**WIPRO ASSIGNMENT - 4**

* **Submitted By:** Anuj Bose
* **User ID:** 34757
* **Batch Name:** WiproNGA\_DWS\_B5\_25VID2550
* **Date:** 8th August, 2025
* **Email ID:** [anujbose01012004@gmail.com](mailto:anujbose01012004@gmail.com)

**Introduction To Cmdlets**

**Cmdlets** are lightweight PowerShell commands that work with the pipeline, taking objects as input and outputting objects. They follow a "Verb-Noun" naming format, where the verb defines the action and the noun specifies the target.

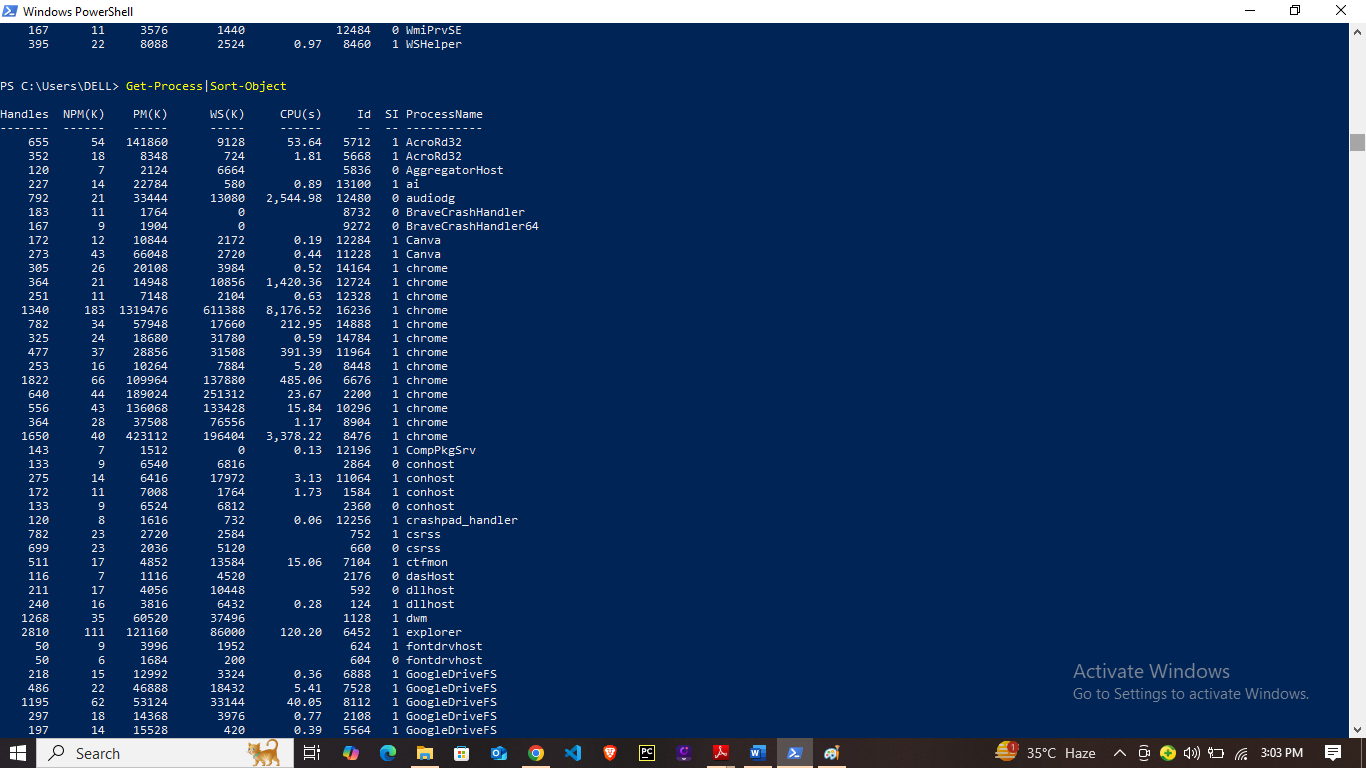
**Key Characteristics of Cmdlets:**

* **Verb-Noun Naming**: Cmdlets use a clear "Verb-Noun" format (e.g., Get-Process, Start-Service).
* **Pipeline-Oriented**: Can be chained in the PowerShell pipeline for complex tasks.
* **Object-Oriented**: Work with objects instead of plain text for more flexible scripting.
* **Built-in and Custom**: PowerShell provides many built-in cmdlets, and users can create custom ones.
* **Not Executables**: Cmdlets are part of PowerShell’s core, not standalone programs.

**How Cmdlets Work:**

1. **Input:** Cmdlets can receive input from the pipeline (other cmdlets or data streams) or can be provided with input directly as parameters.
2. **Processing:** They perform a specific action on the input, based on their verb and noun.
3. **Output:** Cmdlets return objects as output, which can then be piped to subsequent cmdlets in the pipeline.

**Example:** The Get-Process cmdlet retrieves information about running processes. You can use it to get a list of all processes, filter them based on specific criteria, and then pipe the results to another cmdlet like Sort-Object to sort the list.



**Understanding The Powershell Pipeline**

One of the most powerful features of PowerShell is the ability to use Cmdlets in pipelines. This means you can chain multiple Cmdlets together, passing the output of one as the input to the next.

The PowerShell pipeline lets you send the **output** of one command straight into another command as its **input**, without saving it in a variable.

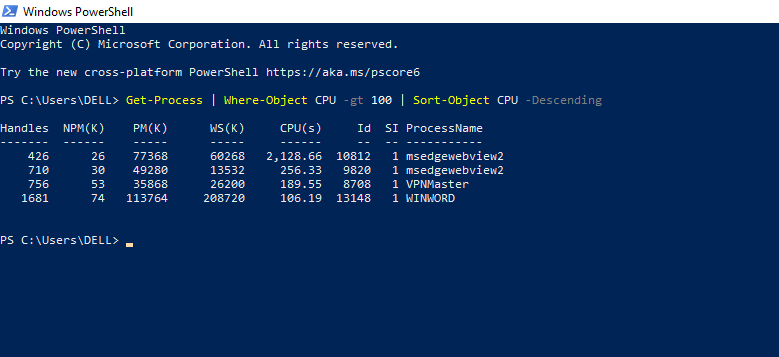
Instead of passing plain text like in traditional command lines, PowerShell passes **objects**—which makes it easier to filter, sort, and work with the data.

**Example:**

**Get-Process | Where-Object CPU -gt 100 | Sort-Object CPU -Descending**

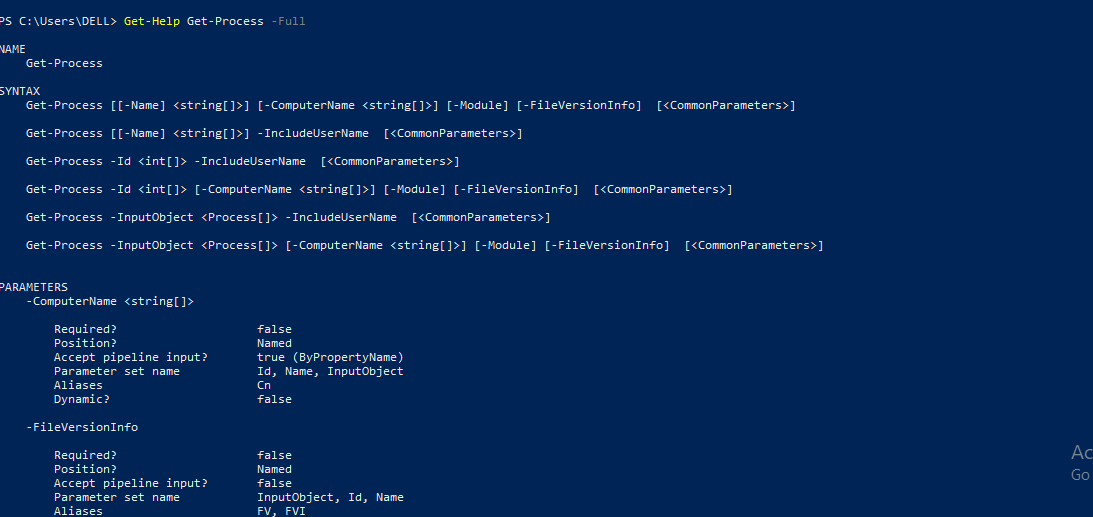
In this example,

* **Get-Process** lists all processes.
* **Where-Object** keeps only those using more than 100 CPU seconds.
* **Sort-Object** sorts them by CPU usage from highest to lowest.

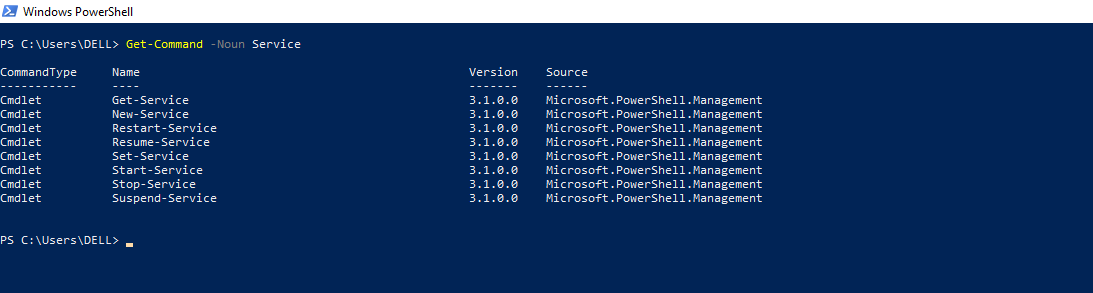


**Key Cmdlets**

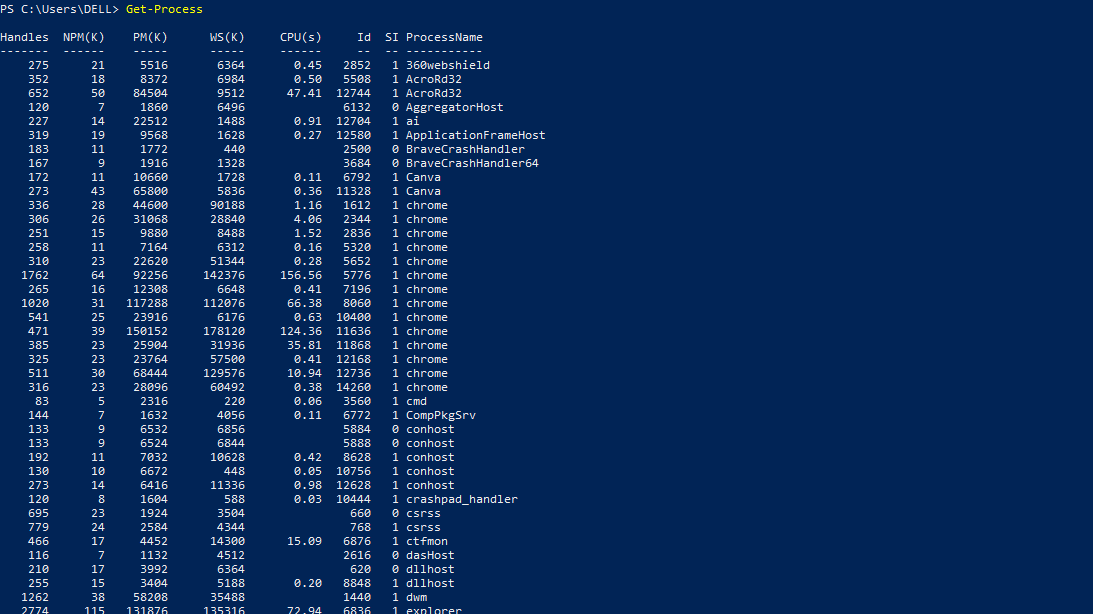
* **Get-Help** – Shows help information about cmdlets, including usage examples and syntax.  
  *Example:* Get-Help Get-Process -Full



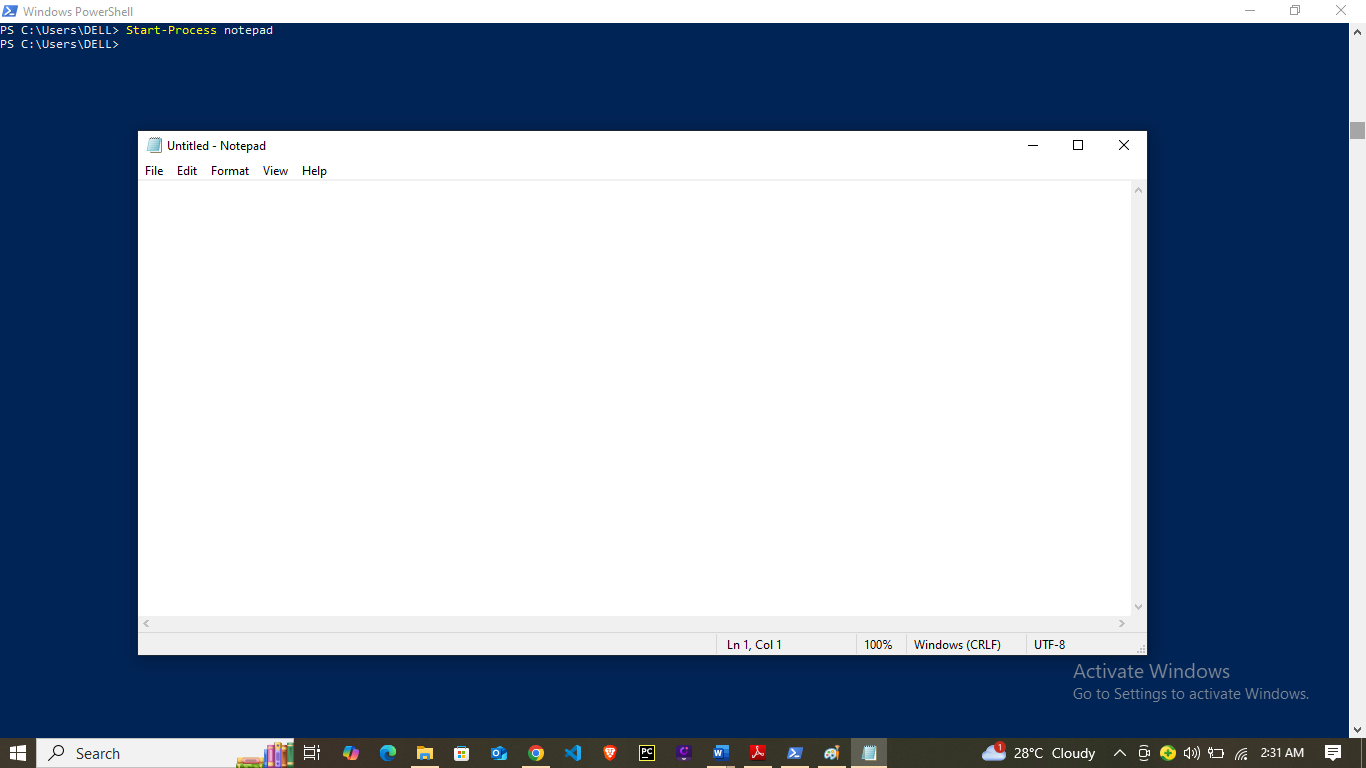
* **Get-Command** – Lists all available cmdlets, functions, scripts, and aliases in your PowerShell session.  
  *Example:* Get-Command -Noun Service



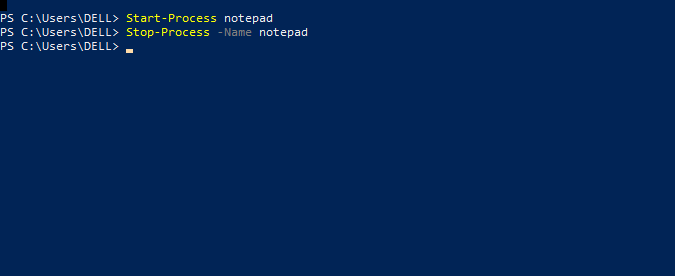
* **Get-Process** – Displays details of processes running on your system.  
  *Example:* Get-Process



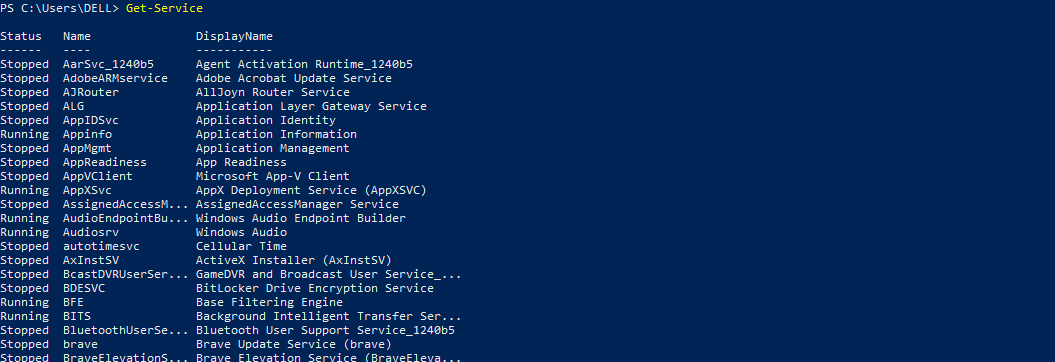
* **Start-Process** – Starts a new process or application.*Example:* Start-Process notepad



* **Stop-Process** – Stops a running process.  
  *Example:* Stop-Process -Name notepad



* **Get-Service** – Shows the status of system services.  
  *Example:* Get-Service



**WMI And Powershell**

**WMI**, or **Windows Management Instrumentation**, is a feature that allows access to detailed system information, including hardware, software, and services.

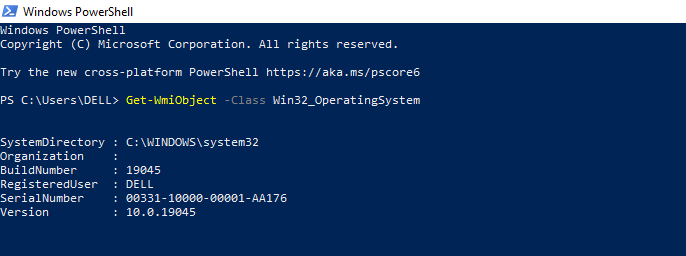
With PowerShell, you can easily retrieve or modify this information using WMI commands.

**Example:**

**Get-WmiObject -Class Win32\_OperatingSystem**

This command displays details about your Windows operating system, such as its version, build number, and more.

WMI is particularly useful for tasks like monitoring disk space, gathering system details, or managing services remotely — all directly through PowerShell.



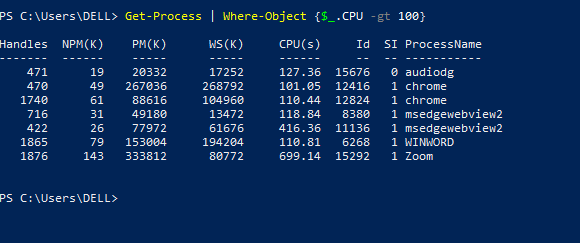
**Pipeline Filtering And Operators**

In PowerShell, a **pipeline** (|) passes the output of one command directly as input to another. **Filtering** helps you work only with the data you need, instead of processing everything.

**Example:**

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

This filters only the processes using more than 100 CPU units.



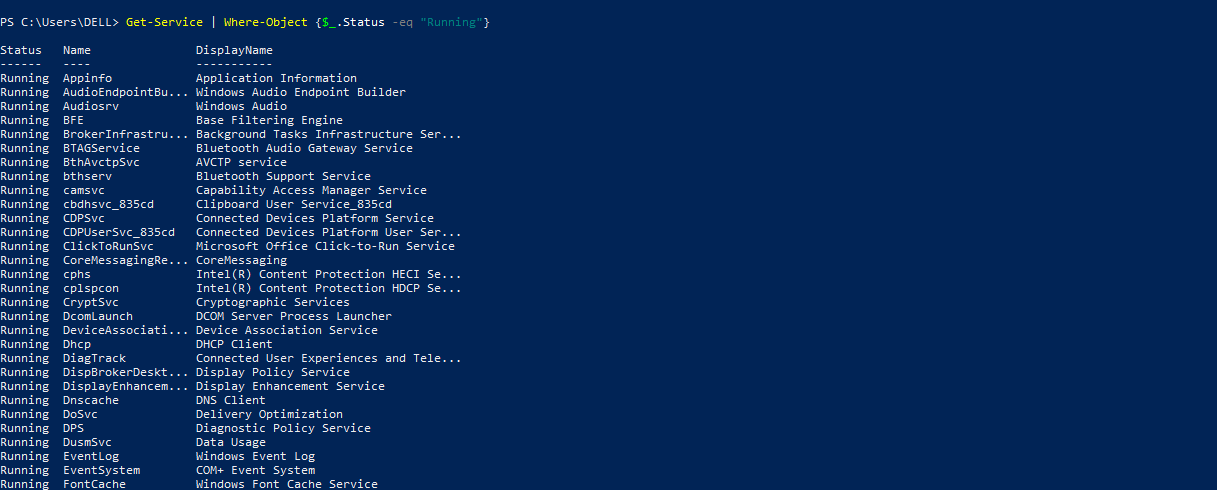
**Operators** help build those conditions:-

* **-eq →** equals
* **-ne →** not equal
* **-gt →** greater than
* **-lt →** less than
* **-like →** matches pattern (wildcards)
* **-and, -or, -not →** logical operators

**Example with Operator:**

**Get-Service | Where-Object {$\_.Status -eq "Running"}**

This shows only running services.



**Input, Output And Formatting**

In PowerShell, **input** refers to the data or commands you provide to a cmdlet or script.

**Output** is the result that PowerShell returns after executing the command.

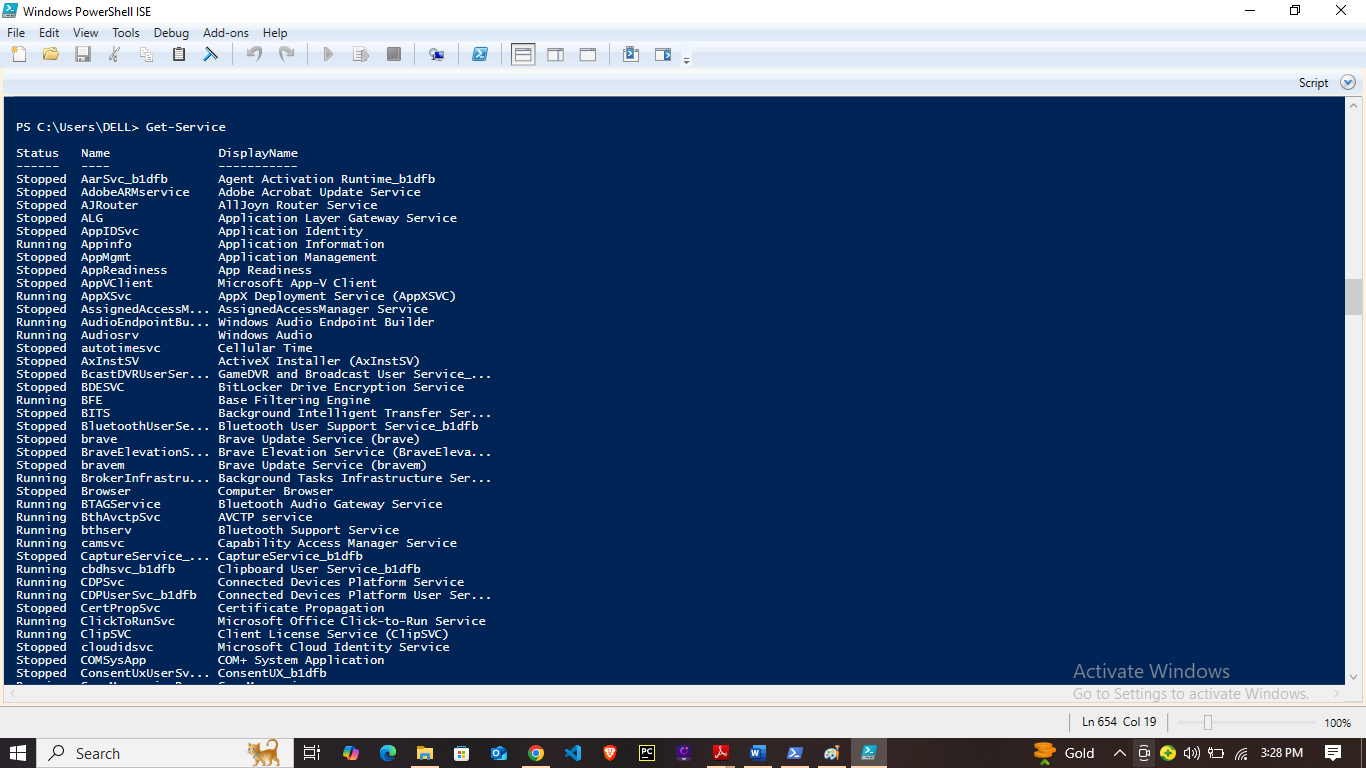
**Formatting** determines how that output is displayed on the screen.

**Example of Input & Output:**

* **Input:** Command → Get-Service
* **Output:** A list of all services along with their names and current statuses

**Formatting Cmdlets** help change how the output is shown:

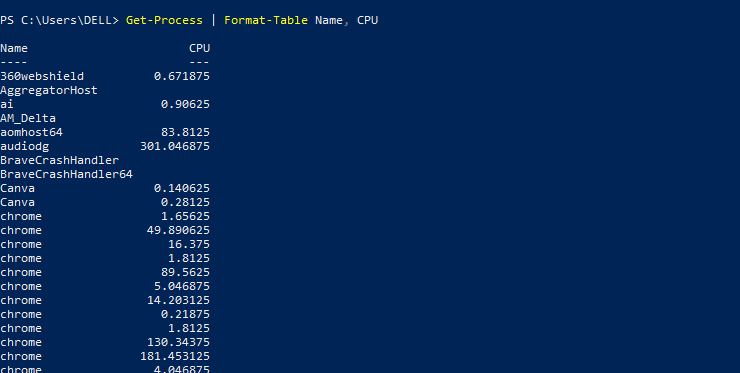
* **Format-Table –** Shows data in table form
* **Format-List –** Shows detailed data in list form
* **Out-File –** Sends output to a file
* **Out-GridView –** Opens output in a grid window
* **Out-Host –** Sends output to the screen (default)



**Example:**

**Get-Process | Format-Table Name, CPU**

This shows process names and CPU usage in a neat table format.



**Scripting**

**PowerShell scripting** means writing a series of commands in a .ps1 file to automate tasks.

Instead of typing each command one by one, you save them in a script and run all at once.

Why we use scripts?

* Save time
* Repeat tasks easily
* Reduce manual errors
* Automate system tasks, app installs, reports, etc.

**Project 1**

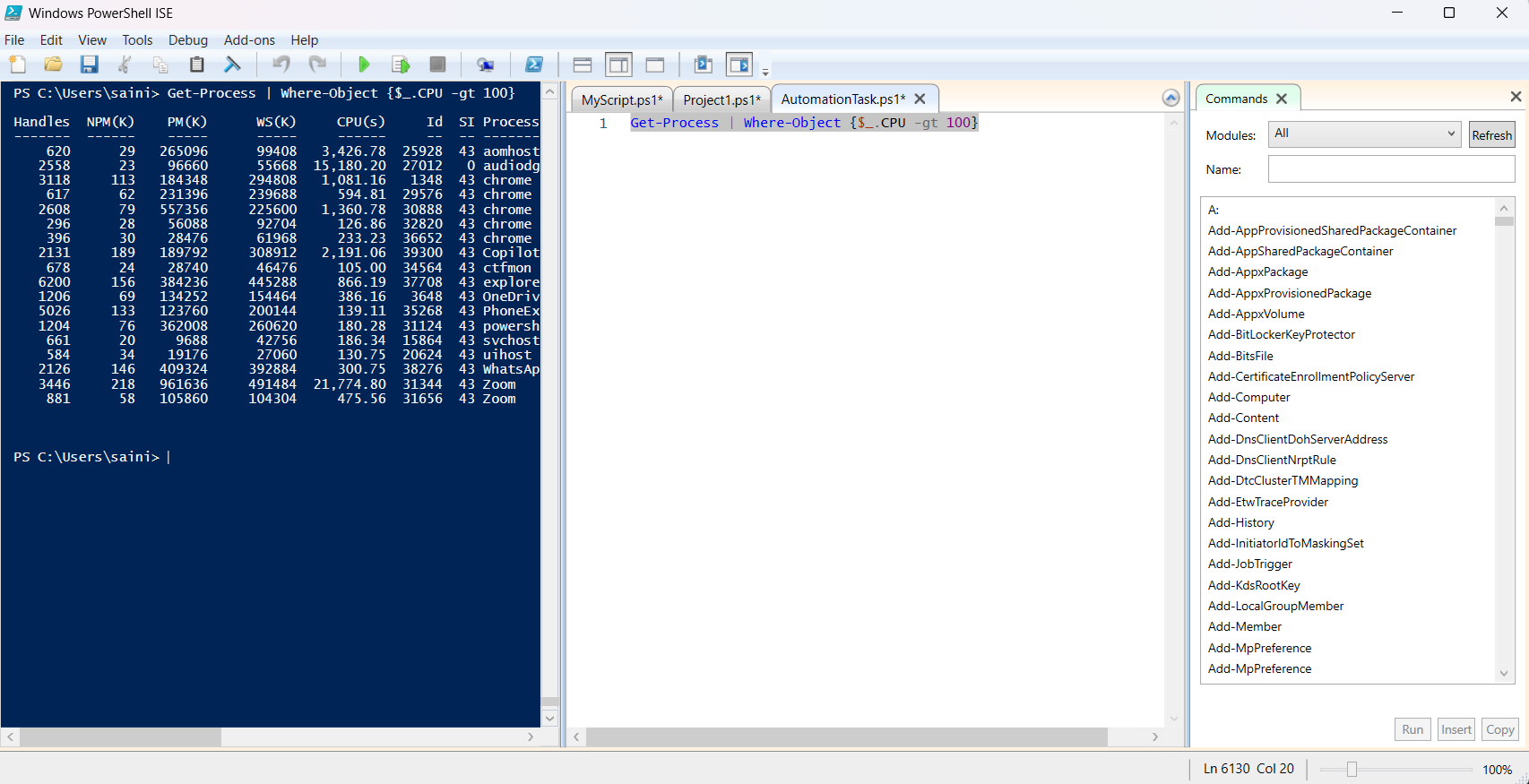
**Automate a Task with a Cmdlet Script**

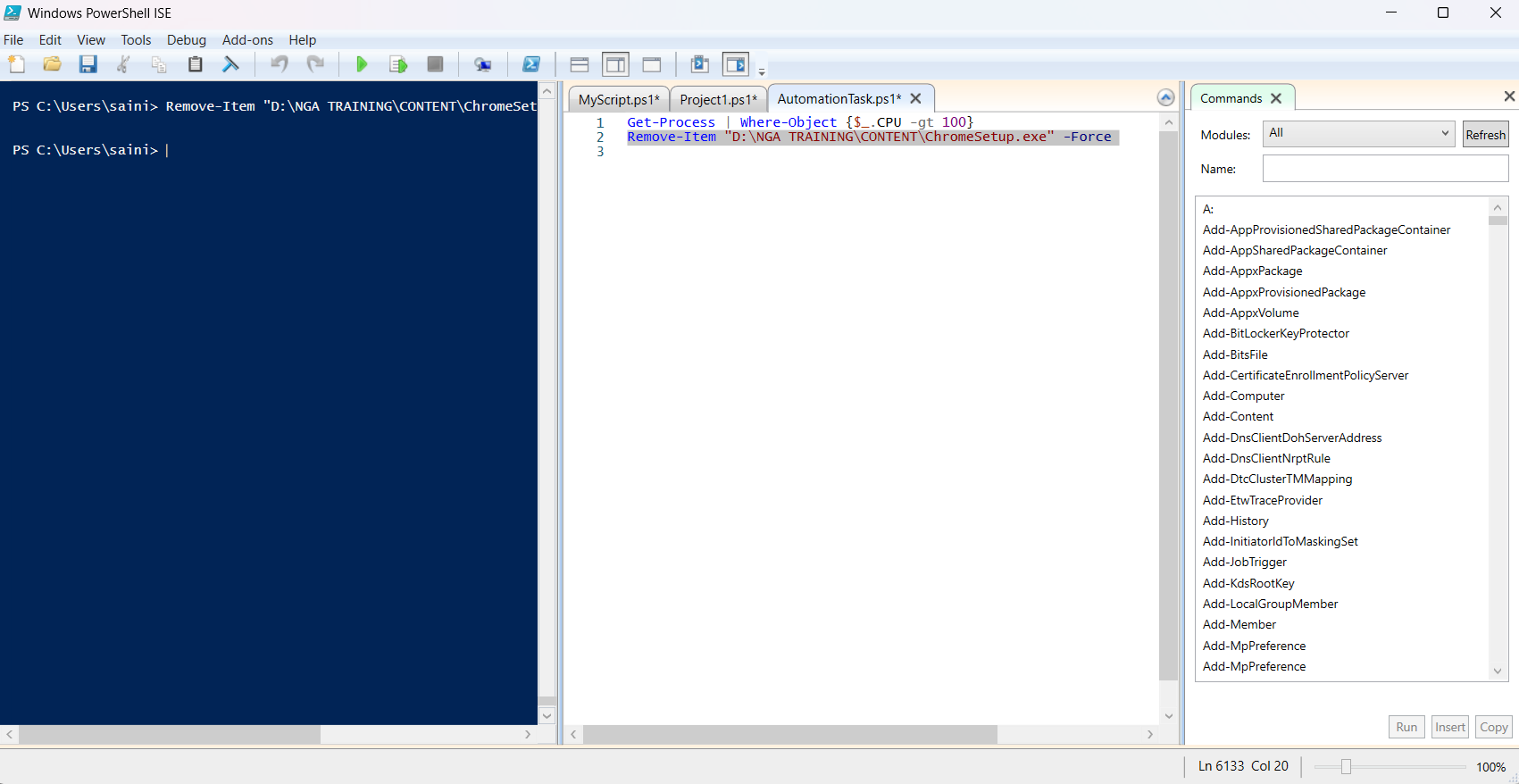
Create a new script file named AutomationTask.ps1.

**Use cmdlets like:**

**●** Get-Process | Where-Object {$\_.CPU -gt 100} to find processes consuming significant CPU.

● Use Remove-Item to delete files in a temporary directory





**Project 2**

**Create a PowerShell Cmdlet Cheat Sheet**

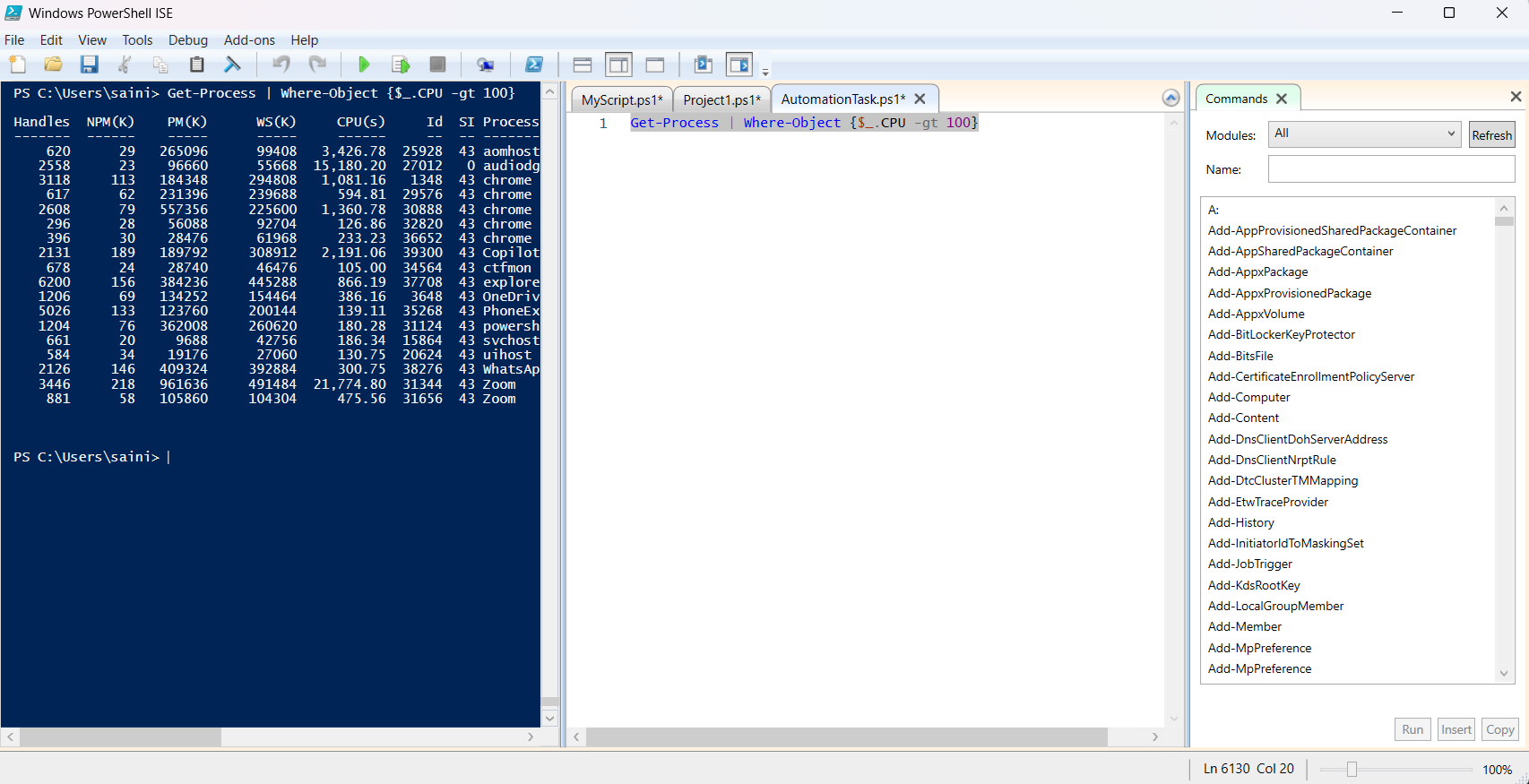
Create a new Markdown file named PowerShellCheatSheet.md.

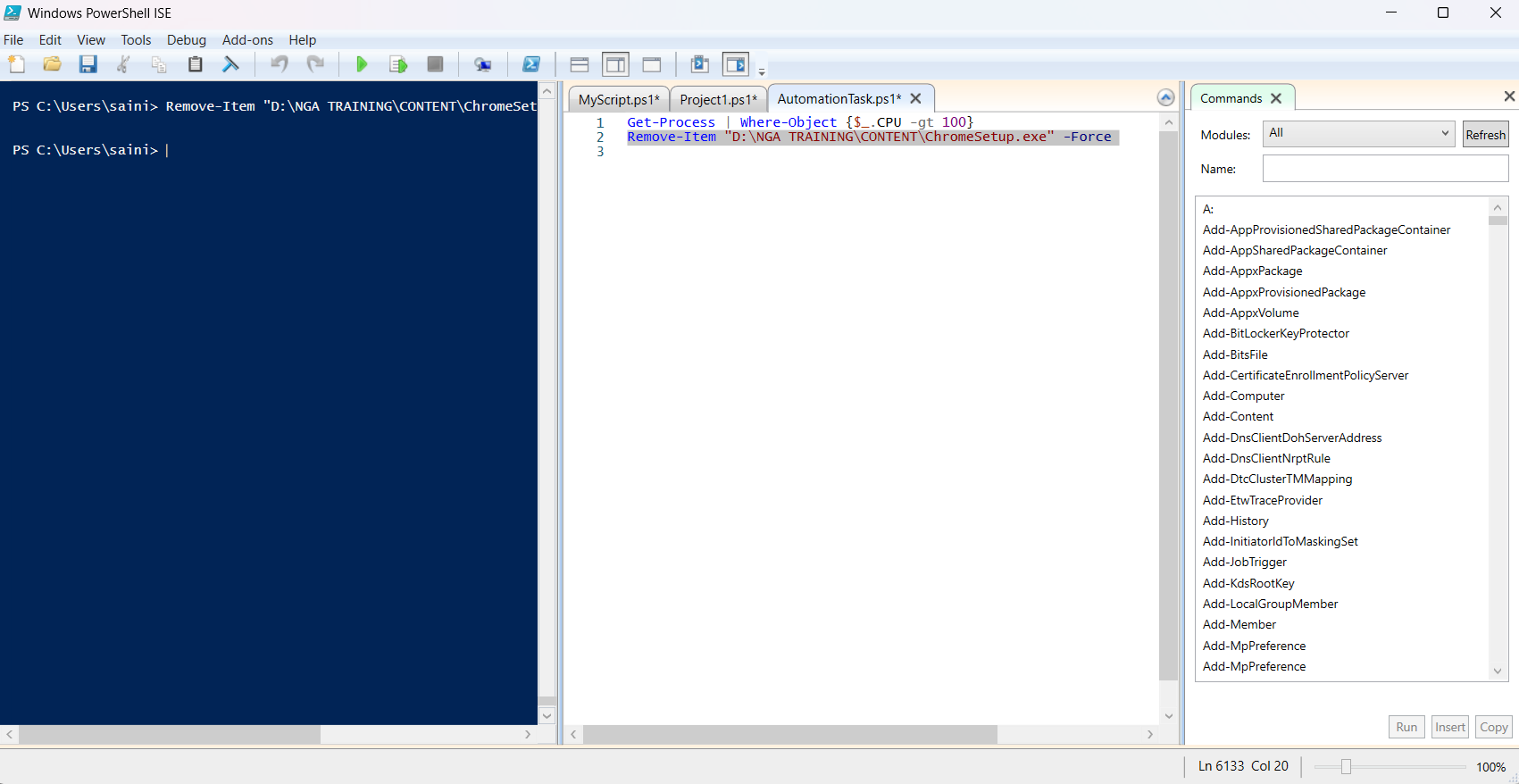
Basic Cmdlets:

List common cmdlets like Get-Help, Get-Command, Get-Content, etc., with a brief description.

● File System Cmdlets: Include cmdlets such as New-Item, Copy-Item, and Remove-Item.

● Network Cmdlets: List cmdlets like Test-Connection and Get-NetIPAddress.





**Project 3**

**Create a PowerShell Cmdlet Cheat Sheet**

Create a new Markdown file named PowerShellCheatSheet.md.

Basic Cmdlets:

List common cmdlets like Get-Help, Get-Command, Get-Content, etc., with a brief description.

● File System Cmdlets: Include cmdlets such as New-Item, Copy-Item, and Remove-Item.

● Network Cmdlets: List cmdlets like Test-Connection and Get-NetIPAddress.

