

## **Lab: 2 – Analyzing Logs and Other Administrators’ Tasks**

Vijaysingh Mahendrasingh Puwar

Seidenberg School of Computer Science and Information Systems

CYB631 – Automating Information Security with Python and Shell Scripting

Professor Alex Tsekansky

September 12, 2025

## Contents

EXERCISES: [ENVIRONMENT: STARTING WITH POWERSHELL ISE].....	7
Step 1 — Launching Windows PowerShell ISE.....	7
Step 2 — Running PowerShell ISE as Administrator and Preparing the Environment.....	7
EXERCISE I: BASIC OBJECT OPERATIONS.....	8
Step 3 — Storing the Value of a Cmdlet Output in a Variable.....	8
Step 4 — Using Static Members of a Class .....	9
Step 5 — Creating an Instance of a .NET Class .....	10
Step 6 — Screenshots of Results .....	11
Exercise II: List, Array, and Hashtables .....	12
Step 7 — Creating a Simple Array with Mixed Items.....	12
Step 8 — Creating an Array of Fixed Size and Assigning Values .....	12
Step 9 — Result of Step 8 .....	13
Step 10 — Creating a Multidimensional Array .....	13
Step 11 — Value of \$arr2[1][1] .....	14
Step 12 — Sorting an Array of Strings .....	14
Step 13 — Result of Step 12 .....	15
Step 14 — Creating and Using a Hashtable.....	15
Step 15 — Paste Results .....	16
EXERCISE III: FILES AND DIRECTORIES .....	17
Step 16 —Examine what files are in the current directory .....	17

Step 18 — Comparing Two Files Using Compare-Object .....	17
Step 19 — Find Files Modified in the Past 10 Days and Sort by Length .....	18
Step 20 — Screenshot .....	18
EXERCISE IV: SET PARAMETER FOR CALCULATING DISK USAGE .....	19
21 — Open and Review Get-DiskUsage.ps1 .....	19
22 — Preparing the Script for Execution .....	19
Step 23 — Creating Test Directories and Files .....	20
Step 24 — Running the Script With and Without the Parameter .....	20
Step 25 — Discussing the Differences in the Results .....	21
Step 26 — Writing the showtoday.ps1 Script .....	21
Step 27-28 — Screenshot of showtoday.ps1- Running showtoday.ps1 in Both Modes .....	22
EXERCISE V: PACKAGE COMMANDS IN A MODULE .....	23
Step 29 — Understanding PowerShell Modules .....	23
Step 30 — Wrapping the Script Inside a Function .....	23
Step 31 — Saving the File as a Module .....	23
Step 32 — Checking the Module Path .....	24
33 — Identifying the User Module Directory .....	24
Step 34 — Creating a Subdirectory for the Module .....	25
Step 35 — Moving the Module File .....	25
Step 36 — Importing the Show-Today Module .....	26
Step 37 — Running the Show-Today Cmdlet .....	26

Step 38 — Verifying the Module Execution .....	26
Exercise VI: Event Logs .....	27
Step 39 — Listing Event Logs .....	27
Step 40 — Determining Application and Service Logs.....	28
Step 41 — Obtaining Recent Security Log Entries .....	28
Step 42 — Filtering Recent System Events .....	29
Step 43 — Listing Today’s System Events .....	30
Step 44 — Importance of Analyzing Logs for Host Security .....	30
Exercise VII: System Service.....	31
Step 45 — Listing Running Services .....	31
Step 46 — Sorting Services by Dependency Count.....	32
Step 46 — Sorting Services by Dependency Count.....	32
Step 48 — Importance of Knowing Running Services for Host Security .....	32
Exercise VIII: Develop your system admin script .....	34
Step 49 — Developing the sys_admin.ps1 Script.....	34
Step 50 — sys_admin.ps1 Script File .....	34
Step 51 — Running sys_admin.ps1 .....	35
Lab and Class Reflection .....	37
1. What did I like about this lab? .....	37
2. What challenges did I encounter? .....	37
3. Suggestions for improving the class: .....	37

4. Turning off virtual machines:.....	37
References .....	38

## Summary

This lab introduced me to more advanced PowerShell features by building upon the basics from Lab 1. I began with object operations, arrays, and hashtables, which reinforced the idea that PowerShell treats all data as objects with properties and methods. I then explored file and directory management, script parameterization, and the process of converting a simple script into a reusable module.

In the middle portion of the lab, I analyzed event logs using both `Get-EventLog` and `Get-WinEvent`, which demonstrated the importance of log data for security monitoring and troubleshooting. I also worked with services, learning how to list, filter, and sort them by dependency count, which highlighted how service management can directly impact system stability and security.

The final task was the development of a complete administrative script, `sys_admin.ps1`. This script automatically generates a system report including timestamp, machine and user details, and event log summaries. When the `-ShowService` parameter is used, it also lists the top five running services sorted by dependent count. Creating this script helped me understand how individual PowerShell concepts can be combined into a practical automation tool for system administration and host security.

Overall, this lab was valuable because it connected foundational scripting skills with real-world administrative and security tasks. It demonstrated how PowerShell can be leveraged not only to automate routine system checks but also to support cybersecurity monitoring and incident response.

## EXERCISES: [ENVIRONMENT: STARTING WITH POWERSHELL ISE]

### Step 1 — Launching Windows PowerShell ISE

For this lab, we will be working within a **Windows 10 environment**, since many PowerShell features and cmdlets are designed for Windows systems. To begin, launch the **Windows PowerShell Integrated Scripting Environment (ISE)**, which provides both a command-line shell and a script editor in one interface.

- To open it, click on the **Start Menu**, then choose **Run** (or press Windows + R on your keyboard).
- In the Run dialog box, type: “PowerShell ISE” and press ENTER

This will start the PowerShell ISE application, which provides color-coded syntax highlighting, script pane, and console pane. Using the ISE instead of the regular PowerShell console makes it easier to edit, debug, and save scripts for later use.

### Step 2 — Running PowerShell ISE as Administrator and Preparing the Environment

Many administrative tasks in this lab require elevated privileges. Therefore, you must open PowerShell ISE with **Run as Administrator**. This ensures that you have the necessary rights to execute commands that affect system policies, directories, and services.

Once the ISE is open with administrative privileges, navigate to the directory where your lab scripts are stored. In this case, the shared lab directory is:

```
C:\Users\Administrator\Desktop\CYB631Labs\lab2
```

Use the `cd` command to set this as your working directory: `cd`  
`C:\Users\Administrator\Desktop\CYB631Labs\lab2`

Your console prompt should confirm the path, similar to: PS  
`C:\Users\Administrator\Desktop\CYB631Labs\lab2>`

## EXERCISE I: BASIC OBJECT OPERATIONS

### Step 3 — Storing the Value of a Cmdlet Output in a Variable

In PowerShell, almost everything is treated as an **object**. Objects contain both data (properties) and actions (methods). One of the most useful examples is the **Get-Date** cmdlet, which returns a `DateTime` object representing the current date and time.

Command sequence:

```
$today = Get-Date  
  
$today.Date  
  
$today.DayOfWeek  
  
$today.DayOfYear  
  
$today.ToLongDateString()
```

Explanation:

`$today = Get-Date` assigns the current date and time to a variable named `$today`.

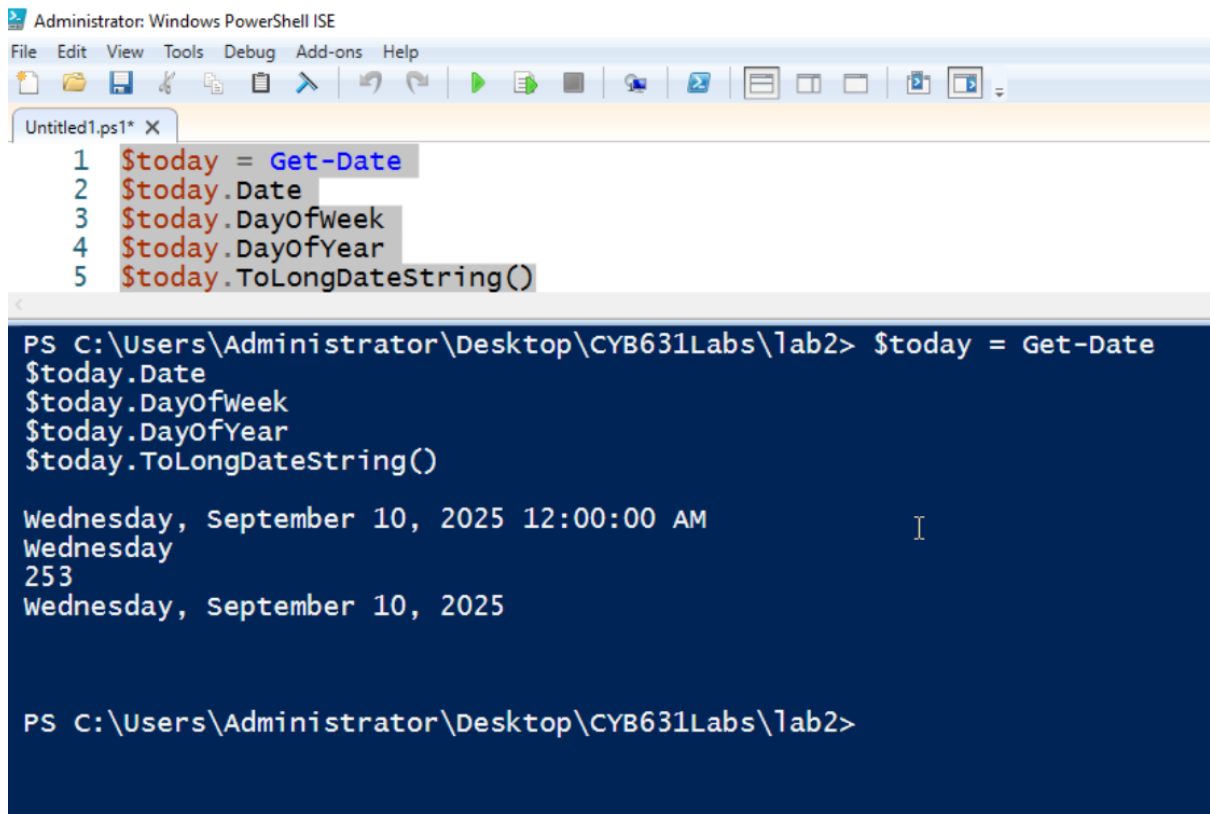
`$today.Date` shows only the date portion (year, month, and day) without the time.

`$today.DayOfWeek` outputs the current day (e.g., Tuesday).

`$today.DayOfYear` outputs the numeric day within the year (e.g., 254 = September 10, 2025).

`$today.ToLongDateString()` formats the date in a more descriptive way (e.g., *Wednesday, September 10, 2025*).





The screenshot shows the Windows PowerShell ISE interface. The top pane displays a script named 'Untitled1.ps1' with five lines of code. The bottom pane shows the output of the script, which is a dark blue console window. The output displays the results of the commands: the current date and time, the day of the week, the day of the year, and the full date string.

```

1 $today = Get-Date
2 $today.Date
3 $today.DayOfWeek
4 $today.DayOfYear
5 $today.ToLongDateString()

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> $today = Get-Date
$today.Date
$today.DayOfWeek
$today.DayOfYear
$today.ToLongDateString()

Wednesday, September 10, 2025 12:00:00 AM
Wednesday
253
Wednesday, September 10, 2025

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2>

```

Figure 1

#### Step 4 — Using Static Members of a Class

PowerShell also allows you to directly access static properties and methods of .NET classes without creating an object. The [System.DateTime] class provides several useful static members.

Command sequence:

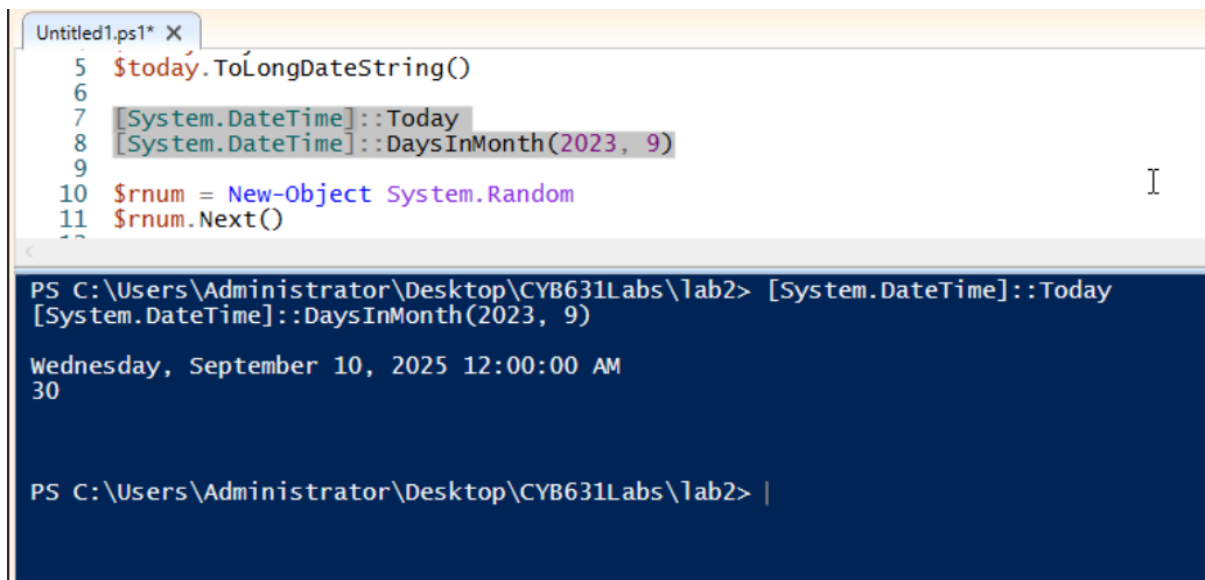
```
[System.DateTime]::Today
```

```
[System.DateTime]::DaysInMonth(2023, 9)
```

Explanation:

[System.DateTime]::Today outputs the current date (at midnight), ignoring the time of day.

[System.DateTime]::DaysInMonth(2023, 9) calculates the number of days in a given month (September 2023 = 30 days).

The image shows a PowerShell script editor window titled 'Untitled1.ps1' with a tab icon. The script contains the following lines:

```
5 $today.ToLongDateString()  
6  
7 [System.DateTime]::Today  
8 [System.DateTime]::DaysInMonth(2023, 9)  
9  
10 $rnum = New-Object System.Random  
11 $rnum.Next()
```

Below the script, a PowerShell console window is open, showing the execution of the first two lines of the script. The prompt is 'PS C:\Users\Administrator\Desktop\CYB631Labs\lab2>'. The output of the first command is 'Wednesday, September 10, 2025 12:00:00 AM'. The output of the second command is '30'. The console window has a dark blue background and white text.

Figure 2

### Step 5 — Creating an Instance of a .NET Class

Sometimes you need to create an object yourself rather than relying on cmdlets. For example, the `System.Random` class generates random numbers.

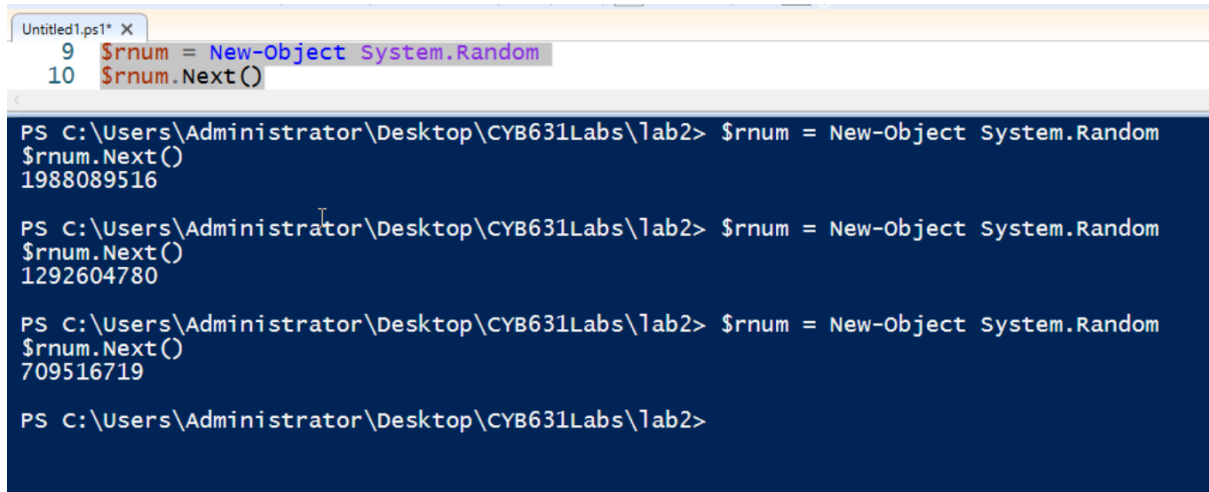
Command sequence:

```
$rnum = New-Object System.Random  
  
$rnum.Next()
```

Explanation:

`$rnum = New-Object System.Random` creates a new instance of the `Random` class.

`$rnum.Next()` generates a random integer. Each time you run it, you'll get a different number.



```
Untitled1.ps1* X
9 $rnum = New-Object System.Random
10 $rnum.Next()

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> $rnum = New-Object System.Random
$rnum.Next()
1988089516

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> $rnum = New-Object System.Random
$rnum.Next()
1292604780

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> $rnum = New-Object System.Random
$rnum.Next()
709516719

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2>
```

Figure 3

### Step 6 — Screenshots of Results

The Screenshot results are provided into Fig 2, 3 and 4 in the above following Steps.

## Exercise II: List, Array, and Hashtables

### Step 7 — Creating a Simple Array with Mixed Items

In PowerShell, an array is a collection of items stored in a single variable. Arrays can hold values of different types (numbers, strings, etc.).

Command sequence:

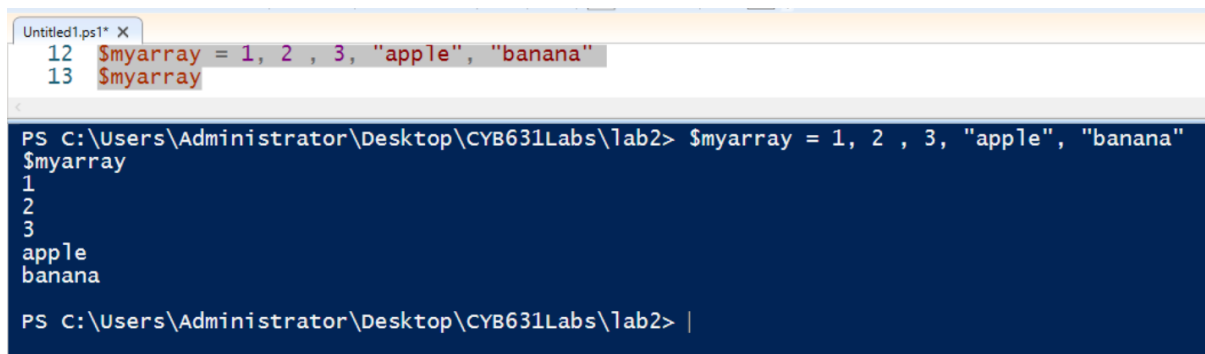
```
$myarray = 1, 2, 3, "apple", "banana"
```

```
$myarray
```

Explanation:

\$myarray is assigned a collection containing three integers and two strings.

Typing \$myarray prints the contents of the array: 1 2 3 apple banana.

A screenshot of a PowerShell console window. The title bar shows 'Untitled1.ps1 \* X'. The console has two lines of input: line 12 is '\$myarray = 1, 2, 3, "apple", "banana"' and line 13 is '\$myarray'. The output shows the array contents: '1', '2', '3', 'apple', and 'banana' on separate lines. The prompt is 'PS C:\Users\Administrator\Desktop\CYB631Labs\lab2>'.

```
12 $myarray = 1, 2, 3, "apple", "banana"
13 $myarray

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> $myarray = 1, 2, 3, "apple", "banana"
$myarray
1
2
3
apple
banana
PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> |
```

Figure 4

### Step 8 — Creating an Array of Fixed Size and Assigning Values

PowerShell allows you to explicitly create an array of a certain type and size.

Command sequence:

```
$myarray = New-Object string[] 10
```

```
$myarray[5] = "apple"
```

```
$myarray[2] = "pear"
```

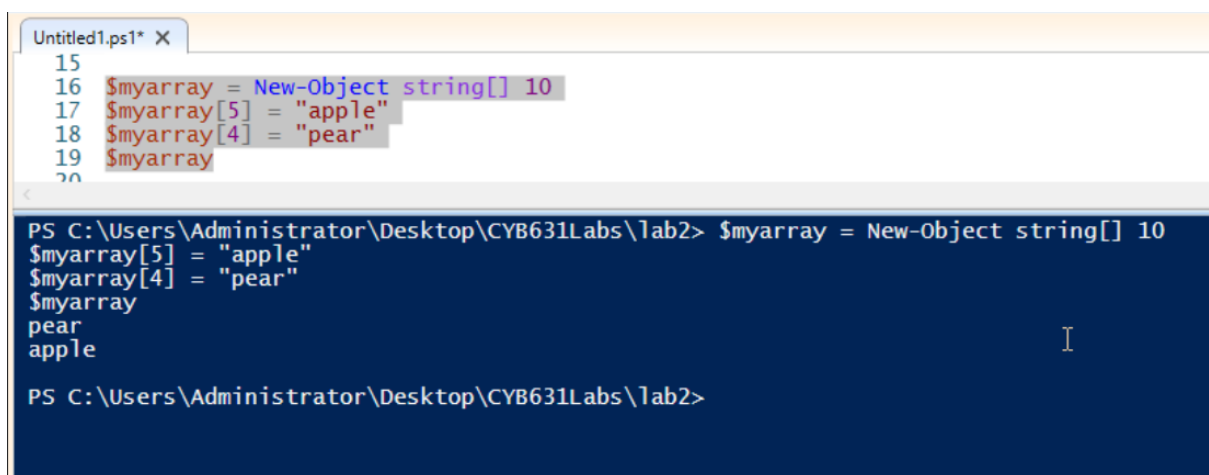
```
$myarray
```

Explanation:

- `New-Object string[] 10` creates an array with **10 slots**, all initially empty (null).
- `$myarray[5] = "apple"` assigns the value "apple" to the 6th element (arrays are 0-indexed).
- `$myarray[2] = "pear"` assigns "pear" to the 3rd element.
- Printing `$myarray` displays the 10-element array, with "pear" in position 2, "apple" in position 5, and the rest blank.

#### Step 9 — Result of Step 8

The output of Step 8 shows:



The screenshot shows a PowerShell script editor window titled 'Untitled1.ps1' with the following code:

```

15
16 $myarray = New-Object string[] 10
17 $myarray[5] = "apple"
18 $myarray[4] = "pear"
19 $myarray
20

```

Below the editor is a PowerShell console window showing the execution of the script:

```

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> $myarray = New-Object string[] 10
$myarray[5] = "apple"
$myarray[4] = "pear"
$myarray
pear
apple
PS C:\Users\Administrator\Desktop\CYB631Labs\lab2>

```

Figure 5

#### Step 10 — Creating a Multidimensional Array

Arrays can also hold nested arrays, effectively creating a multidimensional structure.

Command sequence:

- `$arr2 = @((1,2,3,4), (5,6,7,8))`
- `$arr2[0][1]`

Explanation:

- \$arr2 is assigned two sub-arrays: (1,2,3,4) and (5,6,7,8).
- \$arr2[0][1] accesses the **first sub-array** (1,2,3,4) and then the **second element** of that sub-array, which is 2.

```

19
20
21 $arr2 = @((1,2,3,4), (5,6,7,8))
22 $arr2[0][1]

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> $arr2 = @((1,2,3,4), (5,6,7,8))
$arr2[0][1]
2
PS C:\Users\Administrator\Desktop\CYB631Labs\lab2>

```

Figure 6

#### Step 11 — Value of \$arr2[1][1]

- \$arr2[1] = the second sub-array (5,6,7,8).
- \$arr2[1][1] = the second element of that sub-array, which is 6.
- **Answer:** The value is 6.

```

20
21 $arr2 = @((1,2,3,4), (5,6,7,8))
22 $arr2[1][1]
23

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> $arr2 = @((1,2,3,4), (5,6,7,8))
$arr2[1][1]
6
PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> ls

```

Figure 7

#### Step 12 — Sorting an Array of Strings

PowerShell can sort string arrays using the built-in .NET array methods.

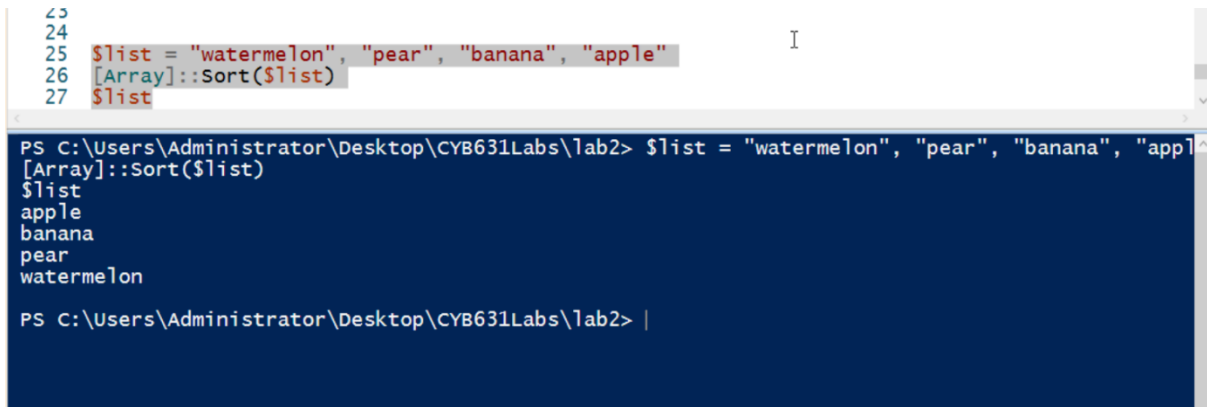
Command sequence:

- \$list = "watermelon","pear","banana","apple"
- [Array]::Sort(\$list)

- \$list

Explanation:

- \$list is initially unsorted: watermelon pear banana apple.
- [Array]::Sort(\$list) sorts the items in ascending (alphabetical) order.
- Printing \$list after sorting shows: apple banana pear watermelon.



```

23
24
25 $list = "watermelon", "pear", "banana", "apple"
26 [Array]::Sort($list)
27 $list

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> $list = "watermelon", "pear", "banana", "apple"
[Array]::Sort($list)
$list
apple
banana
pear
watermelon
PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> |

```

Figure 8

### Step 13 — Result of Step 12

**Answer:** The sorted array is:

apple  
banana  
pear  
watermelon

Also you can refer to Fig 8.

### Step 14 — Creating and Using a Hashtable

**A hashtable** is a collection of key-value pairs, useful for quick lookups.

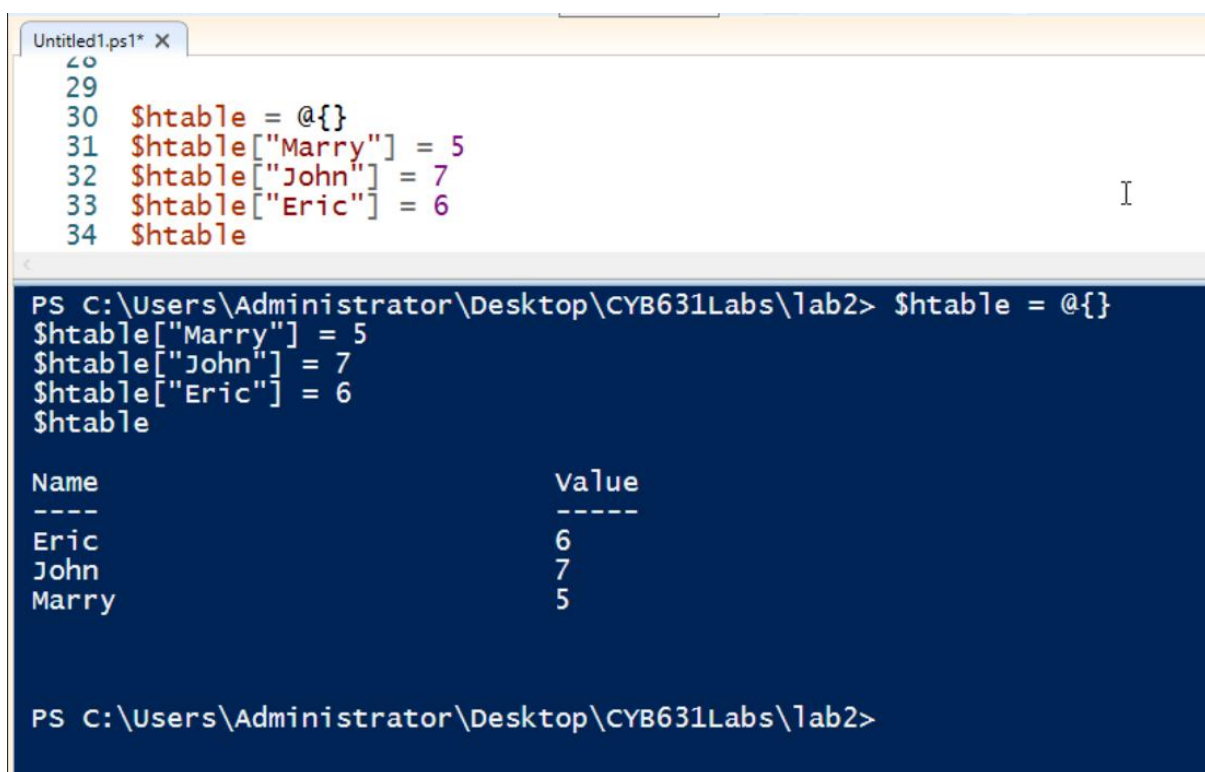
Command sequence:

- \$htable = @{ }
- \$htable["Mary"] = 5

- `$htable["John"] = 7`
- `$htable["Eric"] = 6`
- `$htable`

Explanation:

- `$htable = @{} initializes an empty hashtable.`
- Each line assigns a value to a key ("Mary"=5, "John"=7, "Eric"=6).
- Printing `$htable` displays the key-value pairs.



```
28
29
30 $htable = @{}
31 $htable["Marry"] = 5
32 $htable["John"] = 7
33 $htable["Eric"] = 6
34 $htable
```

```
PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> $htable = @{}
$htable["Marry"] = 5
$htable["John"] = 7
$htable["Eric"] = 6
$htable
```

Name	Value
Eric	6
John	7
Marry	5

```
PS C:\Users\Administrator\Desktop\CYB631Labs\lab2>
```

Figure 9

## Step 15 — Paste Results

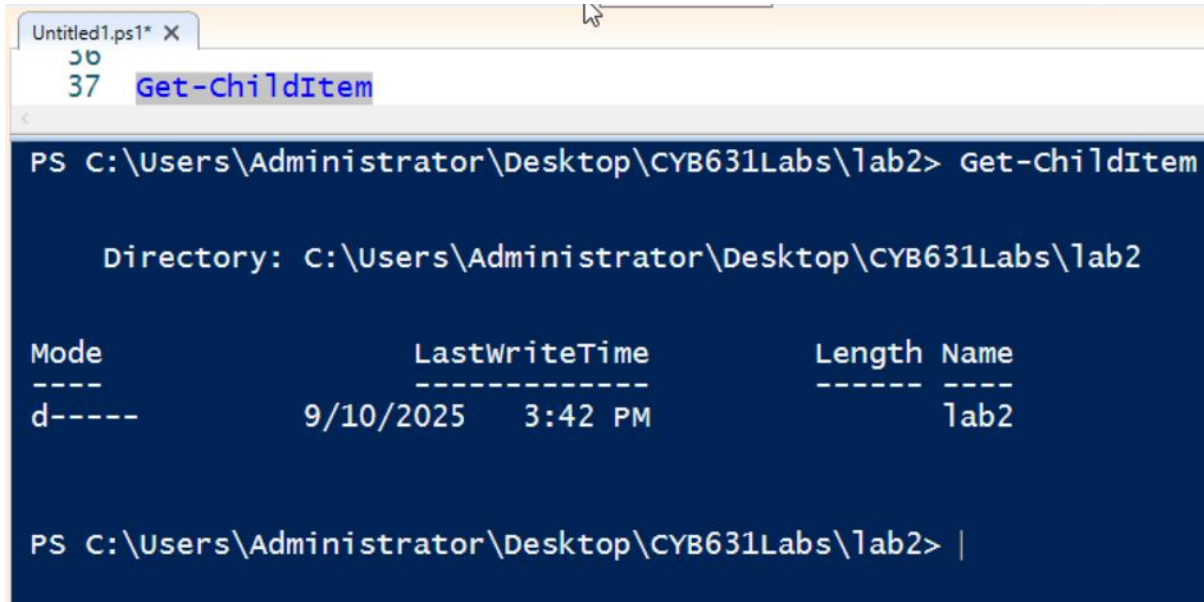
All the screenshots are shared in there respective steps.



### EXERCISE III: FILES AND DIRECTORIES

Step 16 —Examine what files are in the current directory

In PowerShell, the **Get-ChildItem** cmdlet is used to display the contents of the current directory. This includes files and subdirectories, along with key details such as file size and last modification date.



```

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> Get-ChildItem

Directory: C:\Users\Administrator\Desktop\CYB631Labs\lab2

Mode                LastWriteTime         Length Name
----                -
d-----          9/10/2025   3:42 PM           lab2

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> |

```

Figure 10

Step 18 — Comparing Two Files Using Compare-Object

The **Compare-Object** cmdlet highlights differences between two objects or files. In this step, we create two text files with slightly different contents and then compare them.

```
"Ten Apples Up on Top!" > .\file1.txt
```

```
"Ten Apples Up on Top*&^!" > .\file2.txt
```

```
$c1 = Get-Content .\file1.txt
```

```
$c2 = Get-Content .\file2.txt
```

```
Compare-Object $c1 $c2
```

### Step 19 — Find Files Modified in the Past 10 Days and Sort by Length

PowerShell can filter files by their **LastWriteTime** property to show recently modified items. Sorting by file length helps quickly identify which files are larger.

The screenshot shows a PowerShell console window with the following commands and output:

```

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2> Get-ChildItem
Get-ChildItem -Recurse -Filter "*.ps1"
"Ten Apples Up on Top!" > .\file1.txt
"Ten Apple Up on Top*&^!" > .\file2.txt
$c1 = Get-Content .\file1.txt
$c2 = Get-Content .\file2.txt
Compare-Object $c1 $c2
  
```

The output displays the directory path and a table of files:

```

Directory: C:\Users\Administrator\Desktop\CYB631Labs\lab2
  
```

Mode	LastWriteTime	Length	Name
d-----	9/10/2025 3:42 PM		lab2
-a----	9/10/2025 5:27 PM	48	.file1.txt
-a----	9/10/2025 5:27 PM	52	.file2.txt

Below the table, the comparison results are shown:

```

InputObject : Ten Apple Up on Top*&^!
SideIndicator : =>

InputObject : Ten Apples Up on Top!
SideIndicator : <=
  
```

The console ends with the prompt:

```

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2>
  
```

Figure 11

### Step 20 — Screenshot

All the screenshot in the respective steps.

## EXERCISE IV: SET PARAMETER FOR CALCULATING DISK USAGE

### 21 — Open and Review Get-DiskUsage.ps1

This includes files and subdirectories, along with key details such as file size and last modified date. This step introduces a PowerShell script that reports directory sizes and demonstrates how to **accept an input parameter** (a switch) in a script.

```
powershell_ise .\lab2\Get-DiskUsage.ps1
```

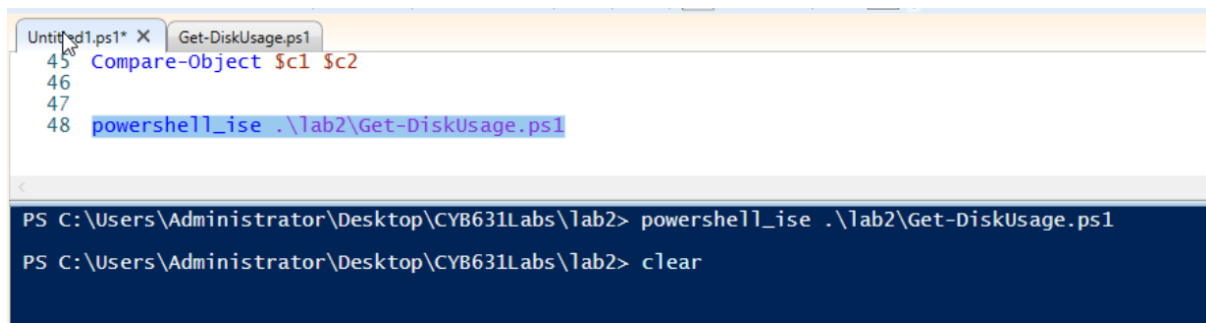


Figure 12

### 22 — Preparing the Script for Execution

Since the script came from Brightspace, it may be blocked by Windows security settings. Before running it, you must ensure scripts can execute.

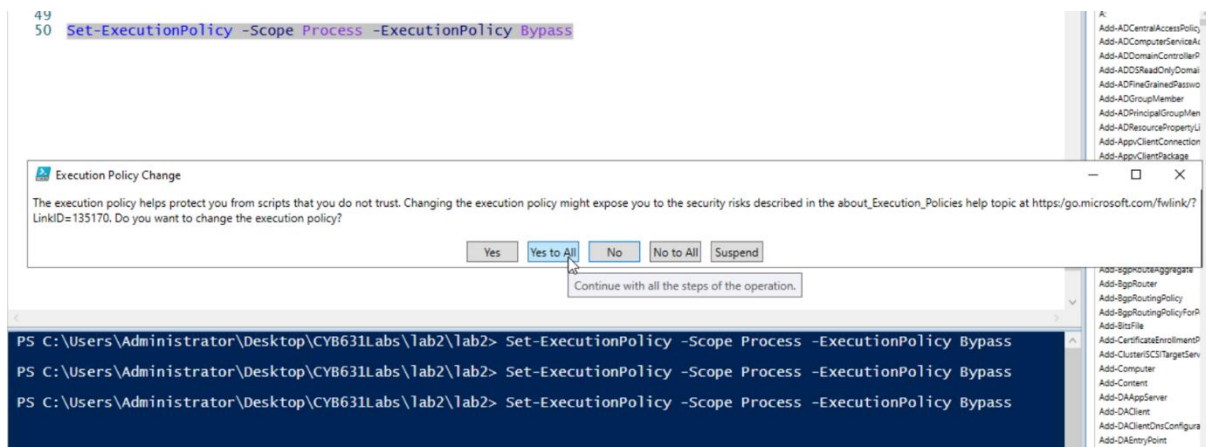
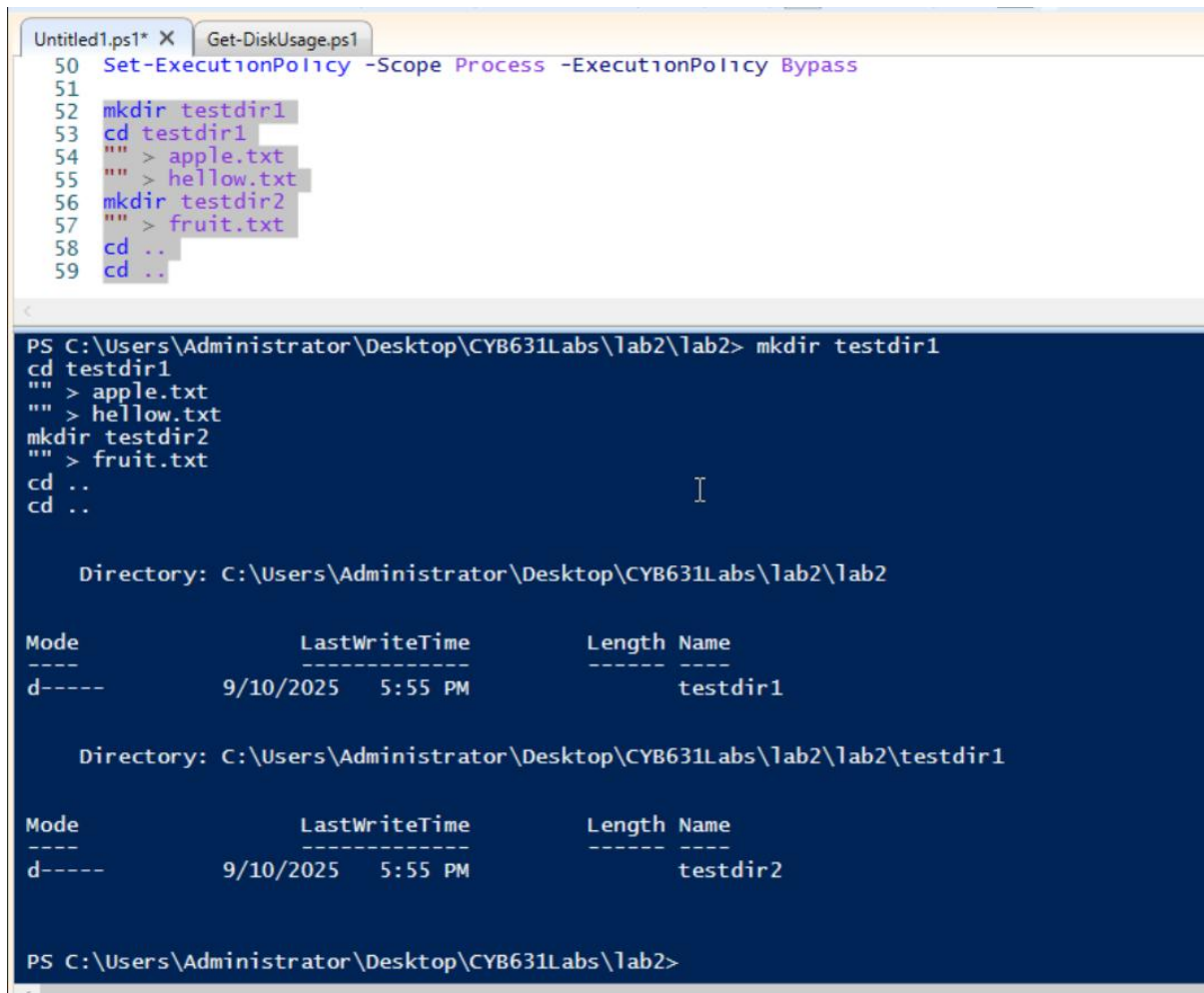


Figure 13

### Step 23 — Creating Test Directories and Files

To test the script, create a folder structure with a few text files. This setup ensures there is meaningful data for the disk-usage report.



The screenshot shows a PowerShell script named `Get-DiskUsage.ps1` in a text editor. The script contains the following commands:

```

50 Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass
51
52 mkdir testdir1
53 cd testdir1
54 "" > apple.txt
55 "" > hellow.txt
56 mkdir testdir2
57 "" > fruit.txt
58 cd ..
59 cd ..

```

Below the script, the output of running the script is shown in a PowerShell console window. The console shows the execution of the commands and the resulting directory structure:

```

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> mkdir testdir1
cd testdir1
"" > apple.txt
"" > hellow.txt
mkdir testdir2
"" > fruit.txt
cd ..
cd ..

Directory: C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2

Mode                LastWriteTime         Length Name
----                -
d-----          9/10/2025   5:55 PM             testdir1

Directory: C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2\testdir1

Mode                LastWriteTime         Length Name
----                -
d-----          9/10/2025   5:55 PM             testdir2

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2>

```

Figure 14

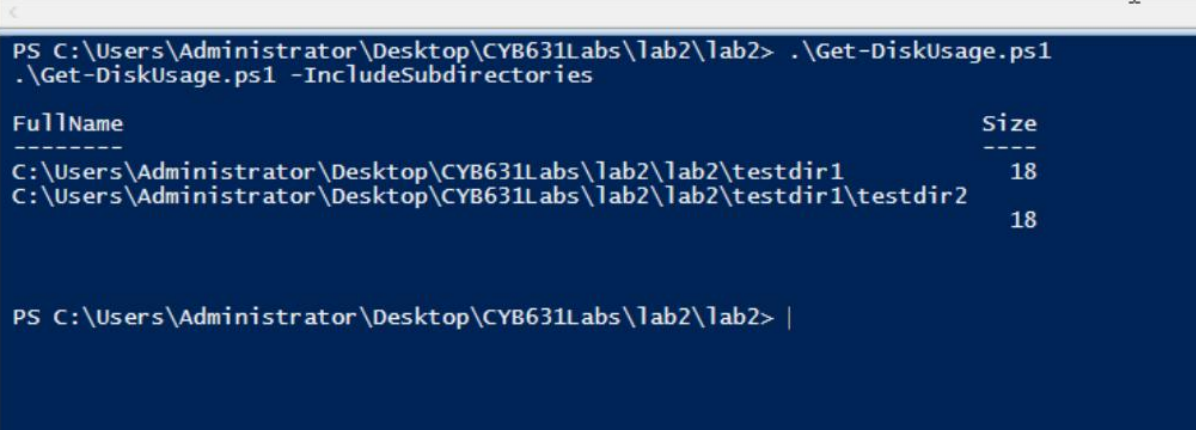
### Step 24 — Running the Script With and Without the Parameter

Now running the script twice — once normally, and once with the `-IncludeSubdirectories` switch.

```

60
61
62 .\Get-DiskUsage.ps1
63 .\Get-DiskUsage.ps1 -IncludeSubdirectories

```



```

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> .\Get-DiskUsage.ps1
.\Get-DiskUsage.ps1 -IncludeSubdirectories

FullName                                     Size
-----
C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2\testdir1      18
C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2\testdir1\testdir2 18

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> |

```

Figure 15

#### Step 25 — Discussing the Differences in the Results

- **Without -IncludeSubdirectories:** The script reports folder sizes independently. The size of testdir1 only includes its direct files.
- **With -IncludeSubdirectories:** The script rolls up sizes to include all nested subfolders. This is why testdir1 appears larger if more files were inside testdir2.

**Answer:** The difference is that the -IncludeSubdirectories parameter provides a **recursive calculation of folder size**, while the default execution only measures **non-recursive sizes**.

#### Step 26 — Writing the showtoday.ps1 Script

We now create a new PowerShell script that demonstrates how to accept and use parameters. This script will display today's date. If the -ShowWeek parameter is included, only the weekday will be shown; otherwise, full date information will be displayed.

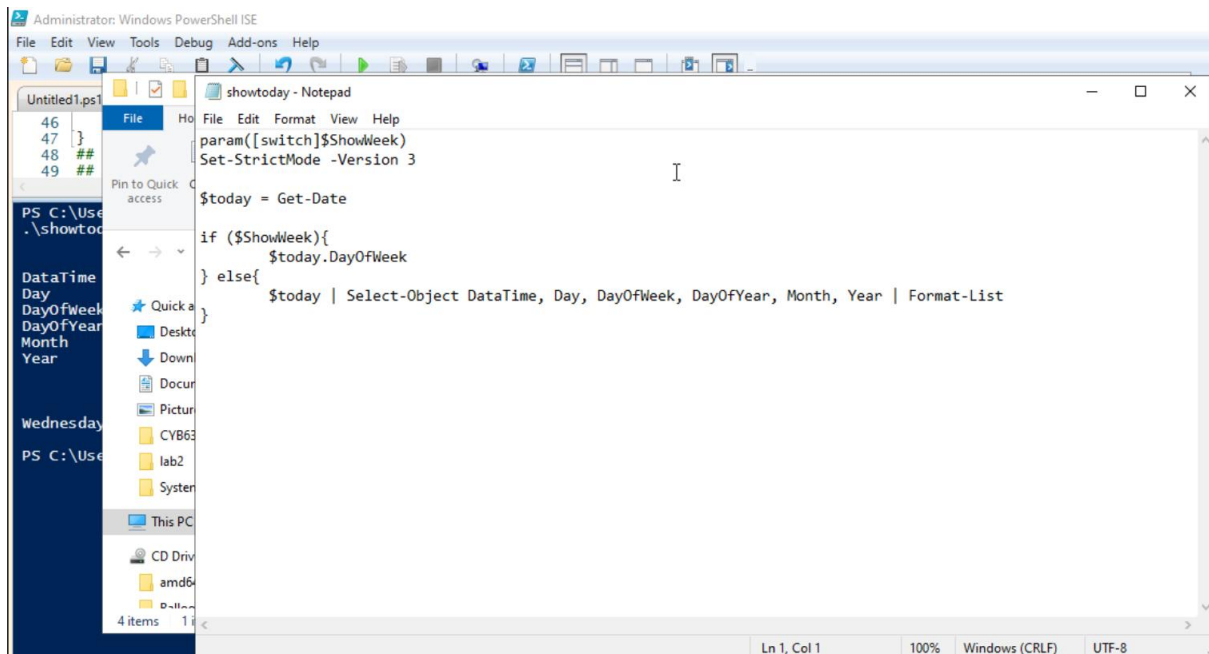


Figure 16

Step 27-28 — Screenshot of showtoday.ps1- Running showtoday.ps1 in Both Modes

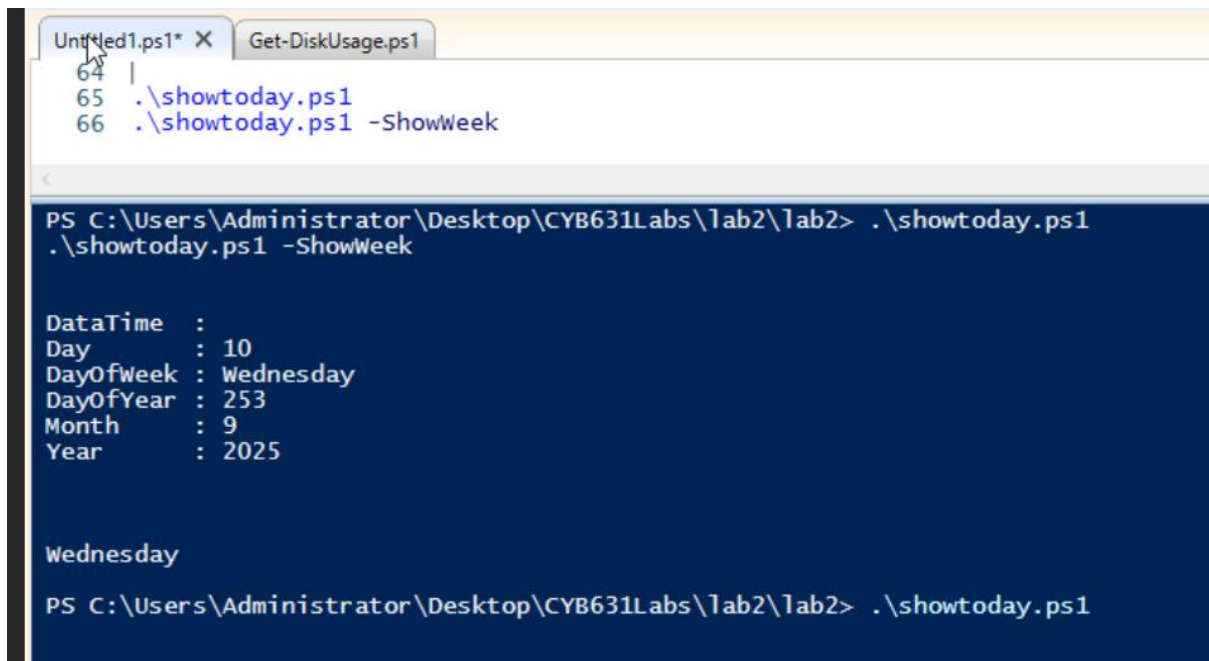


Figure 17

## EXERCISE V: PACKAGE COMMANDS IN A MODULE

### Step 29 — Understanding PowerShell Modules

PowerShell allows scripts and functions to be packaged into reusable **modules**. Once a script is turned into a module, it can be imported and used just like built-in cmdlets. This step introduces the process of converting your `showtoday.ps1` script into a reusable PowerShell module.

### Step 30 — Wrapping the Script Inside a Function

To convert a script into a module, the code must be wrapped inside a **function**. In this case, we will use the function name `Show-Today`.

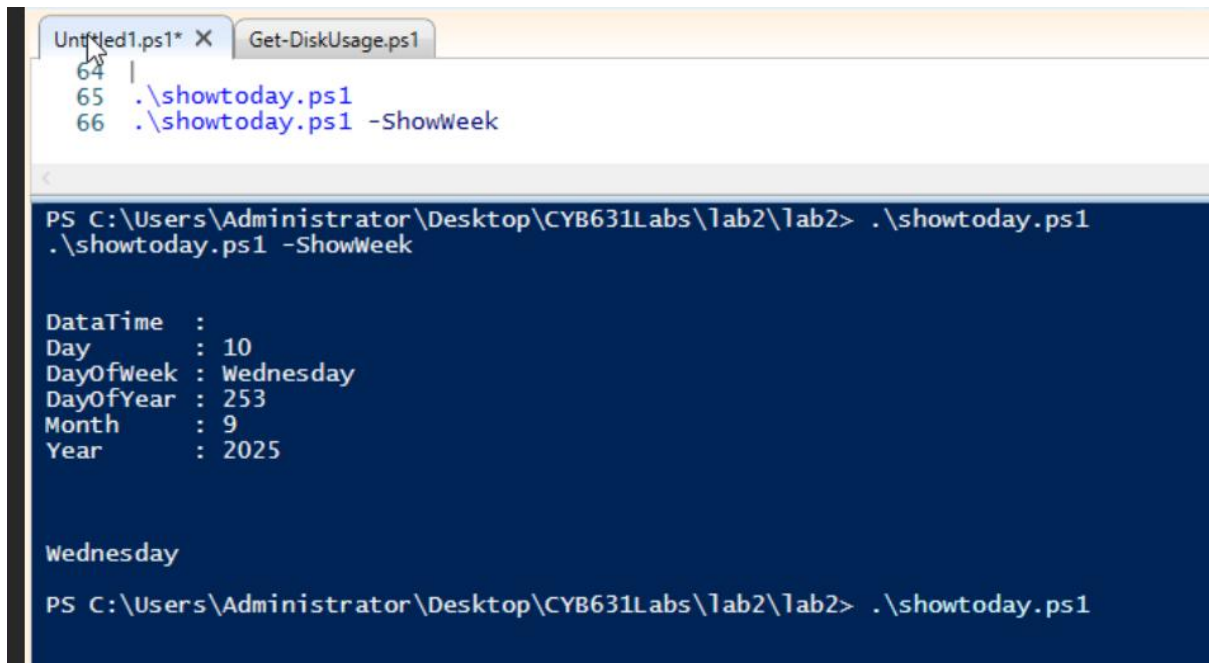
Explanation:

- Wrapping the logic in a function makes the script callable by name (`Show-Today`).
- The functionality remains the same: with the `-ShowWeek` switch, only the day of the week is shown; without it, the full date details are displayed.

### Step 31 — Saving the File as a Module

After editing the script, save it as `Show-Today.psm1`.

- The `.psm1` extension indicates that this file is a **PowerShell module file**.
- Store it temporarily in your Lab 2 working directory before moving it to the module path.



```

Untitled1.ps1* X  Get-DiskUsage.ps1
64 |
65 .\showtoday.ps1
66 .\showtoday.ps1 -ShowWeek

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> .\showtoday.ps1
.\showtoday.ps1 -ShowWeek

DateTime :
Day       : 10
DayOfWeek : Wednesday
DayOfYear : 253
Month     : 9
Year      : 2025

Wednesday

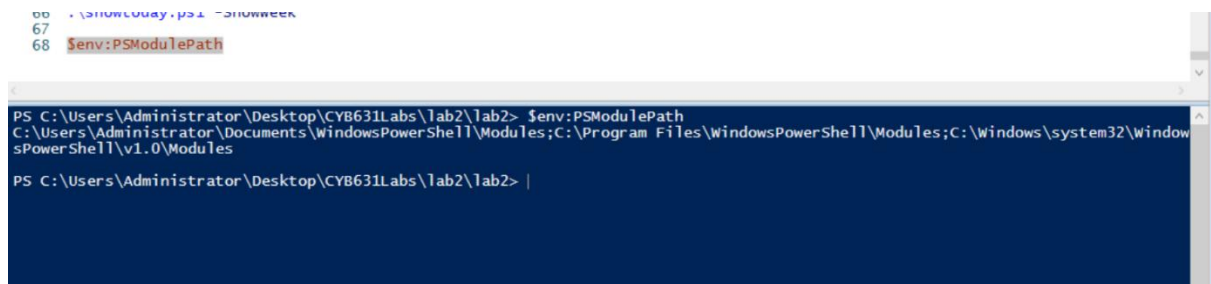
PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> .\showtoday.ps1

```

Figure 18

### Step 32 — Checking the Module Path

To determine where PowerShell expects user modules to be stored, check the environment variable `$env:PSModulePath`.



```

66 .\showtoday.ps1 -ShowWeek
67
68 $env:PSModulePath

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> $env:PSModulePath
C:\Users\Administrator\Documents\WindowsPowerShell\Modules;C:\Program Files\WindowsPowerShell\Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> |

```

Figure 19

### 33 — Identifying the User Module Directory

The `$env:PSModulePath` environment variable shows all directories where PowerShell looks for modules. Running the command reveals multiple paths separated by semicolons.



```

66 .\ShowToday.psm1 -ShowWeek
67
68 $env:PSModulePath

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> $env:PSModulePath
C:\Users\Administrator\Documents\WindowsPowerShell\Modules;C:\Program Files\WindowsPowerShell\Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules
PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> |

```

Figure 20

### Step 34 — Creating a Subdirectory for the Module

Each module must reside in its own folder named after the module. Therefore, inside the modules directory, create a subfolder called **Show-Today**.

```

70
71 mkdir "C:\Users\Administrator\Documents\WindowsPowerShell\Modules\Show-Today"

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> mkdir "C:\Users\Administrator\Documents\WindowsPowerShell\Modules\Show-Today"

Directory: C:\Users\Administrator\Documents\WindowsPowerShell\Modules

Mode                LastWriteTime         Length Name
----                -
d-----          9/10/2025   6:33 PM             Show-Today

PS C:\Users\Administrator\Desktop\CYB631Labs\lab2\lab2> |

```

Figure 21

### Step 35 — Moving the Module File

Move your Show-Today.psm1 file (created earlier) from the Lab 2 working directory into the new Show-Today module folder.

```
mv .\Show-Today.psm1
```

```
"C:\Users\Administrator\Documents\WindowsPowerShell\Modules\Show-Today"
```

### Step 36 — Importing the Show-Today Module

After moving the file to the module directory, you can import it as a module: Import-Module Show-Today

### Step 37 — Running the Show-Today Cmdlet

Once *Show-Today* is imported as a module, you can run it like any other PowerShell cmdlet:

```
Show-Today
```

```
Show-Today -ShowWeek
```

### Step 38 — Verifying the Module Execution

Paste a screenshot of the above commands and their results to show that the module ran successfully.

The screenshot shows a PowerShell console window with the following commands and output:

```

105 $psm1 = Get-Childitem $psm1
106 Test-Path $psm1
107 Get-Childitem $psm1
108
109
110 Import-Module Show-Today -Force
111 Get-Command Show-Today -Module Show-Today
112
113 Show-Today
114 Show-Today -ShowWeek
115

```

The output of the commands is as follows:

```

PS C:\> Get-Command Show-Today -Module Show-Today

CommandType      Name
-----
Function          Show-Today

PS C:\> .\Show-Today
.\Show-Today -ShowWeek

PS C:\> Show-Today
Show-Today -ShowWeek

DateTime : Friday, September 12, 2025 2:29:06 PM
Day       : 12
DayOfWeek : Friday
DayOfYear : 255
Month     : 9
Year      : 2025

Friday
PS C:\>

```

Figure 22: Finally, I made it :)

## Exercise VI: Event Logs

### Step 39 — Listing Event Logs

To determine the event logs available on the system, I ran:

```
Get-EventLog -List
```

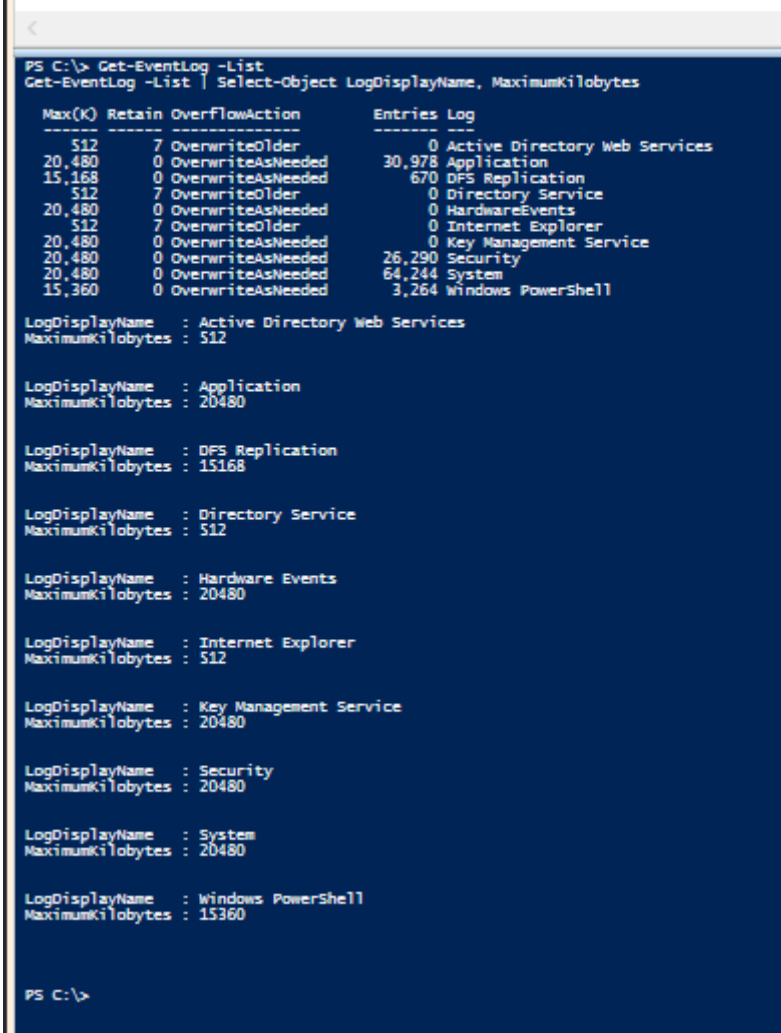
```
Get-EventLog -List | Select-Object LogDisplayName, MaximumKilobytes
```

*Observed result:* Displayed all event logs along with their maximum sizes. This helps identify which logs exist and how much storage is allocated for each.

```

118 Get-EventLog -List
119 Get-EventLog -List | Select-Object LogDisplayName, MaximumKilobytes
120

```



Max(K)	Retain	OverflowAction	Entries	Log
512	7	OverwriteOlder	0	Active Directory Web Services
20,480	0	OverwriteAsNeeded	30,978	Application
15,168	0	OverwriteAsNeeded	670	DFS Replication
512	7	OverwriteOlder	0	Directory Service
20,480	0	OverwriteAsNeeded	0	Hardware Events
512	7	OverwriteOlder	0	Internet Explorer
20,480	0	OverwriteAsNeeded	0	Key Management Service
20,480	0	OverwriteAsNeeded	26,290	Security
20,480	0	OverwriteAsNeeded	64,244	System
15,360	0	OverwriteAsNeeded	3,264	Windows PowerShell

```

LogDisplayName : Active Directory Web Services
MaximumKilobytes : 512

LogDisplayName : Application
MaximumKilobytes : 20480

LogDisplayName : DFS Replication
MaximumKilobytes : 15168

LogDisplayName : Directory Service
MaximumKilobytes : 512

LogDisplayName : Hardware Events
MaximumKilobytes : 20480

LogDisplayName : Internet Explorer
MaximumKilobytes : 512

LogDisplayName : Key Management Service
MaximumKilobytes : 20480

LogDisplayName : Security
MaximumKilobytes : 20480

LogDisplayName : System
MaximumKilobytes : 20480

LogDisplayName : Windows PowerShell
MaximumKilobytes : 15360

PS C:\>

```

Figure 23

## Step 40 — Determining Application and Service Logs

I used Get-WinEvent to enumerate application and service logs:

```
Get-WinEvent -ListLog * | Select-Object LogName, RecordCount
```

```
PS C:\> Get-WinEvent -ListLog * | Select-Object LogName, RecordCount
```

LogName	RecordCount
Windows Power Shell	3268
System	64250
Security	26290
Key Management Service	0
Internet Explorer	0
HardwareEvents	0
Directory Service	0
DFS Replication	670
Application	31013
Active Directory Web Services	0
Windows Networking Vpn Plugin Platform/OperationalVerbose	0
Windows Networking Vpn Plugin Platform/Operational	0
SMSApi	0
Setup	60
OpenSSH/Operational	0
OpenSSH/Admin	0
Network Isolation Operational	0
Microsoft-Windows-WPD-MTPClassDriver/Operational	0
Microsoft-Windows-WPD-CompositeClassDriver/Operational	0
Microsoft-Windows-WPD-ClassInstaller/Operational	0
Microsoft-Windows-Workplace Join/Admin	0
Microsoft-Windows-Workpad/Admin	0
Microsoft-Windows-WMPNSS-Service/Operational	0
Microsoft-Windows-WMI-Activity/Operational	1284
Microsoft-Windows-Wired-AutoConfig/Operational	0
Microsoft-Windows-Winsock-WS2HELP/Operational	0
Microsoft-Windows-Winsock-NameResolution/Operational	0
Microsoft-Windows-Winsock-AFD/Operational	0
Microsoft-Windows-WinRM/Operational	2351
Microsoft-Windows-WinNat/Operational	0
Microsoft-Windows-Winlogon/Operational	948
Microsoft-Windows-WinINet/Operational	0
Microsoft-Windows-WinINet-Config/ProxyConfigChanged	1
Microsoft-Windows-WinINet-Capture/Analytic	0
Microsoft-Windows-WinHttp/Operational	0
Microsoft-Windows-WinHTTP-NDF/Diagnostic	0
Microsoft-Windows-WindowsUpdateClient/Operational	1511
Microsoft-Windows-WindowsUIImmersive/Operational	0
Microsoft-Windows-WindowsSystemAssessmentTool/Operational	83

Figure 24

## Step 41 — Obtaining Recent Security Log Entries

To view the 10 most recent security log entries:

```
Get-EventLog Security -Newest 10 | Format-Table Index, Source, Message -AutoSize
```

```

131 Get-EventLog Security -Newest 10 | Format-Table Index, Source, Message -AutoSize

```

Index	Source	Message
85743	Microsoft-Windows-Security-Auditing	Special privileges assigned to new logon....
85742	Microsoft-Windows-Security-Auditing	An account was successfully logged on....
85741	Microsoft-Windows-Security-Auditing	Special privileges assigned to new logon....
85740	Microsoft-Windows-Security-Auditing	An account was successfully logged on....
85739	Microsoft-Windows-Security-Auditing	Special privileges assigned to new logon....
85738	Microsoft-Windows-Security-Auditing	An account was successfully logged on....
85737	Microsoft-Windows-Security-Auditing	Credential Manager credentials were read....
85736	Microsoft-Windows-Security-Auditing	Credential Manager credentials were read....
85735	Microsoft-Windows-Security-Auditing	Credential Manager credentials were read....
85734	Microsoft-Windows-Security-Auditing	Credential Manager credentials were read....

Figure 25

## Step 42 — Filtering Recent System Events

To search the most recent 100 system events for specific text (example: “service”):

Get-EventLog System -Newest 100 | Where-Object { \$\_.Message -match "service" }

```

132
133 Get-EventLog System -Newest 100 | Where-Object { $_.Message -match "service"}

```

Index	Time	EntryType	Source	InstanceID	Message
83081	Sep 12 14:43	Information	Service Control M...	1073748860	The Windows Modules Installer service entered the running state.
83080	Sep 12 14:38	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the stopped state.
83079	Sep 12 14:38	Information	Service Control M...	1073748864	The start type of the Background Intelligent Transfer Service service was c...
83078	Sep 12 14:36	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83077	Sep 12 14:34	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the stopped state.
83076	Sep 12 14:34	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the stopped state.
83075	Sep 12 14:33	Information	Service Control M...	1073748860	The Network Setup Service service entered the stopped state.
83074	Sep 12 14:30	Information	Service Control M...	1073748860	The Network Setup Service service entered the running state.
83073	Sep 12 14:30	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83072	Sep 12 14:30	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83071	Sep 12 14:30	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83070	Sep 12 14:30	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83069	Sep 12 14:29	Information	Service Control M...	1073748860	The Software Protection service entered the stopped state.
83068	Sep 12 14:29	Information	Service Control M...	1073748860	The Software Protection service entered the running state.
83067	Sep 12 14:28	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the running state.
83066	Sep 12 14:28	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the running state.
83065	Sep 12 14:28	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83064	Sep 12 14:28	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the running state.
83063	Sep 12 14:28	Information	Service Control M...	1073748864	The start type of the Background Intelligent Transfer Service service was c...
83062	Sep 12 14:27	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the stopped state.
83061	Sep 12 14:26	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the stopped state.
83060	Sep 12 14:21	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the stopped state.
83059	Sep 12 14:21	Information	Service Control M...	1073748864	The start type of the Background Intelligent Transfer Service service was c...
83058	Sep 12 14:21	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the running state.
83057	Sep 12 14:21	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the running state.
83056	Sep 12 14:19	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83055	Sep 12 14:19	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the stopped state.
83054	Sep 12 14:18	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the stopped state.
83053	Sep 12 14:13	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83052	Sep 12 14:13	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83051	Sep 12 14:13	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83050	Sep 12 14:13	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83049	Sep 12 14:12	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the running state.
83048	Sep 12 14:12	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the running state.
83047	Sep 12 14:11	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83046	Sep 12 14:11	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the running state.
83045	Sep 12 14:11	Information	Service Control M...	1073748864	The start type of the Background Intelligent Transfer Service service was c...
83044	Sep 12 14:08	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the stopped state.
83043	Sep 12 14:07	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the stopped state.
83042	Sep 12 14:05	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the stopped state.
83041	Sep 12 14:05	Information	Service Control M...	1073748864	The start type of the Background Intelligent Transfer Service service was c...
83040	Sep 12 14:04	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the stopped state.
83039	Sep 12 14:03	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83038	Sep 12 14:02	Information	Service Control M...	1073748860	The Windows Camera Frame Server service entered the stopped state.
83037	Sep 12 14:02	Information	Service Control M...	1073748860	The Windows Camera Frame Server Monitor service entered the stopped state.
83036	Sep 12 14:01	Information	Service Control M...	1073748860	The AppX Deployment Service (AppXSVC) service entered the running state.
83035	Sep 12 14:01	Information	Service Control M...	1073748860	The Windows Camera Frame Server service entered the running state.
83034	Sep 12 14:01	Information	Service Control M...	1073748860	The Windows Camera Frame Server Monitor service entered the running state.
83033	Sep 12 13:59	Information	Service Control M...	1073748860	The Software Protection service entered the stopped state.
83032	Sep 12 13:59	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the running state.
83031	Sep 12 13:59	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the running state.
83030	Sep 12 13:59	Information	Service Control M...	1073748860	The Software Protection service entered the running state.
83029	Sep 12 13:59	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83028	Sep 12 13:57	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83027	Sep 12 13:57	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83026	Sep 12 13:57	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83025	Sep 12 13:57	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83024	Sep 12 13:55	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83023	Sep 12 13:55	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the running state.
83022	Sep 12 13:55	Information	Service Control M...	1073748864	The start type of the Background Intelligent Transfer Service service was c...
83021	Sep 12 13:49	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the stopped state.
83020	Sep 12 13:49	Information	Service Control M...	1073748864	The start type of the Background Intelligent Transfer Service service was c...
83019	Sep 12 13:49	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the stopped state.
83018	Sep 12 13:47	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83017	Sep 12 13:47	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the stopped state.
83016	Sep 12 13:44	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the running state.
83015	Sep 12 13:44	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the running state.

Figure 26

## Step 43 — Listing Today's System Events

The following PowerShell command lists all system events generated **today**: Get-

EventLog System | Where-Object { \$\_.TimeGenerated -ge (Get-Date).Date }

```
137 Get-EventLog System | Where-Object { $_.TimeGenerated -ge (Get-Date).Date }
```

```
PS C:\> Get-EventLog System | Where-Object { $_.TimeGenerated -ge (Get-Date).Date }
```

Index	Time	EntryType	Source	InstanceID	Message
83093	Sep 12 14:49	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the stopped state.
83092	Sep 12 14:47	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the stopped state.
83091	Sep 12 14:46	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83090	Sep 12 14:46	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83089	Sep 12 14:46	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83088	Sep 12 14:46	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83087	Sep 12 14:45	Information	Service Control M...	1073748860	The Windows Modules Installer service entered the stopped state.
83086	Sep 12 14:44	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the running state.
83085	Sep 12 14:44	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the running state.
83084	Sep 12 14:44	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83083	Sep 12 14:44	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the running state.
83082	Sep 12 14:44	Information	Service Control M...	1073748864	The start type of the Background Intelligent Transfer Service service was changed from d...
83081	Sep 12 14:43	Information	Service Control M...	1073748860	The Windows Modules Installer service entered the running state.
83080	Sep 12 14:38	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the stopped state.
83079	Sep 12 14:38	Information	Service Control M...	1073748864	The start type of the Background Intelligent Transfer Service service was changed from a...
83078	Sep 12 14:36	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83077	Sep 12 14:34	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the stopped state.
83076	Sep 12 14:34	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the stopped state.
83075	Sep 12 14:33	Information	Service Control M...	1073748860	The Network Setup Service service entered the stopped state.
83074	Sep 12 14:30	Information	Service Control M...	1073748860	The Network Setup Service service entered the running state.
83073	Sep 12 14:30	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83072	Sep 12 14:30	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83071	Sep 12 14:30	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83070	Sep 12 14:30	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83069	Sep 12 14:29	Information	Service Control M...	1073748860	The Software Protection service entered the stopped state.
83068	Sep 12 14:29	Information	Service Control M...	1073748860	The Software Protection service entered the running state.
83067	Sep 12 14:28	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the running state.
83066	Sep 12 14:28	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the running state.
83065	Sep 12 14:28	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83064	Sep 12 14:28	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the running state.
83063	Sep 12 14:28	Information	Service Control M...	1073748864	The start type of the Background Intelligent Transfer Service service was changed from d...
83062	Sep 12 14:27	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the stopped state.
83061	Sep 12 14:26	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the stopped state.
83060	Sep 12 14:21	Information	Service Control M...	1073748860	The Background Intelligent Transfer Service service entered the stopped state.
83059	Sep 12 14:21	Information	Service Control M...	1073748864	The start type of the Background Intelligent Transfer Service service was changed from a...
83058	Sep 12 14:21	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the running state.
83057	Sep 12 14:21	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the running state.
83056	Sep 12 14:19	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.
83055	Sep 12 14:19	Information	Service Control M...	1073748860	The Client License Service (ClipsVC) service entered the stopped state.
83054	Sep 12 14:18	Information	Service Control M...	1073748860	The Microsoft Account Sign-in Assistant service entered the stopped state.
83053	Sep 12 14:13	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the running state.
83052	Sep 12 14:13	Information	Service Control M...	1073748860	The WMI Performance Adapter service entered the stopped state.

Figure 27

## Step 44 — Importance of Analyzing Logs for Host Security

Analyzing logs is critical for host security because:

- Logs provide visibility into **suspicious or unauthorized activity** (failed logins, privilege escalation).
- They help detect **malware infections, misconfigurations, or hardware failures**.
- Logs serve as **forensic evidence** during incident response and investigations.
- Regular log analysis improves compliance with **security frameworks and regulations** (NIST, ISO 27001, SOC 2).

**In short:** Without monitoring logs, malicious activity could go undetected, making systems vulnerable to compromise.



## Exercise VII: System Service

### Step 45 — Listing Running Services

To view only the services currently running on the system, I used: Get-Service |

Where-Object { \$\_.Status -eq "Running" }

```

128
129
130 get-service | where-object {$_.Status -eq "Running"}

```

Status	Name	DisplayName
Running	alg	Application Layer Gateway Service
Running	apphostsvc	Application Host Helper Service
Running	appinfo	Application Information
Running	salliconservice	Salliconservice
Running	alg	Safe Filtering Engine
Running	backgroundtasksinfrastructure...	Background Tasks Infrastructure Ser...
Running	casvc	Capability Access Manager Service
Running	cdhsvc_7586323	Clipboard User Service_7586323
Running	cdhsvc	Connected Devices Platform Service
Running	cdhsvc_7586323	Connected Devices Platform User Ser...
Running	combaseapi	Combaseapi
Running	cryptsvc	Cryptographic Services
Running	ocmlaunch	OCML Server Process Launcher
Running	ofs	OWS Namespace
Running	ows	OWS Application
Running	ohp	OWS Client
Running	ofagtrack	Connected User Experiences and Tele...
Running	ofagtrackbackt...	Ofagtrack Policy Service
Running	ocache	OWS Client
Running	ocache	Delivery Optimization
Running	ows	Diagnostic Policy Service
Running	ocacv	Device Setup Manager
Running	ocacv	Data Sharing Service
Running	eventlog	Windows Event Log
Running	eventlog	COM+ Event System
Running	fdmact	Function Discovery Provider Host
Running	hwsapi	Function Discovery Resource Public...
Running	hwcach	Windows Host Cache Service
Running	gpcv	Group Policy Client
Running	hlpv	SR Helper
Running	oavp	OWS Web Application
Running	lsmserver	Server
Running	lsmworkstation	Workstation
Running	lsmmanager	Windows License Manager Service
Running	lsmhost	Topology Network Helper
Running	lsm	Local Session Manager
Running	agpcv	Windows Defender Firewall
Running	hwc	Distributed Transaction Coordinator
Running	hwcapi	Windows Internal Database
Running	hwcapi	Network Connection Broker
Running	hwcapi	Network List Service
Running	hwcapi	Network Location Awareness
Running	hwcapi	Network Store Interface Service
Running	hwcapi	Program Compatibility Assistant Ser...
Running	hwcapi	Plug and Play
Running	hwcapi	Power Policy Agent
Running	hwcapi	Power
Running	hwcapi	User Profile Service
Running	hwcapi	GenS Act Agent
Running	hwcapi	Remote Access Management Service
Running	hwcapi	Remote Access Connection Manager
Running	hwcapi	Routing and Remote Access
Running	hwcapi	RPC Endpoint Mapper
Running	hwcapi	Remote Procedure Call (RPC)
Running	hwcapi	Security Accounts Manager
Running	hwcapi	Task Scheduler
Running	hwcapi	System Event Notification Service
Running	hwcapi	Shell Hardware Detection
Running	hwcapi	Spooler Service
Running	hwcapi	Print Spooler
Running	hwcapi	Secure Socket Tunneling Protocol Se...
Running	hwcapi	Store Repository Service
Running	hwcapi	Storage Service
Running	hwcapi	System
Running	hwcapi	System Event Broker
Running	hwcapi	Touch Keyboard and Handwriting Mana...
Running	hwcapi	Themes
Running	hwcapi	Time Broker
Running	hwcapi	Web Account Manager
Running	hwcapi	Distributed Link Tracking Client
Running	hwcapi	User Access Logging Service
Running	hwcapi	User Manager
Running	hwcapi	Update Orchestrator Service
Running	hwcapi	Virtual Disk
Running	hwcapi	Windows Time
Running	hwcapi	World Wide Web Publishing Service
Running	hwcapi	Windows Process Activation Service
Running	hwcapi	Windows Connection Manager
Running	hwcapi	Diagnostic System Host
Running	hwcapi	Microsoft Defender Antivirus Sensor...
Running	hwcapi	Windows Internal Database VSS Writer
Running	hwcapi	Microsoft Defender Antivirus Service
Running	hwcapi	Windows Web Proxy Auto-Discovery Se...
Running	hwcapi	Windows Management Instrumentation
Running	hwcapi	Windows Remote Management (WinRM)
Running	hwcapi	Windows Licensing Monitoring Service
Running	hwcapi	Windows Push Notifications System S...
Running	hwcapi	Windows Push Notifications User Ser...
Running	hwcapi	Windows Update

Figure 28

## Step 46 — Sorting Services by Dependency Count

I then ran the same command but sorted the results based on how many services depend on each:

```
141
142 Get-Service | Where-Object {$_.Status -eq "Running"} | Sort-Object -Descending {$_.DependentServices.Count}
```

Status	Name	DisplayName
Running	DcomLaunch	DCOM Server Process Launcher
Running	RpcEptMapper	RPC Endpoint Mapper
Running	RpcSs	Remote Procedure Call (RPC)
Running	nsI	Network Store Interface Service
Running	BFE	Base Filtering Engine
Running	BrokerInfrastructure...	Background Tasks Infrastructure Ser...
Running	Dhcp	DHCP Client
Running	Dnscache	DNS Client
Running	SamSs	Security Accounts Manager
Running	ProfSvc	User Profile Service
Running	EventSystem	COM+ Event System
Running	Eventlog	Windows Event Log
Running	SstpSvc	Secure Socket Tunneling Protocol Se...
Running	LanmanWorkstation	Workstation
Running	RasMan	Remote Access Connection Manager
Running	WIDWriter	Windows Internal Database VSS Writer
Running	CryptSvc	Cryptographic Services
Running	NlaSvc	Network Location Awareness
Running	WinHttpAutoProx...	WinHTTP Web Proxy Auto-Discovery Se...
Running	Wsmgmt	Windows Management Instrumentation
Running	RemoteAccess	Routing and Remote Access
Running	KeyIso	CNG Key Isolation
Running	netprofm	Network List Service
Running	LanmanServer	Server
Running	NcbService	Network Connection Broker
Running	MSSQL\$MICROSOFT...	Windows Internal Database
Running	iphlpvc	IP Helper
Running	SENS	System Event Notification Service
Running	SystemEventsBroker	System Events Broker
Running	UserManager	User Manager
Running	StateRepository	State Repository Service
Running	WAS	Windows Process Activation Service
Running	fdPHost	Function Discovery Provider Host
Running	StorSvc	Storage Service
Running	SysMain	SysMain
Running	Spooler	Print Spooler

Figure 29

## Step 46 — Sorting Services by Dependency Count

All screenshot provided in above two steps

## Step 48 — Importance of Knowing Running Services for Host Security

It is important for host security to know what services are running because:

- **Attack Surface Reduction:** Each running service is a potential entry point. Disabling unnecessary ones minimizes risk.
- **Malware Detection:** Attackers often install rogue services that persist after reboot. Monitoring reveals anomalies.
- **Performance & Stability:** Resource-heavy or misconfigured services can degrade system performance and availability.



- **Privilege & Access Control:** Some services run with elevated privileges; identifying them helps ensure they are hardened and monitored.
- **Compliance & Forensics:** Auditing services is required for many standards (e.g., NIST, ISO 27001) and helps during incident investigations.

**In summary:** Actively monitoring services ensures that only legitimate, necessary, and secure services are running, thereby reducing risks and supporting overall host hardening.

## Exercise VIII: Develop your system admin script

### Step 49 — Developing the sys\_admin.ps1 Script

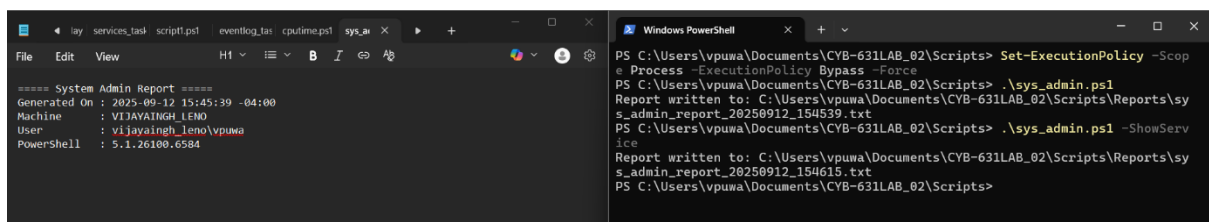
I created a new PowerShell script named **sys\_admin.ps1**. This script was designed for system administrators to collect host information and produce a report. The script includes the following functionality:

- Records the date and time of the report.
- Accepts a **-ShowService** parameter. If included, it lists the top five running services sorted by the number of dependent services. If omitted, this section is skipped.
- In both cases, it summarizes entries from the Security event log, grouped and sorted by source name, showing the top five sources.

The script saves its output as a text file inside the Reports directory, with filenames based on the current timestamp.

### Step 50 — sys\_admin.ps1 Script File

I saved the script as **sys\_admin.ps1** in the Scripts directory of my Lab 2 folder. A screenshot of the script file in the PowerShell ISE/VS Code editor is shown in Figure XX.



```

===== System Admin Report =====
Generated On : 2025-09-12 15:45:39 -04:00
Machine      : VIJAYAINGH_LENO
User         : vijayaingh_leno\vpuma
PowerShell   : 5.1.26100.6584

PS C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts> Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass -Force
PS C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts> .\sys_admin.ps1
Report written to: C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts\Reports\sys_admin_report_20250912_154539.txt
PS C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts> .\sys_admin.ps1 -ShowService
Report written to: C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts\Reports\sys_admin_report_20250912_154615.txt
PS C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts>

```

Figure 30

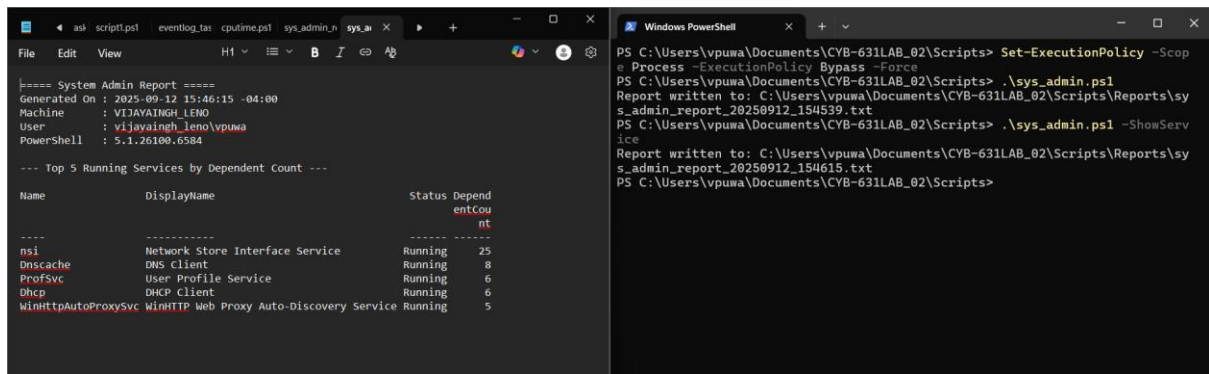


Figure 31

## Step 51 — Running sys\_admin.ps1

I executed the script in two ways:

1. Without parameters:
2. `.\sys_admin.ps1`

This generated a report with timestamp, machine information, and Security log summary.

Example header from the report:

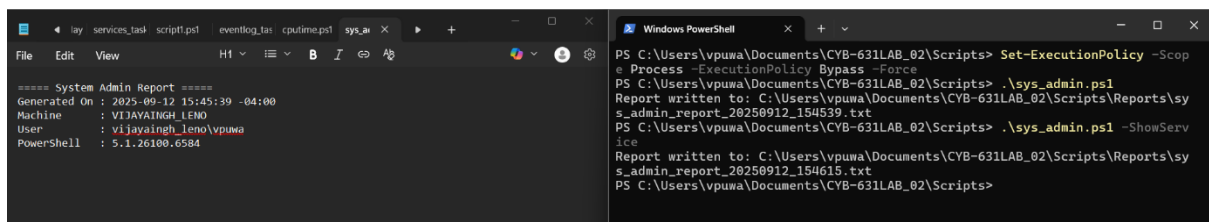


Figure 32

2. With the **-ShowService** parameter:
3. `.\sys_admin.ps1 -ShowService`

This generated the same header along with the **Top 5 Running Services by Dependent Count** table:

The image shows a Windows desktop with two windows. The left window is a Notepad++ editor with the file 'sys\_admin.ps1' open. It displays a 'System Admin Report' generated on 2025-09-12 15:46:15. The report includes system information and a table of the top 5 running services by dependent count.

Name	DisplayName	Status	Dependent Count
nsi	Network Store Interface Service	Running	25
DnsCache	DNS Client	Running	8
ProfSvc	User Profile Service	Running	6
Dhcp	DHCP Client	Running	6
WinHttpAutoProxySvc	WinHTTP Web Proxy Auto-Discovery Service	Running	5

The right window is a Windows PowerShell terminal. It shows the execution of a script to generate the report. The commands executed are:

```
PS C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts> Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass -Force
PS C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts> .\sys_admin.ps1
Report written to: C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts\Reports\sys_admin_report_20250912_154539.txt
PS C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts> .\sys_admin.ps1 -ShowService
Report written to: C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts\Reports\sys_admin_report_20250912_154615.txt
PS C:\Users\vpuma\Documents\CYB-631LAB_02\Scripts>
```

Figure 33

## Lab and Class Reflection

### 1. What did I like about this lab?

I appreciated that this lab combined multiple aspects of PowerShell into a structured workflow. It started with object operations and arrays, then moved into modules, event logs, and finally developing a complete administrative script. The progression showed me how small building blocks can be combined into powerful tools. Writing and running my own `sys_admin.ps1` felt very practical and connected directly to real system administration tasks.

### 2. What challenges did I encounter?

The main challenges were handling execution policies and accessing the Security event log. Initially, I received errors due to restricted script execution and insufficient privileges when trying to read Security logs. I resolved these by using a **process-scoped policy bypass** for testing and running PowerShell with administrative privileges. Another challenge was dealing with null values for `DependentServices.Count`, which required rewriting the script logic to avoid runtime errors.

### 3. Suggestions for improving the class:

It would be helpful if the lab instructions included more examples directly related to **cybersecurity use cases**, such as analyzing failed login attempts or automating log correlation. Including short troubleshooting notes (e.g., why Security log access fails without Admin rights) would save time for students new to PowerShell. Additional practice on error handling in scripts would also reinforce the idea of writing resilient automation tools.

### 4. Turning off virtual machines:

As instructed, I shut down the Windows virtual machine after completing the lab to free up server resources.

## References

Holmes, L. (2021). *Windows PowerShell cookbook* (4th ed.). O'Reilly Media.

Microsoft. (2023, June 12). *Table of basic PowerShell commands*. Microsoft DevBlogs.

<https://devblogs.microsoft.com/scripting/table-of-basic-powershell-commands/>

Pace University. (2025). *CYB 631: Automating Information Security with Python and Shell*

*Scripting – Lab 2: Analyzing Logs and Other Administrators' Tasks* [Course handout].

Department of Computer Science, Seidenberg School of CSIS.