Jenna de Beer

ST10092652

NEPR7312

Portfolio of Evidence: Part One

*Activity One*

## Introduction

As information technology is constantly evolving, Windows PowerShell has emerged as an extremely powerful tool that has taken the IT working world by storm. Windows PowerShell is an extremely powerful command-line shell and scripting language produced by Microsoft that can be run on Windows, Linux, and macOS. It is designed to meet the demands of tech professionals and network administrators for managing intricate systems with the highest standards of security and efficiency.

PowerShell is a robust tool with many useful features for system management. It offers the ability to build scripts, automate complex processes, and streamline routine tasks.

PowerShell is dedicated to five fundamental design goals that contribute to its effectiveness as a tool for intricate technological system challenges. In this report, these fundamental design goals will be explored with practical examples to illustrate PowerShell’s capabilities.

As these fundamental design goals are explored, I will discuss how PowerShell simplifies discoverability, ensures consistency in commands, provides an interactive and scripting environment that can be customised to a user’s needs, uses object-oriented data management, and facilitates the transition from manual commands to more sophisticated scripting. (Microsoft, 2023)

## Description of Windows PowerShell

While traditional shells manipulate text input and output, PowerShell accepts and returns text as structured objects. Windows PowerShell is different from other shells and administrative automation tools as it not only accepts and returns text, but also accepts and returns .NET objects that allow it to provide unique features, like an in-console help system, support for aliases, and command prediction.

The in-console help system is enabled by PowerShell’s integration with .NET objects. The help system can be requested via a cmdlet or module and provides information about the associated .NET objects as well as their properties and methods. This allows users to develop a deeper understanding of the objects they work with.

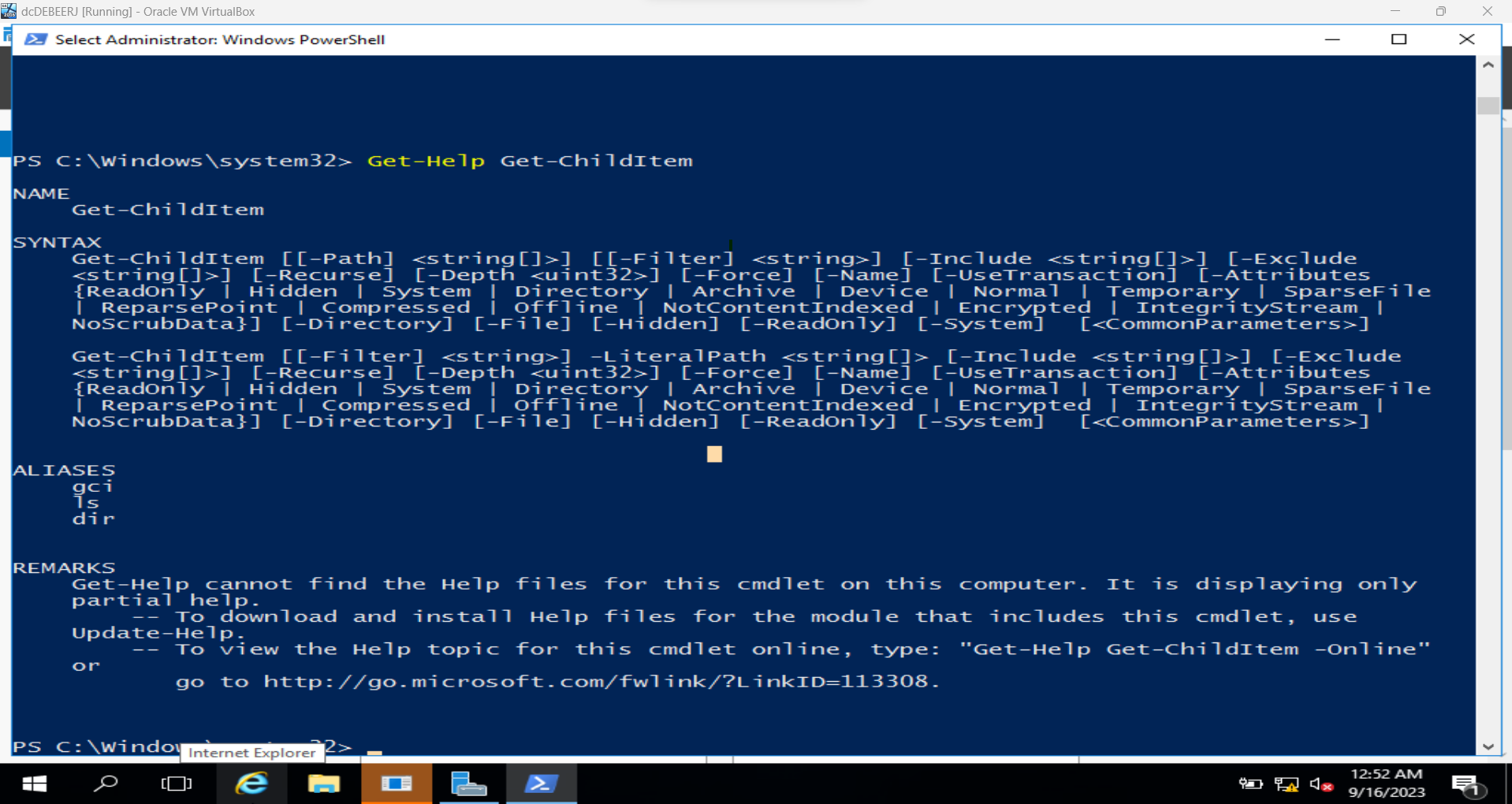


Figure 20 - Get-Help example in which the user gets help for the Get-ChildItem cmdlet

Aliases are also enabled by PowerShell’s integration with .NET. Traditional shells have often relied on textual shortcuts, but PowerShell can use aliases to represent full .NET object names. This allows users to use objects to their full potential with conveniently shorter names. For example, the ‘dir’ alias maps to the ‘Get-ChildItem’ cmdlet which returns the detailed .NET object that represents the directory information.

A screenshot of a computer

Description automatically generated

Figure 21 - Use of Get-ChildItem alias

Command prediction is another feature provided by PowerShell’s integration with .NET. As with many applications, PowerShell can predict what a user will type whether it is text or a cmdlet. By suggesting predictions, the users can avoid errors and accelerate the command-writing process.

With the ability to accept and return .NET objects, PowerShell bridges the gap between textual commands and objects that offer a more efficient environment for managing and automating systems. (BasuMallick, 2022)

## Importance of Windows PowerShell under the following design goals headings:

### Discoverability

Discoverability in PowerShell refers to the ease with which users can find and explore the commands and functionalities. This design goal allows users to find the right tools quickly and easily for their tasks. Some examples of discoverability tools in PowerShell are intuitive command names and tab completion. (Stephen J. Bigelow, 2023)

PowerShell uses intuitive command naming rules for cmdlets. For example, a cmdlet that retrieves information about running processes, and ideal name would be ‘Get-Process’ because it follows the ‘verb-noun’ naming convention. PowerShell also uses a built-in tab completion feature that suggests commands, parameters, and names as users type which reduces errors and speeds of task completion times. For example, if a user were to begin creating a new directory, PowerShell would suggest the completed line of code which the user could access via the tab key rather than typing the line manually. (Microsoft, 2022)

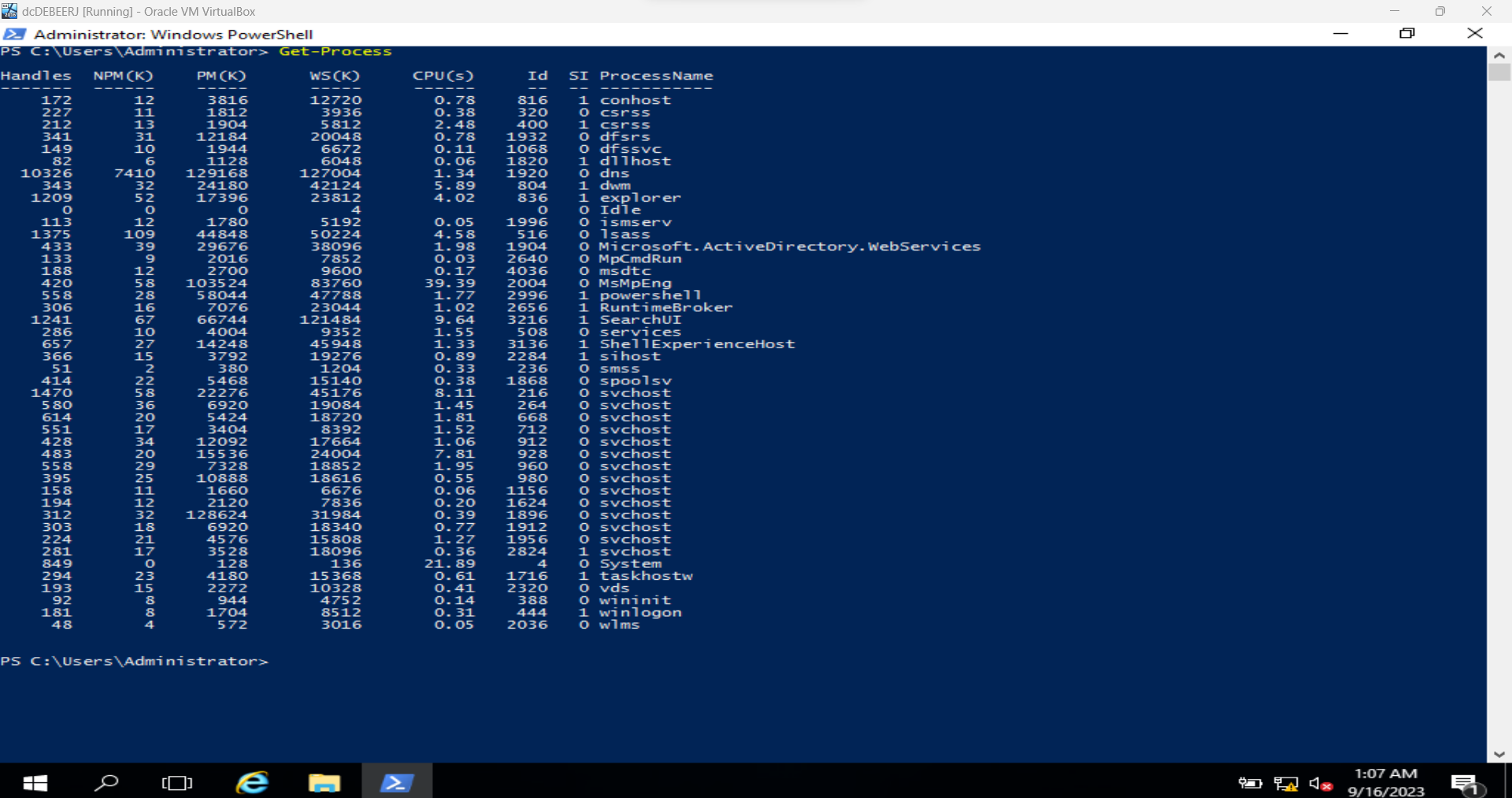


Figure 22 - Example of 'verb-noun' naming convention

### Consistency

PowerShell provides the user with consistency with predictable commands that follow syntax conventions as mentioned under Discoverability. This means that command names describe the action that they perform. For example, to list files in a directory, one can use ‘Get-ChildItem’. The output will be presented in an object-oriented format so that it is easier for the user to work with. (Microsoft, 2021)

A computer screen shot of a blue screen

Description automatically generated

Figure 23 - Use of Get-ChildItem cmdlet to view subfolders in a folder called FolderExample

### Interactive and scripting environment

PowerShell uses the Windows PowerShell Integrated Scripting Environment (ISE) as a host application for Windows PowerShell. The ISE provides useful features for the development of one-time tasks or complex automation scripts for repetitive tasks. Its scripting capabilities are powerful and allow users to create reusable scripts for automating complex tasks in work environments. The interactive shell lets users launch PowerShell and execute commands to receive immediate feedback for testing and development. (Microsoft, 2021)

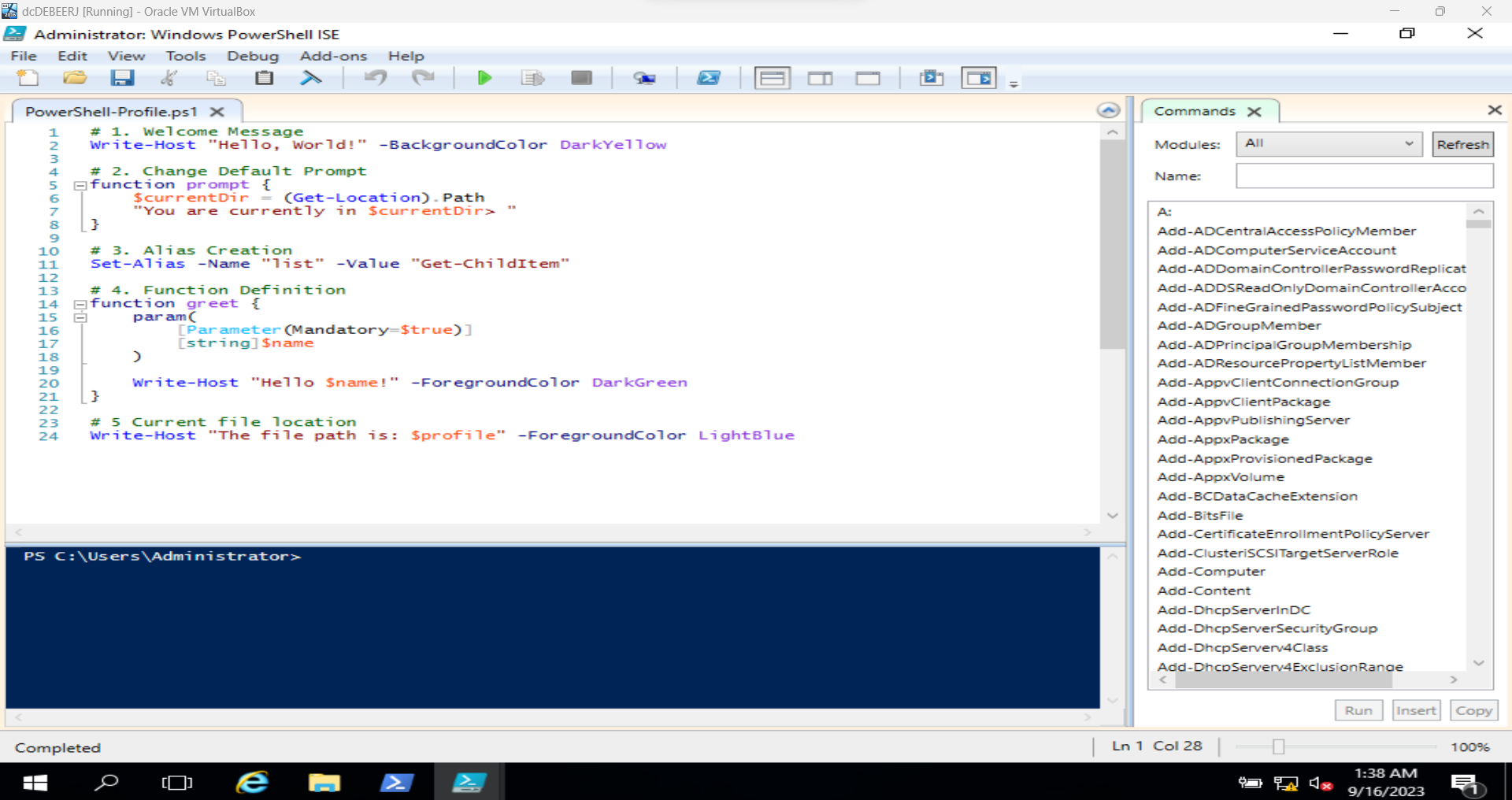


Figure 24 - ISE with script

### Object orientation

As opposed to traditional command-line interfaces, PowerShell uses object-oriented structures so that data can be more flexible. This design goal is achieved by treating output as objects and by the ability to explore object methods and properties.

Many PowerShell cmdlets produce output as objects so that users can access the data and manipulate it more easily. For example, a ‘Get-Service’ cmdlet returns a list of service objects that can be filtered, sorted, or edited. Users can also explore the object properties and methods which allows for a more effective approach in terms of their ability to work with the data. (Ben Heater, 2021)

A screenshot of a computer screen

Description automatically generated

Figure 25 - Get-Service cmdlet

### Easy transition to scripting

PowerShell aims to make the transition from one-off commands to full-fledged scripts straightforward for users. This design goal encourages users to tailor the development of their solutions to their own specific needs. Features that support this design goal are cmdlet discovery and script reusability. Users who discover new cmdlets can seamlessly incorporate them into their scripts and can save and reuse their scripts for future tasks. (Memrise, n.d.)

A screenshot of a computer

Description automatically generated

Figure 26 - Transforming a command into a script

## Conclusion

PowerShell is a versatile tool that allows administrators to do their jobs more efficiently which ultimately enhances their abilities to manage systems and networks.

By emphasising discoverability in terms of consistency in syntax and behaviours across cmdlets, users can easily find and understand commands that can be integrated into their scripts, whether for one-time tasks or for complex automation scripts for repetitive tasks.

The complex tasks seen in traditional command-line interfaces are made to be more understandable and manageable by representing data as structured objects as with an object-oriented approach. This encourages users to transition from manual commands to more powerful scripts that can be reused for other tasks rather than just used for one problem.

To conclude, Windows PowerShell is a game-changer for tech professionals and administrators. It uses the design goals to enhance its efficiency which makes it an essential instrument for the administrator fraternity.

# References

BasuMallick, C. (2022, August 26). *What Is PowerShell? Meaning, Working, Uses, and Advantages*. Retrieved from Spiceworks: https://www.spiceworks.com/tech/devops/articles/what-is-powershell/#:~:text=PowerShell%20can%20be%20used%20as,related%20to%20the%20operating%20system.

Ben Heater. (2021). *Intro to Object-Oriented Programming with PowerShell*. Retrieved from Ben Heater: https://benheater.com/intro-object-oriented-programming-powershell/

Memrise. (n.d.). *Transition to Scripting*. Retrieved from Memrise: https://app.memrise.com/course/700705/learn-powershell/13/

Microsoft. (2021, October 06). *Strongly Encouraged Development Guidelines*. Retrieved from Microsoft: https://learn.microsoft.com/en-us/powershell/scripting/developer/cmdlet/strongly-encouraged-development-guidelines?view=powershell-7.3

Microsoft. (2021, October 07). *The Windows PowerShell ISE*. Retrieved from Microsoft: https://learn.microsoft.com/en-us/powershell/scripting/windows-powershell/ise/introducing-the-windows-powershell-ise?view=powershell-7.3

Microsoft. (2022, December 15). *Chapter 2 - The Help System*. Retrieved from Microsoft: https://learn.microsoft.com/en-us/powershell/scripting/learn/ps101/02-help-system?view=powershell-7.3

Microsoft. (2023, June 28). *What is PowerShell?* Retrieved from Microsoft: https://learn.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.3

Stephen J. Bigelow, J. M. (2023, March). *What is PowerShell and how to use it: The ultimate tutorial*. Retrieved from Tech Target: https://www.techtarget.com/searchwindowsserver/definition/PowerShell