Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI

Section 1 : Strengthening the PowerShell Basics

- PowerShell & Its Importance
- Getting PowerShell
- ISE & Console
- Execution Policy
- PowerShell Security
- PowerShell Help
- Important commands

PowerShell History

Introduction to PowerShell & Its Importance

Microsoft defines PowerShell as:

Built on the .NET Framework, Windows PowerShell is a task-based command-line shell and scripting language; it is designed specifically for system administrators and power-users, to rapidly automate the administration of multiple operating systems (Linux, macOS, Unix, and Windows) and the processes related to the applications that run on those operating systems.

PowerShell is an object-oriented automation engine and scripting language with an interactive command-line shell that Microsoft developed to help IT professionals configure systems and automate administrative tasks.

PowerShell is now open source

Stable Release: 5.1

Article:

https://docs.microsoft.com/en-us/powershell/scripting/powershell-scripting?view=powershell-5.1

Introduction to PowerShell & Its Importance

PowerShell can be considered as "glue" that ties most of Microsoft applications together.

Virtually all of the server products Microsoft is producing right now can be managed through PowerShell. From an administrative standpoint, this means that if you become proficient in PowerShell, you will have the skill set necessary for managing most of Microsoft's newer product

PowerShell is object-based.

This gives us incredible flexibility. Filter, sort, measure, group, compare or take other actions on objects as they pass through the pipeline. Work with properties and methods rather than raw text.

Why PowerShell

Consistency.

A scripted solution will run the exact same script every time

No risk of typos, forgetting to complete the task, or doing the task incorrectly

Audit trail.

There are many tasks where having an audit trail would be helpful, perhaps including what task was performed, important results, errors that occurred, when the task ran, who ran it, and so forth.

Change of pace.

- From Scripting you can achieve a task significantly faster than GUI

PowerShell is easy to adopt, learn, and use because it does not require a background in programming.

Purpose of PowerShell

- Improved Management
- Improved Automation
- Manage real-time
- Manage large scale environments

Getting PowerShell

Starting with Windows 7, PowerShell is part of operating system installation.

However, you can still download and install the same from Microsoft website for free. It is available as Windows Management Framework.

Download & Install:

Link for Windows Management Framework 5.0

https://www.microsoft.com/en-us/download/details.aspx?id=50395

PowerShell Console & ISE

ISE:

The Windows PowerShell Integrated Scripting Environment (ISE) is a host application for Windows PowerShell. In Windows PowerShell ISE, you can run commands and write, test, and debug scripts in a single Windows-based graphic user interface with multiline editing, tab completion, syntax coloring, selective execution, context-sensitive help, and support for right-to-left languages.

Console:

It is recommended for quick one liners commands and executing the scripts

Execution Policy

Windows PowerShell execution policies let you determine the conditions under which Windows PowerShell loads configuration files and runs scripts.

You can set an execution policy for the local computer, for the current user, or for a particular session. You can also use a Group Policy setting to set execution policy for computers and users.

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_execution_policies?view=powershell-5.1

Different Execution Policies:

- **AllSigned:**

  Requires that all scripts and configuration files be signed by a trusted publisher, including scripts that you write on the local computer. Prompts you before running scripts from publishers that you have not yet classified as trusted or untrusted.

- **RemoteSigned:**

  Does not require digital signatures on scripts that you have written on the local computer (not downloaded from the Internet)

- **Unrestricted:**

  Unsigned scripts can run. Warns the user before running scripts and configuration files that are downloaded from the Internet.

- **Restricted**

  Permits individual commands, but will not run scripts.

- **Bypass:**

  Nothing is blocked and there are no warnings or prompts.

Execution Policy…

`Get-ExecutionPolicy`

To get your current execution policy:

`Set-ExecutionPolicy`

The Set-ExecutionPolicy cmdlet changes the user preference for the Windows PowerShell execution policy

Example: `Set-ExecutionPolicy RemoteSigned`

PowerShell Security

- **Association with notepad** & not powershell.exe by default
- Execution Policy **restrictions**
- Have to type **explicit path of the script** in order to execute it
- Can required **script to be signed** else will not execute it.
- Can further add restriction that script signed by **your trustworthy certificate provider only**.

Script Signing

Help New-SelfSignedCertificate -ShowWindow

https://docs.microsoft.com/en-us/powershell/module/pkiclient/new-selfsignedcertificate?view=win10-ps

Get-Help

The **Get-Help** cmdlet displays information about Windows PowerShell concepts and commands, including cmdlets, functions, CIM commands, workflows, providers, aliases and scripts.

Example:

Get-Command -Name "*service*"

Get-Help Get-Service	# Get Help available for any PowerShell command
Get-Help Get-Service **–Full**	# To access full help with examples
Get-Help Get-Service **–online**	# Search for online help
Get-Help Get-Service **-ShowWindow**	#Special window for navigation through Help

More Help related commands

Update-Help

- Downloads and installs the newest help files on your computer.

Save-Help

- Downloads and saves the newest help files to a file system directory.

To get help about PowerShell concepts

```
Get-help *about*    # To get the list of all help topics
Get-Help about_WQL -ShowWindow
```

Get-Command

To get basic information about PowerShell commands: cmdlets, files and functions.

The **Get-Command** cmdlet gets all commands that are installed on the computer, including cmdlets, aliases, functions, workflows, filters, scripts, and applications

```
PS C:\PowerShell\Advanced_PowerShell> Get-Command

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Function        A:
Function        B:
Function        C:
Function        cd..
Function        cd\
Function        Clear-Host
Function        Compress-Archive                                   1.0.1.0    Microsoft.PowerShell.Archive
Function        Configuration                                      1.1        PSDesiredStateConfiguration
Function        ConvertFrom-SddlString                             3.1.0.0    Microsoft.PowerShell.Utility
Function        D:
Function        Disable-DscDebug                                   1.1        PSDesiredStateConfiguration
Function        Disable-NetworkSwitchEthernetPort                  1.0.0.0    NetworkSwitchManager
Function        Disable-NetworkSwitchFeature                       1.0.0.0    NetworkSwitchManager
Function        Disable-NetworkSwitchVlan                          1.0.0.0    NetworkSwitchManager
```

Few Important Commands to Start

	Shell/CMD(PowerShell Alias)		PowerShell Equivalent
✓	pwd	=	Get-location
✓	CD	=	Set-Location
✓	%date% %time%	=	Get-Date
✓	ls or DIR	=	Get-ChildItem
✓	echo	=	Write-Output
✓	cls	=	Clear-Host

PowerShell Version

$PSVersionTable

```
PS C:\PowerShell\Advanced_PowerShell> $PSVersionTable

Name                           Value
----                           -----
PSVersion                      5.1.14409.1005
PSEdition                      Desktop
PSCompatibleVersions           {1.0, 2.0, 3.0, 4.0...}
BuildVersion                   10.0.14409.1005
CLRVersion                     4.0.30319.42000
WSManStackVersion              3.0
PSRemotingProtocolVersion      2.3
SerializationVersion           1.1.0.1
```

$Get-Host

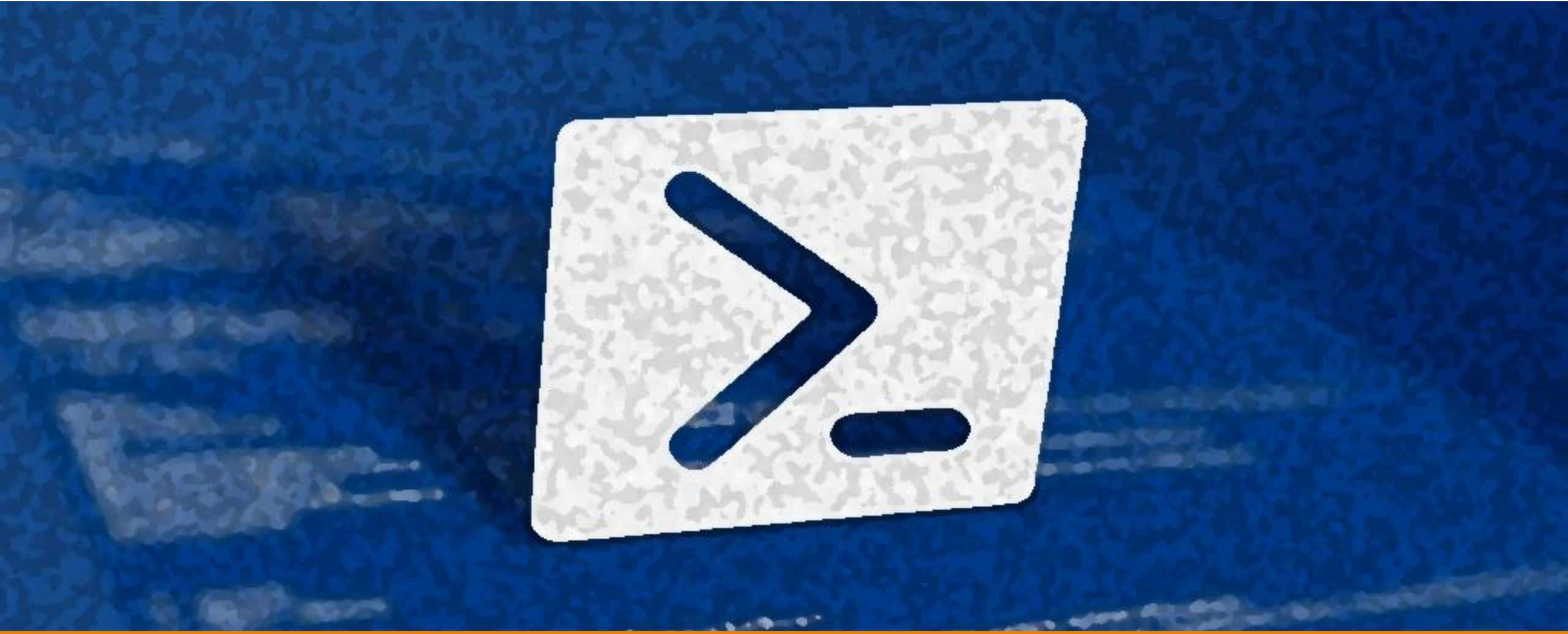
```
PS C:\PowerShell\Advanced_PowerShell> Get-Host


Name             : Windows PowerShell ISE Host
Version          : 5.1.14409.1005
InstanceId       : 2fc39b4c-f4df-4e8a-966b-69c9b46b23b2
UI               : System.Management.Automation.Internal.Host.InternalHostUserInterface
CurrentCulture   : en-US
CurrentUICulture : en-US
PrivateData      : Microsoft.PowerShell.Host.ISE.ISEOptions
DebuggerEnabled  : True
IsRunspacePushed : False
Runspace         : System.Management.Automation.Runspaces.LocalRunspace
```

Few Important Commands to Start

	Shell/CMD(PowerShell Alias)		PowerShell Equivalent
✓	cp or copy	=	Copy-Item
✓	mv or Move	=	Move-Item
✓	ren	=	Rename-Item
✓	del or rm	=	Remove-Item
✓	man or help	=	Get-Help
✓		=	Get-History
✓		=	Clear-History

Please practice all the commands and appear in the first Quiz

Section Covered



Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI

Section 2

Programming Building Blocks

Variables	Data Type &Typecasting	Read, Validate & Write
Comparison Operators	If Else Loop & Switch	Collections
Iteration	Method	Error Handling

Variables

Variables are the names you give to computer memory locations which are used to store values in a computer program

PowerShell uses variables as temporary, named storage for objects. Variable name begins with $

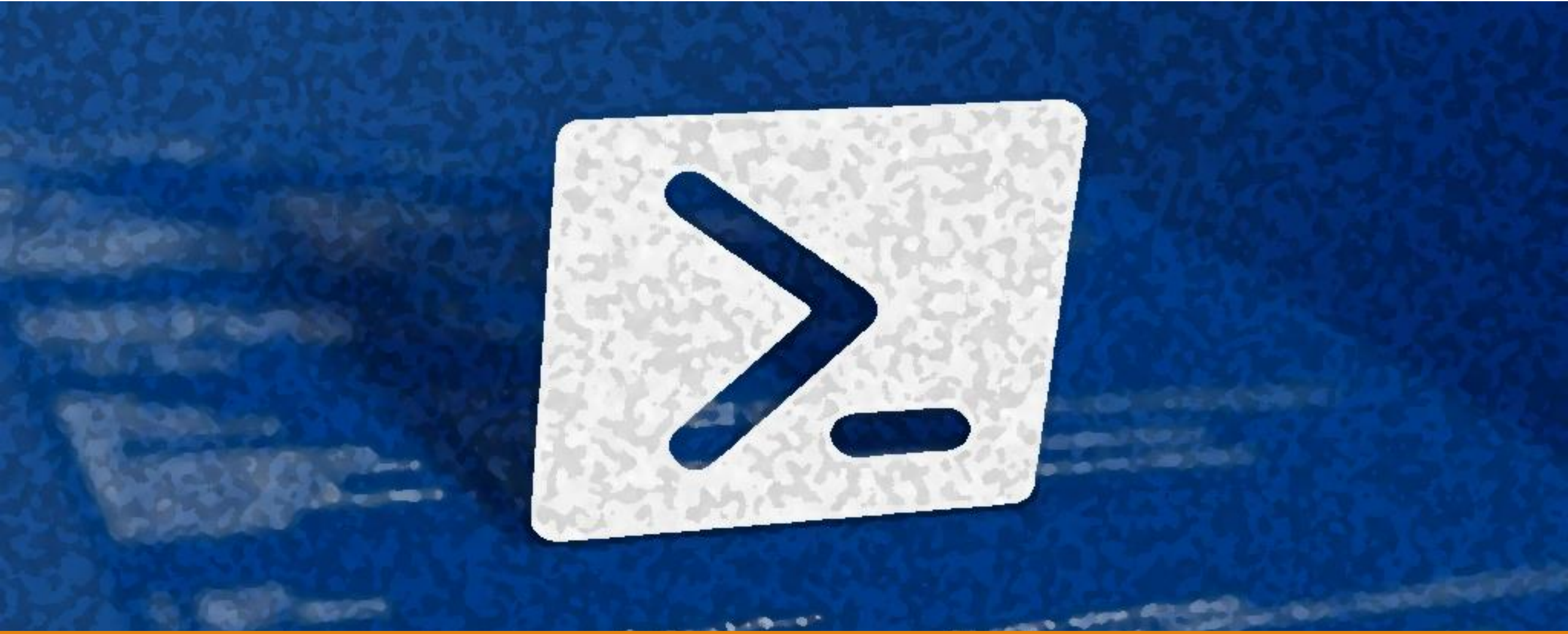
Variables are objects. It is the name of memory location, where objects are stored.

Variables

Creating PowerShell Variables:

```
$MyVariable = "Some String Value“
$global:var_name = “This will be accessible throughout the script”
Set-Variable -Name "myVar" -Value "value of the variable"
New-Variable -Name "myVar" -Value "value of the variable"
```

To access any variable, type $ and then the variable name



Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI

Variables

Other Important Variable related commands,

```
Get-Variable
Clear-Variable
Remove-Variable
```

Variables

By default, variable name cannot contain spaces and special letter(except few), However in PowerShell, If you really need you can give any name to the variable

```
${ give any _name&&**++) } = "Value of this variable"

${ give any _name&&**++) }
```

```
2   ${ give any _name&&**++) } = "Value of this variable"
3   ${ give any _name&&**++) }
```

```
PS C:\PowerShell\Advanced_PowerShell> ${ give any _name&&**++) } = "Value of this variable"

PS C:\PowerShell\Advanced_PowerShell> ${ give any _name&&**++) }
Value of this variable

PS C:\PowerShell\Advanced_PowerShell>
```

Constant

A **constant** is a value that never changes.

Example: value of Pi, speed of light, radius of earth

Syntax

```
Set-Variable test -option Constant -value 100
Set-Variable test -option ReadOnly -value 100
```

Data Type

Type of Data we are storing inside a variable

You don’t have to explicitly declare the data type of a variable; PowerShell automatically chooses the data type for you when you initialize the variable—that is, when you first assign a value.

Some common data types: String, Integer, Boolean, Double etc

Integer

Using Math Operators

```
$var1 = 100
$var2 = 205
$sum = $var1 + $var2  #Sum
$diff =  $var2 -  $var1  #Difference
$Remainder  = $var2%$var1  #Modulus Operator


#Type of the variable
$var1.GetType()
```

Integer

Practice Exercise

$students marks in different subjects(Out of 100) = 74, 85, 77

Calculate percentage of marks

Solution:

(74 + 85 + 77) / (100 + 100 + 100) * 100

Tip:

(Apply your **Bodmas** knowledge here ☺)

String

```
$var_name = "Vijay"
$var_year = "Year: 2018"
$current_temp = "40"
$another_string_var = "$var_name is making a course in $var_year and current
                        temperature is $current_temp"
```

All 3 above variables are String type. Any confusion???

```
PS C:\PowerShell\Advanced_PowerShell> $current_temp = "40"

PS C:\PowerShell\Advanced_PowerShell> $current_temp.GetType()

IsPublic IsSerial Name                                     BaseType
-------- -------- ----                                     --------
True     True     String                                   System.Object
```

Another way of creating String, Place value inside single quotes.

```
$another_string_var = 'This is also String’
```

So, double or single quotes means the same thing ???

NO

Difference is that single quotes DO NOT resolve the variables to its value. Whereas Double quotes resolve the variable to its value and you will find only the value to be printed.

How to avoid resolving of a variable inside double quotes?

use escape character(grave-accent(`)) before every variable which you do not want to be resolved

```
6   $another_string_var = "$var_name  `$var_year  $current_temp"
7   $another_string_var
```

```
PS C:\PowerShell\Advanced_PowerShell> $another_string_var
Vijay  $var_year  40
```

$string_var = "a Random String "

To get the list of different methods available on String

```
$string_var | Get-Member
```

Get Length of String:

```
$string_var.Length
```

Check if it contains a substring or not:

```
$string_var.Contains("Random")
```

Remove the white space from beginning and end:

```
$string_var.Trim()
```

Search & replace:

```
$string_var.Replace("a" , "b")
```

Convert string into upper case:

```
$string_var.ToUpper()
```

```
PS C:\PowerShell\Advanced_PowerShell> $string_var = "a Random String  "
$string_var.Length
17

PS C:\PowerShell\Advanced_PowerShell> $string_var.Contains("Random")
True

PS C:\PowerShell\Advanced_PowerShell>
$string_var.Contains("random")
False

PS C:\PowerShell\Advanced_PowerShell>
$string_var.IndexOf("Random")
2

PS C:\PowerShell\Advanced_PowerShell>
$string_var.Trim()
a Random String

PS C:\PowerShell\Advanced_PowerShell>
$string_var.ToUpper()
A RANDOM STRING

PS C:\PowerShell\Advanced_PowerShell>
$string_var.Replace("a" , "b")
b Rbndom String

PS C:\PowerShell\Advanced_PowerShell>
```

Q.) How to handle quotes inside string?

Ans.) We can use **here-string**.

Just we need to enclose our string value inside @” ”@

```
Clear-Host
$text = @"
Question: "Who are you?"
Answer: "I am so & so and came here for this purpose"
"@

$text
```

```
Question: "Who are you?"
Answer: "I am so & so and came here for this purpose"
PS C:\PowerShell\Advanced_PowerShell>
```

Q.) How to validate the datatype?

Ans.) Using **–is** operator

```
$a_var = 1000
$a_var -is [int]        => TRUE
$a_var -is [String]     =>  False
```

Typecasting

PowerShell, automatically assigns a data type to variable. However, If you explicitly want to define a data type, you are allowed to do so.

Method1: Apply data type on variable

```
[int]$x=100
```

Method2: Apply data type on value

```
$y=[int]100
```

Predict the type of variable `[int]$variable = "121"`

Typecasting Advantage

1.) We don’t have to waste time in writing logics, If we use types smartly

```
$date_string = "01/26/2018"
[DateTime]$date_string = $date_string
$date_string


$date_string.GetType()
```

2.) Type casting can be used for validation of data

Reading User Input

The Read-Host cmdlet reads a line of input from the console. You can use it to prompt a user for input. Because you can save the input as a secure string, you can use this cmdlet to prompt users for secure data, such as passwords, as well as shared data.

Example:

```
[string]$name = Read-host "What is your name"
[int]$age = Read-host "What is your age"
```

Variable Data Validation

It is best practice to validate the data in our script. Especially, when your script is taking the user input or from other independent source.

Few Validation commands:

```
[validateset("y","Y","n","N")]
[ValidateCount(1,5)]
[ValidateLength(1,10)]
[ValidateRange(0,10)]
```

```
Get-Help about_Functions_Advanced_Parameters -ShowWindow
```

Write-Host

The Write-Host cmdlet customizes output. You can specify the color of text by using the Foreground Color parameter, and you can specify the background color by using the Background Color parameter.

Example:

```
Write-Host "Hello There! I am feeling awesome"
Write-Host “Colorful text" -ForegroundColor Cyan
Write-Host “More colors" -ForegroundColor green -BackgroundColor re
```

More ways to write into console

```
Write-Debug "This is Debug message" -Debug

Write-Verbose "This is VERBOSE" -Verbose



Write-Error "This is a Error message"

Write-Warning "This is  Warning message"

Write-Output "This is  Output message"
```

Write-Host vs others

1.) To understand the difference, execute below commands

```
Write-Output "Hello" | Get-Member
Write-Host "Hello" | Get-Member
```

You will understand that Write-host do not send object to the pipeline for other cmdlet whereas Write-Output does.

2.) **Write-Host's output cannot be stored into a variable.** Try below command to understand

```
$var = Write-host "Hello"
$var
$var = Write-Output "Hello"
$var
```

Conclusion: Write-Output is better choice in most of the cases**.** We can use Write-Output if we want to use our script only in console and don't want to create logs/redirection of output etc.

Comparison Operators

Comparison operators let you specify conditions for comparing values and finding values that match specified patterns. To use a comparison operator, specify the values that you want to compare together with an operator that separates these values.

Example:

-eq and –neq

-le and -gt

-like and –notlike

-contains and -notcontains

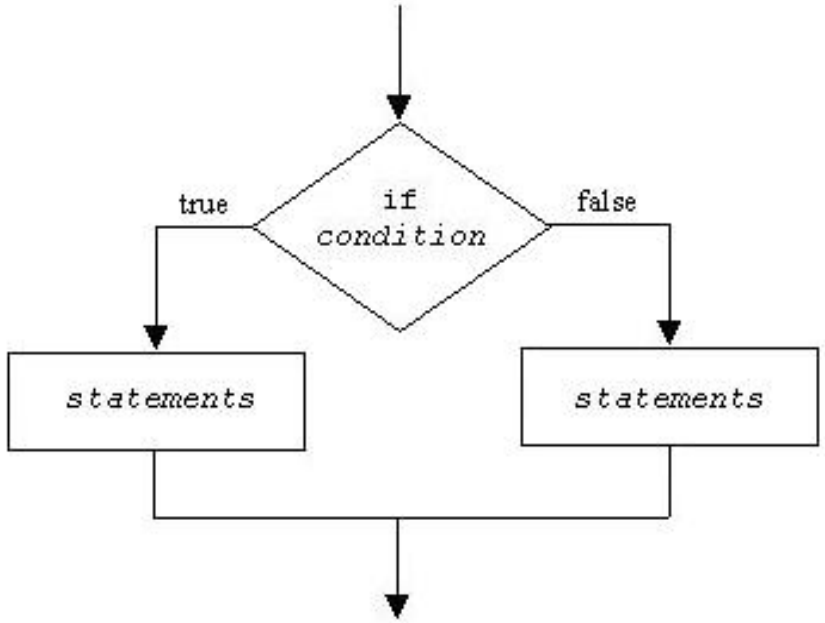
-match and -notmatch

If-Else

Comparison operators returns true or false depending upon the inputs. Depending upon this result, we can proceed to make a decision within the script at runtime and do a set of task, if we get true and another set of task, if output is false.

PowerShell Syntax:

```
 If (condition) {
    #Do stuff if condition is true
} else{
    #Do stuff if condition is true
}
```

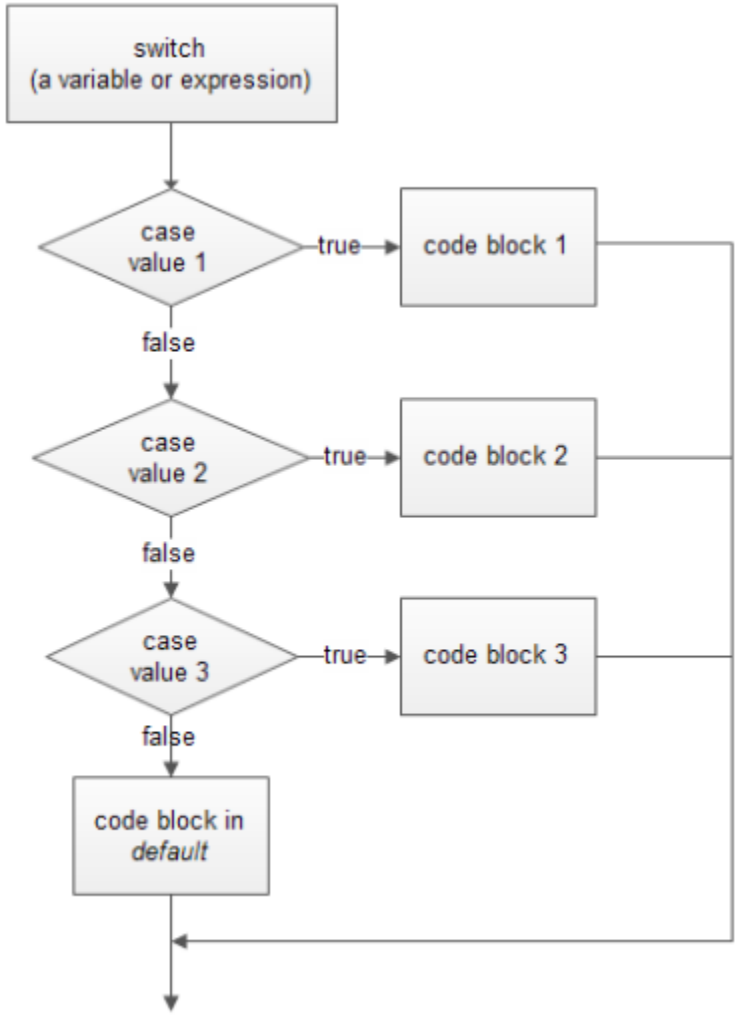


Switch

To check multiple conditions, use a Switch statement. The Switch statement is equivalent to a series of If statements, but it is simpler. The Switch statement lists each condition and an optional action. If a condition obtains, the action is performed.

PowerShell Syntax:

```
Switch (<test-value>)
{
    <condition> {<action>}
    <condition> {<action>}
}
```



We can use If else output and store it inside a PowerShell variable

```
#Example1
$value = if($true){1}else{2}
$value

#Example2
$day = "Sunday"
$activity = if($day -like "Sunday"){ "FUN" }else{ "Work"}
$activity
```

PowerShell Collection

Collection is nothing but an object that groups multiple elements into a single unit.

Collections can be used to store, retrieve and manipulate data.

It is also called as container.

Example: Array, Arraylist, Hashtable etc.

Array

To get the size of an array : `$arrayname.Length`

To access an item from array by index : `$arrayname[index_number]`

To access a range of items : `$arrayname[1..5]`

Array Index:

0 is the index of first item in array

1 is second item

.

(size - 1) is the index of last element in array

Array Structure:

Index	0	1	2 …	N-1
Value	Value1	Value2	Value3	Value N

ArrayList

To frequently add elements to, remove elements from, search, and modify an Array

Class: System.Collections.ArrayList

```
$student_list = New-Object System.Collections.ArrayList
$student_list.Add("Male_Student1")
$student_list.AddRange( ("Male_Student2","Female_Student3") )
$student_list
```

Array & ArrayList

Advantages:

- Simple & Easy to use
- Can be used to store objects of multiple data types

Disadvantages:

- We must know in advance that how many elements are to be stored in array as it is of fixed size
- Searching and Sorting is slow & inefficient

Array & ArrayList

Like most other languages, arrays in PowerShell stay the same length once you create them.

PowerShell allows you to add items, remove items, and search for items in an array, but these operations may be time consuming when you are dealing with large amounts of data.

For example, to combine two arrays, PowerShell creates a new array large enough to hold the contents of both arrays and then copies both arrays into the destination array.

In comparison, the ArrayList class is designed to let you easily add, remove, and search for items in a collection.

HashTable

A hash table, also known as a dictionary or associative array, is a compact data structure that stores one or more key/value pairs.

Example, a hash table might contain a series of IP addresses and computer names, where the IP addresses are the keys and the computer names are the values, or vice versa.

Syntax:

```
$hash = @{}
$student_data = @{
        "name" = "Student1 name";
         "Course" = "Advanced PowerShell"
        "Sex"    = "Male
      }
```

Iterations

1.) While loop:

The logic of this loop is to execute the **while(**the condition is true**) (- - - Do - - -)**.

Pay close attention to the style of brackets, the curly brackets or parentheses are guiding you to write the correct code.

```
$i =1
while ($i -le 10) {
    Write-Output "value if variable i: $i";
    $i +=1  # this is similar to   $i = $i + 1
}
```

Iterations

2.) Do While loop:

```
$i = 10
do{
    Write-Output "Count: $i"
    $i--
}while($i -ge 0)
```

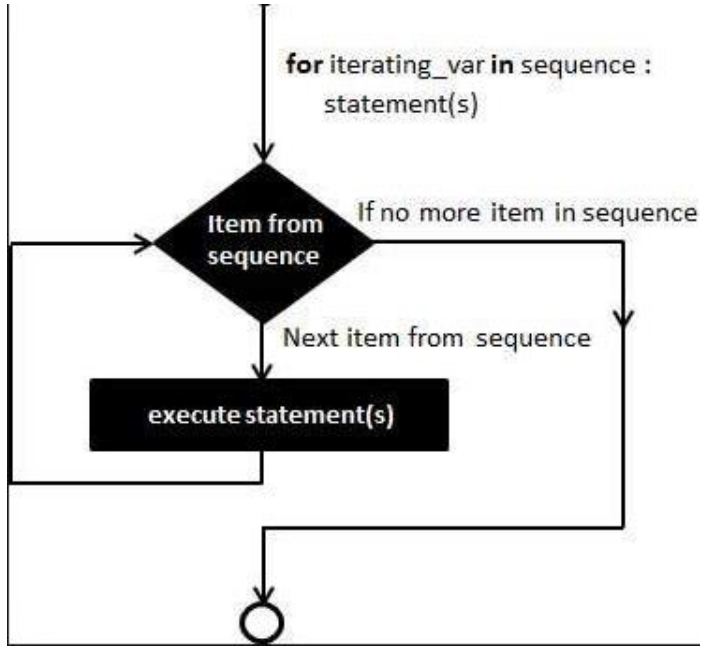
For Loop

The loop to perform a Task, for set number

of iterations until the condition is true.

Syntax:

```
for (init; condition; repeat){
    #statement1
    #statement2
    #
 }
```



PowerShell Functions

A function is a list of Windows PowerShell statements that has a name that you assign. When you run a function, you type the function name. The statements in the list run as if you had typed them at the command prompt.

Syntax:

```
function Verb-Noun {
 #Statement 1
 #Statement 2
}
```

To Call Function: `Verb-Noun`

Error Handling

Error handling is important when creating PowerShell scripts. A script that runs correctly once may not run correctly every time.

There always seems to be some kind of problem that crops up when you least expect it. This is why error handling should be implemented in every important piece of PowerShell code you create.

Why Error Handling ?

Task	Uncertainties (Possibilities of Exception)
Reading/Writing File content	File got deleted, Network dependency, File locked, Insufficient Access
Database Operation	Database table not existing, DB maintenance in progress, Database down, Network dependency, Insufficient Access, Data Duplication not allowed
Sending Daily Report	SMTP Server Issue, Report File locked
Version	Script was developed for a particular version which wasn't available in actual production environment

Types of Error

Termination Error:

A terminating error is an error that will halt/stop the function or an operation.

Example: Syntax Error, Out of memory Exception

Non-Terminating Error:

A non-terminating error is an error that will allow PowerShell to continue with execution of Script or next set of statements

Example: Errors thrown by cmdlets

Error Handling

-ErrorAction Parameter(Alias: -EA):

It is parameter to any Cmdlet to define the action to be taken upon error

All cmdlets accepts error action parameter.

```
Write-Error -Message "4" -ErrorAction
```

- Continue
- Ignore
- Inquire
- SilentlyContinue
- Stop
- Suspend

Most cmdlet, terminates the current statement, but continue with other statements, If we want to stop the script execution right there, we can use **–EA Stop** parameter to stop the execution.

Error Handling

$ErrorActionPreference

This is PowerShell variable which sets the preference for action to be taken on errors for all different cmdlets, instead of defining action on individual cmdlets

- ✓ **Continue**: Display errors, But continue to next statement if possible
- ✓ **SilentlyContinue**: Suppress the errors
- ✓ **Stop**: Display the error. Terminate the script execution there.
- ✓ **Inquire**: Ask the user

This variable can be used for setting the preference for the whole script. However, If we want to set the preference for a individual command, we can do that using ErrorAction Parameter.

Error Handling

-ErrorVariable(Alias: -EV)

Error Variable will store the error message and you can use it later for showing it to the user or logging it with inside your logfile

It will store the error message even if error preference is set to be silently continue

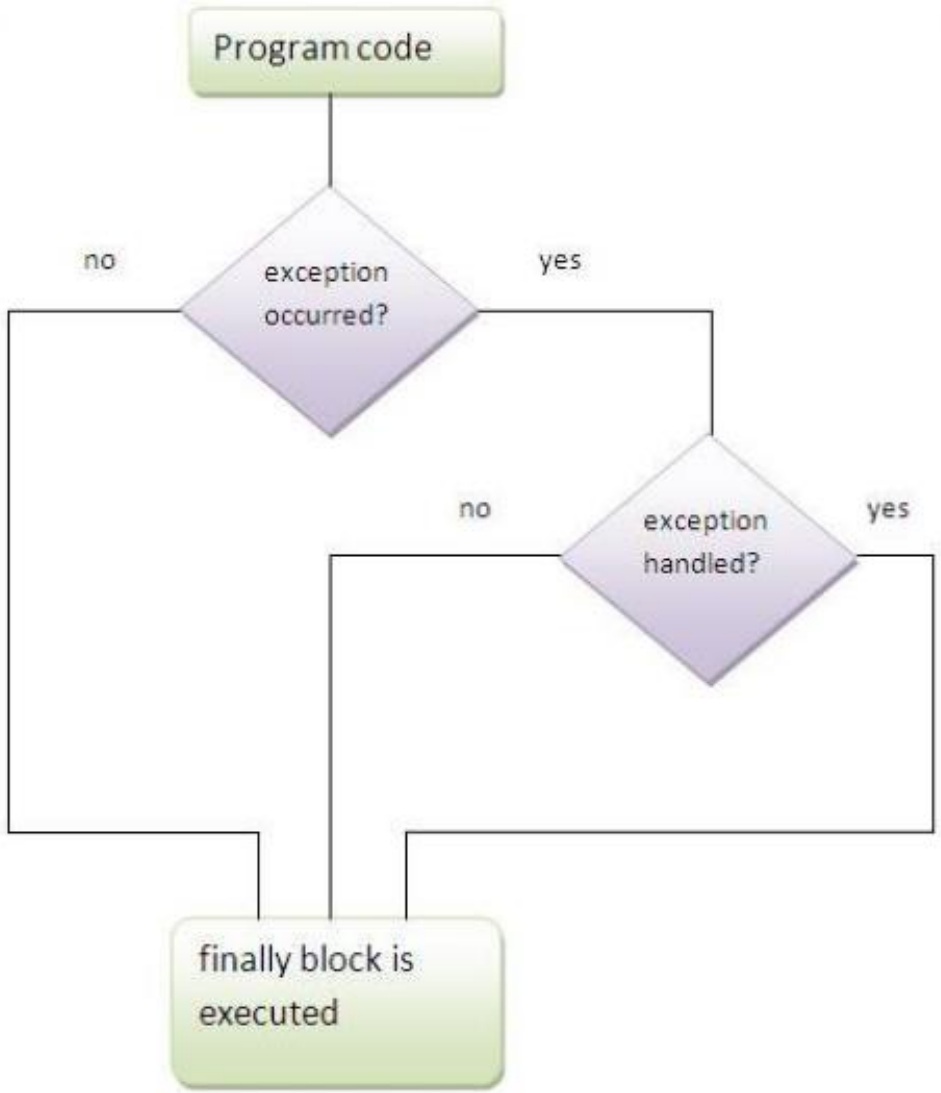
```
PS C:\PowerShell> Get-Content -path "Filename.txt"  -ErrorAction SilentlyContinue -ErrorVariable "error_var"

PS C:\PowerShell> $error_var
Get-Content : Cannot find path 'C:\PowerShell\Filename.txt' because it does not exist.
At line:1 char:1
+ Get-Content -path "Filename.txt"  -ErrorAction SilentlyContinue -Erro ...
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (C:\PowerShell\Filename.txt:String) [Get-Content], ItemNotFoundException
    + FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.GetContentCommand
```

Try Catch

When an exception is thrown anywhere inside of a try block, there's a catch block that's there to catch the thrown exception and do something with it.

In PowerShell, an exception is a terminating error.

A terminating error stops a statement from running.



Try Catch Finally

A Try statement contains a Try block, zero or more Catch blocks, and zero or one Finally block. A Try statement must have at least one Catch block or one Finally block.

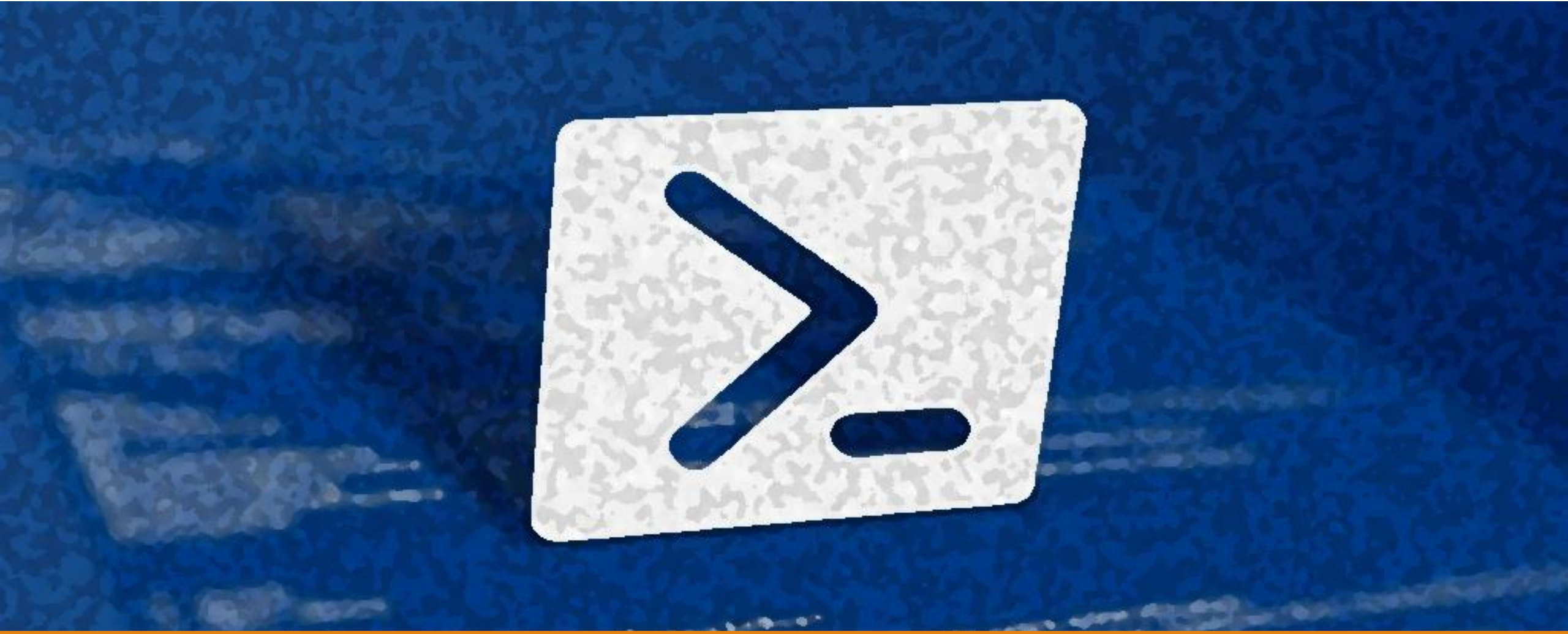
```
try {
        #statement_list  - Your PowerShell commands which does some stuff for you
} catch {
        #statement_list  - This statement will execute only if and exception occurred in Try Block
} finally {
        #statement_list   - Finally this statement will execute irrespective of exception occur or no
}
```

Accessing The Error Records

Inside a catch block we can access the error record, which is stored in the current object variable $_

```
Catch{
   $ErrorMessage = $_.Exception.Message
   $FailedItem = $_.Exception.ItemName
   Write-Output "ErrorMessage : $ErrorMessage " | Out-File -FilePath "Logfile.log" -Append
   Write-Output "FailedItem : $FailedItem " | Out-File -FilePath "Logfile.log" -Append
 }
```

Section Completed ☺



Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI

Text File Handling

CSV File Handling

XML File Handling

Advanced File Handling

Test-Path

```
To check the existence of a file


if (Test-Path C:\PowerShell\test_file\A_Random_File.txt){
    Write-Output "File Exist"
} else {
    Write-Output "File Do NOT Exist"
}
```

Reading a file

To read a text file into PowerShell, we can use Get-Content cmdlet.

You can use options of this command to read the desired content.

```
Get-Content C:\SomeDirectory\A_Random_File.txt
```

Writing into a file

To write into a text file, we can use Out-File cmdlet.

You can use options like –Append, -Force, -Encoding as per your requirement

This cmdlet will create the file if don’t already exists.

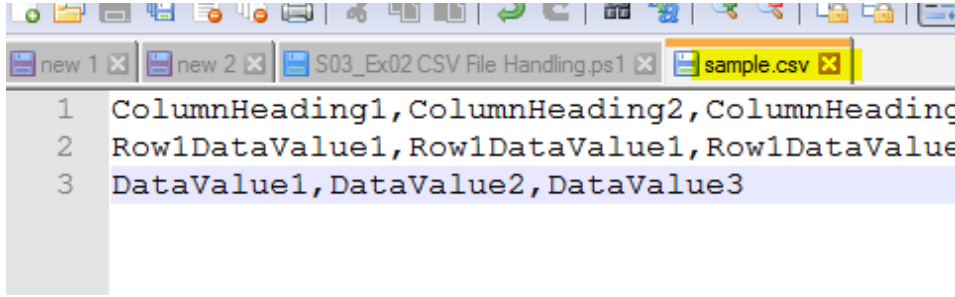
Syntax:

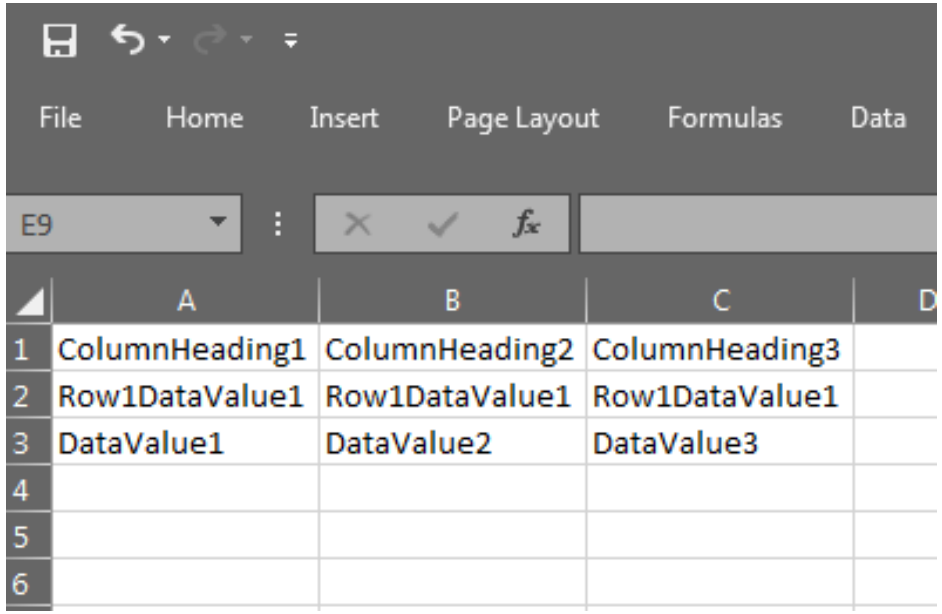
```
Write-Output “Some Text" | Out-File -FilePath output.txt
```

Comma Separated Values(.CSV)

CSV files are plain text file in which values tabular data is saved as comma separated values.

File Extension: .CSV





CSV File Handling

To import CSV file content into a PowerShell variable

```
$csv_content = Import-Csv “File_Location"
```

The Import-Csv cmdlet creates table-like custom objects from the items in CSV files. Each column in the CSV file becomes a property of the custom object and the items in rows become the property values. Import-Csv works on any CSV file, including files that are generated by the Export-Csv cmdlet.

CSV File Handling

```
Add-Content -Path Students_data.csv  -Value '"Name","Class","Percentage"'

$student_data = @(
  ‘”Student 0","Science","95%"'
  '"Student 1","Maths","98%"'
  '"Student2","Geography","60%"'
  )

$student_data | foreach { Add-Content -Path  Students_data.csv -Value $_ }
```

XML Files

Extensible Markup Language (XML) is used to describe data. The XML standard is a flexible way to create information formats and electronically share structured data via the public Internet, as well as via corporate networks.

XML code, a formal recommendation from the World Wide Web Consortium (W3C), is similar to Hypertext Markup Language (HTML). Both XML and HTML contain markup symbols to describe page or file contents.

PowerShell offers a number of different ways to read XML documents, without having to write a lot of code

comp_info_sample.xml | new 1

```xml
<MAIN_NODE>

    <COMP>
        <NAME>machine_name</NAME>
        <IP>123.123.123.123</IP>
        <DOMAIN>domina_name</DOMAIN>
        <APPLICATION_INSTALLED> my_app_name </APPLICATION_INSTALLED>
        <OWNED_BY_TEAM> ownersip_team </OWNED_BY_TEAM>
    </COMP>

    <COMP>
        <NAME>machine_name2</NAME>
        <IP>124.124.124.124</IP>
        <DOMAIN>domina_name2</DOMAIN>
        <APPLICATION_INSTALLED> my_app_name2 </APPLICATION_INSTALLED>
        <OWNED_BY_TEAM> ownersip_team </OWNED_BY_TEAM>
    </COMP>


    <ADDITIONAL_INFO>
        <SERVICES_INSTALLED>BITS,AAA,BBB</SERVICES_INSTALLED>
        <CPU_THREASHOLD_PERCENT>70</CPU_THREASHOLD_PERCENT>
        <RAM_THREASHOLD_PERCENT>90</RAM_THREASHOLD_PERCENT>
    </ADDITIONAL_INFO>

</MAIN_NODE>
```

Root Node

Child Node

First Child

Last Child

Children Nodes

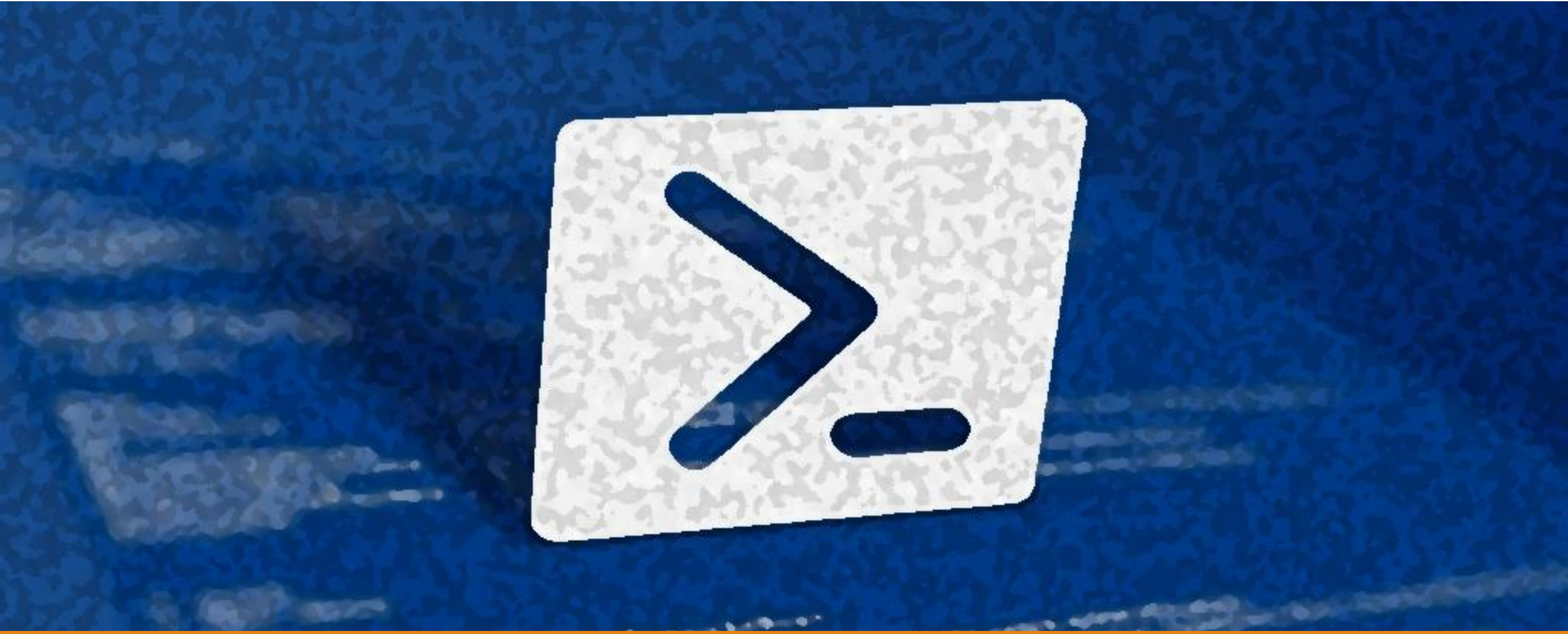
XML File Handling

```
How To read a xml file
[xml]$xml_content=Get-Content C:\Powershell\xml_file.xml
 $xml_content.GetType()
 $xml_content.GetElementsByTagName( your tag name)
```

JSON File Handling

```
How To read a JSON file
$json_object = Get-Content 'C:\path\to\your.json' | Out-String | ConvertFrom-Json

$prop1 = $json_object.Propert1
$prop2 = $json_object.Prop2
```

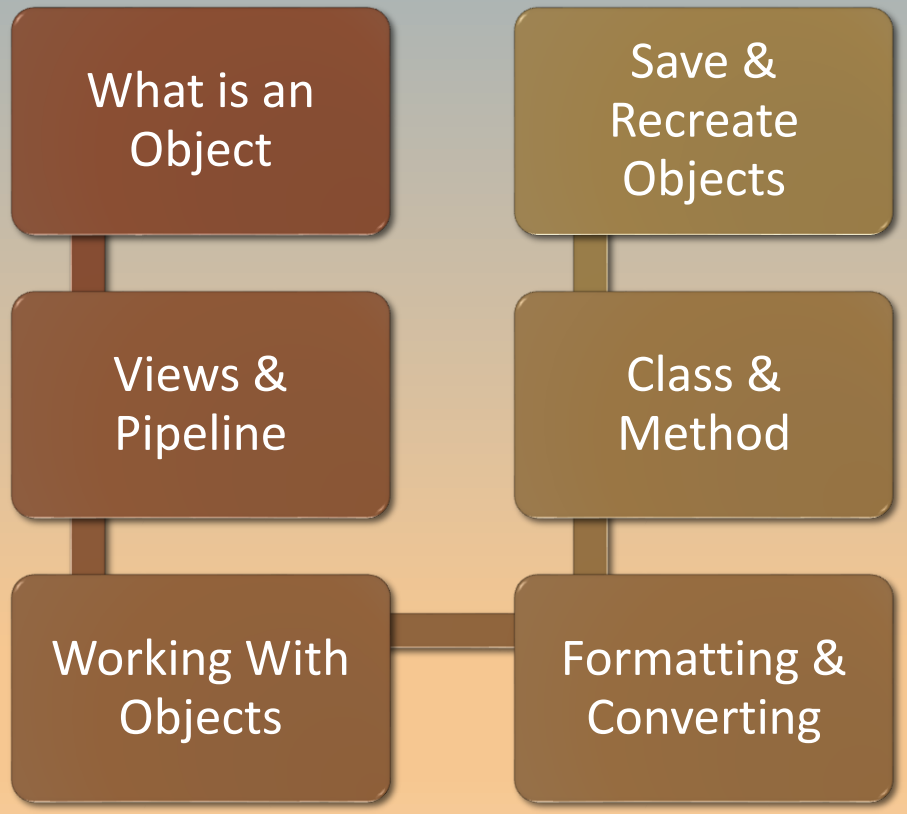
Section Completed ☺



Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI

Section 4: Objects Based PowerShell



A Programming Object

Real-world objects share two characteristics:

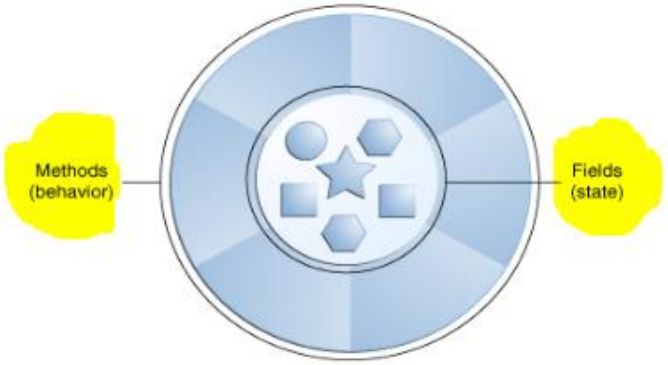
- **They all have "*state*" and "*behavior*".**
- -> Dogs have state (name, color, breed, hungry) and behavior (barking, wagging tail).
- -> Bicycles also have state (current gear, current speed) and behavior (changing gear, applying brakes).

Programming objects are conceptually similar to real-world objects:

they too consist of state and related behavior. An object stores its state in

properties/variables/fields and exposes its behavior through

methods(called as functions in some programming languages)



A software object.

A Programming Object

An object is simply the programmatic representation of anything.

It is a good practice to take a look at Get-Member cmdlet’s output to understand what exactly is particular object and what it can do.

Whatever cmdlets, we have seen so far which seems to be displaying plain text on console, None of that was plain text but they all were programmable Objects.

A Programming Object

Everything in PowerShell is an object

Proof :

In front of any PowerShell entity, do a Get-Member and observe the output, We will see both its properties and methods.

We can also see the type of object as well

```
PS C:\PowerShell\Advanced_PowerShell\Practice Lab\Section3>> "Something"
Something

PS C:\PowerShell\Advanced_PowerShell\Practice Lab\Section3>> "Something" | Get-Member


   TypeName: System.String

Name             MemberType            Definition
----             ----------            ----------
Clone            Method                System.Object Clone(), System.Object ICloneable
CompareTo        Method                int CompareTo(System.Object value), int Compare
Contains         Method                bool Contains(string value)
CopyTo           Method                void CopyTo(int sourceIndex, char[] destination
EndsWith         Method                bool EndsWith(string value), bool EndsWith(stri
Equals           Method                bool Equals(System.Object obj), bool Equals(str
GetEnumerator    Method                System.CharEnumerator GetEnumerator(), System.C
GetHashCode      Method                int GetHashCode()
GetType          Method                type GetType()
GetTypeCode      Method                System.TypeCode GetTypeCode(), System.TypeCode
IndexOf          Method                int IndexOf(char value), int IndexOf(char value
IndexOfAny       Method                int IndexOfAny(char[] anyOf), int IndexOfAny(ch
Insert           Method                string Insert(int startIndex, string value)
IsNormalized     Method                bool IsNormalized(), bool IsNormalized(System.T
```

```
ToUInt64         Method                uint64 IConvertible.ToUInt64(System.IFormatPro
ToUpper          Method                string ToUpper(), string ToUpper(cultureinfo c
ToUpperInvariant Method                string ToUpperInvariant()
Trim             Method                string Trim(Params char[] trimChars), string T
TrimEnd          Method                string TrimEnd(Params char[] trimChars)
TrimStart        Method                string TrimStart(Params char[] trimChars)
Chars            ParameterizedProperty char Chars(int index) {get;}
Length           Property              int Length {get;}
```

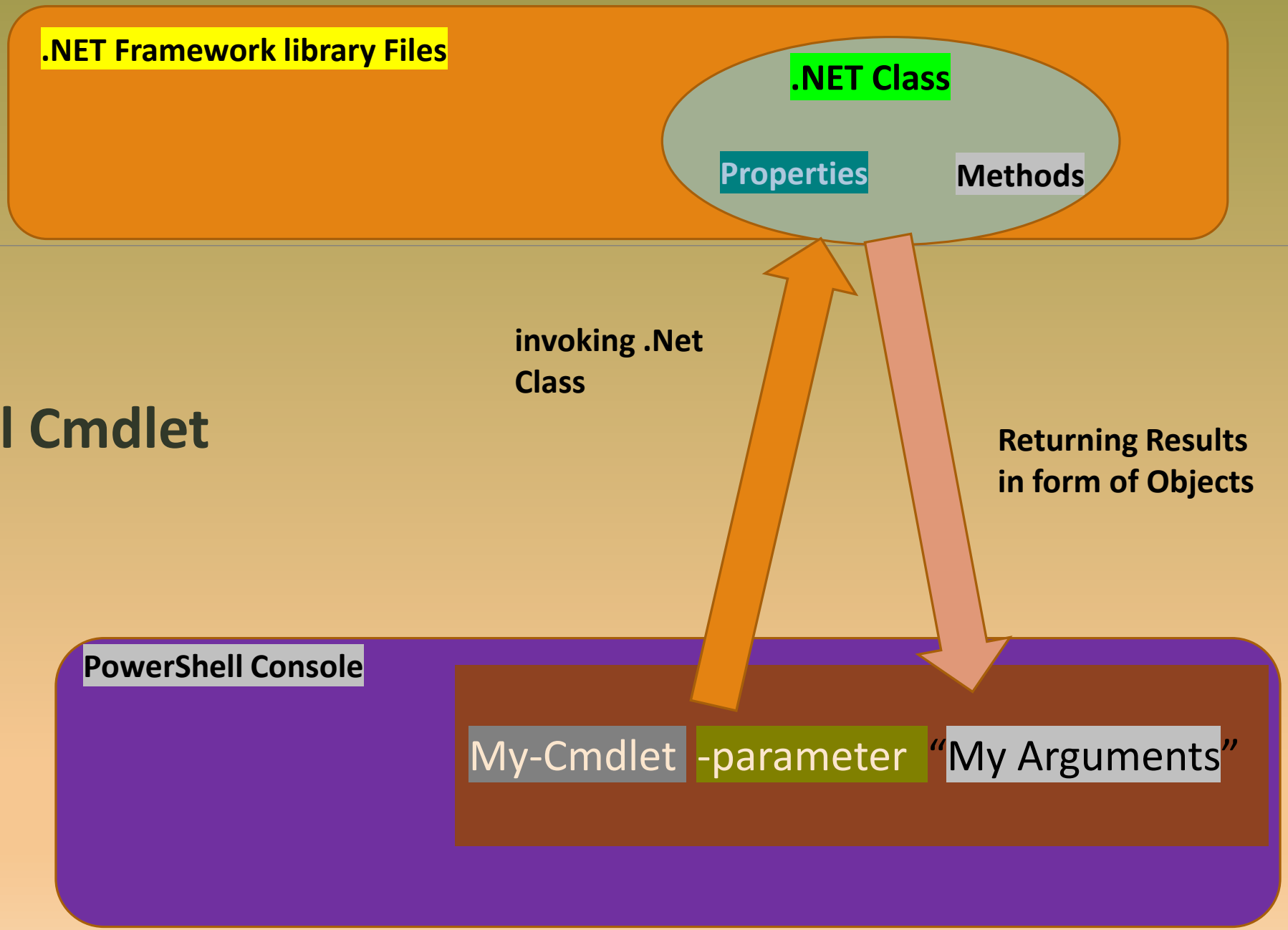
Cmdlet

A cmdlet is a lightweight command that is used in the Windows PowerShell environment. The Windows PowerShell runtime invokes these cmdlets within the context of automation scripts that are provided at the command line. The Windows PowerShell runtime also invokes them programmatically through Windows PowerShell APIs.

Most cmdlets are based on .NET Framework classes that derive from the Cmdlet base class.

```
Get-ChildItem    -Path    C:\Windows
```

PowerShell Cmdlet



Pipeline

Piping works virtually everywhere in Windows PowerShell. Although you see text on the screen, Windows PowerShell does not pipe text between commands. Instead, it pipes objects.

A pipeline is a series of commands connected by pipeline operators (|) (ASCII 124). Each pipeline operator sends the results of the preceding command to the next command.

You can use pipelines to send the objects that are output by one command to be used as input to another command for processing. And you can send the output of that command to yet another command. The result is a very powerful command chain or "pipeline" that is comprised of a series of simple commands.

Command-1 | Command-2 | Command-3

Example:

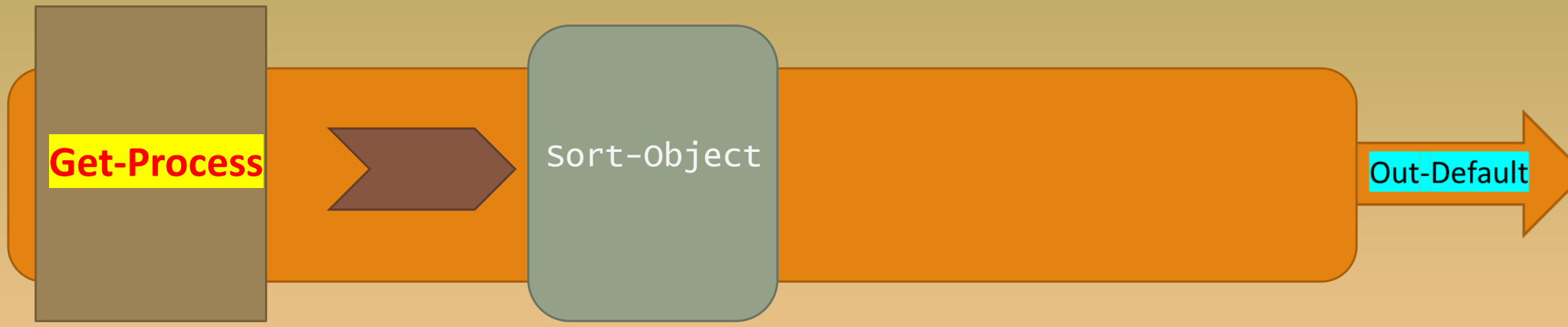
```
Get-ChildItem -Path C:\WINDOWS\ | Out-File -FilePath "OutputFile.txt"
```

Pipeline

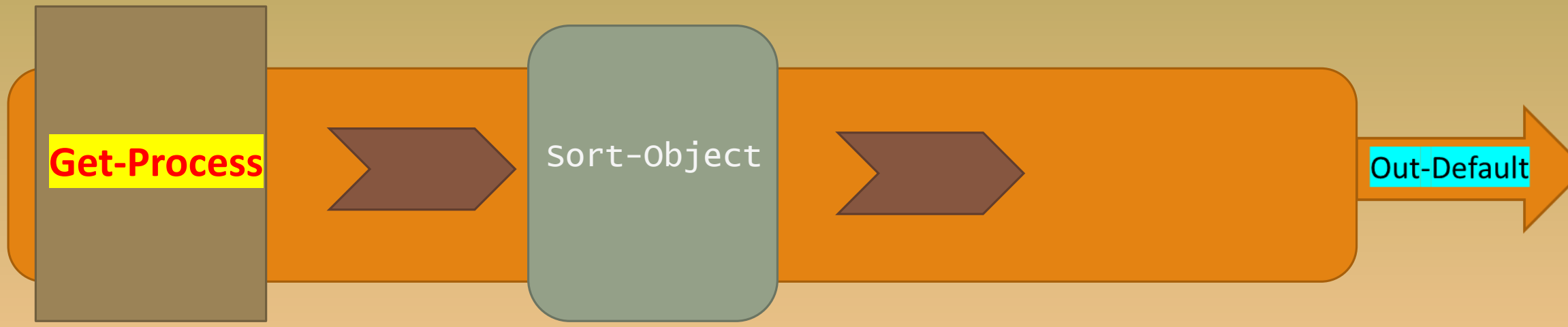
Pipeline

Pipeline

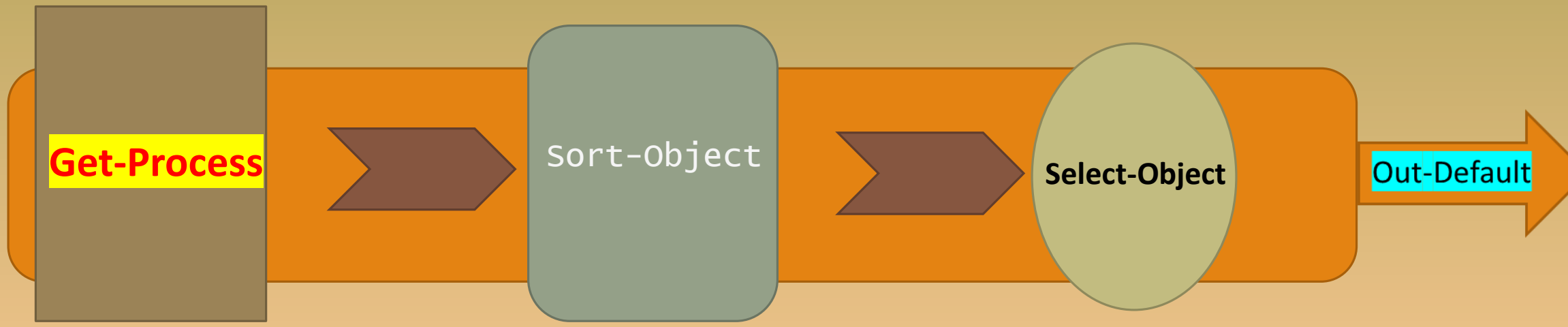
Pipeline



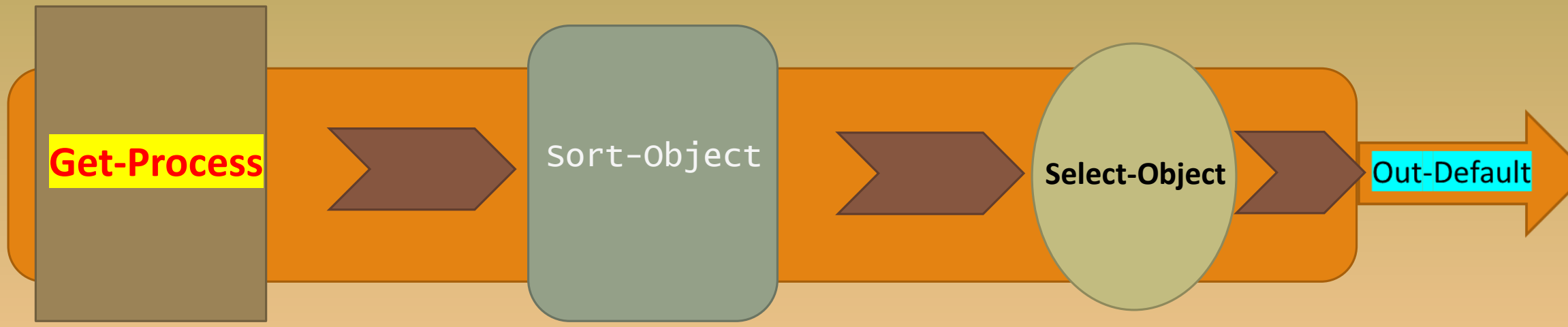
Pipeline



Pipeline



Pipeline



Working With Objects

```
Select-Object

Sort-Object

Where-Object

ForEach-Object
```

Format-List

Formats the output as a list of properties in which each property appears on a new line

Example1: Selecting few properties by name

```
Get-Service  -Name "a*" | Format-list Name,Status,DisplayName
```

Example2: Selecting all properties by wildcard(*)

```
Get-Service  -Name "a*" | Format-list *
```

Format-Table

The Format-Table cmdlet formats the output of a command as a table with the selected properties of the object in each column. The object type determines the default layout and properties that are displayed in each column, but you can use the Property parameter to select the properties that you want to see.

```
Get-Service  -Name "a*" | Format-Table Name, Status
```

PowerShell Class

A class is an extensible program-code-template for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions or methods).

So inside a class we create variables and methods and then we can create Objects for the classes and these objects can be used just like the way we were using cmdlets.

```
Get-help about_Classes -ShowWindow
```

Thank You

Save Objects for Offline Analysis

Purpose:

- Remote Troubleshooting
- Offline Analysis
- It is a similar concept to Serialization
- Can be very effective medium when we need to do remote troubleshooting in customer's machine without having access to there machine. We can ask customer to execute a PowerShell statement (which saves the required process or any other object as output) and send the output. This output can be used for analysis of Customer's system without accessing the system directly.

Save Objects for Offline Analysis

Step1: Save Object's state to a file

```
Get-Process | Select-Object -Last 4 | Export-Clixml 'process_object.xml'
```

Step2: Import File & Recreate the Object and continue analysis/debugging using it

```
$saved_processes_obj = I
mport-Clixml process_object.xml
```

```
Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id  SI ProcessName
-------  ------    -----      -----     ------     --  -- -----------
    134      10     3184       2136       2.00   8236   0 WmiPrvSE
    140      14     5276       7172       5.83   9580   0 WmiPrvSE
    284      21     8724      12312      18.53  11092   0 wmpnetwk
    193      11     1804       1788       0.09   1508   0 WUDFHost
```

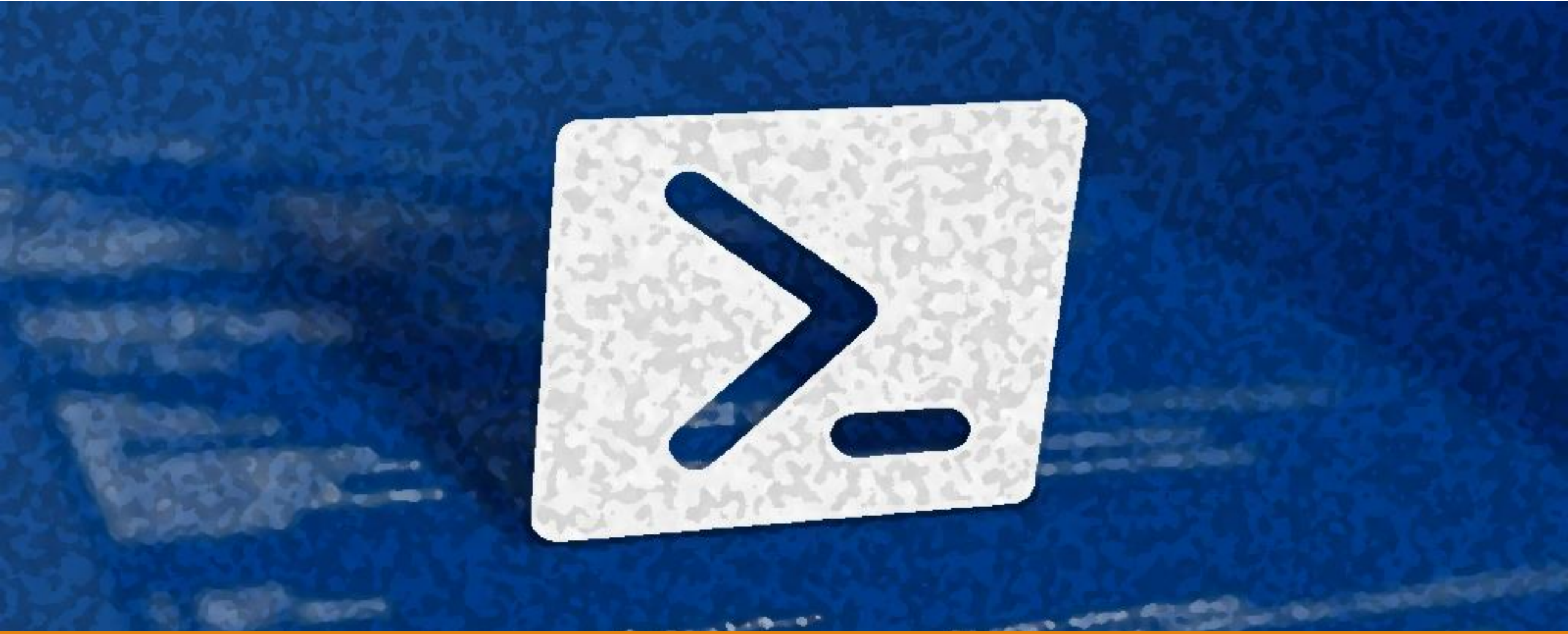
Convert Objects

```
ConvertTo-Json
ConvertTo-Csv
ConvertTo-Xml
ConvertTo-Html
```

Convert Objects

```
ConvertTo-Json
ConvertTo-Csv
ConvertTo-Xml
ConvertTo-Html
```

Section Completed ☺



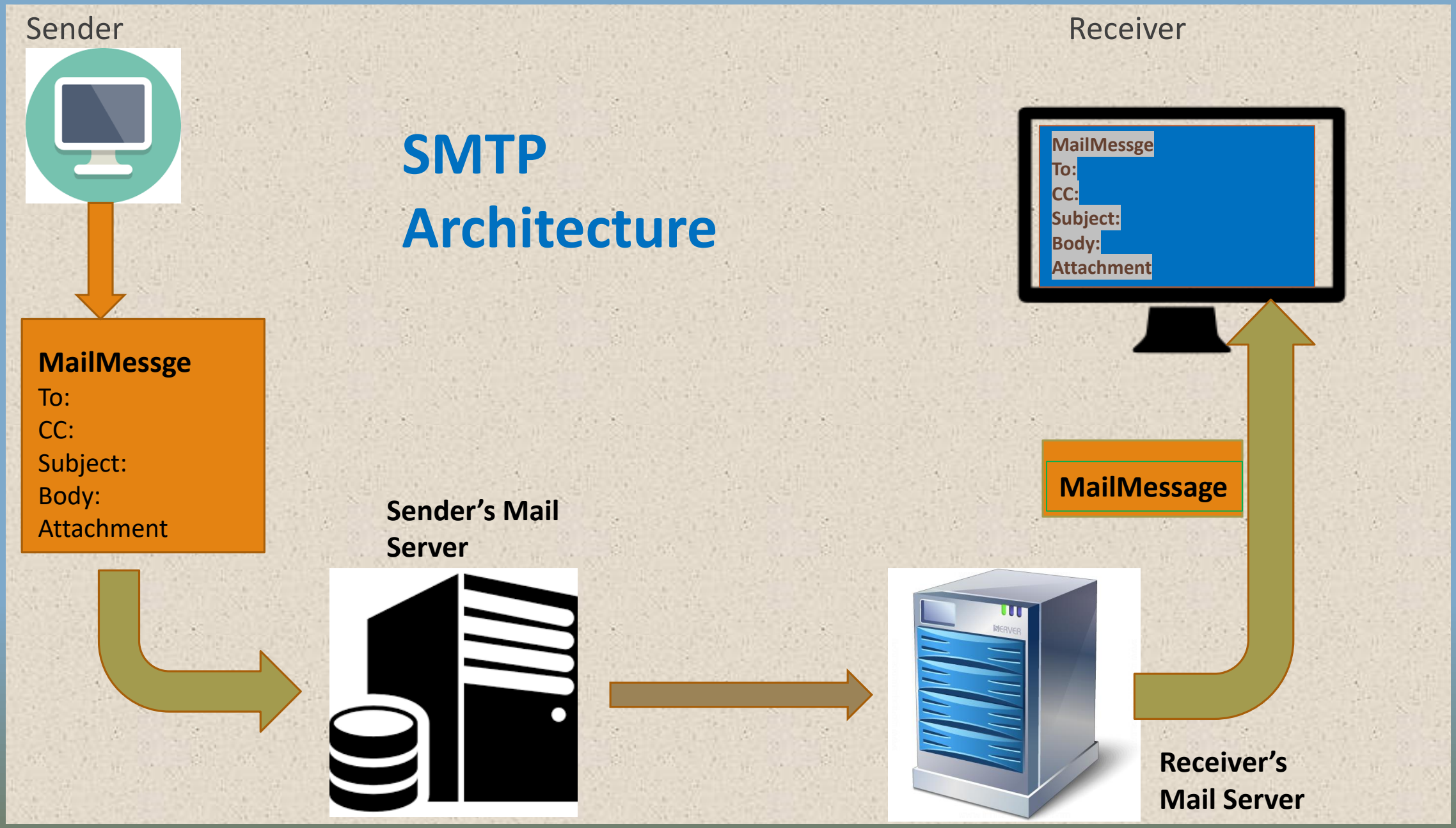
Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI

Section 5: Deep dive into PowerShell

- Sending Email
- Automation of Daily Report
- Script Signing
- Web Scrapping
- Browser based Task Automation
- ISESnippette
- Early vs Late Filtering
- Aliases
- Measure Command
- PSDrives
- Regular Expressions

SMTP Architecture



Requirement

Write a PowerShell Script to :

- Extract System's Performance Data and send it to the Performance Engineering Team on daily basis.
- Report Should be in well-defined format and it should get triggered at a fixed time.
- No Human intervention in sending or formatting this report report

Improvements

- Logging
- Error Handling
- Script should take input from external configuration file like XML

  Example: Name of Services, Email Recements list(To & CC), DB Connection details(if applicable)

Script Signing

Purpose:

Advise PowerShell to execute only those scripts which are digitally signed by a trusted certificate authority.

Types of Certificate:

1.) Certificate from certificate authorities (Example: Verisign)

2.) Self Signed Certificate

```
Get-Help about_Signing -ShowWindow
```

Script Signing

```
945 -</configuration>
946
947   <!-- SIG # Begin signature block -->
948   <!-- MIIbVQYJKoZIhvcNAQcCoIIbRjCCG0ICAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB -->
949   <!-- gjcCAQSgWzBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR -->
950   <!-- AgEAAgEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQUa2wTnBgLzRIMZiu2HsotQDcd -->
951   <!-- QS+gghYqMIIEEjCCAvqgAwIBAgIPAMEAizw8iBHRPvZj7N9AMA0GCSqGSIb3DQEB -->
952   <!-- BAUAMHAxKzApBgNVBAsTIkNvcHlyaWdodCAoYykgMTk5NyBNaWNyb3NvZnQgQ29y -->
953   <!-- cC4xHjAcBgNVBAsTFU1pY3Jvc29mdCBDb3Jwb3JhdGlvbjEhMB8GA1UEAxMYTWlj -->
954   <!-- cm9zb2Z0IFJvb3QgQXV0aG9yaXR5MB4XDTk3MDExMDA3MDAwMFoXDTIwMTIzMTA3 -->
955   <!-- MDAwMFowcDErMCkGA1UECxMiQ29weXJpZ2h0IChjKSAxOTk3IE1pY3Jvc29mdCBD -->
956   <!-- b3JwLjEeMBwGA1UECxMVTWljcm9zb2Z0IENvcnBvcmF0aW9uMSEwHwYDVQQDExhN -->
957   <!-- aWNyb3NvZnQgUm9vdCBBdXRob3JpdHkwggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAw -->
958   <!-- ggEKAoIBAQCpAr3BcOY78k4bKJ+XeF4w6qKpjSVf+P6VTKO3/p2iID58UaKboo9g -->
```

Web Scrapping

Web scraping, web harvesting, or web data extraction is data scraping used for extracting data from websites. Web scraping software may access the World Wide Web directly using the Hypertext Transfer Protocol, or through a web browser.

Web scraping a web page involves fetching it and extracting from it. Fetching is the downloading of a page (which a browser does when you view the page). Therefore, web crawling is a main component of web scraping, to fetch pages for later processing.

An example would be to find and copy names and phone numbers, or companies and their URLs, to a list.

Web Scrapping

Once fetched, then extraction can take place. The content of a page may be parsed, searched, reformatted, its data copied into a spreadsheet, and so on. Web scrapers typically take something out of a page, to make use of it for another purpose somewhere else.

Web pages are built using text-based mark-up languages (HTML and XHTML), and frequently contain a wealth of useful data in text form.

However, most web pages are designed for human end-users and not for ease of automated use. Because of this, tool kits that scrape web content were created. A web scraper is an Application Programming Interface (API) to extract data from a web site.

Source: Wikipedia

Web Scrapping Overview

Invoke-WebRequest

Synopsis:

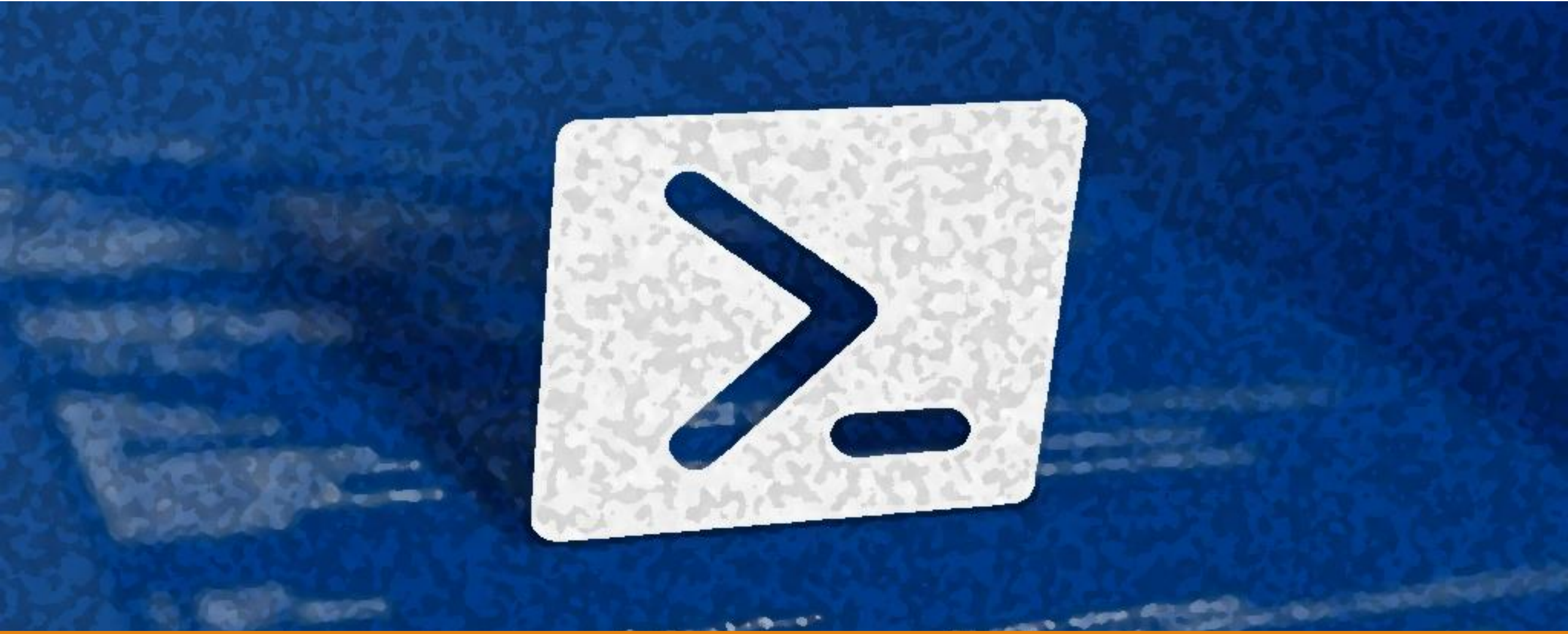
Gets content from a web page on the Internet.

Description:

The Invoke-WebRequest cmdlet sends HTTP, HTTPS, FTP, and FILE requests to a web page or web service. It parses the response and returns collections of forms, links, images, and other significant HTML elements. This cmdlet was introduced in Windows PowerShell 3.0.

```
Get-Help Invoke-WebRequest -ShowWindow
```

Thank You



Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI

Section 6: PowerShell-Database Interaction and CRUD

- Database Basics: A quick Wrap up
- Connecting PowerShell with Database
- Reading Database Tables
- Update Operation
- Delete Operation
- Automation: Solve a real life problem

Database

A database is an organized collection of data. Databases support storage and manipulation of data. Databases make data management easy.

Example:

Microsoft SQL Server

Oracle

MySQL

SQLite

Relation Database Management System(RDBMS):

A relational database is a tabular database in which data is defined so that it can be reorganized and accessed in a number of different ways.

A relational database, more restrictively, is a collection of schemas, tables, queries, reports, views, and other elements.

Structured Query Language (SQL):

It is the standard user and application program interface for a relational database. Relational databases are easy to extend, and a new data category can be added after the original database creation without requiring that you modify all the existing applications.

Process:

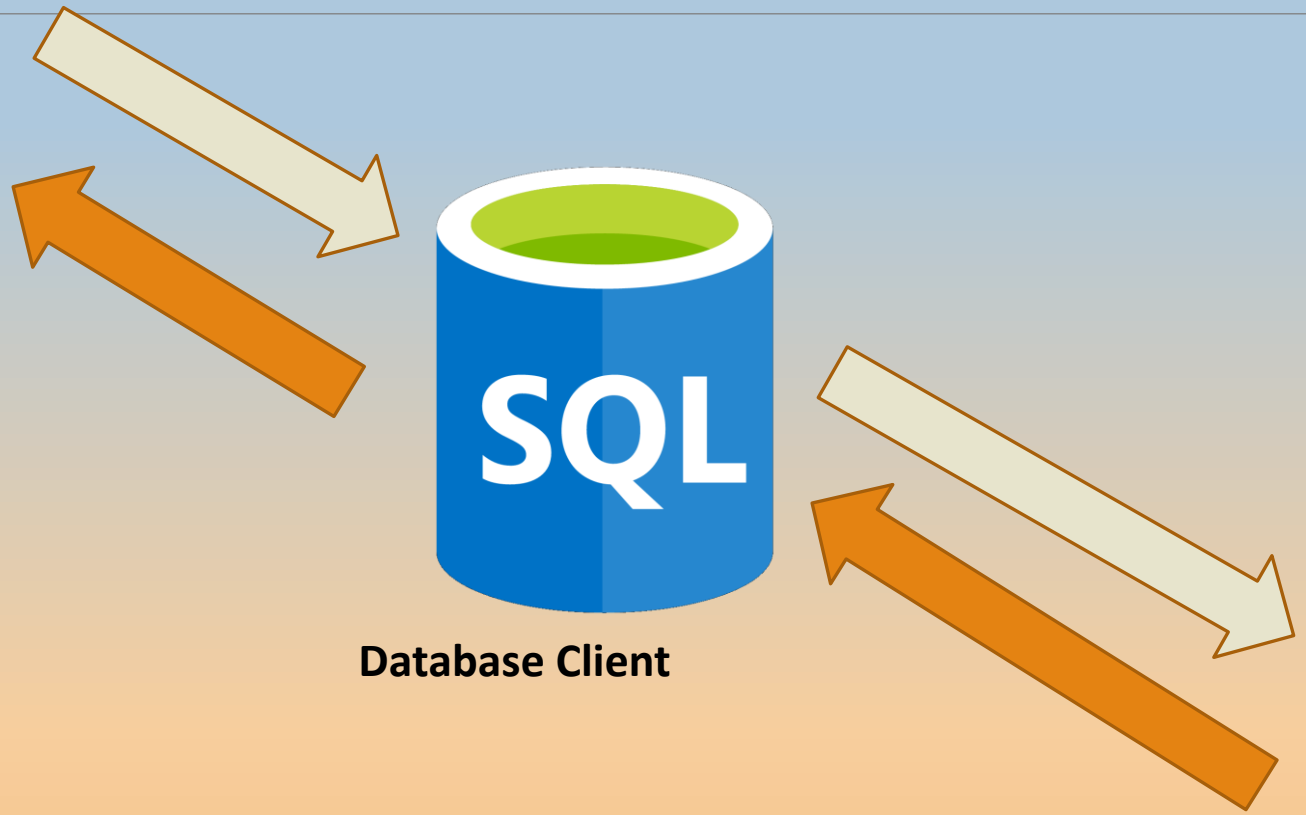
1.) Open a connection with database using a client and pass valid credentials

2.) Use the appropriate command

Create, Read , Update, Delete

3.) Close the connection once done

Database



Database Client

PowerShell

Database : How it can help us

- Get benefitted from mature data storage and management model
- Store and maintain server information at a centralized place
- Log the important information into database table
- Better Security Models
- Improved data security
- Store the User Login-Logout Activities for long term
- We can restore the information using backup in case something goes wrong

Automation: Pull Report from DB and Send

Requirement:

Write a PowerShell Script Solution for automating a manual activity which involves

- Connecting to remote database using specified credentials
- Execute predefined queries on database and fetch results
- Convert the output into beautiful HTML report for better readability.
- Send the output to a list of specified people

Please note that script should execute automatically at 5AM every morning without ‘any’ human intervention.

Script will be required to deploy on multiple servers. So it’s deployment should be easy and it should be re-configurable without making the change in script.

Benefits

- Preparing Report for 1 environment -> 20 minutes
- Number of different environments(DC) -> 5 environments

- Total Effort Saved(Per Year) -> 20 * 5 * 30(days) * 12(months

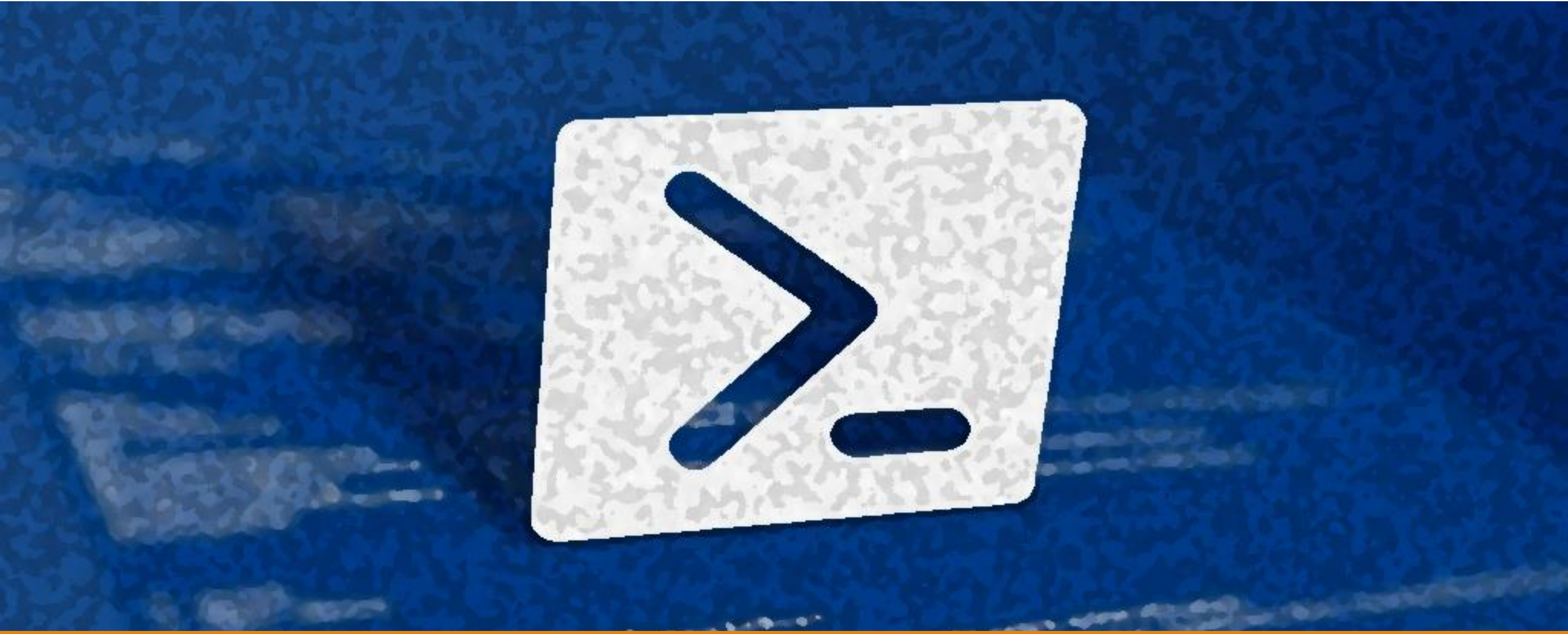
  = 36000 minutes /year

  => 600 hours per year

What additionally you can do to enhance it further?

- Combine Service/Process/Event/Scheduled task status into one report and send on daily basis.
- Just for knowledge purpose can you write a PowerShell script which can adjust itself as per the database type, Lets say there is one more column DB_TYPE in configuration which can have values like MS-SQL,ORACLE,DB2,MySQL etc and inside our code we need to check this column type before loading the appropriate dll(client). This can be done to improve the reusability of this solution across multiple platform
- Make use of SQL queries in such a way that they can be modified dynamically at run time like Where date=$(Get-Date)
- Allow multiple query output to be displayed in output
- Sign the script

Thank You



Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI

Section 7: Windows Management Instrumentation

- Introduction
- GUI tools
- WQL
- Mastering WMI
- Advantages

Introduction

- WMI(Windows management Instrumentation) is Microsoft's implementation of the Web-Based Enterprise Management (WBEM) and Common Information Model (CIM) standards from the Distributed Management Task Force (DMTF).

- WMI allows scripting languages (such as VBScript or Windows PowerShell) to manage Microsoft Windows personal computers and servers, both locally and remotely. WMI comes preinstalled in Windows 2000 and in newer Microsoft OSes.

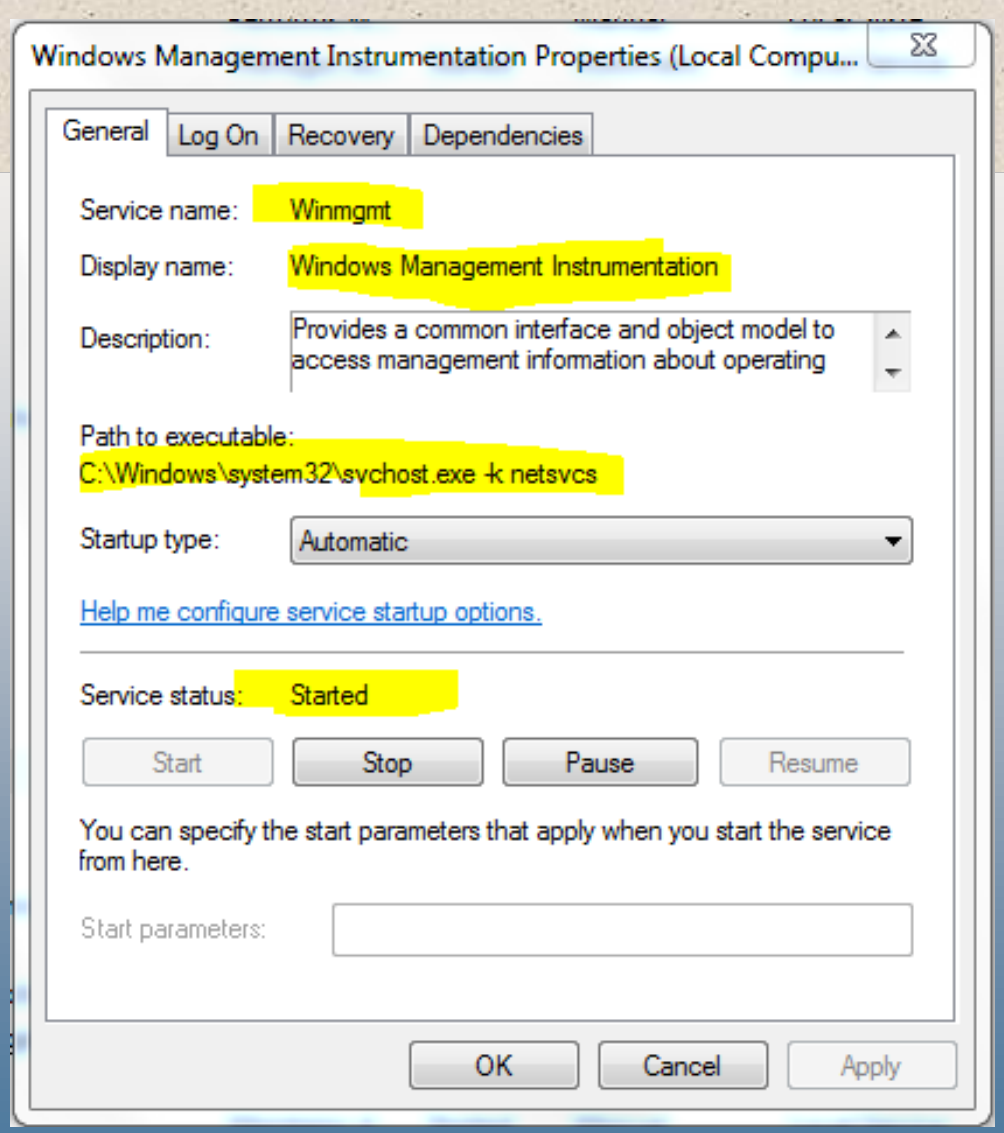
CIM

Common Information Model (CIM), a computer industry standard for defining device and application characteristics so that system administrators and management programs can control devices and applications from multiple manufacturers or sources in the same way.

WMI Purpose

- WMI provides users with information about the status of local or remote computer systems.
- It also supports such actions as the configuration of security settings, setting and changing system properties, setting and changing permissions for authorized users and user groups, assigning and changing drive labels, scheduling processes to run at specific times, backing up the object repository, and enabling or disabling error logging.

WMI


Windows Management Instrumentation Properties (Local Compu...
General
Log On
Recovery
Dependencies
Service name:
Winmgmt
Display name:
Windows Management Instrumentation
Description:
Provides a common interface and object model to access management information about operating
Path to executable:
C:\Windows\system32\svchost.exe -k netsvcs
Startup type:
Automatic
Help me configure service startup options.
Service status:
Started
Start
Stop
Pause
Resume
You can specify the start parameters that apply when you start the service from here.
Start parameters:
OK
Cancel
Apply

WMI Service Running

Request (WMI Query)

Response (Collections of Objects)

Using WMI CIM We can:

- Connect to a chosen system and browse the CIM repository in any namespace available.
- Search for classes by their name, by their descriptions or by property names.
- Review the properties, methods and associations related to a given class.
- See the instances available for a given class of the examined system.
- Perform Queries in the WQL language.

Namespaces

There are several Windows Management Instrumentation (WMI) namespaces created which are aligned to each major product, and depending on the namespace, hundreds of classes can be created under each namespace.

Example:

root\directory\ldap

root\Microsoft\SqlServer\ComputerManagement12\instance_name

root\CCM\Scheduler

root\cimv2

root\CCM\Events

Get-WmiObject (Alias: gwmi)

GWMI can be used for

- Fetching all the properties available for a class, example:

```
Get-WmiObject -Class 'Win32_Service' -ComputerName 'localhost'
```

- Fetching results using a query, example:

```
Get-WmiObject -ComputerName 'localhost' -Query "select * from Win32_Service"
```

WQL

```
43
44  #WMI on remote machines
45  Get-WmiObject -Class 'Win32_Service' -ComputerName 'localhost'
46
47
48
```

```
PS C:\PowerShell\section7> Get-WmiObject -Class 'Win32_Service' -Comput


ExitCode  : 1077
Name      : ADWS
ProcessId : 0
StartMode : Disabled
State     : Stopped
Status    : OK

ExitCode  : 1077
Name      : AJRouter
ProcessId : 0
StartMode : Manual
State     : Stopped
Status    : OK
```

WQL (WMI Query Language)

- Windows Management Instrumentation Query Language (WQL) is Microsoft's implementation of the CIM Query Language (CQL), a query language for the Common Information Model (CIM) standard from the Distributed Management Task Force (DMTF).
- It is a subset of ANSI standard SQL with minor semantic changes
- https://msdn.microsoft.com/en-us/library/aa394606(v=vs.85).aspx

WQL (WMI Query Language)

```
"SELECT WPP.IDProcess, WPP.PercentProcessorTime, WPP.PrivateBytes, WPP.HandleCount,
WNC.NumberBytesinallHeaps FROM Win32_PerfFormattedData_PerfProc_Process AS WPP
INNER JOIN Win32_PerfFormattedData_NETFramework_NETCLRMemory AS WNC ON
WPP.IDProcess = WNC.IDProcess WHERE WPP.Name LIKE 'MyAppName%' " &_ "AND WNC.Name
LIKE 'MyAppName%'",,48
```

Few Important WMI Classes

- Win32_OperatingSystem
- Win32_LogicalDisk
- Win32_Service
- Win32_Process
- win32_PhysicalMemory
- Win32_ComputerSystem

https://msdn.microsoft.com/en-us/library/aa394173(v=vs.85).aspx

DriveType

Data type: **uint32**
Access type: Read-only
Qualifiers: **MappingStrings** ("Win32API|FileFunctions|GetDriveType")

Numeric value that corresponds to the type of disk drive this logical disk represents.

Unknown (0)

No Root Directory (1)

Removable Disk (2)

Local Disk (3)

Network Drive (4)

Compact Disc (5)

RAM Disk (6)

Automation: Requirement

Write an automation to gather the disk space data of multiple disk drives of multiple servers.

Output should be send as an email and disk should be categorized as :

Good: If % free space is above 20%

Warning: If %free space is less than 20% but more than 10%

Error: If % free space is less than 10%

Requirement

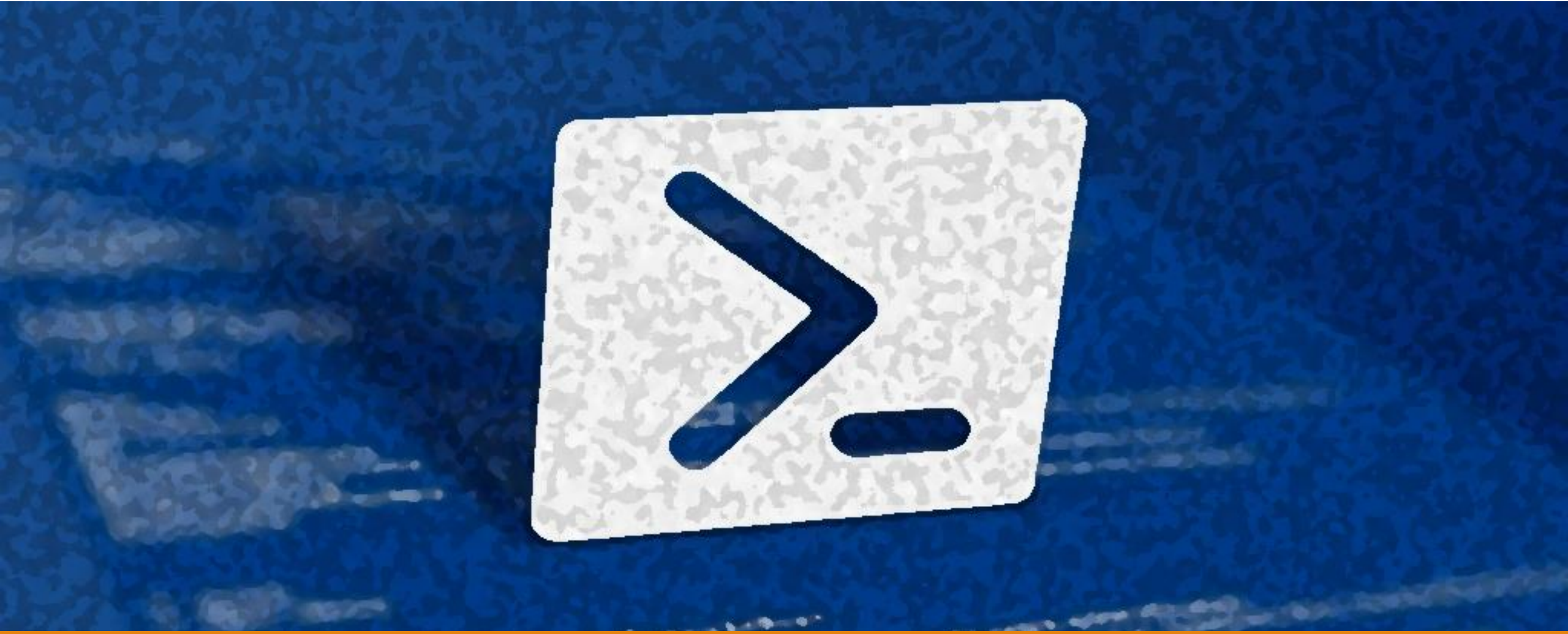
Script can be scheduled to send output in multiple ways:

-> Daily email containing disk space status of all the servers

-> Every hour and send status of only the servers which are in error state

We should use a configuration file for feeding our preferences to script

Thank You



Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI

Section 8 : Event Viewer & Task Scheduler Automation

- Event Viewer
- Access Event Viewer using PowerShell
- Write Into Event Logs
- Task Scheduling using PowerShell
- Automation of event logs delivery

Event Viewer

Event Viewer is a component of Microsoft's Windows NT line of operating systems that lets administrators and users view the event logs on a local or remote machine.

Event Viewer allows you to monitor events in your system. It maintains logs about program, security, and system events on your computer. You can use Event Viewer to view and manage the event logs, gather information about hardware and software problems, and monitor security events.

To access Device Manager, on the Start menu, click Programs , point to Administrative Tools , and then click Event Viewer .

Event Logs

We can use Event Viewer to view and manage the System, Application, and Security event logs.

System Log: The System log records events logged by the Windows system components.

For example, the failure of a driver or other system component to load during startup is recorded in the System log.

Application Log: The Application log records events logged by programs.

For example, a database program might record a file error in the Application log. Program developers decide which events to monitor.

Security Log: The Security log records security events, such as valid and invalid logon attempts, and events related to resource use, such as creating, opening, or deleting files or other objects. The Security Log helps track changes to the security system and identify any possible breaches to security. For example, attempts to log on the system might be recorded in the Security log, if logon and logoff auditing are enabled.

You can view the Security log only if you are an administrator for a computer.

Event Types

Table 14.6 Event Types and Definitions

Event Type	Definition
Error	A significant problem, such as loss of data or loss of functionality.
Warning	An event that might not be significant, but might indicate a future problem.
Information	An event that describes the successful operation of an application, driver, or service.
Success Audit	An audited security access attempt that succeeds.
Failure Audit	An audited security access attempt that fails.

Source: https://technet.microsoft.com/en-us/library/cc938674.aspx

PowerShell Cmdlet to access the event logs

```
Get-EventLog

Get-WmiObject -class Win32_NTLogEvent
```

Writing into Event Logs

The **New-EventLog** cmdlet creates a new classic event log on a local or remote computer. It can also register an event source that writes to the new log or to an existing log.

Register the source of event to one of the existing log

```
New-EventLog -LogName 'Application' -Source “My Script“
```

The New-EventLog cmdlet can be used not only to create a brand new event log on the computer, but it can also create a new source that can be used when you write to the event log.

```
New-EventLog -Source "MyApp" -LogName "MyApp_Log" -MessageResourceFile
      "C:\MyApp_Log.dll"
```

Writing into Event Logs

```
Write into event viewer logs
Write-EventLog –LogName Application
                 –Source “My Script” –EntryType Error
                 –EventID 1  –Message “This is a test message.”
```

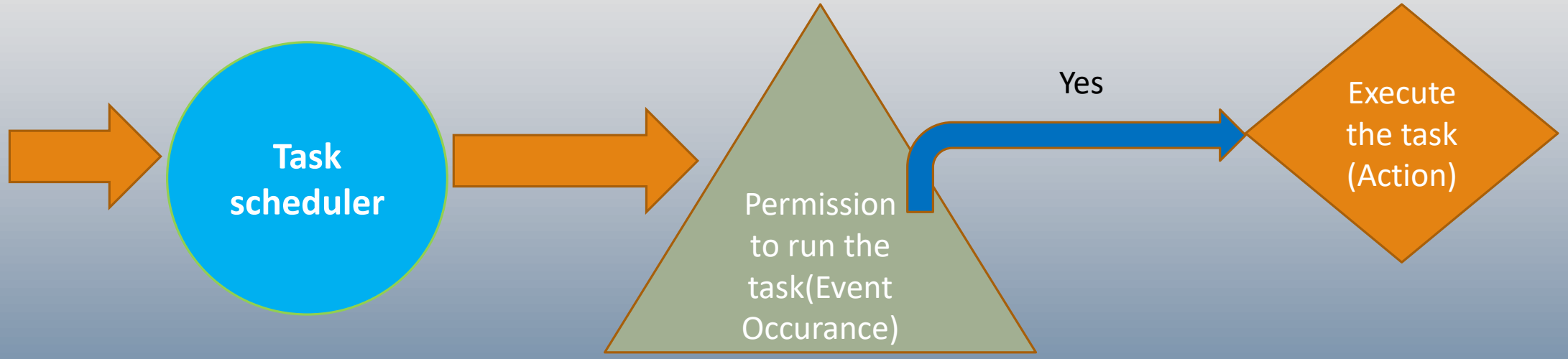
Task Scheduler

Purpose:

The Task Scheduler enables you to automatically perform routine tasks on a chosen computer. The Task Scheduler does this by monitoring whatever criteria you choose to initiate the tasks (referred to as triggers) and then executing the tasks when the criteria is met.

It is a job scheduler which run some sort of script or take some action whenever a well defined situation gets satisfied.

Task Scheduler



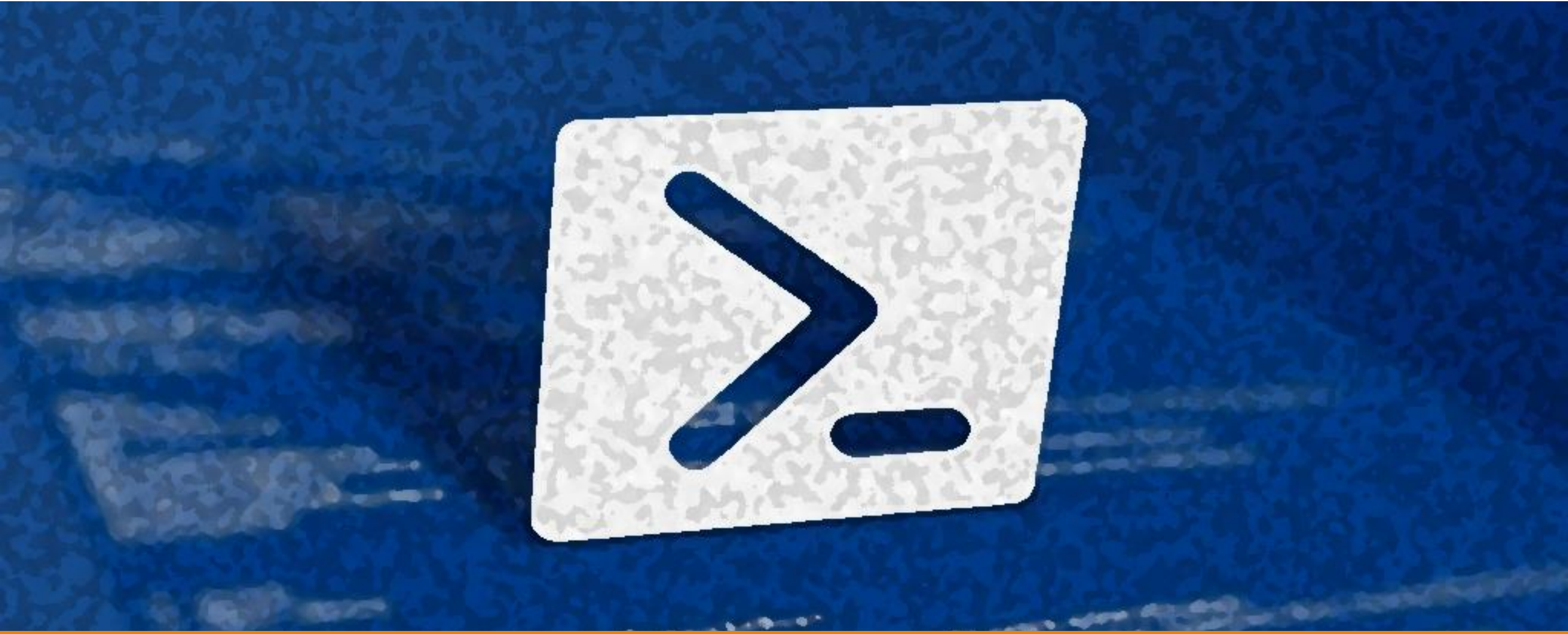
Automation

Requirement:

Write an automation using PowerShell to check the event viewer for a particular type of event and inform the support team with details in case event has occurred in last 10 minutes.

Schedule a task in Task Scheduler for running the above script in every 10 minutes.

Thank You



Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI

Section 9 : Advanced Functions

- Introduction to Advanced Function
- Advanced Function’s Parameter & validation
- Advanced Function Structure
- Exploring Advanced Function | Automation 1 & 2
- PowerShell Modules

Advanced Functions

Advanced functions allow you to write functions that can perform operations that are similar to the operations you can perform with cmdlets.

Advanced functions are helpful when you want to quickly write a function without having to write a compiled cmdlet using a Microsoft .NET Framework language.

These functions are also helpful when you want to restrict the functionality of a compiled cmdlet or when you want to write a function that is similar to a compiled cmdlet.

Advanced Function: CmdletBinding()

The CmdletBinding attribute is an attribute of functions that makes them operate like compiled cmdlets that are written in C#, and it provides access to features of cmdlets.

Windows PowerShell binds the parameters of functions that have the CmdletBinding attribute in the same way that it binds the parameters of compiled cmdlets.

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions_cmdletbindingattribute?view=powershell-6

Advanced Function: CmdletBinding()

1. The ability to add [Parameter()] decorators to parameters
2. The ability to use Write-Verbose and Write-Debug in your script or function, and have their output controlled by -Verbose and -Debug parameters of that script or function
3. Your script or function picks up the other common parameters, too, like -EV and -EA
4. The ability to have -whatif and -confirm added to your script or function

Advanced Function: Confirmation Methods

SupportsShouldProcess

The SupportsShouldProcess argument adds Confirm and WhatIf parameters to the function.

The Confirm parameter prompts the user before it runs the command on each object in the pipeline. The WhatIf parameter lists the changes that the command would make, instead of running the command.

The SupportsShouldProcess tells the shell that your function supports both -confirm and -whatif. The way you actually implement that support is to write conditional code around whatever dangerous stuff your cmdlet is planning to do:

ShouldContinue

This method is called to request a second confirmation message. It should be called when the ShouldProcess method returns $true

Advanced Function: Structure

```
Begin
  {
        # Initialize variables
  }

  Process
  {
        # Body of the function
  }

  End
  {
        # Clean-up
  }
```

Advanced Function: Structure

Begin

This block is used to provide optional one-time preprocessing for the function. The Windows PowerShell runtime uses the code in this block one time for each instance of the function in the pipeline.

Process

This block is used to provide record-by-record processing for the function. This block might be used any number of times, or not at all, depending on the input to the function.

For example, if the function is the first command in the pipeline, the Process block will be used one time. If the function is not the first command in the pipeline, the Process block is used one time for every input that the function receives from the pipeline. If there is no pipeline input, the Process block is not used.

End

This block is used to provide optional one-time post-processing for the function.

Advanced Function

Write an advanced function for getting the sum of whatever numbers are passed to it via pipe

Advanced Function : Practice

Web Scrapping Related Automation using Advanced Function

Invoke-WebRequest

Automation Requirement

Write a reusable advanced function such that it can be used for getting the status one or more URLs.

Function should be written in such a way that debugging remains easy and function could be used for monitoring of multiple websites.

Advanced Function Documents

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions_advanced_parameters?view=powershell-6

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions_advanced_methods?view=powershell-6

https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_functions_advanced_parameters?view=powershell-6

https://blogs.technet.microsoft.com/poshchap/2014/10/24/scripting-tips-and-tricks-cmdletbinding/

PowerShell Modules

A module is a set of related Windows PowerShell functionalities, grouped together as a convenient unit (usually saved in a single directory). By defining a set of related script files, assemblies, and related resources as a module, you can reference, load, persist, and share your code much easier than you would otherwise.

https://docs.microsoft.com/en-us/powershell/developer/module/understanding-a-windows-powershell-module

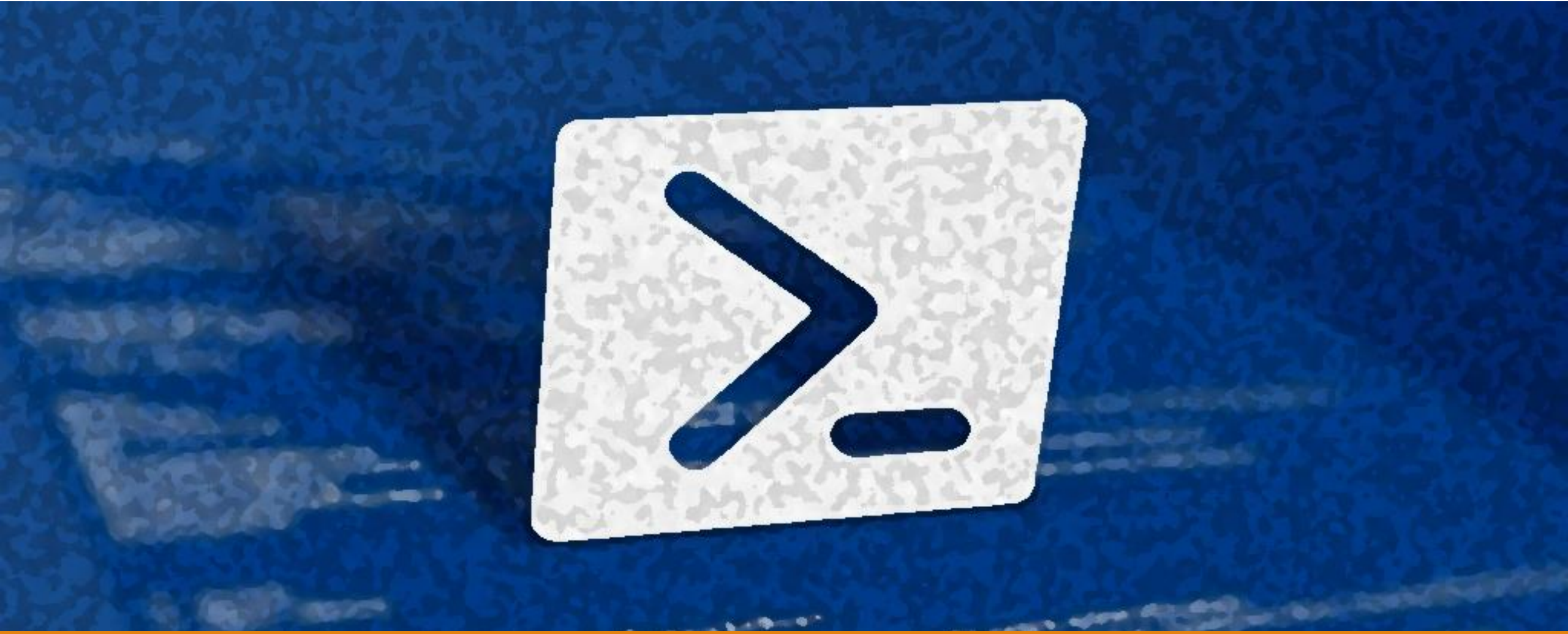
Module Components and Types

A module is made up of four basic components:

- Some sort of code file - usually either a PowerShell script or a managed cmdlet assembly.
- Anything else that the above code file may need, such as additional assemblies, help files, or scripts.
- A manifest file that describes the above files, as well as stores metadada such as author and versioning information..
- A directory that contains all of the above content, and is located where PowerShell can reasonably find it.

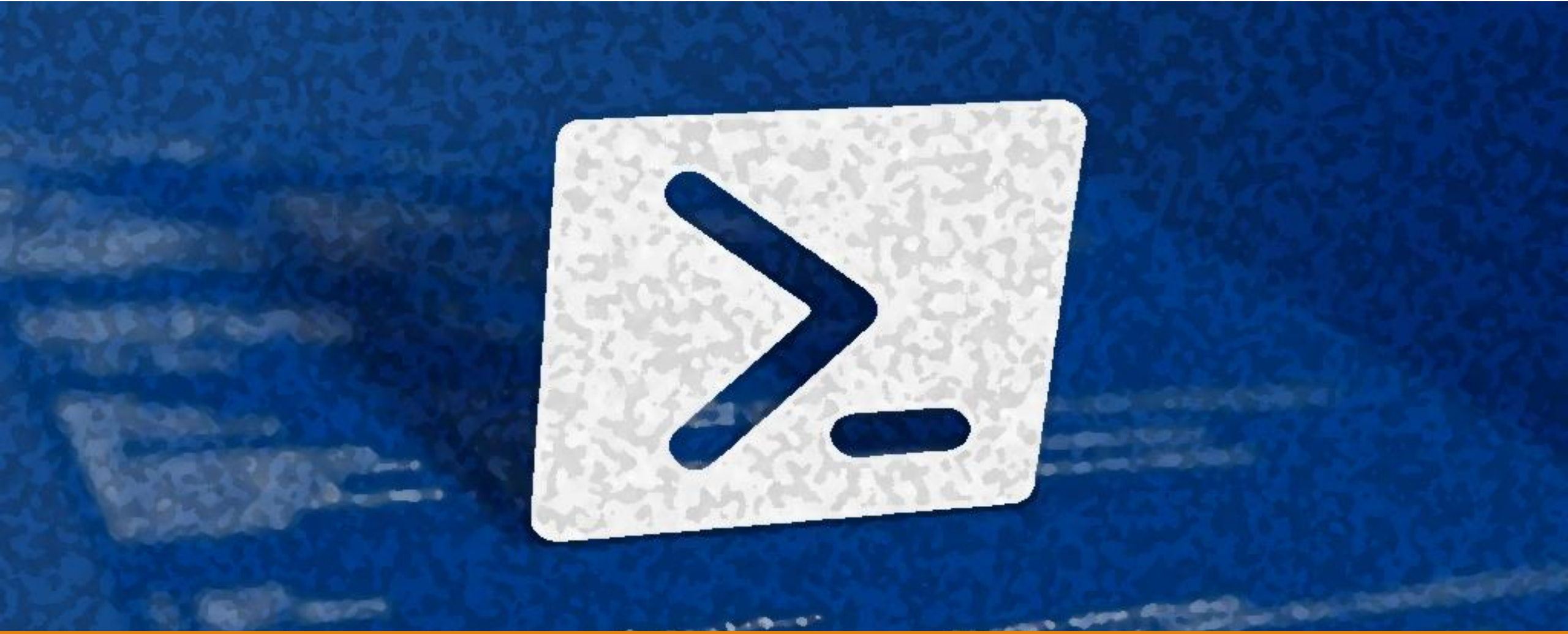
Thank You

Thank You



Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI



Advanced Scripting & Tool Making using Windows PowerShell

VIJAY SAINI

Section 10: Building Graphical User Interfaces

- GUI Advantages / Disadvantages
- Building GUI Using PowerShell
- Building a Simple Form
- Visual Studio for GUI Development
- Sample GUI working Scripts (for self learning)

Simple Interest

User Inputs:

P – Principle

R - Rate Of Interest

T – Time(Tenure period) in years

```
$simple_interest = ($principle * $rate_of_interest * $time) / 100
```

GUI

Advantages:

- Easiness for non-technical people
- Easy to train people
- Programmer or user need not have to understand working of the computer system.
- No prior knowledge is required
- GUIs generally provide users with immediate, visual feedback about the effect of each action

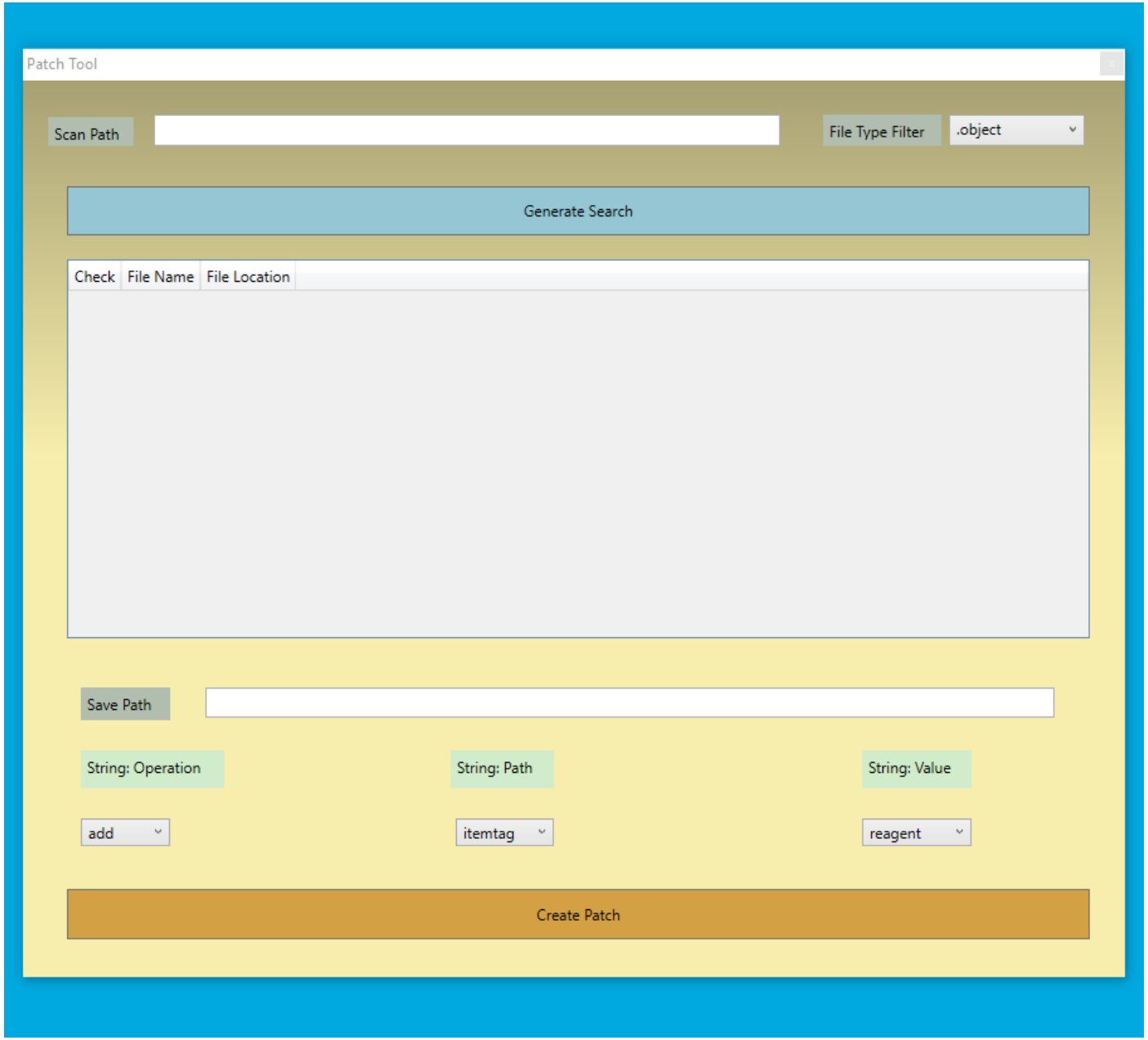
Disadvantages:

- Slower than command line tools
- Development process is slow

GUI: Different form elements

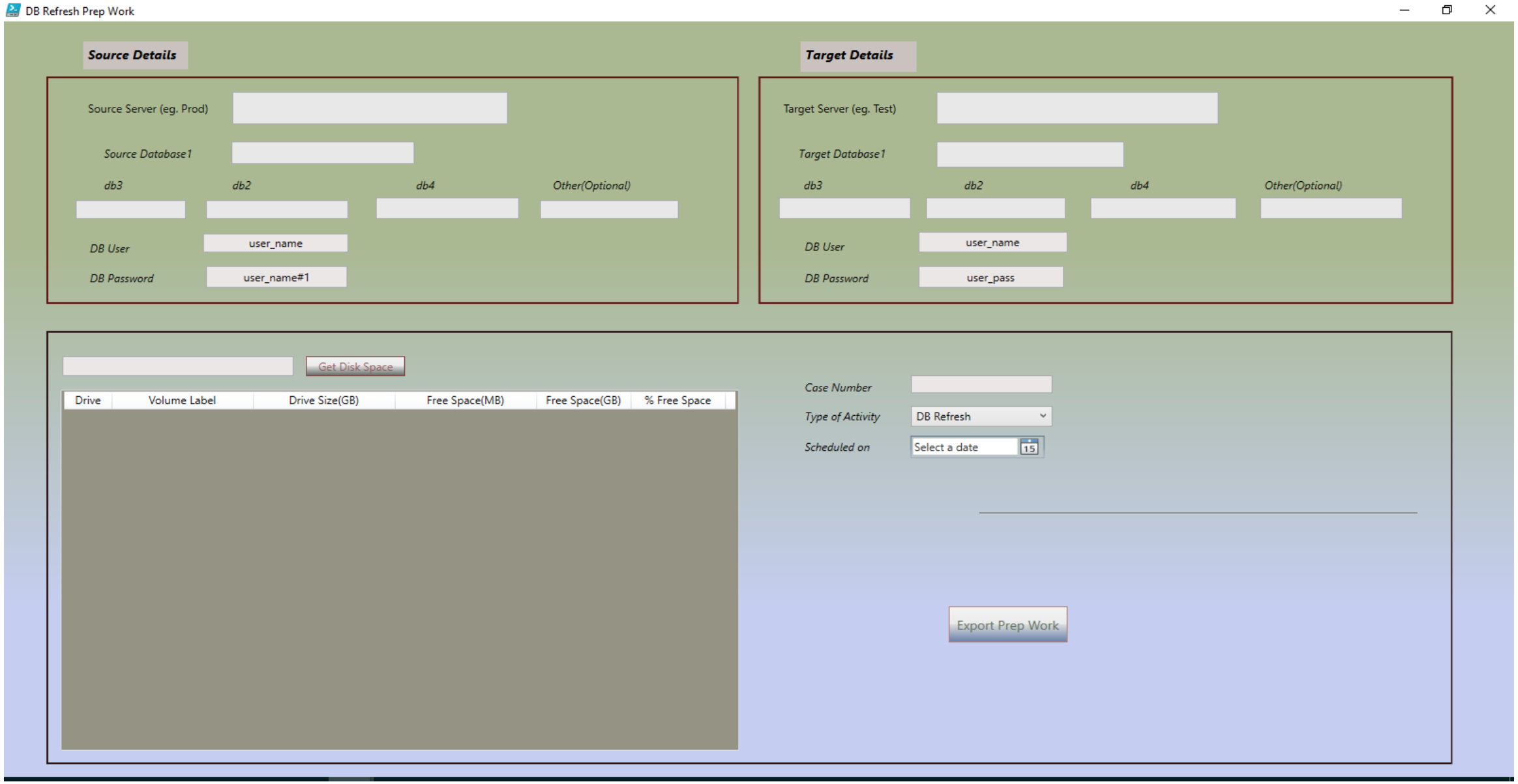
These are few commonly used form elements.

- ✓ TextBox / TextArea
- ✓ Label
- ✓ Button
- ✓ Checkbox
- ✓ SelectBox / DropDown
- ✓ DataList / DataGrid



My GUI Form –Patch Tool

My GUI Form –Space Projection



What is an GUI (Form) Event

An event is an action or occurrence recognized by software

An event handler is a callback subroutine that handles inputs received in a program

https://en.wikipedia.org/wiki/Event_(computing)

Mouse events [edit]

A pointing device can generate a number of software recognisable pointing device gestures. A mouse can generate a number of mouse events, such as mouse move (including direction of move and distance), mouse left/right button up/down[3] and mouse wheel motion, or a combination of these gestures.

Keyboard events [edit]

Pressing a key on a keyboard or a combination of keys generates a keyboard event, enabling the program currently running to respond to the introduced data such as which key/s the user pressed.[3]

Joystick events [edit]

Moving a joystick generates an X-Y analogue signal. They often have multiple buttons to trigger events. Some gamepads for popular game boxes use joysticks.

Touchscreen events [edit]

The events generated using a touchscreen are commonly referred to as touch events or gestures.

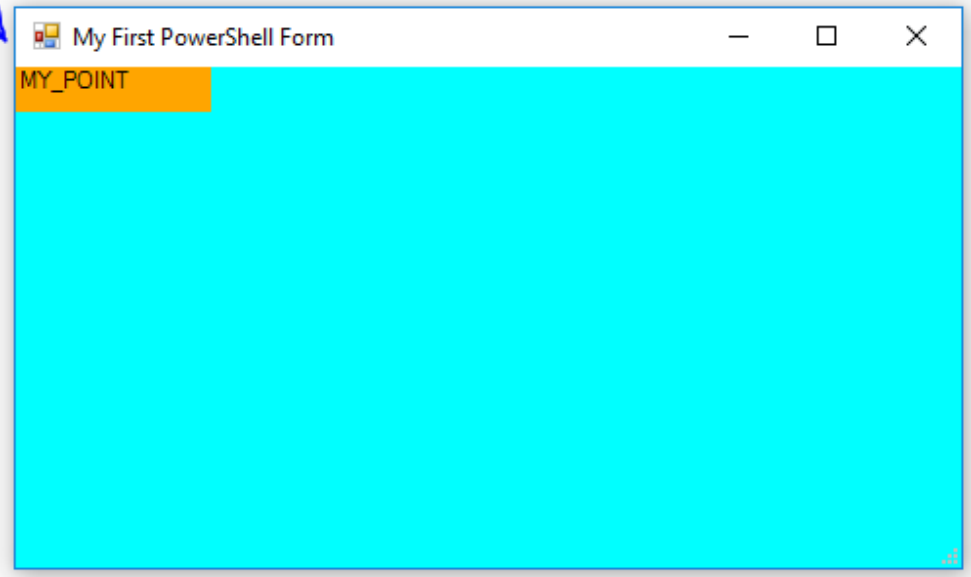
Device events [edit]

Device events include action by or to a device, such as a shake, tilt, rotation, move etc.

How to: Position Controls on Windows Forms

```
$label.Location = New-Object Drawing.Point x-coordinate, y-coordinate
```

```
#Example 2.2: Creating Label
$form= New-Object Windows.Forms.Form

$Form.width = 500
$Form.height = 300
$form.text="My First PowerShell Form"
$form.BackColor="cyan"

# Create the label control and
#set text, size and location
$label = New-Object Windows.Forms.Label
$label.text = "MY_POINT"

$label.Location = New-Object Drawing.Point 00,00

$label.BackColor='orange'
$form.controls.add($label)

$form.ShowDialog()
```

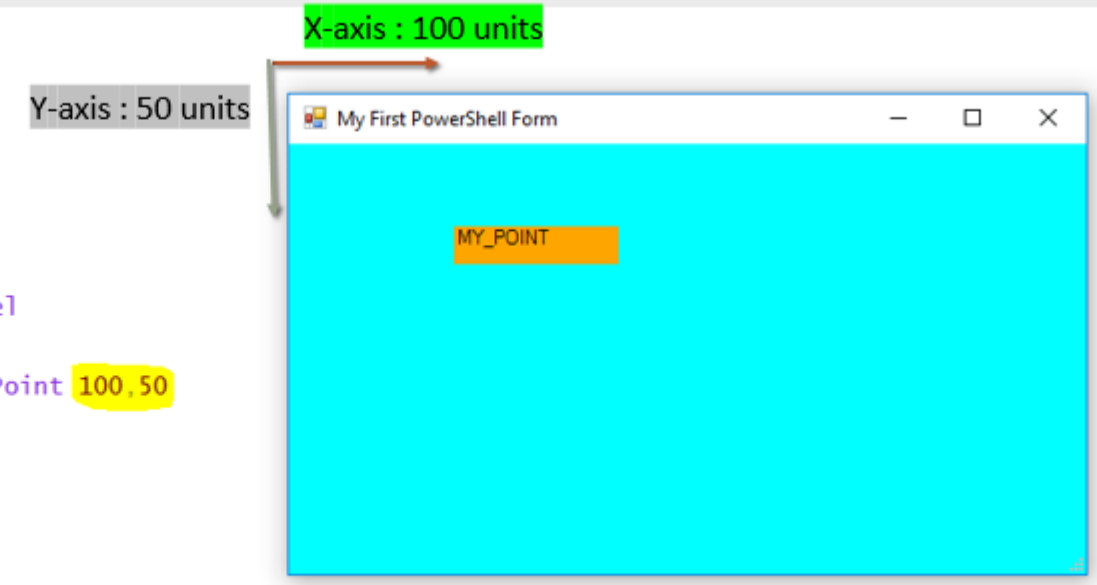


https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/how-to-position-controls-on-windows-forms

How to: Position Controls on Windows Forms

```
$label.Location = New-Object Drawing.Point x-coordinate, y-coordinate
```

```
#Example 2.2: Creating Label
$form= New-Object Windows.Forms.Form

$Form.width = 500
$Form.height = 300
$form.text="My First PowerShell Form"
$form.BackColor="cyan"

# Create the label control and
#set text, size and location
$label = New-Object Windows.Forms.Label
$label.text = "MY_POINT"

$label.Location = New-Object Drawing.Point 100,50

$label.BackColor='orange'
$form.controls.add($label)

$form.ShowDialog()
```



https://docs.microsoft.com/en-us/dotnet/framework/winforms/controls/how-to-position-controls-on-windows-forms

Size of form elements

```
#Example 2.2: Creating Label
$form= New-Object Windows.Forms.Form

$Form.width = 500
$Form.height = 300
$form.text="My First PowerShell Form"
$form.BackColor="cyan"

# Create the label control and
#set text, size and location
$label = New-Object Windows.Forms.Label
$label.text = "MY_POINT"

$label.Width=200
$label.height=300

$label.Location = New-Object Drawing.Point 1

$label.BackColor='orange'
$form.controls.add($label)

$form.ShowDialog()
```

How to Add a PowerShell GUI Event Handler- 3 ways

1.) using embedded code

```
$Button.Add_Click( {
        [System.Windows.Forms.MessageBox]::Show("Hello World." , "My Dialog Box")
       } )
```

2.) using a variable

```
 $Button_Click = {
        [System.Windows.Forms.MessageBox]::Show("Hello World." , "My Dialog Box")
}
$Button.Add_Click($Button_Click)
```

3.) using a function

```
Function Button_Click() {
        [System.Windows.Forms.MessageBox]::Show("Hello World." , "My Dialog Box")
}
$Button.Add_Click({Button_Click})
```

https://social.technet.microsoft.com/wiki/contents/articles/25911.how-to-add-a-powershell-gui-event-handler-part-1.aspx

Thank You