PowerShell – It's Pretty Cool

Name: Chris Thomas
Title: Desktop Engineer
Organization: Ingham ISD
Email: cthomas@inghamisd.org
Twitter: @AutomateMyStuff

Ingham Intermediate School District
A Regional Educational Service Agency

https://github.com/chrisATautomatemystuff/Presentations

PowerShell – It's Pretty Cool

This session is intended to be a beginner to intermediate level session for those with little working knowledge of PowerShell. You should expect to learn how to utilize PowerShell to save time and effort while maintaining consistency.

We will work through PowerShell fundamentals, importing and exporting objects, working with variables, manipulating objects, creating custom objects, flow control and loops, input checking and error handling, creating and manipulating Active Directory objects, managing server administration tasks, and some basics of Office 365 administration tasks.

We will be taking regular participation breaks to practice concepts, but feel free to ask questions when you become curious about something.

GREEN = Link

Link = Something I found of value

Agenda

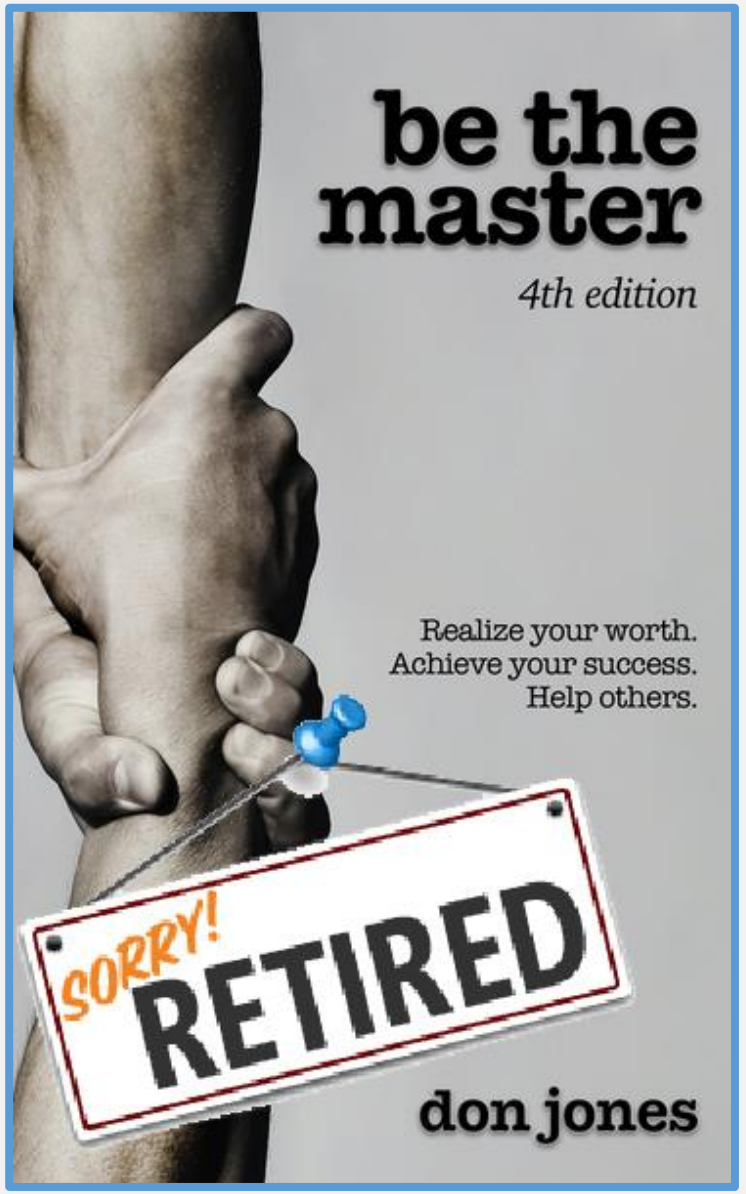
- 09:00 – 09:15 Intro and Expectations
- 09:15 – 09:30 PowerShell Fundamentals
- 09:30 – 09:35 Exercises
- 09:35 – 09:45 PowerShell Core Cmdlets
- 09:45 – 09:50 Exercises
- 09:50 – 10:00 PowerShell Working With Variables
- 10:00 – 10:05 Exercises
- 10:05 – 10:15 PowerShell Working With Data
- 10:15 – 10:20 Exercises
- 10:20 – 10:30 10 Minute Bio/Stretch Break
- 10:30 – 10:45 PowerShell Flow Control
- 10:45 – 10:50 Exercises
- 10:50 – 11:00 Object Manipulation
- 11:00 – 11:05 Exercises
- 11:05 – 11:20 Custom Objects
- 11:20 – 11:25 Exercises
- 11:25 – 11:35 Custom Functions
- 11:35 – 11:40 Exercises
- 12:00 – 12:30 Lunch & Bio/Stretch Break
- 12:30 – 01:30 Active Directory User & Group Management
- 01:30 – 01:40 Exercises
- 01:40 – 01:45 10 Minute Bio/Stretch Break
- 01:45 – 01:55 Azure AD / Office 365 Management
- 01:55 – 02:00 Exercises
- 02:00 – 02:30 PowerShell App Deployment Toolkit
- 02:30 – 02:35 Exercises
- 02:35 – 03:00 Questions/Concerns/Wrap Up

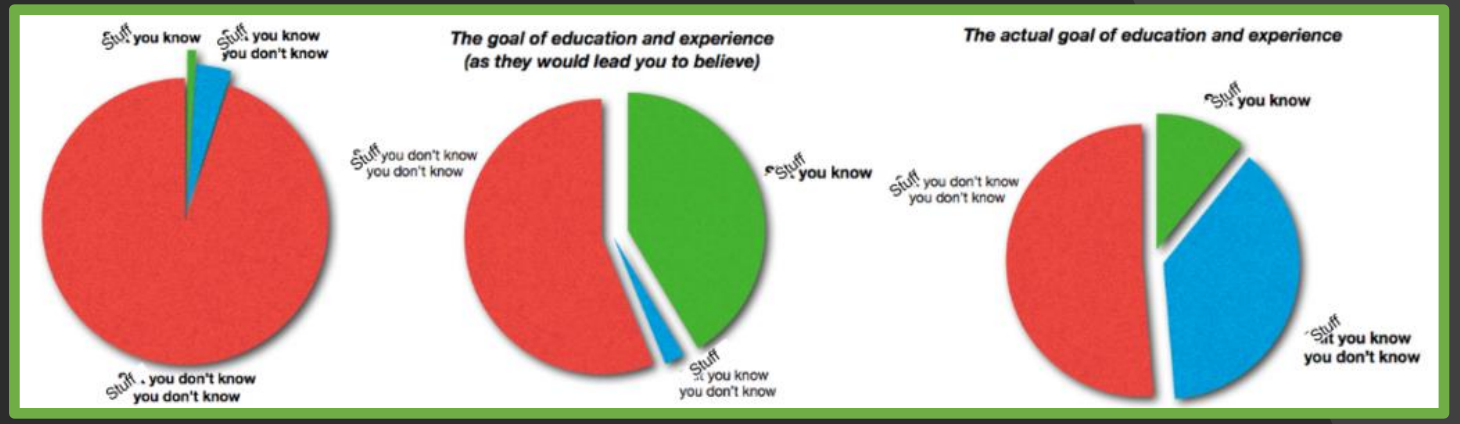
Who Am I?

- 22+ 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.”





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.

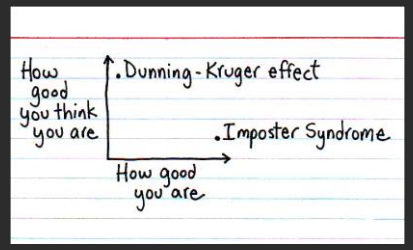
> **Imposter syndrome isn't just a woman thing. It's a human thing.**

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:

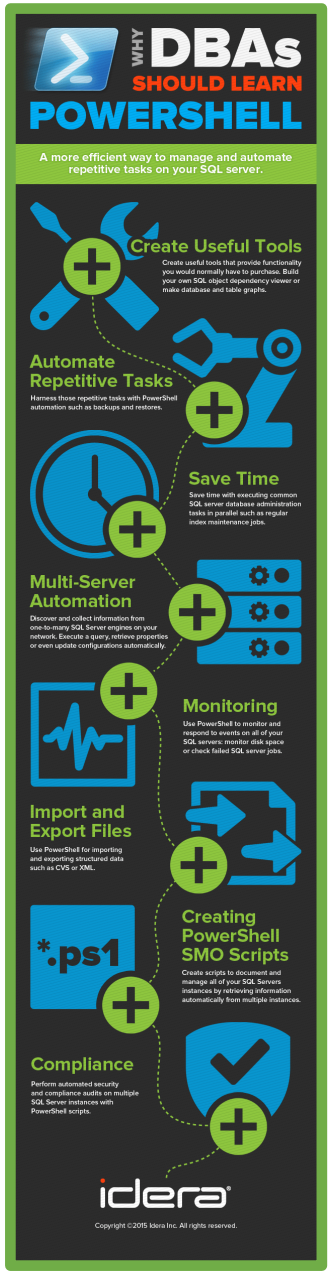
- **Focus On Learning**: Forget appearing awesome. You can get better if you try, so focus on *that*.
- **"Good Enough" Goals**: Stop trying to be prefect. (Yes, that was a typo. I'm not fixing it. It's good enough.)
- **Take Off The Mask**: Talk to someone you think is facing the same issue. You're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

Why PowerShell?

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



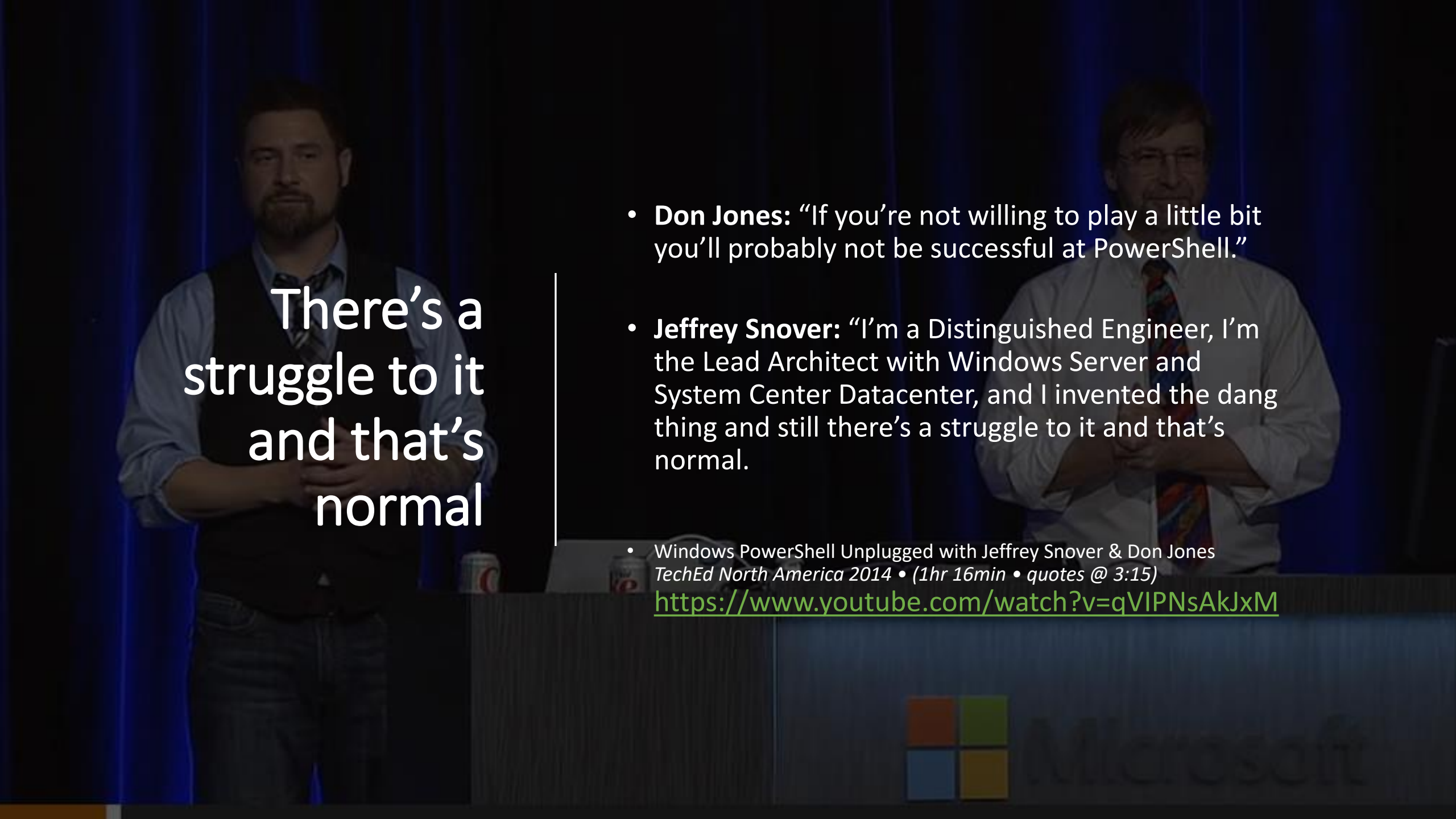
Why PowerShell?

- 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?

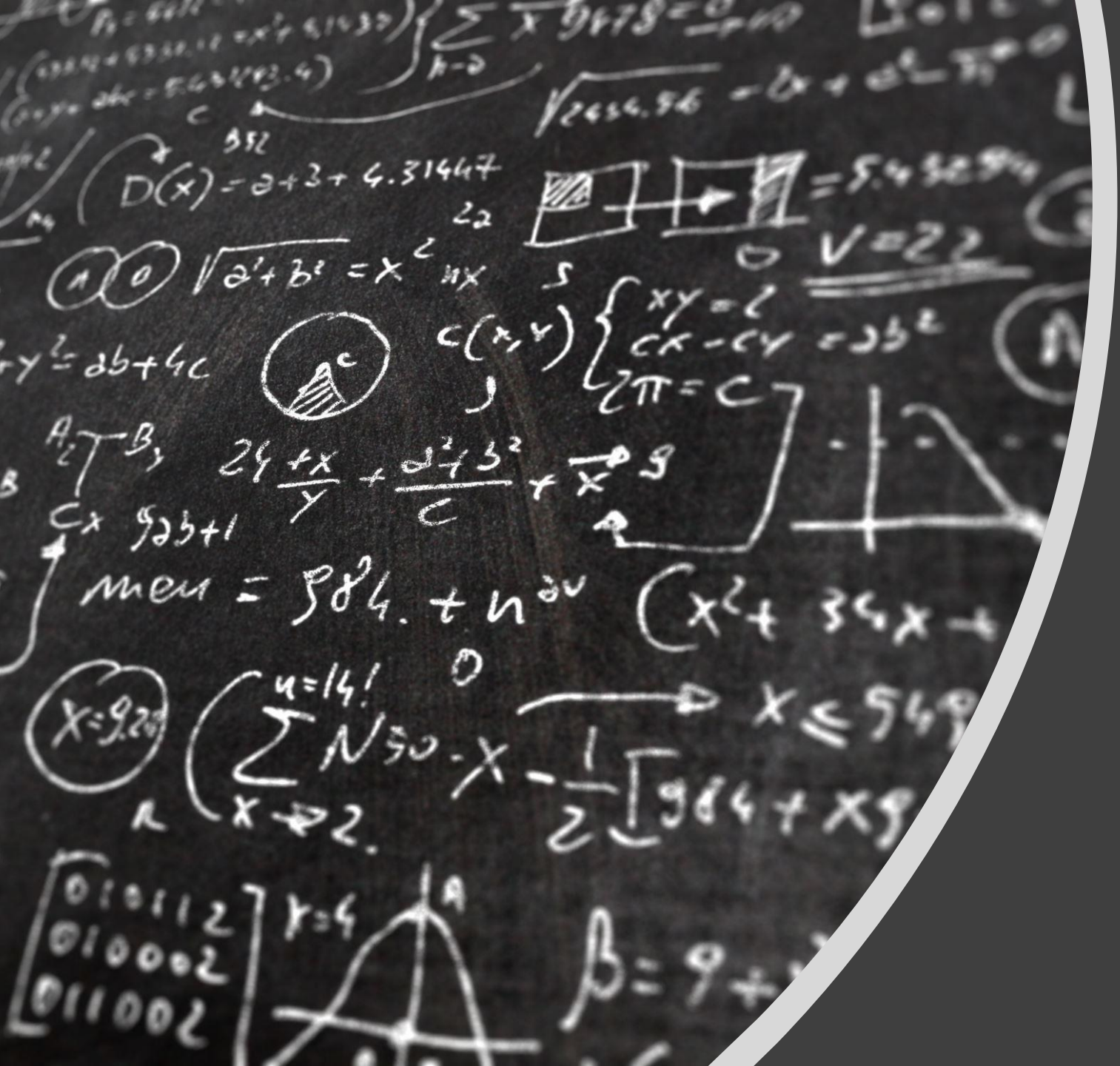
- Active Directory (AD)
- Group Policy (GPO/GPP)
- Office 365
- SharePoint Server
- Microsoft Teams
- SQL Server
- Best Practice Analyzer (BPA)
- RESTful/SOAP API Calls
- Internet Information Services (IIS)
- Remote Desktop Services (RDS)
- System Center Suite
  - Configuration Manager (SCCM)
  - Operations Manager (SCOM)
  - Orchestrator (SCORCH)
  - Service Manager (SCSM)
  - Virtual Machine Manager (SCVMM)
  - Data Protection Manager (SCDPM)
- File System
- Registry
- WMI/CIM
- Event Viewer
- AppLocker



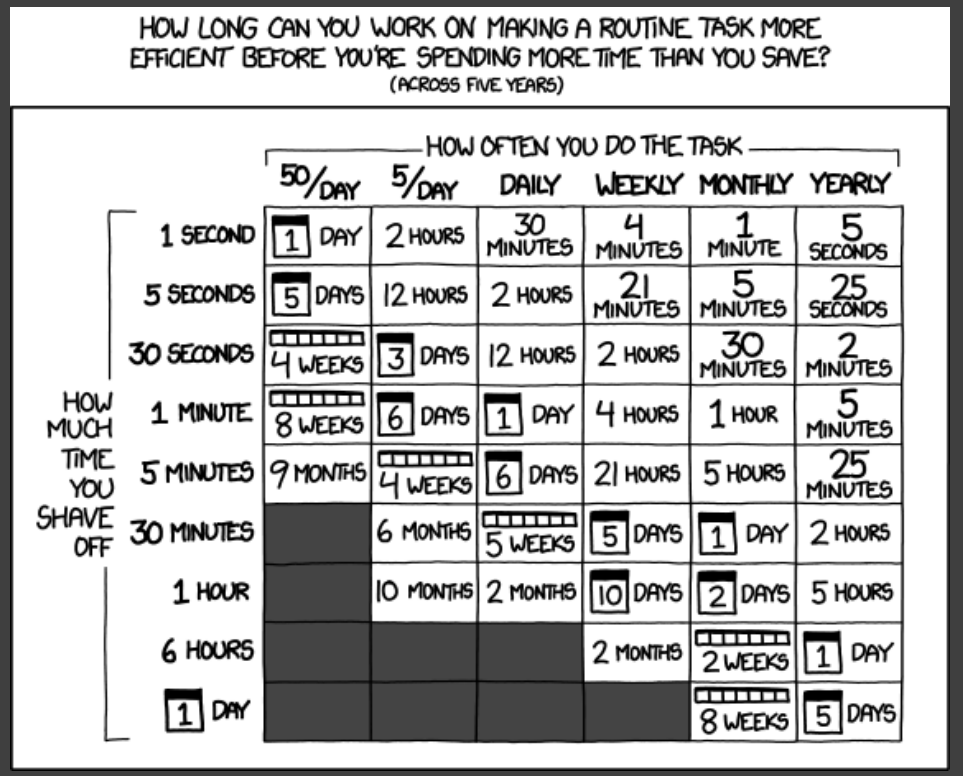
There's a struggle to it and that's normal

- **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



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



HOW LONG CAN YOU WORK ON MAKING A ROUTINE TASK MORE EFFICIENT BEFORE YOU'RE SPENDING MORE TIME THAN YOU SAVE?
(ACROSS FIVE YEARS)

HOW MUCH TIME YOU SHAVE OFF	HOW OFTEN YOU DO THE TASK: 50/DAY	5/DAY	DAILY	WEEKLY	MONTHLY	YEARLY
1 SECOND	1 DAY	2 HOURS	30 MINUTES	4 MINUTES	1 MINUTE	5 SECONDS
5 SECONDS	5 DAYS	12 HOURS	2 HOURS	21 MINUTES	5 MINUTES	25 SECONDS
30 SECONDS	4 WEEKS	3 DAYS	12 HOURS	2 HOURS	30 MINUTES	2 MINUTES
1 MINUTE	8 WEEKS	6 DAYS	1 DAY	4 HOURS	1 HOUR	5 MINUTES
5 MINUTES	9 MONTHS	4 WEEKS	6 DAYS	21 HOURS	5 HOURS	25 MINUTES
30 MINUTES		6 MONTHS	5 WEEKS	5 DAYS	1 DAY	2 HOURS
1 HOUR		10 MONTHS	2 MONTHS	10 DAYS	2 DAYS	5 HOURS
6 HOURS				2 MONTHS	2 WEEKS	1 DAY
1 DAY					8 WEEKS	5 DAYS

How It All Began…

Secretary / Registrar Enrolls Student In SIS → Secretary / Registrar Fills Out Google Form For Student Account Creation Process → Google Form Sends Email To Help Desk Which Generates A Ticket For That One Student

Help Desk Staff Manually Creates Student Account (Active Directory, Google, Office365) → Help Desk Staff Manually Adds Student Account To Appropriate Groups And Distribution Lists → Help Desk Staff Manually License Student Account

Help Desk Staff Manually Update Student Credentials In SIS For Staff Reference → Help Desk Staff Close Ticket And Inform Staff


```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
#$Students = Import-Csv C:\scripts\accounts09162015.csv
#endregion

foreach ($student in $students){

#region Student Variable Definitions
    $FN = $Student.FN
    $LN = $Student.LN
    $NAME = $FN + " " + $LN
    $YOG = $Student.YOG
    $SAM = $Student.SN
    $TEMPPWD = (ConvertTo-SecureString -AsPlainText "IWishYouHad8DigitPasswords" -Force)
    $INSECUREPWD = $user.PWD
    $SECUREPWD = (ConvertTo-SecureString -AsPlainText $INSECUREPWD -Force)
    $UPN = $SAM + "@"
    $EMAIL = $SAM + "@"
    $PATH = "OU=" + $YOG + ",OU=StudentAccounts,OU=UserAccounts,DC="
    $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
}
```

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

Objects The Bicycle Example

Property

- *SOMETHING ABOUT THE OBJECT*
- **Wheel Size**
- **Tire Pressure**
- **Height**
- **Gears**
- **Color**
- **Speed**
- **Price**

Method

- *Something you can do with or to the object*
- **Pedal**
- **Brake**
- **Change Gear**
- **Repair**
- **Re-inflate Tire**

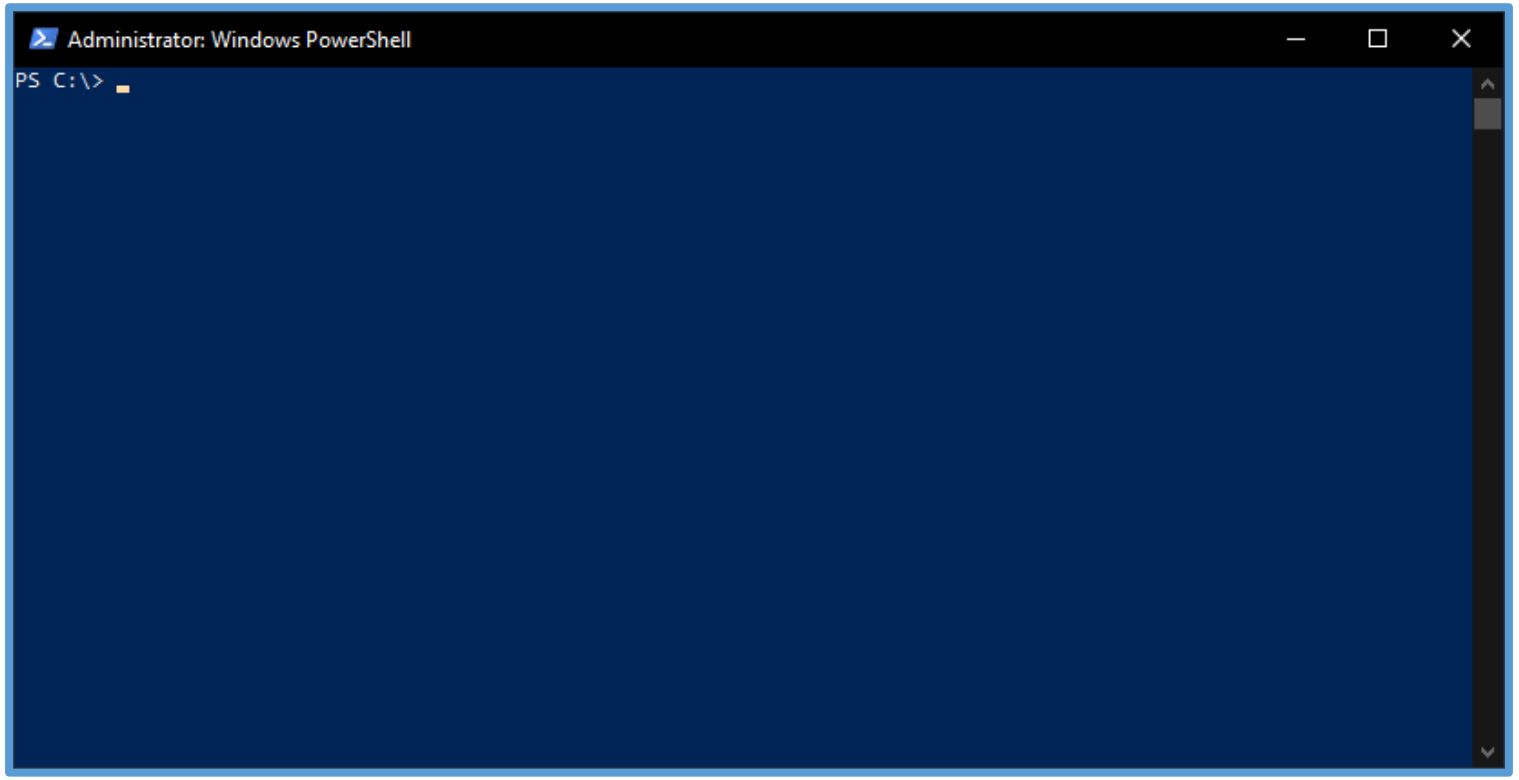
Vocabulary 02/02

- 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

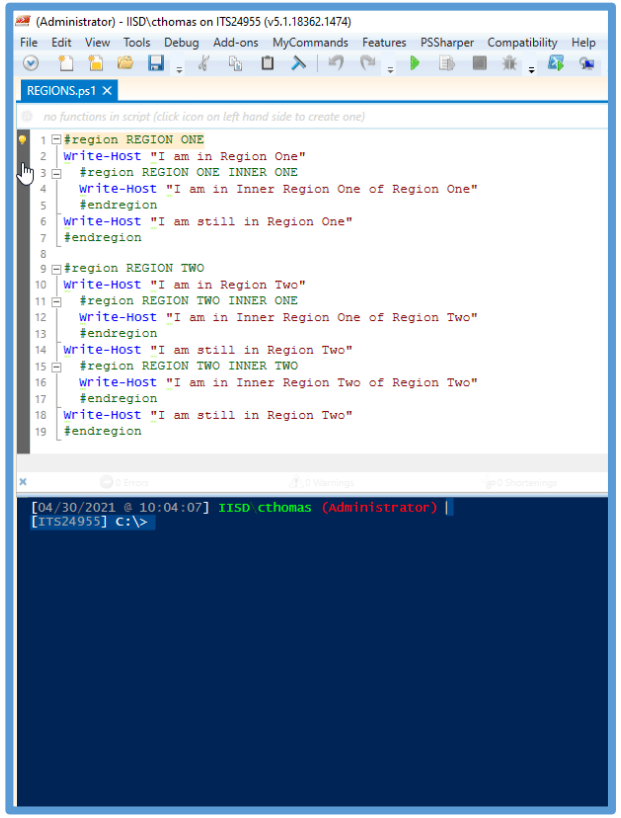
- PS C:\> new-adu<TAB>
- PS C:\> New-ADUser -<TAB>
  - <TAB> to advance through parameters
    - or
  - <SHIFT+TAB> to go back to a previous parameter
- PS C:\> New-ADUser -<CTRL + SPACE>
  - <ARROW KEYS> + <SPACE> to select appropriate parameter



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



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

```powershell
Import-Module ActiveDirectory

#region Import File Locations
$students = Import-Csv C:\scripts\students_import.csv
#endregion

foreach ($student in $students){

#region Student Variable Definitions
    $FN = $student.FN
    $LN = $student.LN
    $NAME = $FN + " " + $LN
    $YOG = $student.YOG
    $SAM = $student.SN
    $TEMPPWD = (ConvertTo-SecureString -AsPlainText "IWishYouH
    $INSECUREPWD = $student.PWD
    $SECUREPWD = (ConvertTo-SecureString -AsPlainText $INSECUR
    $UPN = $SAM + "@<DOMAIN NAME>"
    $EMAIL = $SAM + "@<DOMAIN NAME>"
    $PATH = "OU=" + $YOG + ",OU=StudentAccounts,OU=UserAccount
    $GROUP = "U_All_" + $YOG + "Students"
#endregion

#region Active Directory Student Provisioning
    New-ADUser -Name $NAME -DisplayName $NAME -GivenName $FN -

    Add-ADGroupMember -Identity $GROUP -Members $SAM

    Set-ADAccountPassword -Identity $SAM -Reset -NewPassword $
#endregion
}
```

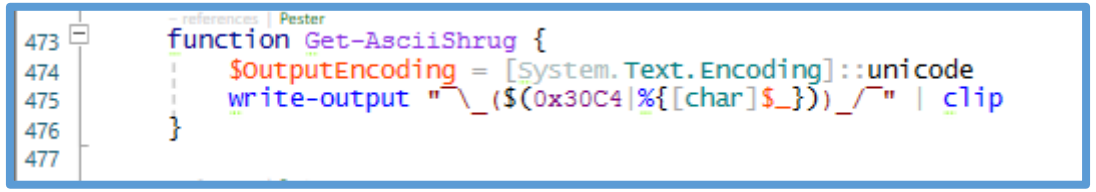
```powershell
Import-Module ActiveDirectory

#region Import File Locations
$students = Import-Csv C:\scripts\students_import.csv
#endregion

foreach ($student in $students){

#region Student Variable Definitions
    $FN = $student.FN
    $LN = $student.LN
    $NAME = $FN + " " + $LN
    $YOG = $student.YOG
    $SAM = $student.SN
    $TEMPPWD = (ConvertTo-SecureString -AsPlainText "IWishYouHad8DigitPasswords" -Force)
    $INSECUREPWD = $student.PWD
    $SECUREPWD = (ConvertTo-SecureString -AsPlainText $INSECUREPWD -Force)
    $UPN = $SAM + "@<DOMAIN NAME>"
    $EMAIL = $SAM + "@<DOMAIN NAME>"
    $PATH = "OU=" + $YOG + ",OU=StudentAccounts,OU=UserAccounts,DC=<DOMAIN NAME>,DC=<DOMAIN TYPE>"
    $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
}
```

```powershell
Import-Module ActiveDirectory

#region Import File Locations
$students = Import-Csv C:\scripts\students_import.csv
#endregion

foreach ($student in $students){

#region Student Variable Definitions
    $FN = $student.FN
    $LN = $student.LN
    $NAME = $FN + " " + $LN
    $YOG = $student.YOG
    $SAM = $student.SN
    $TEMPPWD = (ConvertTo-SecureString -AsPlainText "IWishYouHad8DigitPasswords" -Force)
    $INSECUREPWD = $student.PWD
    $SECUREPWD = (ConvertTo-SecureString -AsPlainText $INSECUREPWD -Force)
    $UPN = $SAM + "@<DOMAIN NAME>"
    $EMAIL = $SAM + "@<DOMAIN NAME>"
    $PATH = "OU=" + $YOG + ",OU=StudentAccounts,OU=UserAccounts,DC=<DOMAIN NAME>,DC=<DOMAIN TYPE>"
    $GROUP = "U_All_" + $YOG + "Students"
#endregion

$newUserSplat = @{
  Name = $NAME
  DisplayName = $NAME
  GivenName = $FN
  Surname = $LN
  SamAccountName = $SAM
  UserPrincipalName = $UPN
  EmailAddress = $EMAIL
  AccountPassword = $TEMPPWD
  CannotChangePassword = $true
  PasswordNeverExpires = $true
  Path = $PATH
  PassThru = $null
}

#region Active Directory Student Provisioning
    New-ADUser @newUserSplat | Enable-ADAccount

    Add-ADGroupMember -Identity $GROUP -Members $SAM

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

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

PowerShell Profile

- If you don't have a $profile, create one and get going!
  - if (! (Test-Path $Profile)) { New-Item -Type File -Path $Profile -Force }
  - powershell_ise $profile



- 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."

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

PRACTICE TIME

PowerShell Core Cmdlets

Update-Help

- PS C:\> Update-Help

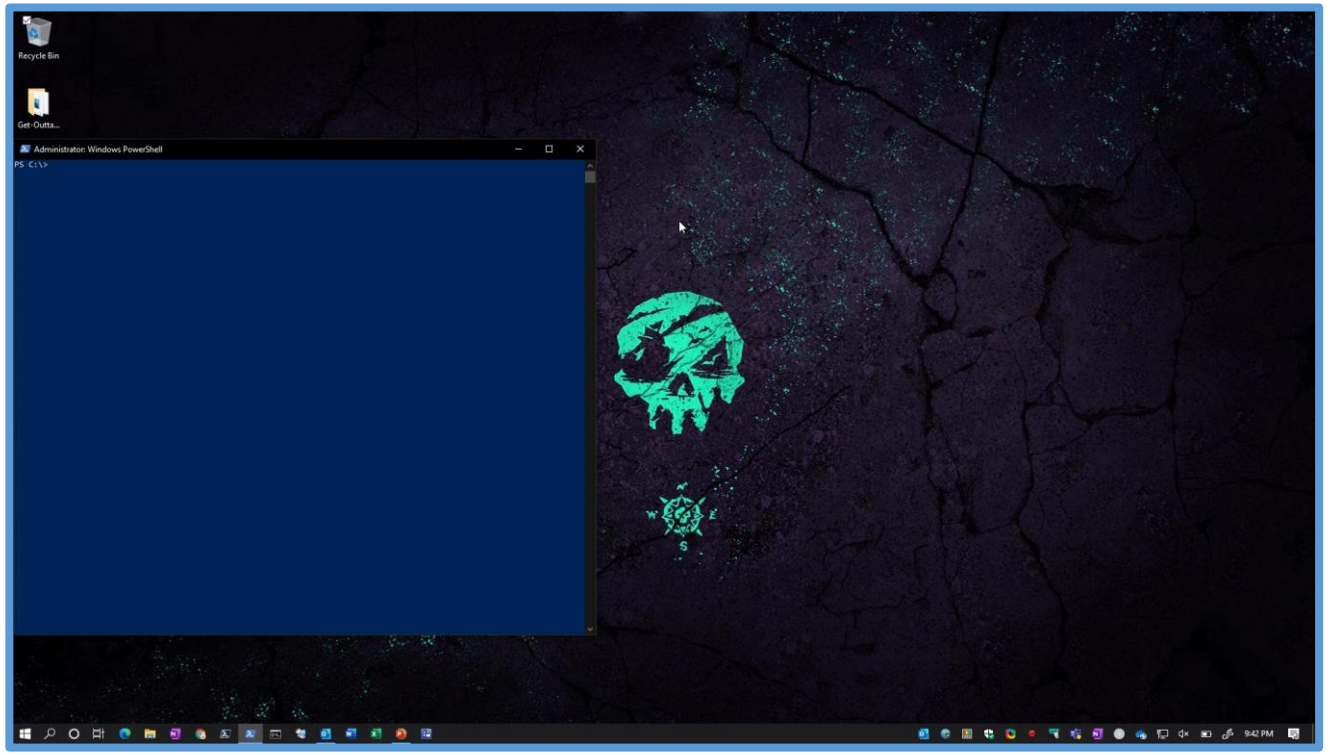
- Update PowerShell with the Update-Help cmdlet
  - “It usually isn't advisable to update help on a single module though. After all, if you're going to update help for your modules, you might as well just go ahead and check for the latest help for all of your modules. Simply running the Update-Help command by itself will tell the command to search for updated help for all of your modules.”
  - “By default, you can only run Update-Help once per day. It downloads only new versions of help and does not download content larger than 1 GB.”

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/update-help

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

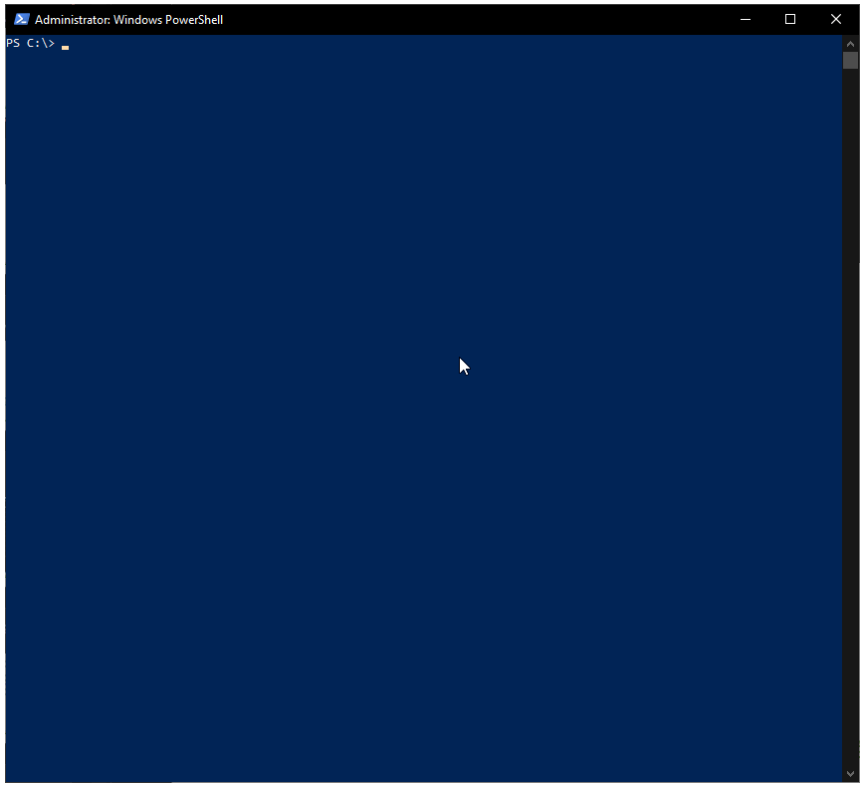
- PS C:\> Get-Command -Module ActiveDirectory
- PS C:\> Get-Command -Verb New
- PS C:\> Get-Command -Verb New -Noun AD*
- PS C:\> Get-Command New-AD*
- PS C:\> Get-Command New-ADUser

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

Get-Module

- PS C:\> Get-Module
- PS C:\> Get-Module -ListAvailable

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

PRACTICE TIME

PowerShell Working With Variables

New-Variable

SYNTAX:

New-Variable –Name <NAME>

New-Variable –Name <NAME> -Value <VALUE>

EXAMPLE:

New-Variable –Name “whateverYouWantItToBeButILikeCamelCase”

New-Variable –Name “verboseVariableName” –Value “because I can”

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/new-variable

Get-Variable

SYNTAX:

Get-Variable –Name <NAME>

Get-Variable –Name <NAME> -ValueOnly

EXAMPLE:

Get-Variable –Name ver*

Get-Variable –Name ver* -ValueOnly

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/get-variable

Set-Variable

SYNTAX:

Get-Variable –Name <NAME>

Get-Variable –Name <NAME> -Value <VALUE>

EXAMPLE:

Set-Variable –Name “verboseVariableName2” –Value “wait, this doesn’t exist yet”

Set-Variable –Name “verboseVariableName3”

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/set-variable

Clear-Variable

SYNTAX:

Clear-Variable –Name <NAME>

EXAMPLE:

Clear-Variable –Name “verboseVariableName”

Clear-Variable –Name “verboseVar*”

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/clear-variable

Remove-Variable

SYNTAX:

Remove-Variable –Name <NAME>

EXAMPLE:

Remove-Variable –Name “verboseVariableName”

Remove-Variable –Name “verboseVar*”

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/remove-variable

Identify or Cast Variable Type

EXAMPLE:

- $int = “12”
- $int | Get-Member
- $int2 = 12
- $int2 | Get-Member
- [string]$int3 = 12
- $int3 | Get-Member

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_variables#types-of-variables

PRACTICE TIME

PowerShell Working With Data

Import-CSV

SYNTAX:

Import-Csv –Path $insertFilePath

EXAMPLE:

$students = Import-Csv -Path $insertFilePath

```
1  $insertFilePath = "[redacted]idm_students_insert.csv"
2  $content = get-content -Path $insertFilePath
3
4  if($content -ne $null)
5  {
```

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

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

Get-Content

SNYTAX:

Get-Content –Path $filepath

EXAMPLE:

If ((Get-Content –Path $insertFilePath) -ne $null) {$students = Import-Csv -Path $insertFilePath}

```
1  $insertFilePath = "                idm_students_insert.csv"
2  $content = get-content -Path $insertFilePath
3
4  if($content -ne $null)
5  {
```

- 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.”

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.management/get-content

Out-File

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
```

- 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."

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/out-file

PRACTICE TIME

BIO/STRETCH BREAK

PowerShell Flow Control

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
`$() @() () @{}`	about_Operators
`. ?.` (member access)	about_Operators
`::` (static)	about_Operators
`[0] ?[0]` (index operator)	about_Operators
`[int]` (cast operators)	about_Operators
`-split` (unary)	about_Split
`-join` (unary)	about_Join
`,` (comma operator)	about_Operators
`++ --`	about_Assignment_Operators
`! -not`	about_Logical_Operators
`..` (range operator)	about_Operators
`-f` (format operator)	about_Operators
`-` (unary/negative)	about_Arithmetic_Operators
`* / %`	about_Arithmetic_Operators
`+ -`	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
`-split` (binary)	about_Split
`-join` (binary)	about_Join
`-is -isnot -as`	about_Type_Operators
`-eq -ne -gt -gt -lt -le`	about_Comparison_Operators
`-like -notlike`	about_Comparison_Operators
`-match -notmatch`	about_Comparison_Operators
`-in -notIn`	about_Comparison_Operators
`-contains -notContains`	about_Comparison_Operators
`-replace`	about_Comparison_Operators

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

OPERATOR	REFERENCE
`-band -bnot -bor -bxor -shr -shl`	about_Arithmetic_Operators
`-and -or -xor`	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
-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

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

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 }

```
#SET THE MIDDLE NAME AND INITIALS IF THE USER HAS THEM
if($middleName -ne $null -AND $middleName -ne '')
{
  Set-ADUser -Identity $samAccountName -OtherName $middleName -Initials $middleInitial
}
else
{
  Set-ADUser -Identity $samAccountName -Clear MiddleName,Initials
}

#ADD THE USER TO THEIR GROUPS
if($groupList -ne $null)
{
  $groupList = $groupList | ForEach-Object {Get-ADGroup -Identity $_}
  $groupList | ForEach-Object {Add-ADGroupMember -Identity $_ -Members $samAccountName}
}
```

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 }
}
```

```powershell
#DEFINE THE STATE BUILDING CODE FOR THE USER
$buildingCode = $student.idm_building01code

#DEFINE BUILDING VARIABLES BASED ON STATE BUILDING CODES
switch ($buildingCode)
{
  '3      0' {
               $buildingShortName = '
               $office = '
               $streetAddress = '
               $city = '
               $postalCode = '
               $officePhone = '

               $passwordInsecure = $student.idm_default_password
               $passwordSecure = (ConvertTo-SecureString -AsPlainText $passwordInsecure -Force)
  }
  '3      2' {
               $buildingShortName = '
               $office = '
               $streetAddress = '
               $city = '
               $postalCode = '
               $officePhone = '

               $passwordInsecure = $student.idm_default_password
               $passwordSecure = (ConvertTo-SecureString -AsPlainText $passwordInsecure -Force)
  }
  '3      3' {
               $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’ }

- PS C:\> <*cmdlet*> | Where-Object –FilterScript { $_.*PropertyName* -like “**query**” }
- PS C:\> Get-WmiObject -Class win32_product | Where-Object -FilterScript {$_.InstallDate -eq 20150922} | Select-Object -Property Name, IdentifyingNumber
  PS C:\> gwmi win32_product | ?{$_.InstallDate –eq 20150922} | select name,identifyingnumber

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 $_ }

PRACTICE TIME

PowerShell
Object Manipulation

Compare-Object

SYNTAX: Compare-Object –ReferenceObject <OBJECT> -DifferenceObject <OBJECT>

- PS C:\> Compare-Object –ReferenceObject $(Get-Content *<file1path>*) -DifferenceObject $(Get-Content *<file2path>*)
- PS C:\> Compare-Object –ReferenceObject $(Get-Content *<file1path>*) -DifferenceObject $(Get-Content *<file2path>*) -IncludeEqual
- PS C:\> $processesBefore = Get-Process
  PS C:\> notepad
  PS C:\> $processesAfter = Get-Process
  PS C:\> Compare-Object -ReferenceObject $processesBefore -DifferenceObject $processesAfter
- PS C:\> Compare-Object –ReferenceObject $processesAfter –DifferenceObject $processesBefore

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/compare-object

Group-Object

SYNTAX: Group-Object –Property <PROPERTYNAME>

- PS C:\> *<cmdlet>* | Group-Object -Property *<propertyName>*
- PS C:\> *<cmdlet>* | Group-Object -Descending
- PS C:\> $events = Get-EventLog -LogName System -Newest 1000
  PS C:\> $events | Group-Object – Property EventID

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/group-object

Measure-Object

SYNTAX: Measure-Object

- PS C:\> *<cmdlet>* | Measure-Object
- PS C:\> Get-ADUser –Filter * | Measure-Object
- PS C:\> Get-MsolUser –All | Measure-Object

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/measure-object

Select-Object

SYNTAX: Select-Object –Property <PROPERTYNAME>

Select-Object –Unique

Select-Object –First <INT>

- PS C:\> <*cmdlet*> | Select-Object -Property <*propertyList*>
- PS C:\> “a”, “b”, “c”, “b”, “c” | Measure-Object
  PS C:\> “a”, “b”, “c”, “b”, “c” | Select-Object -Unique | Measure-Object
- PS C:\> Get-ADUser -Filter * | Select-Object -First 5
- PS C:\> $users = Import-CSV -Path C:\scripts\users.csv | Select-Object -Property UPN, SN, FN, LN, OU

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/select-object

Sort-Object

SYNTAX: Sort-Object

Sort-Object –Property <PROPERTYNAME>

Sort-Object -Property <PROPERTYNAME> -Top <INT>

Sort-Object –Property <PROPERTYNAME> -Bottom <INT>

- PS C:\> *<cmdlet>* | Sort-Object
- PS C:\> *<cmdlet>* | Sort-Object -Unique
- PS C:\> Get-ChildItem | Sort-Object
- PS C:\> Get-MsolUser -All | Sort-Object -Property UserPrincipalName

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/sort-object

Tee-Object

SYNTAX: Tee-Object –FilePath <FILEPATH>

Tee-Object –Variable <VARIABLENAME>

- PS C:\> *<cmdlet>* | Tee-Object -FilePath *<filepath>*
- PS C:\> *<cmdlet>* | Tee-Object -FilePath
- PS C:\> Get-ChildItem -Path $env:USERPROFILE -Recurse | Tee-Object -FilePath C:\scripts\allappdatafiles.txt | Where-Object -FilterScript {$_.Name -like "*.exe"} | Out-File -FilePath C:\scripts\exeappdatafiles.txt

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/tee-object

PRACTICE TIME

PowerShell Custom Objects

New-Object

SYNTAX:

New-Object –TypeName <SOME OBJECT TYPE>

EXAMPLE:

$object = New-Object –TypeName PSObject

$object2 = New-Object –TypeName PSObject –Property @{OS = “Windows”}

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

```
$object = New-Object -TypeNamePSObject
$object | Add-Member -MemberTypeNoteProperty -Name OSBuild -Value $os.BuildNumber
$object | Add-Member -MemberTypeNoteProperty -Name OSVersion -Value $os.Version
$object | Add-Member -MemberTypeNoteProperty -Name BIOSSerial -Value $bios.SerialNumber
Write-Output $object
```

- Windows PowerShell: The Many Ways to a Custom Object
  - “However you choose to create your custom objects, you can create custom objects instead of outputting mere text to the console window. Objects are infinitely more flexible, and they let your script or function output integrate with everything else Windows PowerShell.”
- Browsing in Internet Explorer via PowerShell

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.utility/new-object

Add-Member

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

$array += $object

https://docs.microsoft.com/en-us/powershell/module/Microsoft.PowerShell.Utility/Add-Member

Hashtable or Array

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
```

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

[PSCustomObject]

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
}
```

- 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.”

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

PRACTICE TIME

PowerShell
Custom Functions

Functions

SYNTAX:

function Verb-Noun { DO SOMETHING }

- 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.”

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

PRACTICE TIME

LUNCH & BIO/STRETCH BREAK

Automation End Goal

- Provision
  - User account creation
- Access
  - User account group membership
- Consent
  - OAuth consent controls (O365 and Google)
- Entitle
  - Automated license management
- Federate
  - ‘Sign in with Google’ or ‘Sign in with Microsoft’ vs. ‘Sign in with email address’
- Personalize / Maintain
  - Name changes
  - Password self-service
  - User photos
- De-Provision
  - User account disabling
  - User account access revocation
  - User account deletion
- Sync
  - Azure AD Connect
  - Google Cloud Directory Sync

Active Directory User & Group Management

Provision

ADUC/ADAC

- Manual
- Does not scale
- Potential for human error / typos
- Potential to forget steps
- Potential to lose processes over time / staff changes

Get-ADUser

SYNTAX: Get-ADUser –Filter <* or QUERY>

Get-ADUser –Filter {enabled –eq $true}

Get-ADUser –Identity <USER>

https://docs.microsoft.com/en-us/powershell/module/activedirectory/get-aduser

New-ADUser

SYNTAX: New-ADUser –Name <NAME>

New-ADUser –Name <NAME> -Title <TITLE> -AccountPassword <SECURESTRING PASSWORD>

New-ADUser @newADUserSplat

```
#DEFINE ALL THE NEW USER ATTRIBUTES FOR SPLATTING
$newUserSplat = @{
  Name = $fullName
  DisplayName = $displayName
  GivenName = $firstName
  Surname = $lastName
  SamAccountNAme = $samAccountName
  UserPrincipalName = $userPrincipalName
  EmailAddress = $emailAddress
  AccountPassword = $passwordTemporary
  ChangePasswordAtLogon = $false
  CannotChangePassword = $true
  PasswordNeverExpires = $true
  Path = $ouPath
  StreetAddress = $streetAddress
  City = $city
  State = $state
  PostalCode = $postalCode
  Organization = $organization
  Company = $company
  Office = $office
  OfficePhone = $officePhone
  Department = $department
  Title = $title
  Description = $description
  EmployeeID = $employeeID
  EmployeeNumber = $employeeNumber
  Enabled = $true
}

#CREATE THE USER BASED ON SPLAT
New-ADUser @newUserSplat
Set-ADUser $samAccountName -Add @{                    !=$true}
```

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

New-ADUser Multiple Users

Let's pull everything together…

```powershell
$insertFilePath = "[redacted]idm_students_insert.csv"
$content = get-content -Path $insertFilePath

if($content -ne $null)
{
```

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

```powershell
    foreach ($student in $students)
    {
      #region VARIABLE DEFINITIONS
      #DEFINE ALL THE THINGS FROM EXPORT FILE
      $firstName = $student.idm_first_name
      $firstInitial = $firstName.Substring(0,1)
      $middleName = $student.idm_middle_name
      if($middleName -ne $null -AND $middleName -ne '')
      {
        $middleInitial = $middleName.Substring(0,1)
      }
      $lastName = $student.idm_last_name
      $lastInitial = $lastName.Substring(0,1)

      $fullName = $firstName + ' ' + $lastName
      $displayName = $firstName + ' ' + $lastName

      $samAccountName = $student.idm_samaccountname
      #$userPrincipalName = $student.idm_upn
      $userPrincipalName = $samAccountName + "@students.[redacted].org"
      $emailAddress = $student.idm_email

      $title = 'Student'

      $gradeLevel = $student.idm_student_grade_level
      $yearofGrad = $student.idm_student_graduation_year

      $employeeID = $student.idm_employeeid
      $employeeNumber = $student.idm_employeenumber
      $employeeNumberLength = $employeeNumber.Length

      $studentID = $employeeNumber.Substring(3)
      $studentIDLast4 = -join "$studentID"[-4..-1]
```

```powershell
      #DEFINE THE STATE BUILDING CODE FOR THE USER
      $buildingCode = $student.idm_building01code

      #DEFINE BUILDING VARIABLES BASED ON STATE BUILDING CODES
      switch ($buildingCode)
      {
        '3[redacted]0' {
                        $buildingShortName = '[redacted]'
                        $office = '[redacted]'
                        $streetAddress = '[redacted]'
                        $city = '[redacted]'
                        $postalCode = '[redacted]'
                        $officePhone = '[redacted]'
                        [redacted]
                        $passwordInsecure = $student.idm_default_password
                        $passwordSecure = (ConvertTo-SecureString -AsPlainText $passwordInsecure -Force)
        }
        '3[redacted]2' {
                        $buildingShortName = '[redacted]'
                        $office = '[redacted]'
                        $streetAddress = '[redacted]'
                        $city = '[redacted]'
                        $postalCode = '[redacted]'
                        $officePhone = '[redacted]'
                        [redacted]
                        $passwordInsecure = $student.idm_default_password
                        $passwordSecure = (ConvertTo-SecureString -AsPlainText $passwordInsecure -Force)
        }
        '3[redacted]3' {
                        $buildingShortName = '[redacted]'
```

```powershell
      #DEFINE OU PATH FOR THE SCHOOL
      $ouPath_append = ',DC=[redacted]'
      $ouPath = 'OU=Student,OU=' + $buildingShortName + $ouPath_append
      $ouPath_disabled = 'OU=Disabled Users,DC=[redacted]'

      #DEFINE THE USER DESCRIPTION
      $description = $buildingShortName + ' - Class of ' + $yearofGrad

      #DEFINE DISTRICT SPECIFIC VARIABLES
      $organization = 'Ingham Intermediate School District'
      $state = 'MI'
      $company = 'Ingham Intermediate School District'
      $domainName = '[redacted]'
      $domainAddress = '[redacted]'
```

```powershell
      #DEFINE ALL THE NEW USER ATTRIBUTES FOR SPLATTING
      $newUserSplat = @{
        Name = $fullName
        DisplayName = $displayName
        GivenName = $firstName
        Surname = $lastName
        SamAccountName = $samAccountName
        UserPrincipalName = $userPrincipalName
        EmailAddress = $emailAddress
        AccountPassword = $passwordTemporary
        ChangePasswordAtLogon = $false
        CannotChangePassword = $true
        PasswordNeverExpires = $true
        Path = $ouPath
        StreetAddress = $streetAddress
        City = $city
        State = $state
        PostalCode = $postalCode
        Organization = $organization
        Company = $company
        Office = $office
        OfficePhone = $officePhone
        Department = $department
        Title = $title
        Description = $description
        EmployeeID = $employeeID
        EmployeeNumber = $employeeNumber
        Enabled = $true
      }

      #CREATE THE USER BASED ON SPLAT
      New-ADUser @newUserSplat
      Set-ADUser $samAccountName -Add @{[redacted]=$true}
```

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

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

Access

Add-ADGroupMember

SYNTAX: Add-ADGroupMember –Identity <IDENTITY> -Members <MEMBERSAMACCOUNT>

```
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
```

https://docs.microsoft.com/en-us/powershell/module/activedirectory/add-adgroupmember

Managing Home Drives

- New-Item
- Get-Acl
- Set-Acl

```powershell
function New-HomeFolder ($homeDrivePath, $samAccountName)
{
    $domainName = "iisd"

    #CREATE HOME DIRECTORY
    New-Item -Path $homeDrivePath -ItemType Directory -Force

    #APPLY PERMISSIONS TO HOME FOLDER
    $identityReference = $domainName + '\' + $samAccountName
    $fileSystemAccessRights = [System.Security.AccessControl.FileSystemRights]::Modify
    $inheritanceFlags = [System.Security.AccessControl.InheritanceFlags]::ContainerInherit -bor [System.Security.AccessControl.InheritanceFlags]::ObjectInherit
    $propagationFlags = [System.Security.AccessControl.PropagationFlags]::None
    $accessControl = [System.Security.AccessControl.AccessControlType]::Allow
    $accessRuleSplat = $identityReference, $fileSystemAccessRights, $inheritanceFlags, $propagationFlags, $accessControl
    $accessRule = New-Object System.Security.AccessControl.FileSystemAccessRule $accessRuleSplat
    $homeDirectoryACL = Get-Acl $homeDrivePath
    $homeDirectoryACL.AddAccessRule($accessRule)
    Set-Acl -Path $homeDrivePath -AclObject $homeDirectoryACL
}
```

Consent

OAuth Consent – Microsoft

- Protecting your remote workforce from application-based attacks like consent phishing
- Managing consent to applications and evaluating consent requests
    - Script to list all delegated permissions and application permissions in Azure AD
- Disable all future user consent operations to any application
- Configure how end-users consent to applications

OAuth Consent - Google

- Take charge of your OAuth ecosystem with these best practices
- Control which third-party & internal apps access Google Workspace data

Entitle

Group-based Licensing - Microsoft

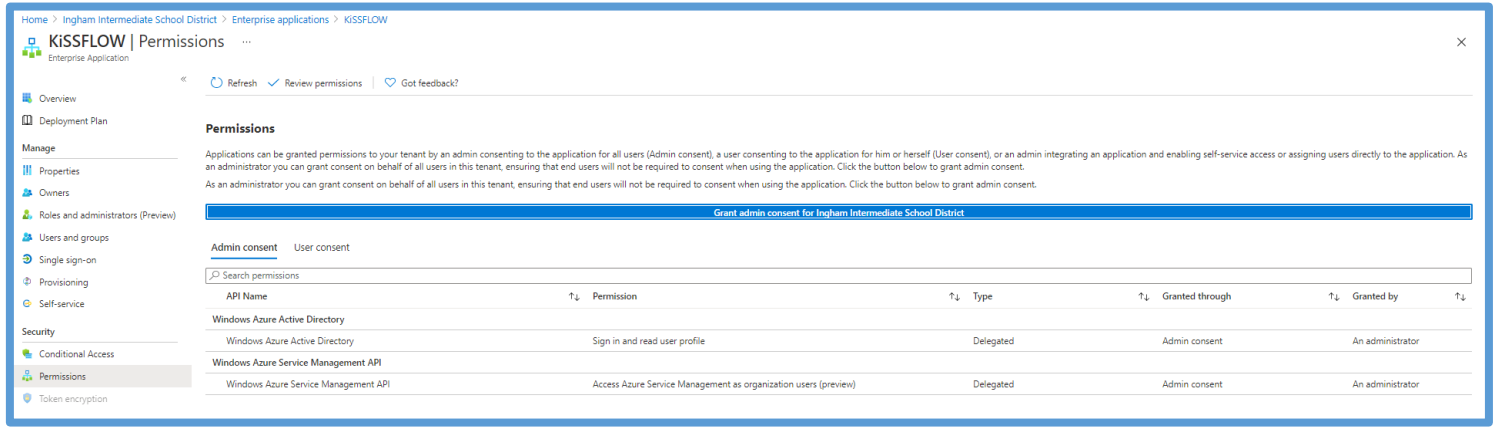
- Assign licenses to users by group membership in Azure Active Directory
- PowerShell and Graph examples for group-based licensing in Azure AD

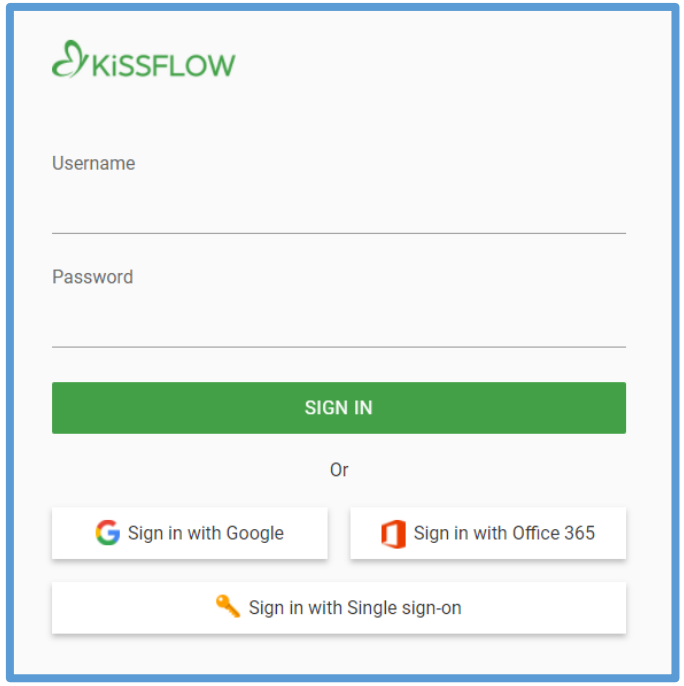
Group-based Licensing - Google

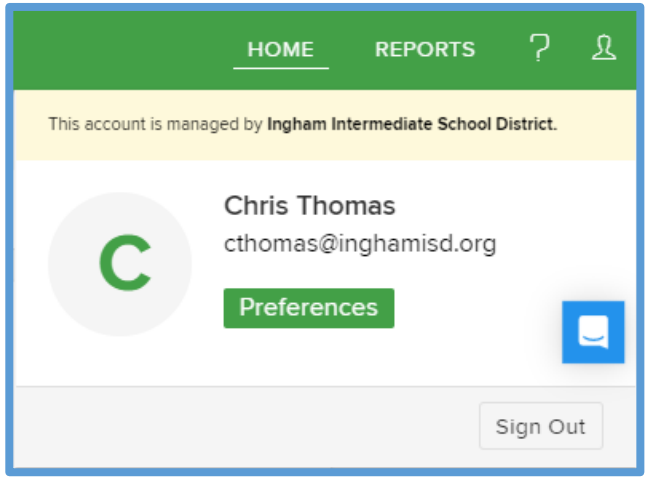
- How licensing works
  - About automatic licensing for organizational units
- GCDS: Manage & assign licenses

Federate

Sign-in with Microsoft / Office 365

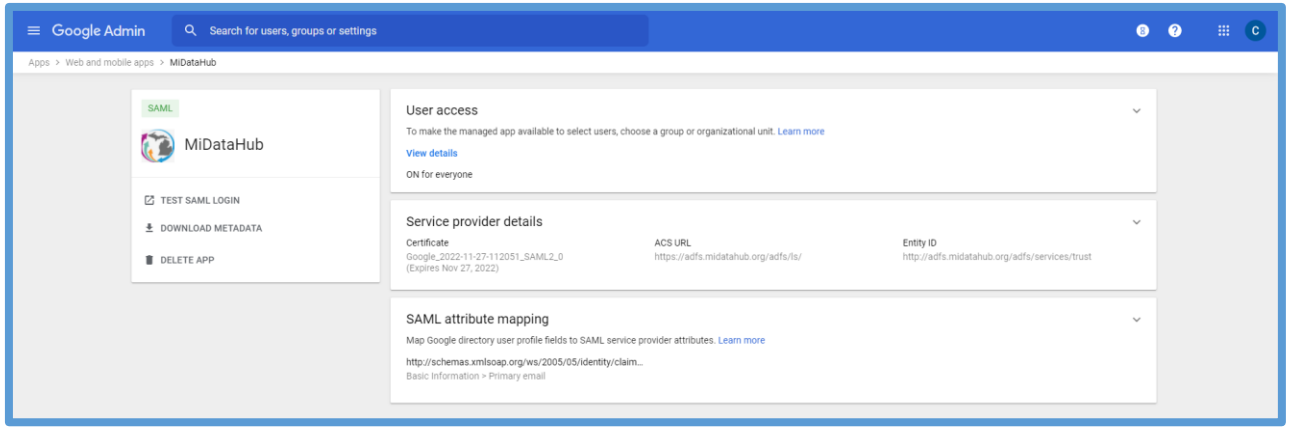


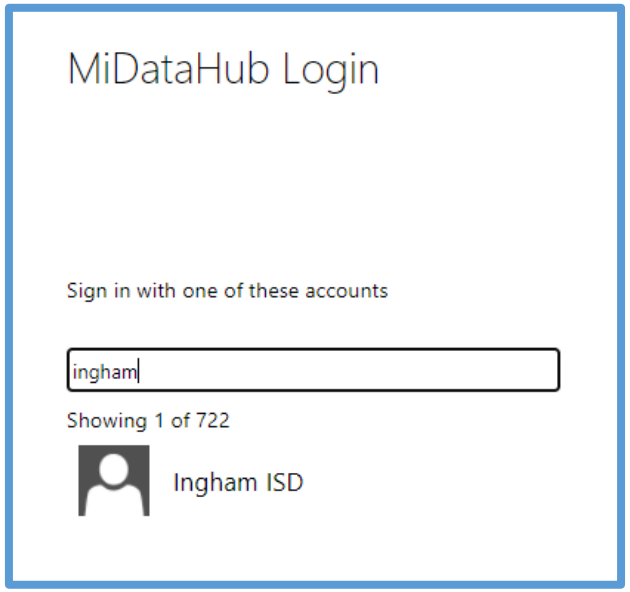


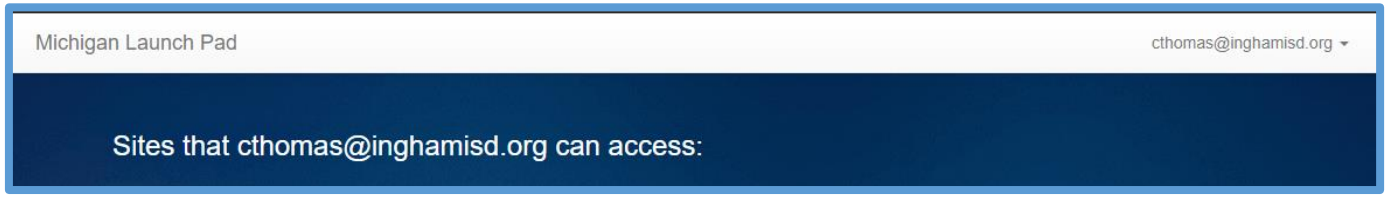


[Quickstart: Set up SAML-based single sign-on (SSO) for an application in your Azure Active Directory (Azure AD) tenant]

Sign-in with Google







https://support.google.com/a/answer/6087519?hl=en

Personalize / Maintain

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-ADUser

SYNTAX: Set-ADUser <CHOOSEYOUROWNADVENTURE>

https://docs.microsoft.com/en-us/powershell/module/activedirectory/set-aduser

Set-ADAccountPassword

SYNTAX: Set-ADAccountPassword –Identity <USER> -NewPassword <SECURESTRING> -Reset

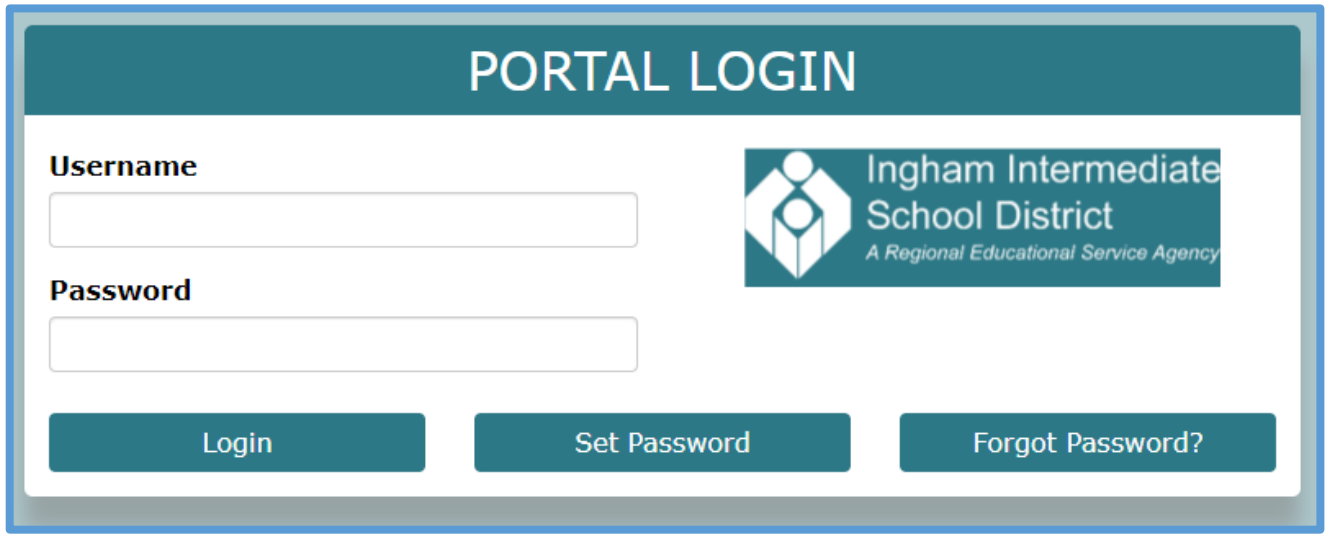
EXAMPLE:

Set-ADAccountPassword –Identity cttestudent –NewPassword (ConvertTo-SecureString –AsPlainText “Sup3rS3cur3!”) -Reset

https://docs.microsoft.com/en-us/powershell/module/activedirectory/set-adaccountpassword

Password Self-Service

- Consider if you will allow your students to change their passwords.
  - If so, what grade levels as this could affect how you create/manage users?
- PortalGuard (paid product) or other on-prem solution that then syncs to cloud
- Azure AD self-service password reset (SSRP) with Azure AD Connect Password Writeback (licensing required for each user who can use it, not tenant)



- https://github.com/unosquare/passcore
- https://github.com/pwm-project/pwm
  - Automate password resets with PWM

https://docs.microsoft.com/en-us/azure/active-directory/authentication/howto-sspr-deployment

De-Provision

Disable-ADAccount
or
Set-ADUser –Enabled $false

SYNTAX: Disable-ADAccount –Identity <USER>

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

```powershell
$usersToDisable = Import-Csv -Path c:\scripts\userstodisable.csv

foreach ($userToDisable in $usersToDisable)
{
    $user = Get-ADUser $userToDisable.SamAccountName
    $user | Disable-ADAccount
    $user | Move-AdObject -TargetPath 'OU=DisabledUsers,DC=<DOMAIN NAME>,DC=<DOMAIN TYPE>'
}
```

```powershell
Import-Module ActiveDirectory

$students = Import-Csv -Path "[redacted]\idm_students_update.csv"
$ouPath_disabled = 'OU=Disabled Users,DC=[redacted]'

foreach ($student in $students)
{
    $samAccountName = $student.idm_samaccountname

    try{
        #IF THE USER EXISTS DO THIS STUFF BELOW
        $user = Get-ADUser -Identity $samAccountName -Properties *

        #IF THE USER IS INACTIVE IN POWERSCHOOL DO THIS STUFF
        if($student.idm_status -eq 'I')
        {
            #DEFINE THE LAST LOGON OF THIS USER
            $lastLogon = $user.LastLogon

            if($lastLogon -eq '0')
            {
                $lastLogonDate = 'never'
            }
            else
            {
                $lastLogonDate = $user.LastLogonDate
            }

            #DEFINE WHERE THE USER WAS ORIGINALLY BEFORE WE MOVE THE OBJECT
            $distinguishedName = $user.DistinguishedName
            $originalOU =  $distinguishedName | ForEach-Object { $_ -replace '^.+?(?<!\\),','' }

            Write-Host "Disabling user: $samAccountName from $originalOU with a last logon of $lastLogonDate"
            Write-Host "I will set their description to: (LL: $lastLogonDate - O-OU: $originalOU - D: $(Get-Date -Format d) - A: IDM)"


            #DISABLE THE USER
            Set-ADUser -Identity $samAccountName -Enabled $false
            #CHANGE THE USER DESCRIPTION
            Set-ADUser -Identity $distinguishedName -Description "LL: $lastLogonDate - O-OU: $originalOU - D: $(Get-Date -Format d) - A: IDM"
            #MOVE THE USER OBJECT TO THE DISABLED OU
            Move-ADObject -Identity $user.DistinguishedName -TargetPath $ouPath_disabled
        }
    }
    catch{
        $_
    }
}
```

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

Move-ADObject

SYNTAX: Move-ADObject –Identity <USER> -TargetPath <OUPATH>

EXAMPLE: Move-ADObject –Identity cttestudent –TargetPath “OU=Test,DC=iisd,DC=local”

https://docs.microsoft.com/en-us/powershell/module/activedirectory/move-adobject

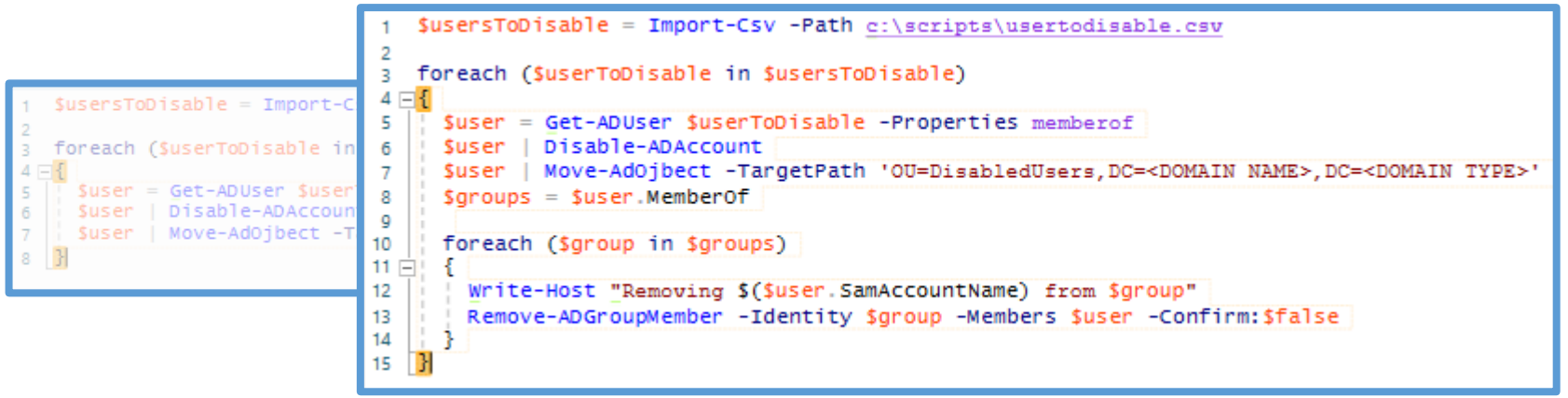
Remove-ADGroupMember

SYNTAX: Remove-ADGroupMember –Identity <GROUP> -Members <USER>

EXAMPLE:

Remove-ADGroupMember –Identity “Students” –Members cttestudent –Confirm:$false

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



```powershell
$usersToDisable = Import-Csv -Path c:\scripts\usertodisable.csv

foreach ($userToDisable in $usersToDisable)
{
  $user = Get-ADUser $userToDisable -Properties memberof
  $user | Disable-ADAccount
  $user | Move-AdOjbect -TargetPath 'OU=DisabledUsers,DC=<DOMAIN NAME>,DC=<DOMAIN TYPE>'
  $groups = $user.MemberOf

  foreach ($group in $groups)
  {
    Write-Host "Removing $($user.SamAccountName) from $group"
    Remove-ADGroupMember -Identity $group -Members $user -Confirm:$false
  }
}
```

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

Remove-ADUser

SYNTAX: Remove-ADUser –Identity <USER>

- 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

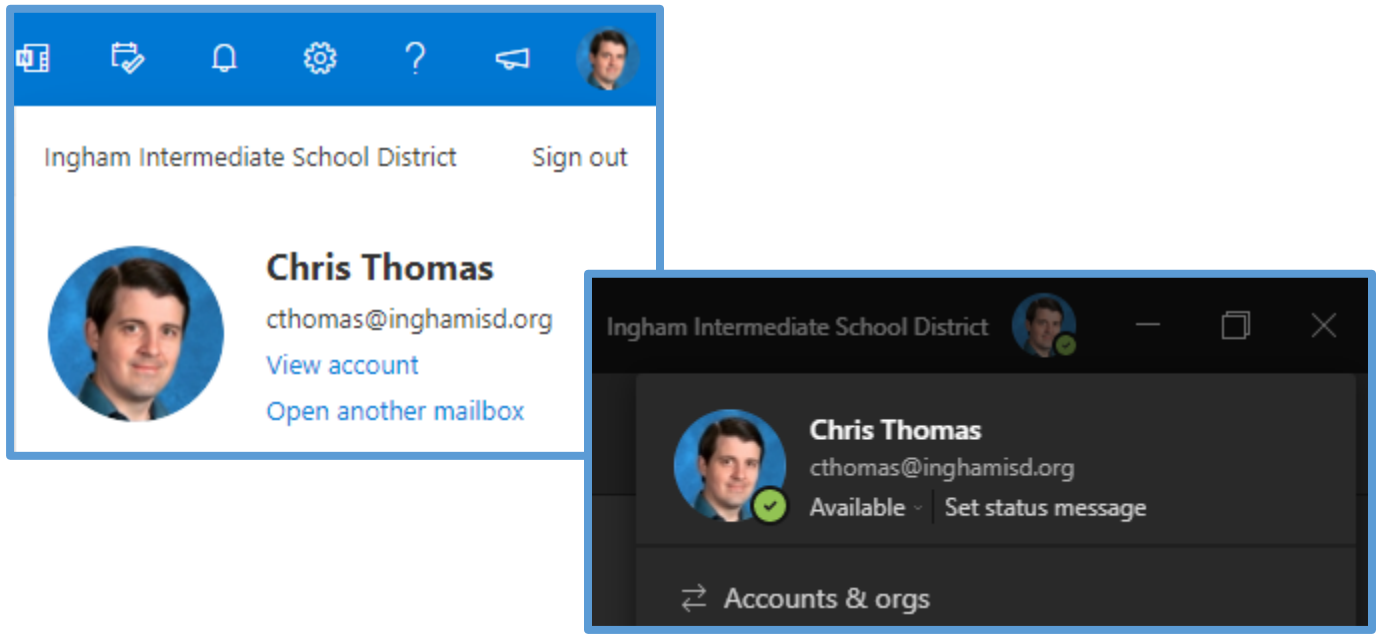
PRACTICE TIME

PowerShell For Microsoft Azure AD / Office 365

Set-UserPhoto

SYNTAX:

Set-UserPhoto -Identity <USER> -PictureData ([System.IO.File]::ReadAllBytes("<PATHTOIMAGE>"))



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

Get-Mailbox

SYNTAX: Get-Mailbox –Identity <USER>

EXAMPLE: Get-Mailbox –Identity cthomas

https://docs.microsoft.com/en-us/powershell/module/exchange/get-mailbox

Get-InboxRule

SYNTAX: Get-InboxRule –Mailbox <USER>

EXAMPLE: Get-InboxRule –Mailbox cthomas@inghamisd.org

https://docs.microsoft.com/en-us/powershell/module/exchange/get-inboxrule

Get-MailboxStatistics

SYNTAX: Get-MailboxStatistics –Identity <USER>

EXAMPLE: Get-Mailboxstatics –Identity cthomas@inghamisd.org | Select *

https://docs.microsoft.com/en-us/powershell/module/exchange/get-mailboxstatistics

New-ComplianceSearch

SYNTAX: New-ComplianceSearch –Name <NAME> -ExchangeLocation All –ContentMatchQuery ‘from:<user>’

https://www.codetwo.com/admins-blog/new-compliancesearch-new-version-of-search-mailbox/

https://docs.microsoft.com/en-us/powershell/module/exchange/new-compliancesearch

Start-ComplianceSearch

SYNTAX: Start-ComplianceSearch –Identity <NAME>

https://docs.microsoft.com/en-us/powershell/module/exchange/start-compliancesearch

Get-ComplianceSearch

SYNTAX: Get-ComplianceSearch

Get-ComplianceSearch –Identity <NAME>

https://docs.microsoft.com/en-us/powershell/module/exchange/get-compliancesearch

New-ComplianceSearchAction

SYNTAX: New-ComplianceSearchAction –SearchName <NAME> -Preview

New-ComplianceSearchAction –SearchName <NAME> -Export

New-ComplianceSearchAction –SearchName <NAME> -Purge –PurgeType HardDelete

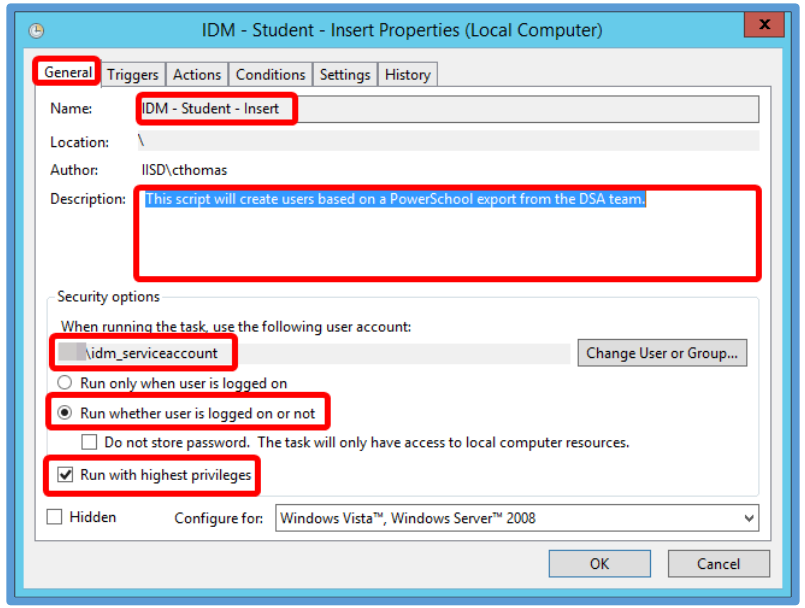
https://docs.microsoft.com/en-us/powershell/module/exchange/new-compliancesearchaction

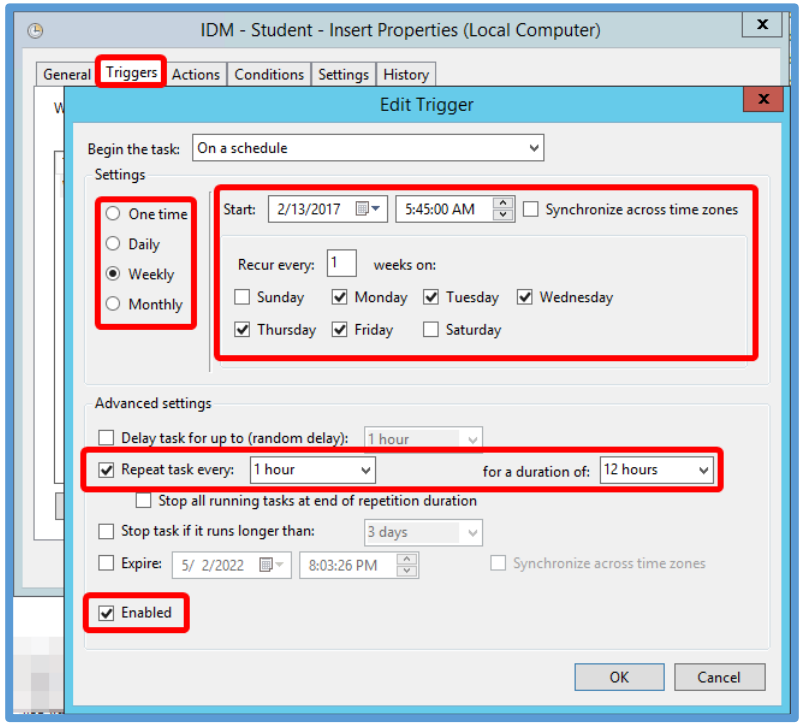
PRACTICE TIME

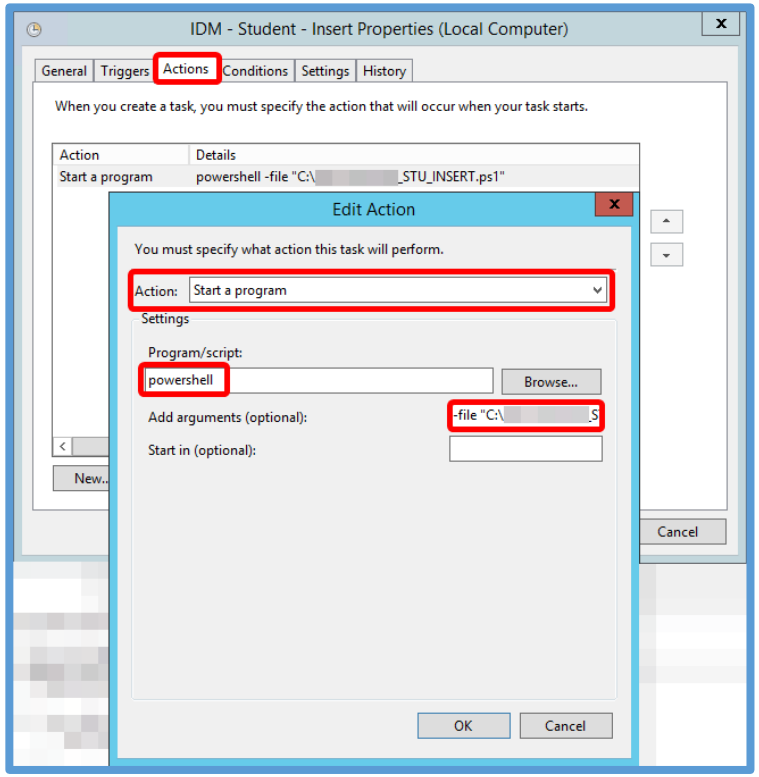
Additional Resources

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







Azure AD Connect

- What is Azure AD Connect?
  - Azure AD Connect sync: Understand and customize synchronization
- Azure AD Connect and Azure AD Connect Health installation roadmap

Google Cloud Directory Sync (GCDS)

Google Suite Password Sync (GSPS)

- About Google Cloud Directory Sync
  - GCDS best practices
- Active Directory user account provisioning

- About G Suite Password Sync

-- The End --

Ask Questions …

… then go forth and iterate.

PAST PRESENTATIONS:
https://github.com/chrisATautomatemystuff/Presentations