|  |  |
| --- | --- |
| Get-Command \*naam\* |  |
| Get-TimeZone |  |
| Get-Help | (stop-process) of iets |
| Get-date | Get-member | |Format-list |
|  | -Examples |
| Find-Module -Tag Telegram |  |
|  |  |
| Get-ADUser  Get-ADGroup  Get-ADObject  Get-ADComputer | -AuthType <ADAuthType>  -Credential <PSCredential>  -Identity] <ADUser>  -Properties <String[]> |
|  |  |
| Get-Command -Module ‘IISAdministration’  Get-IISAppPool  Get-IISServerManager  Get-IISSite -Name ‘naam’  (Get-IISSite -Name ‘naam’).Bindings  Get-IISSiteBinding “naam” | Stop-IISSite -Name “naam”  Start-IISSite -Name “naam” |
|  |  |
| Get-NetFirewallRule | -DisplayName <String> |
|  |  |
| Test-NetConnection 10.63.1.4 -Port 23 |  |
| Test-NetConnection -Computername 130.37.162.107 -Port 443 |  |
|  |  |

|  |  |
| --- | --- |
| SELECT expressions  FROM tables  [WHERE conditions]  [ORDER BY expression [ ASC | DESC ]]; | SELECT \*  FROM customers  WHERE favorite\_website = 'techonthenet.com'  ORDER BY last\_name ASC; |
|  | SELECT orders.order\_id, customers.last\_name  FROM orders  INNER JOIN customers  ON orders.customer\_id = customers.customer\_id  WHERE orders.order\_id <> 1  ORDER BY orders.order\_id; |
|  | SELECT FirstName, LastName, StartDate AS FirstDay  FROM DimEmployee  WHERE EndDate IS NOT NULL  AND MaritalStatus = 'M'  ORDER BY LastName; |
|  | SELECT OrderDateKey, SUM(SalesAmount) AS TotalSales  FROM FactInternetSales  GROUP BY OrderDateKey  ORDER BY OrderDateKey; |
|  |  |

PowerShell IT Managment

<http://aka.lod>

<https://mslearningcampus.com/ClassEnrollment/2600228>

<https://docs.microsoft.com/en-us/powershell/scripting/learn/remoting/jea/overview?view=powershell-7.1>

TCP: 5985 and TCP: 5986 for an SSL connection.

Set-NetFirewallRule -Name 'WINRM-HTTP-In-TCP' -Enabled True

**HELP:**

Review the help for Enter-PSSession by running this command:

Get-Help -Name Enter-PSSession -ShowWindow

Notice the parameter ComputerName, by running this command:

Get-Help -Name Enter-PSSession -Parameter ComputerName

**Remote WinRM:**

Check WinRM Service using Get-Service and notice on the Status and StartType.

Get-Service -Name WinRM | Select-Object -Property Name, Status, StartType, DisplayName

Execute the following command:

Enable-PSRemoting

Confirm that WinRM Service is configured using:

Test-WSMan

Get-Service -Name WinRM | Select-Object -Property Name, Status, StartType, DisplayName

**Remote Connection:**

Connect to member server MS interactively, by running this command:

Enter-PSSession -ComputerName MS

Confirm the computer to which you are connected using with:

PowerShell command: $env:COMPUTERNAME

External command: hostname.exe

Confirm credentials used with:

PowerShell command: $env:USERNAME

External command: whoami.exe

Exit the current session by running this command:

Exit-PSSession

**Using the WIN10 machine, connect to domain controller DC interactively using the domain administrator credentials by running this command:**

Enter-PSSession -ComputerName DC -Credential Contoso\Administrator

Confirm the computer to which you are connected using from within the Windows PowerShell ISE:

PowerShell command: $env:COMPUTERNAME

External command: hostname.exe

Confirm that you are connected using domain administrator credentials:

PowerShell command: $env:USERNAME

External command: whoami.exe

**Declare variable $process with value System by running this command:**

$process = 'System'

Get the System process on remote computer MS using Invoke-Command cmdlet, by running this command:

Invoke-Command -ComputerName MS -ScriptBlock {Get-Process -Name System}

**Type:**

You can look at types by just typing them in square brackets ('[]') PowerShell and use the pipeline to format all properties into a list. Note that some types have aliases called Accelerators that enable you to simply type the name of the type. Other you have to specify the fullname including the namespace.

[string] | Get-Member # This is the fully qualified name (i.e. Namespace.Class)

[System.String] # This will allow you to see the Fullname for an accelerator type.

[System].Fullname

# This will show you the assembly (dll) in which the class/type is defined.

[Microsoft.PowerShell.Commands.LocalGroup].Assembly

# You can also see the order in which assemblies were loaded into PowerShell

# NOTE that assemblies can have dependencies on other assemblies which will be loaded as well.

[System.AppDomain]::CurrentDomain.GetAssemblies().Modules.Name

Look at the dll property of the Get-LocalGroup command. Where is it loaded from (Get-Module -Name (Get-Command Get-LocalGroup).ModuleName -ListAvailable) | format-List -Property \*

The command get-localgroup is implemented in a Module. PowerShell by default will import the module and in doing so will load all assemblies specified in it's manifest.

Because the module is now imported which in turn loads assemblies in which the type is define.

**By getting all members of the local administrators group and assign them to a variable called $admins:**

$admins = get-localgroup -name 'administrators'

Use the force switch parameter of get-member to list all members of the objects stored in $admins.

$admins | Get-Member -Force

**Let us take a quick look at a few properties of the extensible PSObject in the case of a Service object. Execute the following commands to see how objects/classes are extended:**

# Get a ServiceController Object  
$BitsService = Get-Service -Name Bits

# Look at the hidden member type MemberSet   
Get-Member -InputObject $BitsService -MemberType MemberSet -force

# The .NET object abstracted by PSObject  
([System.ServiceProcess.ServiceController]'Bits').psobject.BaseObject

# The members added to the base object  
([System.ServiceProcess.ServiceController]'Bits').psextended

# The base objects members  
([System.ServiceProcess.ServiceController]'Bits').psbase

# The base + extended members jointly called adapted.  
([System.ServiceProcess.ServiceController]'Bits').psadapted

**Hashtable Object which you will populate with Key/Value pairs. Create a variable $PropertiesHashTable as a hashtable with the following data:**

$PropertiesHashTable = @{ FirstName = 'Janet'; LastName = 'Shearon'; Country = 'USA'; state = 'Illinois' }

When you cast a hashtable to a PowerShell PSCustomObject type the hashtable's keys are added to a PSObject as members of type NoteProperty adn the hashtables values are assigned to the corresponding proprorties as their values. Execut the code below to extend a PSObject aka PSCustomObject. Even though it might look like an accelerator, this is just a keyword for the parser to parse the data. It will not work to cast using the fullname (i.e. [System.Management.Automation.PSCustomObject]$PropertiesHashTable). This is PowerShell magic behind the covers which was introduced in PowerShell v3.0.

$MyPSv3Object = [PSCustomObject]$PropertiesHashTable

$MyPSv3Object

In all versions of PowerShell you can enhance objects with PowerShell member types using the add-member cmdlet.

$MyPSObject = New-Object -TypeName System.Management.Automation.PSObject

Add-Member -InputObject $MyPSObject -Name 'FirstName' -MemberType NoteProperty -Value 'Janet'

Add-Member -InputObject $MyPSObject -Name 'LastName' -MemberType NoteProperty -Value 'Shearon'

Add-Member -InputObject $MyPSObject -Name 'Country' -MemberType NoteProperty -Value 'USA'

Add-Member -InputObject $MyPSObject -Name 'state' -MemberType NoteProperty -Value 'Illinois'

$MyPSObject

Another way to extend an object is via Select-Object which also uses a hashtable to add NoteProperty members. Note: There are two keys: Name has a value with the name of the property and expression has a scriptblock ({}) value will return the value for the property.

$user = $MyPSObject | select-object firstname,lastname,country,state,@{name="displayname";expression={$\_.lastname + " " + $\_.firstname}}

Can you now add the samaccountname to the $user object using the first three letters of firstname property and the first three letters of the lastname property. The names are strings. You can use substring method or array indexing to get parts of a string

$user | select-object firstname,lastname,country,state,displayname,@{name="samaccountname";expression={"$($\_.firstname.substring(0,3))$($\_.lastname.substring(0,3))"}}

Add-member -inputobject $user -MemberType scriptmethod -name 'homefolder' -value {"""\\ms\users\$($this.samaccountname)"""} Note: $this is a variable which points to the actual object itself. This was you can access members of an object when defining new members.

PS C:\WINDOWS\system32> $user | get-member TypeName: Selected.System.Management.Automation.PSCustomObject Name MemberType Definition ---- ---------- ---------- Equals Method bool Equals(System.Object obj) GetHashCode Method int GetHashCode() GetType Method type GetType() ToString Method string ToString() Country NoteProperty string Country=USA displayname NoteProperty System.String displayname=Shearon JanetFirstName NoteProperty string FirstName=Janet LastName NoteProperty string LastName=Shearon state NoteProperty string state=Illinois PS C:\WINDOWS\system32> $user.homefolder

**Create and instance of the WScript.Shell class (using the ProgID) and assign this to a variable $WScriptShell:**

$wscriptshell = New-Object -ComObject Wscript.shell

**Look at the members of the object you assigned to $WScriptShell:**

$wscriptshell | get-member

Let us look at these calling Get-childItem -Path HKCR:\WScript.\*.

Lets now create a variable called $wscriptnetwork and assign it an instance of Wscript.network COM class (i.e. and Object) using the New-Object cmdlet.

Hint

You can create new objects for com classes with the cmdlet new-object

Answer

$wscriptnetwork = New-Object -ComObject WScript.Network

Look at the members of the object you assigned to $wscriptnetwork.

Answer

$wscriptnetwork | get-member

# Start notepad

$Notepad = $wscriptshell.Exec('Notepad.exe')

# Allow time for the process to start

Start-Sleep -Seconds 2

# Ensure that the notepad windows is active before sending input

$wscriptshell.AppActivate($Notepad.ProcessID)

# simulate keyboard input using sendkeys

$wscriptshell.SendKeys("Each on of these characters were sent to Notepad's edit window via the sendkeys method.")

$createshortcut = $wscriptshell.createshortcut('C:\Users\power\Desktop\sysvol.lnk')

Look at the members of the COM object references in $createshortcut.

Answer

$createshortcut | get-member

Set the targetpath as well as some of the properties and finally call the save method to persist the shortcut to disk.

Hint

$createshortcut.targetpath('s:')

Answer

$createshortcut.save()

<https://github.com/vinaypamnani/wmie2>

Lets set up a warning system whenever a route steal occurs.

Create a new event trigger called "Routestealaction". The following actions should be put in motion:

Log an event in the eventviewer as ID 2 with the message "Someone is changing your internet routes."

popup a message saying "Someone is changing your internet routes."

Hint

Think about the COM classes from the first exercises. It has methods to complete the task.

Register-WmiEvent -SourceIdentifier "RouteTableChangeOccured" -Class Win32\_IP4RouteTableEvent -action {

$shell = New-Object -ComObject wscript.shell;$shell.LogEvent('2','Someone is changing your internet routes')

$shell.Popup('Someone is changing your internet routes')

}

Unregister-Event -SourceIdentifier RouteTableChangeOccured  
Get-Job -SourceIndentifier RouteTableChangeOccured  
  
will return a Job that has stopped. To remove it you must call  
  
Remove-Job -Name RouteTableChangeOccured

<https://docs.microsoft.com/en-us/windows/desktop/WmiSdk/wql-sql-for-wmi>

You can also use WQL to start hunting for programs that use a large amount of the Page file, for example.

Get-CimInstance -Query "SELECT \* FROM WIn32\_Process WHERE PageFileUsage > 100000" | Sort-Object -Property PageFileUsage -Descending | Format-Table -Property Name,ProcessId,PageFileUsage

Get-CimInstance -ComputerName $env:COMPUTERNAME -Query "SELECT \* FROM Win32\_Service WHERE Name='BITS'"  
Get-WmiObject -ComputerName $env:COMPUTERNAME -Query "SELECT \* FROM Win32\_Service WHERE Name='BITS'"

What types of object are each of these calls returning and what Methods do each of the Objects expose?

Hint

Use Get-Member -MemberType Methods

Get-CimInstance -ComputerName $env:COMPUTERNAME -Query "SELECT \* FROM Win32\_Service WHERE Name='BITS'" | Get-Member -MemberType methods  
Get-WmiObject -ComputerName $env:COMPUTERNAME -Query "SELECT \* FROM Win32\_Service WHERE Name='BITS'" | Get-Member -MemberType Methods

How can you remotely stop a service using the Cim cmdlets?

Hint

Invoke-CimMethod

Invoke-CimMethod -Query "SELECT \* FROM Win32\_Service WHERE Name='BITS'" -MethodName StopService -ComputerName $env:COMPUTERNAME

You can use the New-PSSession cmdlet to create a new PSSession to a remote computer. Use this command to open a new PSSession to MS and store it in a variable called $Session: $Session = New-PSSession -ComputerName MS

Run Get-PSSession to get a list of all PSSessions in the current session.

Return the value in the $Session variable.

You can now see that a new PSSession has been opened to the remote computer and that the object exists in your $Session variable. Connect to this PSSession using the Enter-PSSession cmdlet while specifying the open PSSession in the $Session variable: Enter-PSSession -Session $Session

Use the New-PSSession cmdlet you learned in the previous task to create a new PSSession to the remote computer MS and store it in the variable $Session.

New-PSSession Cmdlet

$Session = New-PSSession -ComputerName MS

Return the value for the variable $Session and note the properties of State and Availability. What are the values for these properties?

Answer

Currently the session state is Opened and availability is Available. This means that session is ready to be connected to and available to work in.

Connect to the new PSSession by running: Enter-PSSession -Session $Session

Exit the current session by running Exit-PSSession.

Disconnect-PSSession will disconnect a PSSession from the local session but maintain it on the remote computer. Run Disconnect-PSSession -Session $Session to disconnect the PSSession.

When you are finished with the PSSession, run Remove-PSSession -Session $Session to delete the PSSession and release the resources it was using.

Task 2.2 Configure and use SSL endpoint for PowerShell remoting

Right-click the PowerShell icon on the taskbar and select the Run PowerShell as Administrator option. This will launch the Windows PowerShell console with elevated privileges.

Before connecting to a remote computer using an SSL endpoint, the remote computer will need to be configured to accept SSL PowerShell connections. This can be done by logging into the machine directly, or by using the remote PowerShell skills covered previously. Create a new remote PowerShell session and connect to the MS computer, where you will configure an SSL endpoint.

Create a PowerShell Session

$session = New-PSSession MS

Enter-PSSession -session $session

While connected to MS, you can check what listeners are currently configured in the WSMan PSProvider. Run Get-ChildItem -Path WSMan:\localhost\Listener to display the current listeners, what do you see?

Answer

By default, only the HTTP Listener is configured. This is enabled when Enable-PSRemoting is run and allows for the PowerShell remoting you have used in the previous tasks.

By default, non-SSL remote PowerShell connections use port 5985. Since SSL remote PowerShell connections use port 5986, you need to open inbound traffic over that port. Run the following command in your PowerShell session to create a new firewall rule to accept SSL PowerShell connections.

New-NetFirewallRule -Name WINRM-HTTPS-In-TCP -Protocol TCP -LocalPort 5986 -Profile Any -DisplayName 'Windows Remote Management (HTTPS-In)' -Description 'Inbound rule for Windows Remote Management via WS-Management. [TCP 5986]'

Before enabling an SSL endpoint, you will need a self-signed certificate to be used for Server Authentication. Create a self-signed certificate in the LocalMachine\My store with a subject of 192.168.1.2 and type of Server Authentication. Store the output in a variable $Cert for easy access in an upcoming step.

$Cert = New-SelfSignedCertificate -CertStoreLocation "Cert:\LocalMachine\My" -Subject "192.168.1.2" -Type SSLServerAuthentication

Confirm the creation of the self-signed certificate with the proper values: Get-ChildItem -Path Cert:\LocalMachine\my -SSLServerAuthentication

Now that you have a trusted certificate, you can create a new listener in the WSMan provider. This listenter will use the HTTPS Transport, accept all IP addresses, and use the newly created certificate for authentication.

New-Item -Path WSMan:\localhost\Listener -ItemType Listener -Address \* -Transport HTTPS -CertificateThumbPrint $cert.PSChildName -Force

Use the command you learned earlier in the task to view the current listeners and confirm that the new listener has been added.

Display current listeners

Get-ChildItem -Path WSMan:\localhost\Listener\

Because the certificate is self-signed, it is not trusted on other machines, such as Win10. To make the certificate trusted, first we need to export the certificate, copy it to our Win10 machine, and then import it as a Trusted Root CA. Export the new certificate with a filename of C:\MS\_IP.cer and then copy it using your session variable to your Win10 machine. Then import it as a Trusted Root CA in the Cert:\LocalMachine\Root path.

Export the certificate

Export-Certificate -Cert $cert -FilePath: C:\MS\_IP.cer

Copy the certificate to Win10

Exit-PSSession

Copy-Item -FromSession $Session -Path C:\MS\_IP.cer -Destination C:\MS\_IP.cer

Import the certificate

Import-Certificate -FilePath C:\Win10\_IP.cer -CertStoreLocation Cert:\LocalMachine\Root

Now that a listener has been created to use SSL, run Invoke-Command with administrator credentials to the IP Address of 192.168.1.2 again. This time, include the -UseSSL parameter to use PowerShell Remoting using SSL.

UserName: Contoso\Power

Password: Pa$$w0rd

Invoke-Command using SSL

Invoke-Command -ComputerName 192.168.1.2 -Scriptblock {Get-Service -Name WinRM} -Credential Contoso\Administrator -UseSSL

Remove the certificate from your trusted root store and attempt the connection again.

Remove the trusted certificate

$cert = Get-ChildItem Cert:\LocalMachine\Root | Where-Object Subject -eq "CN=192.168.1.2"

Remove-Item Cert:\LocalMachine\Root\$($cert.ThumbPrint)

Attempt the connection again

Invoke-Command -ComputerName 192.168.1.2 -Scriptblock {Get-Service -Name WinRM} -Credential Contoso\Administrator -UseSSL

Was it successful this time?

Answer

No! Now that the certificate is no longer trusted, the connection fails.

Close PowerShell console.

Task 3.1 Store Password in the Registry as an encrypted string

In this task you will convert the domain admin password to an encrypted string and will store it into registry for future use. Values stored as encrypted string are encrypted twice - with Computer key and User key. This means that it could be un-encrypted only on the same computer with the same user credentials!

Right-click the PowerShell icon on the taskbar and select the Run PowerShell as Administrator option. This will launch the Windows PowerShell console with elevated privileges.

Passwords can be stored in variables as secure strings. Store the password for the domain administrator as a secure string in a variable called $SecureStringPass.

$SecureStringPass = ConvertTo-SecureString -String 'Pa$$w0rd' -AsPlainText -Force

After the password has been converted to a secure string, it can be converted to an encrypted string. Using Convertfrom-SecureString, convert the secure string password into an encrypted string and store it into the variable $EncryptedStringPass.

Convert to an Encrypted String

$EncryptedStringPass = Convertfrom-SecureString -SecureString $SecureStringPass

Before you can store the encrypted string in the registry you will need to create a new registry key for it. Create a new registry key for the encrypted string: New-Item -Path HKCU:\Software\Contoso

Now you can create a new string value called ScheduledPass with the content of the $EncryptedStringPass variable in the registry.

New-ItemProperty -Path HKCU:\Software\Contoso -Name ScheduledPass -Value $EncryptedStringPass

Confirm that the new value has been created successfully in the registry. Notice that the password is not saved in clear text. Get-Item -Path HKCU:\Software\Contoso

Task 3.2 Create a PSCredential object from an encrypted string and use it in a script

In this task you will create a PowerShell script that will use PowerShell Remoting with alternate credentials for the domain administrator, using the encrypted string stored in the registry.

Right-click the PowerShell icon on the taskbar and select the Run ISE as Administrator option. This will launch the Windows PowerShell ISE with elevated privileges.

The code in the next few lines will be added in the script pane. Create a variable, $User that contains the administrator username as the value.

$User = 'Contoso\Administrator'

Add the PowerShell command to retrieve the property ScheduledPass, created in the previous task, from the registry.

$RegistryString = (Get-ItemProperty -Path HKCU:\Software\Contoso -Name ScheduledPass).ScheduledPass

After the encrypted string has been retrieved, convert it back to a secure string:

$Password = ConvertTo-SecureString -String $RegistryString

At this point, the script has the proper code to build a PSCredential object. PSCredential objects will accept a plain text string for a username, $User, and a secure string for the password, $Password. Create a new PSCredential object with these two values.

$cred = New-Object -TypeName PSCredential -ArgumentList $User, $Password

Now that a PSCredential object has been created, you can run any PowerShell code against a remote computer

with these alternate credentials. Add the below command to run Invoke-Command on MS with the PSCredential object to your script.

Invoke-Command -ComputerName MS -ScriptBlock {Write-Output "Running Scheduled Task on Computer $($env:COMPUTERNAME) with Credentials: $(whoami.exe)"} -Credential $cred

Add a line to keep the window opened after execution.

Read-Host -Prompt 'Press Enter to continue...'

Your script is now complete. Save the script, which will be used in the next task: C:\PShell\Labs\ScheduledPSTask.ps1

Run script to confirm it is working correctly. You should receive the following output:

Running Scheduled Task on Computer MS with Credentials: contoso\administrator' Press Enter to continue.

Task 3.3 Create Scheduled Task using PowerShell

In this task you will create a scheduled task using PowerShell which will run the script you created in the previous task.

Right-click the PowerShell icon on the taskbar and select the Run PowerShell as Administrator option. This will launch the Windows PowerShell console with elevated privileges.

The New-ScheduledTaskAction cmdlet allows you to create a Scheduled Task Action which PowerShell will use to execute the script you created in the previous task. Create this Scheduled Task Action object and store it into a variable called $action.

New-ScheduledTaskAction

$Action = New-ScheduledTaskAction -Execute 'PowerShell.exe' -Argument '-File C:\PShell\Labs\ScheduledPSTask.ps1'

The New-ScheduledTaskTrigger cmdlet will allow to define when a scheduled task will run, or what will trigger it. Create a Scheduled Task Trigger that occurrs daily at a time a few minutes from now and store it in a variable called $trigger.

New-ScheduledTaskTrigger

$Trigger = New-ScheduledTaskTrigger -Daily -At (Get-Date).AddMinutes(5)

The last thing needed to create a scheduled task is to use the scheduled task action and scheduled task trigger to register the scheduled task. Register a Scheduled Task with the action and trigger defined in the previous steps and name it Working With Passwords.

Register-ScheduledTask

Register-ScheduledTask -Action $action -Trigger $trigger -TaskName 'Working With Passwords'

Wait a couple of minutes, whenever you set the trigger time to, to allow the Scheduled task to execute. You should see the following output in a PowerShell console triggered by the scheduled task:

Running Scheduled Task on Computer MS with Credentials: contoso\administrator'

Press Enter to close the window.

You can also manually start a scheduled task from within PowerShell. Use the Start-ScheduledTask cmdlet to start the scheduled task.

Start-ScheduledTask

Start-ScheduledTask -TaskName 'Working With Passwords'

Press Enter to close the window.

**Task 4.1: Creating scheduled jobs, triggers, and job options**

In this task, you will walk through the process of creating, disabling, and deleting scheduled jobs, job triggers, and job options.

Begin by opening Windows PowerShell ISE by right clicking on the PowerShell icon and selecting "Run ISE as an Administrator, or by clicking Run ISE as Administrator.

Create a new job trigger to execute a job at a specific time. For this exercise, you will schedule the job to execute only once, 2 minutes from the time you create the trigger

$myTrigger = New-JobTrigger -Once -At (Get-Date).AddMinutes(2)

Supply the variable holding the job trigger object as the argument to the -trigger parameter of the Register-ScheduledJob cmdlet. A unique name for the job is mandatory.

Register-ScheduledJob -Trigger $myTrigger -ScriptBlock {Get-Process} -Name myTestJob

Create a second job trigger. Assign one or more job triggers to an existing scheduled job object.

$myTrigger2 = New-JobTrigger –At (Get-Date).AddMinutes(3) –Weekly –DaysOfWeek Monday,Friday

Add-JobTrigger –Name myTestJob –Trigger $myTrigger2

Use the Get-ScheduledJob cmdlet to review the new scheduled job.

Get-ScheduledJob | Format-List -Property \*

Alter scheduled job behavior by modifying its JobOption property. The New-ScheduledJobOption cmdlet is a convenient way to list the default settings of a scheduled job’s JobOption property.

New-ScheduledJobOption

Modify the default job option by using the Get-ScheduledJobOption and Set-ScheduledJobOption cmdlets.

Note: The -Name parameter accepts the name of an existing Scheduled Job object.

Get-ScheduledJobOption -Name myTestJob | Set-ScheduledJobOption -MultipleInstancePolicy Queue

Note: The MultipleInstancePolicy parameter determines how the system responds to a request to start an instance of a scheduled job while another instance of the job is running. With Queue value the new job instance starts as soon as the current instance completes.

The Get-ScheduledJobOption cmdlet also displays the changes you have made to the job option object.

Get-ScheduledJobOption –Name myTestJob | Format-List –Property \*

Use the pipeline to display the job trigger associated with the scheduled job.

Get-ScheduledJob -Name myTestJob | Get-JobTrigger

T**ask 5.1: Creating scheduled jobs, triggers, and job options**

Begin by opening Windows PowerShell ISE by right clicking on the PowerShell icon and selecting "Run ISE as an Administrator, or by clicking Run ISE as Administrator. get--

Create a scriptblock named $script that lists all processes where the CPU property is greater than "5". Sort them by CPU in descending order. Include only the Name and CPU properties of each process object.

$script={Get-Process | Where-Object {$\_.CPU -gt 5} | Select-Object -Property Name,CPU,ID | Sort-Object -Property CPU -Descending}

Create a Scheduled job, called myTestJob5, to run daily at 2am and execute the script created in the previous step.

Register-ScheduledJob -Name myTestJob5 -Trigger (New-JobTrigger -Daily -At 1:00AM) -ScriptBlock $script

Open the Windows Task Scheduler by typing the following in PowerShell console.

taskschd.msc

In Task Scheduler Browse to \ Microsoft \ Windows \ PowerShell \ ScheduledJobs

Run the scheduledjob named myTestJob5

Open a new Windows PowerShell ISE by right clicking on the PowerShell icon and selecting "Run ISE as an Administrator, or by clicking Run ISE as Administrator.

Now in this new PowerShell ISE run the following command.

Receive-Job –Name myTestJob5

Note: Even though you are in a new session you still can receive the results!

In task Scheduler re run the scheduledjob named myTestJob5 multiple times.

Compose a pipeline command to display the job’s name, start time, end time, and command properties. Sorted the results in descending order by the start time.

Get-Job | Sort-Object PSBeginTime -Descending | Format-Table Name, PSBeginTime, PSEndTime, Command -AutoSize

**JIT JEA**

<https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/new-pssessionconfigurationfile?view=powershell-6>

dfgsd

Visual Studio Code: 🡺 <https://www.youtube.com/watch?v=LJNdK0QrIo8>

We'll explore some of the useful VSCode settings that help you automatically format your PowerShell code, change your Integrated Terminal to PowerShell, from whatever your operating system's default shell is, and launch PowerShell as soon as VSCode starts, every time! Visual Studio Code: [https://code.visualstudio.com](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqazdBRWNqQkFIYUNfcFdqZUllcldncDFJQks3QXxBQ3Jtc0tsa3VBWFZtU3FjdXFLcmwwWHFRYXJHRG9HcHRoSmtXMWIyemtZMzlSa2djSGV3NkptelU4aU5aYThaOWx3YWg5ZktnRVp4c29uMXRYLUVuSVZIM3J1TG5ab19ENHd4MW5tQ0ZCNm9OdUUydUgwVWxNMA&q=https%3A%2F%2Fcode.visualstudio.com) PowerShell Core: [https://github.com/powershell/powershell](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbVhxT1hubjdTXzNRVVA1Y0hhQkZGbG9jYzQ2Z3xBQ3Jtc0ttYkEzcjlLb2xOTEp3eU1Ra1hwZnBPblZfendMNGUzQkxjN1I0NkgxV2xFaEM5S2FlZmk2eDJ2andDMy1kdVY5ZmhUcFJfYTZYb3RLMDdoWWhFSk1nYzA2enhMazVTZ3pvRHQtVjctVjZSUDJvWUo0cw&q=https%3A%2F%2Fgithub.com%2Fpowershell%2Fpowershell) Visual Studio Code PowerShell Extension: [https://github.com/powershell/vscode-...](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbEk3Ry1SS3FTQ3pnV2x5X3ZLX0FQcktLLTlYd3xBQ3Jtc0trbkdRYkliMm1weExjbU9kamM4X0p0QzNnYUtIbkZLTWgwMnFIYUpuVVhKVFJTcmRkdGJhcTV5dDFaVUpkblBFZXdRU2QtcXFlRl9yckdKQzQ1aUxUZHZHYk1JeTJ0Mng2VDR5WXlZS2JWT2lQMkhZaw&q=https%3A%2F%2Fgithub.com%2Fpowershell%2Fvscode-powershell) Website: [https://trevorsullivan.net](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbFM3WDl6NXRoWmNlU0xfT0pxUGY0bFIxcmdTd3xBQ3Jtc0ttdTdsYWF0NkZQRHhEQjlqcjQyZzBBbVdZVkFvRHpBN2xLRmRqNkxBTmowTUtaUVNDWmVHT2ZqM29LVTZUWEZ6WDNDZXkwMWd1LWtHYlhDaU9kdEJkNUlFWkd1ODdZeFV1RHNSN0hBcTZKZkFXaHBsQQ&q=https%3A%2F%2Ftrevorsullivan.net) Twitter: [https://twitter.com/pcgeek86](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbTNMMURUcmxjTFRDVWpmRzFqMjJZTjVXYk4yQXxBQ3Jtc0tuaHVYWEtvZ01WMXYzbnlNY3pFWkNzNXZEOU0zRzRNSC1RWjY2bUNOcUtkWFhHOGpQSDBOTVVpdEZNWEhUeFZZWEVZYkJXdkpsVjI0U19NZmVZMzhiRlJQMHZ6bHpSRG0wU1lPUWxyYlJXaWJLMTE3UQ&q=https%3A%2F%2Ftwitter.com%2Fpcgeek86) GitHub: [https://github.com/pcgeek86](https://www.youtube.com/redirect?event=video_description&redir_token=QUFFLUhqbHd3OUcxaHJHNDl4U2I5OWc4N0hkT19hcUFJZ3xBQ3Jtc0tsRy1uNjRrWS1QUURFMjNZMmJCSThtTTdXX3lXZ2Q3aGx3dmtsNWhLT0ZwS2luWlVTS1lLWmF4TmlDLUxHODltVXhuRlZ5TzRNcDlqV2lvNFVweVpzTnE4bHhmUmZYbko4eFRyX1BOU1VXOVJkdFFNQQ&q=https%3A%2F%2Fgithub.com%2Fpcgeek86)

<https://docs.microsoft.com/en-us/sql/t-sql/queries/select-transact-sql?view=sql-server-ver15>

https://www.youtube.com/watch?v=UmEbsG2SEYE

regsvr32 adfs.dll

/u Unregisters server.

/s Runs Regsvr32 without displaying messages.

/n Runs Regsvr32 without calling DllRegisterServer. (Requires the /i parameter.)

/i:<cmdline> Passes an optional command-line string (cmdline) to DllInstall. If you use this parameter in conjunction with the /u parameter, it calls DllUninstall.

<DllName> The name of the .dll file that will be registered.

/? Displays help at the command prompt.