Batch Name: WiproNGA_DWS_B5_25VID2550
First Name: MahammadTanvir
Last Name: Khatri
User ID: 34936
Batch ID: B5-25VID2550
Assignment – 08/08/2025

1.Introducing to Cmdlets

1. **What are Cmdlets?**

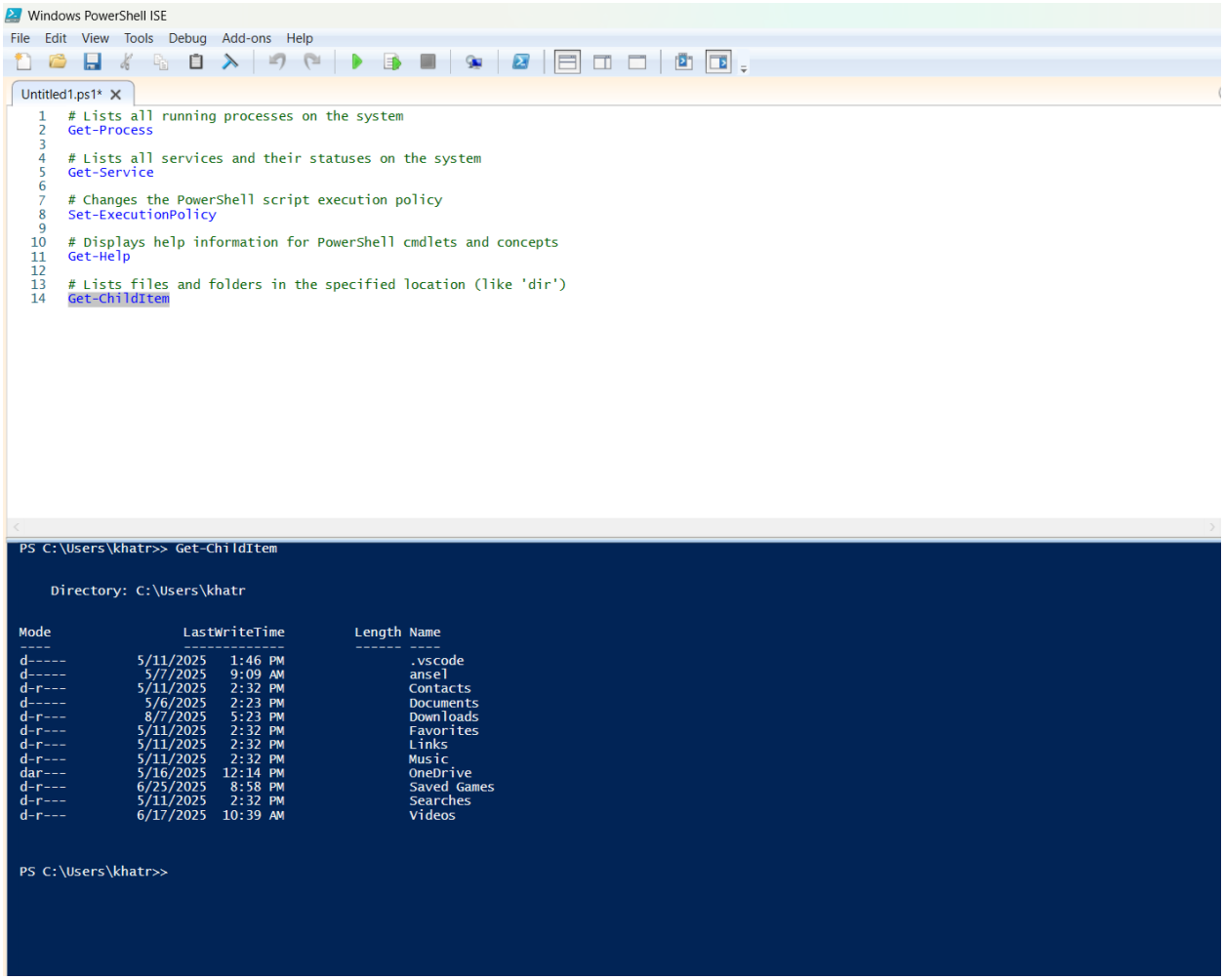
- Definition: Cmdlets (pronounced command-lets) are lightweight, single-function commands built into PowerShell.
- They are specialized .NET classes that perform specific tasks.
- Unlike traditional command-line tools, Cmdlets return objects, not plain text — making it easier to pass data between commands.

2. **Characteristics of Cmdlets**

- Verb-Noun Naming Convention: Example: Get-Process, Set-Date, New-Item.
- Consistent Syntax: 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.

3. **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.



2.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**

3. First command runs and produces objects.

4. These objects are streamed one by one into the next command.

5. The next command processes each object and passes the result to the next stage.

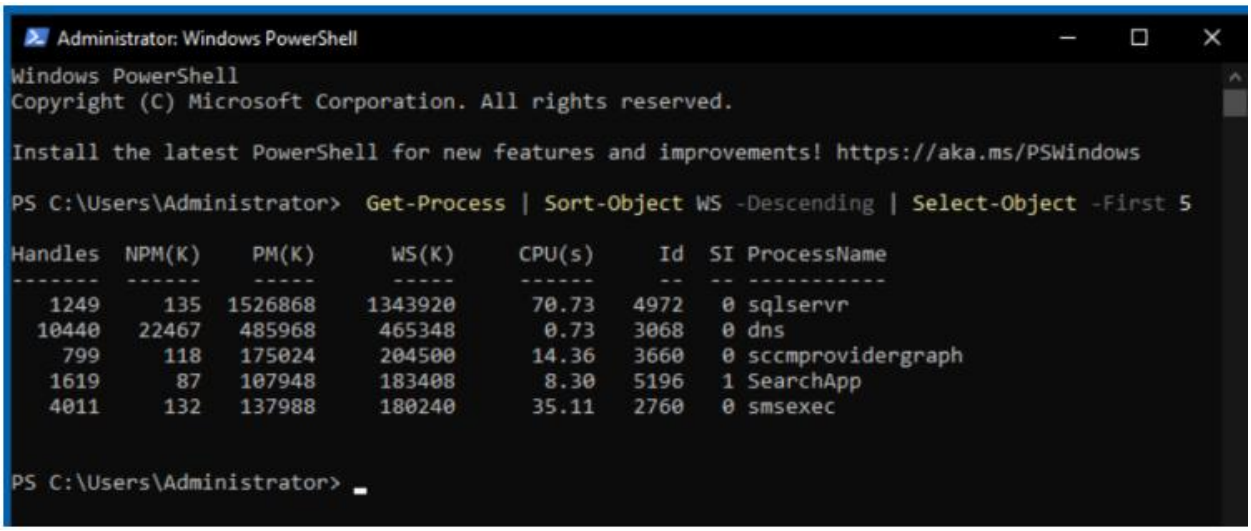
Example:

powershell

CopyEdit

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.



3.Key Cmdlets

- Cmdlets are lightweight PowerShell commands built into the shell or added via modules.
- 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.

Untitled1.ps1* | Untitled2.ps1* | Untitled3.ps1* | Untitled4.ps1* | Untitled5.ps1* | Untitled6.ps1*

```
Get-Command
 # Lists all available cmdlets, functions, and scripts
Get-Help
# Shows help and usage details for a cmdlet
Get-Process
# Displays running processes
Stop-Process
 # Terminates a running process
Set-ExecutionPolicy
 # Changes script execution permissions
Get-Service
# Shows the status of system services
Start-Service
# Starts a stopped service
Stop-Service
# Stops a running service
Restart-Service
# Restarts a service
Get-EventLog
# Retrieves event log entries
Get-WmiObject
# Gets management info using WMI (legacy)
Get-CimInstance
# Gets management info using CIM (modern)
Invoke-WmiMethod
# Executes a method of a WMI object
Set-WmiInstance
# Creates or updates a WMI object
Remove-WmiObject
# Deletes a WMI object
Get-ChildItem
# Lists files and folders (like dir)
Copy-Item
# Copies files or folders
Move-Item
# Moves files or folders
Remove-Item
# Deletes files or folders
Out-File
# Sends output to a text file
```

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**

- Get-WmiObject (legacy) → Retrieves WMI class instances. Example:

powershell

CopyEdit

```
Get-WmiObject -Class Win32_OperatingSystem
```

- Get-CimInstance (recommended) → Modern alternative to Get-WmiObject; uses WS-Man protocol.
- Invoke-WmiMethod → Executes a method of a WMI object.
- Set-WmiInstance → Modifies a WMI object instance.
- Remove-WmiObject → Deletes a WMI object instance.

4.Pipeline Filtering & Operators

1. **Pipeline in PowerShell**

- Passes output of one command to another.
- Example:

powershell

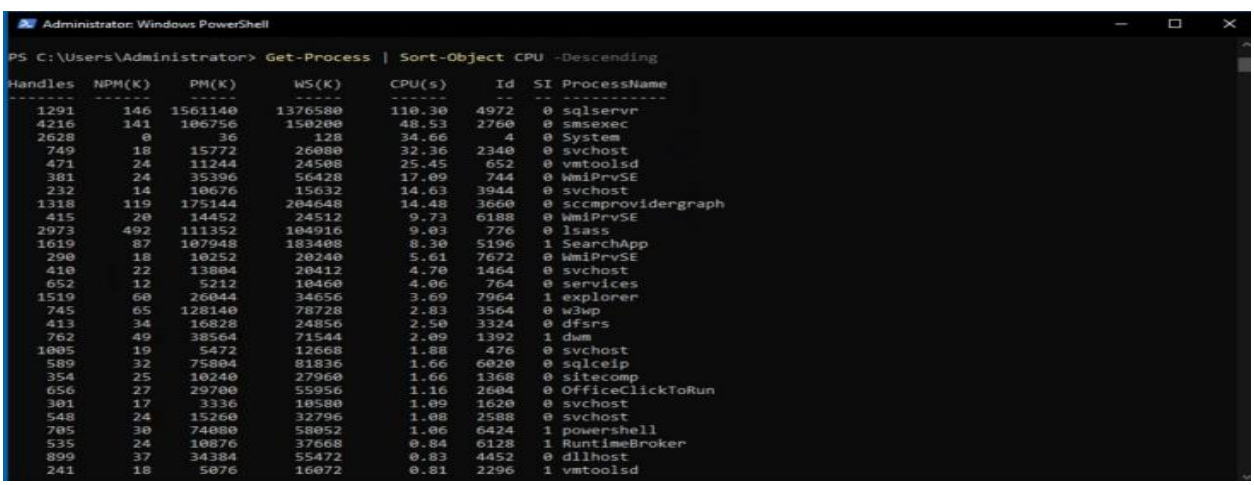
CopyEdit

Get-Process | Sort-Object CPU -Descending

  - Get-Process → lists processes.
  - Sort-Object → sorts them.

Key Points:

- Allows chaining commands.
- Reduces need for temporary variables.
- Processes data one object at a time.



```
Administrator: Windows PowerShell
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 sccmprovidergraph
    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
```

2. **Filtering in Pipeline**

- Where-Object – Filters based on a condition.
- Select-Object – Picks properties or limits results.
- Sort-Object – Sorts results by properties.

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)

```
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\khatr> 15 -gt 13
True
PS C:\Users\khatr> 30 -lt 16
False
PS C:\Users\khatr>
```

5.Input, Output & Formatting

1. **Input:** From keyboard (Read-Host), from parameters, or from pipeline.

a. From Keyboard (Read-Host)

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

```
PS C:\Users\khatr> $name = Read-Host "Please enter your name"
    Write-Host "Hello, $name!"
Please enter your name: Tanvir
Hello, Tanvir!

PS C:\Users\khatr>
```

Untitled6.ps1 | Untitled4.ps1 | Untitled5.ps1 | Untitled2.ps1 | Untitled3.ps1

```
1  $name = Read-Host "Please enter your name"
2      Write-Host "Hello, $name!"
```

b. From Parameters

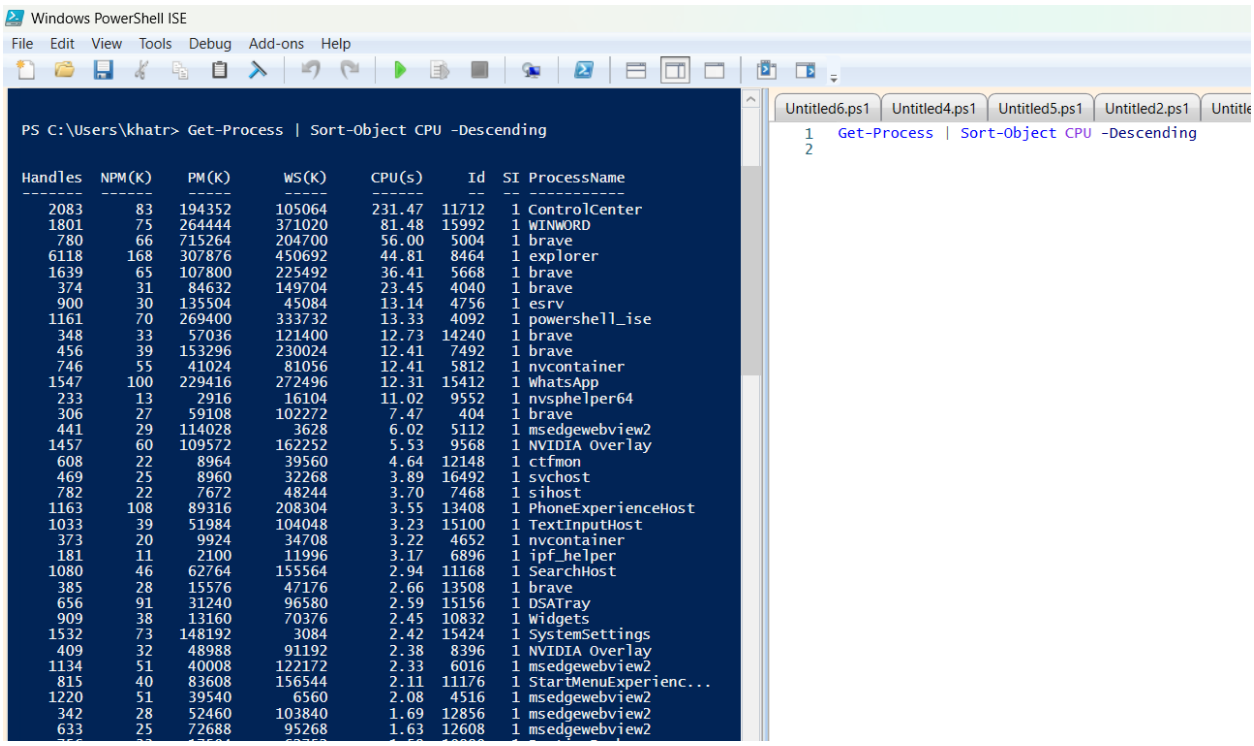
- Cmdlets and functions accept parameters directly

Untitled6.ps1 | Untitled4.ps1 | Untitled5.ps1 | Untitled2.ps1 | Untitled3.ps1 | Untitled1.ps1 | Untitled9.ps1*

```
1  Get-Process -Name Notepad
2
3  # Get-Process -> This cmdlet retrieves a list of processes that are currently active on your system.
4  # -Name -> This parameter specifies that results should be filtered based on the process name.
5  # Notepad -> This is the value provided to the -Name parameter. In this example,
6  # it requests details for the process named "notepad".
7
```

C. From pipeline

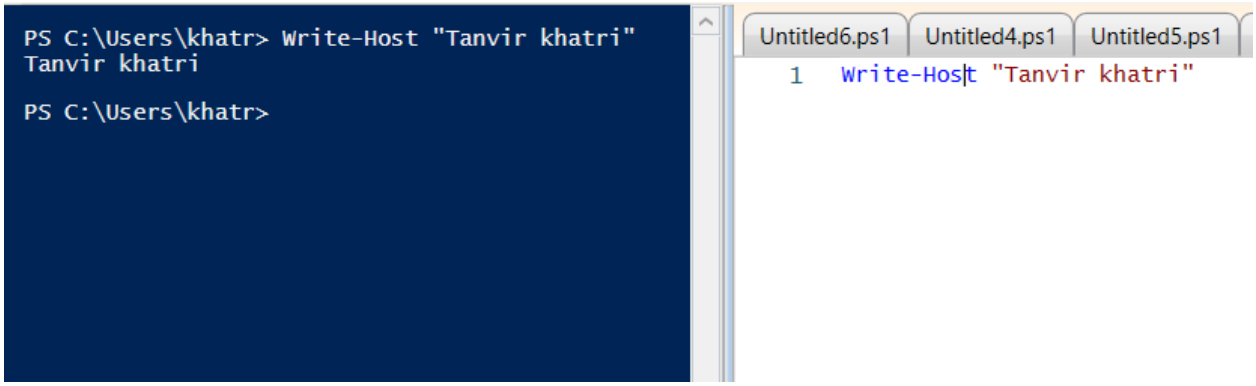
- One command's output becomes another's input.



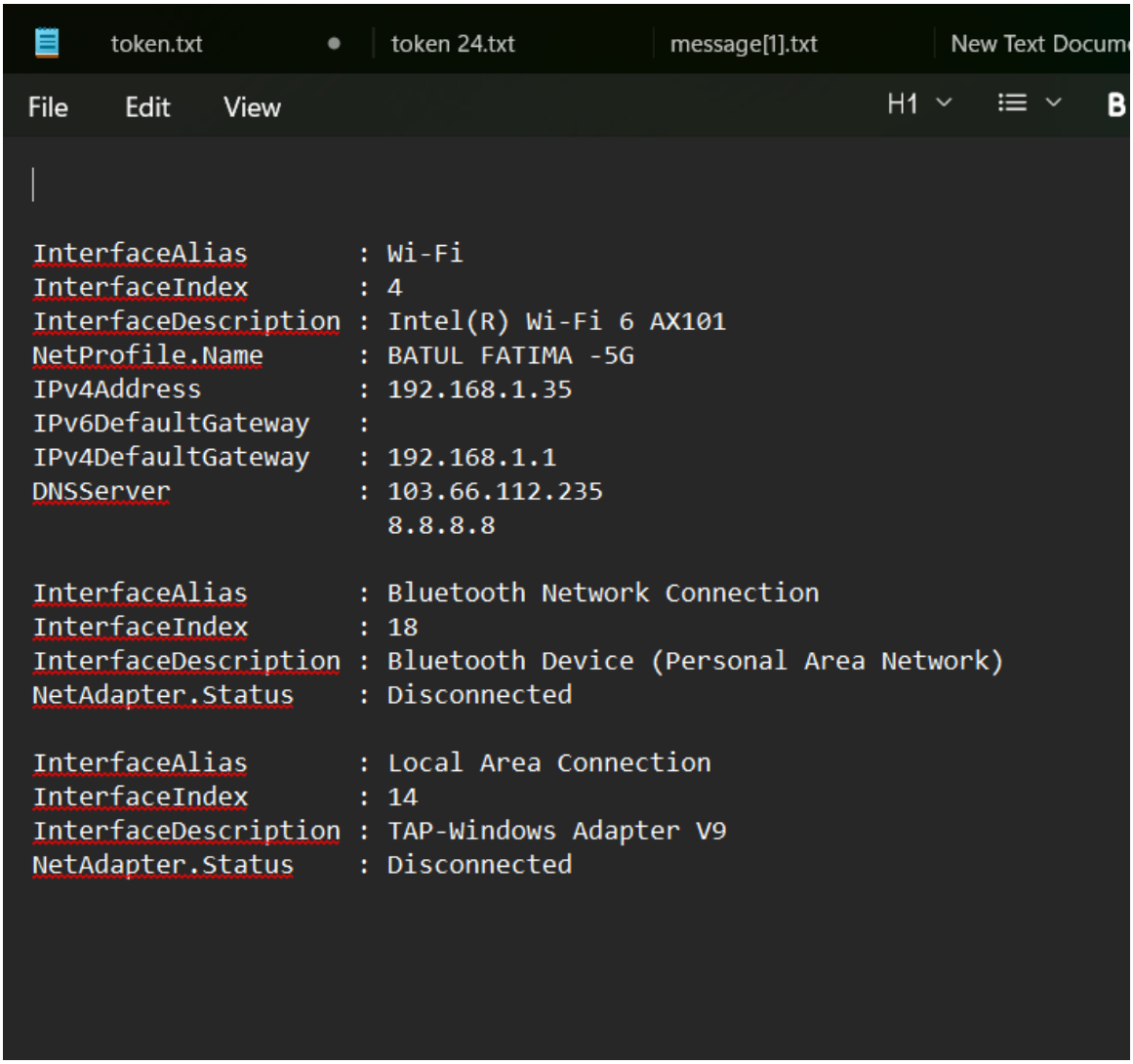
2.Output:

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

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

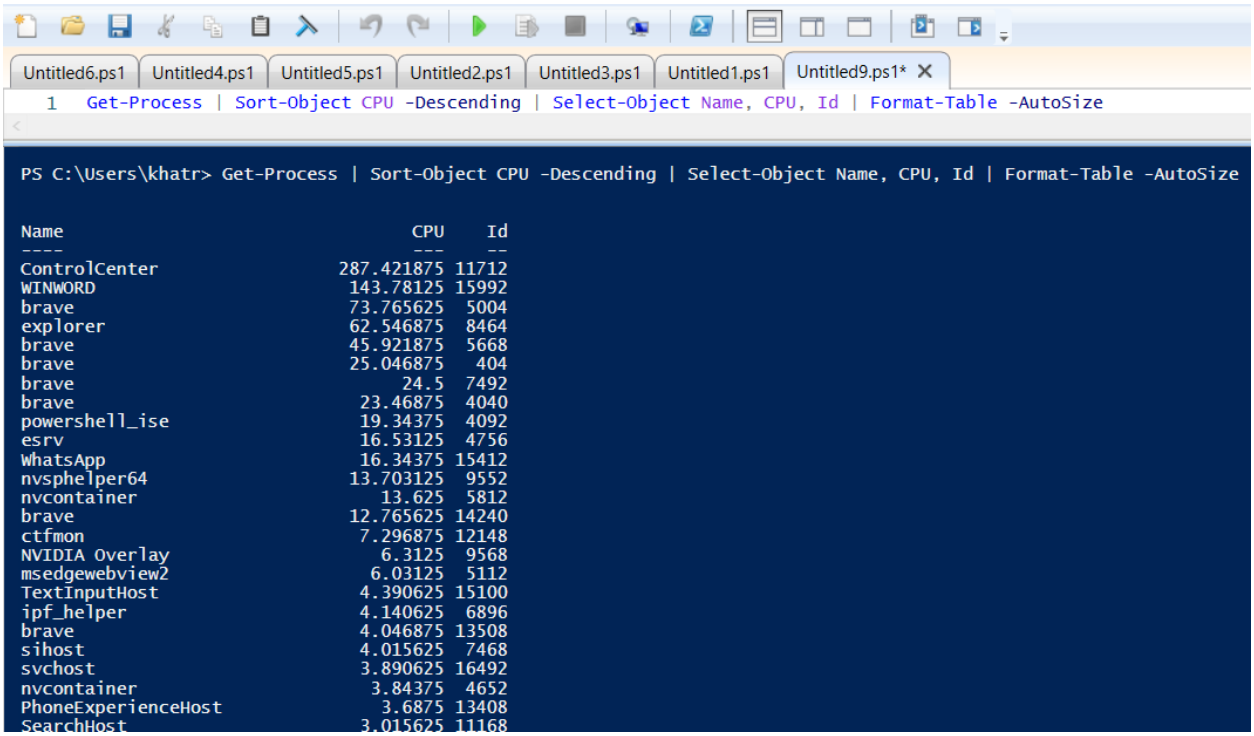


- **Sending Output to Files**

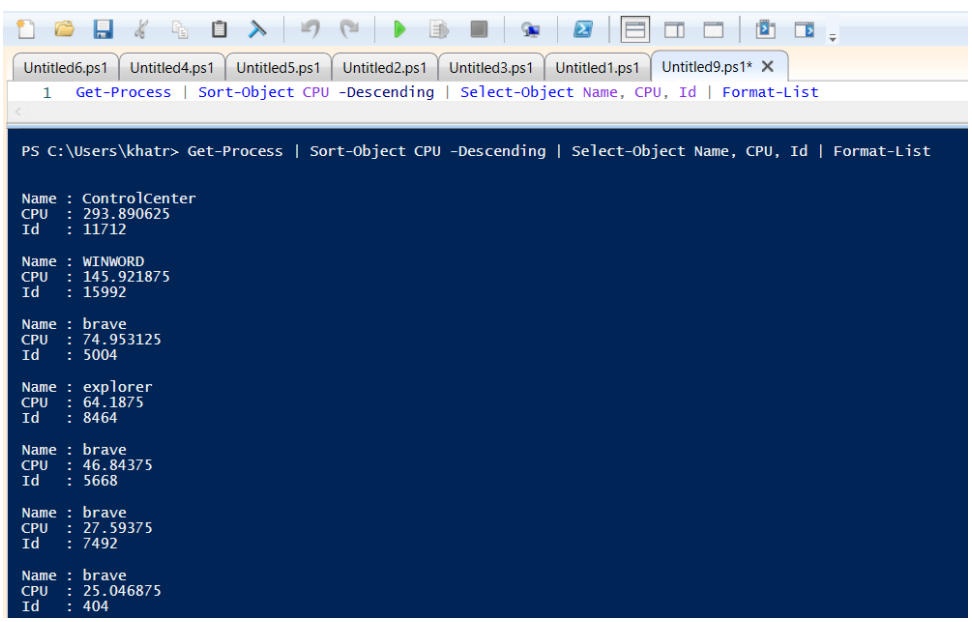


2. Formatting:

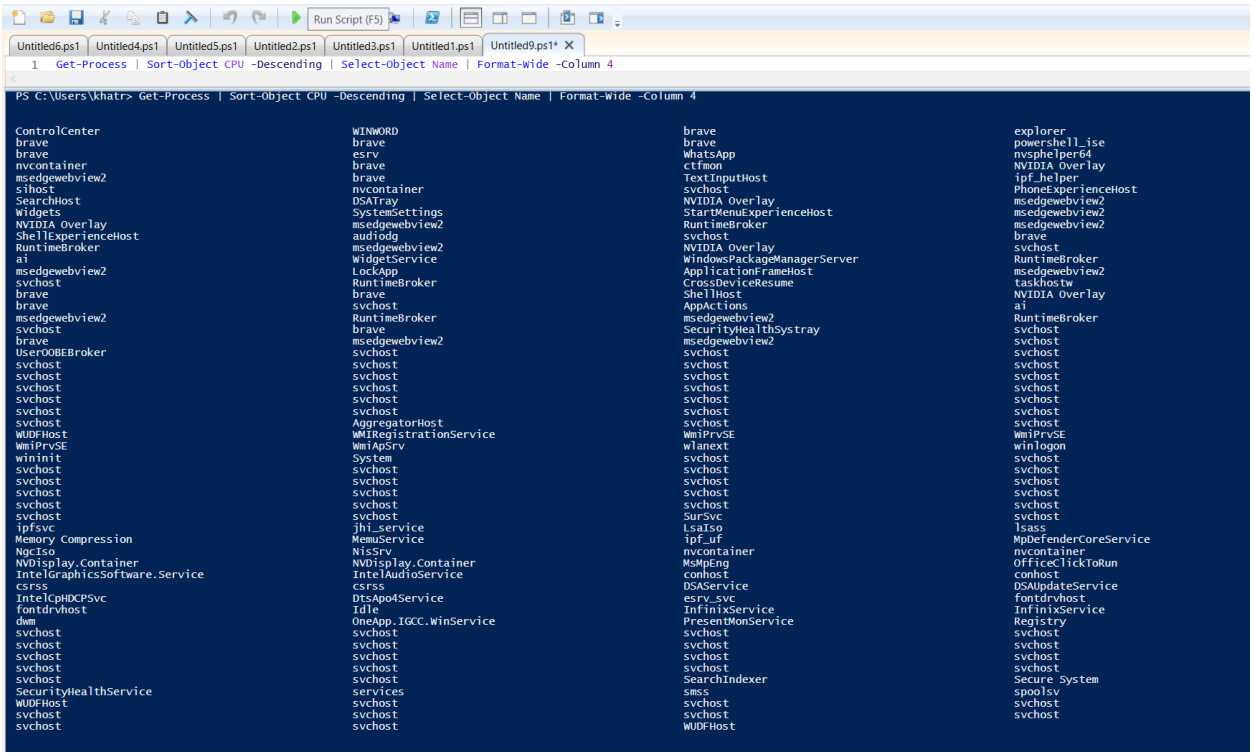
- Format-Table – Table output.



- Format-List – List output.



- Format-Wide – Single property per line.



6. Scripting Overview

1. **What is a PowerShell Script?**

- A script is a collection of PowerShell commands saved in a .ps1 file.

2. **Benefits:**

- Automation
- Consistency
- Time-saving
- Scalability
- Integration with Windows, WMI, .NET, APIs