**Training Material - Windows Powershell**

**Trainer : Sheik Dawood K**

**Level : Beginner & Intermediate**

**Introduction:**

Getting to know about Windows PowerShell

Jeffrey Snover is the Founder of Powershell. Powershell is the native Scripting language for Windows Operating Systems. It is developed on the .Net Platform. As on Aug 2016, Powershell has been declared as an Open Source. The Powershell module for iMac, Unix and other non-windows platform can be downloaded, installed and the cmdlets can be used efficiently.

Command Line Interface, Capabilities of PowerShell

Type Get-Command on your Powershell. It will provide the list of all the Commands available on the Powershell. The List of commands available to specific to version of the Operating system. The List of commands differ from version to Version and Operating System to Operating System.

Basic commands and Command design structure

Every cmdlet in Windows Powershell is designed in a Verb-Noun structure. There are no spaces on any of the Windows Powershell commands. The Verb and noun are separated by the – symbol. The Auto complete option on Powershell will help to automatically select the Cmdlet from the list of available commands.

Understanding the get-help in PowerShell

Powershell has an effective Internal Help repository, which resembles the Technet Article on the Internet. To update the Help type Update-Help, this requires Internet connection and Administrator rights on Windows Powershell. There are two categories of Help.

1. Help of a Command Eg: Get-help Get-Service
2. Help of a Topic Eg: Get-help About\_Comparison\_Operators

There are 4 types of help. They are –Examples, -detailed, - Full, -Online

**Get-help Get-Service -examples**

Understanding the basic “Syntax” for PowerShell.

The Syntax in the Help of a command is the boundary or scope of the Powershell command. The Powershell command can perform all the capabilities listed in the Syntax. To understand about the [CommonParameters] type

get-help About\_Commonparameters

**Getting started with Technical commands:**

Understanding the Headers

In Powershell all the attributes are categorized under the Header. The Header is the Address which will help in

Working with Pipeline

The Pipeline is an Operators which takes Output of one command and provides as an Input for the next command. No Powershell command will neither Start or End with a Pipeline. It is mandatory that the Pipeline should be followed by a valid cmdlet.

Get-help About\_Operators

Where-object and Syntax

Where-object is a Filtering command, that filters according to the Property.

Get-help Where-object

The syntax $\_ is an Automatic Variable that supplies one row at a time to where-object to completely traverse through the results during the filtering.

Get-help about\_Automatic\_Variables

**Working with Operators:**

Arithmetic Operators

Get-help About\_Arithematic\_Operators

Comparison Operators

The Comparison Operators help in comparing the existing values in the command with the search Criteria. The list of Comparison Operators in powershell can be seen on the below help topic.

Get-help About\_comparison\_operators

Logical Operators

When you want to search and compare in more than one headers, we need to combine two or more searches using the Logical Operators.

Get-help About\_Logical\_Operators.

AND Operator – Will provide you the result only if both or all the values are TRUE.

OR Operator – Will provide you the result if either one of the value is TRUE

XOR – Will provide you the result if one value is TRUE and other is FALSE.

-NOT or ! – Checks if the Variable follows it is Empty or not.

Working with Multiple Operators

You can use any number of comparison and Logical Operators in a sequence. The Command will be executed in a Left to Right Sequence. Combining two comparisons using ( ) will prioritize the comparison.

Designing a single liner powershell command

Design a single liner powershell command with combination of a Cmdlet, Automatic Variable, Comparison Operator & Logical Operator.

**Formatting Options:**

Format-Table

Filters only the headers required for the Output. The required headers are separated by , when using Format-Table. Wildcard Operator (\*) can be used to filter all the headers with the search criteria. Eg: Time\* in Eventlogs will provide the result TimeGenerated and TimeWritten.

Using Format-Table, you can change the Header name on the Output using the Label and Expression. (Reference Example – 5)

Get-help Format-Table -examples

Select-Object

In Powershell, there are some cmdlets that does the same job, but has its own advantages and disadvantages. Select-object is also a cmdlet that filters the required headers, but it is a smarter command than Format-Table. The Output of Format-table cannot be provided as an Input for another command for processing, whereas Select-Object can do the above.

Format-list

Format-List provides the result of a command in a listed format where every Property will be in new line. FL is generally used my Administrators to view all properties of a command while exploring a new cmdlet. Format-Table will show only the default headers but Format-List will display all possible headers will correct information.

Eg: Format-Table displays Time in EventLogs, the actual header name is TimeGenerated and TimeWritten which can be found using Format-List.

Get-Member:

Get-Member cmdlet provides the complete skeletal structure of a cmdlet. It provides the result of all the Properties of the command. This command provides all the components connected with the cmdlet.

Get-Service | Get-member

Sort-object

This cmdlet helps in sorting the result. In sort-object by default it is ascending Sort and parameter available for descending sort. Two headers cannot be sorted at the same time, because every row items are tightly coupled.

**Scripting:**

Creating your First Script file/Multiple liner.

A Powershell script is a combination of one or more powershell Cmdlets bundled together and saved as a .ps1 File. Not all the cmdlets, but some cmdlets show a considerable changes while executed on the Powershell directly and Executed via a script. Eg: Get-process shows the Table format when ran on Powershell prompt but gives the result in listed format when ran via a Powershell script.

Input statement

The Input statement in PowerShell is Read-Host, which can take the input only from the console. You can specify a message after Read-Host, which will clarify the user what is the desired Input to be provided. Use –asSecureString in case a Password is required as an input.

Get-help Read-Host

Output statement

Write-host is the one of the Output Statements in Powershell. This cmdlet can write the output only to the console. This command is a Standalone command and doesn’t understand any input through Pipeline from another command. There are some color coding available with the –backgroundcolor and –foregroundcolor parameters.

The difference between a Single Quote and Double quote in Windows powershell while using with Write-Host :

Double Quotes expand the Variables value

Single Quote doesn’t expand the variable value.

Get-help Write-host

Working with Variables in Powershell

A Variable is a unit of memory that stores a value. In Powershell variables starts with a $ symbol. Eg: $num1, $Result etc. Avoid Symbols, punctuations and spaces while creating a variable.

There are three types of variable in Windows Powershell:

1. User Created Variables : These variables are created and maintained by the user. These variables exists only till the time your Powershell session is open & removed when PS Session is closed.

Eg: $Result, $num2, $Option

1. Automatic Variables : These variables are created and maintained by the Powershell. It stores the state of the PowerShell. At any point of time, user cannot change the value of an Automatic Variable.

Get-help About\_Automatic\_Variables

1. Preference Variables : These variables are created by Windows Powershell, but if the user requires to change the value, it is possible.

Get-help About\_Preference\_Variables

Importing contents from a File:

There are some cmdlets that can help in importing your files in to Powershell. Eg: Get-content and Import-csv. The choice of command changes as per the requirement of the task.

Get-content will import the file in a plain format with comma separated file as you could see on a Notepad.

Get-Help Get-content

Import-csv is a smarter cmdlet which can understand and categorize the first row as a header and contents are arranged in a table in the structure Powershell can understand. The import-csv can be stored on to a variable and the variable is ready for processing within Powershell.

Get-Help Import-Csv

Exporting the result to a file.

There are some cmdlets that can help in Exporting your results from Powershell to a desired Output File. Eg: Out-File and Export-CSV

Out-File is a simpler cmdlet which can export the contents to a plain notepad file format. If you are not much worried about the Format and the file is not required to be processed again by Powershell, this cmdlet can solve the purpose.

Export-csv is a smarter cmdlet which can Export the Output of a Powershell cmdlet to a table and header structure which can be easily understood by Powershell. If the Output file, need to be supplied as an input to another cmdlet, this will help a lot.

By default, the TYPE information is printed on the Output of Export-csv. You can use the –NoTypeInformation to remove the Type information from the First row of the exported file.

Get-Help Export-CSV

**Looping/Branching Statements:**

If, Elseif, Else statement

Key Points to remember :

1. After an If and ElseIf – Condition is Mandatory
2. After an Else Statement – not having a condition is Mandatory.
3. There should be one If and can be multiple ElseIf statements
4. Else is an Optional statement.

Get-Help About\_If

Do-while & Do-Until Statement

Key Points to remember :

1. The Statements are execute first and the condition is evaluated.
2. At any point of time, the statements within Do will be executed atleast once.
3. In a Do-While Statement, the looping will happen if the condition is evaluated to “True”.
4. In a Do-Until Statement, the Looping will happen if the condition is evaluated to “False”

Get-Help About\_Do

Switch Statements

A Switch Statement in Powershell is equivalent to a Series of If Statements, but it is simpler. Instead of writing If <Condition> and multiple ElseIf statement, you can declare a Switch statement with a test value and call just by the test value to execute the statements that satisfy the condition.

Switch statement cannot replace the If Statement in all the scenarios, but best suit in a single comparison statements.

Get-help About\_Switch

**For Statement**

Key Points to remember:

1. Three components of a FOR Statement – Initial Variable, Condition and Repeat.
2. A FOR Statement doesn’t know to handle a False condition. As soon as the condition becomes False, the FOR Loop will exit.
3. You can use any of the Assignment Operators to decide the Repeat condition.
   1. Get-help About\_Assignment\_Operators.

Get-help About\_For

**Break Statements:**

If there is a requirement to exit out of the For, While, Do or Switch Statements – Break statements can be used. Break will Exit out from the Inner loop where it is placed. Script will be continued after exiting the Inner most loop.

**Eg:**

In a Multiplication table, the condition is specified to check till 20. When a Break statement is used to exit from the For Loop after 10, even though For statement has to check till 20, Break will have a higher precedence and exit the For Loop at 10.

For ($i=1;$i -le 20;$i++)

{

If ($i -eq 6)

{Break}

$Result = $i \* 10

Write-host “$i X 10 = $result”

}

The result will be as follows:

1 X 10 = 10

2 X 10 = 20

3 X 10 = 30

4 X 10 = 40

5 X 10 = 50

**Continue Statement:**

The continue statements can be used in a situation where a condition needs to be skipped and the looping has to continue.

Eg:

In the same example if the Continue statement has been used in place of Break, the continue will take the control back to main For loop by skipping the execution of Result and Write-host. As a result, you will get the Multiplication Table without 6.

For ($i=1;$i -le 20;$i++)

{

If ($i -eq 6)

{Continue}

$Result = $i \* 10

Write-host “$i X 10 = $result”

}

ForEach statements

Key Points to remember:

1. There are two components of a ForEach Statement : $Item and $Collection.
2. $Collection is the complete CSV File stored on to a Variable using Import-CSV cmdlet.
3. $Item is one row at a time in the $Collection which will be traversed till End of Collection.
4. The header in the csv file to be placed correctly, so that PowerShell can understand the contents of the file.
5. The values of the csv file will be denoted as $item.<headername>
6. On the Output, by default the header information will not be available. Use Select-Object to get the header information.

Get-help About\_ForEach

Try Catch Finally statement

Key points to remember:

1. A Terminating Error in Powershell stops the execution of the Scripts from the point where the error is encountered.
2. When we need to customize the default error message provided by Powershell, we use Try and Catch Statement.
3. 3 blocks – Try, Catch and Finally block.
4. Try Block will have all the Statements. Upon encountering a Terminating Error, Try statement will look up for the respective Catch Statement and execute the statements inside the Catch block.
5. There can be multiple Catch blocks for different Exceptions handled by the Try block.
6. Finally Statement is an Optional statement works like a Footer message in a script.

Get-Help About\_Try

About Throw

Throw creates a Terminating Error. If you wish to generate a Terminating Error and need to pass to the Catch Statement, Throw Statement can be used. Generally, this is used by Developers for generating an error, but as an Administration, if there is a valid reason for generating Error and handling the same, you can use the Throw Keyword.

PS C:\Windows\system32> throw

ScriptHalted

At line:1 char:1

+ throw

+ ~~~~~

+ CategoryInfo : OperationStopped: (:) [], RuntimeException

+ FullyQualifiedErrorId : ScriptHalted

Get-Help About\_Throw

About Trap:

A Terminating Error Stops the script from Execution. If there is a requirement, that the script should continue even after an error has encountered, the Trap statement comes in to picture. The Trap Statement also handles the Terminating Error as a Catch Statement does but allows the script to proceed to the next lines, whereas the Catch Statement cuts off the script execution in the Try Block and proceeds with the list of statements in the Catch block.

Get-Service

Get-Processes

Trap { Write-host “Error Occurred, Script continuing using Trap Statement”}

Get-eventlog -logname Application -newest 2

Technically if there is no Trap Statement, the Terminating Error would throw a CommandNotFound Exception and exit out of the script without Executing the “Get-Eventlog”. As we are using the Trap Statement, the Error will be absorbed by the Trap Statement and the script will allow the commands to continue. Hence, the result of Get-eventlog will be executed.

The list of Errors occurred can be accessed on the Automatic Variable $Error Array Variable.

$Error[0] or $Error[1]

Get-Help about\_Trap

**Functions and Parameters**

Understanding the Powershell Functions

Key Points to remember:

1. A Function is a group of one or more cmdlets that are bundled together.
2. The Function has a name that you assign.
3. When you want to call a Function, call by its name.
4. The cmdlets execute the same way as it executes on the command prompt.
5. During the Runtime there will not be any changes noticed by the End-user.

Get-Help About\_Functions

Creating Functions in the Scripts : Lab Exercise

Understanding the Powershell Parameters

Key points to remember:

1. When there is a requirement to make your script as an Unattended Script, Parameters can be used.
2. The value for all the Input variables can be converted as a Parameters and the value to be supplied during the Runtime.
3. Remove all the Input Statements from the script and declare the Parameters at the beginning of the script.
4. [Parameter(Mandatory=$true)] – to declare a Parameter as a Mandatory Parameter.

Using Parameters and Functions in the Script – Lab Exercise

**Remote Management**

Introduction to Remote Management

1. The Remote Management in Powershell is used to connect to a remote computer and use the Functionality of the remote computer without Logging/RDP to the server.
2. Eg: Connecting to an ActiveDirectory Server and able to view the AD Users and other AD related information via Powershell without RDP.

Understanding the Powershell Modules

1. A Module is a collection of cmdlets, scripts, functions bundled together.
2. Every Windows Operating System and application has its own module.
3. Cmdlets within the module also differs from each other.
4. Modules get installed while installing the Operating system and applications.
5. Import-Module command downloads the respective module from the nearest machine to the local machine.
6. Importing the module at the beginning of the script makes the script independent and works anywhere from your domain.
7. The imported module will be available till the session is active and will be disconnected when the PS window is closed.
8. As a best practice, use Remove-Module at the end of your script to remove the downloaded module to avoid the Vulnerability of command on the downloaded session.

Understanding the Powershell PSSession

1. Different applications allows different methods of remote Management.
2. Import-PSSession connects to the Powershell Session of the remote computer.
3. Import-PsSession has the Parameters like –ConfigurationName, -ConnectionUri to connect to the remote computer PS Session.
4. Import-PSSession has the privilege to restrict the list of commands that can be imported to the local PS Session. We can use –CommandName with WildCard Search to get specific commands. You can use comma (,) separated list of commands to be downloaded.
5. Use Remove-PsSession to remove the connection at the end of your script.

Importing a PSSession from Remote server

1. Importing a PsSession is a 3 Step process.
   1. Create a PsSession using New-PsSession command.
   2. Store the newly created on to a Variable.
   3. Import the newly created PsSession.

$Session=New-PSSession -ConfigurationName Microsoft.Exchange -ConnectionUri http://<ServerFQDN>/PowerShell/ -Authentication Kerberos -Credential $UserCredential

Import-PSSession $Session -DisableNameChecking

**WMI (Windows Management Instrumentation)**

Introduction to WMI

WMI (Windows Management Instrumentation)

1. Windows Management Instrumentation is a collection of Classes. The Classes for all the Hardware, Software and Network Components of the Computer are categorized in WMI.
2. To search for a Class Get-WMiObject –List \*Keyboard\*
3. To expand the class type Get-WMIObject –Class Win32\_Keyboard
4. To get all the properties Get-WMiObject –class Win32\_Keyboard | format-List
5. To get the skeletal structure of the class Get-WMIObject –class Win32\_Keyboard | get-Member

Understanding WMI and CIM

When a WMI Class is searched we notice that there are two classes with the same name but one starts with CIM and other WIN32. CIM – Common Information Model is the Open Standard from DMTF (Distributed Management Task Force). The Win32 is the Microsoft’s Implementation of the Open standard according to the Windows Environment.

WMI vs CIM

A new cmdlet Get-CIMInstance helps to get the information from the WMI in a formatted way. The Get-WMIobject extracts the LastBootupTime data in an unformatted way and it requires Manual intervention to order in a readable format. Using Get-CIMInstance gives the much clear way of fetching the data from the repository.

Scripting with WMI

If you know the class where your required Data is stored, rest all the steps are same as writing the Scripts and formatting in the required structure.

Most commonly used scripts which uses WMI are

1. To get the LastReboot of all the Machines.
2. Disk Space Management.

**DSC (Desired State Configuration):**

<https://www.red-gate.com/simple-talk/sysadmin/powershell/powershell-desired-state-configuration-the-basics/>