**PRACTICE ACTIVITY 4**

**Assignment: Introducing to Cmdlets, The powershell pipeline, Key Cmdlets, WMI & Powershell, Pipeline filtering & operators, Input output & Formatting, Scripting Overview.**

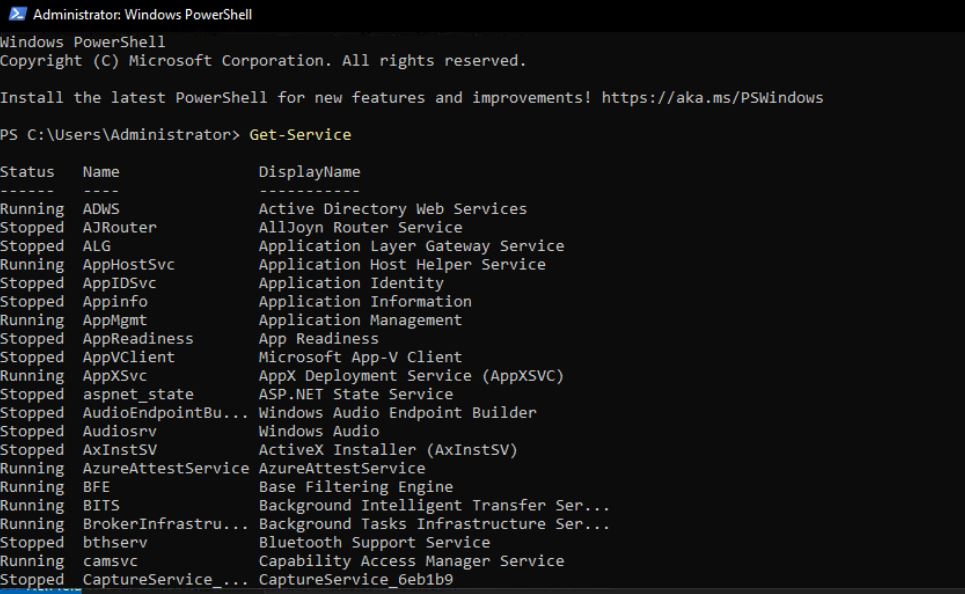
**Name:** Sajiya

**Email id:** [sajiyadl888@gmail.com](mailto:sajiyadl888@gmail.com)

**User id:** 34764

**1.Introducing to Cmdlets**

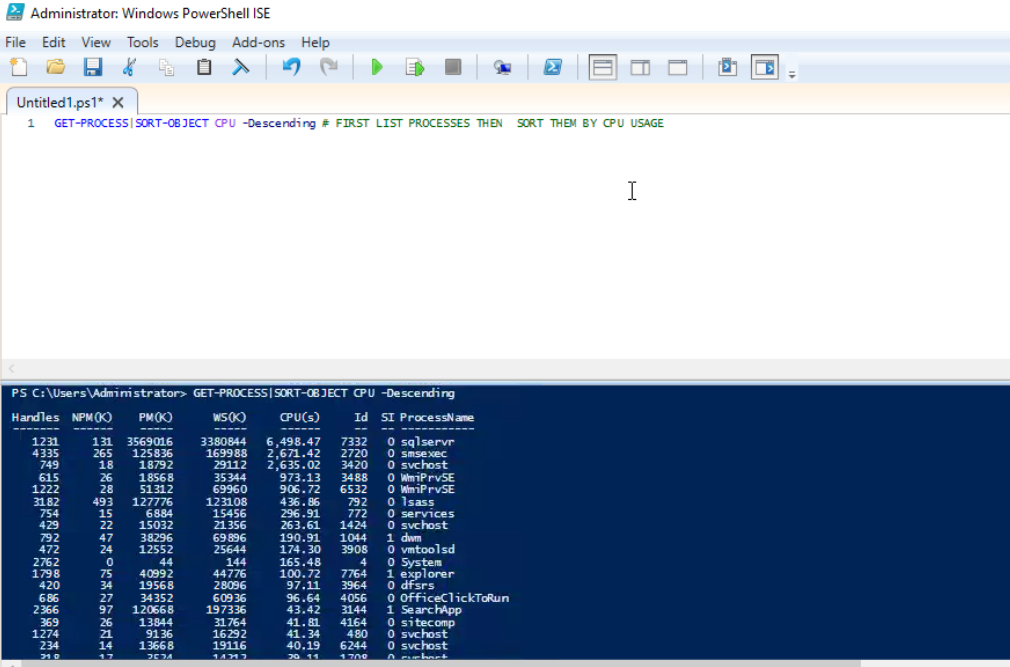
Cmdlets are small commands used in Windows PowerShell to do specific tasks. They are written in a **Verb-Noun** format, such as Get-Process or Start-Service, where the verb is the action and the noun is the thing the action is done on. Cmdlets work with the PowerShell **pipeline**, which means the result of one cmdlet can be sent to another to create a chain of actions. They deal with **objects** instead of plain text, making them more powerful for automation. Cmdlets can take input through parameters or from another cmdlet, perform the action, and then give output that can be used again. For example, Get-Process shows running processes, and its results can be sent to Sort-Object to arrange them in order.

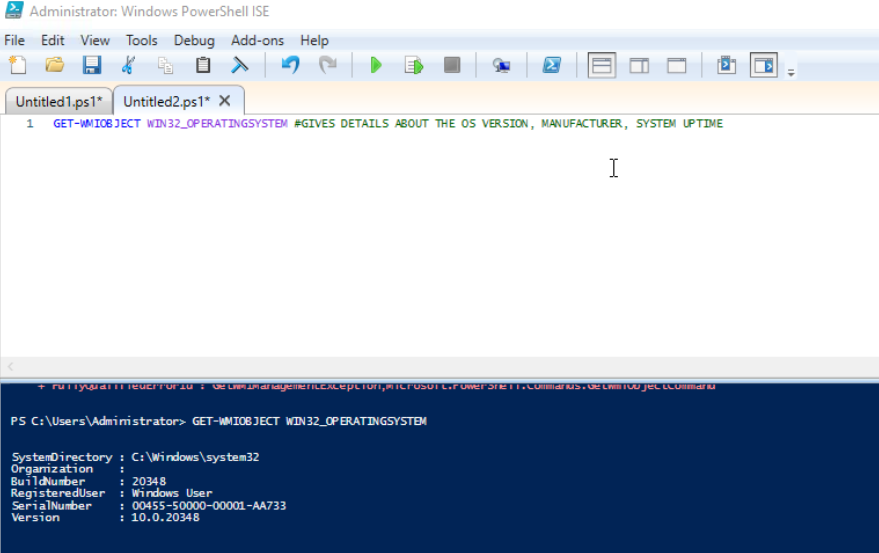
(screenshot of Get-service command)

**2. The powershell pipline**

The PowerShell pipeline is a feature that lets you connect multiple commands so that the output of one command becomes the input of the next. This makes it easy to perform a series of actions without having to store results in between. For example, Get-Process | Sort-Object CPU -Descending first lists processes, then sorts them by CPU usage. The pipeline is powerful because it works with objects instead of plain text, meaning you can pass complete data structures from one command to another.

(screenshot of get-process | sort-object cpu command)

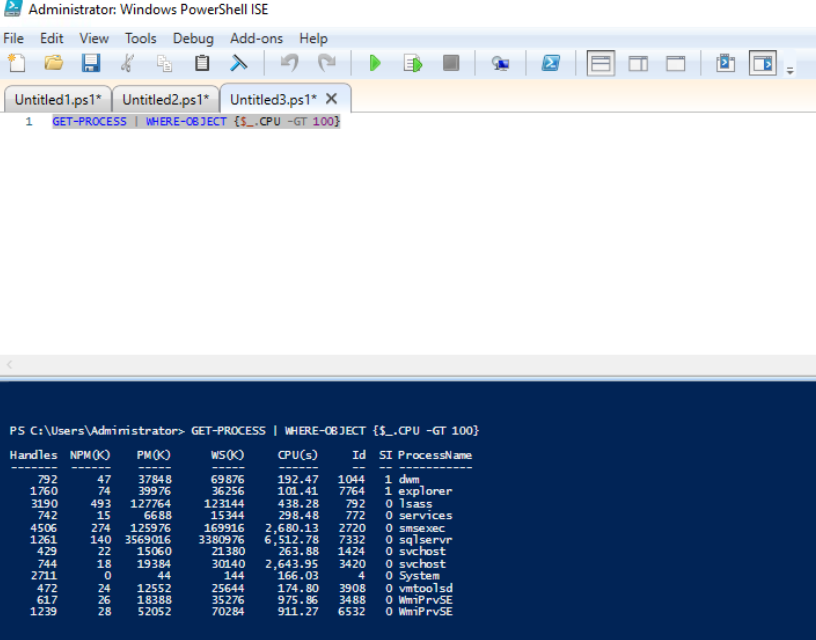
**3. Key Cmdlets**  
Cmdlets are the basic commands in PowerShell used to perform specific tasks. They follow a **Verb-Noun** naming style, such as Get-Process, Start-Service, and New-Item. Cmdlets can take input from the pipeline or through parameters, process it, and give output that can be used again. They work with objects, making them more versatile than traditional commands. For example, Get-Service can be used to list services, and Stop-Service can stop a particular one.

**4. WMI & PowerShell**  
WMI (Windows Management Instrumentation) is a framework in Windows that allows you to access system information and manage devices. PowerShell can use WMI to gather detailed data about hardware, software, and configurations. For instance, Get-WmiObject Win32\_OperatingSystem can give details about the OS version, manufacturer, and system uptime. Combining PowerShell with WMI makes it possible to automate tasks like querying disk space, checking hardware status, or managing remote machines.

(screenshot of get-wmiobject win32\_operatingsystem command)

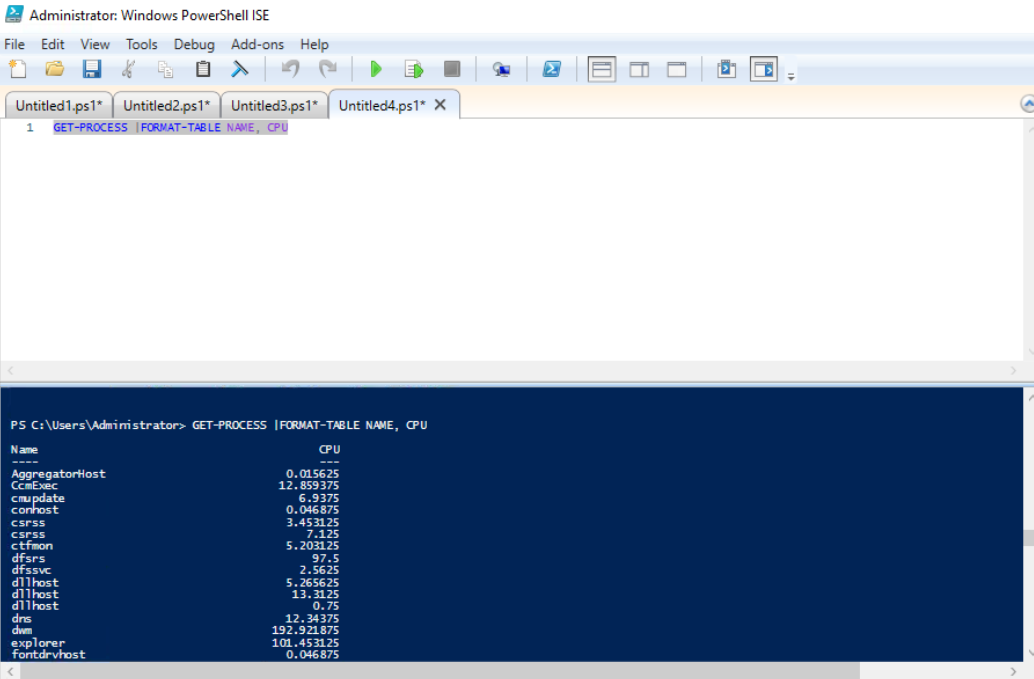
**5. Pipeline Filtering & Operators**  
Filtering in the pipeline allows you to narrow down results to only what you need. PowerShell has cmdlets like Where-Object to filter data and operators like -eq (equals), -gt (greater than), and -like (pattern matching) for conditions. For example, Get-Process | Where-Object { $\_.CPU -gt 100 } shows only processes using more than 100 CPU seconds. These filters make scripts more efficient and targeted.

(screenshot of Get-Process | Where-Object { $\_.CPU -gt 100 } command)



**6. Input/Output & Formatting**  
PowerShell can take input from the keyboard, files, or other commands, and can send output to the screen, files, or even other programs. You can use cmdlets like Read-Host for input, and Out-File to save output to a file. Formatting cmdlets like Format-Table or Format-List help display data neatly. For example, Get-Process | Format-Table Name, CPU shows process names and CPU usage in a clean table.

(screenshot of Get-Process | Format-Table Name, CPU command)

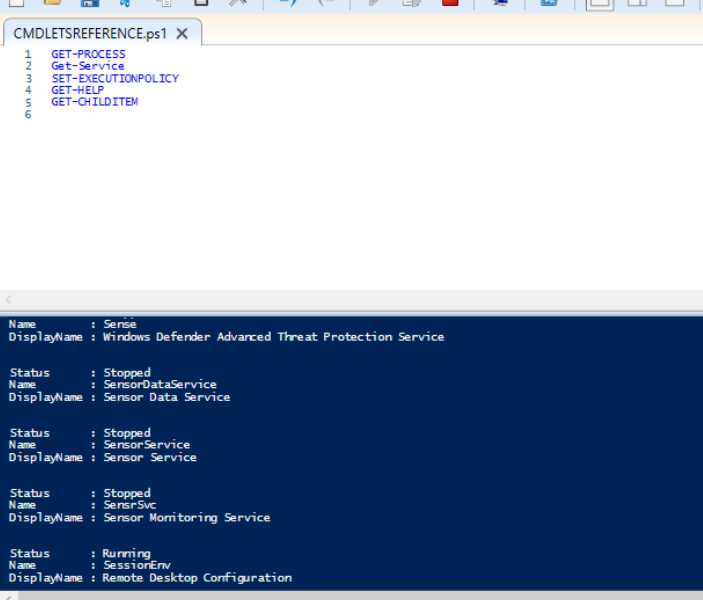


**7. Scripting Overview**  
Scripting in PowerShell means writing a set of commands in a .ps1 file so they can run together automatically. Scripts can include variables, loops, conditions, and functions to make them more dynamic. This is useful for automating repetitive tasks like creating user accounts, checking system health, or installing software. Before running scripts, you might need to change the execution policy using Set-ExecutionPolicy. PowerShell scripting helps save time and reduces manual errors.

**Project 1: Exploring Cmdlet Syntax**

This project focuses on understanding how PowerShell cmdlets are written and used. A **Cmdlet Reference Guide** is created by opening PowerShell ISE or any editor and making a script named CmdletReference.ps1. Choose five cmdlets like Get-Process, Get-Service, Set-ExecutionPolicy, Get-Help, and Get-ChildItem. For each one, write its description, syntax, common parameters, and a usage example. After writing, run the script to test them.  
**Outcome:** Improves understanding of cmdlet structure, parameters, and documentation for better use in real tasks.

(screenshot of Get-Process, Get-Service, Set-ExecutionPolicy, Get-Help, and Get-ChildItem command)

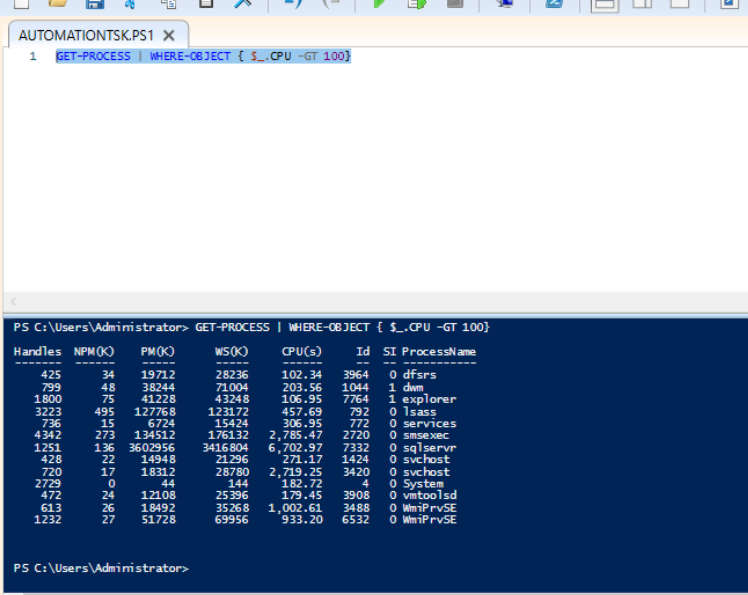


## ****Project 2: Automate a Task with a Cmdlet Script****

Here, the aim is to create a script to perform a routine task automatically. For example, checking for processes with high CPU usage or cleaning temporary files. Create a script named AutomationTask.ps1 that uses:

* Get-Process | Where-Object {$\_.CPU -gt 100} to find high CPU processes.
* Remove-Item to delete temporary files.  
  Combine them and run the script with proper permissions.  
  **Outcome:** Teaches how to chain cmdlets, write automation scripts, and manage system tasks efficiently.

(screenshot of Get-Process | Where-Object {$\_.CPU -gt 100} command)



## ****Project 3: Create a PowerShell Cmdlet Cheat Sheet****

This project involves creating a quick reference for cmdlets. Make a file named PowerShellCheatSheet.md and organize it into:

* **Basic Cmdlets** – e.g., Get-Help, Get-Command, Get-Content.
* **File System Cmdlets** – e.g., New-Item, Copy-Item, Remove-Item.
* **Network Cmdlets** – e.g., Test-Connection, Get-NetIPAddress.  
  Include short examples for each cmdlet.  
  **Outcome:** Helps store and organize cmdlet knowledge for quick access during scripting and troubleshooting.

**(screenshot of New-Item -Path "C:\Users\test.txt" -ItemType "File" command)**

