# The Basics of PowerShell

Diagram

Description automatically generated

CMDlets are native PowerShell commands that are written in a .net language

Functions – Functions are written in PowerShell’s own scripting language. You can create your own Functions in PowerShell

# Installing and Updating PowerShell

check our version of powershell that we are currently running.  
$PSVersionTable  
Here we see were running PowerShell Version 5.1

## Updating the Help System –

open the console version of PowerShell as an Administrator.

## Commands for updating the Help System:

Update-help -force

(Will download the latest help from the internet everytime you run it) Or you could just run update-help. If you get an error, try running update-help –verbose

Error – Failed to update help for the module WindowsUpdateProvider. You can ignore  
this error.

## Push Help file if no internet

save a copy of the help files from the internet.

Save-Help

Then you run update-help on the computer with no internet connectivity, pointing the  
command to the downloaded copy of the help file .

create a help folder on the C: drive of your VM.  
  
Save-help -DestinationPath “C:\help” -Force -verbose

Open C:\help and checkout all the help files

Now, you will need to get those help files to the computer without an internet connection.  
Just create a help folder on your USB drive and copy all these files to the drive. Now copy the help folder to the C:\ root of the computer that doesn’t have an internet connection

Update-Help -Sourcepath “c:\help” -Force -Verbose

Error Failed to update help for the UI (user Interface) or the English Language) That means that there is not currently a help file for windowsupdateprovider.

copying these help files into the help system.

# The Text Based Console - Part 2 - An Overview

 16 lucida screen text green

Pin ise to task, shift+right click properties>shortcut>adv>run as admin>

# 11. Part 1- Getting Help and Finding Commands

## cmdlet

Verb-noun

Get add export import

## Help System

Get-Help Get-Service -detailed

-examples

-detailed

-full

-online

Here is a list of all four parameters.  
• -examples – Basically gives you examples only. 11 examples really makes learning PS  
much easier.  
• -detailed – Syntax, parameters, examples  
• -full – Syntax, parameters, pipeline, aliases, examples, related links  
• -online – Takes you to the on-line help page

# 12. Part 2 - Getting Help and Finding Commands

get-verb

Get-Command to show commands

Add cursor to front of cmd and hit f1 for windowed help

To find all commands with a word

> help \*service\* \*process\* \*log\*

## Help examples

Get-Help Clear-EventLog -Examples

Get-Help Start-Process -Examples

All commands that begin with an “a”

Get-Command -Name a\*

Help get-a\*

# 13. Part 3 Getting Help and Finding Commands

* Remember to use F1

# 14. Help System Questions

* Get the dependent services of the service you choose
* List all logs on computer
* Get all types of get-command
* Get content
* Clear history

## Get-alias

* Show alias for “ G\* “
  + Get-Alias -Name g\*

Show alias for object

* Get-Alias -Definition Get-ChildItem

# 15. Help System Answers 1 - 4

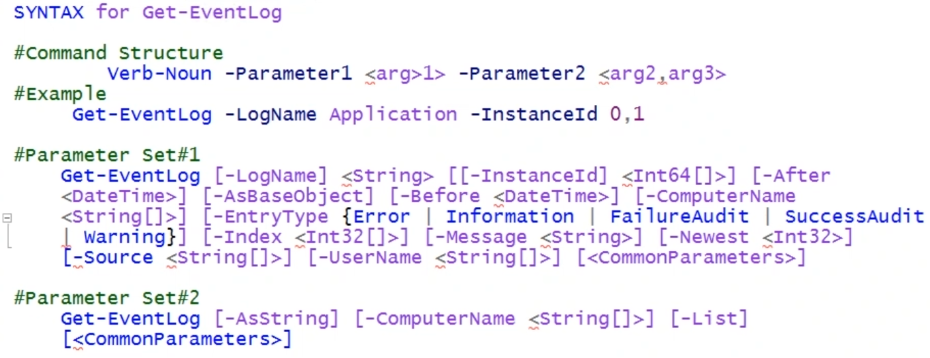
* Get-Service “BFE” -DependentServices
* Get-EventLog -List

# 16. Help System Answers 5 - 9

Shows output of service debug

Start-service bits -Passthrough

# 19. Command Syntax Part 1



Set 1 params don’t work with set 2 unless explicitly stated.

If <int64[ ] > square brackets in angled brackets you can use comma to separate

So -Newest <int32> uses 1 integer

“Don’t wait. The time will never be just right. Start where you stand, and work whatever tools you may have at your command and better tools will be found as you go along.”

# 20. Command Syntax Part 2

Text

Description automatically generated with medium confidence

## Optional

Parameters in SQUARE brackets are optional

## Required

Angled (args) are required

To square brackets surrounded by angled means they can be separated by comma

## Positional

In help window, see parameters, see position

0,1, named

0 and 1 refer to the actual order the cmdlets need be placed

Graphical user interface, text, chat or text message

Description automatically generated

## Named

Can be placed in an order in the cmdlet

## Curly

Options for params, can be comma-space separated



# 21. Command Syntax Part 3

# 22. Command Syntax Questions

# f23. Command Syntax Answers

A picture containing timeline

Description automatically generated

# Objects-Properties and Methods - Part 2

Get-Process -Name notepad | gm

Get member

Tells type name, this is the information going down the pipeline

Type: System.diagnostics.process

We are going to use process

Method = action

To use a method of an object, put cmdlet and arg in para’s. and must use ending para even if empty

(Get-ChildItem C:\path\y.txt).CopyTo(“C:\path.x.txt”)

(get-process notepad).kill()

# Objects-Properties and Methods - Part 3

Get-ChildItem | gm

Look at property, most common way to get value of a property of an object is dot method

Surround parameter and path in para.property

(Get-ChildItem $PSHome\powershell.exe).CreationTime

Help select-object -showwindow

Get-eventlog -logname security -newest 6 | gm

Select property

Get-eventlog -logname security -newest 6 | select-object source, timewritten, machinename, methods

# 32. The Pipeline Part 1

## Remove \*.txt in sub dir and if found ask.

Get-ChildItem C:\Company\\*.txt -Recurse | Remove-Item -WhatIf

Get-childitem – Is like using dir command

I created two folders C:\Company and C:\Company\HR

\*.txt – The files to look for

-Recurse – tells PowerShell to Search Subdirectories, here’s our pipe operator.

Remove-item – Equivalent to delete  
-whatif – Test but won’t complete our command. Press Return

Now let’s type the same command but this time we’ll use the -confirm parameter

## To remove with q & a

Get-ChildItem C:\Company\\*.txt -Recurse | Remove-Item -confirm

## Output to file

get-eventlog -logname application | out-file c:\app.txt

## Read Contents

get-content -path c:\app.txt

## which commands will work together on the pipeline

get-service -name bits | Stop-service

## how does PowerShell know which commands will work together on the pipeline

To make this happen there only four of pieces of information that PowerShell needs.

1. Command #1 We’ll look at the TypeName: (using get-member)

Text, email

Description automatically generated

1. Can Command #2 use the same TypeName as Command #1 (use get-help)
2. Does Command #2 accept pipeline input (use get-help)

4. Can Command #2 accept pipeline input (ByValue or ByProperty

Text

Description automatically generated

# 33. The Pipeline Part 2

use notepad to create a csv (Comma separated value) file and copy that to our test folder.

**The csv headers must match the accepted properties of command 2**

Now type the following:

Name, Value  
L, eventlog  
List, get-childitem  
P, ping  
W, winver

Save file as “All Files”

the parameter binding process that PowerShell uses to determine what commands pass through the pipeline is basically the same for byproperty and for byvalue, with a few differences.

For command #1 you still use get-member and for command #2 you still use get-help. The difference is that now you are looking for things that are byproperty instead of byvalue.

• Command 1 which in this case is import-csv. We’ll **use get-member to get the property and methods of the object import-csv**.

import-csv -path c:\test\aliases.csv | gm

• Command 2 will be New-Alias. We’ll us get-help –full to get the parameters that take byproperty

open two PowerShell windows

Objects can have properties or methods. But what we’re interested in is properties and in this case Powershell displays them as noteProperty

• (NoteProperties are just generic properties that are created by Powershell)

• Notice that Name and Value match the column headings from our csv file.

use get-help to determine if Command 2 (New-alias) has a parameter that will bind with command 1 ByProperty

get-help new-alias -full

A picture containing graphical user interface

Description automatically generated

## Example 2

Create a new csv, notice name in csv header and stop service property -name, and real service



Name

Bits



# Powershell Providers

## How to Use PSDrive

The PSDrive cmdlet allows you to view, create and remove  
PowerShell drives. A PSDrive is considered a data store location that can represent the file system, a registry  
hive and network share, among other things as well.

open PowerShell as the current user. We will use the cmdlet New-PSDrive to create a

temporary or persistent drive that is mapped to or associated with a location in a data store

Get-Help New-PSdrive

use the cmdlet New-PSDrive to create a  
temporary or persistent drive that is mapped to or associated with a location in a data store

## how to use New-PS drive to map that registry location to a name. And make it assessable like any file system drive

New-PSDrive -Name PSReg -PSProvider Registry -Root HKLM:\SOFTWARE\Microsoft\PowerShell\3\

parameters

-Name specifies a name for the new drive

-PSProvider this shows that the drive is associated with the registry.

-Root specifies the data store location to which a PowerShell drive is mapped.

-HKLM is the target registry hive.

To access the newly created PSDrive Type “cd psreg: && dir”

## Additional cmdlets

Get-PSDrive

This command that gets the drive in the current session.

New-PSDrive

creates a PowerShell drive that's mapped to a location in a data store, such as a network drive, a directory on the local computer, or a registry key.

Remove-PSDrive.

This cmdlet deletes our PowerShell drives that were created by using the New-PSDrive cmdlet

## Create a PSDrive and map it to a local folder.

New-PSDrive -Name "Win" -PSProvider "FileSystem" -Root “C:\windows\media

Parameters

-Name specifies the name for the new drive

-PSprovider it'll be file system

-root will be the folder C:\ windows\ media.

To access your new PSDrive, type CD space win: then type dir.

\*\* One important point to note here is that all your PSdrives are non-persistent and that means they'll

disappear as soon as your session is closed.

To make your PSDrive persistent and last even if your session is closed, you can map the PSDrive to

appear in Windows Explorer using the switch -persist

## psdrive (loopback/network share persist)

New-PSDrive -Name 'L' -PSProvider FileSystem -Root '\\127.0.0.1\C$\$Recycle.Bin’ -persist

parameters

-name specifies the name for the new drive. L.

-PSProvider is associated with the file system

-Root we're using 127.0.0.1 and what this means is the root parameter can be associated with a network share. But in this case, we're using  
the loop back IP address for the local machine.

C$ represents the local C:\recycle folder.

-persist maps the drive to Windows Explorer.   
To access the drive type cd space L: and type dir and press return.  
And there's our files.

Now close the PowerShell session and open Windows Explorer. You should see L: as your new  
persistent PowerShell drive with all the files and folders.

• Here's a tip if you recall. I opened PowerShell as the current user and not as the administrator. In this case, your mapped drives will only appear in Windows Explorer when you're logged into Windows as  
the current user.  
• Now let's get back into PowerShell. To remove our persistent PSDrive, go ahead and type Remove-PSDrive -name L and press return. Now, if we open windows explorer. We see that our L  
drive has been removed.

## Powershell provider cmdlets

### Stack locations

Diagram

Description automatically generated with medium confidence

From your C: drive create a folder called books, then create the four sub-folders. Stop the video  
while you do this.

Now lets add all four folders to the stack by using our pushd alias cmdlet  
Type pushd c:\books\1984 (pressh return)  
Type pushd c:\books\gulivers\_travels  
Type pushd c:\books\hamlet  
Type pushd c:\books\moby\_dick

Now let’s view the stack by using the get-location -stack (parameter)  
remember (LIFO) last in first out

Type get-location -stack (press return)  
Because get-location displays our current location, moby\_dick is in the  
stack but not shown in the stack.  
Now from PS C:\books\moby\_dick location type popd  
Type get-location -stack (moby\_dick) has been removed from the stack)  
From PS C:\books\Hamlet location type popd  
Type get-location -stack now hamlet has been removed from the stack  
From PS C:\books\Gulivers\_Travels location type popd  
Type get-location -stack (gulivers travels has been removed from the stack)  
From PS C:\books\1984 location type popd  
Type get-location -stack (1984 has been removed from the stack)  
Type get-location -stack and now the stack is empty.  
As you can see we can move between these four folders and easily remove the folders from the stack that are no longer need

### Search for all the DOCX files in the Documents and settings folder and subfolders

get-childitem -path ‘C:\Documents and Settings’ -include "\*docx" -Recurse -file

Parameters:

-Path This parameter specifies the path of one or more locations.

-include This is a string parameter and when this parameter is used, it displays specific files

and folders.

-recurse This parameter instructs powerShell commands such as Get-ChildItem to repeat in

sub directories.

-file The file attribute gives an output of only files under that container

### New-Item –

Creates a new item and sets its value. This can be a folder, file or multiple files inside a directory. Use get-help New-item to view all the parameters.

#### Create a folder on your C: drive called PSTest

New-Item -Path C:\ -Name “PSTest” -Itemtype “directory”

For Parameters

-Itemtype Can be a file or a folder.

We can check for the folder in windows explorer.

#### Create a file w/ txt to the file.

New-Item -Path C:\PSTest -Name “test1.txt” -Itemtype “file” -Value “Hey boss, I want a

50% raise.”

Parameters

-Value allows you to add txt as a value.

open windows explorer again, you can click on test1.txt and checkout our value Hey boss I want a 50% raise.

#### overwrite an existing file using the -confirm parameter.

New-Item -Path C:\PSTest -Name “test1.txt” -Itemtype “file” -Confirm

The result is that the -confirm parameter displays a warning, and asks you to confirm this.

• If you want to overwrite the existing file just add the -force parameter.

#### create multiple files in a designated folder.

New-Item -ItemType "file" -Path "C:\PSTest\child1", "C:\PSTEST\child2"

Now from Windows Explorer if we open the PSTest folder there are our two files that we just created.

## Content Cmdlets

Now let’s take a look at Content cmdlets. These cmdlets Append, clear, get and replace the

content of a file.

### Add-Content – Appends content to a file.

• From the PSTest folder on your C: drive create a file called test.txt, use the command New-Item to create the file.

New-Item -path "C:\PSTest" -name "test.txt" -itemtype "file"

Use the add-content cmdlet to append the following text to the file.

Add-content "C:\PSTest\test.txt" -Value "Let's go to lunch by noon"

### delete the contents of the test.txt file using the clear-content cmdlet.

Clear-content "C:\PSTest\test.txt"

Now from Windows Explorer, If we take a look at test.txt we can see that the contents have been cleared.

# Powershell provider cmdlets2

In this lecture we will take a look at Item property and path cmdlets.

## ItemProperty cmdlets

• New-ItemProperty - creates a new property for a specified item and sets its value.  
Typically, this cmdlet is used to create new registry values.

• Get-itemProperty gets the property of an item like viewing registry properties and their  
values. You can use this cmdlet to get information about directories, files or registry  
entries.

• Remove-ItemProperty – This deletes a property and its value from an item. You can use  
it to delete registry values and the data that they store.

Please note: That the registry entry must exist before you can edit the entry.

First, we need to create a new registry key in the HKLM registry hive. We will use the New-  
item cmdlet.

In the file system, New-Item creates files and folders. In the registry, New-Item creates registry keys and entries.

Be sure to open PowerShell as an Administrator.

## create a registry key named MyCompany.

New-Item -Path "HKLM:\Software\MyCompany"

Displayed is our registry key MyCompany

## add a new registry entry to the MyCompany registry key.

New-Itemproperty -Path "HKLM:\Software\MyCompany" -name NewProperty -value  
NewPropertyValue

We just added the name NewProperty to the registry.

## add a new value of 400 employees to my company.

New-itemproperty -path HKLM:\Software\MyCompany -name NoOfEmployees -value 400  
  
  
Let’s take a look at the results by taking a look at the windows registry editor.  
• From the search bar type registry  
• From HKLM, follow the path software\MyCompany  
• Click MyCompany  
And there is our registry key and our entries.

Now let’s take a look at the results by using the get-itemproperty cmdlet.  
  
Get-itemproperty hklm:\software\mycompany  
  
And you can see all our entries.

## remove the registry value NewProperty

that we created using Command #2  
Command#5

Remove-Itemproperty -Path "HKLM:\Software\MyCompany" -name NewProperty  
  
Now run Command #4 again and notice that the NewProperty value has been removed.

## remove the MyCompany registry entry by using the cmdlet remove-item

Command #6  
Remove-Item -Path "HKLM:\Software\MyCompany"  
  
Rerun Command #4 and you will see that MyCompany has been removed.

## Path cmdlets

Now let’s checkout several of the path cmdlets

### Test-Path

- Determines whether all elements of the path exist.

This cmdlet is handy if you are writing a script and you want to test whether the path to a file is true or false.  
Test path can save you a lot of time.

For Example: Perhaps you need to check to see if the password log is present in the debug  
directory.

Test-Path 'C:\windows\debug\PASSWD.LOG'  
Press return  
The cmdlet returns true, so we know that passwd.log file is in that folder.

### Resolve-path

• Displays the items and containers that match the wildcard pattern at the  
location specified.  
Now let’s say you need to take a look at all the paths in the Windows directory.  
  
Resolve-path "C:\windows\\*"

Notice the wildcard in the command.  
  
This command lists all the folders, subfolders and files in the Windows directory.

### Split-path

Split-path returns a string that describes the location of the items and returns the specified part of a path. Let's say you want to list all the log files in this C:\Windows folder. You can use the split-path cmdlet.  
  
Split-Path -Path "C:\windows\\*.Log" -Leaf -resolve  
  
Now for Parameters we used:  
-Path is all the log files located in C:\windows  
-Leaf This command displays the files that are referenced by the split path. Because this  
path is split to the last item, also known as the leaf, the command displays only the file  
names.  
-resolve The resolve parameter tells Split path to display the items that display pass  
references. Instead of displaying the split path.

# Powershell arrays and Variables

## Powershell Variables

Store a single element

### Variable a process and select property

$process = Get-Process

$process | Select-object id, processname ( lists by header)

### Variable user input

$UserInput = Read-host -prompt “please enter your name”

$UserInput > file.txt

### Variable user input to save in file

$UserInput = Read-host “please enter your name”

Set-content -Value $userinput -path file.txt

Cat file.txt

# Powershell Arrays

Can store multiple elements

$myarray = @(“elm0”, “elm1”, “elm2”)

$myarray[0]

$myarray[1]

## Show all variables

$myarray

## Empty array

$myarray = @()

## Add to array

$myarray += “apple”

$myarray

$myarray += @(“peppers”,”olives”)

$myarray[1]

$alpha =@(“c”,”d”,”a”,”e”)

$alpha

## Modify the output

Alphabetical order

$alpha | sort

Reverse Alphabetical order

$alpha | sort -descending

## Modify the variable

$alpha = $alpha | sort

## Shows line with no peppers, array has not yet been modified

* $\_ means that the beginning of what is in pipeline is called

$Myarray | where {$\_ -ne “peppers”}

## Remove item from array

$Myarray = $Myarray | where {$\_ -ne “peppers”}

## Remove item based on array position

$Myarray = $Myarray | where {$\_ -ne $Myarray[1]}

## Empty the array

Assign array contents to null

$Myarray = $null

# PowerShell Loops

## For Loops

Based on AMOUNT of times

For ($i=0; $i -lt 5; $i++) {

Echo “In a loop: $i”

}

## Loop an array

### For a specific amount of 5 times, listed by $i -lt 5, can lead to miss count

$Myarray = @(“cars”,”trucks”,motrcys”,”bikes”)

For ($i=0; $i -lt 5; $i++) {

Echo (“el $i loop: ” + $Myarray[$i])

}

### For the amount of items in the array

$Myarray = @(“cars”,”trucks”,motrcys”,”bikes”)

For ($i=0; $i -lt $MyArray.count; $i++) {

Echo (“el $i loop: ” + $Myarray[$i])

## Do Loop

$i=0

do

{

$i++

}

while ($i -eq 5)

echo "value of i = $i"

### DoWhile/Until

$i=10

do

{

$i++ #condition here get ran until

}

until #until or while get checked after the do code block

($i -lt 20)

echo "value of i = $i"

## While Loop

### Based on while condition met

**Example 1**

while ((Read-Host "type 'xyz' to stop loop") -ne "xyz")

{

}

**Example 2**

### While, if, elseif w/ userinput

\* #You can only run “else” if you don’t have parenthesis as a condition

while ($userinput -ne "quit") {

# Gather user input

$userinput = Read-Host "say 'yes' if you like scripting (enter 'quit' to stop the loop)"

#eval user input

if($userinput -eq "yes") {echo "I also love scripting! That's cool" }

elseif($userinput -eq "no") {echo "I hate scripting! That's not cool" }

}

if($userinput -eq "quit") {echo "Your Exiting Quiter" }

$userinput = $null

## ForEach Loop

$People = @("P hill", "j hill", "p lopp")

foreach ($person in $People)

{

echo "Creating new AD for $person"

#New-ADUser

}

# Loop Breaks/continue

Try to use break or continue in the if statements

## Break

$People = @("P hill", "j hill", "p lopp")

foreach ($person in $People)

{

if ($person -eq "p lopp") { break } #exits a loop, and quits when the specific element is found.

$person

}

## To skip a command/ element your “search” or “Foreach” loop

$People = @("P hill", "j hill", "p lopp")

foreach ($person in $People)

{

if ($person -eq "j hill") { continue } #skips element and continues on after.

$person

}

# Functions and Conditional Statements

## PowerShell Conditional Statements (If Then Else, Switch)

### Example 1 if

if (1 -lt 3){

echo "hotdogs" # if rings true

}else{

echo "no hotdogs" # if rings false

}

### Example 2 variable conditions

$var1 = 1

$var2 = 1

if ($var1 -eq $var2){

echo "hotdogs" # if rings true

}else{

echo "no hotdogs" # if rings false

}

### Example 3 user options

ElseIf in a if block:

the code will stop doing comparisons immediately when the elseIf rings true.

“Ifs” will continue to go through and check comparisons if one rings true

Else is a catch all

$choice = Read-Host "Please select an option(1-3)"

if ($choice -eq 1 ){

echo "you chose option 1" #

} elseif ($choice -eq 2) {

echo "no hotdogs" # if rings false

}

elseif ($choice -eq 3) {

echo "no hotdogs"

} elseif ($choice -eq 4) {

echo "numero 4"

} else {

echo "you didnot choose wisely"

}

Instead of multiple elseif’s, use the switch statement

### Switch

This is performs the exact same thing as above

switch (Read-Host "Please select an option(1-3)") {

1 {"you chose option 1"}

2 {"you chose option 2"}

3 {"you chose option 3"}

default {"you chose option badly"}

}

## Understanding PowerShell Functions and Function Scope

Graphical user interface, text

Description automatically generated

### Simple Function Declaration

Graphical user interface, text, application, email

Description automatically generated

### Function Scopes

Graphical user interface, text, application, email

Description automatically generated

The output of var will be empty because the variable is declared inside the function

### Available Function Scopes

Graphical user interface, text, application

Description automatically generated

### Declare a function scope

To declare a function scope use

Function scope:function\_name

Graphical user interface

Description automatically generated with medium confidence

This will not make your variables accessible by the outside. Just makes it a global function

## PowerShell Functions

If you call a function before the function you will get an error message.

If you run the script again it will work as the function is loaded into memory.

You can also call the function name and it will run.

\*Call the functions after you declare.

### Pass an argument to a function

#### Example 1

Function EchoText {

echo $args

}

EchoText "i am passing an arg"

#### Example 2 multiple args/array

Function Text {

echo ("Argument: " + $args[1] + " " + $args[0])

}

Text "this is the first arg" "this is the second arg"

#### Example 3

Function add {

echo ($args[0] + " " + $args[1])

}

Add 10 5

#### Example 4 Parameters ex1

This is different because we can specify the order we want to pass parameters to,

Oppose to arguments, firsdt arguments will always be first.

* Need better explanation than above

Function text {

param (

$FirstParameter,

$SecondParameter

)

echo ("first: " + $FirstParameter)

echo ("Second: " + $SecondParameter)

}

text -SecondParameter "this is the second" -FirstParameter "This is the first"

#### Example 5 Parameters ex2

Function text1 ($FirstParameter,$SecondParameter) {

echo ("first: " + $FirstParameter)

echo ("Second: " + $SecondParameter)

}

text1 -SecondParameter "xyz" -FirstParameter "abc"

# Basic Domain administration

Configuring ps execution policy with group policy

You can write them but will be blocked once saved.

Graphical user interface, text, application

Description automatically generated with medium confidence

Get-ExecutionPolicy -list

All execution policies

Undefined means it defaults to the value of restriction

It shows that there should or should not write any scripts

Tools>gpo manament > domains > place gpo needs to be linked to something that has computers inside of the OU or it has the target computers.

Right click, new gpo,

Name: powershell script execution > right click>computer config

So group policies, administrative templates, windows components and scroll down.

And we're looking for Windows Power Show,

Double click turn on script execution, enabled, choose allow all scripts.

close out of this group policy management,

choose computer power, shell script execution, click on settings,

generate a report here, we're going to look on our computer configuration, we're going to expand administrative templates, Windows components, scroll down here, we can see that we have the execution policy now configured to allow all scripts.

run a group policy update on our machine, close out of this group policy management window.

Gpudate /force

Logoff and sign back in

Get-ExecutionPolicy -List

Run a .\script

To see what gpo is being applied to the computer Rsop.msc>comp configuration>admin templates>winComponets>win powershell

Can also

Set-ExecutionPolicy Unrestricted -force -scope CurrentUser

\*Group policy always parents over local policy

## Execution Policy Precedence

Text

Description automatically generated

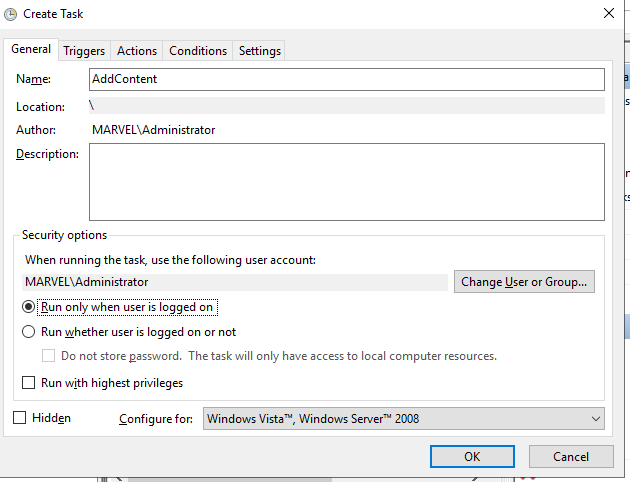
## Installing and configuring server roles

Get-Command | Select-String “feature”

## Running Powershell Scripts as Scheduled Task

Create a ps1 script that does something

Go to scheduled task



Create a different account that’s not admin and not your account. Least amount of privs to run the script

Change user to that different account

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Right click hit run to start task

## PowerShell Splatting

Creating a variable that contains all these different keys and there values and you can access these parameters at a later time

Graphical user interface, text, application, email

Description automatically generated

To grab a parameter info

$parameters.EmailAddress

Add new user with defined options

New-ADUser @parameters

If you have an incorrect parameter, the code will fail.

# Administrating AD

## Creating OUs (Organizational Units) with PowerShell

### Find all OU and see objects

Get-ADOrganizationUnit -Filter “\*”

### Format-Table and include what objects you want to include

Get-ADOrganizationUnit -Filter “\*” | Format-Table Name, DistinguishedName

### Find specific unit per OU

find all of the organizational units that are directly beneath a OU organization

Get-ADOrganizationUnit -Filter “\*” -SearchBase “OU=dc,dc=domain,dc=com”

| Format-Table Name, DistinguishedName

\*Search base DistinguishedName of OU “Name” you are looking for “OU=dc,dc=domain,dc=com”

There are different levels of scope to find Org units

Sub-tree, one level, base

### Grab units one level below top OU

Get-ADOrganizationUnit -Filter “\*” -SearchBase “OU=dc,dc=domain,dc=com” -SearchScope OneLevel

| Format-Table Name, DistinguishedName

### Create a new OU

New-ADOrgUnit -Name “Test123”

New-ADOrgUnit -Name “Test123” -Protectefromautodeletion $False

### Remove an OU

Get-ADOrganizationUnit -Filter “\*” -SearchBase “OU=dc,dc=domain,dc=com” -SearchScope OneLevel

| Format-Table Name, DistinguishedName

Get DN

Remove- ADOrganizationUnit -Identity “OU=dc,dc=domain,dc=com”

Y = yes to all

check

Get-ADOrganizationUnit -Filter “\*” -SearchBase “OU=dc,dc=domain,dc=com” -SearchScope OneLevel

| Format-Table Name, DistinguishedName

#### \*\*\* If protected run

Set-ADObject -ProtectedFromAccentdel $False -Identity ““OU=dc,dc=domain,dc=com”

Try again

Remove- ADOrganizationUnit -Identity “OU=dc,dc=domain,dc=com”

check

### Add OU to specific ou

New-ADOrgUnit -Name “Test123 -Path “OU=dc,dc=domain,dc=com”

## Creating and Managing Active Directory User Accounts with PowerShell

## Move all AD Users in a Group to a Specific Organizational Unit

Use case we need to move 100’s of users to a group ou and you don’t know where all the users are located.

Move user Administrator in Users folder that is in DA group to the DA OU

-Recursive searches nested objects

Powershell

Import-Module ActiveDirectory

Get-ADGroupMember “domain admins”

Get-ADGroupMember “domain admins” | select-Object SamAccountName

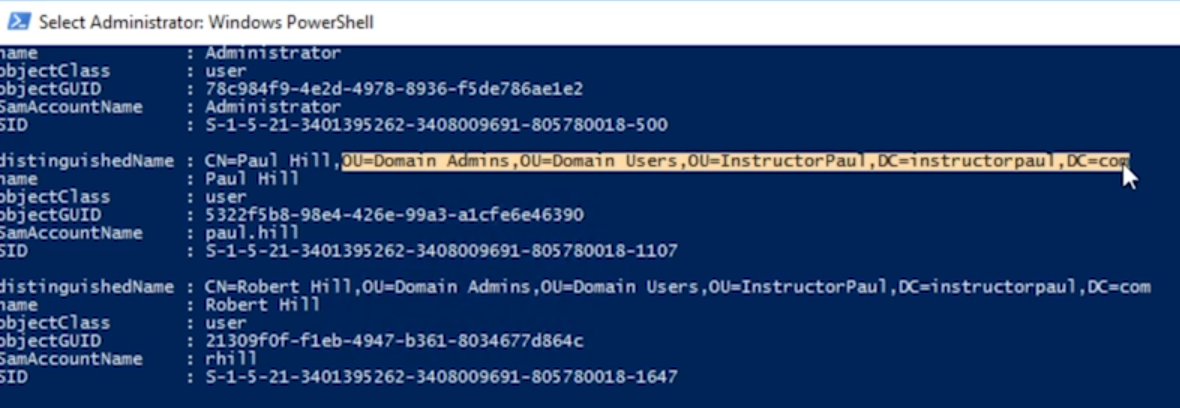
Get specific group sam and DN

Get-ADGroupMember “domain admins” | select-Object SamAccountName, distinguisgedname

Save to text file

Get-ADGroupMember “domain admins” | select-Object SamAccountName, distinguisgedname > domain.txt

Grab everything to the right of the C Line of local user Administrator



Get-ADGroupMember “domain admins” | move-ADObject -TargetPath “OU=dc,dc=domain,dc=com”

Check that local admin is in DA groups

Get-ADGroupMember “domain admins” | select-Object SamAccountName, distinguisgedname