**Course Overview:**

1. Why Windows PowerShell?
2. History
3. PowerShell or CMD
4. .NET Integration
5. Getting and Installing PowerShell
6. Configuring the console
7. PowerShell Profiles
8. PowerShell Commands
9. Pipeline
10. Creating a PowerShell Script
11. Executing a PowerShell Script
12. Variable and Constants
13. Comparison Operators
14. If Statements and Looping
15. Regular Expressions
16. Escape Character
17. Using Parameters with Script
18. PowerShell Providers
19. Working with files and folders
20. Reading and writing data
21. HTML/CSV/XML
22. Working with Processes
23. PowerShell and WMI
24. PowerShell 5.0
25. **Why Windows PowerShell?**

* Think way back what has happened in the world of Windows Computer administration. First, we had few admins, and few computers for administration purpose. As time pass by, we maintained the administration staff and added lot of computers. But Budget, cost, computations overhead came in. As the time progressed, we came to know that there are very few admins and way many computers. The process of administering, managing, and maintaining our Network got lot more complex.

The Evolution of Windows Administration:

1. The windows OS conquered the corporate and home desktop first, then the servers.
2. Administration was GUI based, that is it was Easy on single Machine
3. Overtime, environment grew larger as the IT department Shrunk in size.
4. Graphical Tools are difficult to automate
5. On the contrary, Command Line tools are easy to automate. For example: DOS, sequenced commands in a batch file. Relatively easy to perform multiple actions in a script. But the available actions were extremely limited.

***So, what exactly is PowerShell?***

PowerShell provides command line tools for windows administrator, similar to the tools used by UNIX/LINUX admins. It also adds functionalities (based on windows operating system, .Net framework)

PowerShell provides the following:

Cmdlets for performing common system administration tasks to manage Registry, Services, Processes, Event Logs, Windows Management Instrumentation (WMI). It is a task-based scripting language and support for existing scripts and command line tools. It has consistent design, common syntax, naming convention and powerful feature of pipelining. We also get simplified command-based capability of navigating the OS. That is the registry and other data stores can be accessed by using the same techniques that are used to navigate the file system. Powerful Object Manipulation capabilities, Objects can be directly manipulated or sent to other tools or databases. It has Extensible Interface that is Independent software vendors and enterprise developers can build the custom tools and utilities to administer their Software.

*PowerShell or CMD:*

First thought everyone comes across …hey isn’t it just a CMD with blue background. No there is much more to it.

Although they do have similarities:

1. Both provide command line environment
2. Both allow creation of Scripts (for automating tasks)
3. Old CMD familiar commands work perfectly well in PowerShell. Example: dir, cls…etc

***So, what is the difference between PowerShell and Cmd.exe?***

CMD.exe provided a command line interface for a very limited set of functionality. It is a text-based scripting environment that makes it difficult to deal with the command results. It does not provide access to many aspects of the OS and applications. On the contrary, PowerShell is Object based scripting language. Since, everything is object, dealing with command result is very easy and efficient. It provides access to .NET framework which is the building block of windows functionalities. It provides the programming power and accessibility of the object-oriented programming language in a scripting environment.

.NET Framework Integration:

PowerShell is closely integrated with the .NET Framework. Everything in PowerShell is an object. That leaves us hanging with two questions namely What is .Net framework? And What exactly is an object?

What is .NET Framework?

It is foundation for Microsoft Windows application programming. It provides consistent object-oriented programming environment. It is a collection of classes providing easy access to all aspects of windows environment.

What is an Object?

An object is a software representation of a real-world entity it can be a file, folder, Parking lot etc. Objects are based on classes. A class is the blueprint describes the properties and characteristic of the object.

Let’s think about a file on a computer

The file has properties like Name, Size, Location, Permission etc. There are actions we need to perform on the file. Example: Reading data from the file, writing data to the file, Copy the file or Moving the content to the file.

A class on the other hand is the blueprint for representing this file on the hard disk as a software object. When we want to work with a file, a file class is used as a blueprint to create a file object in a memory. Let’s say we name that object myFileObj. Getting the file size now is very easy using the Size property $a = myFileObj.Size variable a now contains size of the file.

PowerShell utilizes the .NET framework to represent everything as an object in the scripting environment. This provides us with many advantages

The .NET framework can automate many tasks for us in the background. Example: Parsing/ Converting text etc when pipelining. It also standardizes many properties like length, size etc. It also exposes hierarchical objects very efficiently example: directory structure.

*Backward Compatibility:* May face some issue but can be overcome by referring to the documentation.

Getting PowerShell:

For Windows OS, a version of PowerShell is already installed. It is a part of Windows Management Framework toolset. Never Install a higher version of PowerShell on production machine unless final release occurs. The version of PowerShell you have depends on the OS.

Administrative privileges:

Default 🡪 PowerShell runs under normal user access privileges. Some functions will require Administrative privileges. There are couple of ways to run PowerShell with admin privileges. Type in PowerShell 🡪 Start-process PowerShell -verb runas. Or Right click on PowerShell icon and choose to run as administrator

Post Installation:

We need to perform three tasks they are as follows:

1. Install/update PowerShell help files
2. Enable PowerShell script execution
3. Enable PowerShell remoting

Install/update help files:

Learn and master PowerShell using PowerShell help functionality. Execute Update-Help in the PowerShell console. You should execute this command time to time to ensure you have the latest help file information.

Enable PowerShell script execution:

As a security precaution, the ability to execute PowerShell script is disabled by default. The Execution policy must be set. Execute Set-ExecutionPolicy RemoteSigned in the PowerShell console.

Enable PowerShell remoting:

By default, the PowerShell is configured to run remote commands on other windows computers. However, the computer will not allow remote commands to be executed on them. On each computer that you want remote commands to be executed on, run the following PowerShell command Enable -PSRemoting.

Configuring the console:

PowerShell console can be customized to suit one’s preference. Open PowerShell top left corner you will see icon, right click on that icon go to properties 🡪 play with it.

PowerShell IDE:

PowerShell console is a basic, command-line shell environment. There is a PowerShell Integrated Scripting Environment.

PowerGUI pro from Quest Software (free)

PowerShellPlus from Idera (free)

But one IDE comes with Windows OS

PowerShell Profiles:

The purpose and function of PowerShell profile is simple. It is a text file that is automatically loaded every time PowerShell is started. It is similar to Windows User profile. PowerShell profiles are used to persist certain customizations: Aliases, Functions, Variables.

Remember you don’t have a profile unless you create one. Your PowerShell execution profile must permit you to load configuration file. To create a PowerShell profile, create a text file with specified name in the specified location based on the type of profile you want to use.

There are 4 types of PowerShell profiles:

1. All Users All Host
2. Applies to all users and all shell.
3. %windir%\System32\WindowsPowerShell\v1.0\profile.ps1
4. All Users Current Host
5. Applies to all users, but only Microsoft.PowerShell shell
6. %windir%\System32\WindowsPowerShell\v1.0\Microsoft.PowerShell\_profile.ps1
7. Current User All Host
8. Applies only to current user, but all shell
9. %UserProfile%\MyDocuments\WindowsPowerShell\profile.ps1
10. Current User Current Host
11. Applies only to current user and Microsoft.PowerShell shell
12. %UserProfile%\MyDocuments\WindowsPowerShell\ Microsoft.PowerShell\_profile.ps1

To create a PowerShell profile:

Create a text file with specified name: profile.ps1 or Microsoft.PowerShell\_profile.ps1 in the specified location

Creating a PowerShell profile:

$Profile built in variable displays the profile location. But path and profile do not exist until we create them

Test Path $Profile return false if no profile created

-New -Item -path $profile -type file -force

A customized profile file can be remotely deployed to other computers to ensure a consistent environment and functionality across machines.