

Batch Information:

- **Batch Start Date:** 2025-08-11
- **Batch Name:** WiproNGA_DWS_B5_25VID2550
- **First Name:** Sameer
- **Last Name:** Katre
- **User ID:** 34930
- **Batch ID:** B5-25VID2550

ASSIGNMENTS

- ✓ **Introducing to Cmdlets**
 - ✓ **The PowerShell Pipeline**
 - ✓ **Key Cmdlets**
 - ✓ **WMI & PowerShell**
 - ✓ **Pipeline Filtering & Operators**
 - ✓ **Input, Output & Formatting**
 - ✓ **Scripting Overview**
-

➤ Introducing to Cmdlets.

1. What are Cmdlets?

- **Definition:** Cmdlets (pronounced *command-lets*) are lightweight, single-function commands built into PowerShell.
- They are **specialized .NET classes** that perform specific tasks.
- Unlike traditional command-line tools, Cmdlets **return objects**, not plain text — making it easier to pass data between commands.

2. Characteristics of Cmdlets

- **Verb-Noun Naming Convention:** Example: Get-Process, Set-Date, New-Item
- **Consistent Syntax:** The same structure applies to all Cmdlets, making them easier to learn.
- **Integrated with the .NET Framework:** They can access system APIs and objects directly.
- **Pipeline Support:** Cmdlets can accept input from other Cmdlets and send output to others.

4. Common Cmdlets Examples

- **Get-Command** → Lists all available Cmdlets and functions.
- **Get-Help** → Displays help information about a Cmdlet.
- **Get-Process** → Shows running processes.
- **Stop-Process** → Stops a specific process.
- **Set-ExecutionPolicy** → Changes script execution permission.

The screenshot shows a Windows PowerShell console window with a list of system services and their status. The services are listed in two columns: the first column shows the status (e.g., Stopped, Running) and the second column shows the service name (e.g., wercplsupport, WerSvc). Below the list, the user has entered the command `Stop-Service` and the prompt `cmdlet Stop-Service at command pipeline position 1`. The user has also entered `Start-Service` and the prompt `cmdlet Start-Service at command pipeline position 1`. To the right of the console window, there is a script editor window titled `Untitled1.ps1*` containing a PowerShell script with 12 lines of code, including comments and commands like `Get-Process`, `Get-Service`, `Stop-Service`, and `Start-Service`.

```
File Edit View Tools Debug Add-ons Help
wercplsupport Problem Reports Control Panel Support
WerSvc Windows Error Reporting Service
WFDSCComMgrSvc Wi-Fi Direct Services Connection Ma...
whesvc Windows Health and Optimized Experi...
WiaRpc Still Image Acquisition Events
WinDefend Microsoft Defender Antivirus Service
WinHttpAutoProx... WinHTTP Web Proxy Auto-Discovery Se...
Winmgmt Windows Management Instrumentation
WinRM Windows Remote Management (WS-Manag...
WISvc Windows Insider Service
WlanSvc WLAN AutoConfig
Wlidsvc Microsoft Account Sign-in Assistant
Wlpasvc Local Profile Assistant Service
WManSvc Windows Management Service
WmiApSrv WMI Performance Adapter
WMIRegistration... Intel(R) Management Engine WMI Prov...
WMPNetworkSvc Windows Media Player Network Sharin...
workfolderssvc Work Folders
WpcMonSvc Parental Controls
WPDBusEnum Portable Device Enumerator Service
WpnService Windows Push Notifications System S...
WpnUserService... Windows Push Notifications User Ser...
WSAIFabricSvc WSAIFabricSvc
wscsvc Security Center
WSearch Windows Search
Wuauserv Windows Update
WwanSvc WWAN AutoConfig
XblAuthManager Xbox Live Auth Manager
XblGameSave Xbox Live Game Save
XboxGipSvc Xbox Accessory Management Service
XboxNetApiSvc Xbox Live Networking Service

PS C:\Users\hs150> Stop-Service
cmdlet Stop-Service at command pipeline position 1
Supply values for the following parameters:
InputObject[0]:
PS C:\Users\hs150>
Start-Service
cmdlet Start-Service at command pipeline position 1
Supply values for the following parameters:
InputObject[0]:
PS C:\Users\hs150>
```

```
1 Get-Process
2 #- Shows all running processes on your system.
3
4 Get-Service
5 #- Lists all services and their status.
6
7 Stop-Service
8 #- Stops a running service.
9
10 Start-Service
11 # Starts a stopped service.
12
```

➤ The PowerShell Pipeline.

1. What is the PowerShell Pipeline?

- The pipeline (|) in PowerShell allows you to pass the output of one command directly as the input to another command.
- This helps chain multiple commands together to perform complex tasks efficiently.
- It is similar to pipelines in Unix/Linux shells but works differently because PowerShell passes objects, not plain text.

2. How the Pipeline Works

1. First command runs and produces objects.
2. These objects are streamed one by one into the next command.
3. The next command processes each object and passes the result to the next stage, and so on.

Example: `Get-Process | Where-Object CPU -gt 100 | Sort-Object CPU -Descending`

- **Get-Process** → gets all running processes (object data).
- **Where-Object** → filters processes with CPU usage greater than 100.
- **Sort-Object** → sorts them in descending CPU usage.

```
Administrator: Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Administrator> Get-Process | Sort-Object WS -Descending | Select-Object -First 5
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
1249	135	1526868	1343920	70.73	4972	0	sqlservr
10440	22467	485968	465348	0.73	3068	0	dns
799	118	175024	204500	14.36	3660	0	sccmprovidergraph
1619	87	107948	183408	8.30	5196	1	SearchApp
4011	132	137988	180240	35.11	2760	0	smsexec

```
PS C:\Users\Administrator>
```

➤ Key Cmdlets.

1. What are Cmdlets?

- **Definition:** Cmdlets (pronounced *command-lets*) are lightweight PowerShell commands built into the shell or added via modules.

```
SystemInfo.ps1  Untitled6.ps1* X
1 # Get help on a cmdlet
2 get-help Get-Process -Full
3
4 # List all available cmdlets and functions
5 get-command
6
7 # List commands related to 'Service'
8 get-command *Service*
9
10 # Show properties and methods of a process object
11 get-process | Get-Member
12
13 # --- Working with Data and Files ---
14 # Read content from a file
15 get-content "C:\Temp\sample.txt"
16
17 # Write content to a file (overwrite)
18 "Hello world" | Set-Content "C:\Temp\sample.txt"
19
20 # Append content to a file
21 "New line added" | Add-Content "C:\Temp\sample.txt"
22
23 # Get running processes
24 get-process
25
26 # Get details for a specific process
27 get-process notepad
28
29 # Stop a process by name
30 Stop-Process -Name notepad -Force
31
32 # List all windows services
33 get-service
34
35 # Start a specific service
36 Start-Service -Name "Spooler"
37
38 # Stop a specific service
39 Stop-Service -Name "Spooler"
40
41 # Filter processes with CPU time greater than 50 seconds
42 Get-Process | Where-Object {$_.CPU -gt 50}
43
44 # Sort services alphabetically by Display Name
45 Get-Service | Sort-Object DisplayName
46
47 # Create a variable
48 Set-Variable -Name MyVar -value "Tirtha"
49
50 # Display variable value
51 Write-Output $MyVar
52
53 # Display text directly to the console in green
54 Write-Host "This is PowerShell!" -ForegroundColor Green
55
56
57 # --- Combined Example ---
58 # Get all stopped services, sort them, and export to csv
59 Get-Service | Where-Object {$_.Status -eq "Stopped"} |
60 Sort-Object DisplayName |
61 Export-Csv "C:\Temp\stopped_services.csv" -NoTypeInformation
62
63 Write-Host "Script completed. Stopped services exported to csv." -ForegroundColor cyan
```

- **Format:** Verb-Noun (e.g., Get-Process, Set-Item).
- **Purpose:** Designed to perform a single function, but can be combined in pipelines to accomplish complex tasks

➤ WMI & PowerShell.

1. What is WMI?

- **Full Form:** Windows Management Instrumentation
- **Purpose:** Provides a standardized way to access and manage Windows system components (hardware, OS settings, applications) locally or remotely.
- **Data Source:** Information is stored in the CIM (Common Information Model) repository.

2. Why Use WMI with PowerShell?

- **Automation:** Perform administrative tasks without manually using GUI tools.
- **Remote Management:** Query or configure computers over the network.
- **Detailed Information:** Access system hardware details, running processes, services, network configurations, etc.
- **Scripting Power:** Combine WMI queries with PowerShell cmdlets for reporting, monitoring, and troubleshooting.

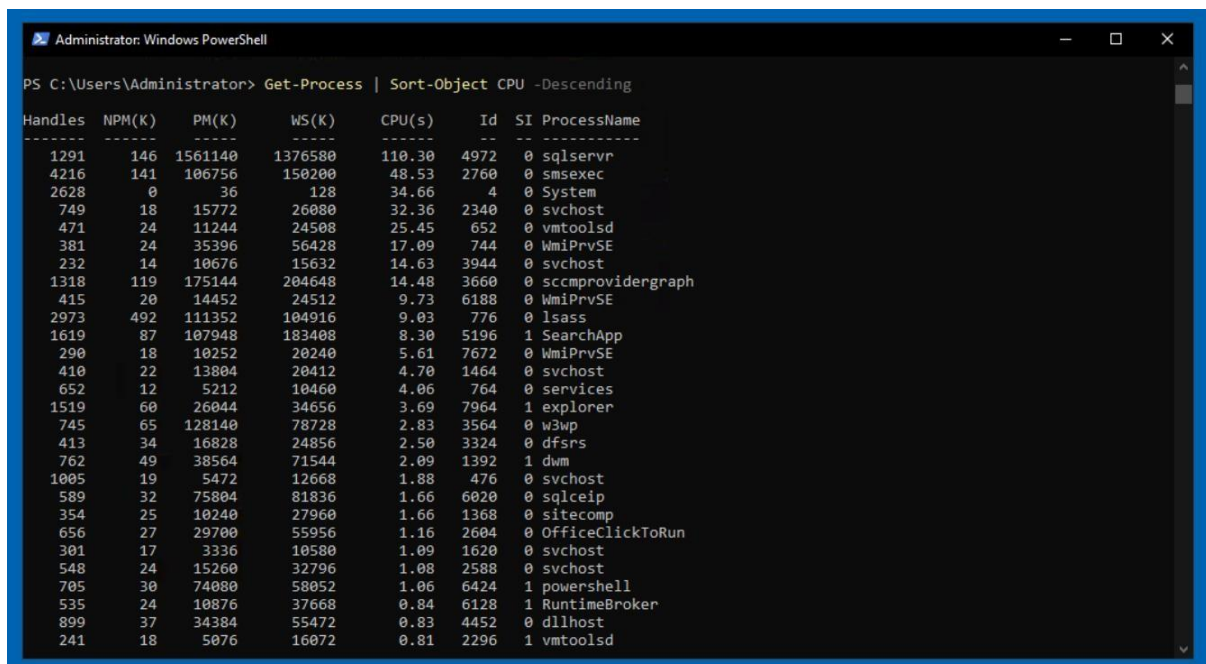
3. Key PowerShell Cmdlets for WMI.

Cmdlet	Purpose
Get-WmiObject (<i>legacy</i>)	Retrieves WMI class instances. Example: Get-WmiObject -Class Win32_OperatingSystem
Get-CimInstance (<i>recommended</i>)	Modern alternative to Get-WmiObject; uses WS-Man protocol for better compatibility.
Invoke-WmiMethod	Executes a method of a WMI object.
Set-WmiInstance	Modifies a WMI object instance.
Remove-WmiObject	Deletes a WMI object instance.

➤ Pipeline Filtering & Operators.

1. What is a Pipeline in PowerShell?

- A **pipeline** (|) in PowerShell passes the **output** of one command as the **input** to another.
- Example: ***Get-Process | Sort-Object CPU -Descending***
 - Get-Process lists processes.
 - Sort-Object takes that list (from the pipeline) and sorts it.



```
PS C:\Users\Administrator> Get-Process | Sort-Object CPU -Descending
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
1291	146	1561140	1376580	110.30	4972	0	sqlservr
4216	141	106756	150200	48.53	2760	0	smsexec
2628	0	36	128	34.66	4	0	System
749	18	15772	26080	32.36	2340	0	svchost
471	24	11244	24508	25.45	652	0	vmtoolsd
381	24	35396	56428	17.09	744	0	WmiPrvSE
232	14	10676	15632	14.63	3944	0	svchost
1318	119	175144	204648	14.48	3660	0	scmprovidergraph
415	20	14452	24512	9.73	6188	0	WmiPrvSE
2973	492	111352	104916	9.03	776	0	lsass
1619	87	107948	183408	8.30	5196	1	SearchApp
290	18	10252	20240	5.61	7672	0	WmiPrvSE
410	22	13804	20412	4.70	1464	0	svchost
652	12	5212	10460	4.06	764	0	services
1519	60	26044	34656	3.69	7964	1	explorer
745	65	128140	78728	2.83	3564	0	w3wp
413	34	16828	24856	2.50	3324	0	dfsrs
762	49	38564	71544	2.09	1392	1	dwm
1005	19	5472	12668	1.88	476	0	svchost
589	32	75804	81836	1.66	6020	0	sqlceip
354	25	10240	27960	1.66	1368	0	sitecomp
656	27	29700	55956	1.16	2604	0	OfficeClickToRun
301	17	3336	10580	1.09	1620	0	svchost
548	24	15260	32796	1.08	2588	0	svchost
705	30	74080	58052	1.06	6424	1	powershell
535	24	10876	37668	0.84	6128	1	RuntimeBroker
899	37	34384	55472	0.83	4452	0	dllhost
241	18	5076	16072	0.81	2296	1	vmtoolsd

Key Points:

- Allows chaining multiple commands.
- Reduces need for temporary variables.
- Processes data in a **streaming** fashion (one object at a time).

2. Filtering in the Pipeline.

Filtering means narrowing results to only what you need.

- **Where-Object** – Filters based on a condition.

Get-Process | Where-Object { \$_.CPU -gt 100 }

- **Select-Object** – Picks specific properties or limits results.

Get-Process | Select-Object Name, CPU

Get-Process | Select-Object -First 5

- **Sort-Object** – Sorts results by properties.

Get-Service | Sort-Object Status, Name

3. Operators in PowerShell

Operators are symbols that perform actions on data.

a) Comparison Operators

Operator	Description	Example
-eq	Equal to	5 -eq 5
-ne	Not equal to	5 -ne 3
-gt	Greater than	10 -gt 5
-lt	Less than	3 -lt 5
-ge	Greater or equal	5 -ge 5
-le	Less or equal	5 -le 10
-like	Wildcard match	"file.txt" -like "*.txt"
-match	Regex match	"Hello" -match "H.*o"

b) Logical Operators

Operator	Description	Example
-and	Both conditions true	(5 -gt 2) -and (3 -lt 5)
-or	At least one true	(5 -lt 2) -or (3 -lt 5)
-not / !	Negates condition	-not (5 -gt 10)


```
PS C:\Users\Administrator> 5 -gt 5
False
PS C:\Users\Administrator> 56 -lt 87
True
PS C:\Users\Administrator> _
```

➤ Input, Output & Formatting.

In PowerShell, **input**, **output**, and **formatting** control how data is received from the user or system, how results are returned, and how those results are displayed.

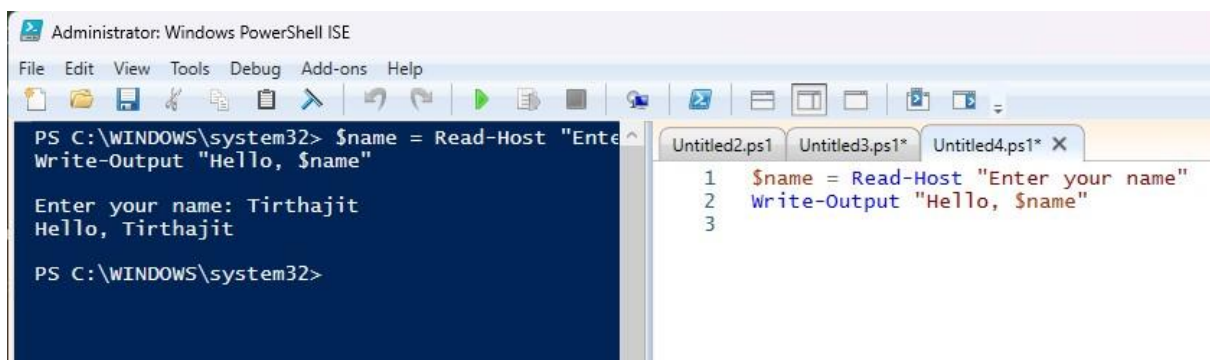
- **Input:** Data or commands provided to a script, function, or cmdlet.
- **Output:** Data or results sent from a cmdlet, script, or pipeline.
- **Formatting:** How output data is presented (table, list, custom views, etc.).

1. Input in PowerShell

a. Types of Input

1. From the Keyboard (User Input)

- Read-Host → Prompts the user to enter text or data during script execution.



```
Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
PS C:\WINDOWS\system32> $name = Read-Host "Enter your name"
Write-Output "Hello, $name"
Enter your name: Tirthajit
Hello, Tirthajit
PS C:\WINDOWS\system32>

1 $name = Read-Host "Enter your name"
2 Write-Output "Hello, $name"
3
```

- Use -AsSecureString to hide sensitive input like passwords.

2. From Command Parameters

- Cmdlets and functions accept parameters directly.

```
Untitled2.ps1  Untitled3.ps1*  Untitled4.ps1*  Untitled5.ps1* X
1  Get-Process -Name Notepad
2
3  # Get-Process → This is the cmdlet. It lists the processes currently running on your system
4  # -Name → This is a parameter. It tells the cmdlet to filter results by process name.
5  # notepad → This is the argument (value) you pass to the -Name parameter – in this case,
6  # you're asking for the process named "notepad".
```

3. From Pipeline

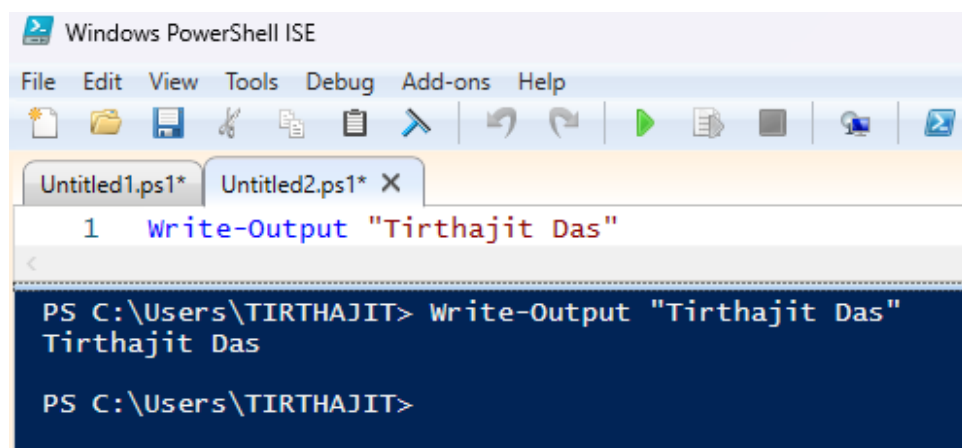
- One command's output becomes another's input.

```
Untitled1.ps1* X
1  Get-Process | Where-Object {$_.CPU -gt 100}
2
3  # Step 1: Get-Process outputs all processes.
4  # Step 2: Each process object is passed one-by-one to Where-Object.
5  # Step 3: Only those processes with CPU usage greater than 100 seconds are kept.
6  # Step 4: The filtered list is shown as the final output.
```

3. Output in PowerShell

a. Output Cmdlets

1. **Write-Output** → Sends objects to the pipeline (default output behavior).



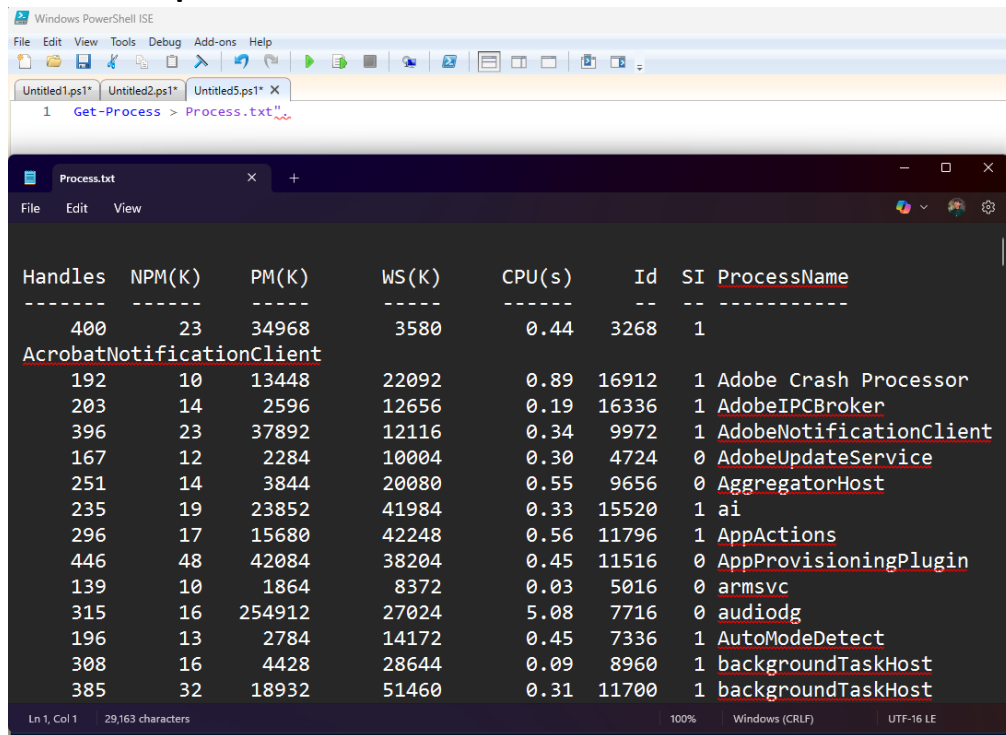
The screenshot shows the Windows PowerShell ISE interface. The menu bar includes File, Edit, View, Tools, Debug, Add-ons, and Help. Below the menu is a toolbar with icons for file operations and execution. Two tabs are open: 'Untitled1.ps1*' and 'Untitled2.ps1* X'. The active tab 'Untitled1.ps1*' contains the command `1 Write-Output "Tirthajit Das"`. The console window at the bottom shows the command being executed: `PS C:\Users\TIRTHAJIT> Write-Output "Tirthajit Das"`, followed by the output `Tirthajit Das`, and then the prompt `PS C:\Users\TIRTHAJIT>`.

2. **Write-Host** → Displays text directly on the console (doesn't send to pipeline).

```
PS C:\Users\TIRTHAJIT> Write-Host "Tirtha"
Tirtha
PS C:\Users\TIRTHAJIT>
```

b. Sending Output to Files

- **Redirect Operators:**



The screenshot shows the Windows PowerShell ISE interface. The command prompt shows the command `Get-Process > Process.txt` being executed. Below the command prompt, a file explorer window titled 'Process.txt' displays the output of the command as a table of system processes.

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
400	23	34968	3580	0.44	3268	1	
<u>AcrobatNotificationClient</u>							
192	10	13448	22092	0.89	16912	1	Adobe Crash Processor
203	14	2596	12656	0.19	16336	1	AdobeIPCBroker
396	23	37892	12116	0.34	9972	1	AdobeNotificationClient
167	12	2284	10004	0.30	4724	0	AdobeUpdateService
251	14	3844	20080	0.55	9656	0	AggregatorHost
235	19	23852	41984	0.33	15520	1	ai
296	17	15680	42248	0.56	11796	1	AppActions
446	48	42084	38204	0.45	11516	0	AppProvisioningPlugin
139	10	1864	8372	0.03	5016	0	armsvc
315	16	254912	27024	5.08	7716	0	audiodg
196	13	2784	14172	0.45	7336	1	AutoModeDetect
308	16	4428	28644	0.09	8960	1	backgroundTaskHost
385	32	18932	51460	0.31	11700	1	backgroundTaskHost

4. Formatting Output

PowerShell outputs objects, not plain text, so formatting changes **how** they're displayed — without changing the data.

a. Common Formatting Cmdlets

1. Format-Table (ft) – Displays data in a table.

```
PS C:\Users\Administrator> Get-Process | Format-Table NAME, CPU, Id
```

Name	CPU	Id
AggregatorHost	0.03125	6180
CcmExec	2.59375	3124
cmupdate	0.75	8280
conhost	0.015625	3716
csrss	0.65625	520
csrss	0.515625	628
ctfmon	0.734375	1860
dfsrs	7.234375	3324
dfssvc	0.125	3288
dllhost	1.640625	4452
dllhost	0.453125	5228
dllhost	0.15625	7540
dns	0.875	3068
dwm	4.109375	1392
explorer	6.234375	7964
fontdrvhost	0.015625	3796
fontdrvhost	0.109375	3804
Idle		0
inetinfo	0.1875	656
IpOverUsbSvc	0.078125	3352
ismserv	0	704
lsass	37.546875	776
Microsoft.ActiveDirectory.WebServices	1.453125	2332
msdtc	0.046875	6924
OfficeClickToRun	1.265625	2604
powershell_ise	10.71875	5452
Registry	1.296875	172

2. Format-List (fl) – Shows properties in a vertical list.

```
PS C:\Users\Administrator> Get-Process | Format-List *
```

Name	: AggregatorHost
Id	: 6180
PriorityClass	: Normal
FileVersion	:
HandleCount	: 74
WorkingSet	: 4567040
PagedMemorySize	: 860160
PrivateMemorySize	: 860160
VirtualMemorySize	: 52928512
TotalProcessorTime	: 00:00:00.0312500
SI	: 0
Handles	: 74
VM	: 2203371151360
WS	: 4567040
PM	: 860160
NPM	: 5840
Path	: C:\Windows\System32\AggregatorHost.exe
Company	:
CPU	: 0.03125
ProductVersion	:
Description	:
Product	:
__NounName	: Process
BasePriority	: 8
ExitCode	:
HasExited	: False
ExitTime	:
Handle	: 5116

3. Format-Wide – Displays a single property across the screen.

```
PS C:\Users\TIRTHAJIT> Get-Process | Format-Wide Name -Column 3

AcrobatNotificationClient      Adobe Crash Processor          AdobeIPCBroker
AdobeNotificationClient      AdobeUpdateService            AggregatorHost
ai                             AppActions                    AppProvisioningPlugin
armsvc                        audiodg                       AutoModeDetect
backgroundTaskHost           backgroundTaskHost             backgroundTaskHost
CCXProcess                   chrome                        chrome
chrome                       chrome                        chrome
chrome                       chrome                        chrome
chrome                       chrome                        chrome
chrome                       chrome                        chrome
conhost                      conhost                       conhost
CrossDeviceResume            csrss                         csrss
ctfmon                       DataExchangeHost              DAX3API
DAX3API                      dllhost                       dllhost
dllhost                      dwm                           ElevocControlService
explorer                     FaceBeautify                  FnHotkeyCapsLKNumLK
FnHotkeyUtility              fontdrvhost                   fontdrvhost
gamingservices               gamingservicesnet             Idle
igcc                         igccTray                     IntelAudioService
IntelCpHDCPSvc              ipf_helper                    ipf.uf
ipfsvc                       jhi_service                   Lenovo.Modern.InController
LenovoUtilityService         LenovoVantage- (GenericMessagingAddin)
LenovoVantageService         LNBITSvc                     LenovoVantage- (VantageCoreAddin)
LsaIso                       LsaRpcServer                  Locator
LsaToast                     Memory Compression            Tsass
MpDefenderCoreService        msedgebview2                 Microsoft.SharePoint
msedgebview2                 msedgebview2                 msedgebview2
msedgebview2                 msedgebview2                 msedgebview2
```

b. Customizing Output

- Select-Object to pick properties: *Get-Process | Select-Object Name, Id*
- Sort-Object for ordering: *Get-Process | Sort-Object CPU -Descending*

➤ Scripting Overview.

1. What is a PowerShell Script?

A **PowerShell script** is simply a collection of PowerShell commands saved in a file with the .ps1 extension. Instead of running commands one by one in the console, we can automate tasks by executing the script file.

2. Benefits of Using PowerShell Scripts

- **Automation** – Execute repetitive tasks without manual intervention.
- **Consistency** – Ensure the same actions are performed every time.
- **Time-saving** – Execute multiple commands in seconds.
- **Scalability** – Manage hundreds of systems at once.
- **Integration** – Works with Windows, WMI, .NET, and external APIs.

Untitled1.ps1* X

```
1 # Script: SystemInfo.ps1
2 # Author: Tirthajit Das
3 # Description: Displays basic system information
4
5 Write-Host "COMPUTER INFORMATION"
6 Write-Host "-----"
7
8 $COMPUTERNAME = $env:COMPUTERNAME
9
10 $os = Get-CimInstance Win32_OperatingSystem
11
12 $cpu = Get-CimInstance Win32_Processor
13
14 Write-Host "Computer Name: $COMPUTERNAME"
15 Write-Host "Operating System: $os"
16 Write-Host "CPU: $cpu"
```

```
PS C:\Users\TIRTHAJIT> # Script: SystemInfo.ps1
# Author: Tirthajit Das
# Description: Displays basic system information

Write-Host "COMPUTER INFORMATION"
Write-Host "-----"

$COMPUTERNAME = $env:COMPUTERNAME

$os = Get-CimInstance Win32_OperatingSystem

$cpu = Get-CimInstance Win32_Processor

Write-Host "Computer Name: $COMPUTERNAME"
Write-Host "Operating System: $os"
Write-Host "CPU: $cpu"
COMPUTER INFORMATION
-----
Computer Name: TIRTHAJIT
Operating System: Win32_OperatingSystem: Microsoft Windows 11 Home Single Language
CPU: Win32_Processor: Intel64 Family 6 Model 186 Stepping 2 (DeviceID = "CPU0")

PS C:\Users\TIRTHAJIT>
```