

**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.

The screenshot shows the Windows PowerShell ISE interface. The top pane contains a script with the following commands:

```
1 # Lists all running processes on the system
2 Get-Process
3
4 # Lists all services and their statuses on the system
5 Get-Service
6
7 # Changes the PowerShell script execution policy
8 Set-ExecutionPolicy
9
10 # Displays help information for PowerShell cmdlets and concepts
11 Get-Help
12
13 # Lists files and folders in the specified location (like 'dir')
14 Get-Childitem
```

The bottom pane shows the output of the `Get-Childitem` command, displaying a directory listing for `C:\Users\khatr`:

```
PS C:\Users\khatr>> Get-Childitem

Directory: C:\Users\khatr

Mode                LastWriteTime         Length Name
----                -
d-----          5/11/2025   1:46 PM             .vscode
d-----          5/7/2025    9:09 AM             anse1
d-r-----        5/11/2025   2:32 PM           Contacts
d-----          5/6/2025   2:23 PM           Documents
d-r-----        8/7/2025    5:23 PM           Downloads
d-r-----        5/11/2025   2:32 PM           Favorites
d-r-----        5/11/2025   2:32 PM             Links
d-r-----        5/11/2025   2:32 PM             Music
d-ar-----       5/16/2025  12:14 PM          OneDrive
d-r-----        6/25/2025    8:58 PM       Saved Games
d-r-----        5/11/2025   2:32 PM           Searches
d-r-----        6/17/2025  10:39 AM           Videos

PS C:\Users\khatr>>
```

---

## 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.

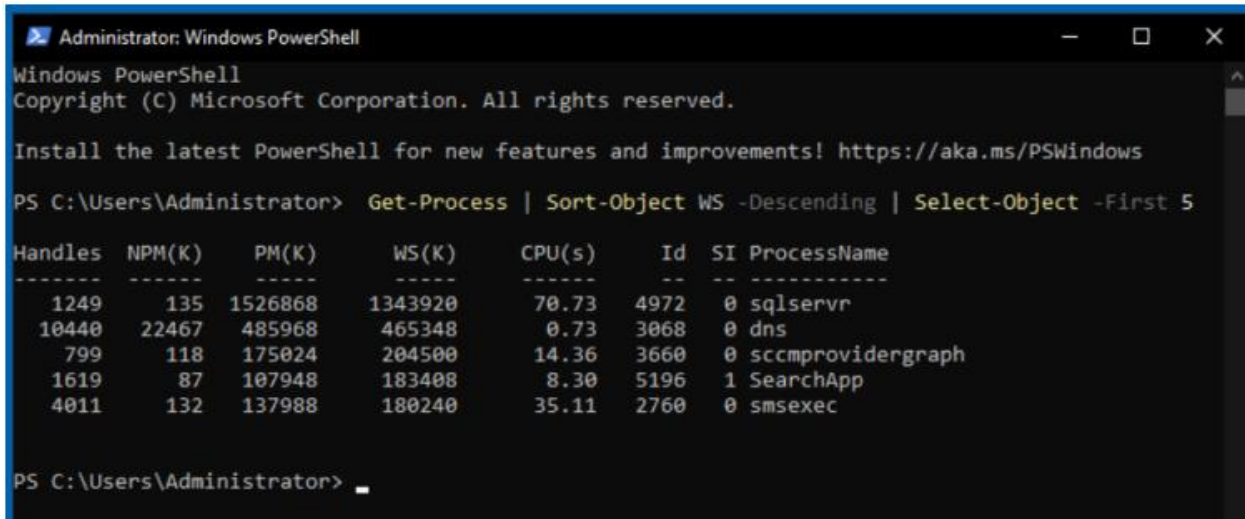
### Example:

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.

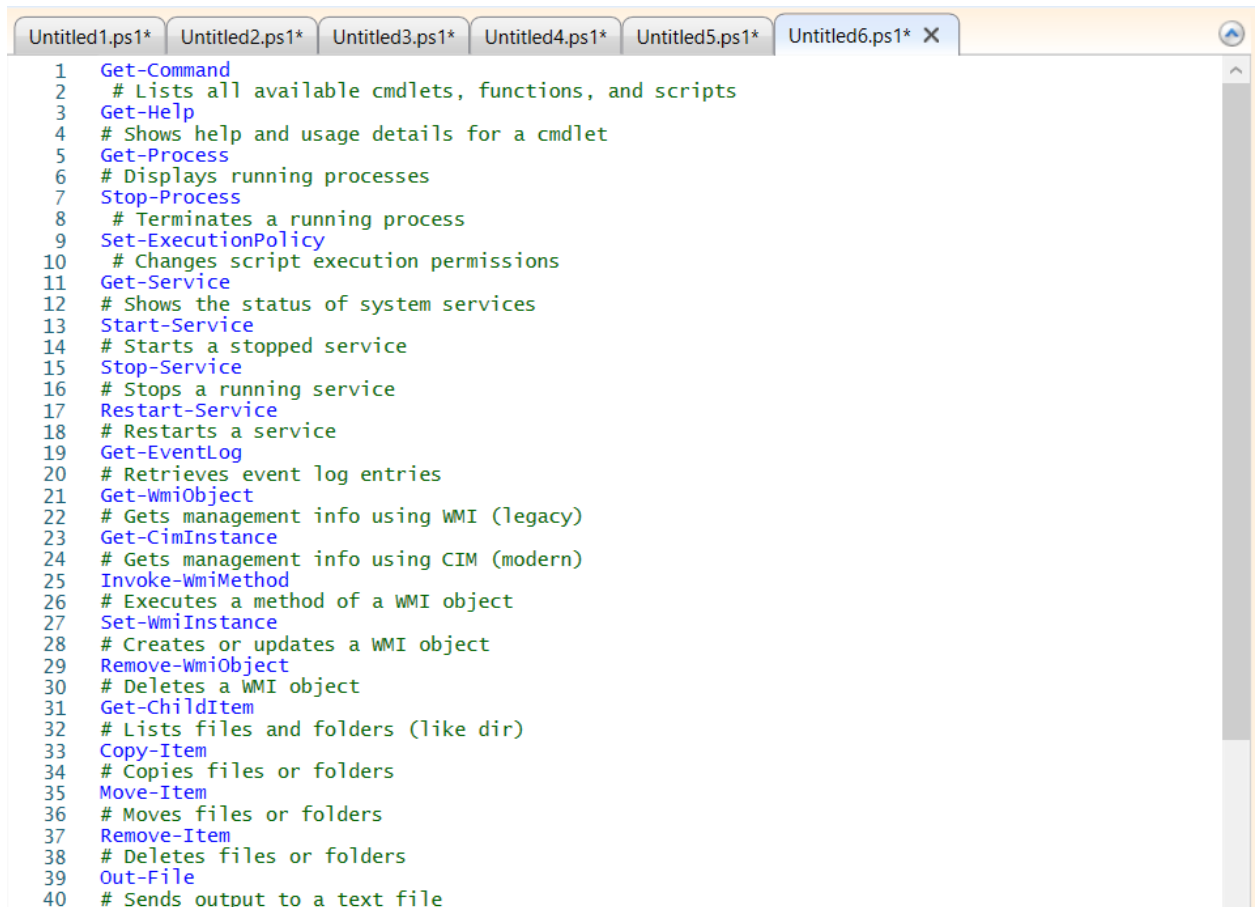


The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command entered is `Get-Process | Sort-Object WS -Descending | Select-Object -First 5`. The output is a table of process information, sorted by Working Set (WS) in descending order.

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	sccmprovidengraph
1619	87	107948	183408	8.30	5196	1	SearchApp
4011	132	137988	180240	35.11	2760	0	smsexec

### 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.



```
1  Get-Command
2  # Lists all available cmdlets, functions, and scripts
3  Get-Help
4  # Shows help and usage details for a cmdlet
5  Get-Process
6  # Displays running processes
7  Stop-Process
8  # Terminates a running process
9  Set-ExecutionPolicy
10 # Changes script execution permissions
11 Get-Service
12 # Shows the status of system services
13 Start-Service
14 # Starts a stopped service
15 Stop-Service
16 # Stops a running service
17 Restart-Service
18 # Restarts a service
19 Get-EventLog
20 # Retrieves event log entries
21 Get-WmiObject
22 # Gets management info using WMI (legacy)
23 Get-CimInstance
24 # Gets management info using CIM (modern)
25 Invoke-WmiMethod
26 # Executes a method of a WMI object
27 Set-WmiInstance
28 # Creates or updates a WMI object
29 Remove-WmiObject
30 # Deletes a WMI object
31 Get-ChildItem
32 # Lists files and folders (like dir)
33 Copy-Item
34 # Copies files or folders
35 Move-Item
36 # Moves files or folders
37 Remove-Item
38 # Deletes files or folders
39 Out-File
40 # 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

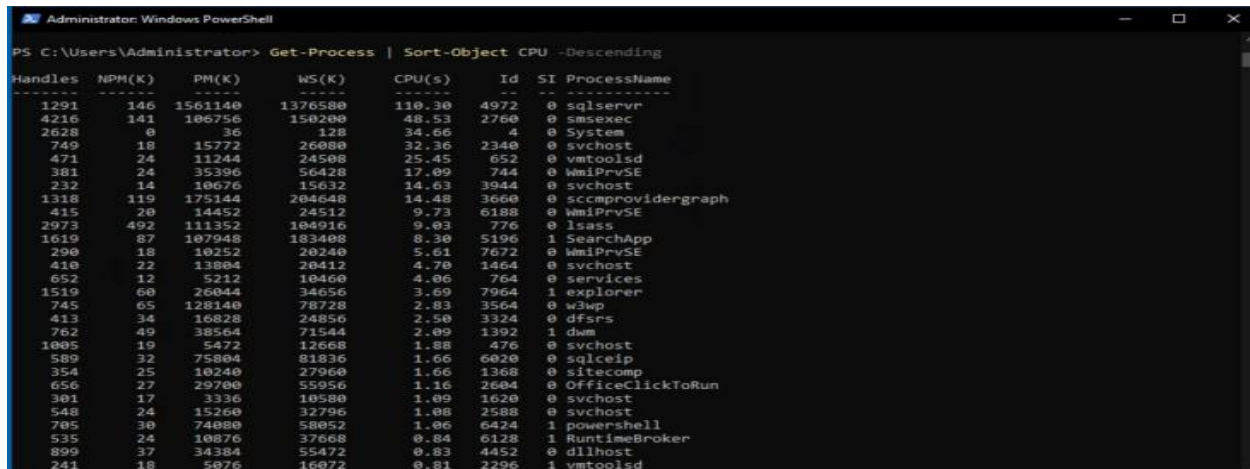
CopyEdit

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 dm
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
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>
```

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

### b. From Parameters

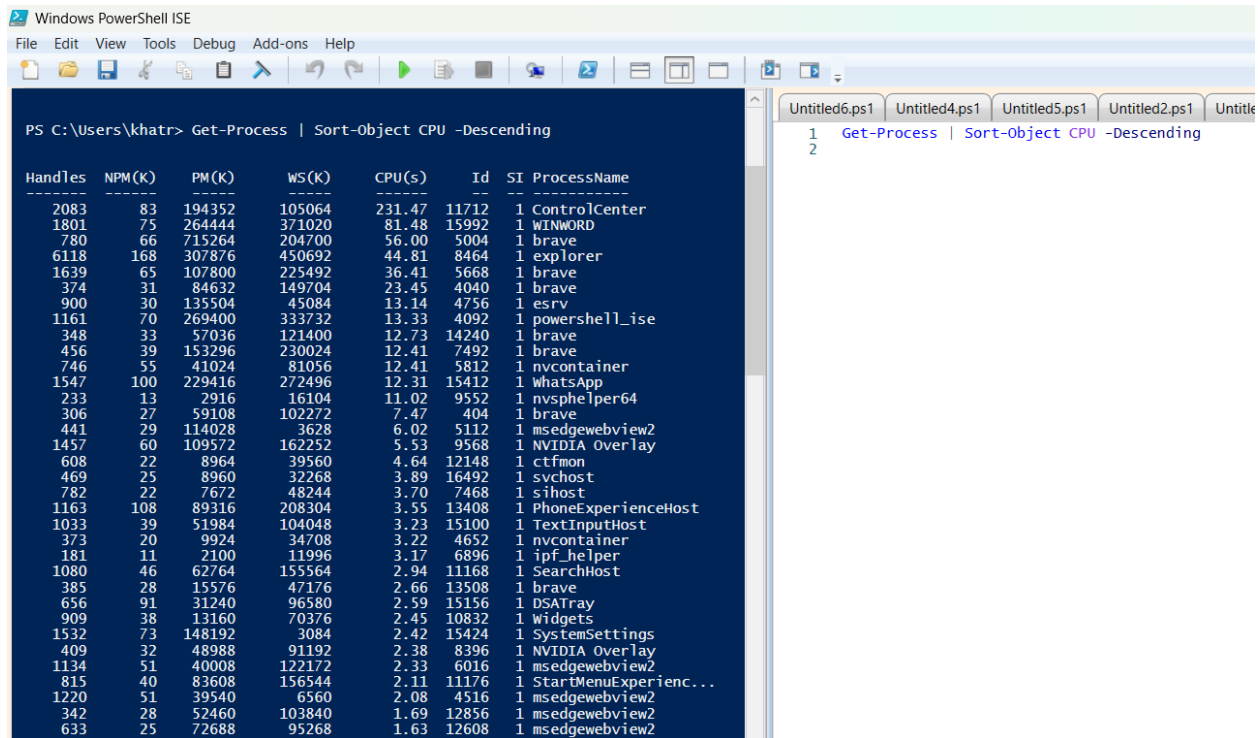
- Cmdlets and functions accept parameters directly

```
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.

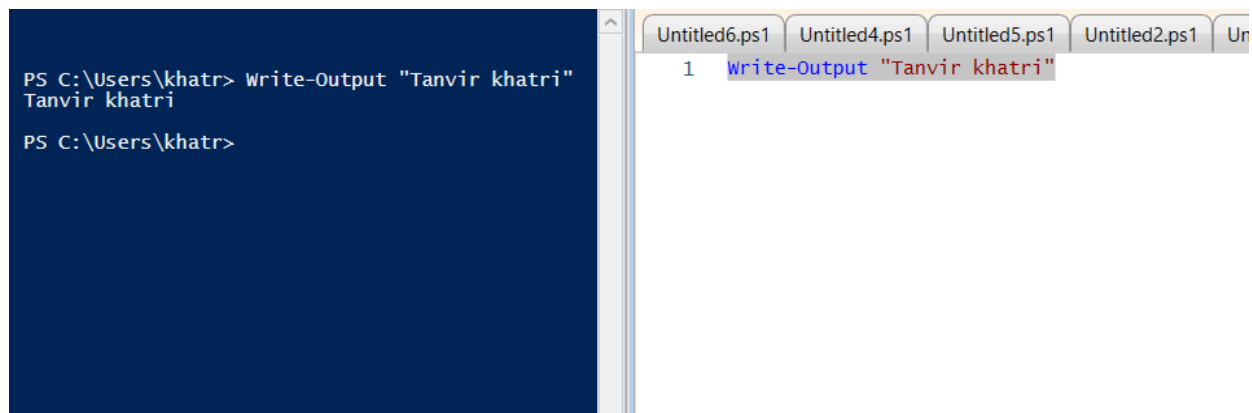


The screenshot shows the Windows PowerShell ISE interface. The command prompt shows the command `Get-Process | Sort-Object CPU -Descending` being executed. The output is a table of process information, sorted by CPU usage in descending order. The table has columns: Handles, NPM(K), PM(K), WS(K), CPU(s), Id, SI, and ProcessName. The processes listed include ControlCenter, WINWORD, brave, explorer, esrv, powershell\_ise, nvcontainer, whatsapp, nvsphelper64, msedgeview2, NVIDIA Overlay, ctfmon, svchost, slhost, PhoneExperienceHost, TextInputHost, ipf\_helper, SearchHost, DSATray, widgets, SystemSettings, NVIDIA Overlay, StartMenuExperienc..., msedgeview2, and msedgeview2.

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
2083	83	194352	105064	231.47	11712	1	ControlCenter
1801	75	264444	371020	81.48	15992	1	WINWORD
780	66	715264	204700	56.00	5004	1	brave
6118	168	307876	450692	44.81	8464	1	explorer
1639	65	107800	225492	36.41	5668	1	brave
374	31	84632	149704	23.45	4040	1	brave
900	30	135504	45084	13.14	4756	1	esrv
1161	70	269400	333732	13.33	4092	1	powershell_ise
348	33	57036	121400	12.73	14240	1	brave
456	39	153296	230024	12.41	7492	1	brave
746	55	41024	81056	12.41	5812	1	nvcontainer
1547	100	229416	272496	12.31	15412	1	whatsapp
233	13	2916	16104	11.02	9552	1	nvsphelper64
306	27	59108	102272	7.47	404	1	brave
441	29	114028	3628	6.02	5112	1	msedgeview2
1457	60	109572	162252	5.53	9568	1	NVIDIA Overlay
608	22	8964	39560	4.64	12148	1	ctfmon
469	25	8960	32268	3.89	16492	1	svchost
782	22	7672	48244	3.70	7468	1	slhost
1163	108	89316	208304	3.55	13408	1	PhoneExperienceHost
1033	39	51984	104048	3.23	15100	1	TextInputHost
373	20	9924	34708	3.22	4652	1	nvcontainer
181	11	2100	11996	3.17	6896	1	ipf_helper
1080	46	62764	155564	2.94	11168	1	SearchHost
385	28	15576	47176	2.66	13508	1	brave
656	91	31240	96580	2.59	15156	1	DSATray
909	38	13160	70376	2.45	10832	1	widgets
1532	73	148192	3084	2.42	15424	1	SystemSettings
409	32	48988	91192	2.38	8396	1	NVIDIA Overlay
1134	51	40008	122172	2.33	6016	1	msedgeview2
815	40	83608	156544	2.11	11176	1	StartMenuExperienc...
1220	51	39540	6560	2.08	4516	1	msedgeview2
342	28	52460	103840	1.69	12856	1	msedgeview2
633	25	72688	95268	1.63	12608	1	msedgeview2

## 2. Output:

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

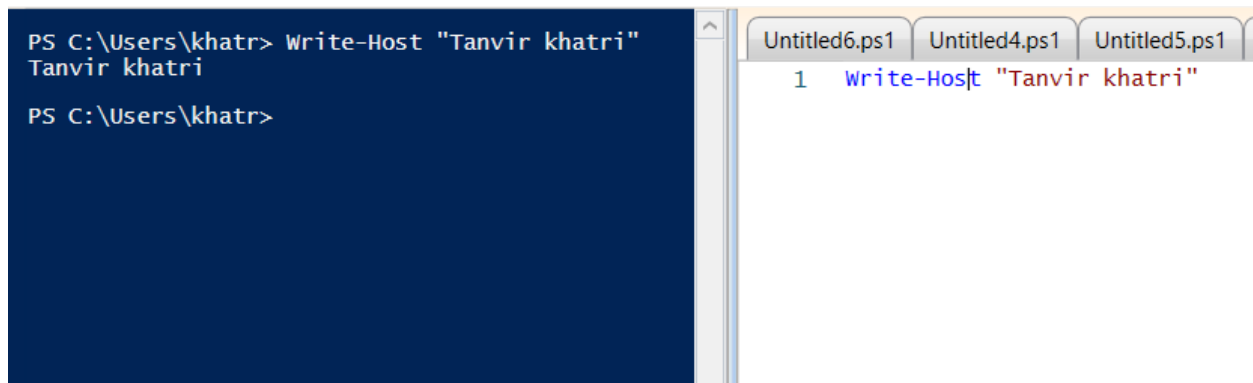


The screenshot shows the Windows PowerShell ISE interface. The command prompt shows the command `Write-Output "Tanvir khatri"` being executed. The output is the string "Tanvir khatri".

```
PS C:\Users\khatr> Write-Output "Tanvir khatri"
Tanvir khatri

PS C:\Users\khatr>
```

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

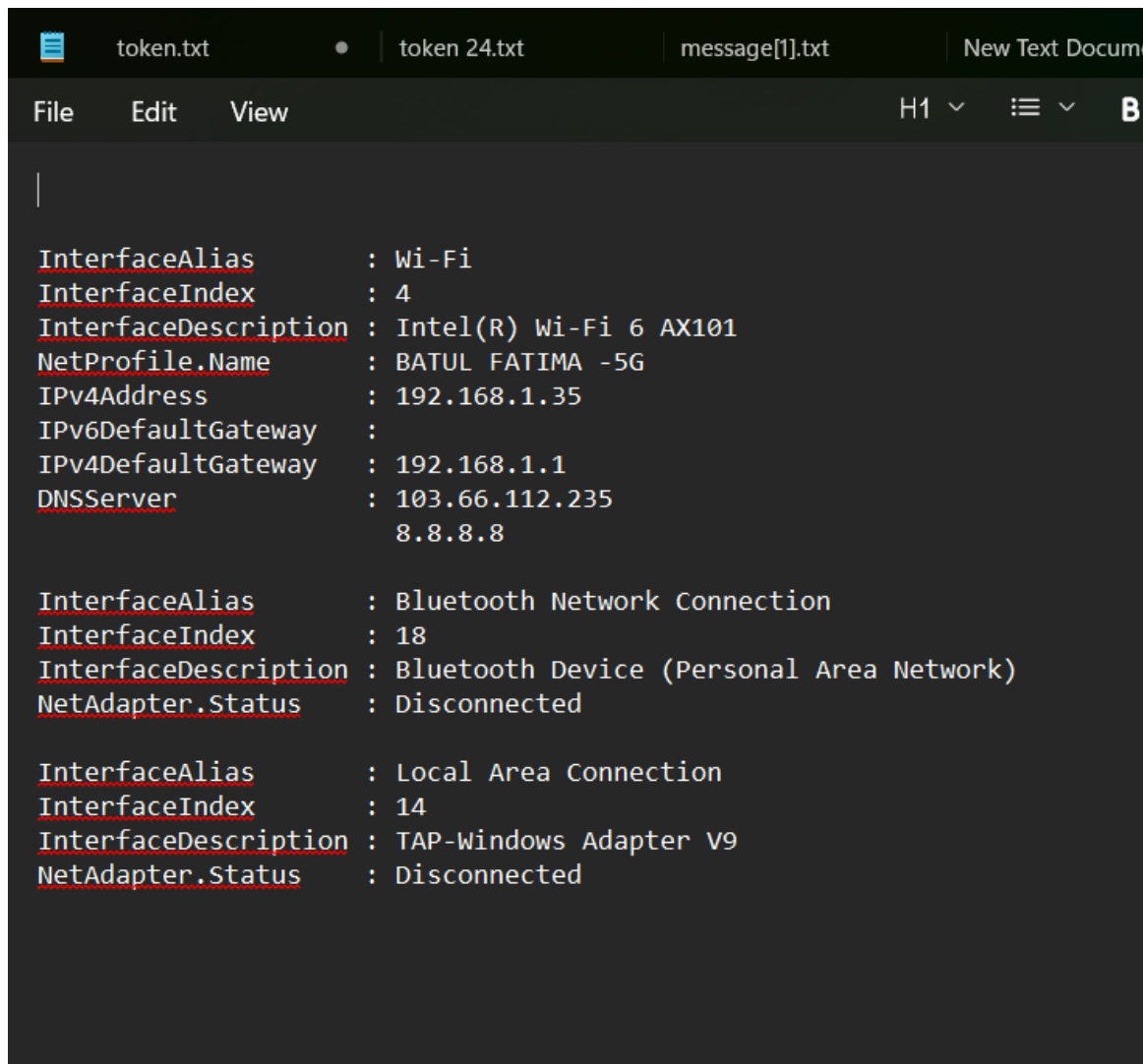


The image shows a PowerShell console window on the left and a VS Code editor window on the right. The PowerShell console displays the command `Write-Host "Tanvir khatri"` and its output `Tanvir khatri`. The VS Code editor shows a file named `Untitled6.ps1` with the same command on line 1.

```
PS C:\Users\khatr> Write-Host "Tanvir khatri"
Tanvir khatri
PS C:\Users\khatr>
```

```
1 Write-Host "Tanvir khatri"
```

- **Sending Output to Files**



The image shows a text editor window with a dark theme. The editor has tabs for `token.txt`, `token 24.txt`, `message[1].txt`, and `New Text Document`. The main text area contains network configuration details for three interfaces: Wi-Fi, Bluetooth Network Connection, and Local Area Connection. Each interface's properties are listed on a new line, with the property name underlined.

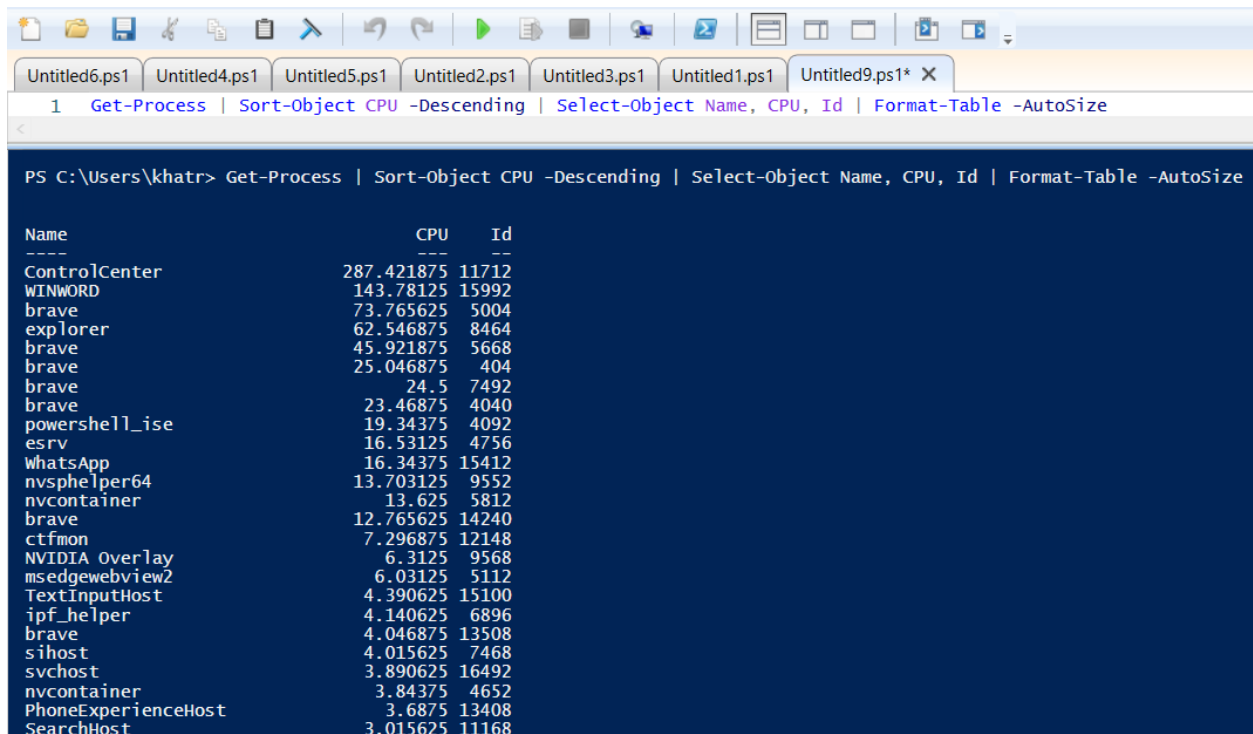
```
InterfaceAlias      : Wi-Fi
InterfaceIndex      : 4
InterfaceDescription : Intel(R) Wi-Fi 6 AX101
NetProfile.Name     : BATUL FATIMA -5G
IPv4Address         : 192.168.1.35
IPv6DefaultGateway  :
IPv4DefaultGateway  : 192.168.1.1
DNSServer           : 103.66.112.235
                   : 8.8.8.8

InterfaceAlias      : Bluetooth Network Connection
InterfaceIndex      : 18
InterfaceDescription : Bluetooth Device (Personal Area Network)
NetAdapter.Status   : Disconnected

InterfaceAlias      : Local Area Connection
InterfaceIndex      : 14
InterfaceDescription : TAP-Windows Adapter V9
NetAdapter.Status   : Disconnected
```

## 2. Formatting:

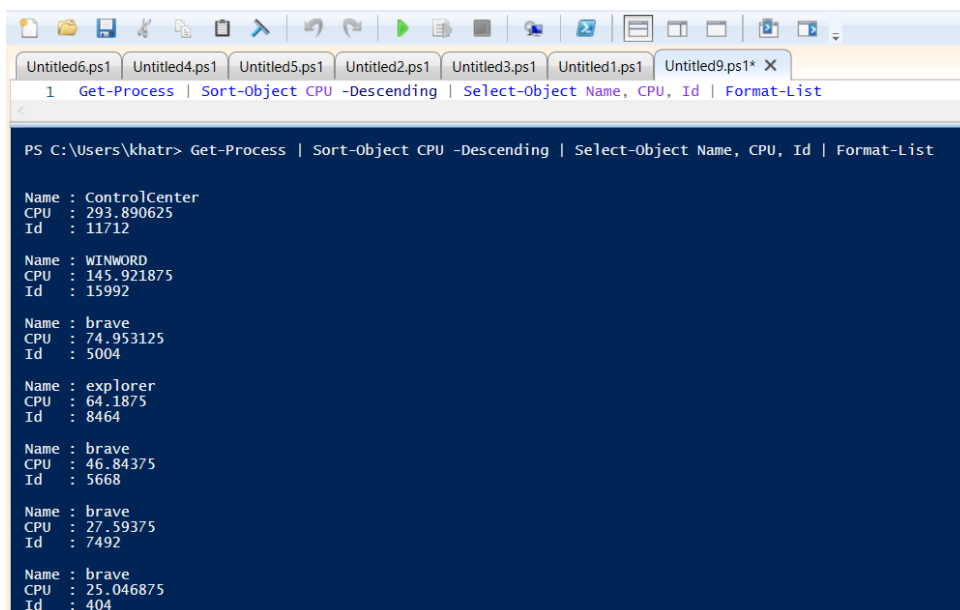
- Format-Table – Table output.



```
PS C:\Users\khatr> Get-Process | Sort-Object CPU -Descending | Select-Object Name, CPU, Id | Format-Table -AutoSize
```

Name	CPU	Id
ControlCenter	287.421875	11712
WINWORD	143.78125	15992
brave	73.765625	5004
explorer	62.546875	8464
brave	45.921875	5668
brave	25.046875	404
brave	24.5	7492
brave	23.46875	4040
powershell_ise	19.34375	4092
esrv	16.53125	4756
whatsApp	16.34375	15412
nvshelper64	13.703125	9552
nvcontainer	13.625	5812
brave	12.765625	14240
ctfmon	7.296875	12148
NVIDIA Overlay	6.3125	9568
msedgewebview2	6.03125	5112
TextInputHost	4.390625	15100
ipf_helper	4.140625	6896
brave	4.046875	13508
sihost	4.015625	7468
svchost	3.890625	16492
nvcontainer	3.84375	4652
PhoneExperienceHost	3.6875	13408
SearchHost	3.015625	11168

- Format-List – List output.



```
PS C:\Users\khatr> Get-Process | Sort-Object CPU -Descending | Select-Object Name, CPU, Id | Format-List
```

```
Name : ControlCenter
CPU  : 293.890625
Id   : 11712

Name : WINWORD
CPU  : 145.921875
Id   : 15992

Name : brave
CPU  : 74.953125
Id   : 5004

Name : explorer
CPU  : 64.1875
Id   : 8464

Name : brave
CPU  : 46.84375
Id   : 5668

Name : brave
CPU  : 27.59375
Id   : 7492

Name : brave
CPU  : 25.046875
Id   : 404
```

- Format-Wide – Single property per line.

```
PS C:\Users\khatr> Get-Process | Sort-Object CPU -Descending | Select-Object Name | Format-Wide -Column 4
```

Name	CPU
ControlCenter	WINWORD
brave	brave
brave	esrv
nvcntainer	brave
msedgewebview2	brave
sihost	nvcntainer
SearchHost	DSATray
widgts	SystemSettings
NVIDIA Overlay	msedgewebview2
ShellExperienceHost	audiojog
RuntimeBroker	msedgewebview2
a1	WidgetService
msedgewebview2	LockApp
svchost	RuntimeBroker
brave	brave
brave	svchost
msedgewebview2	RuntimeBroker
svchost	brave
UserOOBEBroker	msedgewebview2
svchost	svchost
svchost	svchost
svchost	svchost
svchost	svchost
svchost	svchost
svchost	svchost
svchost	svchost
WUDFHost	AggregatorHost
WinPrvSE	WMIRegistrationService
wininit	WinAPSPV
svchost	System
svchost	svchost
svchost	svchost
svchost	svchost
svchost	svchost
svchost	svchost
ipfsvc	JRNLService
Memory Compression	MemusService
NgcIso	NisSrv
NVDisplay.Container	NVIDisplay.Container
IntelGraphicsSoftware.Service	IntelAudioService
csrss	csrss
IntelCpuIDCPsvc	DtsAppl4Service
fntdtrvhst	Idle
dsm	OneApp_ICGC.WinService
svchost	svchost
svchost	svchost
svchost	svchost
svchost	svchost
svchost	svchost
SecurityHealthService	services
WUDFHost	svchost
svchost	svchost
svchost	svchost

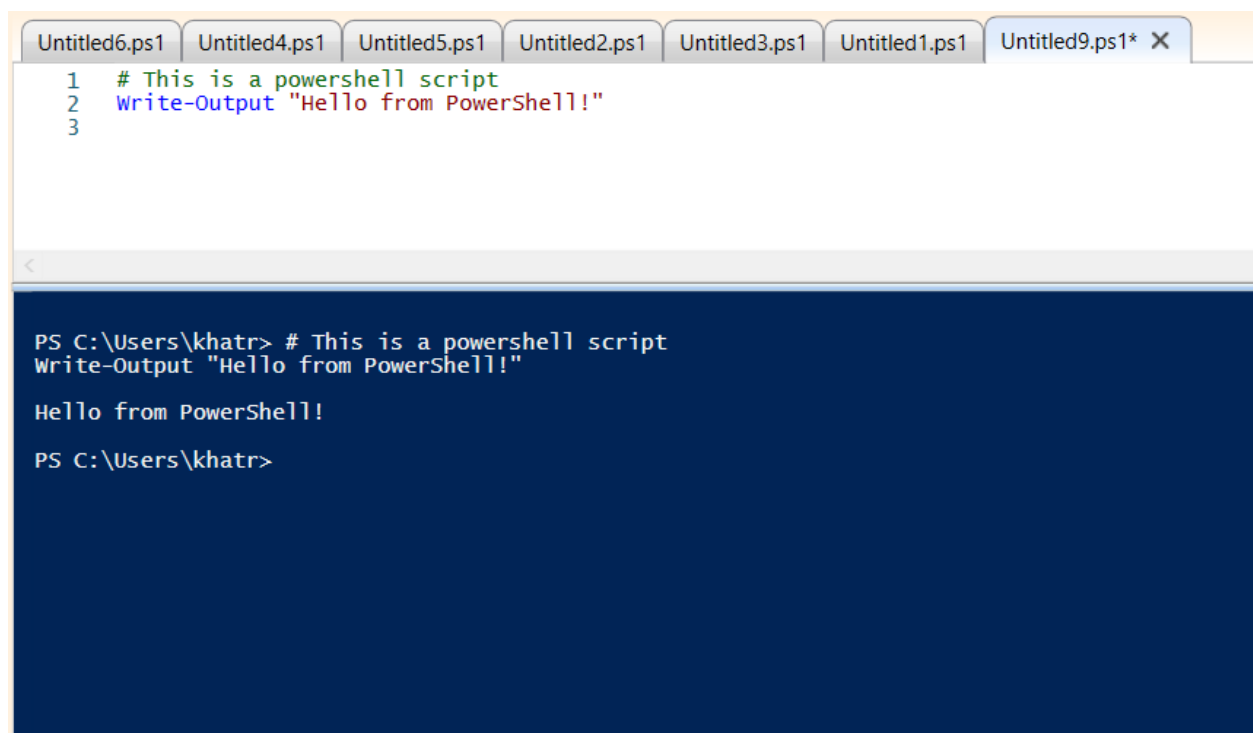
## 6. Scripting Overview

### 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



The image shows a PowerShell script editor window with several tabs. The active tab, 'Untitled9.ps1', contains a script with three lines: a comment, a command, and a blank line. Below the editor, a dark blue terminal window shows the execution of the script, displaying the command and its output.

```
1 # This is a powershell script
2 Write-Output "Hello from PowerShell!"
3
```

```
PS C:\Users\khatr> # This is a powershell script
Write-Output "Hello from PowerShell!"

Hello from PowerShell!

PS C:\Users\khatr>
```