MOREnet Technical Training Summit 2025

PowerShell Scripting 101-301

Brad Jones

#whoami

Brad Jones

Started working on computers in College Fall 1991

High School - BASIC, Batch files

College - C++, COBOL, Fortran, LegoScript, Pascal

Post College - BASH, PowerShell, Python, SQL

Disclaimer:

I am not a programmer, nor an expert in programming.

I like automations, and scripting is a tool to help automate tasks.

Google is your friend. Use any of the following information at your own risk.

PowerShell Agenda

Very light PowerShell overview

Scripts to use with MS Active Directory
- Start small and grow to something you can use to automate user management.
- Script to modify existing users
  - Reset student passwords each August
  - Script to execute a password reset for a single user?

Managing network switches
- Script to backup running configurations
- Script to update firmware on switches (HP/Aruba in this example)
- Script to simply bounce a specific switch port

Script to monitor report exports for 24×7 door access permissions

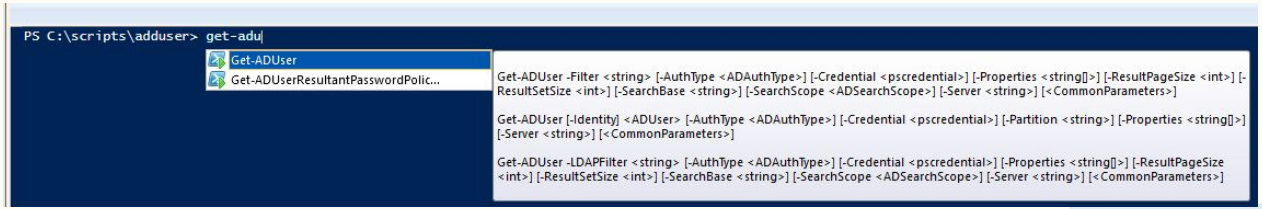
PowerShell 101

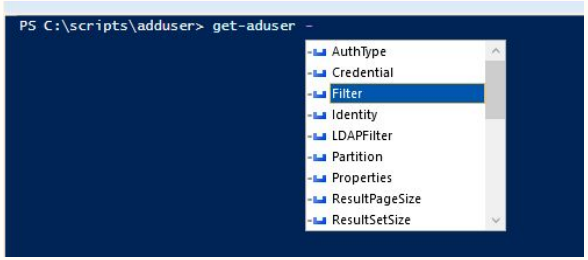
PowerShell command line vs PowerShell ISE *(no longer active development August 2024)*
https://learn.microsoft.com/en-us/powershell/scripting/windows-powershell/ise/introducing-the-windows-powershell-ise?view=powershell-7.4

Use whichever you are comfortable with

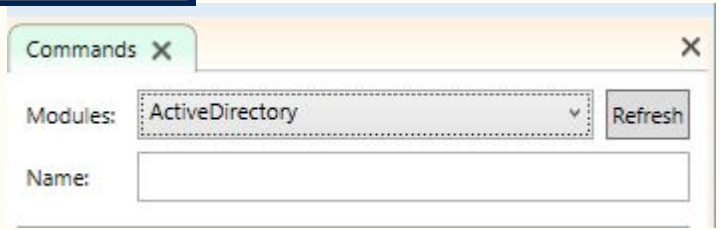
Both interfaces support autocomplete

ISE will help give you prompt boxes as you are typing





ISE has a panel to help you review commands that are available for the different modules like ActiveDirectory

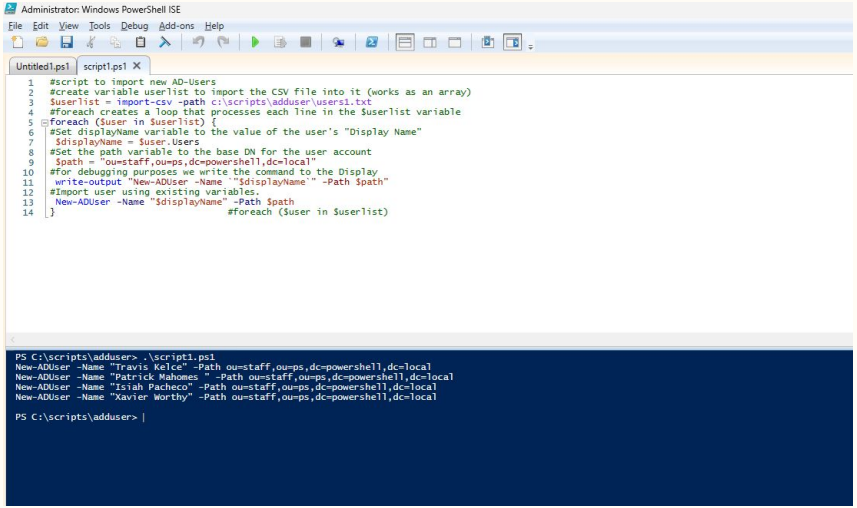


PowerShell 101

PowerShell command line vs PowerShell ISE *(no longer active development August 2024)*
https://learn.microsoft.com/en-us/powershell/scripting/windows-powershell/ise/introducing-the-windows-powershell-ise?view=powershell-7.4

ISE provides you a separate script and execution pane

Allows you to have your script on the same screen open while debugging



PowerShell 101

By default, Powershell is NOT Case Sensitive
Powershell cmdlets we are going to use:

Add-Content	#add data to a file
Clear-Content	#remove content from a file or directory
Get-ADUser	#find users in active directory
Get-Help	#display information about cmdlet.
New-ADUser	#add user to AD
ForEach (.x.) {.y.}	#goes thru a list of values x and process y commands
If (.x.) {.y.}	#a comparison x that equals true or false and process y commands based on result
Import-CSV	#import/read a text file into a variable (array)
Write-Output	#send information to the screen, use for debugging, general information

PowerShell 101

#DOS Commands also work in PowerShell

Create a Script folder on the C drive

#Make a directory for our addusers scripts

mkdir c:\scripts\addusers

#Change to the addusers directory

cd \scripts\addusers

PowerShell 101

#We first need to find the LDAP structure AD is using

Get-ADUser -Filter * # -filter is required for this cmdlet

#the output will have the following 3 records:

#DistinguishedName : CN=Administrator,CN=Users,DC=PowerShell,DC=local
#DistinguishedName : CN=Guest,CN=Users,DC=PowerShell,DC=local
#DistinguishedName : CN=krbtgt,CN=Users,DC=PowerShell,DC=local

we can look at the following record for the information we need:
#DistinguishedName : CN=Administrator,CN=Users,DC=PowerShell,DC=local

#DC=PowerShell,DC=local is our LDAP AD search base

#CN=Users,DC=PowerShell,DC=local is our LDAP Users search base

PowerShell 101

#Let's create a Staff and Student user inside a Staff and Student OU using ADUC

#And then re-run get-aduser

Get-aduser -filter *

#the output is the same as before (get-aduser cmdlet only finds user objects)
#but with 2 more user objects

#... same 3 as earlier plus
#DistinguishedName : CN=Staff A,OU=Staff,OU=PS,DC=PowerShell,DC=local
#DistinguishedName : CN=Student A,OU=Students,OU=PS,DC=PowerShell,DC=local

PowerShell 101

#Let's add a user to AD

New-ADUser -Name "Test User"

#notice where it adds the user. Default CN=Users folder, not where we want it.

#We want the user account to go into a specific OU.

New-ADUser -Name "Test User2" -Path "ou=staff,ou=ps,dc=powershell,dc=local"

script1.ps1

PowerShell 101

#Lets import a list of users from a file (Copy *.csv and *.txt files from backup folder)
#PowerShell works well with CSV files that have a header row
#We will create a file to execute so variables used in the script won't be held in memory

#script to import new AD-Users
$userlist = import-csv -path c:\scripts\adduser\users1.txt

script1.ps1

PowerShell 101

```
#Lets import a list of users from a file
#PowerShell works well with CSV files that have a header row
#We will create a file to execute so variables used in the script won't be held in memory

#script to import new AD-Users
$userlist = import-csv -path c:\scripts\adduser\users1.txt
foreach ($user in $userlist) {
```

script1.ps1

PowerShell 101

#Lets import a list of users from a file
#PowerShell works well with CSV files that have a header row
#We will create a file to execute so variables used in the script won't be held in memory

#script to import new AD-Users
$userlist = import-csv -path c:\scripts\adduser\users1.txt
foreach ($user in $userlist) {
$displayName = $user.Users

script1.ps1

PowerShell 101

#Lets import a list of users from a file
#PowerShell works well with CSV files that have a header row
#We will create a file to execute so variables used in the script won't be held in memory

#script to import new AD-Users

```
$userlist = import-csv -path c:\scripts\adduser\users1.txt
foreach ($user in $userlist) {
 $displayName = $user.Users
 $path = "ou=staff,ou=ps,dc=powershell,dc=local"
```

script1.ps1

PowerShell 101

#Lets import a list of users from a file
#PowerShell works well with CSV files that have a header row
#We will create a file to execute so variables used in the script won't be held in memory

#script to import new AD-Users

```
$userlist = import-csv -path c:\scripts\adduser\users1.txt
foreach ($user in $userlist) {
 $displayName = $user.Users
 $path = "ou=staff,ou=ps,dc=powershell,dc=local"
 write-output "New-ADUser -Name `"$displayName`" -Path $path"
```

script1.ps1

PowerShell 101

#Lets import a list of users from a file
#PowerShell works well with CSV files that have a header row
#We will create a file to execute so variables used in the script won't be held in memory

#script to import new AD-Users

```
$userlist = import-csv -path c:\scripts\adduser\users1.txt
foreach ($user in $userlist) {
 $displayName = $user.Users
 $path = "ou=staff,ou=ps,dc=powershell,dc=local"
 write-output "New-ADUser -Name `"$displayName`" -Path $path"
 New-ADUser -Name "$displayName" -Path $path
}
```

#Now review the users you created into the Staff OU

PowerShell 101

Other LDAP Attributes we need to know about when adding new users into AD:

cn - common name
displayName - a visual aspect of the users name
distinguishedName - starts with the CN attribute and adds in the -Path (OU structure)
employeeID - to help with duplicates, identifying conflicting names, unique ID?
givenName - first name
mail - email address
name - display name
o - organization - use as school\$samAccountName (needed for GCPW)
path - OU structure for the object
primaryGroupID - use if you want to set the primary group of the user
sAMAccountName - username (security Accounts Manager) **(what is it's catch?)**
sn - last name
userPrincipalName - username@domain.name (sAMAccountName replacement)

PowerShell 101

PowerShell cmdlet parameters:

5 types

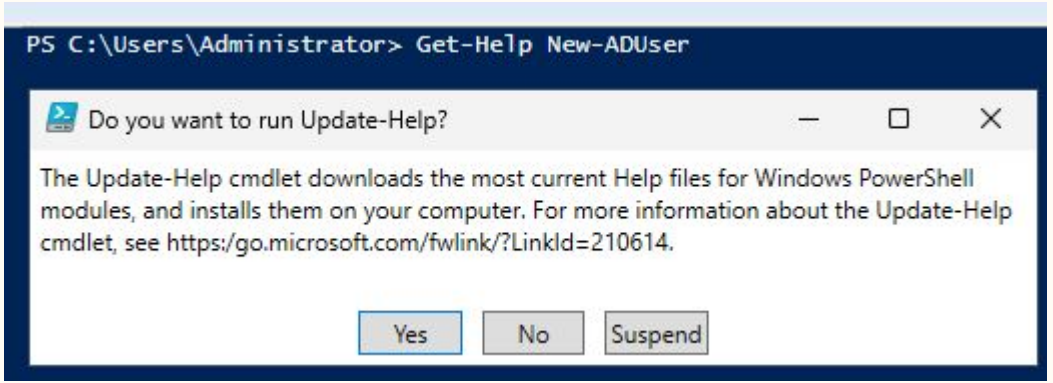
Positional vs Named (named - -samAccountName $username)

Required vs Optional (required - Get-ADUser -Filter ...)

Switch

Get-Help will ask to download help Files the first time you run the Command.

Get-Help New-ADUser



PowerShell 101

Why normal people don’t like to write scripts:

SYNTAX

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

PowerShell 101

PowerShell New-ADUser info

Description:

You **WANT TO** specify the SamAccountName parameter to create a user.

Our first example did not specify SamAccountName
New-ADUser -Name "Test User"

Get-Help New-ADUser -examples

Get-Help New-ADUser -detailed

Get-Help New-ADUser -full (shows the “required” status of the parameter)

Get-Help New-ADUser -online (opens the URL for the online version of full)

Hint: use 2 or more screens to help when writing scripts

PowerShell 101

New-ADUser parameters I recommend using

[-Name] <System.String>
[-AccountPassword <SecureString>]
[-ChangePasswordAtLogon <System.Boolean>]
[-DisplayName <System.String>]
[-EmailAddress <System.String>]
[-EmployeeID <System.String>]
[-Enabled <System.Boolean>]
[-GivenName <System.String>]
[-Organization <System.String>] (for Google Client Provider for Windows [GCPW])
[-Path <System.String>]
[-SamAccountName <System.String>]
[-Surname <System.String>]
[-UserPrincipalName <System.String>]

PowerShell 101

Other New-ADUser parameters you might be interested in

[-CannotChangePassword <System.Boolean>]
[-Department <System.String>]
[-Description <System.String>]
[-EmployeeNumber <System.String>]
[-HomeDirectory <System.String>]
[-HomeDrive <System.String>]
[-LogonWorkstations <System.String>]
[-OtherAttributes <Hashtable>]
[-PasswordNeverExpires <System.Boolean>]
[-ProfilePath <System.String>]
[-ScriptPath <System.String>]

PowerShell 101

Let’s expand on our New-ADUser script to use a CSV that has more values so we can create an user account with all the necessary information.

PowerShell variables makes this process much easier

Variables we will use:

$changePasswordAtLogon
$displayName
$employeeID
$email
$givenName
$upn
$password
$path
$samAccountName
$surName

PowerShell 101

Look at our CSV file Header Row: (users2.csv)

FirstName,LastName,username,DisplayName,UserID,Password,email,upn,path

Using a Header row:

- Pull values into variables by using the column name
- Makes a very simple process for this task

Using no Header row:

- Is possible
- For this purpose, it's a little more complicated

Copy script1.ps1 to script1.b.ps1 (leaves a working backup in tact)

PowerShell 101

#FirstName,LastName,Username,DisplayName,UserID,Password,Email,UPN,Path

#setting variables (the easy way) - add the following into the ForEach {} section

#force new user to change password at next logon
$changePasswordAtLogon = 1
$givenName = $user.FirstName
$surName = $user.LastName
$samAccountName = $user.Username
$displayName = $user.DisplayName
$employeeID = $user.UserID
$password = $user.Password
$email = $user.email
$upn = $user.UPN
$path = ou=staff,ou=ps,dc=powershell,dc=local

PowerShell 101

Execute script1.b.ps1

Errors were generated about the password variable
We will discuss this in a little bit.

Notice that the accounts were still created.

Delete these 4 accounts (we will recreate them again.)

- Matt Araiza
- Nick Bolton
- Hollywood Brown
- Deon Bush

PowerShell 102

Now that you have a simple script to import users.
How about we learn some ways to make this process simpler?

Can we generate the same information without having to:

- Hand modify the CSV file
- Populate all values in all the rows/columns in the CSV file

You don't want to have to store all these fields in another system.

We can build variables from values of other fields.

PowerShell 102

#FirstName,LastName,Username,DisplayName,UserID,Password,Email,UPN,Path

#Start with our unique identifier sAMAccountName
could use mail, but there's a conflict you need to be aware of (will discuss later)

#we will start with creating our username so we can use it in other variables
$samAccountName = $user.FirstName + $user.LastName
#example statement for a username that is firstInitialLastName
$samAccountName = $user.FirstName[1] + $user.LastName

#to eliminate having to have a DisplayName value in your CSV file
$displayName = $user.FirstName + " " + $user.LastName
#could also create displayName from 2 other variables we populate
$displayName = $givenName + " " + $surName

PowerShell 102

#FirstName,LastName,Username,DisplayName,UserID,Password,Email,UPN,Path

#to eliminate having to have an $email value in your CSV file
$email = $samAccountName + “@powershell.com”
$email = $user.FirstName + $user.LastName + “@powershell.com”
#but later you may see having the email address is helpful

#we don’t need to have UPN value, as it’s going to be the value of $email
$upn = $email #no need to create a variable that is the same as another variable.
$path = ou=staff,ou=ps,dc=powershell,dc=local #(can be stated in the script)

#create a password with a standard secret key everyone knows,
#but then add a unique code only the employee would know.
$password = “secretkey” + $user.Password

PowerShell 102

#putting this together one step at a time. (create script2.ps1 from scratch)
#Let's start with setting up the loop in the script.

#set the variable to import the CSV file into
$userlist = Import-Csv -Path c:\scripts\adduser\Users2.csv

#start the loop to process the file
ForEach ($user in $userlist) {

#combine first and last names to create username
$givenName = $user.FirstName
$surName = $user.LastName
$samAccountName = $givenName + $LastName

Write-Output "$givenName,$surName,$samAccountName"

} #Execute script

script2.ps1

PowerShell 102

#Let's add one piece at a time (we need to start with creating a valid username structure)

ForEach ($user in $userlist) {

#combine first and last names to create username
 $samAccountName = $givenName + $LastName
#remove non-alphanumeric characters (PS is case insensitive, think like approval list)
$samAccountName = $samAccountName -Replace "[^a-z0-9]"

 Write-Output "$givenName,$surName,$samAccountName"
} #Execute script

PowerShell 102

```
#Let's add one piece at a time  (we need to start with creating a username)

ForEach ($user in $userlist) {

#combine first and last names to create username
 $samAccountName = $givenName + $LastName
#remove non-alphanumeric characters (PS is case insensitive, think like approval list)
 $samAccountName = $samAccountName -Replace "[^a-z0-9]"
#samAccountName cannot be over 20 characters (will test later)
 if ($samAccountName.Length -gt 20) {
  $samAccountName = $samAccountName.substring(0,20)
 }


 Write-Output "$givenName,$surName,$samAccountName"
}                              #Execute script
```

script2.ps1

PowerShell 102

```
#Let's add one piece at a time  (we need to start with creating a username)

ForEach ($user in $userlist) {

#combine first and last names to create username
 $samAccountName = $givenName + $LastName
#remove non-alphanumeric characters (PS is case insensitive, think like approval list)
 $samAccountName = $samAccountName -Replace "[^a-z0-9]"
#samAccountName cannot be over 20 characters (will test later)
 if ($samAccountName.Length -gt 20) {
  $samAccountName = $samAccountName.substring(0,20)
 }
 $samAccountName = $samAccountName.tolower()

 Write-Output "$givenName,$surName,$samAccountName"
}                              #Execute script
```

script2.ps1

PowerShell 102

```
ForEach ($user in $userlist) {

#setting other variables
 $displayName = "$givenName $surName"
 $employeeID = $user.UserID
 $email = $samAccountName + "@powershell.com"
 $path = "ou=staff,ou=ps,dc=powershell,dc=local"

#you can replace the write-output line, or modify it to add additional info
 Write-Output "$displayName,$employeeID,$email,$path"

}                          #Execute script
```

script2.ps1

PowerShell 102

#Last bit of info necessary info - adding passwords to AD accounts via PowerShell

ForEach ($user in $userlist) {

$password = “secretKey” + $user.Password

#you cannot import a plaintext password, so we pass it thru a secureString process
$secpassword = ConvertTo-SecureString -String $password -AsPlainText -Force
#$secpassword = ConvertTo-SecureString -String $user.Password -AsPlainText -Force

Write-Output "$secpassword"

}

script2.ps1

PowerShell 102

```
#Now to put it all together

$userlist = Import-Csv -Path c:\scripts\adduser\Users2.csv
ForEach ($user in $userlist) {
 $samAccountName = $user.FirstName + $user.LastName
 $samAccountName = $samAccountName -Replace "[^a-z0-9]"
 if ($samAccountName.Length -gt 20) {
  $samAccountName = $samAccountName.substring(0,20)
 }
 $samAccountName = $samAccountName.tolower()
 $givenName = $user.FirstName
 $surName = $user.LastName
 $displayName = "$givenName $surName"

#...
```

PowerShell 102

```
#...

 $employeeID = $user.UserID
 $email = $samAccountName + "powershell.com"
 $path = "ou=staff,ou=ps,dc=powershell,dc=local"
 $changePasswordAtLogon = 1
 $password = $user.Password
 $secpassword = ConvertTo-SecureString -String $password -AsPlainText -Force

 Write-output
"$samAccountName,$givenName,$surName,$displayName,$employeeID,$email,$path,
$changePasswordAtLogon,$password,$secpassword"

}                          #Execute script
```

PowerShell 102

```
#last change is to process the new-aduser command
#... Add below the Write-Output cmdlet
New-ADUser -AccountPassword $secpassword `
                -changepasswordatlogon $changePassAtLogon `
                -DisplayName "$displayName" `
                -EmailAddress $email `
                -EmployeeID $employeeID `
                -Enabled 1 `
                -GivenName "$givenName" `
                -Name "$displayName" `
                -Path $path `
                -SamAccountName $samaccountname `
                -Surname "$surName" `
                -UserPrincipalName $email
}                       #Execute script
```

PowerShell

QUESTIONS???

WAIT!
We aren’t done yet!

Wouldn’t it be nice to not have to manage a CSV file of just new additions?

Is there a way we can just extract the same file from our management systems so we can simplify the import process?

Of Course there is!

We just need a “unique variable” to determine if this is a new user or not.
Any guesses on what this “unique variable” could be?

PowerShell

Email address or sAMAccountName to the rescue!

2 ways to handle this.

Easy way

- Store email addresses in management system
- Include this value in the export

Harder way but doable

- Build email address value on every user when processing the script.
- Same process we just went thru.
- More prone to duplication of user errors

We will take the ‘easy way’ route for this class. Copy script2.ps1 to script3.ps1

script3.ps1

PowerShell 201

```
ForEach ($user in $userlist) {

#We need to verify email address field in the CSV file contains @powershell.com
 if ($user.email -like "*@powershell.com"){

#create samAccountName from email address truncating @domain.com
 $samAccountName = $user.email.Split('@')[0]
 }
 else {
 $samAccountName = $user.FirstName + $user.LastName



#...
```

PowerShell 201

```
#...
 $samAccountName = $samAccountName.tolower()
 }

#Validate if samAccountName already exists in AD
 If (@(Get-ADUser -Filter { samAccountName -eq $samAccountName }).count -eq 0) {
 $givenName = $user.FirstName

#...

#add the following to the bottom of the script
 } else {
 Write-Output “User $samAccountName already exists”
 }
}       #last closing bracket and execute script
```

PowerShell

You will see 4 errors on existing accounts.

These were the first 4 accounts we imported using a CSV of just names.

Not enough info to do it correctly, but we will fix that later. (if there's time)

So you can import all your staff with an export of staff to a CSV file,

Can we do this for Students too???

OF COURSE WE CAN!

BUT WAIT! How do we get the email addresses back into our data source!

script3.ps1

PowerShell 201

#...

#Add to the top of the script
#initialize variables
#append to file mail addresses that need to be imported back into data source
$staffimportfile = "c:\scripts\adduser\staffimport.csv"
#Clear data in import file to have a clean file to start with
Clear-Content -Path $staffimportfile

#Add the following either below or above the New-ADUser cmdlet
Add-Content -Path $staffimportfile -Value "$employeeID,$email"

#Execute script and inspect staffimport.csv to see contents.

PowerShell

Three thoughts on ways to process Staff and Student data:

1) Use 1 combined CSV file:
    All students and staff in one file?
    Each record must have a column in which to designate the staff/student
    Requires less code manipulation

2) Use 2 separate CSV files:
    One consisting of all Staff
    One consisting of all Students
    Requires more code manipulation
    Works when you have 2 data sources

3) Use 2 Completely separate scripts?

3 is too easy, so we are going to go with #2. Copy script3.ps1 to script4.ps1

PowerShell 202

```
# Building on script3.ps1, we need to add a loop outside the entire script's main loop
# this loop will look for 2 files
# the filename will be used to determine which part of the script is going to be processed
# we are replacing the current Import-Csv line with the following code

ForEach ($fileToProcess in "staff.csv", "student.csv") {
 If ($fileToProcess -eq "staff.csv") {
  $userlist = Import-Csv -Path c:\scripts\adduser\staff.csv
 } else {
  $userlist = Import-Csv -Path c:\scripts\adduser\student.csv
 }

#rest of script goes here... with more modifications on the following slides
#add the following } character to the bottom of the script to close off the outside foreach

}
```

PowerShell 101

Now we need to test the script to make sure it works properly.

Remark any parts of the script that will make changes:
i.g. The New-ADUser section of code

Execute the script4.ps1 and verify it executes without error.

This validates syntax is correct.

This is a great practice as you compose your script so you don't forget what the last change you made was.

Also, make backups as you go.
Sometimes it's just easier to start back from a working script.

PowerShell 202

```
#next step is to check which file we are processing
#add the If line below the existing ForEach line

ForEach ($user in $userlist) {

 If ($fileToProcess -eq “staff.csv”) {

if ($user.email -like “*@powershell.com”) {

#add the following to the end of the script above the last }
  } else {
# section to process students
  }
 }

}
```

PowerShell 202

The script should execute without error

There are no changes necessary for the Staff part of the script

Don’t fix it if it ain’t broke!

PowerShell 202

If all syntax is correct, we are using a copy of users3.csv for staff.csv

Next steps are to replicate the steps necessary to create students
We will look at placing each student in a Grade Level OU.

We will start down in the portion of the script marked by
 # section to process students

Many lines will be the same as the staff section

PowerShell 202

#start by adding the following at the top of the script.
#this will help in calculating gradyears we are adding to student usernames

#initialize variables
#current Graduation Year
$baseGrade = 2025
#append to file email addresses that need to be imported back into data source
$studentimportfile = "c:\scripts\adduser\studentimport.csv"
#Clear data in import file to have a clean file to start with
Clear-Content -Path $studentimportfile

PowerShell 202

```
# section to process students  (all new content starting below the } else { from the if staff)
 if ($user.email -like "*@powershell.com"){
#create samAccountName from email address truncating @domain.com
  $samAccountName = $user.email.Split('@')[0]
 }
 else {
  $samAccountName = $user.FirstName + $user.LastName
  $samAccountName = $samAccountName -Replace “[^a-z0-9]”
  if ($samAccountName.Length -gt 18) {
   $samAccountName = $samAccountName.substring(0,18)
  }
  $samAccountName = $samAccountName.tolower()

#...
```

script4.ps1

PowerShell 202

```
#...

#calculate gradyear to append to username
 $grade = $user.Grade
 switch ($grade) {
   "K"  { $gradyear = $baseGrade += 12; break }
   "1"  { $gradyear = $baseGrade += 11; break }
   "2"  { $gradyear = $baseGrade += 10; break }
   "3"  { $gradyear = $baseGrade += 9; break }
   "4"  { $gradyear = $baseGrade += 8; break }
   "5"  { $gradyear = $baseGrade += 7; break }
   "6"  { $gradyear = $baseGrade += 6; break }
   "7"  { $gradyear = $baseGrade += 5; break }

#...
```

PowerShell 202

```
#...
    "8"  { $gradyear = $baseGrade += 4; break }
    "9"  { $gradyear = $baseGrade += 3; break }
    "10" { $gradyear = $baseGrade += 2; break }
    "11" { $gradyear = $baseGrade += 1; break }
    "12" { $gradyear = $baseGrade; break }
   default {$gradyear = "invalid"; $validAccount = "False" }
  }
#for another time, we should test for invalid data in the Grade field to catch issues
#or you can just write-output if validAccount is false

#...
```

script4.ps1

PowerShell 202

```
#...

#convert $gradyear to string to grab last 2 characters
   $gradyear = $gradyear.ToString()
   $gy = $gradyear.substring($gradyear.Length -2)
   $samAccountName = $samAccountName + $gy


#...
```

script4.ps1

PowerShell 202

```
#...

#Validate existing samAccountName already exists.

  if (@(Get-ADUser -Filter { samAccountName -eq $samAccountName }).count -eq 0) {
##set user variables
   $givenName = $user.FirstName
   $surName = $user.LastName
   $displayName = "$givenName $surName"
   $employeeID = $user.UserID
   $email = $samAccountName + "@powershell.com"
   $grade = $user.Grade
   $path = "ou=$grade,ou=students,ou=ps,dc=powershell,dc=local"

#...
```

PowerShell 202

#...

```
#so we don’t have to have students change passwords on initial login
   $changePasswordAtLogon = 0
   $password = $user.Password
   $secpassword = ConvertTo-SecureString -String $password -AsPlainText -Force

   Write-output
"$samAccountName,$givenName,$surName,$displayName,$employeeID,$email,
$path,$changePasswordAtLogon,$password,$secpassword"
```

#...

script4.ps1

PowerShell 202

```
#...

#the following section can be copied and is a block remark using the <# ... #> tags
<#
  New-ADUser       -AccountPassword $secpassword `
                   -changepasswordatlogon $changePassAtLogon `
                   -DisplayName "$displayName" `
                   -EmailAddress $email `
                   -EmployeeID $employeeID `
                   -Enabled 1 `
                   -GivenName "$givenName" `
                   -Name "$displayName" `
                   -Path $path `
#...
```

PowerShell 202

```
#...
                    -SamAccountName $samaccountname `
                    -Surname "$surName" `
                    -UserPrincipalName $email
#>
 #Add the following either below or above the New-ADUser cmdlet
Add-Content -Path $studentimportfile -Value "$employeeID,$email"
#the below is the rest of the script
   }
        else {
    Write-Output "User $samAccountName already exists"
   }
  }
 }
}
```

PowerBreak

Execute script4.ps1 and verify student accounts were created.

PowerBreak as we move to another type of script with AD

Next scripts will handle Modifying LDAP attributes in AD.

PowerShell 101 - More Cmdlets

New Powershell cmdlets we are going to use:

Disable-ADAccount	#disabled AD Account
Enable-ADAccount	#enables AD Account
Set-ADAccountPassword	#sets password on AD Account
Read-Host	#Accept input from terminal

PowerShell 202

How many of you reset passwords using ADUC?

How many of you reset student passwords each year?

How about “disabling” student accounts (and staff), if they are no longer in the district?

What if the following was possible:

Disable all students
Reset passwords and (re)enable current students
By running a single command?

So, let’s start by creating the directory and changing into c:\scripts\passwordReset
mkdir c:\scripts\passwordReset
cd \scripts\passwordReset

script5.ps1

PowerShell (6-7 lines of code?)

```
#copy over the students.csv file from the backup folder

#script to reset passwords in CSV file
#disable all accounts in the Students OU
Get-ADUser -Filter * -SearchBase "ou=students,ou=ps,dc=powershell,dc=local" | Disable-ADAccount

#import file into $userlist variable
$userlist = Import-Csv -Path C:\scripts\passwordReset\student.csv

#loop to enable and reset passwords of each account in import file
ForEach ($user in $userlist) {
 $samaccountname = $user.email.Split('@')[0]
 Enable-ADAccount -Identity "$samaccountname"


#...
```

PowerShell (Time for some FUN!)

#..

#modify password variable in CSV file with a secret key to make the password longer

$password = "wsd" + $user.Password

Set-ADAccountPassword -Identity "$samaccountname" -NewPassword (ConvertTo-SecureString -AsPlainText "$password" -Force)

$secpassword = ConvertTo-SecureString "$password" -AsPlainText -Force
Set-ADAccountPassword -Identity "$samaccountname" -Newpassword $secpassword
Set-ADAccountPassword -Identity “$samaccountname” -NewPassword (ConvertTo-SecureString -AsPlainText “$user.password” -Force)

}

PowerShell (Can we make this any easier?)

How often do you need to reset passwords

Do you use ADUC?

Is there a simpler method?

Can we write a short script to help make this easier?

3 lines of code says we can!

script6.ps1

PowerShell (Can this be any easier?)

```
#prompt for the username
$username = Read-Host -Prompt "Provide the username."
#Write-Output $username

#prompt for the password
$secpassword = Read-Host -Prompt "Provide the password." -AsSecureString
#Write-Output $secpassword

#Set the password
Set-ADAccountPassword -Identity $username -NewPassword $Secpassword
```

Questions?

Any questions before we move to another way to use PowerShell?

PowerShell Backing up Switch Configurations

Network Switches - for those not moving to cloud management

Script to copy your running configuration to a local machine.

You need the kitty.exe program
https://www.9bis.net/kitty/index.html#!index.md

Scripts on in the backup folder, but since we can't all have a switch in this session, I'll just run thru what these scripts do.

PowerShell Backing up Switch Configurations

```
#set variables
$pword = "password"
$server = "ip.ad.dres.s"            #IP Address of TFTP server (solar winds)

#set csvFile variable to hold our switch inventory information
$csvFile = Import-Csv -Path ".\SwitchInventory.csv"

#loop thru each line in the csvFile
Foreach ($line in $csvFile) {
$uname = "username"        # set here due to newer switch having different login.
#use IP address of switch to name created files
 $ip = $line."IP Address"      #csv file header has a space in it, so quotes are required
#store path and filename to save config file in a subfolder called conf
 $ipFile = "conf`\" + $ip + ".conf"

#...
```

PowerShell Backing up Switch Configurations

```
#...

#Compare string of Switch model number from inventory to determine which commands to use
 if ( $line.Model -eq "5304XL") {
  .\kitty.exe $ip -telnet -cmd "\p $uname\n$pword\n\p
#       kitty.exe - command line executable program that can parse thru values in a command
#       $ip - IP Address of the switch
#       -telnet - use the telnet protocol
#       -cmd - execute the following command sequence of characters
#               " - starts the command sequence of characters
#               \p - insert a pause before continuing with the command
#               " " - space character due to this model switch needing a key pressed to prompt for username
#               $uname\n - insert username in sequence and return
#               $pword\n - insert password in sequence and return
#               \p - insert a pause before continuing with the command
#...
```

PowerShell Backing up Switch Configurations

```
#...

# continuing with the command sequence of characters
 copy startup-config tftp $server $ipFile\n
  logout\n
  yes\n"
#               copy startup-config tftp $server $ipFile - switch command to copy startup-config to tftp server
#               \n - return
#               logout\n - switch command to end session
#               yes\n - switch command answering question to confirm logout.
 }              # end of 5304 section



#note: you do not have to put logout\n and Exit\n on it's own line
        - just easier to read
```

PowerShell Backing up Switch Configurations

```
#...

#Compare string of Switch model number from inventory to determine which commands to use
 if ( $line.Model -eq "Aruba-2930F-24G-740W-PoEP-4SFPP" -or `
     $line.Model -eq "Aruba-2930F-48G-740W-PoEP-4SFPP" -or `
     $line.Model -eq "Aruba-2930F-24G-PoEP-4SFPP" -or `
     $line.Model -eq "Aruba-2930F-48G-4SFPP" -or `
     $line.Model -eq "Aruba-2930F-8G-PoEP-2SFPP") {
  .\kitty.exe $ip -telnet -cmd "\p$uname\n$pword\n\pcopy startup-config tftp $server $ipFile\nlogout\nyes\n"

# went telnet because ssh would not logout of switch completely
 }

#Note: use of "-or `" to continue with different but like models that use the same command sequence
#...
```

PowerShell Backing up Switch Configurations

```
#...

#Next switch model from inventory to determine which commands to use
 if ( $line.Model -eq "JL666A 6300F 24G CL4 PoE 4SFP56 Sw") {
 $uname = "admin"       #stated different here because the manager account is different
 $ipFile = "conf`/" + $ip + ".conf"       #stated different because switch uses /, not \
#use kitty program via ssh and execute the commands to copy startup config
  .\kitty.exe $ip -ssh -l $uname -pw $pword -cmd "\p
#       kitty.exe - command line executable program that can parse thru values in a command
#       $ip - IP address
#       -ssh - use the SSH protocol
#       -l - login with user
#       -pw - use password
#       -cmd - execute the following command sequence of characters
#               " - starts the command sequence of characters
#               \p - insert a pause before continuing with the command
#...
```

PowerShell Backing up Switch Configurations

```
#...

copy startup-config tftp://$server/$ipFile cli\n
#               copy startup-config tftp://$server/$ipFile cli - switch command to copy startup-config vi cli
#               \n - return
 Exit\n"
#               Exit - switch command to end session
#               \n - return
#               " - ends the command sequence of characters
 }


#note: you do not have to put Exit\n on it's own line, it's just makes it easier to read
#...
```

PowerShell Backing up Switch Configurations

```
#...

#add in a delay between connections
 start-sleep -seconds 10
}
```

PowerShell Updating Switch Firmware

#...

Just add in the line to flash the firmware

```
if ( $line.Model -eq "Aruba-2930F-24G-740W-PoEP-4SFPP" -or `
     $line.Model -eq "Aruba-2930F-48G-740W-PoEP-4SFPP" -or `
     $line.Model -eq "Aruba-2930F-24G-PoEP-4SFPP" -or `
     $line.Model -eq "Aruba-2930F-48G-4SFPP" -or `
     $line.Model -eq "Aruba-2930F-8G-PoEP-2SFPP") {
 .\kitty.exe $ip -telnet -cmd "\p$uname\n$pword\n\pcopy startup-config tftp $server $ipFile\n
  copy tftp flash $server WC_16_11_0014.swi primary\ny\nlogout\nyes\n"

 }
```

#Note: moved the logout\nyes\n to the last line.

PowerShell - Let’s bounce a switch port

```
#set variables
$username = "admin"
$password = "admin123"
$switchInventory = "c:\NetAdmins\Scripts\Switches\SwitchInventory.csv.bak"
$columnName = "IP Address"
$switchIP = Read-Host -Prompt "Provide the switch IP address."
$switchPort = Read-Host -Prompt "Provide the switch port #."
$switchInfo = Import-Csv $switchInventory | Where-Object { $_."IP Address" -like "*$switchIP" }
$switchModel = $switchInfo.Model
##################################################################################


#...
```

PowerShell - Let’s bounce a switch port

```
#...
if ( $switchModel -eq "HP-2920-24G-PoEP" -or `
    $switchModel -eq "HP-2920-48G-POE+" -or `
    $switchModel -eq "HP-2920-48G") {
 .\kitty.exe $switchIP -telnet -cmd "\p$username\n$password\n\p
 config\n
 interface $switchPort\n
 no power\n
 \p\p\p\p\p\p\p\p\p\p
 power\n
 write memory\n
 logout\n
 yes\n"
}
```

PowerShell - Let’s bounce a switch port

```
#####################################################################################
if ( $switchModel -eq "ProCurve Switch 2610-24-PWR") {
 .\kitty.exe $switchIP -telnet -cmd "\p $username\n$password\n\p
 config\n
 interface $switchPort\n
 no power\n
 \p\p\p\p\p\p\p\p\p\p
 power\n
 write memory\n
 logout\n
 yes\n"
}
```

PowerShell 101 - More Cmdlets

New Powershell cmdlets we are going to use:

Select-Object	#pick items to choose to operate with
Out-File	#name of a file you want to send data to

PowerShell Something Completely Different

Using PowerShell to pull data from a CSV file

In this example, we will take CSV files from our Door Access System

We want to know who has 24/7 access

We want to run this monthly

No way do we want to manually look at the raw data in a spreadsheet.

PowerShell 301

Something Completely Different

```
# import csv file skipping first row and set headers of data columns to the named headers"

$f = Import-Csv -Path C:\Scripts\doorAccess\PersonnelReport.csv `
 -Header @("H1","Photo","FName","LName","CardNumber", "Department",`
"GroupName","AccessLevel","Sites","CardStatus") `
 | Select-Object -Skip 2

# Our import file does not have a header row we can use.
# we will skip the first 2 rows by using "| select-object -skip 2"
# and set the header names by using the "-Header @(...)" section

#...
```

PowerShell Something Completely Different

```
#...

#initialize files
$ContractorFile = "C:\Scripts\doorAccess\OutputFiles\ContractorsGroup.csv"
$CustFile = "C:\Scripts\doorAccess\OutputFiles\CustodialGroup.csv"
$DistAdminFile = "C:\Scripts\doorAccess\OutputFiles\DistrictAdminsGroup.csv"
$SROFile = "C:\Scripts\doorAccess\OutputFiles\SROsGroup.csv"
$SuppServFile = "C:\Scripts\doorAccess\OutputFiles\SupportServicesGroup.csv"
$TechFile = "C:\Scripts\doorAccess\OutputFiles\TechnologyGroup.csv"
$TempFile = "C:\Scripts\doorAccess\OutputFiles\TempFile.csv"

#create an array of files used to clear the contents with less code
$filelist = "$ContractorFile","$CustFile","$DistAdminFile","$SROFile","$SuppServFile","$TechFile"

#...
```

PowerShell Something Completely Different

```
#...

#write a header row to reset each file in $filelist
foreach ( $file in $filelist ) {
 write-output "Group,First Name,Last Name,Department" | Out-File $file
}

#write a different header row to our $TempFile
write-output "FName,LName,Department,GroupName,AccessLevel,CardStatus" `
 | Out-File $TempFile



#...
```

PowerShell Something Completely Different

#...

#read each line ($employee) in the file and assign values to variables
```
foreach ($employee in $f) {
 $fname = $employee.'FName'
 $lname = $employee.'LName'
 $dept = $employee.'Department'
 $group = $employee.'GroupName'
 $acl = $employee.'AccessLevel'
 $cardstatus = $employee.'CardStatus'
```

#...

PowerShell Something Completely Different

#...

#take the values above and search for the following groups to pull out and summarize those group members
#employees in a below group will only be added to one group's file (i.e. if I am in Technology and SRO's, my name will only show up in SROs.)

```
 if ( $group -like "*WSD All Doors Contractors 24/7 Access*" ) {
  add-content $ContractorFile "`"$group`",$fname,$lname,$dept"
 }
# $group is string of every building's access level 5a-8p, or 24/7
#-like is used to match the string in the quotes"
#if string is matched, add-content will add "staff info" into variable $ContractorFile
```

#...

PowerShell Something Completely Different

```
#...
 elseif ( $group -like "*WSD All Doors Custodial 24/7 Access*" ) {
  add-content $CustFile "`"$group`",$fname,$lname,$dept"
 }
 elseif ( $group -like "*WSD All Doors District Admins 24/7 Access*" ) {
  add-content $DistAdminFile "`"$group`",$fname,$lname,$dept"
 }
 elseif ( $group -like "*WSD All Doors SROs 24/7 Access*" ) {
  add-content $SROFile "`"$group`",$fname,$lname,$dept"
 }
#elseif's are used as most of the groups we are filtering out are people only in 1 group
#but has ACL's for multiple buildings, so they would also show up under each building
#which does not need to be reported on

#...
```

PowerShell Something Completely Different

```
#...

 elseif ( $group -like "*WSD All Doors Support Services 24/7 Access*" ) {
 add-content $SuppServFile "`"$group`",$fname,$lname,$dept"
}
 elseif ( $group -like "*WSD All Doors Technology 24/7 Access*" ) {
 add-content $TechFile "`"$group`",$fname,$lname,$dept"
}
#If the employee is not a member of the above group and has 24/7 access to 1 or more buildings, they should be added to a temp file to reparse later
 elseif ( $acl -like "*24/7*") {
 add-content $TempFile
"$fname,$lname,$dept,`"$group`",`"$acl`",$cardstatus"
}

#...
```

PowerShell Something Completely Different

```
#...

#next session was put in to show output that failed all the above tests.
 else {

#  $tempvar = $employee.'lname' + "," + $employee.'fname' + "," + $employee.'acl'
#  Write-Output "$fname`t$lname`t$acl"
 }
}

# next part of script is to parse the 24/7 employees that have access to individual buildings,
# and not a global group listed above.
# employees may be a member of more than one group.

#...
```

PowerShell Something Completely Different

```
#...

$f2 = Import-Csv -Path C:\Scripts\DAC\TempFile.csv

#initialize variable for Building files
$BNames =
"AC","ACA","BEC","BPC","BTE","CRE","DRE","DUE","FMS","GTE",`
            "HB","HES","HHS","JES","LVE","LHS","NPH","NPM","PH","PRE",`
            "PVE","SMS","SCE","SS","THS","TRANS","WBE","WMS"
# initialize each file with a header row
foreach ( $building in $BNames ) {
 $file = "C:\Scripts\DAC\" + $building + "_File.csv"
 write-output "AccessLevel,First Name,Last Name,Department,Group"  | Out-File $file
}
```

PowerShell Something Completely Different

```
#...

#$f2 is the TempFile, we are going to process each line
foreach ( $employee in $f2 ) {
#initialize variables
 $fname = $employee.'FName'
 $lname = $employee.'LName'
 $dept = $employee.'Department'
 $group = $employee.'GroupName'


#...
```

PowerShell Something Completely Different

```
#...

#process each building name looking for and ACL mentioned in the User record from $f2
 foreach ( $building in $BNames ) {
  if ( $employee.'AccessLevel' -like "*$building 24/7*" ) {
   $acl = "$building 24/7"
   $file = "C:\Scripts\DAC\" + $building + "_File.csv"
   add-content $file "$acl,$fname,$lname,$dept,`"$group`""
  }
 }                                  #foreach ( $building in $BNames )
}                                   #foreach ( $employee in $f2 )
```

PowerShell Experts Speak Up

Questions???

The 411

Brad Jones

brad.pctech@att.net (StingyPC)
314.378.8936

bradjones@wsdr4.org (Wentzville R-IV School District)
636.332.3751×22334

I want to give thanks to our loving and gracious Creator who has provided me with all the knowledge I have to be able to do what I do.

Thank you God!