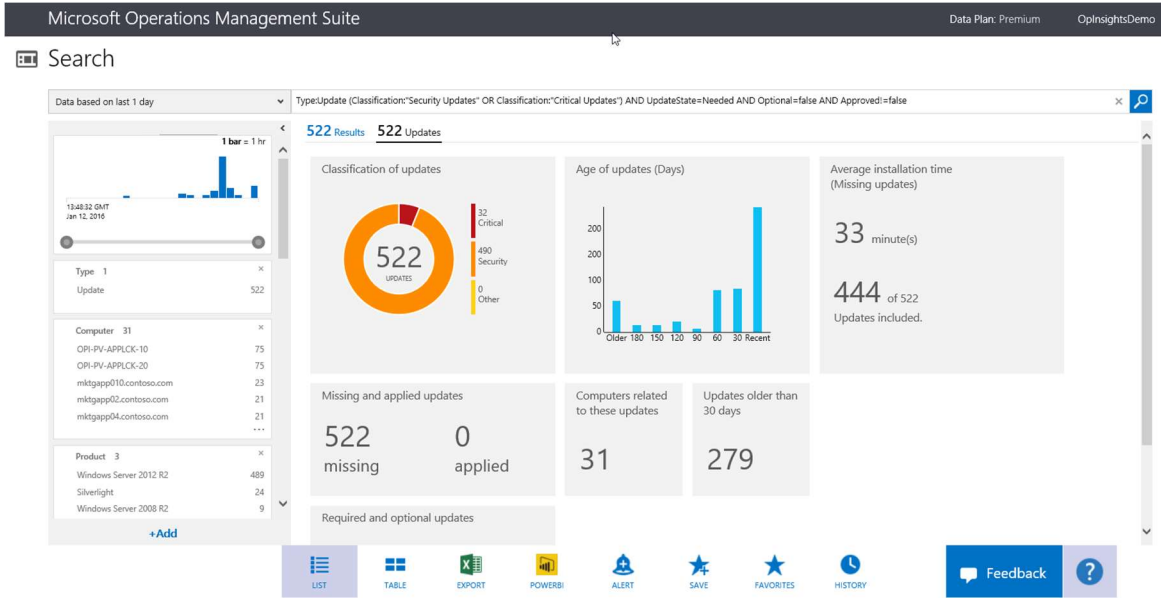


Figure 10 – Microsoft OMS Suite

After I save the data in a location I can find, I open the data in a Microsoft Excel spreadsheet:

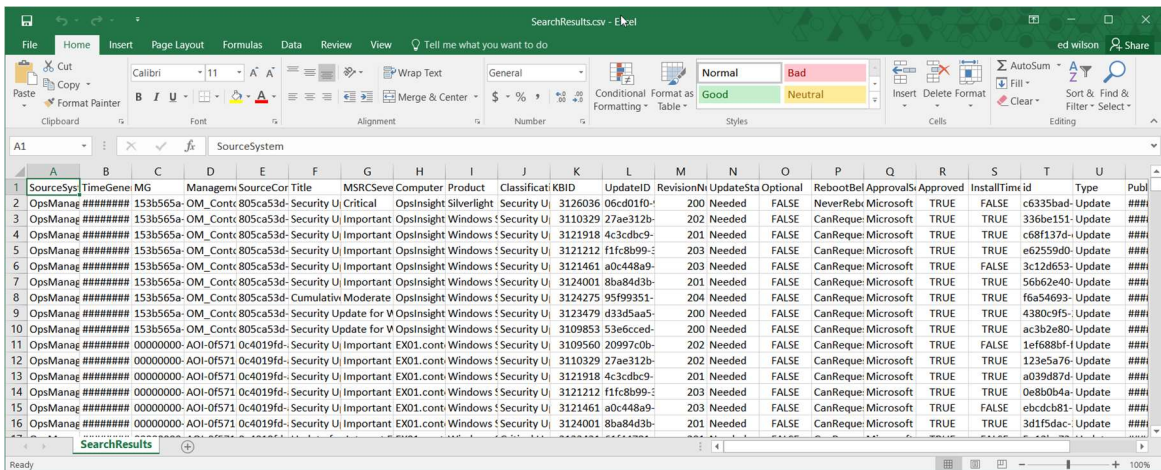


Figure 11 – CSV Output from the Export

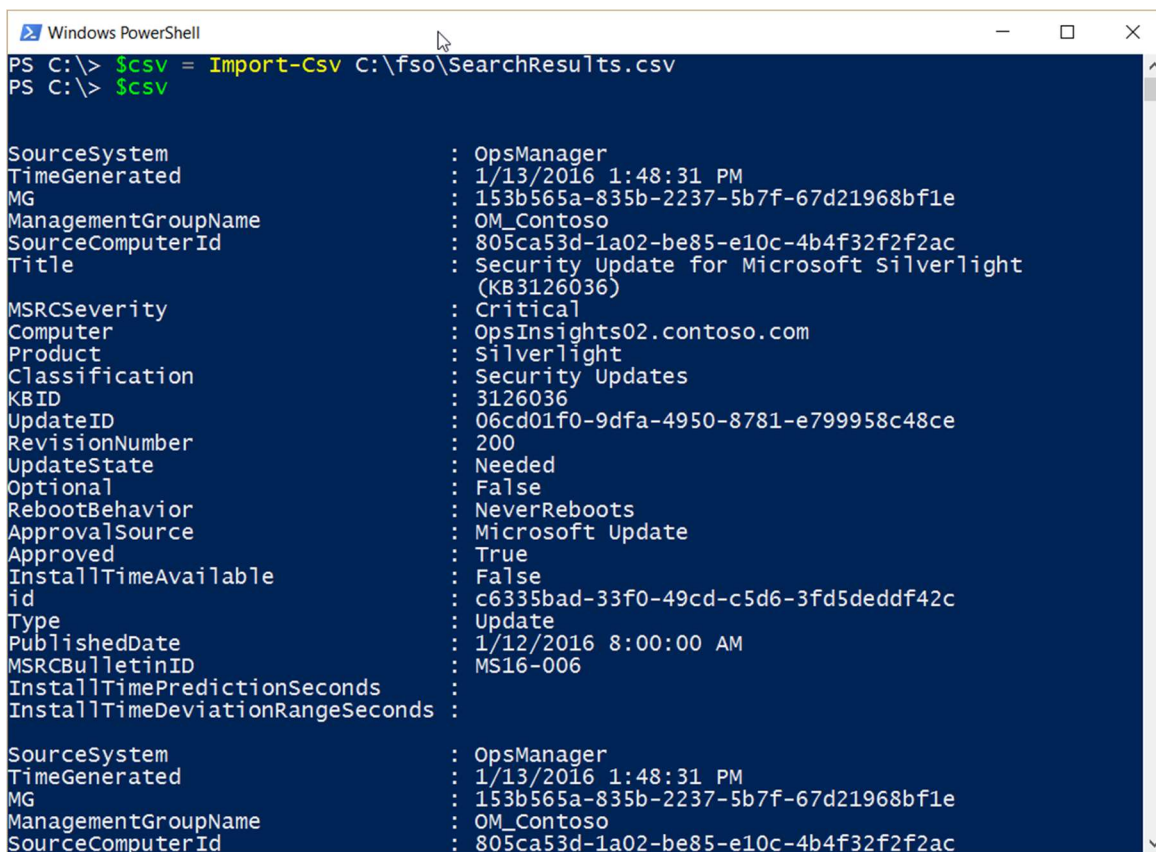
Importing the CSV file using PowerShell

The way cool thing about importing the CSV file with Windows PowerShell is that I obtain objects that represent each row that appears in my Excel sheet. This makes the data easier to read and manipulate.

Importing the CSV file is easy, I use the Import-CSV cmdlet and store the returned objects in a variable that I name \$csv:

```
$csv = Import-Csv C:\fso\SearchResults.csv
```

When I call the \$csv variable, the objects that I stored appear in my Windows PowerShell console...and scroll and scroll and scroll. The commands and a portion of the output are shown here:



```
Windows PowerShell
PS C:\> $csv = Import-Csv C:\fso\SearchResults.csv
PS C:\> $csv

SourceSystem      : OpsManager
TimeGenerated     : 1/13/2016 1:48:31 PM
MG               : 153b565a-835b-2237-5b7f-67d21968bf1e
ManagementGroupName : OM_Contoso
SourceComputerId  : 805ca53d-1a02-be85-e10c-4b4f32f2f2ac
Title            : Security Update for Microsoft Silverlight
                  (KB3126036)
MSRCSeverity      : Critical
Computer         : OpsInsights02.contoso.com
Product          : Silverlight
Classification    : Security Updates
KBID             : 3126036
UpdateID         : 06cd01f0-9dfa-4950-8781-e799958c48ce
RevisionNumber    : 200
UpdateState      : Needed
Optional         : False
RebootBehavior   : NeverReboots
ApprovalSource   : Microsoft Update
Approved         : True
InstallTimeAvailable : False
id              : c6335bad-33f0-49cd-c5d6-3fd5deddf42c
Type             : Update
PublishedDate    : 1/12/2016 8:00:00 AM
MSRCBulletinID  : MS16-006
InstallTimePredictionSeconds :
InstallTimeDeviationRangeSeconds :

SourceSystem      : OpsManager
TimeGenerated     : 1/13/2016 1:48:31 PM
MG               : 153b565a-835b-2237-5b7f-67d21968bf1e
ManagementGroupName : OM_Contoso
SourceComputerId  : 805ca53d-1a02-be85-e10c-4b4f32f2f2ac
```

Figure 12 – Importing the CSV from OMS

Each of the column headings now appear as a property associated with each instance of the object. So how many objects (rows) do I have? The following command shows me that there are 522:

```
PS C:\> $csv.Count
```

522

I may be interested in how many servers need which update. I can do this by grouping the updates by title, and then sorting the output. I use the following command (ft is an alias for Format-Table):

```
$csv | group title -NoElement | sort count -Descending | ft count, name
```

Here is the output:

```

PS C:\> $csv | group title -NoElement | sort count -Descending | ft count, name
Count Name
-----
29 Security Update for Windows Server 2012 R2 (KB3121918)
29 Security Update for Windows Server 2012 R2 (KB3109853)
29 Security Update for Windows Server 2012 R2 (KB3121212)
29 Security Update for Windows Server 2012 R2 (KB3124001)
28 Security Update for Windows Server 2012 R2 (KB3123479)
27 Cumulative Security Update for Internet Explorer 11 for Windows Server 2012...
27 Security Update for Windows Server 2012 R2 (KB3110329)
21 Security Update for Microsoft Silverlight (KB3126036)
15 Update for Windows Server 2012 R2 (KB3112336)
15 Security Update for Windows Server 2012 R2 (KB3108347)
15 Security Update for Windows Server 2012 R2 (KB3109094)
15 Security Update for Windows Server 2012 R2 (KB3109103)
15 Security Update for Windows Server 2012 R2 (KB3108381)
13 Security Update for Windows Server 2012 R2 (KB3081320)
13 Security Update for Windows Server 2012 R2 (KB3092601)
13 Security Update for Windows Server 2012 R2 (KB3102939)
13 Security Update for Windows Server 2012 R2 (KB3101246)
12 Security Update for Microsoft .NET Framework 4.5.1 and 4.5.2 on Windows 8.1...
12 Security Update for Microsoft .NET Framework 4.5.1 and 4.5.2 for Windows 8...
10 Security Update for Windows Server 2012 R2 (KB3121461)
3 Security Update for Windows Server 2012 R2 (KB3011780)
3 Security Update for Microsoft Silverlight (KB3106614)
3 Cumulative Security Update for Internet Explorer 11 for Windows Server 2012...
2 Security Update for Windows Server 2012 R2 (KB3082089)
2 Security Update for Windows Server 2012 R2 (KB3084135)
2 Security Update for Windows Server 2012 R2 (KB3076895)
2 Security Update for Windows Server 2012 R2 (KB3078601)
2 Security Update for Microsoft .NET Framework 4.5.1 and 4.5.2 on Windows 8.1...
2 Security Update for Microsoft .NET Framework 4.5.1 and 4.5.2 on Windows 8.1...
2 Security Update for Windows Server 2012 R2 (KB3087088)
2 Security Update for Windows Server 2012 R2 (KB3072595)

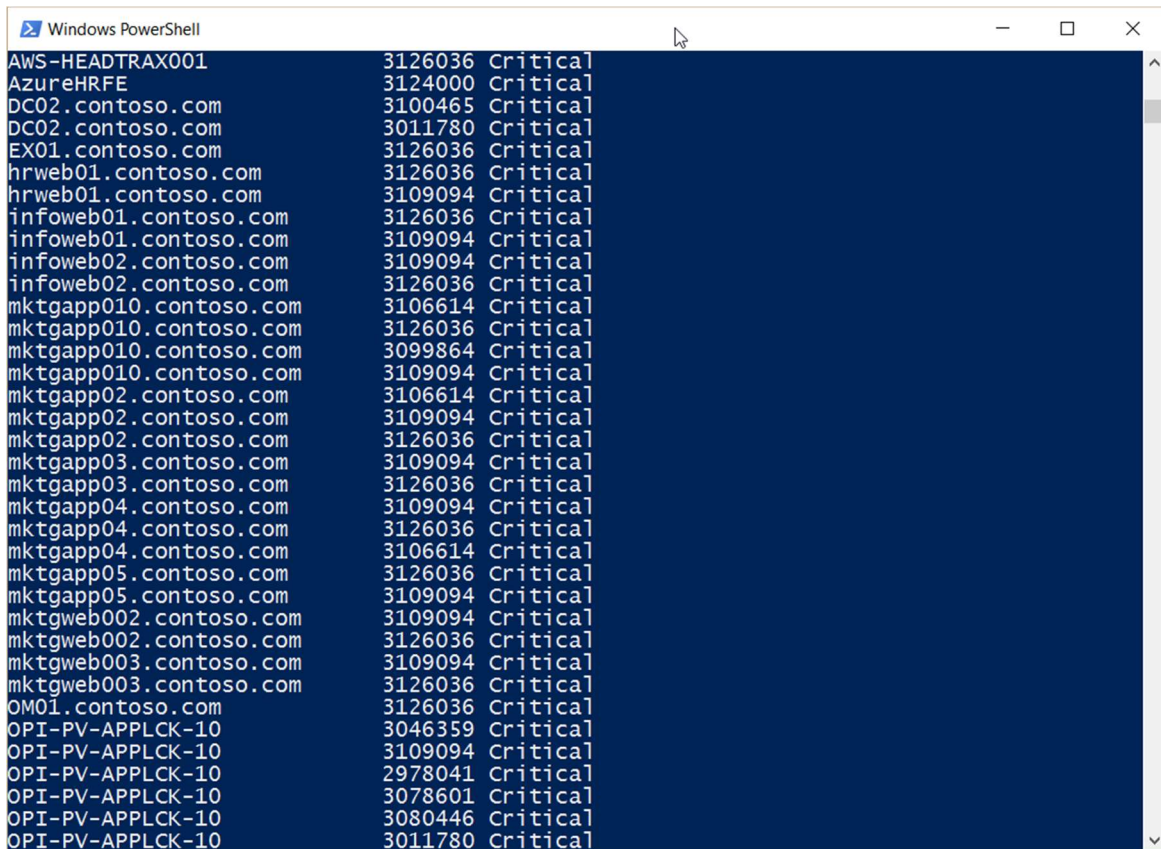
```

Figure 13 – Grouping the output

Perhaps I am interested in the KB ID, the severity, and the computers to which the update applies. I can easily use a command such as the following to obtain the information:

```
$csv | select computer, kbid, msrseverity | sort msrseverity, computer
```

The output will be something like the following:



```

Windows PowerShell
AWS-HEADTRAX001      3126036 Critical
AzureHRFE            3124000 Critical
DC02.contoso.com    3100465 Critical
DC02.contoso.com    3011780 Critical
EX01.contoso.com    3126036 Critical
hrweb01.contoso.com 3126036 Critical
hrweb01.contoso.com 3109094 Critical
infoweb01.contoso.com 3126036 Critical
infoweb01.contoso.com 3109094 Critical
infoweb02.contoso.com 3109094 Critical
infoweb02.contoso.com 3126036 Critical
mktgapp010.contoso.com 3106614 Critical
mktgapp010.contoso.com 3126036 Critical
mktgapp010.contoso.com 3099864 Critical
mktgapp010.contoso.com 3109094 Critical
mktgapp02.contoso.com 3106614 Critical
mktgapp02.contoso.com 3109094 Critical
mktgapp02.contoso.com 3126036 Critical
mktgapp03.contoso.com 3109094 Critical
mktgapp03.contoso.com 3126036 Critical
mktgapp04.contoso.com 3109094 Critical
mktgapp04.contoso.com 3126036 Critical
mktgapp04.contoso.com 3106614 Critical
mktgapp04.contoso.com 3126036 Critical
mktgapp05.contoso.com 3109094 Critical
mktgapp05.contoso.com 3109094 Critical
mktgweb002.contoso.com 3109094 Critical
mktgweb002.contoso.com 3126036 Critical
mktgweb003.contoso.com 3109094 Critical
mktgweb003.contoso.com 3126036 Critical
OM01.contoso.com    3126036 Critical
OPI-PV-APPLCK-10   3046359 Critical
OPI-PV-APPLCK-10   3109094 Critical
OPI-PV-APPLCK-10   2978041 Critical
OPI-PV-APPLCK-10   3078601 Critical
OPI-PV-APPLCK-10   3080446 Critical
OPI-PV-APPLCK-10   3011780 Critical

```

Figure 14 – Filtering to show only Critical updates

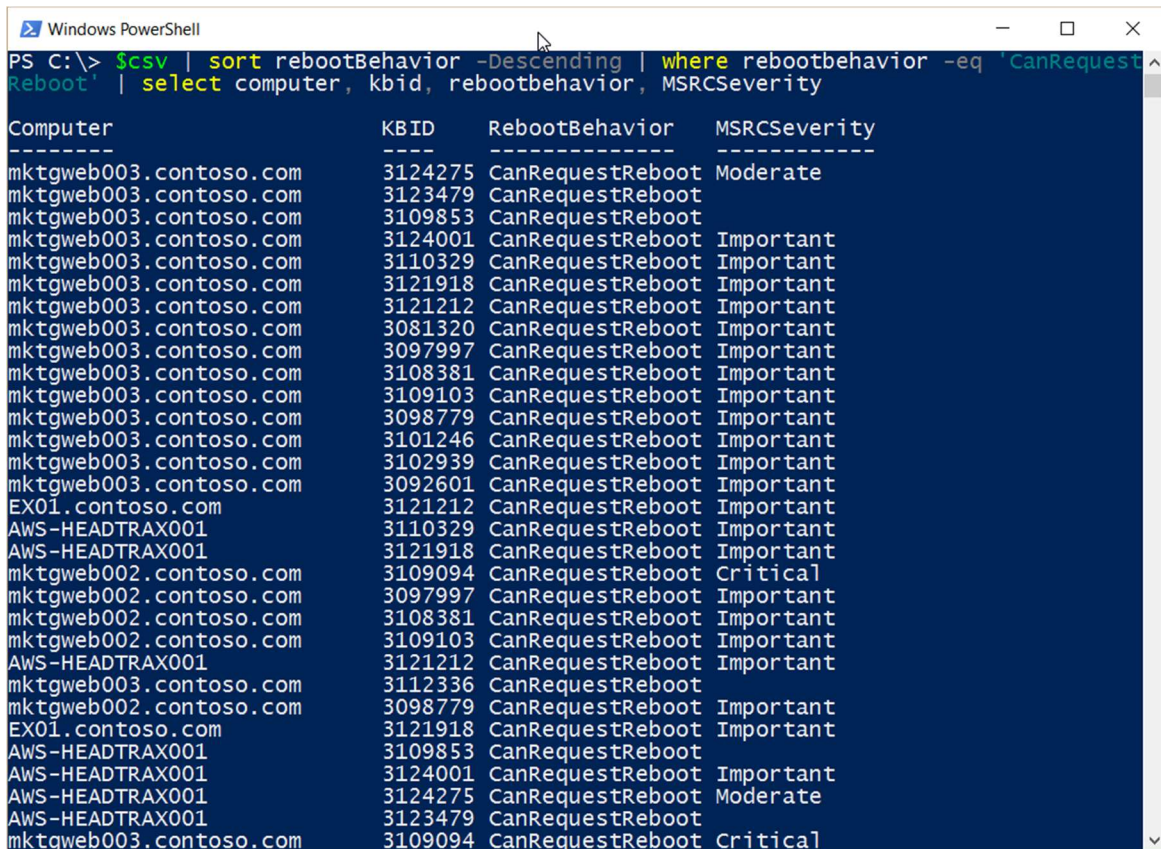
Maybe what I want to know is how many updates require a reboot. I can obtain this information by using the following command:

```

$csv | sort rebootBehavior -Descending | where rebootbehavior -eq
'CanRequestReboot' |
select computer, kbid, rebootbehavior, MSRCseverity

```

The command and its output are shown here:



```

PS C:\> $csv | sort rebootBehavior -Descending | where rebootBehavior -eq 'CanRequestReboot' | select computer, kbid, rebootbehavior, MSRCSeverity

```

Computer	KBID	RebootBehavior	MSRCSeverity
mktgweb003.contoso.com	3124275	CanRequestReboot	Moderate
mktgweb003.contoso.com	3123479	CanRequestReboot	
mktgweb003.contoso.com	3109853	CanRequestReboot	
mktgweb003.contoso.com	3124001	CanRequestReboot	Important
mktgweb003.contoso.com	3110329	CanRequestReboot	Important
mktgweb003.contoso.com	3121918	CanRequestReboot	Important
mktgweb003.contoso.com	3121212	CanRequestReboot	Important
mktgweb003.contoso.com	3081320	CanRequestReboot	Important
mktgweb003.contoso.com	3097997	CanRequestReboot	Important
mktgweb003.contoso.com	3108381	CanRequestReboot	Important
mktgweb003.contoso.com	3109103	CanRequestReboot	Important
mktgweb003.contoso.com	3098779	CanRequestReboot	Important
mktgweb003.contoso.com	3101246	CanRequestReboot	Important
mktgweb003.contoso.com	3102939	CanRequestReboot	Important
mktgweb003.contoso.com	3092601	CanRequestReboot	Important
EX01.contoso.com	3121212	CanRequestReboot	Important
AWS-HEADTRAX001	3110329	CanRequestReboot	Important
AWS-HEADTRAX001	3121918	CanRequestReboot	Important
mktgweb002.contoso.com	3109094	CanRequestReboot	Critical
mktgweb002.contoso.com	3097997	CanRequestReboot	Important
mktgweb002.contoso.com	3108381	CanRequestReboot	Important
mktgweb002.contoso.com	3109103	CanRequestReboot	Important
AWS-HEADTRAX001	3121212	CanRequestReboot	Important
mktgweb003.contoso.com	3112336	CanRequestReboot	
mktgweb002.contoso.com	3098779	CanRequestReboot	Important
EX01.contoso.com	3121918	CanRequestReboot	Important
AWS-HEADTRAX001	3109853	CanRequestReboot	
AWS-HEADTRAX001	3124001	CanRequestReboot	Important
AWS-HEADTRAX001	3124275	CanRequestReboot	Moderate
AWS-HEADTRAX001	3123479	CanRequestReboot	
mktgweb003.contoso.com	3109094	CanRequestReboot	Critical

Figure 15 – Filtering the output to see which updates require reboots

One thing that is kind of fun is to see how long it will take to install of these updates. There is an `InstallTimePredictionSeconds` property that I can use. All I need to do is add all the values together. (Note that some of the updates do not include a prediction time.) I use the following command:

```
PS C:\> $a = 0
```

```
PS C:\> $csv | % {$a+=$_installtimepredictionseconds}
```

```
PS C:\> $a
```

```
12343.6000473499
```

Dividing by 60 gives me the number of minutes:

```
PS C:\> $a / 60
```

```
205.726667455832
```

This is just scratching the surface of how I can use Windows PowerShell to parse the data exported from Microsoft Operations Management Suite. Remember, it is Windows PowerShell and CSV manipulation. I have written several posts about using PowerShell with CSV files.

Chapter 15

Use PowerShell to Explore Office 365 Installation

By: Ed Wilson – The Scripting Guy

It is now time to use some Windows PowerShell cmdlets to explore my Office 365 tenant installation. One of the things I figured out yesterday is that I do not want to always have to be typing my credentials. For one thing, I don't remember them very well. For another thing, when I change the credentials to a nice long pass phrase, I will have an even harder time getting it right. Also, I tend to open and close the Windows PowerShell console on a routine basis—so that makes things that much harder. Therefore, my first order of business is to securely store my Office 365 credentials.

One of the neatest tricks I have ever run across, I got from Lee Holmes. Actually, I have gotten several really cool tricks from Lee, but this is one that I use routinely. I want to store my credentials for Office 365, so I first use the `Get-Credential` cmdlet to obtain my credentials and to turn the user name and password into a credential object. I then store the credential object in a file by using the `Export-CliXML` cmdlet. I can actually do this in a single line. The command is shown here (this is a single-line command that I broke at the pipeline character for readability on this blog):

```
Get-Credential "admin@ScriptingGuy.OnMicrosoft.Com" |  
Export-Clixml c:\fso\ScriptingGuyCredential.xml
```

That is it. I have now saved my Scripting Guy credential to a file. Ah, but what about security? After all, I could have simply saved the password in a text file, and then used the **ConvertTo-Secure** string to help me recreate a credential object. But dude, that would not be very secure. So, I asked Lee Holmes about this. Here is his reply...

The underlying security comes from the **ConvertFrom-SecureString** functionality, which **Export-CliXml** relies on to export secure strings to their serialized form.

The credentials are encrypted with the Windows Data Protection API (DPAPI), which means that you can literally share it with people, put it in untrusted storage, or put it on the billboard of Times Square. The reason you are allowed to encrypt it and decrypt it without typing any passwords is that they key is automatically generated from a combination of your user account and the current machine.

Because of that, even somebody else on your same machine can't decrypt your credentials.

Cool, so I have stored my credentials in a secure fashion. Now what do I need to do to get them back so I can continue to work? Well, that is even simpler. I use the `Import-Clixml` cmdlet, and I store the returned credential object in a variable like I always do when doing when I am remoting or anything else. Here is the command I use:

```
$cred = Import-Clixml C:\fso\ScriptingGuyCredential.xml
```

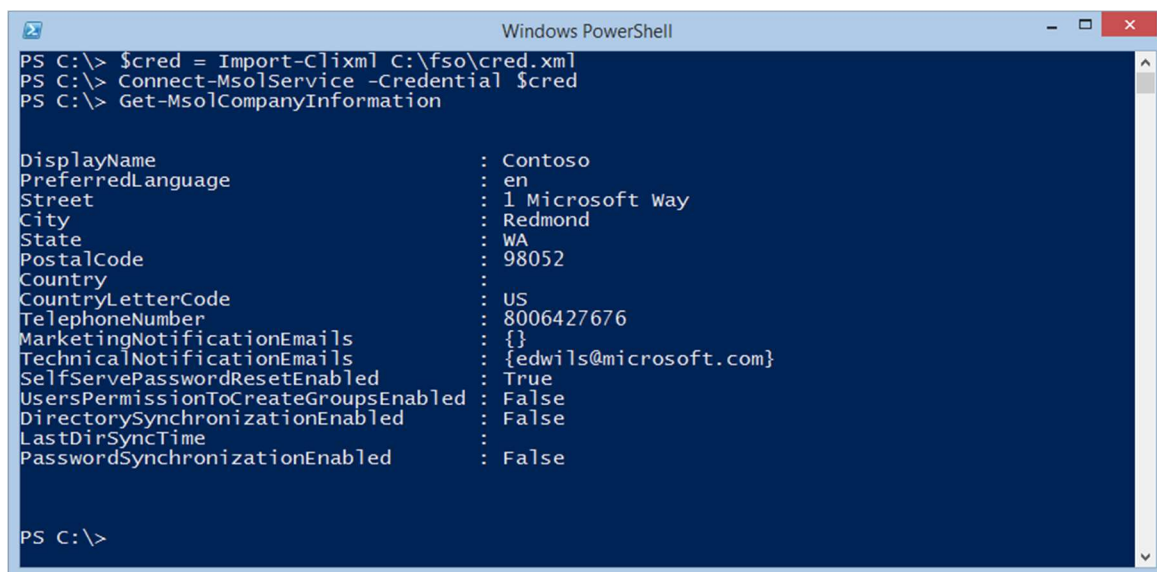
Next, I use the `Connect-MsolService` cmdlet like I did yesterday to connect to my tenant installation:

```
Connect-MsolService -Credential $cred
```

At this point, I can use any of the cmdlets from the `MSOnline` module. For example, one of the things I want to do is to look at the company information. It is easy. I use the `Get-MsolCompanyInformation` cmdlet. This command is shown here:

```
Get-MsolCompanyInformation
```

The command and the output from the command are shown here:



```
Windows PowerShell
PS C:\> $cred = Import-Clixml C:\fso\cred.xml
PS C:\> Connect-MsolService -Credential $cred
PS C:\> Get-MsolCompanyInformation

DisplayName           : Contoso
PreferredLanguage     : en
Street                : 1 Microsoft Way
City                  : Redmond
State                 : WA
PostalCode            : 98052
Country               :
CountryLetterCode    : US
TelephoneNumber       : 8006427676
MarketingNotificationEmails : {}
TechnicalNotificationEmails : {edwils@microsoft.com}
SelfServePasswordResetEnabled : True
UsersPermissionToCreateGroupsEnabled : False
DirectorySynchronizationEnabled : False
LastDirSyncTime      :
PasswordSynchronizationEnabled : False

PS C:\>
```

Figure 16 – Connecting to Office 365 using PowerShell

Next, I need to check to see how many active units I have, how many consumed units I have, and what kind of license pack I have. The command is simple. I use the `Get-MsolAccountSku` cmdlet, and it provides exactly what I need. The command and output are shown here:

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

```
AccountSkuId          ActiveUnits  WarningUnits  ConsumedUnits
```

```
ScriptingGuy:ENTERPRISEPACK          25          0          25
```

Note It is important to remember to use Tab expansion when working with the MSOnline module because it does not export any cmdlet aliases. I checked it by using this command:

```
gcm -Module MSOnline | % { gal -Def $_.name -ea 0}
```

To find the contact for the Office 365 enterprise tenant installation, I use the **Get-MsolContact** cmdlet, as shown here:

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

EmailAddress	DisplayName
bobk@tailspintoys.com	Bob Kelly (TAILSPIN)

That is all there is to using Windows PowerShell to explore the settings and capabilities of my Office 365 installation.

Chapter 16

PowerShell 5 Classes: Constructor Overloading

By: Ed Wilson – The Scripting Guy

Microsoft Scripting Guy, Ed Wilson, is here. Today I want to talk about overloaded constructors. I know, I know, I know...

It sounds like some sort of engineering failure. But that is not the case. In fact, constructor overloading makes your code much more flexible, and even easier to read—when you learn the trick. And today we are going to unmask the secrets of constructor overloading with Windows PowerShell 5.0 classes in Windows 10.

This is way cool stuff, and soon you will be able to entertain friends at night, regal workshop audiences in real-time, and maybe even get more work done sooner and with less effort. After all, that is the real payoff. So, let's get started.

What's an overload anyway?

An overload is more than one way of doing something. So for example, when I call a method and pass one or more parameters to it, that calls for an overloaded method definition. This means in the code, I need to be able to handle one or more ways of doing things. It also means that when I call that method, I have more than one way of calling the method.

A simple example is the `GetProcessesByName` method from the `System.Diagnostics.Process` class. It has two overloads: one in which I pass only the process name and the other where I pass the process name and the name of the computer. This overload definition gives me the ability to return processes locally or remotely. Here are the overload definitions:

```
PS C:\Users\mredw> [System.Diagnostics.Process]::GetProcessesByName
```

OverloadDefinitions

```
static System.Diagnostics.Process[] GetProcessesByName(string processName)
static System.Diagnostics.Process[] GetProcessesByName(string processName, string
machineName)
```

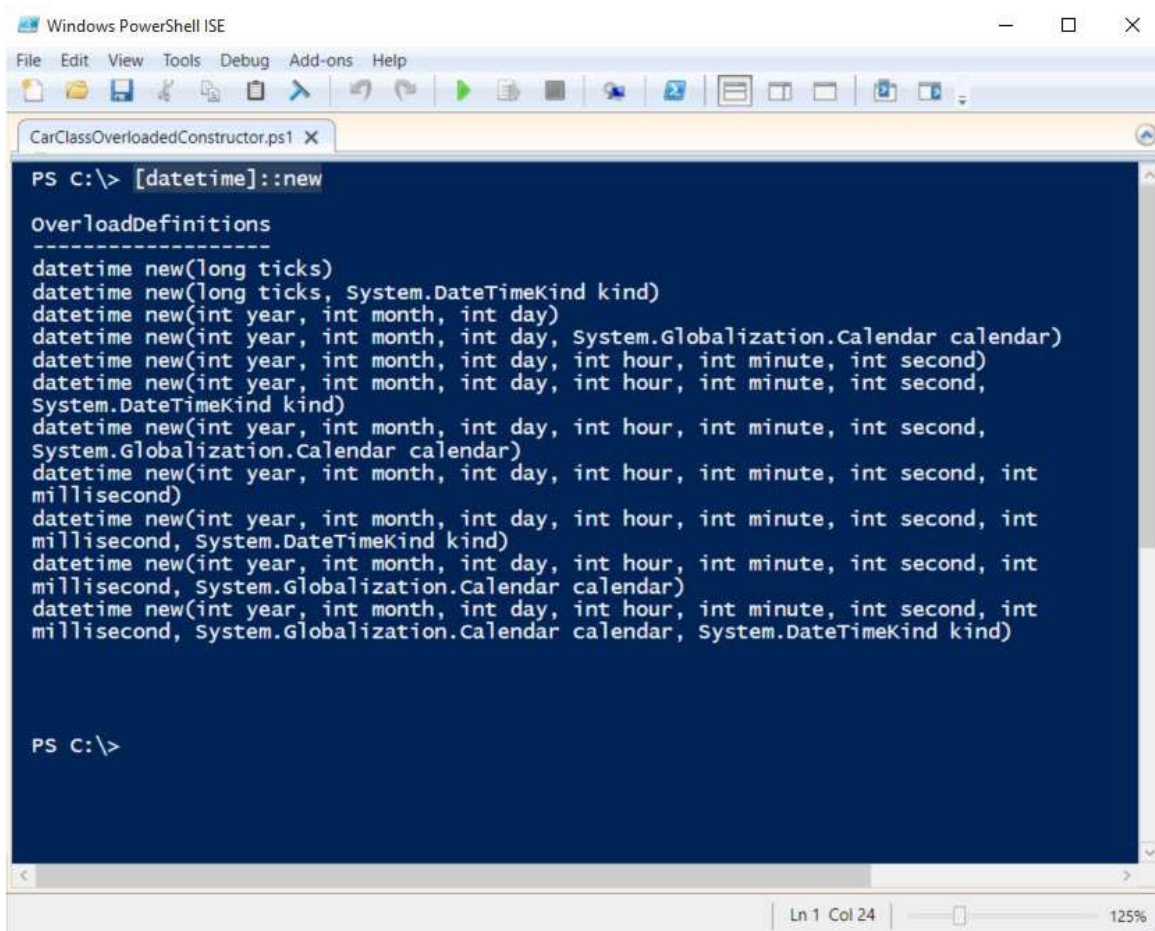
Just as I can overload a method definition, I can also overload the constructor. The constructor is what is used to create a new instance of a class.

Remember that when I have a new instance of a class I have created an object. Also remember that a class itself is simply a template for creating objects. I talk about this in my Introduction to PowerShell 5 Classes blog post and in Introduction to PowerShell 5 classes—The Video.

There are also different ways I can create a new instance of a class. As an example, consider the `System.DateTime` class that represents an instance of date and time. I can create an instance of this class in many different ways. To find these ways, I use the static `New` method, for example:

```
[datetime]::new
```

Here is an image of the overload output:



```
Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
CarClassOverloadedConstructor.ps1 X
PS C:\> [datetime]::new

OverloadDefinitions
-----
datetime new(long ticks)
datetime new(long ticks, System.DateTimeKind kind)
datetime new(int year, int month, int day)
datetime new(int year, int month, int day, System.Globalization.Calendar calendar)
datetime new(int year, int month, int day, int hour, int minute, int second)
datetime new(int year, int month, int day, int hour, int minute, int second,
System.DateTimeKind kind)
datetime new(int year, int month, int day, int hour, int minute, int second,
System.Globalization.Calendar calendar)
datetime new(int year, int month, int day, int hour, int minute, int second, int
millisecond)
datetime new(int year, int month, int day, int hour, int minute, int second, int
millisecond, System.DateTimeKind kind)
datetime new(int year, int month, int day, int hour, int minute, int second, int
millisecond, System.Globalization.Calendar calendar)
datetime new(int year, int month, int day, int hour, int minute, int second, int
millisecond, System.Globalization.Calendar calendar, System.DateTimeKind kind)

PS C:\>
```

Figure 17 – Overload Output

Creating overloaded constructors

To create a constructor (overloaded or otherwise) in a Windows PowerShell 5.0 class, I need to remember that I am really creating a method. The method has the same name as the class name. When I do this, I have a constructor.

For example, I want to add a constructor to my Car class (the simple class I talked about in PowerShell 5: Create Simple Class).

Note You also need to understand creating methods because, after all, a constructor is really just a method. For information about methods, please refer to Adding Methods to a PowerShell 5 Class.

```
Class Car
{
    [String]$vin
    static [int]$numberOfWheels = 4
    [int]$numberOfDoors
    [datetime]$year
    [String]$model
}
```

Now I add a method with the name of Car. Here is the syntax:

```
Name of Class
A pair of parentheses for the input
A script block that does something
```

All cars have a VIN, so it makes sense to have a VIN input when creating an instance of the class. Here is my constructor:

```
Car ([string]$vin)
    {$this.vin = $vin}
```

The complete script now looks like this:

```
Class Car
{
    [String]$vin
    static [int]$numberOfWheels = 4
    [int]$numberOfDoors
    [datetime]$year
    [String]$model
    Car ([string]$vin)
        {$this.vin = $vin}
}
```

When I run it and load the class, I can now create a new instance of the class by using my constructor. The syntax is shown here:

```
PS C:\> [car]::new(1234)
```

```
vin  numberOfDoors  year      model
---  -
1234      0 1/1/0001 12:00:00 AM
```

To overload this constructor, I add additional variations—each using the same name. Windows PowerShell will figure out what I want to do. Here I add overloads to my constructor:

```
class Car
{
    [String]$vin
    static [int]$numberOfWheels = 4
    [int]$numberOfDoors
    [datetime]$year
    [String]$model
    Car ([string]$vin)
    { $this.vin = $vin }
    Car ([string]$vin, [string]$model)
    { $this.vin = $vin
      $this.model = $model }
    Car ([string]$vin, [string]$model, [datetime]$year)
    { $this.vin = $vin
      $this.model = $model
      $this.year = $year }
    Car ([string]$vin, [string]$model, [datetime]$year, [int]$numberOfDoors)
    { $this.vin = $vin
      $this.model = $model
      $this.year = $year
      $this.numberOfDoors = $numberOfDoors }
}
```

The first thing I want to do is to see what overloads I have. This is shown here:

```
PS C:\> [car]::new
```

```
OverloadDefinitions
```

```
-----
Car new(string vin)
Car new(string vin, string model)
Car new(string vin, string model, datetime year)
Car new(string vin, string model, datetime year, int numberOfDoors)
```

This means I can create the car in a variety of methods. Here are a few examples:

```
PS C:\> [car]::new(1234)
```

```
vin  numberOfDoors  year      model
---  -
```

1234 0 1/1/0001 12:00:00 AM

```
PS C:\> [car]::new(1234, "chevy", "1/2/2015", 3)
```

```
vin  numberOfDoors  year      model
---  -
1234  3 1/2/2015 12:00:00 AM chevy
```

That is all there is to using overloaded constructors in PowerShell 5.0.

Chapter 17

Filtering Event Logs with PowerShell

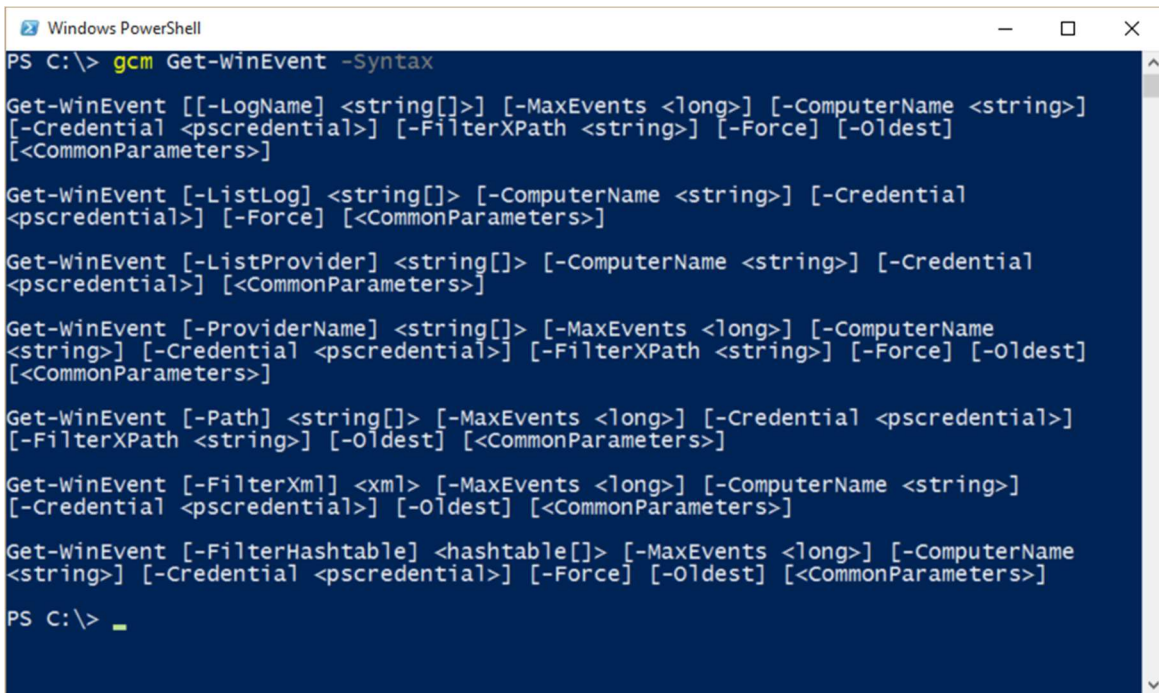
By: Ed Wilson – The Scripting Guy

Hey, Scripting Guy! I try to use the **Get-WinEvent** cmdlet to search event logs, but it is pretty hard to do. Also, I don't see the nice switches that I had with **Get-EventLog**, so I don't see why I should use the other cmdlet and have to pipe everything to **Select-Object** or **Where-Object**.

Microsoft Scripting Guy, Ed Wilson, is here. One of the things that you need to realize is that with Windows PowerShell, one should always filter to the left of the pipeline. This is the prime directive when it comes to working with large amounts of data. Event logs can be huge and contain massive amounts of data. They can consume huge amounts of bandwidth when they are delivered across the network or other places. Like the alligator that the Scripting Wife and I saw while we were hiking the other day, this can be a hidden trap with serious outcomes if the network admin is not paying attention.

Seven parameter sets

The **Get-WinEvent** cmdlet has a number of parameter sets. In fact, it has seven parameter sets. For the sake of the IT pro who needs to filter data from event logs, there are exactly three parameter sets. The parameter sets are shown here:



```

Windows PowerShell
PS C:\> gcm Get-WinEvent -Syntax

Get-WinEvent [[-LogName] <string[]>] [-MaxEvents <long>] [-ComputerName <string>]
[-Credential <pscredential>] [-FilterXPath <string>] [-Force] [-Oldest]
[<CommonParameters>]

Get-WinEvent [-ListLog] <string[]> [-ComputerName <string>] [-Credential
<pscredential>] [-Force] [<CommonParameters>]

Get-WinEvent [-ListProvider] <string[]> [-ComputerName <string>] [-Credential
<pscredential>] [<CommonParameters>]

Get-WinEvent [-ProviderName] <string[]> [-MaxEvents <long>] [-ComputerName
<string>] [-Credential <pscredential>] [-FilterXPath <string>] [-Force] [-Oldest]
[<CommonParameters>]

Get-WinEvent [-Path] <string[]> [-MaxEvents <long>] [-Credential <pscredential>]
[-FilterXPath <string>] [-Oldest] [<CommonParameters>]

Get-WinEvent [-FilterXml] <xml> [-MaxEvents <long>] [-ComputerName <string>]
[-Credential <pscredential>] [-Oldest] [<CommonParameters>]

Get-WinEvent [-FilterHashtable] <hashtable[]> [-MaxEvents <long>] [-ComputerName
<string>] [-Credential <pscredential>] [-Force] [-Oldest] [<CommonParameters>]

PS C:\>

```

Figure 18 – Getting the syntax for Get-WinEvent

Here are the three filter parameters:

```

PS C:\> ((gcm Get-WinEvent | select -expand parametersets).parameters).where({$_.name
-match '^filter'}) | select name -Unique

```

Name

FilterXPath
FilterXml
FilterHashtable

Of the three filter parameters, the easiest for me to use is FilterHashTable. The FilterHashTable parameter accepts...wait for it...you will never guess this one...

...a hash table.

That is right, the FilterHashTable parameter accepts a hash table as the input parameter.

Note If you need a refresher about hash tables, see [Learn the Basics of PowerShell Hash Tables](#).

Here is the most important thing you need to understand when using the FilterHashTable parameter:

Everything goes into the hash table.

The syntax is shown here:

```
Get-WinEvent [-FilterHashtable] <hashtable[]> [-MaxEvents <long>] [-ComputerName <string>] [-Credential <pscredential>] [-Force] [-Oldest] [<CommonParameters>]
```

I said everything—well obviously, not everything. But things used for filtering the events, such as the event log name, the ID, and stuff like that go into the hash table. Here is a table of things you can use when creating a filter hash table:

Key name	Value data type	Accepts wildcard characters?
LogName	<String[]>	Yes
ProviderName	<String[]>	Yes
Path	<String[]>	No
Keywords	<Long[]>	No
ID	<Int32[]>	No
Level	<Int32[]>	No
StartTime	<DateTime>	No
EndTime	<DateTime>	No
UserID	<SID>	No
Data	<String[]>	No
*	<String[]>	No

Figure 19 – Filter Hash Table

The filter hash table takes the following form:

- At sign
- Opening Curly bracket
- Keyname
- Equals
- Value
- Closing Curly bracket

Here is a simple example that returns all the events from the application log:

```
Get-WinEvent -FilterHashtable @{logname='application'}
```

Although PowerShell is often very good at converting input to the required data type (dynamic type system), the filter hash table must have the string values placed in single or double quotation marks. For example, if I don't put my value for the LogName keyword in quotation marks, the following error message appears:

```

Windows PowerShell
PS C:\> Get-WinEvent -FilterHashtable @{{logname=application}} -MaxEvents 1
application : The term 'application' 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:41
+ Get-WinEvent -FilterHashtable @{{logname=application}} -MaxEvents 1
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (application:String) [], CommandNotFo
undException
+ FullyQualifiedErrorId : CommandNotFoundException

PS C:\> Get-WinEvent -FilterHashtable @{{logname='application'}} -MaxEvents 1

ProviderName: Microsoft-Windows-Security-SPP

TimeCreated          Id LevelDisplayName Message
-----
10/19/2015 11:54:14 AM    903 Information    The Software Protection servi...

PS C:\>

```

Figure 20 – Showing syntax errors and LogName requiring quotation marks

Note When testing a filter hash table for the **Get-WinEvent** cmdlet, it is a good idea to limit the amount of data returned to just a few records.

This is where `MaxEvents` is a useful parameter.

To add another key name/value combination to `FilterHashTable`, separate the key name=value pair with a semicolon. This is shown here, where I search the Application log for event ID 413.

```
Get-WinEvent -FilterHashtable @{{logname='application'; id=413}}
```

I can get an idea about the properties and values of an event log record by selecting a single event and piping the output to the `Format-List` cmdlet (`fl` is an alias). I then select all of the properties by using the asterisk (`*`) wildcard character. This is shown here:

```

Windows PowerShell
PS C:\> Get-WinEvent -FilterHashtable @{logname='application'; id=413} -MaxEvents 1
Message
      : SettingSyncHost (392) Unable to create a new logfile because
      : the database cannot write to the log drive. The drive may be
      : read-only, out of disk space, misconfigured, or corrupted.
      : Error -1032.
Id
     : 413
Version
     :
Qualifiers
     : 0
Level
     : 2
Task
     : 3
Opcode
     :
Keywords
     : 36028797018963968
RecordId
     : 9890
ProviderName
     : ESENT
ProviderId
     :
LogName
     : Application
ProcessId
     :
ThreadId
     :
MachineName
     : edLT
UserId
     :
TimeCreated
     : 10/18/2015 8:32:34 AM
ActivityId
     :
RelatedActivityId
     :
ContainerLog
     : application
MatchedQueryIds
     : {}
Bookmark
     : System.Diagnostics.Eventing.Reader.EventBookmark
LevelDisplayName
     : Error
OpcodeDisplayName
     : Info
TaskDisplayName
     : Logging/Recovery
KeywordsDisplayNames
     : {Classic}
Properties
     : {System.Diagnostics.Eventing.Reader.EventProperty,
     : System.Diagnostics.Eventing.Reader.EventProperty,
     : System.Diagnostics.Eventing.Reader.EventProperty,
     : System.Diagnostics.Eventing.Reader.EventProperty}

PS C:\>

```

Figure 21 – Filtering for application log eventid =413

By looking at the level parameter, and knowing that my entry is an error record, I can surmise that I can filter specifically on events that have the ID of 413 and are error records. Again, I use a semicolon to separate my key name=value pair. Here is my revised query:

```
Get-WinEvent -FilterHashtable @{logname='application'; id=413; level=2}
```

The output is shown here:

```
PS C:\> Get-WinEvent -FilterHashtable @{logname='application'; id=413; level=2} -
MaxEvents 1
```

```

ProviderName: ESENT
TimeCreated      Id LevelDisplayName Message
-----
10/18/2015 8:32:34 AM    413 Error      SettingSyncHost (392) Unable ...
PS C:\>

```

That is how you can use Windows PowerShell to read the event logs.

Chapter 18

Use the PowerShell 5 Convert-String Cmdlet

By: Ed Wilson – The Scripting Guy

I am continuously amazed with the things I am finding. In the past, I have always advocated that serious network administrators learn at least the basics of Regular Expressions. They are a powerful way of manipulating strings, and they are everywhere in Windows PowerShell (at least as an option). But in Windows PowerShell 5.0, that becomes a little bit less of a requirement.

Convert-String

One of the stealth cmdlets making an appearance in Windows PowerShell 5.0 is the **Convert-String** cmdlet. It is possible you have heard about **ConvertFrom-String**, which is also a new cmdlet—but more than likely, **Convert-String** has remained under the covers. In fact, when I first was reading through the documentation for Windows PowerShell 5.0, I saw **ConvertFrom-String**, and then in the very next sentence, I saw **Convert-String**. I thought I had discovered a documentation bug.

Not so. **Convert-String** is way cool, extremely powerful, and more than a little finicky. This is a cmdlet that demands to be played with and explored to see what it can do for you. You will want to spend half-a-day or more experimenting before you can get really comfortable with it.

But the payoff will permit you to do amazing things from the command line. In mere minutes, you can accomplish what would have taken days of writing custom string manipulation code. It is that good.

The old flipty dipty

A very common task among IT operations is manipulating user's names. Typically, this takes the form of taking first name/last name and switching them around to last name/first initial. In the past with Windows PowerShell, this was not a major pain (if one had written such code), but it was a bit complex. I mean, I had to split the name at something like a comma, and create an array of

name elements, then move the array of name elements around, and then use a string method to select the first initial of the second element in the array.

Like I said, not horribly difficult if you had done that a lot—but still pretty hard to write from scratch the first time around.

With **Convert-String**, I don't even need to write a script to do this. I can do it from the command line. Here is an example:

```
"Mu Han", "Jim Hance", "David Ahs", "Kim Akers" | Convert-String -Example "Ed Wilson=Wilson, E."
```

The way **Convert-String** works is that I can pipeline strings. Then I specify an example of what I want my output to look like. If input strings match up properly with my pattern, the output I specify will be created. Here is what my previous code looks like when I run it on Windows PowerShell 5.0 on my laptop running Windows 10:

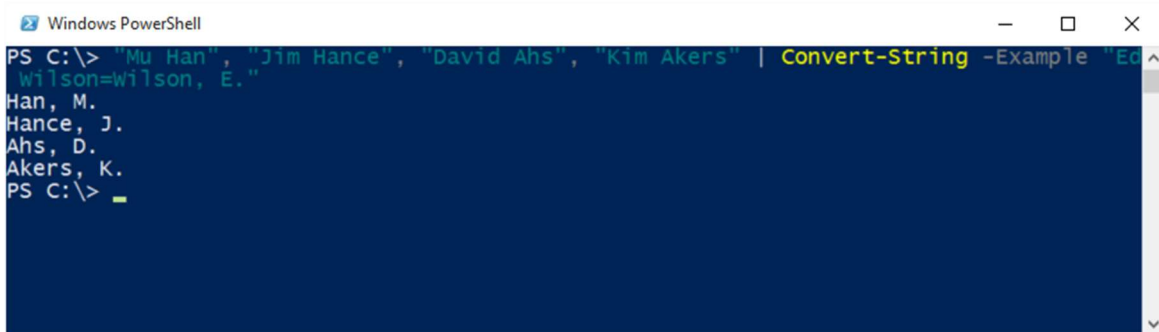


Figure 22 – Viewing the output from Convert String

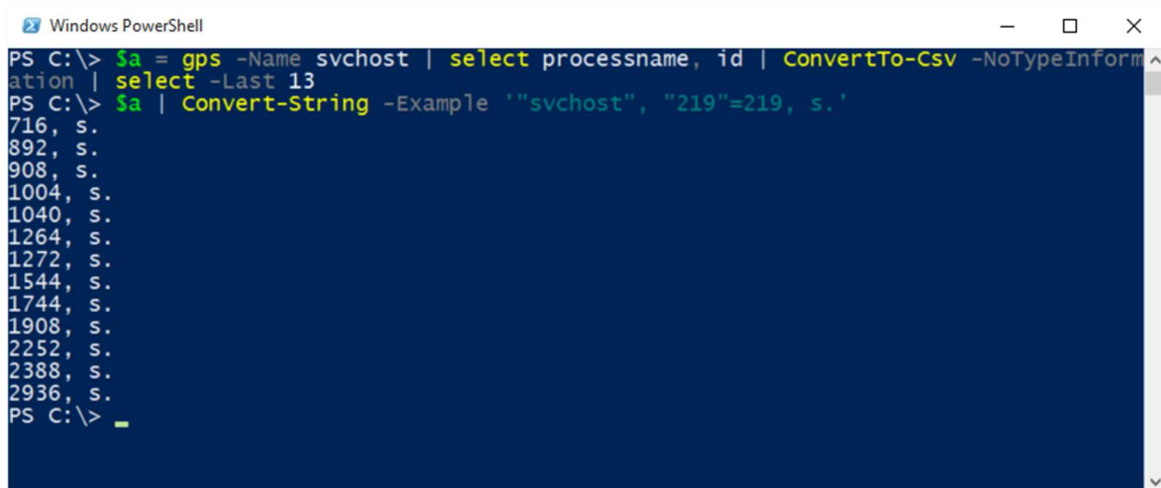
Expanding the string idea

I know that I don't have to manually type in strings to get the Convert-String cmdlet to work. I mean, that would be like so last century. But can I dynamically create my input on the fly? Well, the following example proves this. I do not really have a specific use case scenario for my example, but it will give you an idea of some of the things you might want to play with.

I collect a collection of processes, and I look only for Svchost processes. I then choose the last 13 entries in my CSV that I create on the fly, and store them back into the \$a variable. The \$a variable now contains SVCHOST and the PID. Then I grab the PID and the SVCHOST name so that I can manipulate my output to form a specific pattern. Here is the code:

```
$a = gps -Name svchost | select processname, id | ConvertTo-Csv -  
NoTypeInformation | select -Last 13  
$a | Convert-String -Example '"svchost", "219 "=219, s.'
```

When I run the code, I obtain the output shown here:



```
Windows PowerShell  
PS C:\> $a = gps -Name svchost | select processname, id | ConvertTo-Csv -NoTypeInformation | select -Last 13  
PS C:\> $a | Convert-String -Example '"svchost", "219 "=219, s.'
```

716, s.
892, s.
908, s.
1004, s.
1040, s.
1264, s.
1272, s.
1544, s.
1744, s.
1908, s.
2252, s.
2388, s.
2936, s.
PS C:\> _

Figure 23 – expanding options with *Convert-String*

That is all there is to using the **Convert-String** cmdlet.

Chapter 19

Use PowerShell to Detect if a Location is a Directory or a Symlink

By: Thomas Rayner – MVP

In PowerShell, symbolic links (symlinks) appear pretty transparently when you're simply navigating the file system. If you're doing other work, though, like changing ACLs, bumping into symlinks can be a pain. Here's how to tell if a directory in question is a symlink or not.

```
PS C:\Users\ThmsRynr> ((get-item c:\symlink).Attributes.ToString())  
Directory, ReparsePoint
```

```
PS C:\Users\ThmsRynr> ((get-item c:\normaldir).Attributes.ToString())  
Directory
```

Here, we're just running a **Get-Item** command on two locations, getting the `Attributes` property and converting to a string. The first item is a symlink and includes "ReparsePoint" in its attributes. The second item is a normal directory and does not include "ReparsePoint".

So that means we can do something as easy as this.

```
PS C:\Users\ThmsRynr> ((get-item c:\symlink).Attributes.ToString() -match  
"ReparsePoint")  
True
```

```
PS C:\Users\ThmsRynr> ((get-item c:\normaldir).Attributes.ToString() -match  
"ReparsePoint")  
False
```

Easy. If the above values have "ReparsePoint" in them, we know they are a symlink and not just a regular directory. In my case, my script to apply ACLs to a group of directories avoided symlinks with ease.

Chapter 20

Bypassing PowerShell Execution Policy

By: Thomas Rayner – MVP

Let me be absolutely clear about this chapter. I do not in any way encourage or support people who wish to use the below information to circumvent the controls put in place by companies and administrators. This post is strictly for academic purposes and for the sake of sharing information.

PowerShell Execution Policies control whether or not a system may run a PowerShell script based on whether the script is signed or not. See the [about_Execution_Policies](#) Technet page for more information if you are unfamiliar with execution policies or how to apply them. Execution policies do not, however, limit a user or service from running commands in a PowerShell shell (PowerShell.exe).

So, what if you have an unsigned script you want to run but your execution policy is preventing it? Well, there's a way to bypass the execution policy. And it's run from a PowerShell shell.

Administrative users can easily bypass the execution policy with this command.

```
PowerShell.exe -noprofile -executionpolicy bypass -file "\\path\to\file.ps1"
```

But what about limited users? Well there's something for them, too.

```
Powershell.exe -NoProfile -Command {.[[scriptblock]::create((Get-Content  
"\\path\to\script.ps1" | out-string))}}
```

That's right, just one line. No registry hacking, no weird developer program strangeness, just a command that allows a user or service to subvert the execution policy of the machine.

Let's break down the command. We're launching PowerShell.exe, not exactly a puzzler. We want it with no profile and we're telling it to run a command. The trick is that the command we're running is effectively going to be the script that our execution policy would otherwise block.

The dot is basically an alias for "execute" and in this case, we're telling it to execute what's in the proceeding round brackets. The round brackets contain instructions to create a new ScriptBlock out of the contents of the .ps1 file that the execution policy would otherwise prevent from running.

I think it's clear that this is not really something that Microsoft intends for you to do. Use (or not) wisely at your own discretion.

Chapter 20

Starting up and Shutting down a list of VM's in a specific order

By: Sean Kearney – MVP

A friend asked me if this was hard to do with PowerShell in Hyper-V. He tapped me on the back and literally asked....

“When I have to shutdown a pile of Virtual servers for maintenance, I have a particular order they need to shutdown in, then startup needs to be the reverse. Is there a nice easy way to do this in PowerShell?”

So, I thought about it. We could do this with one script actually (Which is the neat bit)

First let's keep this simple. I want a simple array of Servers. We can do this in the following manner.

```
[array]$VMList=('Server1', 'Server2', 'Server3', 'Server4')
```

With a list like this I can just step through them with PowerShell and Shutdown them in the order provided with a simple ForEach Loop

```
    Foreach ($VM in $VMList)
    {
Stop-VM -vm $VM
    }
```

Now the second challenge REVERSING the order. I was going to try and come up something really neat for this by getting count of elements in the array when I ran across this PowerTip from www.scriptingguy.com on Reverse an Array with PowerShell.

So to get the REVERSE order I just pass my Array into the Reverse method like this.

```
[array]::reverse($VMList)
```

Now I can run a SIMILIAR loop to Start the Virtual Machines.

```
Foreach ($VM in $VMList)
{
Start-VM -vm $VM
}
```

Cool. Now I just need some way to tell the script whether I starting or stopping. For this I can just drop in a Switch as a parameter and tell it to default to Stopping until specified.

```
param(
[switch]$Start=$True
)
```

At this point we can't get fancy and maybe pop the \$array of Servers as a parameter. With a little IF Then Else to trap for the Switch our script looks a little like this:

```
param(
[switch]$Start=$True,
[array]$VMlist=('Server1','Server2','Server3','Server4','Server5')
)
if ($Stop)
{
Foreach ($VM in $VMlist)
{
Stop-VM -VM $VM
}
}
Else
{
[array]::Reverse($vmlist)
Foreach ($VM in $VMlist)
{
Start-VM -VM $VM
}
}
}
```

Pretty neat what you can put together with just a little bit of Powershell eh?

Chapter 21

Join us at MVPDays and meet great MVP's like this in person

If you liked their book, you will love to hear them in person.

Live Presentations

Dave frequently speaks at Microsoft conferences around North America, such as TechEd, VeeamOn, TechDays, and MVPDays Community Roadshow.

Cristal runs the MVPDays Community Roadshow.

You can find additional information on the following blog:

www.checkyourlogs.net

www.mvpdays.com

Video Training

For video-based training, see the following site:

www.mvpdays.com

Live Instructor-led Classes

Dave has been a Microsoft Certified Trainer (MCT) for more than 15 years and presents scheduled instructor-led classes in the US and Canada. For current dates and locations, see the following sites:

- www.truesec.com
- www.checkyourlogs.net

Consulting Services

Dave and Cristal have worked with some of the largest companies in the world and have a wealth of experience and expertise. Customer engagements are typically between two weeks and six months. For more information, see the following site:

www.triconelite.com and www.rsvccorp.com

Twitter

Dave, Cristal, Émile, Thomas, Allan, Sean, and Ed on Twitter tweet on the following aliases:

- Dave Kawula: @DaveKawula
- Cristal Kawula: @SuperCristal1
- Émile Cabot: @Ecabot
- Thomas Rayner: @MrThomasRayner
- Allan Rafuse: @AllanRafuse
- Sean Kearney: @EnergizedTech
- Ed Wilson: @ScriptingGuy