

PowerShell Intro –

Come Here Before You Attend Jim's Sessions

Name: Chris Thomas
Title: Desktop Engineer
Organization: Ingham ISD
Email: cthomas@inghamisd.org
Twitter: [@AutomateMyStuff](https://twitter.com/AutomateMyStuff)



**Ingham Intermediate
School District**

A Regional Educational Service Agency

<https://github.com/chrisATAutomatemystuff/Presentations>

PowerShell Intro –

Come Here Before You Attend Jim's Sessions

An introduction session to PowerShell and how it can benefit your life. This session will cover the basics of how PowerShell works, the first cmdlets you'll want to get comfortable with, and examples of how you can use PowerShell in your daily tasks. You'll want to hear this session before you attend any of Jim Tyler's deeper dive sessions on PowerShell.

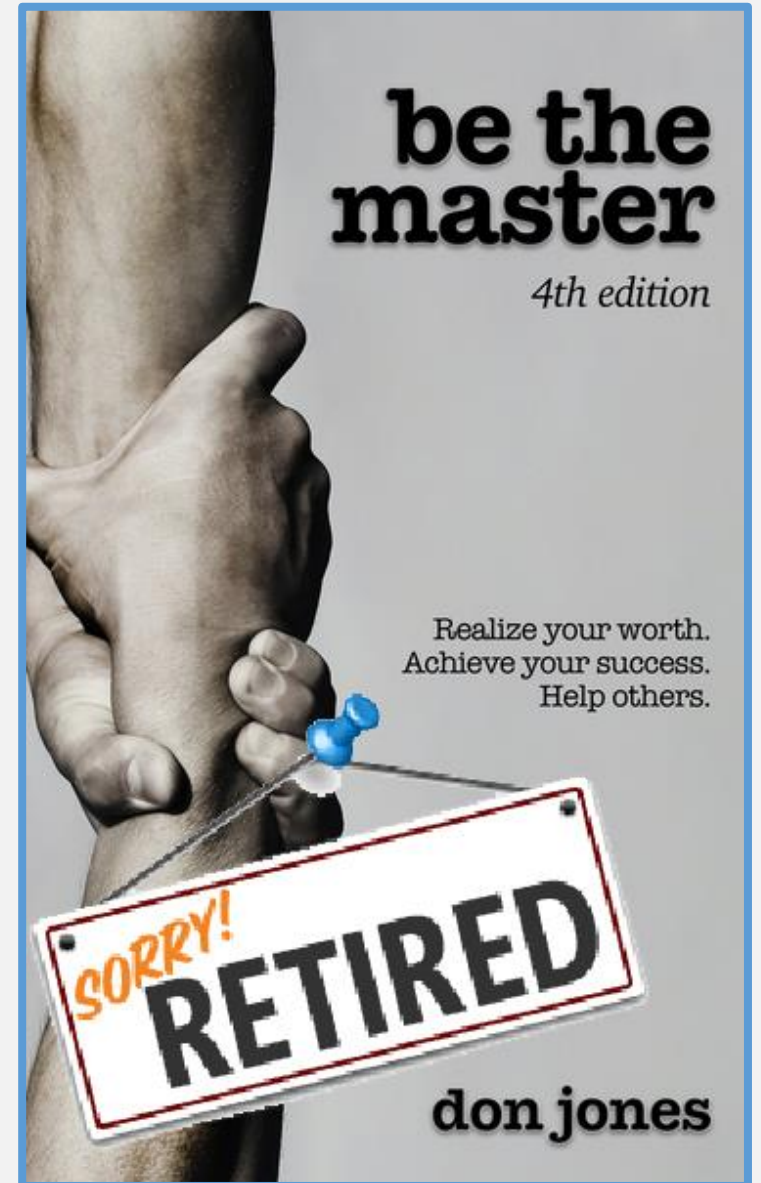


Who Am I?

- 23+ Years In K12 Technology
 - Intern, Technician, Coordinator, Engineer
- Lifelong Learner
 - [/r/sysadmin](#), [/r/k12sysadmin](#), RSS feeds
- Relentlessly Inquisitive
 - Let's Ask The [WinAdmins Community](#)
- Problem Solver
 - Professional [Googler](#)
- Voracious Reader
 - [docs.microsoft.com](#)
- Community Minded
 - [MAEDS](#), [MISCUG](#), [WMISMUG](#)
- Past Presentations
 - [github.com/chrisATautomatemystuff/Presentations](#)
- [#ImpostorSyndromeBeDamned](#)

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.”





Steve Schwartz
@jangosteve

Owner of Alfa Jango, CTO of Genomenon, co-founder of Carcode (acquired in 2014), engineer, developer, open-source enthusiast, guitarist, racecar driverist.

📍 Ann Arbor, Michigan 🌐 jangosteve.com 📅 Joined April 2008

195 Following 744 Followers



Don Jones®
@concentrateddon

Author. Foodie. Visit donjones.com for my books and more! Explore my fictional world at Witchkind.com.

📍 Las Vegas, NV 🌐 donjones.com 📅 Born November 22
📅 Joined September 2008

41 Following 18.2K Followers

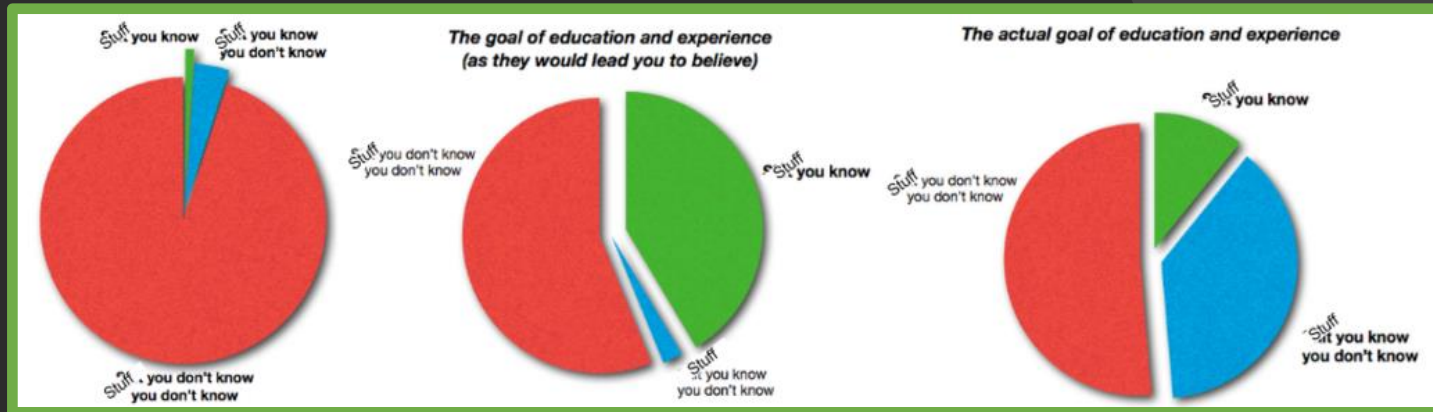


Dona Sarkar at #MBAS
@donasarkar

🚀 Leading Advocacy @microsoft ⚡ #PowerPlatform #FusionTeam 🐼 Code-blooded beast 🦄 @FastCompany 🏆 Most Productive 🧠 Dyslexic & dealing 💧 Designer

📍 Grounded 🤪 🌐 PrimaDonaStudios.com 📅 Joined July 2009

2,698 Following 75.1K Followers



Man, these guys are a lot smarter than me. I'll just sit here and try to look pretty.

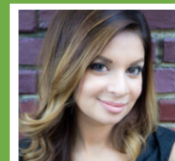
Here's a little secret: Every one of us feels that way, almost all the time.

It's called "Outsider Syndrome" or "Imposter Syndrome." *I never went to school for this. I can't believe how much they know about this. I'll never be that much of an expert. I can't possibly offer advice to anyone, because everyone else knows so much more.*

This attitude is, unfortunately, utter bullshit (not "udder bullshit," autocorrect, that's gross). And if you've ever caught yourself thinking that, you need to recognize where it comes from, and how to knock it off, because it's not only holding you back – it's preventing you from being a resource to others who could use your help.

Lose the Syndrome

Own what you know. Be confident, not arrogant. Ask questions. *Be wrong.* But don't be silent, and don't think you don't belong in the group.



The keynote speaker and MC of the show was [Dona Sarkar](#).

Dona is the engineering leader of the Windows Insider Program at Microsoft; a multi-published author; a fashion designer; co-founder of Fibonacci Sequins (a style blog devoted to showcasing awesome people in STEM); and a public speaker.

And Dona had a powerful message on banishing imposter syndrome that I thought needed to be shared with the Think Tank community.

“Imposter syndrome isn't just a woman thing. It's a human thing.”

Dona shared that men suffer from imposter syndrome too. She told stories of [Howard Shultz](#) and [Neil Armstrong](#) feeling like imposters or fakes.





I think the more you know, the more you realize just how much you don't know. So paradoxically, the deeper down the rabbit hole you go, the more you might tend to fixate on the growing collection of unlearned peripheral concepts that you become conscious of along the way.

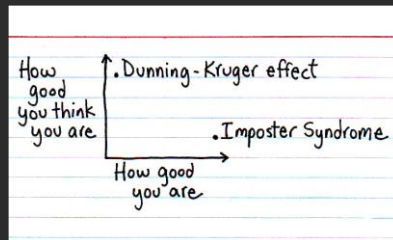
That can manifest itself as feelings of fraudulence when people are calling you a "guru" or "expert" while you're internally overwhelmed by the ever-expanding volumes of things you're learning that you don't know.

*However, I think it's important to tamp those insecurities down and continue on with confidence enough to continue learning. After all, you've got the advantage of having this long list of things you know you don't know, whereas most people haven't even taken the time to uncover that treasure map yet. What's more, no one else has it all figured out either. **We're all just fumbling around in the adjacent possible, grasping at whatever good ideas and understanding we can manage to wrap our heads around.***



Here's how to overcome impostor syndrome:

- › **Focus On Learning:** Forget appearing awesome. You can get better if you try, so focus on *that*.
- › **"Good Enough" Goals:** Stop trying to be perfect. (Yes, that was a typo. I'm not fixing it. It's good enough.)
- › **Take Off The Mask:** Talk to someone you think is facing the same issue. You're not alone.



1. I need to do more.
2. I need to forget about #1, and realize that I just want to do more, and desire a certain level of success that could come if I just focused more.
3. I need to not be so hard on myself with #1 and #2, and breathe.
4. Life can happen in the blink of an eye, no matter what age, so create the habit to build momentum.
5. There is a danger in comparison.

Why PowerShell?

Go away or I will replace you
with a
very small script.



jsnover
@jsnover



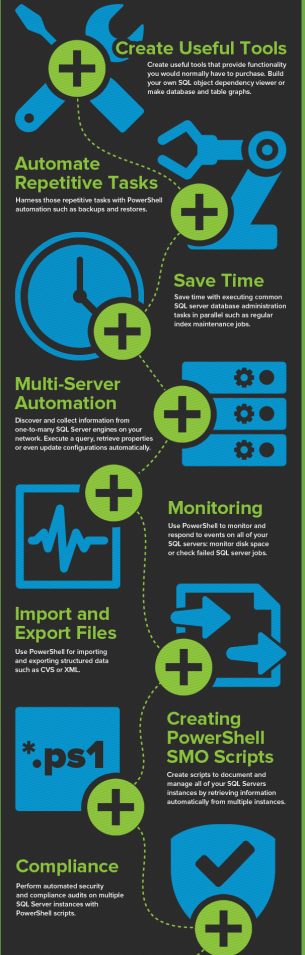
Following

GUI-only, Click-Next, no-value-add Admins
will be replaced with a new type of Admin -
the kind that greet you in the lobby
#LearnPowerShell



WHY DBAs SHOULD LEARN POWERSHELL

A more efficient way to manage and automate repetitive tasks on your SQL server.



idera

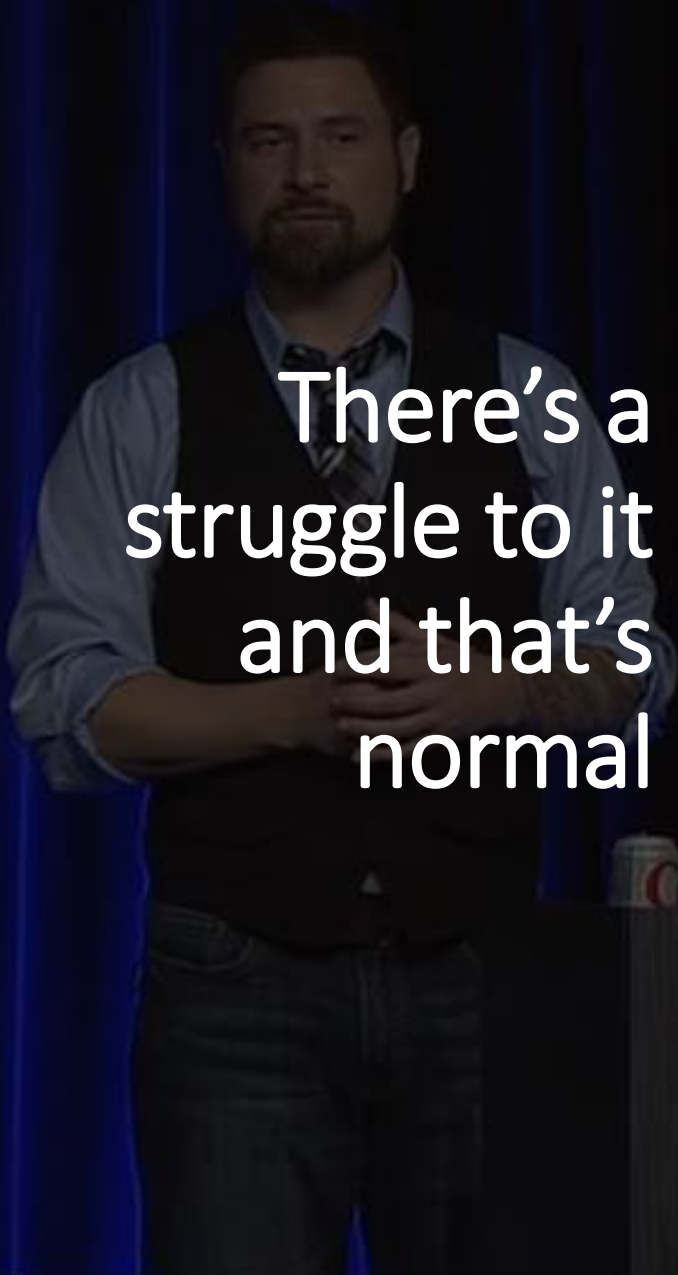
Copyright ©2015 Idera Inc. All rights reserved.

Why PowerShell?

- [Discoverability](#)
- [Consistency](#)
- [Verb-Noun Syntax](#)
- [Update-Help](#)
- [Object Orientation](#)
- [PSDrive](#)
- [Parameters](#)
- [Functions](#)
- [Modules](#)
 - [PowerShell App Deployment Toolkit](#)
 - [PowerCLI](#) (VMWare)
 - [AutoBrowse](#) (Internet Explorer Automation)
 - [WASP](#) (GUI Automation)
 - [ShowUI](#) (GUI Building)
 - [PSCX](#) (PowerShell Community Extensions)
 - [Posh-SSH](#) (SSH Automation)
- **NOT FREE, but awesome**
 - [ISESteroids](#) (ISE GUI enhancement, version control, script refactoring)

What does PowerShell work with?

- Active Directory (AD)
- Group Policy (GPO/GPP)
- Office 365
- SharePoint Server
- Microsoft Teams
- SQL Server
- Best Practice Analyzer (BPA)
- [RESTful](#)/[SOAP](#) API Calls
 - “You will learn Graph API and you will enjoy it” – Microsoft
- Internet Information Services (IIS)
- Remote Desktop Services (RDS)
- Configuration Manager (ConfigMgr/SCCM)
- File System
- Registry
- WMI/CIM
- Event Viewer
- AppLocker



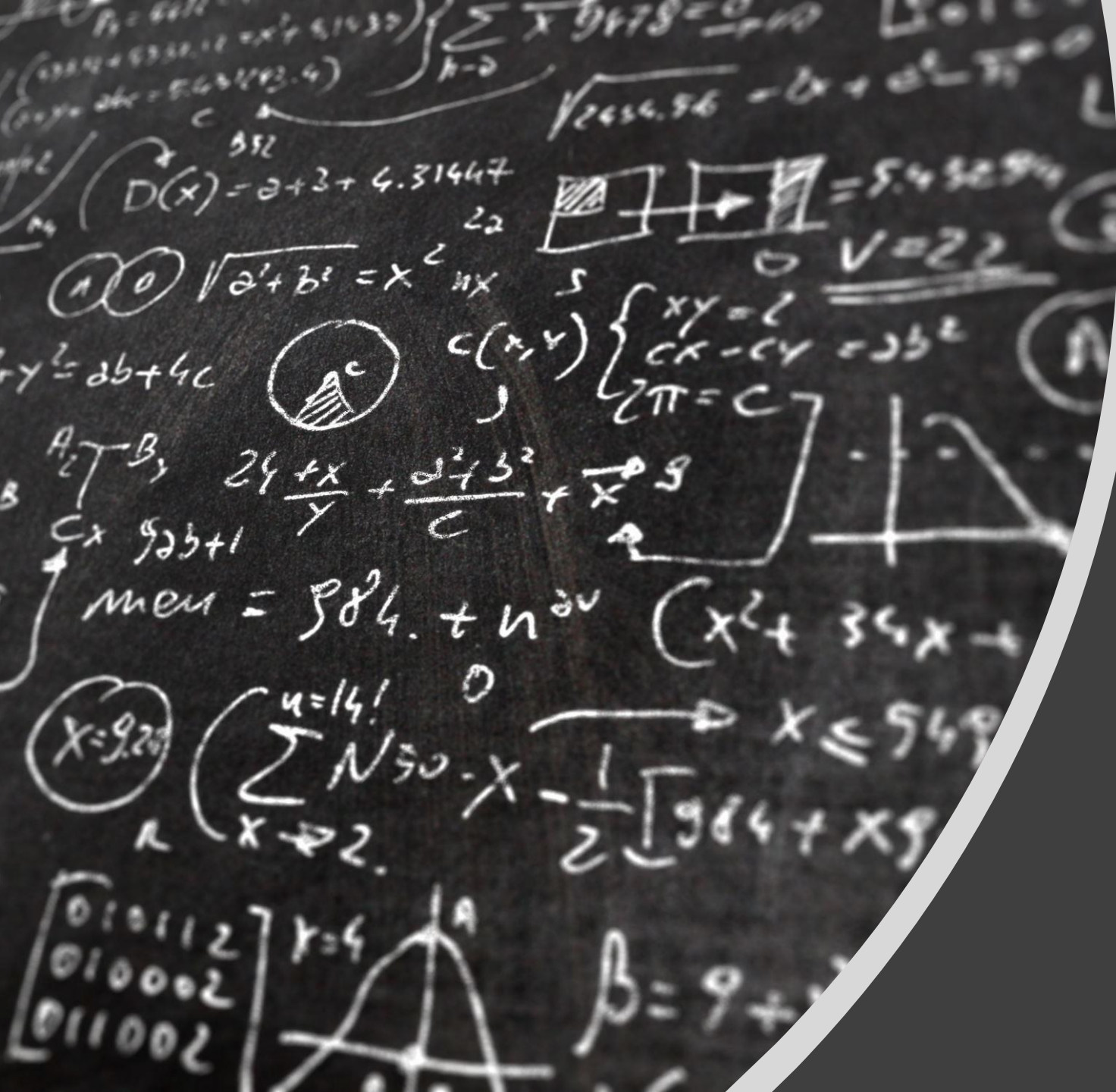
There's a
struggle to it
and that's
normal

- **Don Jones:** "If you're not willing to play a little bit you'll probably not be successful at PowerShell."
- **Jeffrey Snover:** "I'm a Distinguished Engineer, I'm the Lead Architect with Windows Server and System Center Datacenter, and I invented the dang thing and still there's a struggle to it and that's normal."

- Windows PowerShell Unplugged with Jeffrey Snover & Don Jones
TechEd North America 2014 • (1hr 16min • quotes @ 3:15)

<https://www.youtube.com/watch?v=qVIPNsAkJxM>

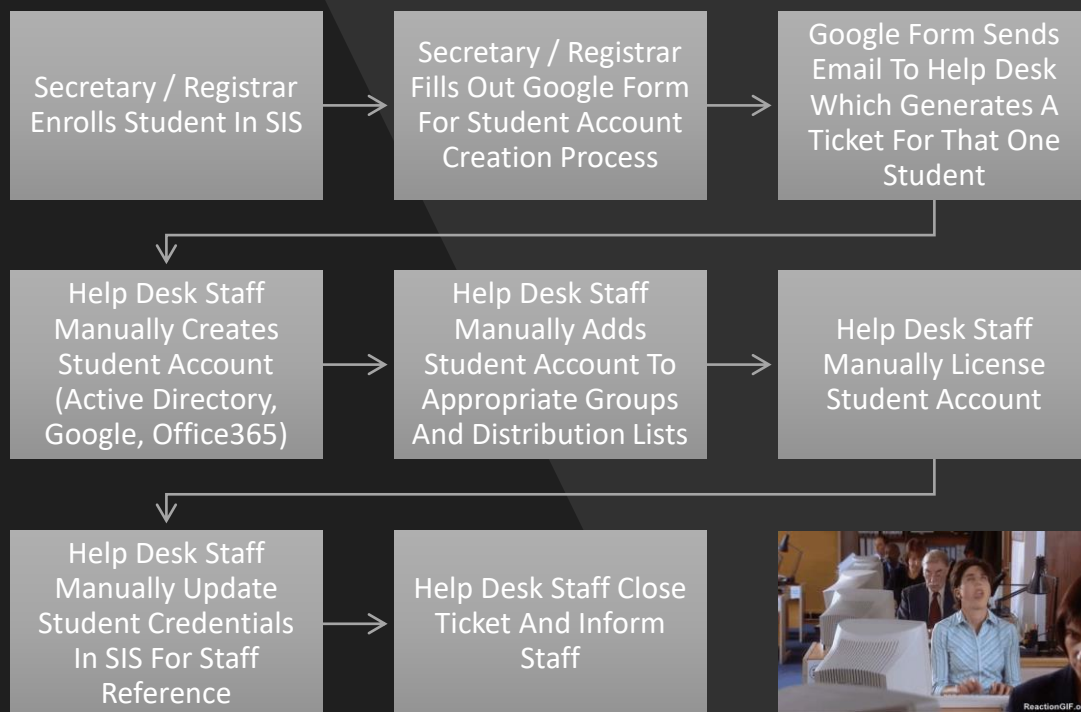




BRAINSTORM:

What manual task
annoys you the most that
you'd like to automate?

How It All Began...



```

1 <#>
2
3 Hoping this script at some point can grow into the ability to enter-ssession or invoke-ssession
4 to each domain controller and create students nightly based on automatic exports from PowerSchool.
5 Wonder if Tom can get it to export the headers that'd be nice to work with. Will have to add in
6 a check if student already exists, but still write-verbose or out-file to a report file at the end.
7 Should also check for presence of
8
9 #>
10
11 Import-Module ActiveDirectory
12
13 #region Import File Locations
14 #Students = Import-Csv C:\scripts\accounts09162015.csv
15 #endregion
16
17 foreach ($student in $students){
18
19 #region Student Variable Definitions
20 $FN = $student.FN
21 $LN = $student.LN
22 $NAME = $FN + " " + $LN
23 $YOG = $student.YOG
24 $SAM = $student.SN
25 $TEMPPWD = (ConvertTo-SecureString -AsPlainText "IWishYouHad8DigitPasswords" -Force)
26 $INSECUREPWD = $user.PWD
27 $SECUREPWD = (ConvertTo-SecureString -AsPlainText $INSECUREPWD -Force)
28 $UPN = $SAM + "@. "
29 $EMAIL = $SAM + "@. "
30 $PATH = "OU=" + $YOG + ",OU=StudentAccounts,OU=UserAccounts,DC= "
31 $GROUP = "U_All_" + $YOG + "Students"
32 #endregion
33
34 #region Active Directory Student Provisioning
35 New-ADUser
36 -Name $NAME
37 -DisplayName $NAME
38 -GivenName $FN
39 -SurName $LN
40 -SamAccountName $SAM
41 -UserPrincipalName $UPN
42 -EmailAddress $EMAIL
43 -AccountPassword $TEMPPWD
44 -CannotChangePassword $true
45 -PasswordNeverExpires $true
46 -Path $PATH
47 -PassThru | Enable-ADAccount
48
49 Add-ADGroupMember -Identity $GROUP -Members $SAM
50
51 Set-ADAccountPassword -Identity $SAM -Reset -NewPassword $SECUREPWD
52 #endregion
53
54 #region Office 365 Student Provisioning
55 #endregion
56
57
58 #region Google Apps Student Provisioning
59 #endregion
60
61 }
    
```



**It is better to
KNOW HOW TO LEARN
than to know.**

-Dr. Seuss

PowerShell Fundamentals

Vocabulary 01/02

- Let's get on the same page with our language...

- **Shell:** the command interpreter that is used to pass commands to the operating system
- **ISE:** the Integrated Scripting Environment is an application where you can run, test, and debug scripts in a single GUI with tab completion, syntax coloring, selective execution, and context-sensitive help
- **Cmdlet:** a task-oriented command that is typically used to return a .NET Framework object to the next command in the pipeline
- **Variable:** a name given to stored information, e.g. \$users, \$iWishYouHadAStrongPassword, \$x

- **Parameter:** input values or arguments used by the cmdlet or script to make it more dynamic

- **Named:**

```
PS C:\>  
PS C:\> Get-ChildItem -Path $env:USERPROFILE\AppData\Local\Temp -Filter *.exe
```

- **Positional:**

```
PS C:\>  
PS C:\> Get-ChildItem $env:USERPROFILE\AppData\Local\Temp *.exe
```

- **Switch:**

```
PS C:\>  
PS C:\> Get-ChildItem -Path $env:USERPROFILE\AppData\Local\Temp -Filter *.exe -Recurse
```

Object: a representation of something with methods to take actions against it and properties to access information stored within it

Vocabulary 02/02

- Let's get on the same page with our language...

- **Module:** a set of related Windows PowerShell functionalities, grouped together as a convenient unit (usually saved in a single directory)
 - **Script Module:** a file (.psm1) that contains PowerShell code for functions, variables and more
 - **Binary Module:** a .NET Framework assembly (.dll) that contains compiled code
 - **Dynamic Module:** a module that only exists in memory (Import-PSSession)
- **Pipeline (|):** a method to send the results of the preceding command as input to the next command
- **\$_:** "the current object in the pipeline", or "this", or \$PSItem
- **Function:** a command or series of commands grouped to run together
- **Alias:** shortcut to a command, cmdlet or function
- **Array:** a data structure that serves as a collection of multiple items. You can iterate over the array or access individual items using an index
- **Comment:** You can type a comment symbol (#) before each line of comments, or you can use the (<#) and(#>) symbols to create a comment block
- **Escape Character:** the grave-accent (`) is a continuation character if used at end of line or displays literal value of a string / variable
- **Command History:** <up arrow> or <down arrow> and Get-History

Tab Expansion

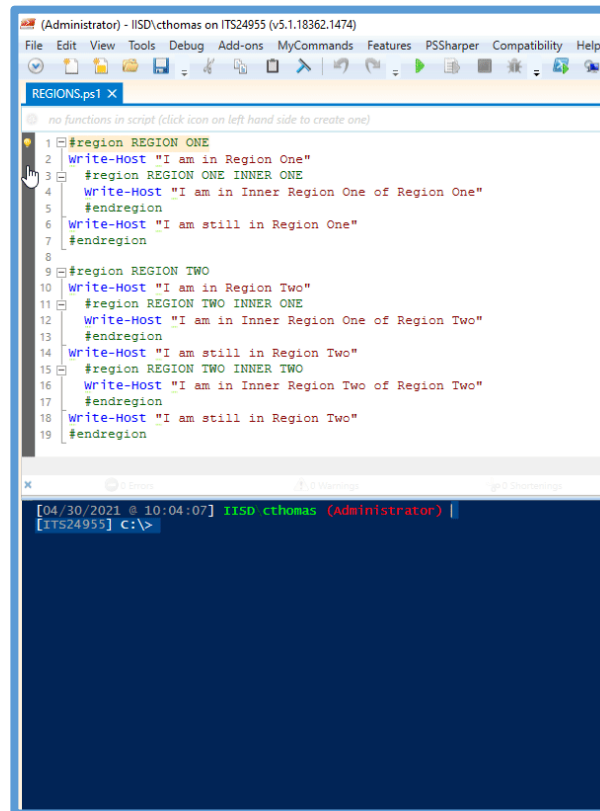
- PS C:\> new-adu<TAB>
- PS C:\> New-ADUser -<TAB>
 - <TAB> to advance through parameters
 - or
 - <SHIFT+TAB> to go back to a previous parameter
- PS C:\> New-ADUser -<CTRL + SPACE>
 - <ARROW KEYS> + <SPACE> to select appropriate parameter



<https://docs.microsoft.com/en-us/powershell/scripting/learn/using-tab-expansion>

Regions

- Separate code into collapsible sections
- <CTRL> + M to collapse/expand all regions
- Collapsing regions only affects readability, not runability



The screenshot shows the PowerShell ISE (Integrated Scripting Environment) interface. The title bar indicates the user is Administrator on a machine named ITS24955. The menu bar includes File, Edit, View, Tools, Debug, Add-ons, MyCommands, Features, PSSharper, Compatibility, and Help. The toolbar contains various icons for file operations, editing, and execution. The script editor displays a file named 'REGIONS.ps1'. The script content is as follows:

```
1 #region REGION ONE
2 Write-Host "I am in Region One"
3 #region REGION ONE INNER ONE
4 Write-Host "I am in Inner Region One of Region One"
5 #endregion
6 Write-Host "I am still in Region One"
7 #endregion
8
9 #region REGION TWO
10 Write-Host "I am in Region Two"
11 #region REGION TWO INNER ONE
12 Write-Host "I am in Inner Region One of Region Two"
13 #endregion
14 Write-Host "I am still in Region Two"
15 #region REGION TWO INNER TWO
16 Write-Host "I am in Inner Region Two of Region Two"
17 #endregion
18 Write-Host "I am still in Region Two"
19 #endregion
```

The script is executed in the console window at the bottom, which shows the command prompt for the user 'c:\>'.

<https://devblogs.microsoft.com/scripting/use-regions-in-powershell-ise-2/>

Command Syntax

- <command-name> -<Required Parameter Name> <Required Parameter Value>
- [-<Optional Parameter Name> <Optional Parameter Value>]
- [-<Optional Switch Parameters>]
- [-<Optional Parameter Name>] <Required Parameter Value>

SYNTAX

```
New-ADUser [-Name] <String> [-AccountExpirationDate <DateTime>] [-AccountNotDelegated <Boolean>] [-AccountPassword <SecureString>] [-AllowReversiblePasswordEncryption <Boolean>] [-AuthenticationPolicy <ADAuthenticationPolicy>] [-AuthenticationPolicySilo <ADAuthenticationPolicySilo>] [-AuthType {Negotiate | Basic}] [-CannotChangePassword <Boolean>] [-Certificates <X509Certificate[]>] [-ChangePasswordAtLogon <Boolean>] [-City <String>] [-Company <String>] [-CompoundIdentitySupported <Boolean>] [-Country <String>] [-Credential <PSCredential>] [-Department <String>] [-Description <String>] [-DisplayName <String>] [-Division <String>] [-EmailAddress <String>] [-EmployeeID <String>] [-EmployeeNumber <String>] [-Enabled <Boolean>] [-Fax <String>] [-GivenName <String>] [-HomeDirectory <String>] [-HomeDrive <String>] [-HomePage <String>] [-HomePhone <String>] [-Initials <String>] [-Instance <ADUser>] [-KerberosEncryptionType {None | DES | RC4 | AES128 | AES256}] [-LogonWorkstations <String>] [-Manager <ADUser>] [-MobilePhone <String>] [-Office <String>] [-OfficePhone <String>] [-Organization <String>] [-OtherAttributes <Hashtable>] [-OtherName <String>] [-PassThru] [-PasswordNeverExpires <Boolean>] [-PasswordNotRequired <Boolean>] [-Path <String>] [-POBox <String>] [-PostalCode <String>] [-PrincipalsAllowedToDelegateToAccount <ADPrincipal[]>] [-ProfilePath <String>] [-SamAccountName <String>] [-ScriptPath <String>] [-Server <String>] [-ServicePrincipalNames <String[]>] [-SmartcardLogonRequired <Boolean>] [-State <String>] [-StreetAddress <String>] [-Surname <String>] [-Title <String>] [-TrustedForDelegation <Boolean>] [-Type <String>] [-UserPrincipalName <String>] [-Confirm] [-WhatIf] [<CommonParameters>]
```

SYNTAX

```
Add-ADGroupMember [-Identity] <ADGroup> [-Members] <ADPrincipal[]> [-AuthType {Negotiate | Basic}] [-Credential <PSCredential>] [-Partition <String>] [-PassThru] [-Server <String>] [-Confirm] [-WhatIf] [<CommonParameters>]
```

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_command_syntax

Splatting

- Remove the need for horizontal scrolling
- Remove the need for backticks (or troubleshooting forgotten backticks...)
- Cleaner look to the script

```
1 Import-Module ActiveDirectory
2
3 #region Import File Locations
4 $students = Import-Csv C:\scripts\students_import.csv
5 #endregion
6
7 foreach ($student in $students){
8
9 #region Student Variable Definitions
10 $FN = $student.FN
11 $LN = $student.LN
12 $NAME = $FN + " " + $LN
13 $YOG = $student.YOG
14 $SAM = $student.SN
15 $TEMPPWD = (ConvertTo-SecureString -AsPlainText "IWishYouHad8DigitPasswords" -Force)
16 $INSECUREPWD = $student.PWD
17 $SECUREPWD = (ConvertTo-SecureString -AsPlainText $INSECUREPWD -Force)
18 $SUPN = $SAM + "@<DOMAIN NAME>"
19 $EMAIL = $SAM + "@<DOMAIN NAME>"
20 $PATH = "OU=" + $YOG + ",OU=StudentAccounts,OU=UserAccounts,DC=<DOMAIN NAME>,DC=<DOMAIN TYPE>"
21 $GROUP = "U_All_" + $YOG + "Students"
22 #endregion
23
24 #region Active Directory Student Provisioning
25 New-ADUser -Name $NAME -DisplayName $NAME -GivenName $FN
26
27 Add-ADGroupMember -Identity $GROUP -Members $SAM
28
29 Set-ADAccountPassword -Identity $SAM -Reset -NewPassword $SECUREPWD
30 #endregion
31 }
```

```
1 Import-Module ActiveDirectory
2
3 #region Import File Locations
4 $students = Import-Csv C:\scripts\students_import.csv
5 #endregion
6
7 foreach ($student in $students){
8
9 #region Student Variable Definitions
10 $FN = $student.FN
11 $LN = $student.LN
12 $NAME = $FN + " " + $LN
13 $YOG = $student.YOG
14 $SAM = $student.SN
15 $TEMPPWD = (ConvertTo-SecureString -AsPlainText "IWishYouHad8DigitPasswords" -Force)
16 $INSECUREPWD = $student.PWD
17 $SECUREPWD = (ConvertTo-SecureString -AsPlainText $INSECUREPWD -Force)
18 $SUPN = $SAM + "@<DOMAIN NAME>"
19 $EMAIL = $SAM + "@<DOMAIN NAME>"
20 $PATH = "OU=" + $YOG + ",OU=StudentAccounts,OU=UserAccounts,DC=<DOMAIN NAME>,DC=<DOMAIN TYPE>"
21 $GROUP = "U_All_" + $YOG + "Students"
22 #endregion
23
24 #region Active Directory Student Provisioning
25 New-ADUser
26     -Name $NAME
27     -DisplayName $NAME
28     -GivenName $FN
29     -Surname $LN
30     -SamAccountName $SAM
31     -UserPrincipalName $SUPN
32     -EmailAddress $EMAIL
33     -AccountPassword $TEMPPWD
34     -CannotChangePassword $true
35     -PasswordNeverExpires $true
36     -Path $PATH
37     -PassThru | Enable-ADAccount
38
39 Add-ADGroupMember -Identity $GROUP -Members $SAM
40
41 Set-ADAccountPassword -Identity $SAM -Reset -NewPassword $SECUREPWD
42 #endregion
43 }
```

```
1 Import-Module ActiveDirectory
2
3 #region Import File Locations
4 $students = Import-Csv C:\scripts\students_import.csv
5 #endregion
6
7 foreach ($student in $students){
8
9 #region Student Variable Definitions
10 $FN = $student.FN
11 $LN = $student.LN
12 $NAME = $FN + " " + $LN
13 $YOG = $student.YOG
14 $SAM = $student.SN
15 $TEMPPWD = (ConvertTo-SecureString -AsPlainText "IWishYouHad8DigitPasswords" -Force)
16 $INSECUREPWD = $student.PWD
17 $SECUREPWD = (ConvertTo-SecureString -AsPlainText $INSECUREPWD -Force)
18 $SUPN = $SAM + "@<DOMAIN NAME>"
19 $EMAIL = $SAM + "@<DOMAIN NAME>"
20 $PATH = "OU=" + $YOG + ",OU=StudentAccounts,OU=UserAccounts,DC=<DOMAIN NAME>,DC=<DOMAIN TYPE>"
21 $GROUP = "U_All_" + $YOG + "Students"
22 #endregion
23
24 $newUserSplat = @{
25     Name = $NAME
26     DisplayName = $NAME
27     GivenName = $FN
28     Surname = $LN
29     SamAccountName = $SAM
30     UserPrincipalName = $SUPN
31     EmailAddress = $EMAIL
32     AccountPassword = $TEMPPWD
33     CannotChangePassword = $true
34     PasswordNeverExpires = $true
35     Path = $PATH
36     PassThru = $null
37 }
38
39 #region Active Directory Student Provisioning
40 New-ADUser @newUserSplat | Enable-ADAccount
41
42 Add-ADGroupMember -Identity $GROUP -Members $SAM
43
44 Set-ADAccountPassword -Identity $SAM -Reset -NewPassword $SECUREPWD
45 #endregion
46 }
```

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_splatting

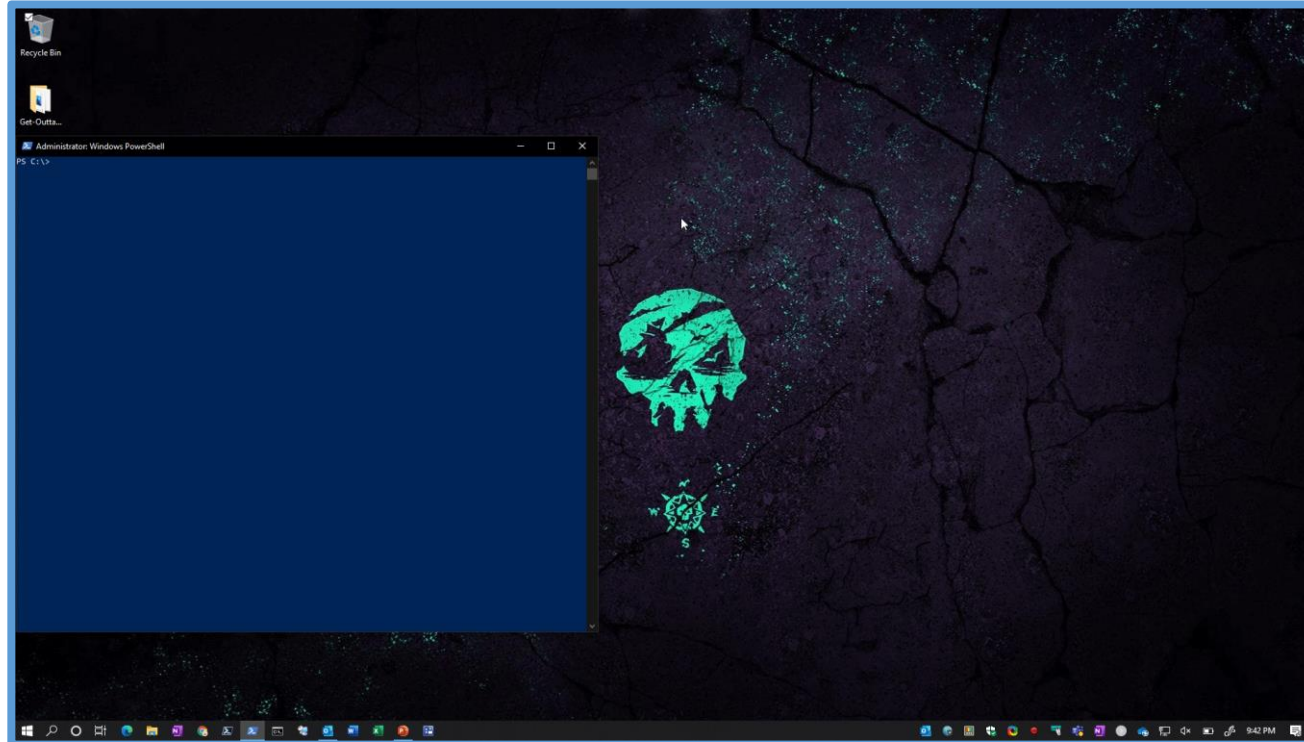
PowerShell

First Cmdlets

Get-Help

- PS C:\> Get-Help New-ADUser
- PS C:\> Get-Help New-ADUser -Examples
- PS C:\> Get-Help New-ADUser -Full
- PS C:\> Get-Help New-ADUser -ShowWindow
- PS C:\> Get-Help New-ADUser -Online

<https://docs.microsoft.com/en-us/powershell/scripting/learn/ps101/02-help-system>



<https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/get-help>

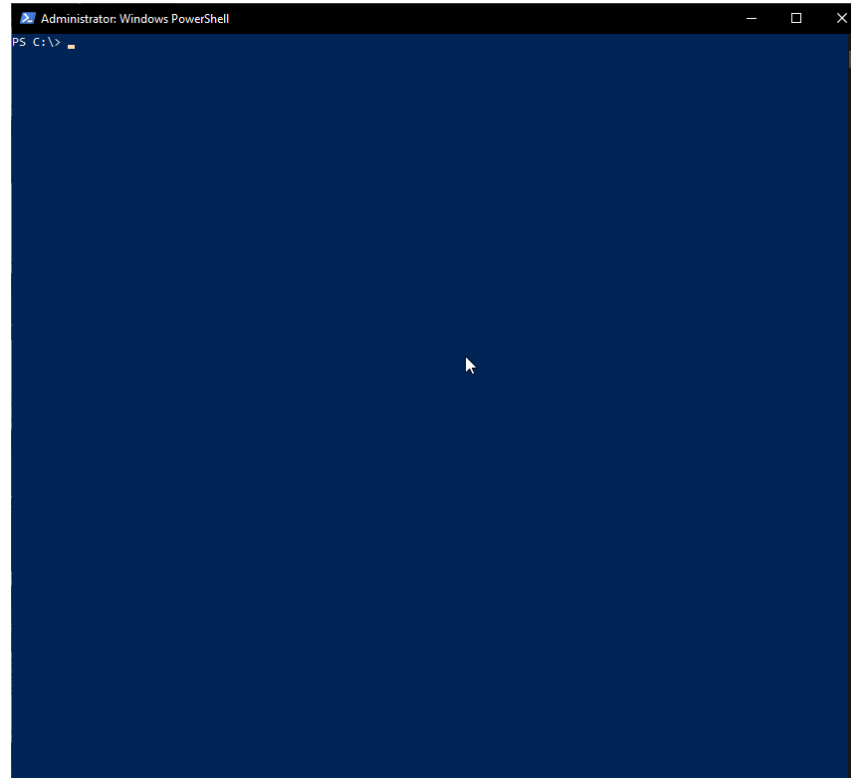
Get-Command

- PS C:\> Get-Command -Module ActiveDirectory
- PS C:\> Get-Command -Verb New
- PS C:\> Get-Command -Verb New -Noun AD*
- PS C:\> Get-Command New-AD*
- PS C:\> Get-Command New-ADUser

<https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/get-command>

Get-Member

- PS C:\> <cmdlet> | Get-Member
- Caveat
- PS C:\> Get-ADUser cthomas | Get-Member
- vs.
- PS C:\> Get-ADUser cthomas -Properties * | Get-Member



<https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-member>

Import-CSV

- SYNTAX:
- Import-Csv -Path \$insertFilePath
- EXAMPLE:
- \$students = Import-Csv -Path \$insertFilePath

```
1 $insertFilePath = "C:\Users\idm\students\insert.csv"
2 $content = get-content -Path $insertFilePath
3
4 if($content -ne $null)
5 {
6
```

```
18 $students = Import-Csv -Path $insertFilePath
```

<https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/import-csv>

Export-CSV

- SYNTAX:
- Export-CSV -Path \$exportCSVPath
- EXAMPLE:
- \$array | Export-Csv -Path C:\IISD_IDM\IISD_NEW_STAFF_\$(currentDate).csv -NoTypeInformation -Append -Force

```
20 #CREATE EMPTY ARRAY TO STORE NEW USERS DISCOVERED
21 $i = 0
22 $array = @()
```

```
237 #region USER CREATION
238 $i = $i + 1
239 $array += $student
240
241 Write-Host "Creating user: $samAccountName" -ForegroundColor Green
```

```
322 $array
323 $array | Export-Csv -Path C:\IISD_IDM\IISD_NEW_$(currentDate).csv -NoTypeInformation -Append
```

<https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/export-csv>

PowerShell Flow Control & Automation

Flow Control Operators

- [About Operator Precedence](#)

In the following list, operators are listed in the order that they are evaluated. Operators on the same line, or in the same group, have equal precedence.

The Operator column lists the operators. The Reference column lists the PowerShell Help topic in which the operator is described. To display the topic, type `get-help <topic-name>`.

OPERATOR	REFERENCE
<code>\$() @() () @()</code>	about_Operators
<code>. ?.</code> (member access)	about_Operators
<code>::</code> (static)	about_Operators
<code>[<i>e</i>] ?[<i>e</i>]</code> (index operator)	about_Operators
<code>[<i>int</i>]</code> (cast operators)	about_Operators
<code>-split</code> (unary)	about_Split
<code>-join</code> (unary)	about_Join
<code>,</code> (comma operator)	about_Operators
<code>++ --</code>	about_Assignment_Operators
<code>! -not</code>	about_Logical_Operators
<code>..</code> (range operator)	about_Operators
<code>-f</code> (format operator)	about_Operators
<code>-</code> (unary/negative)	about_Arithmetic_Operators
<code>* / %</code>	about_Arithmetic_Operators
<code>+ -</code>	about_Arithmetic_Operators

The following group of operators have equal precedence. Their case-sensitive and explicitly case-insensitive variants have the same precedence.

OPERATOR	REFERENCE
<code>-split</code> (binary)	about_Split
<code>-join</code> (binary)	about_Join
<code>-is -isnot -as</code>	about_Type_Operators
<code>-eq -ne -gt -lt -le</code>	about_Comparison_Operators
<code>-like -notlike</code>	about_Comparison_Operators
<code>-match -notmatch</code>	about_Comparison_Operators
<code>-in -notin</code>	about_Comparison_Operators
<code>-contains -notcontains</code>	about_Comparison_Operators
<code>-replace</code>	about_Comparison_Operators

The list resumes here with the following operators in precedence order:

OPERATOR	REFERENCE
<code>-band -bnot -bor -bxor -shr -shl</code>	about_Arithmetic_Operators
<code>-and -or -xor</code>	about_Logical_Operators

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_operators

Flow Control Comparison Operators

- By default, all comparison operators are case-insensitive. To make a comparison operator case-sensitive, add a c after the -. For example, -ceq is the case-sensitive version of -eq. To make the case-insensitivity explicit, add an i before -. For example, -ieq is the explicitly case-insensitive version of -eq.

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

Flow Control Logical Operators

- The PowerShell logical operators connect expressions and statements, allowing you to use a single expression to test for multiple conditions.

- -and TRUE when both statements are TRUE
- -or TRUE when either statement is TRUE
- -xor TRUE when only one statement is TRUE
- -not Negates the statements that follows
- ! Same as -not

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_logical_operators

Flow Control

Conditional Statements

If / Elseif / Else

SYNTAX:

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

```
279 #SET THE MIDDLE NAME AND INITIALS IF THE USER HAS THEM
280 if($middleName -ne $null -AND $middleName -ne '')
281 {
282     Set-ADUser -Identity $samAccountName -OtherName $middleName -Initials $middleInitial
283 }
284 else
285 {
286     Set-ADUser -Identity $samAccountName -Clear MiddleName,Initials
287 }
288
289 #ADD THE USER TO THEIR GROUPS
290 if($groupList -ne $null)
291 {
292     $groupList = $groupList | ForEach-Object {Get-ADGroup -Identity $_}
293     $groupList | ForEach-Object {Add-ADGroupMember -Identity $_ -Members $samAccountName}
294 }
295
```

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_if

Flow Control

Conditional Statements

Switch

SYNTAX:

Switch (\$thing)

{

Value1 { Do-Stuff }

default { Do-ThisStuffIGuess }

}

```
70 #DEFINE THE STATE BUILDING CODE FOR THE USER
71 $buildingCode = $student.idm_building01code
72
73 #DEFINE BUILDING VARIABLES BASED ON STATE BUILDING CODES
74 switch ($buildingCode)
75 {
76     '3' {
77         $buildingShortName = ' '
78         $office = ' '
79         $streetAddress = ' '
80         $city = ' '
81         $postalCode = ' '
82         $officePhone = ' '
83
84         $passwordInsecure = $student.idm_default_password
85         $passwordSecure = (ConvertTo-SecureString -AsPlainText $passwordInsecure -Force)
86     }
87     '2' {
88         $buildingShortName = ' '
89         $office = ' '
90         $streetAddress = ' '
91         $city = ' '
92         $postalCode = ' '
93         $officePhone = ' '
94
95         $passwordInsecure = $student.idm_default_password
96         $passwordSecure = (ConvertTo-SecureString -AsPlainText $passwordInsecure -Force)
97     }
98     '3' {
99         $buildingShortName = ' '
```

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_switch

Flow Control Conditional Statements Where-Object

SYNTAX:

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

EXAMPLE:

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

<https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/where-object>

Flow Control Looping Statements foreach vs ForEach (ForEach-Object)

- [Getting to Know ForEach and ForEach-Object](#)

- 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

SYNTAX:

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

EXAMPLE:

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

- 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

SYNTAX:

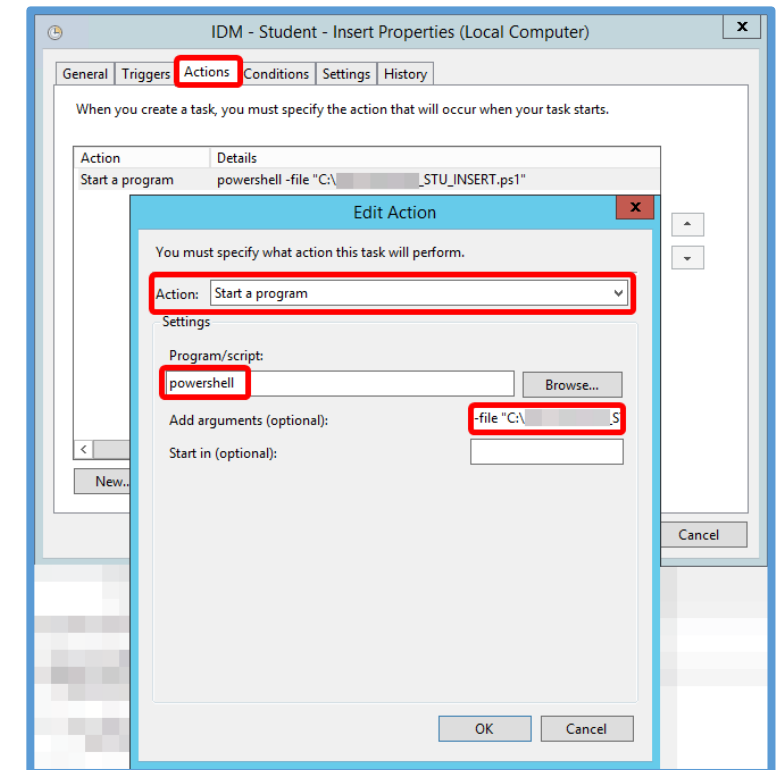
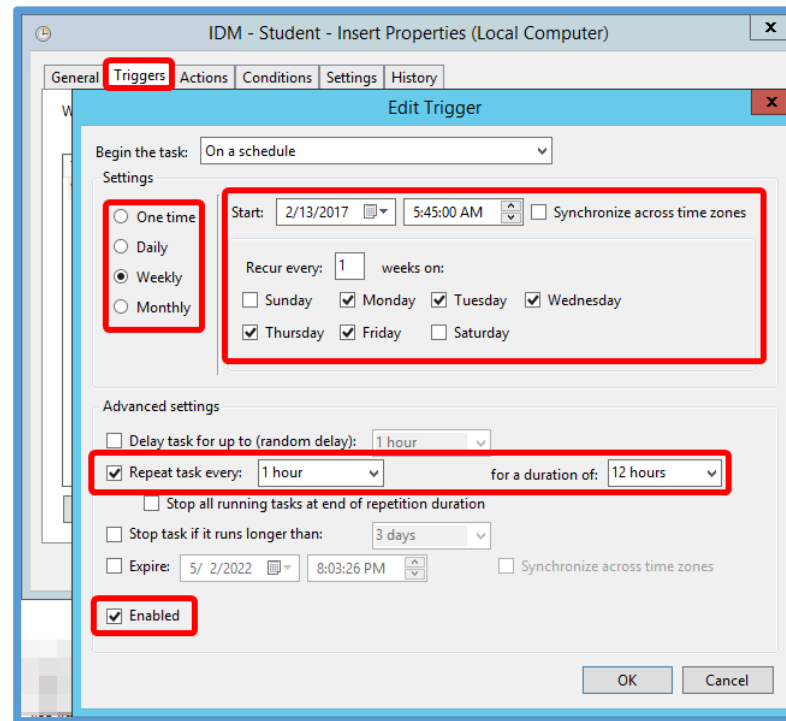
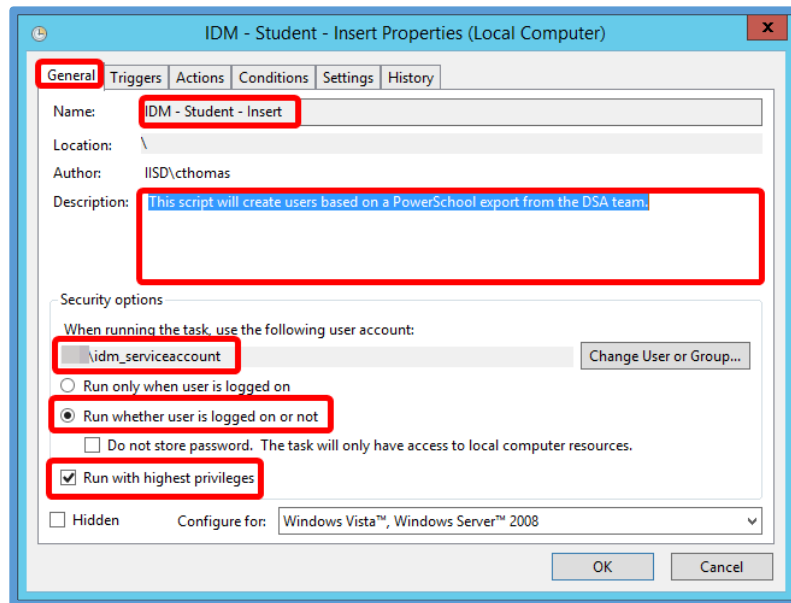
```
ForEach-Object -InputObject $things -Process { Do-Stuff }
```

EXAMPLE:

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

Task Scheduler

- Run scripts on-demand until you are comfortable and then schedule them to run automatically so you can focus on other tasks
 - [How to Schedule PowerShell Script using Task Scheduler](#)
- Use a service account with delegated permissions, not Domain Admin, to run your tasks
 - [Active Directory Delegated Permissions Best Practices](#)
 - [Delegate AD group management](#)



Useful Cmdlets To Know

New-ADUser Multiple Users

Let's pull everything together...

```
1 $insertFilePath = "i:\ad\students\insert.csv"
2 $content = get-content -Path $insertFilePath
3
4 if($content -ne $null)
5 {
```

```
18 $students = Import-Csv -Path $insertFilePath
```

```
25 foreach ($student in $students)
26 {
27     #region VARIABLE DEFINITIONS
28     #DEFINE ALL THE THINGS FROM EXPORT FILE
29     $firstName = $student.idm_first_name
30     $firstInitial = $firstName.Substring(0,1)
31     $middleName = $student.idm_middle_name
32     if($middleName -ne $null -AND $middleName -ne '')
33     {
34         $middleInitial = $middleName.Substring(0,1)
35     }
36     $lastName = $student.idm_last_name
37     $lastInitial = $lastName.Substring(0,1)
38
39     $fullName = $firstName + ' ' + $lastName
40     $displayName = $firstInitial + ' ' + $lastName
41
42     $samAccountName = $student.idm_samaccountname
43     $userPrincipalName = $student.idm_upn
44     $userPrincipalName = $samAccountName + "@students.ingham.mi.edu"
45     $emailAddress = $student.idm_email
46
47     $title = 'Student'
48
49     $gradeLevel = $student.idm_student_grade_level
50     $yearOfGrad = $student.idm_student_graduation_year
51
52     $employeeID = $student.idm_employeeid
53     $employeeNumber = $student.idm_employeenumber
54     $employeeNumberLength = $employeeNumber.Length
55
56     $studentID = $employeeNumber.Substring(3)
57     $studentIDLast4 = -join "$studentID"[-4..-1]
```

```
70 #DEFINE THE STATE BUILDING CODE FOR THE USER
71 $buildingCode = $student.idm_building01code
72
73 #DEFINE BUILDING VARIABLES BASED ON STATE BUILDING CODES
74 switch ($buildingCode)
75 {
76     '3000' {
77         $buildingShortName = '3000'
78         $office = '3000'
79         $streetAddress = '3000'
80         $city = '3000'
81         $postalCode = '3000'
82         $officePhone = '3000'
83
84         $passwordInsecure = $student.idm_default_password
85         $passwordSecure = (ConvertTo-SecureString -AsPlainText $passwordInsecure -Force)
86     }
87     '3002' {
88         $buildingShortName = '3002'
89         $office = '3002'
90         $streetAddress = '3002'
91         $city = '3002'
92         $postalCode = '3002'
93         $officePhone = '3002'
94
95         $passwordInsecure = $student.idm_default_password
96         $passwordSecure = (ConvertTo-SecureString -AsPlainText $passwordInsecure -Force)
97     }
98     '3003' {
99         $buildingShortName = '3003'
```

```
216 #DEFINE OU PATH FOR THE SCHOOL
217 $ouPath_append = ',DC='
218 $ouPath = 'OU=Student,OU=' + $buildingShortName + $ouPath_append
219 $ouPath_disabled = 'OU=Disabled Users,DC='
220
221 #DEFINE THE USER DESCRIPTION
222 $description = $buildingShortName + ' - Class of ' + $yearOfGrad
223
224 #DEFINE DISTRICT SPECIFIC VARIABLES
225 $organization = 'Ingham Intermediate School District'
226 $state = 'MI'
227 $company = 'Ingham Intermediate School District'
228 $domainName = 'ingham.mi.edu'
229 $domainAddress = 'ingham.mi.edu'
```

```
243 #DEFINE ALL THE NEW USER ATTRIBUTES FOR SPLATTING
244 $newUserSplat = @{
245     Name = $fullName
246     DisplayName = $displayName
247     GivenName = $firstName
248     Surname = $lastName
249     SamAccountName = $samAccountName
250     UserPrincipalName = $userPrincipalName
251     EmailAddress = $emailAddress
252     AccountPassword = $passwordSecure
253     ChangePasswordAtLogon = $false
254     CannotChangePassword = $true
255     PasswordNeverExpires = $true
256     Path = $ouPath
257     StreetAddress = $streetAddress
258     City = $city
259     State = $state
260     PostalCode = $postalCode
261     Organization = $organization
262     Company = $company
263     Office = $office
264     OfficePhone = $officePhone
265     Department = $department
266     Title = $title
267     Description = $description
268     EmployeeID = $employeeID
269     EmployeeNumber = $employeeNumber
270     Enabled = $true
271 }
272
273 #CREATE THE USER BASED ON SPLAT
274 New-ADUser @newUserSplat
275 Set-ADUser $samAccountName -Add @{ } -Replace @{ }
```

```
294 #SET THE USERS PASSWORD
295 #IF FINE-GRAINED PASSWORD POLICIES ARE IN EFFECT THEY WILL BE HONORED
296 #IF THE USER IS IN THE APPROPRIATE SECURITY GROUP BEFORE THIS COMMAND
297 Set-ADAccountPassword -Identity $samAccountName -Reset -NewPassword $passwordSecure
```

<https://docs.microsoft.com/en-us/powershell/module/activedirectory/new-aduser>

Add-ADGroupMember

```
58 |  
59 | #CREATE AN EMPTY ARRAY TO STORE THE SECURITY GROUPS TO ADD THE USER TO  
60 | $groupList = @()  
61 |  
62 | #ADD THE USER TO THE DEFAULT GROUPS  
63 | $groupList += 'Content Filter - Students'  
64 | $groupList += 'Students'  
65 |
```

```
131 | $groupList += 'STV Students'
```

```
189 | $groupList += 'PasswordComplexityDisable'
```

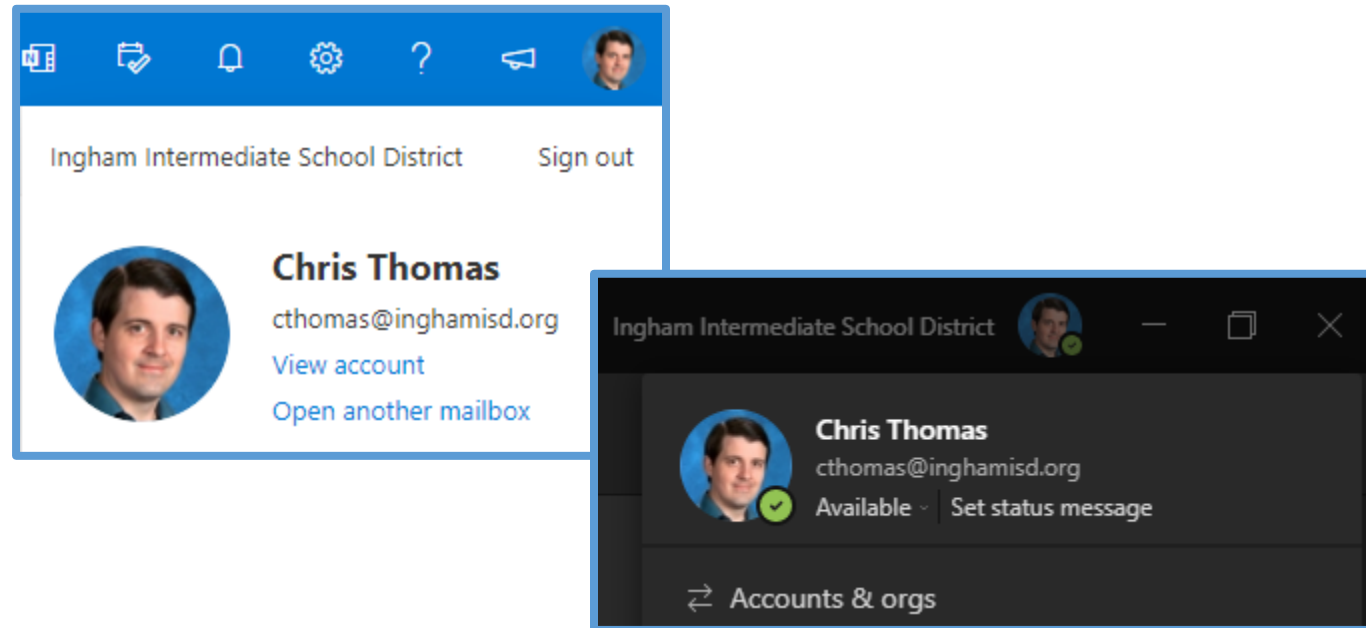
```
288 |  
289 | #ADD THE USER TO THEIR GROUPS  
290 | if($groupList -ne $null)  
291 | {  
292 |     $groupList = $groupList | ForEach-Object {Get-ADGroup -Identity $_}  
293 |     $groupList | ForEach-Object {Add-ADGroupMember -Identity $_ -Members $samAccountName}  
294 | }  
295 |
```

Name Changes

- Consider if/how you will support changing first names:
 - Christopher vs. Chris
 - William vs. Bill
 - Barbara vs. Barb
 - Margaret vs. Peggy
 - Secretary / Registrar Mistake
- Consider if/how you will support changing last names:
 - Marriage / Divorce
 - Adoption
 - Secretary / Registrar Mistake
- Consider only accepting changes through your authoritative source:
 - SIS for student data
 - HR system for staff data
- What user object attributes will change if you support name changes?
 - DisplayName
 - GivenName / Surname
 - HomeDirectory
 - Mail
 - Name
 - SamAccountName
 - UserPrincipalName
- What impact will user object attribute changes have on your environment?
 - How long will you support proxyAddresses for a user?
 - Are your online services flexible enough to support UserPrincipalName/SamAccountName/Mail changes and still retain the user's data/history?

Set-UserPhoto

- “Don’t Be A Nobody”
 - Outlook Prescence
 - Teams Prescence



<https://docs.microsoft.com/en-us/powershell/module/exchange/set-userphoto>

Disable-ADAccount or Set-ADUser -Enabled \$false

- Regularly check for and remove inactive user accounts in Active Directory
 - Understanding the AD Account attributes - LastLogon, LastLogonTimeStamp and LastLogonDate

```
1 $usersToDisable = Import-Csv -Path c:\scripts\userstodisable.csv
2
3 foreach ($userToDisable in $usersToDisable)
4 {
5     $user = Get-ADUser $userToDisable.SamAccountName
6     $user | Disable-ADAccount
7     $user | Move-ADObject -TargetPath 'OU=DisabledUsers,DC=<DOMAIN NAME>,DC=<DOMAIN TYPE>'
8 }
```

```
1 Import-Module ActiveDirectory
2
3 $students = Import-Csv -Path ".\idm_students_update.csv"
4 $ouPath_disabled = 'OU=Disabled Users,DC='
5
6 foreach ($student in $students)
7 {
8     $samAccountName = $student.idm_samaccountname
9
10    try{
11        #IF THE USER EXISTS DO THIS STUFF BELOW
12        $user = Get-ADUser -Identity $samAccountName -Properties *
13
14        #IF THE USER IS INACTIVE IN POWERSCHOOL DO THIS STUFF
15        if($student.idm_status -eq 'I')
16        {
17            #DEFINE THE LAST LOGON OF THIS USER
18            $lastLogon = $user.LastLogon
19
20            if($lastLogon -eq '0')
21            {
22                $lastLogonDate = 'never'
23            }
24            else
25            {
26                $lastLogonDate = $user.LastLogonDate
27            }
28
29            #DEFINE WHERE THE USER WAS ORIGINALLY BEFORE WE MOVE THE OBJECT
30            $distinguishedName = $user.DistinguishedName
31            $originalOU = $distinguishedName | ForEach-Object { $_ -replace '^.+?(?<!\|\\)', '' }
32
33            Write-Host "Disabling user: $samAccountName from $originalOU with a last logon of $lastLogonDate"
34            Write-Host "I will set their description to: (LL: $lastLogonDate - O-OU: $originalOU - D: $(Get-Date -Format d) - A: IDM)"
35
36            #DISABLE THE USER
37            Set-ADUser -Identity $samAccountName -Enabled $false
38            #CHANGE THE USER DESCRIPTION
39            Set-ADUser -Identity $distinguishedName -Description "LL: $lastLogonDate - O-OU: $originalOU - D: $(Get-Date -Format d) - A: IDM"
40            #MOVE THE USER OBJECT TO THE DISABLED OU
41            Move-ADObject -Identity $user.DistinguishedName -TargetPath $ouPath_disabled
42        }
43    }
44    catch{
45        $_.
46    }
47 }
48 }
```

<https://docs.microsoft.com/en-us/powershell/module/activedirectory/disable-adaccount>

Remove-ADGroupMember

- Consider auditing / logging the users current group memberships before stripping them of membership ... just in case you need to revert the changes.

```
1 $usersToDisable = Import-Csv -Path c:\scripts\usertodisable.csv
2
3 foreach ($userToDisable in $usersToDisable)
4 {
5     $user = Get-ADUser $userToDisable -Properties memberof
6     $user | Disable-ADAccount
7     $user | Move-ADObject -TargetPath 'OU=DisabledUsers,DC=<DOMAIN NAME>,DC=<DOMAIN TYPE>'
8 }
```

```
1 $usersToDisable = Import-Csv -Path c:\scripts\usertodisable.csv
2
3 foreach ($userToDisable in $usersToDisable)
4 {
5     $user = Get-ADUser $userToDisable -Properties memberof
6     $user | Disable-ADAccount
7     $user | Move-ADObject -TargetPath 'OU=DisabledUsers,DC=<DOMAIN NAME>,DC=<DOMAIN TYPE>'
8     $groups = $user.MemberOf
9
10    foreach ($group in $groups)
11    {
12        Write-Host "Removing $($user.SamAccountName) from $group"
13        Remove-ADGroupMember -Identity $group -Members $user -Confirm:$false
14    }
15 }
```

<https://docs.microsoft.com/en-us/powershell/module/activedirectory/remove-adgroupmember>

Remove-ADUser

- Consider if you have Active Directory Recycle Bin enabled
 - [Introduction to Active Directory Administrative Center Enhancements \(Level 100\)](#)
- Consider running manually and only automating if the organization has agreed to the filtering limits you've defined in your script.



<https://docs.microsoft.com/en-us/powershell/module/activedirectory/remove-aduser>

-- The End --

Ask Questions ...

... then go forth and iterate.

PAST PRESENTATIONS:

<https://github.com/chrisATautomatemystuff/Presentations>

