PowerShell - Intro

CHRIS THOMAS

DESKTOP ENGINEER - INGHAM ISD

CTHOMAS@INGHAMISD.ORG

@AUTOMATEMYSTUFF

Ingham Intermediate School District

A Regional Educational Service Agency

Session Description

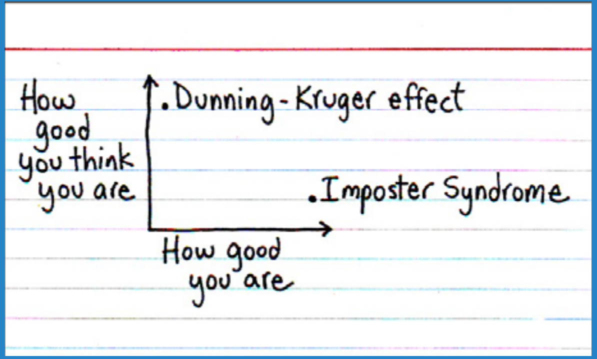
A follow-up intermediate session on Windows PowerShell that will dig in a little deeper than just the cmdlets you have at your disposal. You've overcome your fears of the shell and you took the dive ... now let's talk about stepping away from one-liners and working with scripting logic, importing data, exporting data, creating and manipulating custom objects, custom functions, personal profiles and so much more.

Hopefully PowerShell can help you and your organization figure out where your data is.

See what I did there?

I incorporated the title of this conference into my slideshow.

I'm clever like that…



Who Am I?

- 20 Years In K12 Technology
  - Intern, Tech, Coordinator, Engineer
- Lifelong Learner
  - /r/sysadmin, /r/k12sysadmin, RSS feeds
- Relentlessly Inquisitive
  - Let's Ask The WinAdmins Slack
- Problem Solver
  - Professional Googler
- Voracious Reader
  - docs.microsoft.com
- Community Minded
  - MAEDS, MISCUG, WMISMUG
- #ImpostorSyndromeBeDamned

Past Presentations

2013 MAEDS Fall Conference
Attended 'Marketing Yourself' by Kris Young and Kevin Galbraith
"Our Name, Reputation and Skill Sets Need to be KNOWN"

2014 MAEDS Spring PD Day
First time presenting and it's a 'ConfigMgr panel' for a half day session.

2014 MAEDS Fall Conference
Presented 'PADT and SCCM'

2015 MAEDS Spring PD Day
Presented 'ConfigMgr panel' for a half day session.

2015 MAEDS Fall Conference
Presented 'Automate ALL THE THINGS with PowerShell App Deployment Toolkit'

2017 MAEDS Fall Conference
Presented 'PowerShell – Intro session for those that have been too afraid to take the plunge'

2018 MAEDS Fall Conference
Presented 'PowerShell - Intermediate Session for Those That Overcame Their Fears and Took the Plunge'

2019 MACUL Conference
Presented 'Office 365 Administration'

Expectations

- PowerShell – Intro Session For Those That Have Been Too Afraid To Take The Plunge
  - MAEDS 2017 Fall Conference - http://bit.ly/2NRCuGm

Expectations

- PowerShell – Intro Session For Those That Have Been Too Afraid To Take The Plunge
  - MAEDS 2017 Fall Conference - http://bit.ly/2NRCuGm
- Working knowledge of PowerShell cmdlets

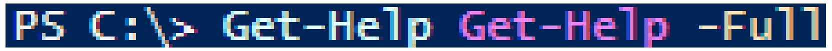
Expectations

- PowerShell – Intro Session For Those That Have Been Too Afraid To Take The Plunge
  - MAEDS 2017 Fall Conference - http://bit.ly/2NRCuGm
- Working knowledge of PowerShell cmdlets
- Basic understanding of objects

Objects
The Bicycle Example

PROPERTY SOMETHING ABOUT THE OBJECT	METHOD SOMETHING YOU CAN DO WITH OR TO THE OBJECT
Wheel Size	Pedal
Tire Pressure	Brake
Height	Change Gear
Gears	Repair
Color	Re-inflate Tire
Speed	
Price	

Expectations

- PowerShell – Intro Session For Those That Have Been Too Afraid To Take The Plunge
  - MAEDS 2017 Fall Conference - http://bit.ly/2NRCuGm
- Working knowledge of PowerShell cmdlets
- Basic understanding of objects
- Ability to read and understand 

```
PS C:\> Get-Help Get-Help -Full
```

Expectations

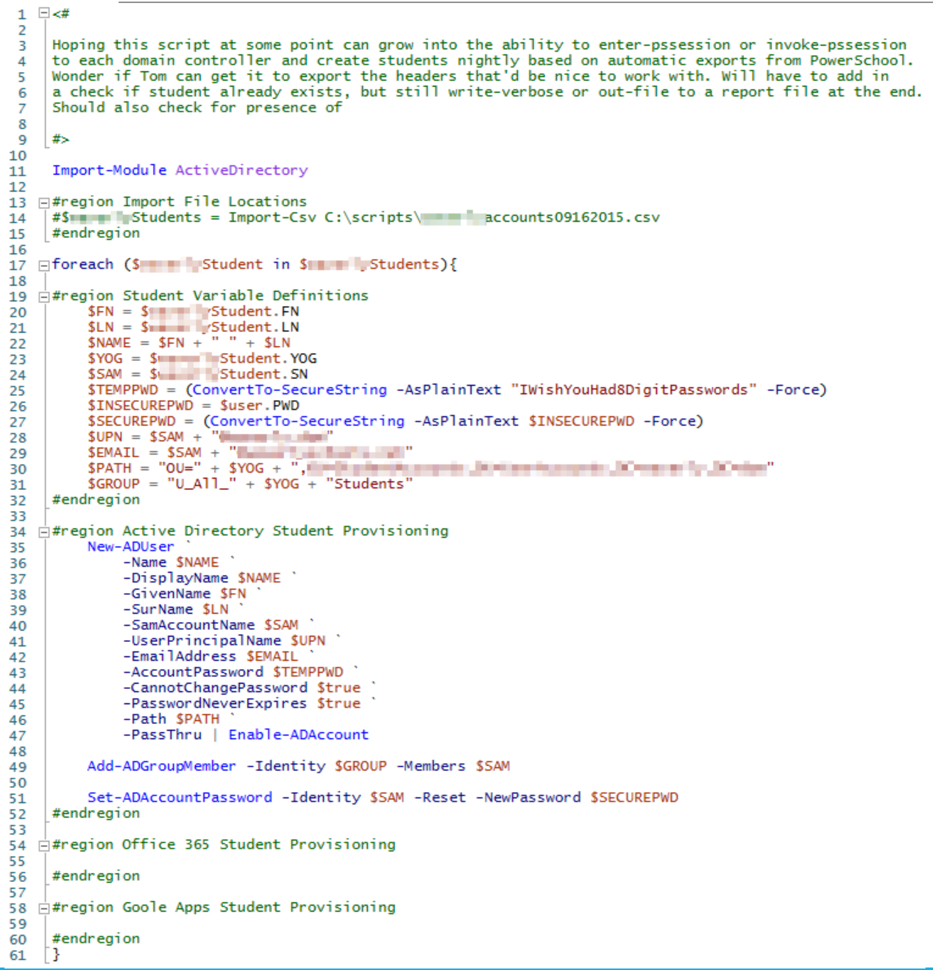
- PowerShell – Intro Session For Those That Have Been Too Afraid To Take The Plunge
  - MAEDS 2017 Fall Conference - http://bit.ly/2NRCuGm
- Working knowledge of PowerShell cmdlets
- Basic understanding of objects
- Ability to read and understand

```
PS C:\> Get-Help Get-Help -Full
```

- Understanding that the scripting process is iterative
  - StudentProvisioning.ps1 (2015)

    vs.
  - <DISTRICT>_STU_INSERT.ps1 (2018)

StudentProvisioning.ps1 (2015)

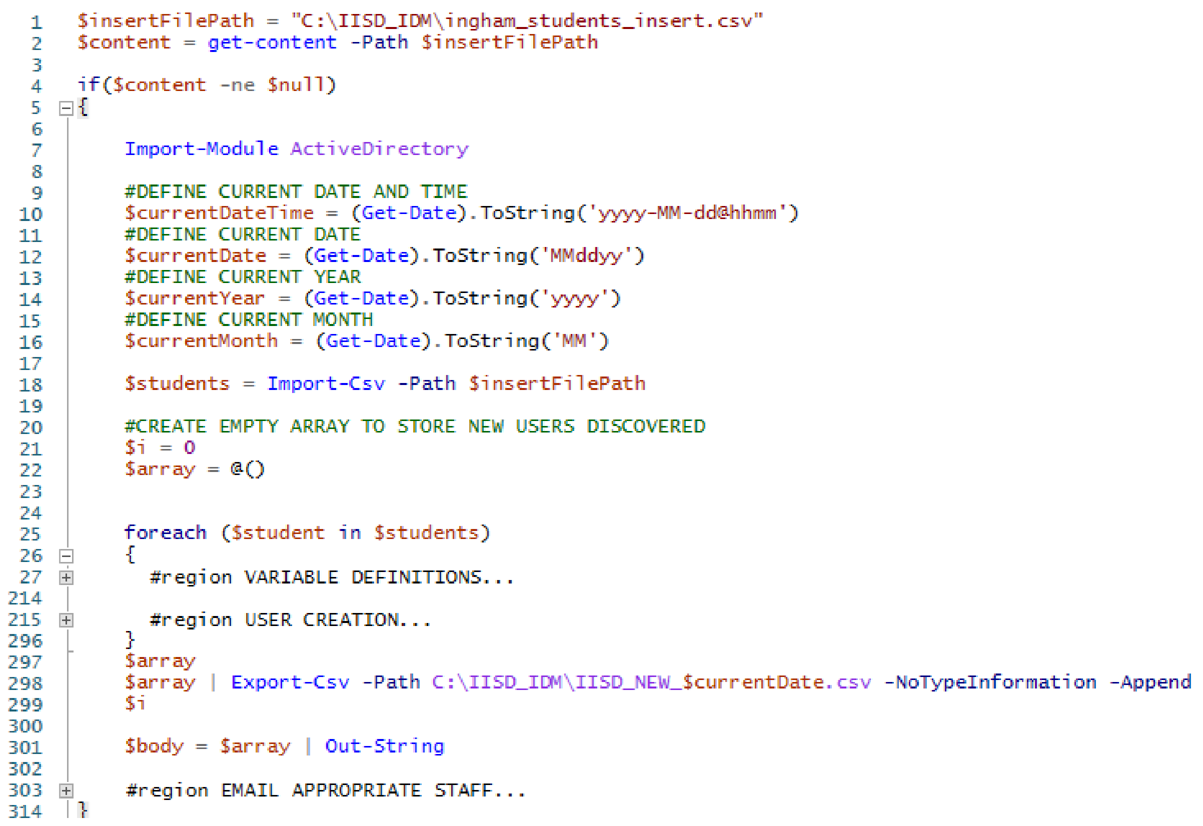


```powershell
<#

Hoping this script at some point can grow into the ability to enter-pssession or invoke-pssession
to each domain controller and create students nightly based on automatic exports from PowerSchool.
Wonder if Tom can get it to export the headers that'd be nice to work with. Will have to add in
a check if student already exists, but still write-verbose or out-file to a report file at the end.
Should also check for presence of

#>

Import-Module ActiveDirectory

#region Import File Locations
#$[redacted]Students = Import-Csv C:\scripts\[redacted]accounts09162015.csv
#endregion

foreach ($[redacted]Student in $[redacted]Students){

#region Student Variable Definitions
    $FN = $[redacted]Student.FN
    $LN = $[redacted]Student.LN
    $NAME = $FN + " " + $LN
    $YOG = $[redacted]Student.YOG
    $SAM = $[redacted]Student.SN
    $TEMPPWD = (ConvertTo-SecureString -AsPlainText "IWishYouHad8DigitPasswords" -Force)
    $INSECUREPWD = $user.PWD
    $SECUREPWD = (ConvertTo-SecureString -AsPlainText $INSECUREPWD -Force)
    $UPN = $SAM + "[redacted]"
    $EMAIL = $SAM + "[redacted]"
    $PATH = "OU=" + $YOG + ",[redacted]"
    $GROUP = "U_All_" + $YOG + "Students"
#endregion

#region Active Directory Student Provisioning
    New-ADUser `
        -Name $NAME `
        -DisplayName $NAME `
        -GivenName $FN `
        -SurName $LN `
        -SamAccountName $SAM `
        -UserPrincipalName $UPN `
        -EmailAddress $EMAIL `
        -AccountPassword $TEMPPWD `
        -CannotChangePassword $true `
        -PasswordNeverExpires $true `
        -Path $PATH `
        -PassThru | Enable-ADAccount

    Add-ADGroupMember -Identity $GROUP -Members $SAM

    Set-ADAccountPassword -Identity $SAM -Reset -NewPassword $SECUREPWD
#endregion

#region Office 365 Student Provisioning

#endregion

#region Goole Apps Student Provisioning

#endregion
}
```

- Plans for the future in my comment block
  - Nightly imports
  - Consistent headers
  - Validation Checks
  - Creation Reports
- Laying out regions to add code to later when there is time

NOTICE: I didn't even finish my thought in the comment block. I was probably distracted by something shiny

<DISTRICT>_STU_INSERT.ps1 (2018)

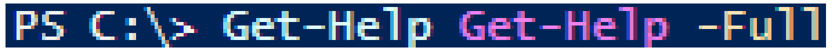


```
$insertFilePath = "C:\IISD_IDM\ingham_students_insert.csv"
$content = get-content -Path $insertFilePath

if($content -ne $null)
{

    Import-Module ActiveDirectory

    #DEFINE CURRENT DATE AND TIME
    $currentDateTime = (Get-Date).ToString('yyyy-MM-dd@hhmm')
    #DEFINE CURRENT DATE
    $currentDate = (Get-Date).ToString('MMddyy')
    #DEFINE CURRENT YEAR
    $currentYear = (Get-Date).ToString('yyyy')
    #DEFINE CURRENT MONTH
    $currentMonth = (Get-Date).ToString('MM')

    $students = Import-Csv -Path $insertFilePath

    #CREATE EMPTY ARRAY TO STORE NEW USERS DISCOVERED
    $i = 0
    $array = @()


    foreach ($student in $students)
    {
      #region VARIABLE DEFINITIONS...

      #region USER CREATION...
    }
    $array
    $array | Export-Csv -Path C:\IISD_IDM\IISD_NEW_$currentDate.csv -NoTypeInformation -Append
    $i

    $body = $array | Out-String

    #region EMAIL APPROPRIATE STAFF...
}
```

- Goals achieved
  - Hourly inserts!
  - DSA team provides consistent headers in each district regardless of SIS used
  - DSA team provides separate insert and update files based on user object existence
  - Creation Reporting
    - Hourly emails to appropriate district secretaries and Help Desk staff IF a user was created
    - Daily report file written to disk for reference
  - Network team provides O365 accounts through Azure AD Connect
  - Network team provides Google accounts through GCDS and GAPS

- 61 lines of code to 300+ lines after MANY iterations

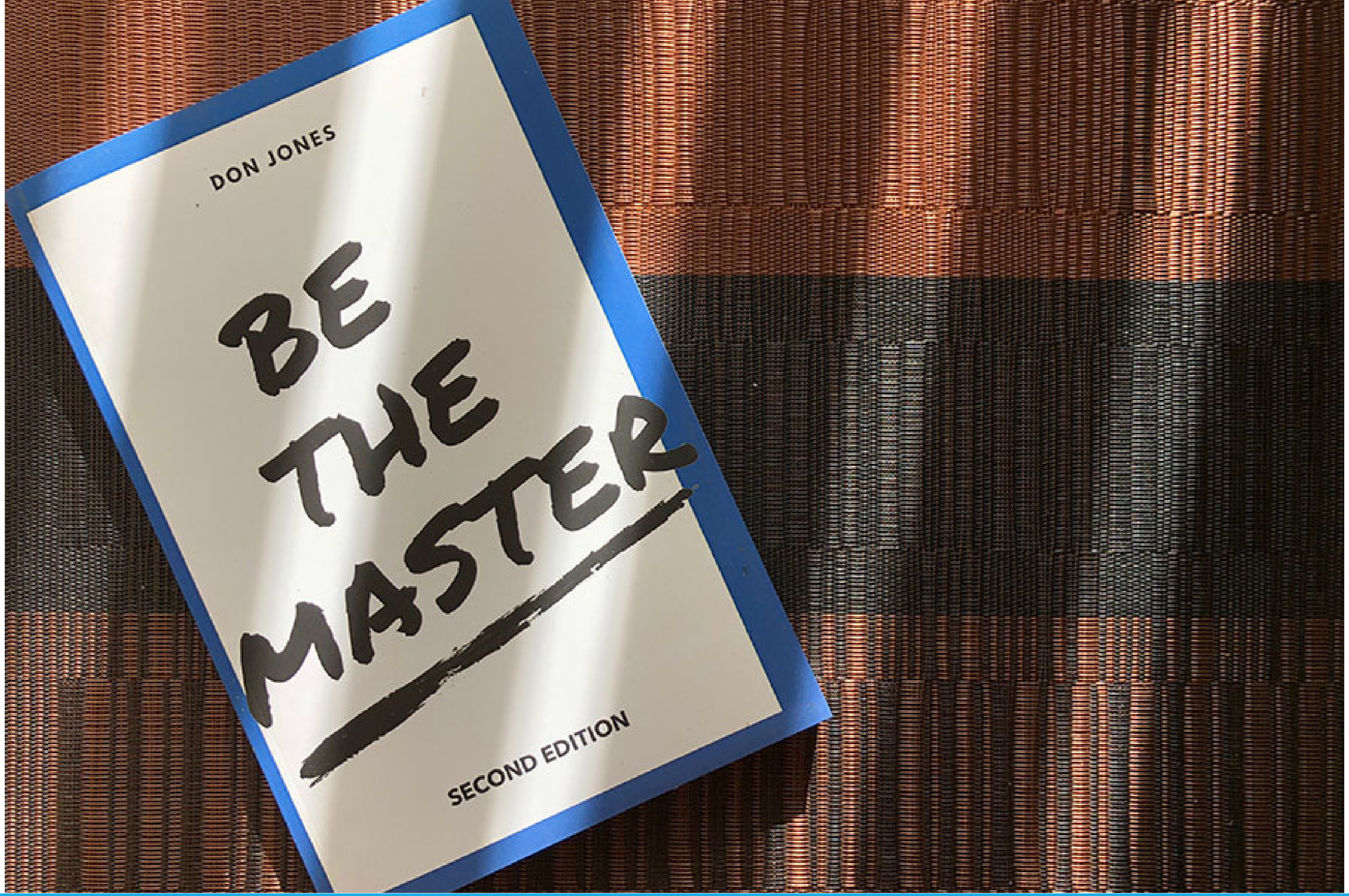
Expectations

- PowerShell – Intro Session For Those That Have Been Too Afraid To Take The Plunge
  - MAEDS 2017 Fall Conference - http://bit.ly/2NRCuGm
- Working knowledge of PowerShell cmdlets
- Basic understanding of objects
- Ability to read and understand 

```
PS C:\> Get-Help Get-Help -Full
```

- Understanding that the scripting process is iterative
  - StudentProvisioning.ps1 (2015)

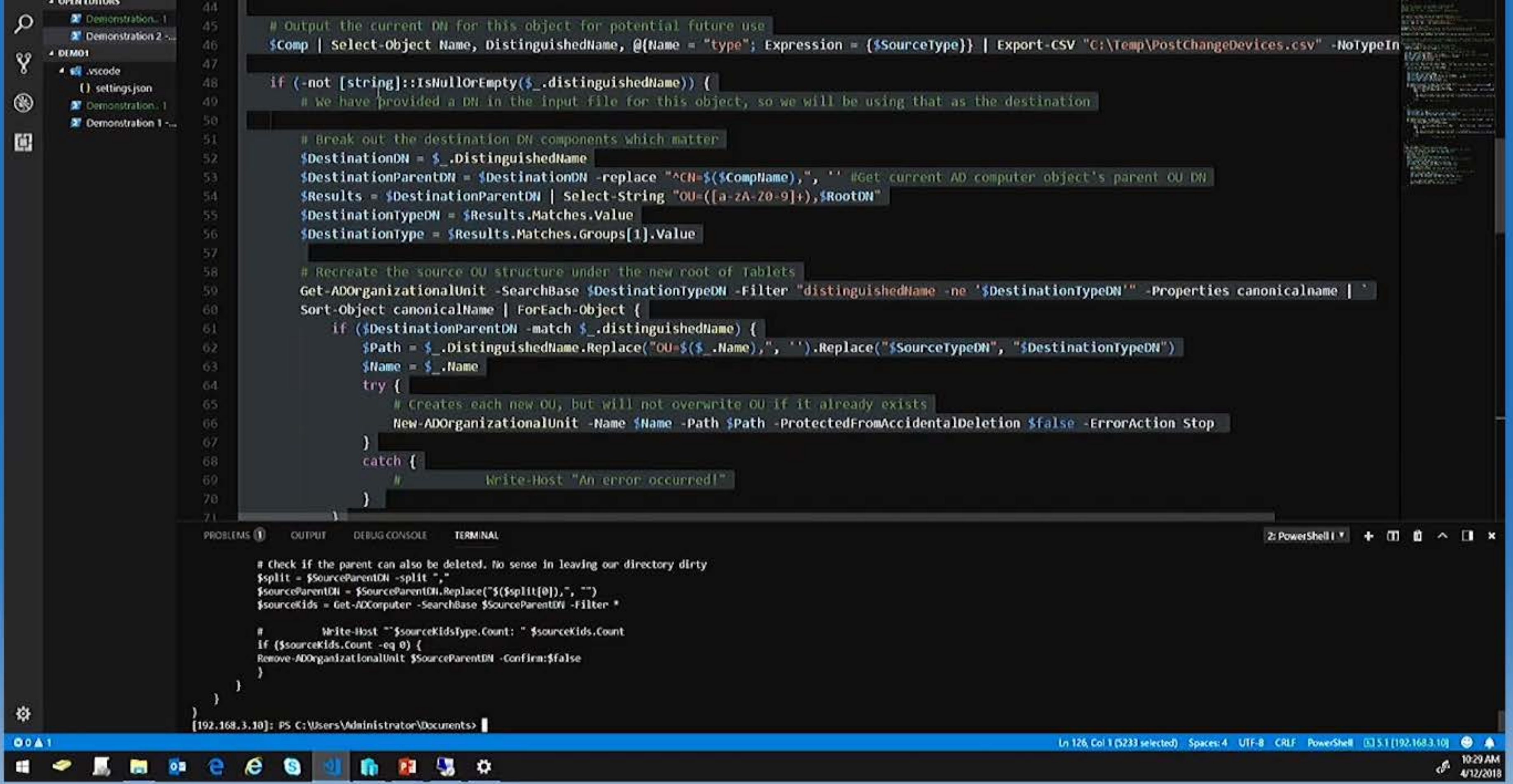
    vs.
  - <DISTRICT>_STU_INSERT.ps1 (2018)
- Some of you in this room know more than I do …



Be The Master

"Teaching does not always feel rewarding. It doesn't need to be. It is a repayment of something that was done for you. It is not a good thing that you do; it is an obligation that you have."

https://donjones.com/2017/10/19/become-the-master-or-go-away/



```
        # Output the current DN for this object for potential future use
        $Comp | Select-Object Name, DistinguishedName, @{Name = "type"; Expression = {$SourceType}} | Export-CSV "C:\Temp\PostChangeDevices.csv" -NoTypeIn

        if (-not [string]::IsNullOrEmpty($_.distinguishedName)) {
            # We have provided a DN in the input file for this object, so we will be using that as the destination

            # Break out the destination DN components which matter
            $DestinationDN = $_.DistinguishedName
            $DestinationParentDN = $DestinationDN -replace "^CN=$($CompName),", '' #Get current AD computer object's parent OU DN
            $Results = $DestinationParentDN | Select-String "OU=([a-zA-Z0-9]+),$RootDN"
            $DestinationTypeDN = $Results.Matches.Value
            $DestinationType = $Results.Matches.Groups[1].Value

            # Recreate the source OU structure under the new root of Tablets
            Get-ADOrganizationalUnit -SearchBase $DestinationTypeDN -Filter "distinguishedName -ne '$DestinationTypeDN'" -Properties canonicalname | `
            Sort-Object canonicalName | ForEach-Object {
                if ($DestinationParentDN -match $_.distinguishedName) {
                    $Path = $_.DistinguishedName.Replace("OU=$($_.Name),", '').Replace("$SourceTypeDN", "$DestinationTypeDN")
                    $Name = $_.Name
                    try {
                        # Creates each new OU, but will not overwrite OU if it already exists
                        New-ADOrganizationalUnit -Name $Name -Path $Path -ProtectedFromAccidentalDeletion $false -ErrorAction Stop
                    }
                    catch {
                        #           Write-Host "An error occurred!"
                    }
```

```
        # Check if the parent can also be deleted. No sense in leaving our directory dirty
        $split = $SourceParentDN -split ","
        $sourceParentDN = $SourceParentDN.Replace("$($split[0]),", "")
        $sourceKids = Get-ADComputer -SearchBase $SourceParentDN -Filter *

        #           Write-Host "`$sourceKidsType.Count: " $sourceKids.Count
        if ($sourceKids.Count -eq 0) {
        Remove-ADOrganizationalUnit $SourceParentDN -Confirm:$false
        }
      }
    }
}
[192.168.3.10]: PS C:\Users\Administrator\Documents>
```

[Scripting Big Changes With Small Risk by Brad Sterkenburg]

It's not always just about what the presenter is saying…

if (-not [string]::IsNullOrEmpty($variable))

vs.

if($variable -ne $null -AND $variable -ne '')

Topics of Discussion

What I plan to cover, but will probably run over and Eric Krebill will be giving me the ‘hurry up’ motions while tapping his wrist where a watch should be…

- Scripting Logic
- Importing Data
- Exporting Data
- Custom Objects
- Custom Functions
- Personal Profile
- So Much More…

Scripting Logic

- Comparison Operators
- foreach vs. ForEach vs. ForEach-Object
- Where-Object
- If, ElseIf, and Else Statements
- Switch Statements

Comparison Operators

Learn to accept that you can't use the symbols you might be used to from other languages.

There are no:

>, >=, =, <=, <

symbols to use.

The sooner you can get over that, the sooner you can stop making code to debug.

- Get-Help about_Comparison_Operators

-eq	equals
-ne	not equals
-gt	greater than
-ge	greater than or equal
-lt	less than
-le	less than or equal
-like	returns $true if string matches wildcard
-notlike	returns $true if string does not match wildcard
-match	returns $true if string matches regex
-notmatch	returns $true if string does not match regex
-contains	returns $true if reference value in a collection
-notcontains	returns $true if reference value not in a collection
-in	returns $true if test value in a collection
-notin	returns $true if test value not in a collection
-replace	replaces a string pattern
-is	returns $true if both objects are the same type
-isnot	returns $true if the objects are not the same type

foreach
vs.
ForEach
vs.
ForEach-Object

The main thing to remember is that ForEach-Object can pass data through the pipeline for the next command, but foreach will throw an error.

It will be annoying to troubleshoot.

Especially when you remember that you learned that last time you had to troubleshoot this same mistake you made.

- Get-Help about_foreach
  - foreach is a **language keyword**
  - Less memory efficient
  - Faster results (if you have the memory for it)
  - Cannot pass output to another command via the pipeline
- Get-Help ForEach-Object
  - ForEach is shorthand for ForEach-Object
  - % is also shorthand for ForEach-Object
  - More memory efficient
  - Slower results
  - Can pass output to another command via the pipeline

Getting to Know ForEach and ForEach-Object

SYNTAX:

```
foreach ( $thing in $things ) { Do-Stuff }
```

```
ForEach-Object –InputObject $things –Process { Do-Stuff }
```

EXAMPLE:

```
Measure-Command { 1..100000 | ForEach-Object $_ }
```

vs.

```
Measure-Command { foreach ( $i in ( 1..100000 ) ) { $i } }
```

Where-Object

This is what you'll use to limit the objects returned to you.

Sometimes you only want to import the 6th graders from a CSV or only select a specific staff member from an Active Directory query.

Sometimes you might want to identify when a rogue device enters a school district/building for the day.

- Get-Help Where-Object
  - ? is shorthand for Where-Object

SYNTAX:

```
| Where-Object { $_.Property –eq Statement }
```

EXAMPLE:

```
Import-CSV –Path C:\scripts\students.csv | ? { $_.GradeLevel –eq '6' }
```

EXAMPLE:

```
$scopes = Get-DhcpServerv4Scope -ComputerName $dhcpserver

foreach($scope in $scopes)

{

Get-DhcpServerv4Lease -ComputerName $dhcpserver -ScopeId
$scope.ScopeId | ?{ $_.ClientId -like "*59-51*" }

}
```

If, ElseIf, and Else Statements

Evaluates criteria against possible results.

It can be as simple as the ‘If’ statement, with a condition in parentheses, and a script block in curly brackets. If the condition evaluates as true it will run the code in the script block. If the condition evaluates as false it will skip that step.

From there you could use ‘ElseIf’ statements and/or ‘Else’ statement to run code against those situations.

- Get-Help about_If

SYNTAX:

```
If ( $thing –like “*$this*” ) { Do-Stuff }
```

EXAMPLE:

```
If ( ( $currentHomeDirectory -like $correctHomeDirectory ) –AND
     ( $homeDrive -like 'H:‘ ) )

{ if ( Test-Path –Path $correctHomeDirectory ) { Write-Host “Done” }

  else { New-HomeFolder –correctHomeDirectory $correctHomeDirectory
                         –samAccountName  $samAccountName }

ElseIf ($currentHomeDirectory -eq $null)

{ New-HomeFolder –correctHomeDirectory $correctHomeDirectory
                 –samAccountName $samAccountName

  Set-ADUser –Identity $samAccountName –HomeDirectory
                   $correctHomeDirectory –HomeDrive 'H:'

}

Else { <evaluate more stuff!> }
```

Switch Statements

Evaluates a single criteria against multiple possibilities. It’s equivalent to a bunch of if statements, but without having to write ‘if’ a ton.

- Get-Help about_Switch
- Everything you ever wanted to know about the switch statement

SYNTAX:

```
Switch ($thing)
{
Value1 { Do-Stuff }
default { Do-ThisStuffIGuess }
}
```

EXAMPLE:

```
switch ($buildingCode)
{
‘33000-D-12' { $buildingShortName = ‘ITS‘ }
'33000-D-01' { $buildingShortName = 'SUP‘ }
default { $buildingShortName = ‘UNPLACED’ }
}
```

Importing Data

- TEXT FILES
  - Get-Content
- CSV FILES
  - Import-Csv
- XML FILES
  - Import-Clixml

Get-Content

It's like reading a file, but not having it as straight text…but each line of text is an object and you get returned a collection of objects to work with.

Hopefully you get to deal with CSV or XML files mostly or have a Linux friend to help you parse text. Tell them it's kind of like 'cat', but you need to use foreach loops to work with each object of text…

- Get-Help Get-Content
- Why Get-Content Ain't Yer Friend
  - "Actually, the real problem is that most newcomers don't really understand that PowerShell is an object-oriented, rather than a text-oriented shell"
  - "just remember that, by default, Get-Content isn't just reading a stream of text all at once. You'll be getting, and need to be prepared to deal with, a *collection* of objects."
- A Faster Get-Content Cmdlet
  - "Extensive testing seems to indicate I get better performance by reading a large number of lines, but that a value of 0 is slightly counterproductive, at least on large files."

SNYTAX:

```
Get-Content –Path $filepath
```

EXAMPLE:

```
If ( ( Get-Content –Path $insertFilePath ) -ne $null )

{

$students = Import-Csv -Path $insertFilePath

}
```

Import-Csv

Creates table-like custom objects from the items in the CSV file. Each column in the CSV file becomes a property of the custom object and the items in the rows become the property values.

- Get-Help Import-Csv
- Comparing Figures From Two CSV Files, Writing A 3rd File

SYNTAX:

```
Import-Csv –Path $insertFilePath
```

EXAMPLE:

```
$students = Import-Csv -Path $insertFilePath
```

EXAMPLE:

```
$merged = Import-Csv -Path $insertFilePath `
| Select-Object -Property
idm_samaccountname,idm_first_name,idm_middle_name,idm_last_name,id
m_upn,idm_title,idm_status,idm_email,idm_employeeid,idm_employeenumb
er,@{n='idm_association_code';e={}},@{n='idm_association_name';e={}} `

| % { if ($departmentInformation.ContainsKey($_.idm_employeenumber))
{ $_.idm_association_code =
$departmentInformation[$_.idm_employeenumber][0]
$_.idm_association_name =
$departmentInformation[$_.idm_employeenumber][1]
} $_ }
```

Import-Clixml

This will create an object for each entity in the XML file if it is formatted correctly. Most useful when used in conjunction with Export-Clixml that saved to a file.

Most often you'll see this being used to store and retrieve credentials from a local file without revealing the password.

- Get-Help Import-Clixml
- Securely Store Credentials on Disk
  - "When reading the Solution, you might at first be wary of storing a password on disk. While it is natural (and prudent) to be cautious of littering your hard drive with sensitive information, the Export-CliXml cmdlet encrypts credential objects using the Windows standard Data Protection API. This ensures that only your user account can properly decrypt its contents."
- Not to be confused with when you typecast a xml object with Get-Content
  - $xmlDocument = [XML](Get-Content C:\scripts\Stupid_Lifetouch_CD\Student.xml)

SYNTAX:

```
Import-Clixml –Path $credPath
```

EXAMPLE:

```
$credential = Import-Clixml –Path $credPath
```

Exporting Data

- TEXT FILES
  - Out-File
- CSV FILES
  - Export-Csv
- XML FILES
  - Export-Clixml

Out-File

This will dump whatever you feed it as a file type you choose. I find it often easier/more useful to use Export-Csv instead of this, but it can be useful when dealing with HTML files.

- Get-Help Out-File
- Controlling PowerShell's Results with Out-File
  - "The biggest danger is 'over-think'; just remember that PowerShell takes care of basic file operations automatically. Consequently, there is no need to waste time looking for non-existent open-file, or save-file commands. If the file specified by Out-File does not already exist, PowerShell even creates it for you."

SYNTAX:

```
Out-File –Filepath $outFilePath
```

EXAMPLE:

```
Get-WmiObject –Class Win32_Service |

Select-Object –property Name,State |

Where-Object –Filter { $_.StartMode –eq 'Auto' –and $_.State –ne 'Running' }
| ConvertTo-HTML | Out-File BadNews.html
```

Export-Csv

Creates a CSV file from the objects that you feed it. Each row will be a separate object and the row will contain comma-separated values for the objects properties.

- Get-Help Export-Csv
- Export-Csv: The PowerShell Way to Treat CSV Files as First Class Citizens
  - "The PowerShell cmdlet Export-Csv is one of the many reasons that this language has saved everyone so much time. This cmdlet has transformed what was once a troublesome task of trying to wrangle loose text into some structured format with VBScript. Instead of messing around with Excel, we can instead use PowerShell to create CSV files."

SYNTAX:

```
Export-CSV –Path $exportCSVPath
```

EXAMPLE:

```
$array | Export-Csv -Path C:\IISD_IDM\IISD_NEW_STAFF_$currentDate.csv -NoTypeInformation -Append -Force
```

Export-Clixml

This will create a XML file that is formatted correctly when used in conjunction with Import-Clixml.

Most often you'll see this being used to store and retrieve credentials from a local file without revealing the password.

- Get-Help Export-Clixml
- Securely Store Credentials on Disk
  - "When reading the Solution, you might at first be wary of storing a password on disk. While it is natural (and prudent) to be cautious of littering your hard drive with sensitive information, the Export-CliXml cmdlet encrypts credential objects using the Windows standard Data Protection API. This ensures that only your user account can properly decrypt its contents."

SYNTAX:

```
Export-Clixml –Path $credPath
```

EXAMPLE:

```
$credential = Get-Credential

$credential | Export-Clixml –Path $credPath
```

Custom Objects

- New-Object
- Add-Member
- Hashtable or Array
  - PowerShell v3 of later
- [PSCustomObject]
  - PowerShell v3 or later

New-Object

This will create a.NET Framework or COM object.

I mostly use it to create .NET objects inside a loop that I can export at the end. It might be to export only the new users that were created that day or it might be to export data about something from multiple data source locations.

Sometimes I use it to open a COM object if I want to interact with Internet Explorer or Word.

- Get-Help New-Object
- Windows PowerShell: The Many Ways to a Custom Object
- Browsing in Internet Explorer via PowerShell

SYNTAX:

New-Object –TypeName <SOME OBJECT TYPE>

EXAMPLE:

$object = New-Object –TypeName PSObject

EXAMPLE:

$object = New-Object –TypeName PSObject –Property @{OS = “Windows”}

EXAMPLE:

$ie = New-Object –ComObject “InternetExplorer.Application”

Add-Member

This will be used to add properties and values to custom objects. It’s the ‘old school’ v1 way of doing things… I still feel it’s easier for new users to understand if they read a script.

That being said, you should move to the newer methods for creating objects.

- Get-Help Add-Member
- Using Add-Member in a Powershell One-Liner to Create Calculated Properties

SYNTAX:

```
Add-Member –MemberType <SOME MEMBER TYPE>
           –Name <NAME>
           –Value $value
```

EXAMPLE:

```
$object | Add-Member –MemberType NoteProperty –Name IP –Value
$ipaddress

$object | Add-Member –MemberType NoteProperty –Name HOSTNAME
–Value (Get-WmiObject -ComputerName $ipaddress
Win32_OperatingSystem).csname

$object | Add-Member –MemberType NoteProperty –Name MODEL –Value
(Get-WmiObject -ComputerName $ipaddress Win32_ComputerSystem).model

$object | Add-Member –MemberType NoteProperty –Name SERVICETAG
–Value (Get-WmiObject -ComputerName $ipaddress
Win32_BIOS).serialnumber

$array += $object
```

Hashtable or Array

PowerShell v3 or later

This is functionally the same as using Add-Member, but becomes a little cleaner and less verbose with this newer method.

That being said, there is one more method you can use…

- Get-Help about_Hash_Tables
- Powershell: Everything you wanted to know about PSCustomObject

SYNTAX:

```
$hashtable = @{

property1 = $value1
property2 = $value2

}

$object = New-Object –TypeName PSObject –Property $hashtable
```

EXAMPLE:

```
$hashtable = @{
IP = $ipaddress
HOSTNAME = (Get-WmiObject -ComputerName $ipaddress
Win32_OperatingSystem).csname
MODEL = (Get-WmiObject -ComputerName $ipaddress
Win32_ComputerSystem).model
SERVICETAG = (Get-WmiObject -ComputerName $ipaddress
Win32_BIOS).serialnumber
}

$object = New-Object –TypeName PSObject –Property $hashtable
```

[PSCustomObject]

PowerShell v3 or later

The last method I'll mention is typecasting the hashtable or array directly to a PSCustomObject.

- Get-Help about_Object_Creation
- Powershell: Everything you wanted to know about PSCustomObject
  - "I love using [PSCustomObject] in Powershell. Creating a usable object has never been easier. Because of that, I am going to skip over all the other ways you can create an object but I do need to mention that most of this is Powershell v3.0 and newer."

SYNTAX:

```
$hashtable = [PSCustomObject]@{
property1 = value1
property2 = value2
}
```

EXAMPLE:

```
$hashtable = [PSCustomObject]@{
IP = $ipaddress
HOSTNAME = (Get-WmiObject -ComputerName $ipaddress
Win32_OperatingSystem).csname
MODEL = (Get-WmiObject -ComputerName $ipaddress
Win32_ComputerSystem).model
SERVICETAG = (Get-WmiObject -ComputerName $ipaddress
Win32_BIOS).serialnumber
}
```

Custom Functions

- Get-Help about_Functions
- Windows PowerShell: Build a Better Function
  - "Obviously, not all of you are going to be in a position to create reusable tools for yourself and your coworkers. However, if you are creating reusable tools, these advanced functions are the only way to go."
- How To Make Use Of Functions in PowerShell
  - "Since PowerShell v3 automatic cmdlet discovery and module loading has been supported … However, it would be a good practice to add the Import-Module line to your script, so that another user is aware of where you are getting the functionality from."
- Building PowerShell Functions: Best Practices
  - "I spend a good deal of time wrapping common tasks into PowerShell functions. Here are a few best practices I've picked up along the way."
- Standard and Advanced PowerShell functions
  - "When you have been working with PowerShell for some time, creating reusable tools is an obvious evolution to avoid writing the same code over and over again. You will want to have modular pieces of code that only do one job and do it well - that's the role of functions."

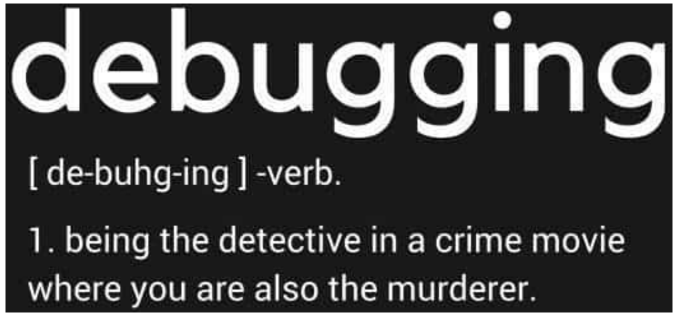
Personal Profile

- Get-Help about_Profiles
- Enhance your PowerShell experience by automatically loading scripts
  - "Here you go, you can now put as many ".ps1" files as you need in your "autoload" folder, and these will be executed everytime you launch your PowerShell session."
- Persistent PowerShell: The PowerShell Profile
  - "The profile is a simple yet powerful tool available to you, not at all complicated to use, and its uses are limited only by your imagination. About the only downside is all the time you are now going to spend searching for handy and clever things to add to your profile."
- If you don't have a $profle, create one and get going!

  -
    ```
    if ( ! ( Test-Path $Profile ) ) { New-Item -Type File -Path $Profile -Force }
    powershell_ise $profile
    ```
- Let's take a look at mine…

So Much More…

- https://docs.microsoft.com/en-us/powershell/
- http://ramblingcookiemonster.github.io/Pages/PowerShellResources/
  - PowerShell Basic Cheat Sheet
- http://jdhitsolutions.com/blog/essential-powershell-resources/
- PowerShell: Everything You Need to Know
- Getting Started with PowerShell 3.0 - Jeffrey Snover and Jason Helmick
- Advanced Tools & Scripting with PowerShell 3.0 – Jeffrey Snover and Jason Helmick
- Monad Manifesto – Jeffrey Snover's original vision from 2002 that would become PowerShell
- Monad Manifesto – Jeffrey Snover's post from 2011 with some history of the Monad Manifesto
- PowerShell + DevOps Global Summit
  - OnRamp Program – Content designed for entry-level technology professionals
  - OnRamp Scholarship – Applications are due by November 1st, 2018



debugging

[de-buhg-ing] -verb.

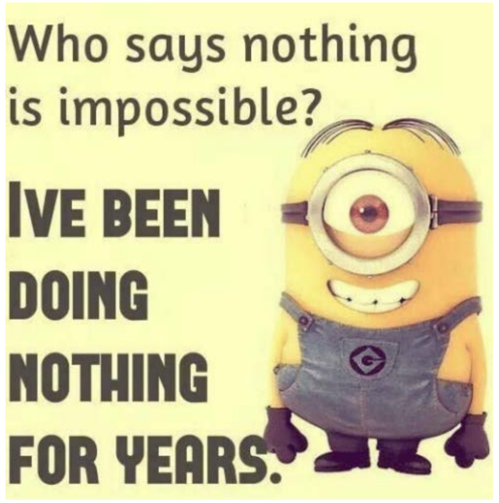
1. being the detective in a crime movie where you are also the murderer.

Final Thoughts

Learn and Network! NETWORKING PEOPLE AND TECHNOLOGY

Identify like-minded individuals and find a way to stay in contact with them or learn from them. It doesn't have to be from events like MAEDS or product-specific user groups. Though they are a great way to build a rapport with those people you may be asking for help in the future.

- Follow the MVPs and who they follow on Twitter
- Populate your favorite RSS Reader (R.I.P. Google Reader) with blogs
- Participate in the WinAdmins Slack group
- Watch conference videos on YouTube or Channel 9
- Connect on LinkedIn (anyone willing to endorse me for PowerShell?)


Who says nothing is impossible?
IVE BEEN DOING NOTHING FOR YEARS.