Batch Information:

- **Batch Start Date:** 2025-08-04
- **Batch Name:** WiproNGA_DWS_B5_25VID2550
- **First Name:** Tirthajit
- **Last Name:** Das
- **User ID:** 34940
- **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?

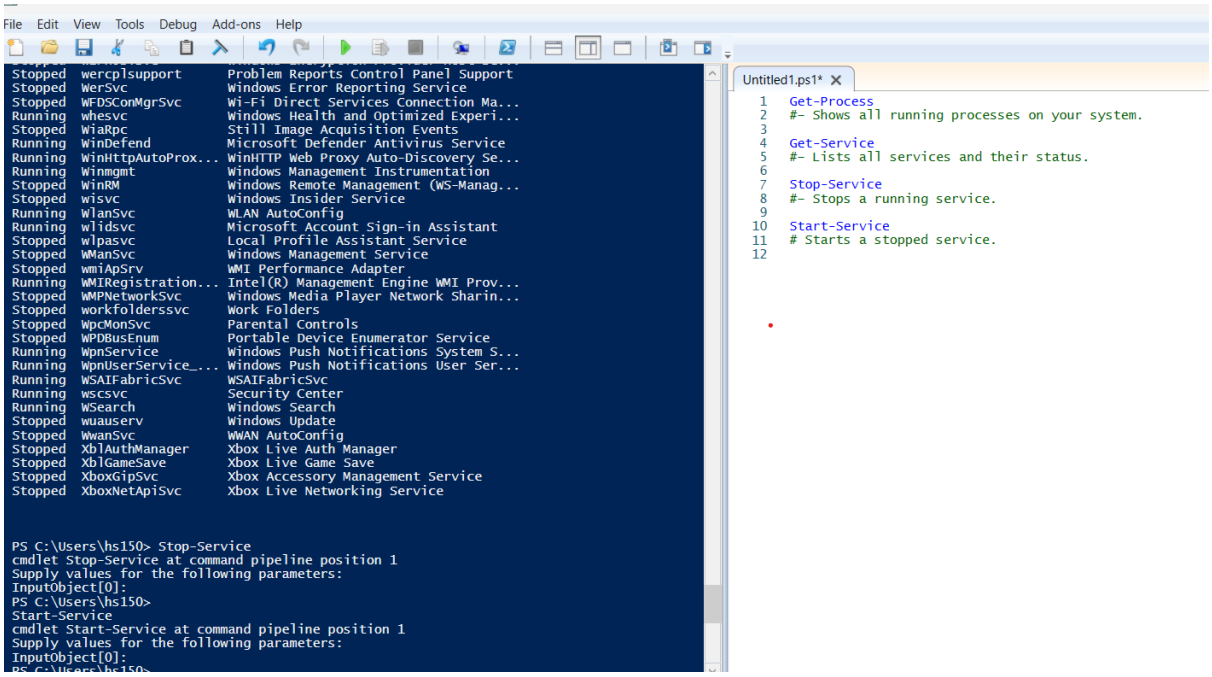
- **Definition:** Cmdlets (pronounced *command-lets*) are lightweight, single-function commands built into PowerShell.
- They are **specialized .NET classes** that perform specific tasks.
- Unlike traditional command-line tools, Cmdlets **return objects**, not plain text — making it easier to pass data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

```powershell
# Get help on a cmdlet
Get-Help Get-Process -Full

# List all available cmdlets and functions
Get-Command

# List commands related to 'Service'
Get-Command *Service*

# Show properties and methods of a process object
Get-Process | Get-Member

# --- Working with Data and Files ---
# Read content from a file
Get-Content "C:\Temp\sample.txt"

# Write content to a file (overwrite)
"Hello World" | Set-Content "C:\Temp\sample.txt"

# Append content to a file
"New line added" | Add-Content "C:\Temp\sample.txt"

# Get running processes
Get-Process

# Get details for a specific process
Get-Process notepad

# Stop a process by name
Stop-Process -Name notepad -Force

# List all Windows services
Get-Service

# Start a specific service
Start-Service -Name "Spooler"

# Stop a specific service
Stop-Service -Name "Spooler"

# Filter processes with CPU time greater than 50 seconds
Get-Process | Where-Object {$_.CPU -gt 50}

# Sort services alphabetically by Display Name
Get-Service | Sort-Object DisplayName

# Create a variable
Set-Variable -Name MyVar -Value "Tirtha"

# Display variable value
Write-Output $MyVar

# Display text directly to the console in green
Write-Host "This is PowerShell!" -ForegroundColor Green


# --- Combined Example ---
# Get all stopped services, sort them, and export to CSV
Get-Service | Where-Object {$_.Status -eq "Stopped"} |
Sort-Object DisplayName |
Export-Csv "C:\Temp\stopped_services.csv" -NoTypeInformation

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.

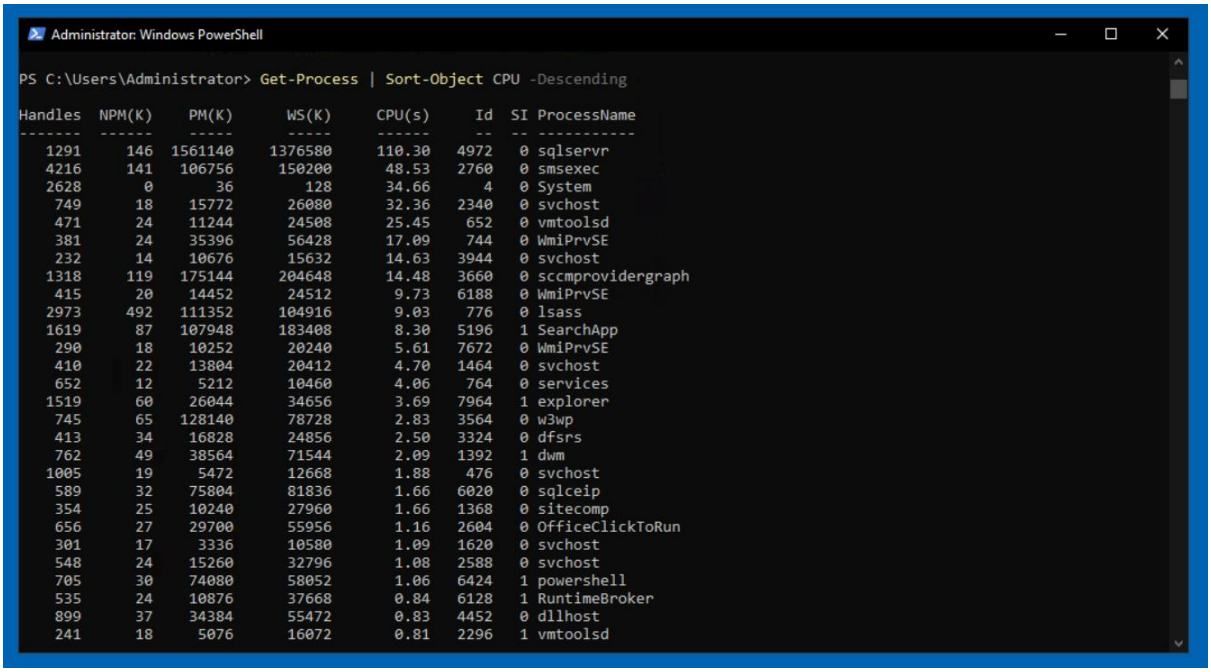
3. Key PowerShell Cmdlets for WMI.

Cmdlet	Purpose
Get-WmiObject *(legacy)*	Retrieves WMI class instances. Example: Get-WmiObject -Class Win32_OperatingSystem
Get-CimInstance *(recommended)*	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.



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

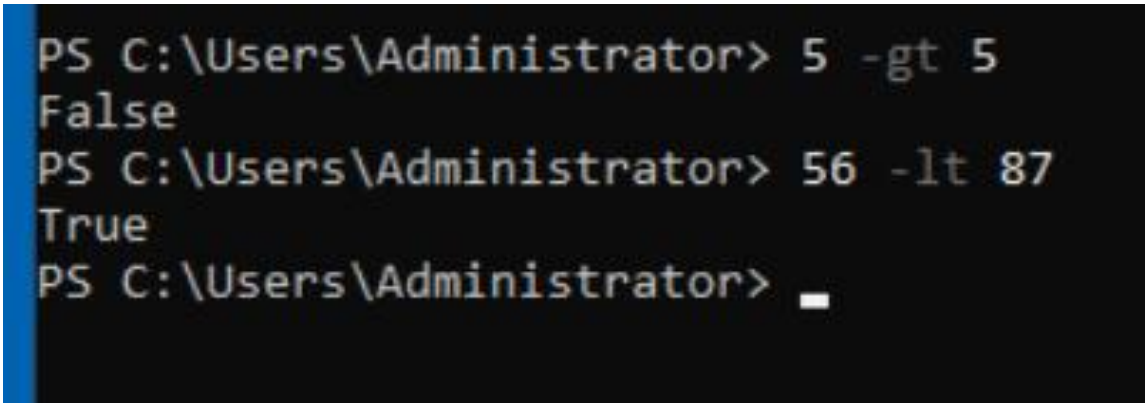
Operators are symbols that perform actions on data.

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)



➢ Input, Output & Formatting.

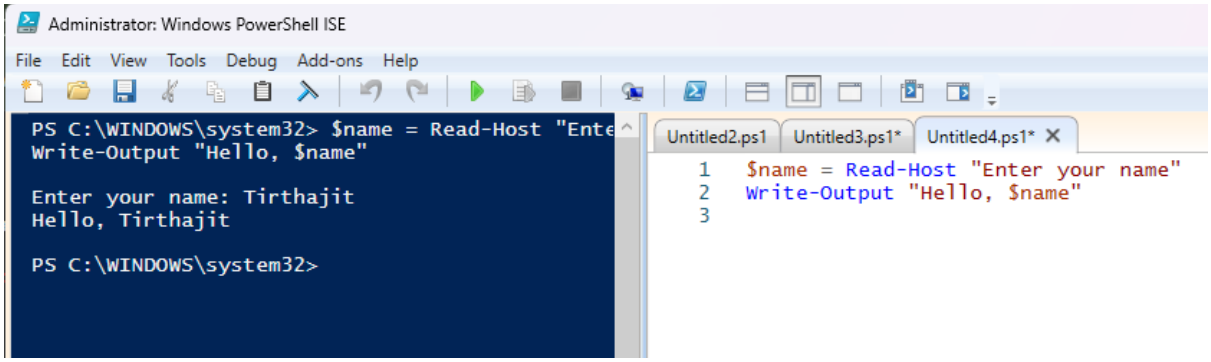
In PowerShell, **input**, **output**, and **formatting** control how data is received from the user or system, how results are returned, and how those results are displayed.

- **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.



    - Use -AsSecureString to hide sensitive input like passwords.

2. **From Command Parameters**
    - Cmdlets and functions accept parameters directly.

```
Get-Process -Name Notepad

# Get-Process → This is the cmdlet. It lists the processes currently running on your system
# -Name → This is a parameter. It tells the cmdlet to filter results by process name.
# notepad → This is the argument (value) you pass to the -Name parameter – in this case,
# you're asking for the process named "notepad".
```

3. **From Pipeline**
    - One command's output becomes another's input.

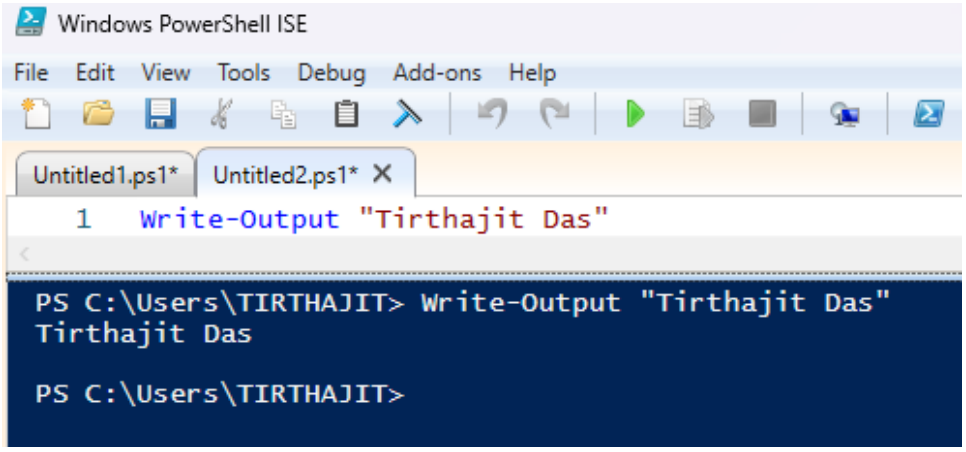
```
Get-Process | Where-Object {$_.CPU -gt 100}

# Step 1: Get-Process outputs all processes.
# Step 2: Each process object is passed one-by-one to Where-Object.
# Step 3: Only those processes with CPU usage greater than 100 seconds are kept.
# Step 4: The filtered list is shown as the final output.
```

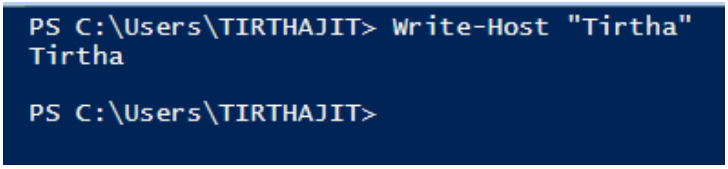
3. Output in PowerShell

a. Output Cmdlets

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

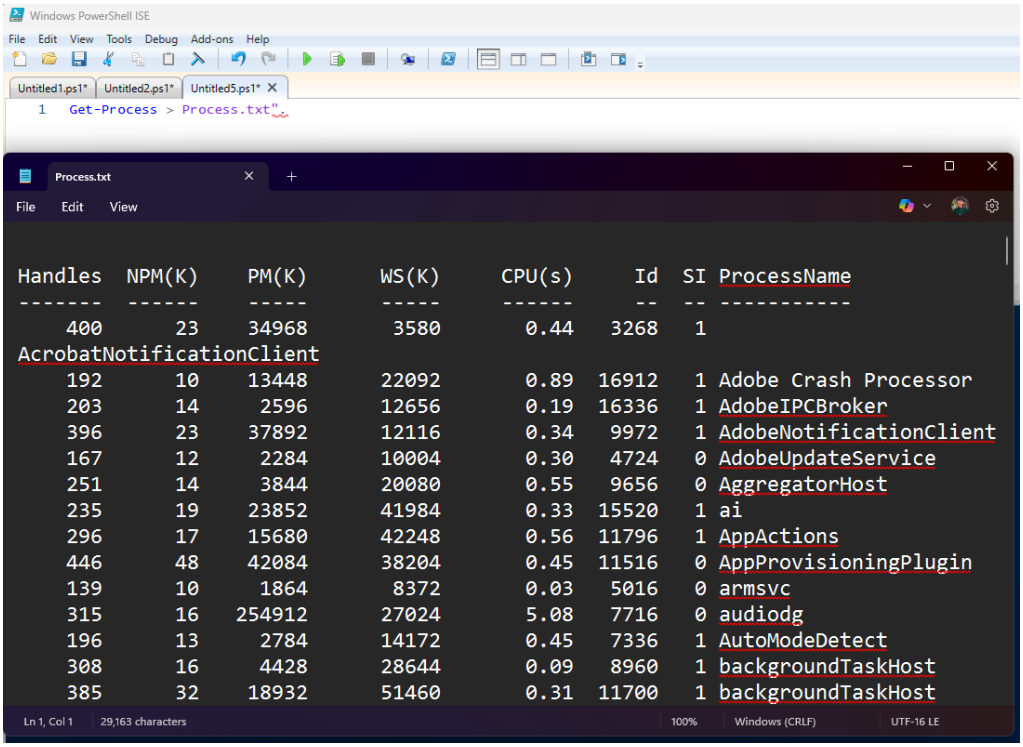


2. **Write-Host** → Displays text directly on the console (doesn't send to pipeline).



b. Sending Output to Files

- **Redirect Operators**:

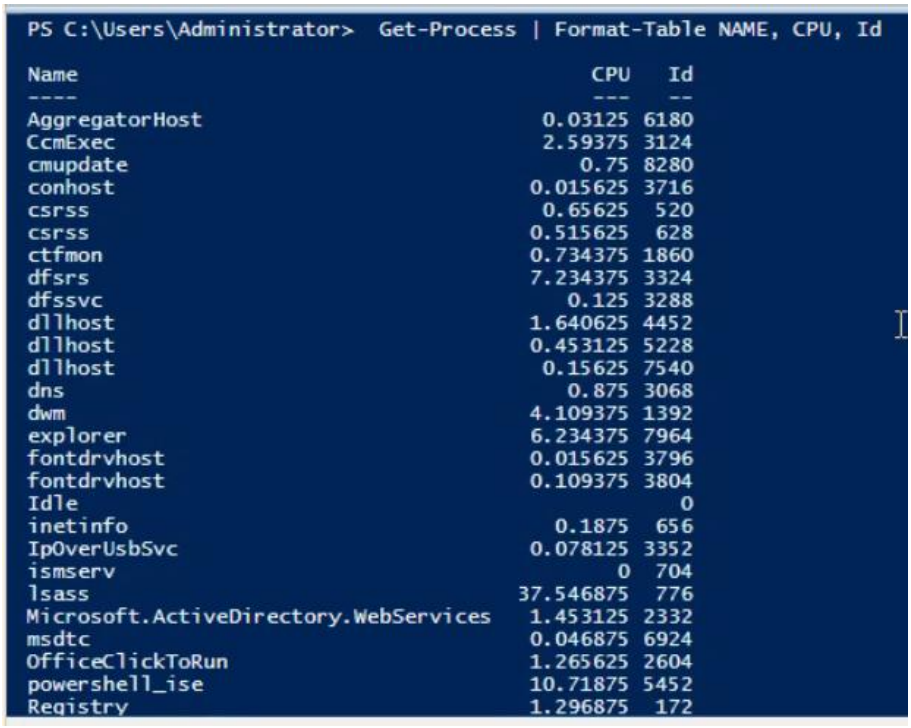


4. Formatting Output

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

a. Common Formatting Cmdlets

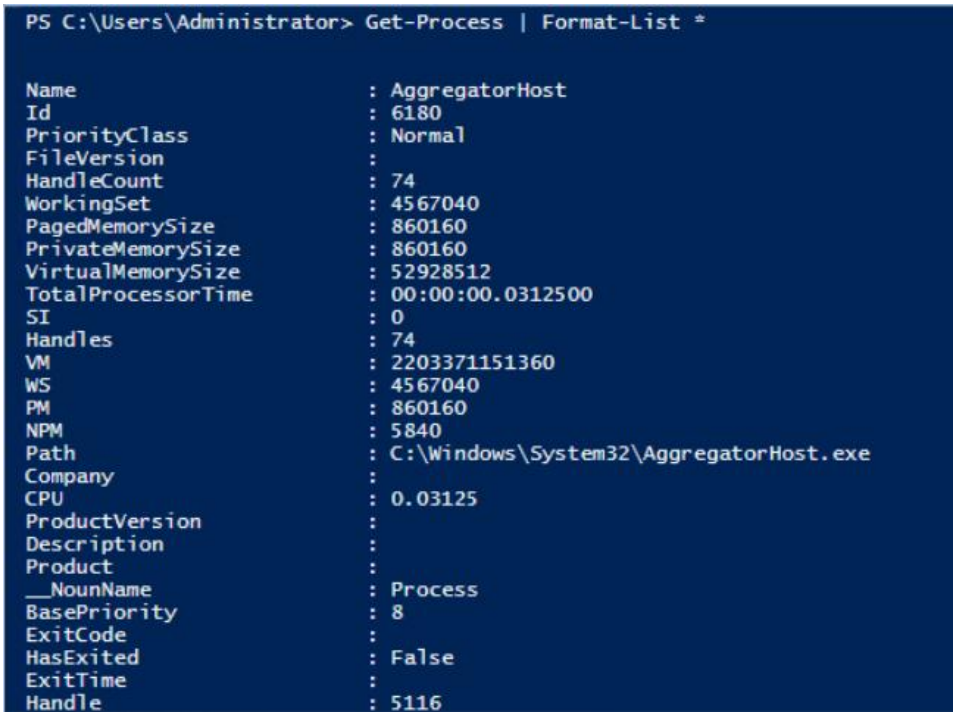
1. **Format-Table (ft)** – Displays data in a table.



```
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
```

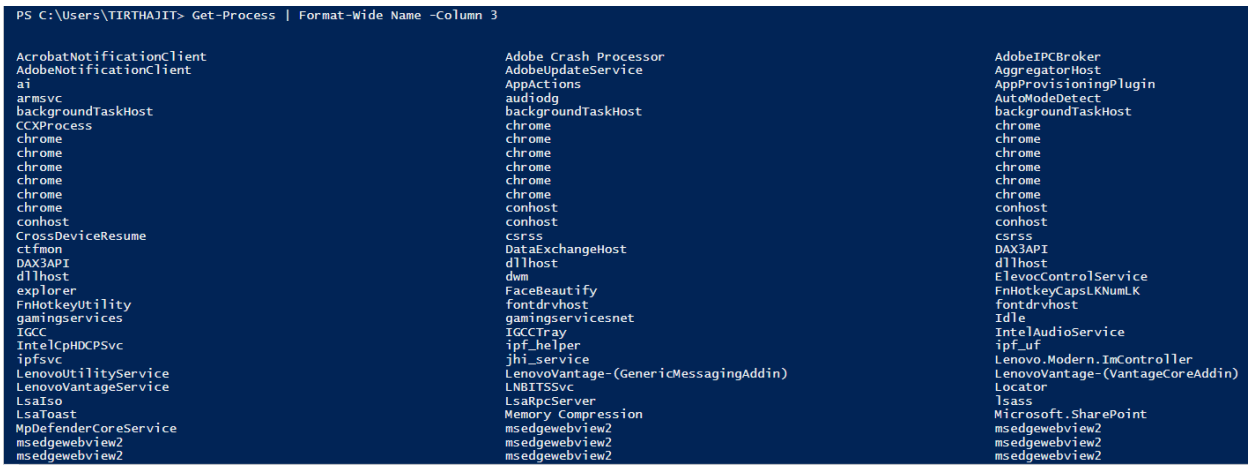
2. **Format-List (fl)** – Shows properties in a vertical list.



```
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.



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*

```powershell
# 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"
```

```
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 Langua
ge
CPU: Win32_Processor: Intel64 Family 6 Model 186 Stepping 2 (DeviceID = "CPU0")

PS C:\Users\TIRTHAJIT>
```