**Today Assignment – Date: August 9,2025**

**Name: Lalit joshi**

**GAMIL:joshilalit2275@gmail.com**

**Powershell cmdlets Scripts and Commands**

**1.Get-NetIPConfiguration | Out-File D:\powershellcommands\datafiles.txt**

This command gathers detailed network configuration information—like IP addresses, adapters, and subnet details—and saves all output to the specified file, making it easy to review network settings later or share them for troubleshooting, without displaying anything in the PowerShell console.

**2.Get-NetIPConfiguration | Out-File D:\powershellcommands\netfileipconfigs.txt**

Just like before, this command collects the current computer's network configuration details, such as IP addresses and DNS settings, and saves the results to another file. It helps create an organized record of network information that can be referenced or compared with previous outputs in troubleshooting scenarios.

**3.Get-Service| Out-File D:\powershellcommands\processfile -Append**

Lists every Windows service running or available on your computer, including their state (stopped, running, etc.). The results are appended to an existing file, meaning it adds the new service list after the file's current contents—helpful for tracking service status over time without overwriting older data.

**4.Get-ChildItem | Out-File D:\powershellcommands\allfiles**

Displays all files and folders in the current directory and exports that listing to a file. This allows you to archive or share a snapshot of the current directory’s contents. It's useful for keeping directory inventories or verifying which files exist at a specific moment.

**5.Get-Process | Sort-Object CPU -Descending | Select-Object -First 5 | Select-Object Name, CPU**

Shows the five processes currently using the most CPU resources. It lists the process name and CPU usage, sorted from highest to lowest. This helps quickly identify what’s consuming the most processing power on your system, which is especially useful for performance troubleshooting.

**6.Get-EventLog -LogName System -Newest 5**

Displays the five most recent entries in the Windows System event log, helping you quickly see the latest system-level events like errors or warnings. This can be key for diagnosing issues right after they occur and gives a snapshot of your system's recent health and activity.

**7.Get-PSDrive -PSProvider FileSystem | Out-File D:\powershellcommands\datafiles.txt -Append**

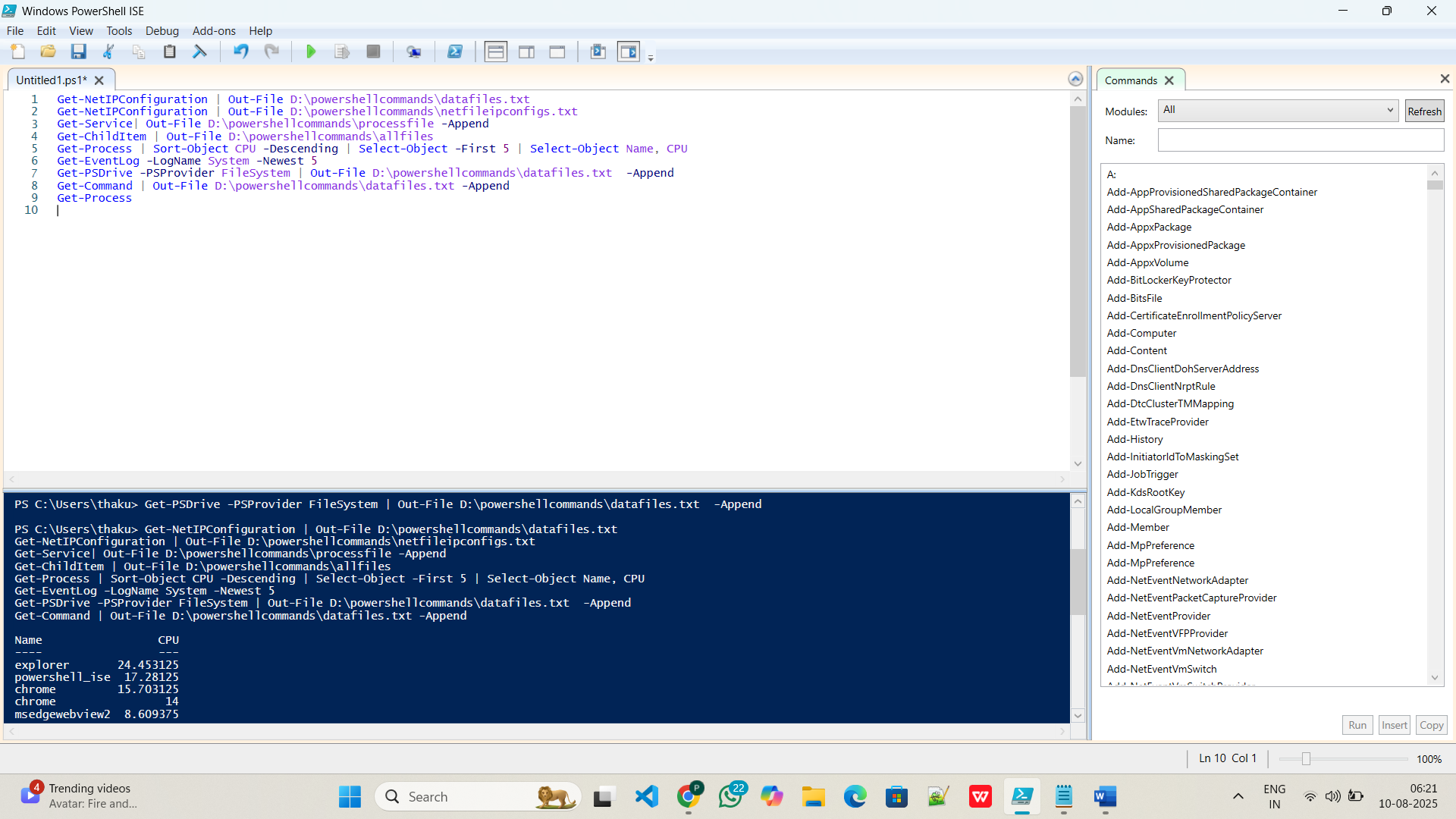
Lists all file system drives PowerShell recognizes (like C:, D:). It exports this list to the named file, appending the details without erasing its current contents. This is useful for tracking what drives have been added or removed on your machine over time.

**8.Get-Command | Out-File D:\powershellcommands\datafiles.txt -Append**

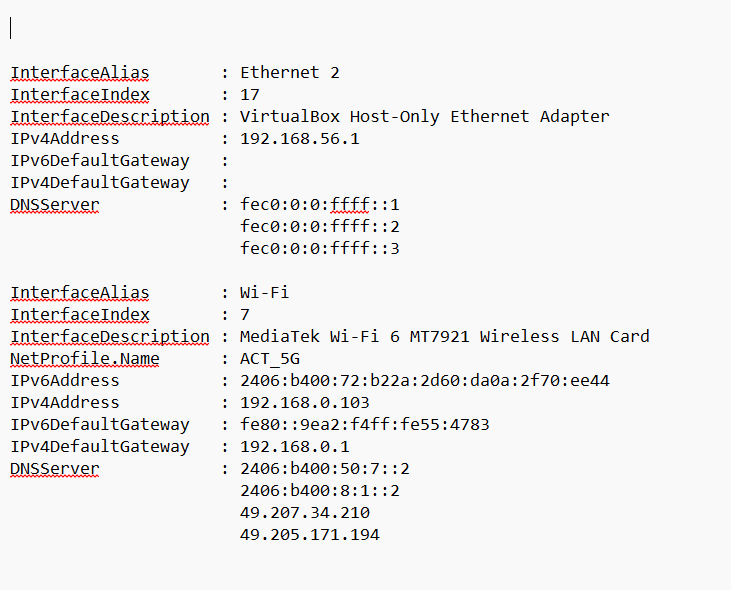
Lists all available PowerShell commands (cmdlets, functions, aliases). It appends the output to the same file as earlier, building a growing record of your session’s command availability. This helps users reference what commands are present at a given time or compare environments.

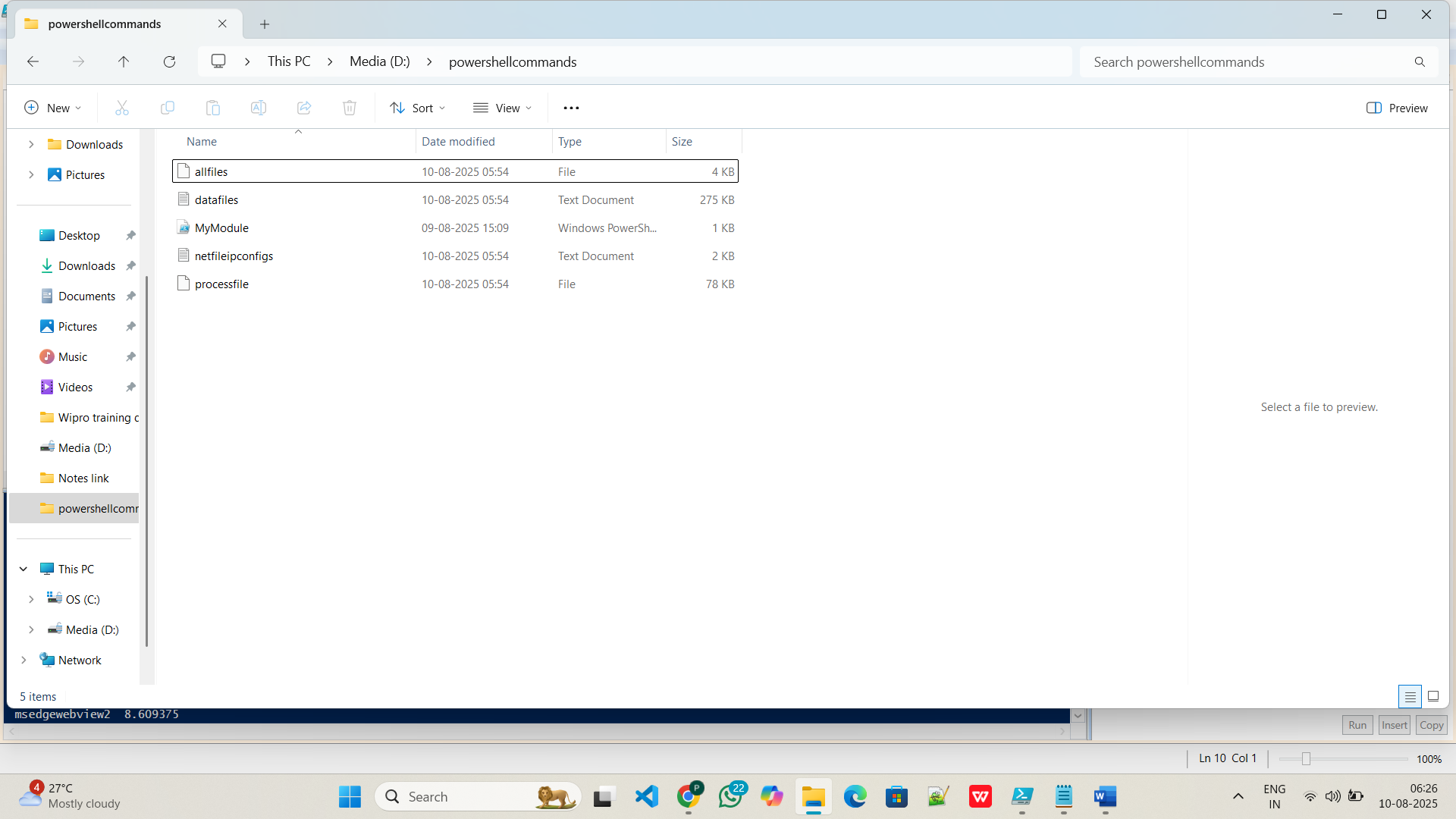
**9.Get-Process**

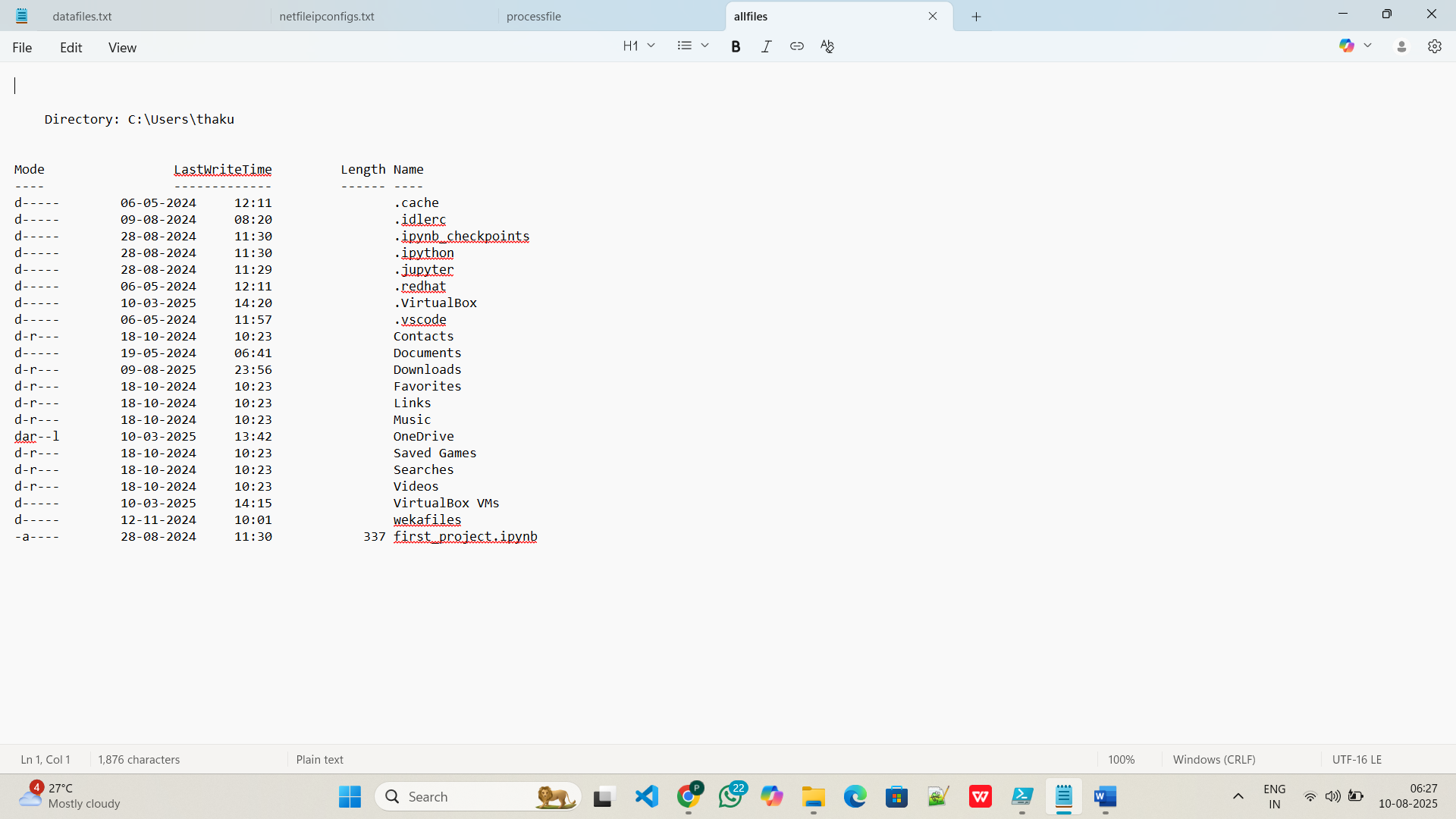
Lists all running processes on your computer, including their names and key metrics (like memory and CPU usage). It helps monitor system activity and check what's currently active. By itself, it displays the output in the console and doesn't save or organize the information elsewhere.



**Practice Screenshots**







* **$name = Read-Host "please enter name"**

**Write-Output "Helo ,lets welcome $name"**

This PowerShell script prompts the user to type their name, stores it in a variable, and then displays a personalized welcome message. Read-Host collects user input; Write-Output sends text or data to the console or output stream.

**Output-**

PS C:\Users\thaku> $name = Read-Host "please enter name"

Write-Output "Hello ,lets welcome $name"

please enter name: Prasang Kumar Singh

Hello ,lets welcome Prasang Kumar Singh

* **$name = "Prasang Kumar Singh"**

**$age = 19**

**"My name is {0} and I am {1} years old" -f $name, $age**

**Output-**

PS C:\Users\thaku> $name = "Prasang Kumar Singh"

$age = 19

"My name is {0} and I am {1} years old" -f $name, $age

My name is Prasang Kumar Singh and I am 19 years old

* **$fruits = "banana","apple"**

**$fruits[0]**

**$fruits+="kiwi"**

**$fruits[2]**

**Output-**

PS C:\Users\thaku> $fruits = "banana","apple"

$fruits[0]

$fruits+="kiwi"

$fruits[2]

banana

kiwi

* **$age = 20**

**if ($age -ge 22) {**

**write-Host "Age is greater"**

**}**

* **$age = 25**

**if ($age -ge 18) {**

**Write-Host "Adult"**

**} elseif ($age -ge 13) {**

**Write-Host "Teenager"**

**} else {**

**Write-Host "Child"**

**}**

**Output:-**

PS C:\Users\thaku> $age = 20

if ($age -ge 19) {

write-Host "Age is smaller"

}

$age = 25

if ($age -ge 18) {

Write-Host "Adult"

} elseif ($age -ge 13) {

Write-Host "Teenager"

} else {

Write-Host "Child"

}

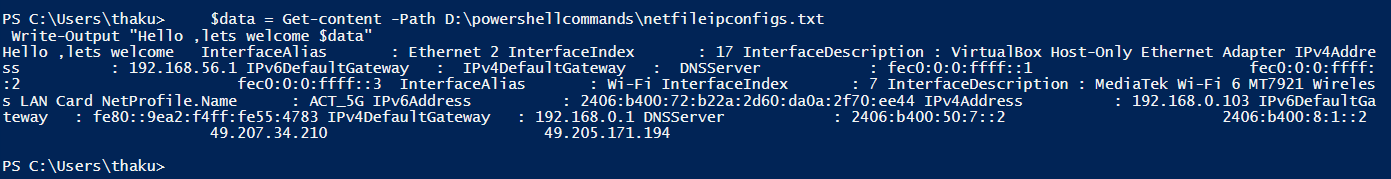
Age is smaller

Adult

**$data = Get-content -Path D:\powershellcommands\netfileipconfigs.txt**

**Write-Output "Hello ,lets welcome $data"**

This PowerShell script reads the contents of the file at the given path into the $data variable, then displays a greeting message that includes whatever text or information was read from that file using Write-Output.



**Get-Service | Get-Member | Out-File D:\powershellcommands\opfilecommands.txt**

This script lists all services on the system using Get-Service, passes the service objects to Get-Member to show their properties and methods, and appends this detailed member information to a text file for reference or documentation purposes.

**Output-**

TypeName: System.ServiceProcess.ServiceController

Name MemberType Definition

---- ---------- ----------

Name AliasProperty Name = ServiceName

RequiredServices AliasProperty RequiredServices = ServicesDependedOn

Disposed Event System.EventHandler Disposed(System.Object, System.EventArgs)

Close Method void Close()

Continue Method void Continue()

CreateObjRef Method System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)

Dispose Method void Dispose(), void IDisposable.Dispose()

Equals Method bool Equals(System.Object obj)

ExecuteCommand Method void ExecuteCommand(int command)

GetHashCode Method int GetHashCode()

GetLifetimeService Method System.Object GetLifetimeService()

GetType Method type GetType()

InitializeLifetimeService Method System.Object InitializeLifetimeService()

Pause Method void Pause()

Refresh Method void Refresh()

Start Method void Start(), void Start(string[] args)

Stop Method void Stop()

WaitForStatus Method void WaitForStatus(System.ServiceProcess.ServiceControllerStatus desiredStatus), void WaitForStatus(System.S...

CanPauseAndContinue Property bool CanPauseAndContinue {get;}

CanShutdown Property bool CanShutdown {get;}

CanStop Property bool CanStop {get;}

Container Property System.ComponentModel.IContainer Container {get;}

DependentServices Property System.ServiceProcess.ServiceController[] DependentServices {get;}

DisplayName Property string DisplayName {get;set;}

MachineName Property string MachineName {get;set;}

ServiceHandle Property System.Runtime.InteropServices.SafeHandle ServiceHandle {get;}

ServiceName Property string ServiceName {get;set;}

ServicesDependedOn Property System.ServiceProcess.ServiceController[] ServicesDependedOn {get;}

ServiceType Property System.ServiceProcess.ServiceType ServiceType {get;}

Site Property System.ComponentModel.ISite Site {get;set;}

StartType Property System.ServiceProcess.ServiceStartMode StartType {get;}

Status Property System.ServiceProcess.ServiceControllerStatus Status {get;}

ToString ScriptMethod System.Object ToString();

**Get-Date | Get-Member | Out-File D:\powershellcommands\opfilecommands.txt -Append**

This script gets the current date and time with Get-Date, examines its properties and methods using Get-Member, and appends this object type information to a text file for documentation, learning, or automation reference purposes.

Output:-

TypeName: System.DateTime

Name MemberType Definition

---- ---------- ----------

Add Method datetime Add(timespan value)

AddDays Method datetime AddDays(double value)

AddHours Method datetime AddHours(double value)

AddMilliseconds Method datetime AddMilliseconds(double value)

AddMinutes Method datetime AddMinutes(double value)

AddMonths Method datetime AddMonths(int months)

AddSeconds Method datetime AddSeconds(double value)

AddTicks Method datetime AddTicks(long value)

AddYears Method datetime AddYears(int value)

CompareTo Method int CompareTo(System.Object value), int CompareTo(datetime value), int IComparable.CompareTo(System.Object obj),...

Equals Method bool Equals(System.Object value), bool Equals(datetime value), bool IEquatable[datetime].Equals(datetime other)

GetDateTimeFormats Method string[] GetDateTimeFormats(), string[] GetDateTimeFormats(System.IFormatProvider provider), string[] GetDateTim...

GetHashCode Method int GetHashCode()

GetObjectData Method void ISerializable.GetObjectData(System.Runtime.Serialization.SerializationInfo info, System.Runtime.Serializati...

GetType Method type GetType()

GetTypeCode Method System.TypeCode GetTypeCode(), System.TypeCode IConvertible.GetTypeCode()

IsDaylightSavingTime Method bool IsDaylightSavingTime()

Subtract Method timespan Subtract(datetime value), datetime Subtract(timespan value)

ToBinary Method long ToBinary()

ToBoolean Method bool IConvertible.ToBoolean(System.IFormatProvider provider)

ToByte Method byte IConvertible.ToByte(System.IFormatProvider provider)

ToChar Method char IConvertible.ToChar(System.IFormatProvider provider)

ToDateTime Method datetime IConvertible.ToDateTime(System.IFormatProvider provider)

ToDecimal Method decimal IConvertible.ToDecimal(System.IFormatProvider provider)

ToDouble Method double IConvertible.ToDouble(System.IFormatProvider provider)

ToFileTime Method long ToFileTime()

ToFileTimeUtc Method long ToFileTimeUtc()

ToInt16 Method int16 IConvertible.ToInt16(System.IFormatProvider provider)

ToInt32 Method int IConvertible.ToInt32(System.IFormatProvider provider)

ToInt64 Method long IConvertible.ToInt64(System.IFormatProvider provider)

ToLocalTime Method datetime ToLocalTime()

ToLongDateString Method string ToLongDateString()

ToLongTimeString Method string ToLongTimeString()

ToOADate Method double ToOADate()

ToSByte Method sbyte IConvertible.ToSByte(System.IFormatProvider provider)

ToShortDateString Method string ToShortDateString()

ToShortTimeString Method string ToShortTimeString()

ToSingle Method float IConvertible.ToSingle(System.IFormatProvider provider)

ToString Method string ToString(), string ToString(string format), string ToString(System.IFormatProvider provider), string ToSt...

ToType Method System.Object IConvertible.ToType(type conversionType, System.IFormatProvider provider)

ToUInt16 Method uint16 IConvertible.ToUInt16(System.IFormatProvider provider)

ToUInt32 Method uint32 IConvertible.ToUInt32(System.IFormatProvider provider)

ToUInt64 Method uint64 IConvertible.ToUInt64(System.IFormatProvider provider)

ToUniversalTime Method datetime ToUniversalTime()

DisplayHint NoteProperty DisplayHintType DisplayHint=DateTime

Date Property datetime Date {get;}

Day Property int Day {get;}

DayOfWeek Property System.DayOfWeek DayOfWeek {get;}

DayOfYear Property int DayOfYear {get;}

Hour Property int Hour {get;}

Kind Property System.DateTimeKind Kind {get;}

Millisecond Property int Millisecond {get;}

Minute Property int Minute {get;}

Month Property int Month {get;}

Second Property int Second {get;}

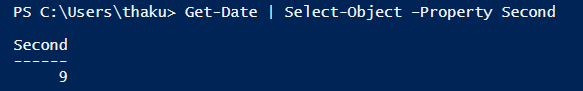
Ticks Property long Ticks {get;}

TimeOfDay Property timespan TimeOfDay {get;}

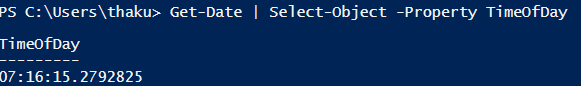
Year Property int Year {get;}

DateTime ScriptProperty System.Object DateTime {get=if ((& { Set-StrictMode -Version 1; $this.DisplayHint }) -ieq "Date")...

**Get-Date | Select-Object -Property Second**  
This command retrieves the current date and time, then extracts only the Second value (0–59) from it. It’s useful when you specifically need to know the current second value for logging or timing operations.



**Get-Date | Select-Object -Property TimeOfDay**  
This command gets the system’s current date and time, then displays the TimeOfDay property as a TimeSpan value showing the elapsed time since midnight, including hours, minutes, seconds, and fractions, often used for time calculations or performance measurements.



**Get-Command \*hotfix\* | Out-File D:\Powershellcommands\opfilecommands.txt -Append**

This command finds all PowerShell commands containing “hotfix” in their names, then appends the list of those commands to the specified file, preserving existing contents instead of replacing them, useful for documentation or tracking command availability over time.



**Get-HotFix | Get-Member**

This command retrieves installed Windows hotfixes, then uses Get-Member to list all properties and methods of each hotfix object. It helps understand available data fields for filtering, reporting, or scripting purposes when working with update information.

**Output:-** PS C:\Users\thaku> Get-HotFix | Get-Member

TypeName:

System.Management.ManagementObject#root\cimv2\Win32\_QuickFixEngineering

Name MemberType Definition

---- ---------- ----------

PSComputerName AliasProperty PSComputerName = \_\_SERVER

Caption Property string Caption {get;set;}

CSName Property string CSName {get;set;}

Description Property string Description {get;set;}

FixComments Property string FixComments {get;set;}

HotFixID Property string HotFixID {get;set;}

InstallDate Property string InstallDate {get;set;}

InstalledBy Property string InstalledBy {get;set;}

Name Property string Name {get;set;}

ServicePackInEffect Property string ServicePackInEffect {get;set;}

Status Property string Status {get;set;}

\_\_CLASS Property string \_\_CLASS {get;set;}

\_\_DERIVATION Property string[] \_\_DERIVATION {get;set;}

\_\_DYNASTY Property string \_\_DYNASTY {get;set;}

\_\_GENUS Property int \_\_GENUS {get;set;}

\_\_NAMESPACE Property string \_\_NAMESPACE {get;set;}

\_\_PATH Property string \_\_PATH {get;set;}

\_\_PROPERTY\_COUNT Property int \_\_PROPERTY\_COUNT {get;set;}

\_\_RELPATH Property string \_\_RELPATH {get;set;}

\_\_SERVER Property string \_\_SERVER {get;set;}

\_\_SUPERCLASS Property string \_\_SUPERCLASS {get;set;}

PSStatus PropertySet PSStatus {\_\_PATH, Status}

ConvertFromDateTime ScriptMethod System.Object ConvertFromDateTime();

ConvertToDateTime ScriptMethod System.Object ConvertToDateTime();

InstalledOn ScriptProperty System.Object InstalledOn {get=if ([envir...

**Get-Hotfix | Select-Object –Property InstalledBy | Out-File D:\powershellcommands\netipconfigs -Append**

This command lists all installed hotfixes, selects only the “InstalledBy” field to show who installed them, and appends that information to the specified file so records of the installer are kept for later reference without deleting previous data.

**Output:-**

InstalledBy

-----------

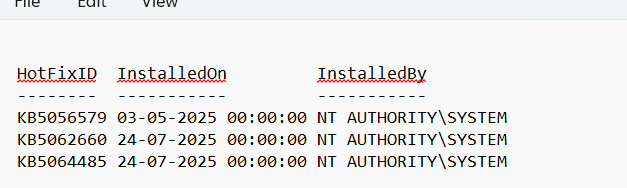
NT AUTHORITY\SYSTEM

NT AUTHORITY\SYSTEM

NT AUTHORITY\SYSTEM

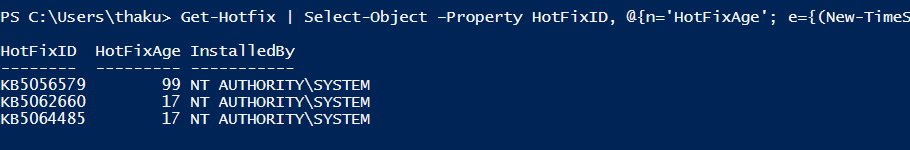
**Get-Hotfix | Select-Object –Property HotFixID, InstalledOn, InstalledBy | Out-File D:\powershellcommands\netipconfigs -Append**

Retrieves all installed hotfixes, selects their ID, installation date, and installer name, and appends this detailed list to the file. This helps maintain a record of what updates were installed, when they were installed, and by whom.



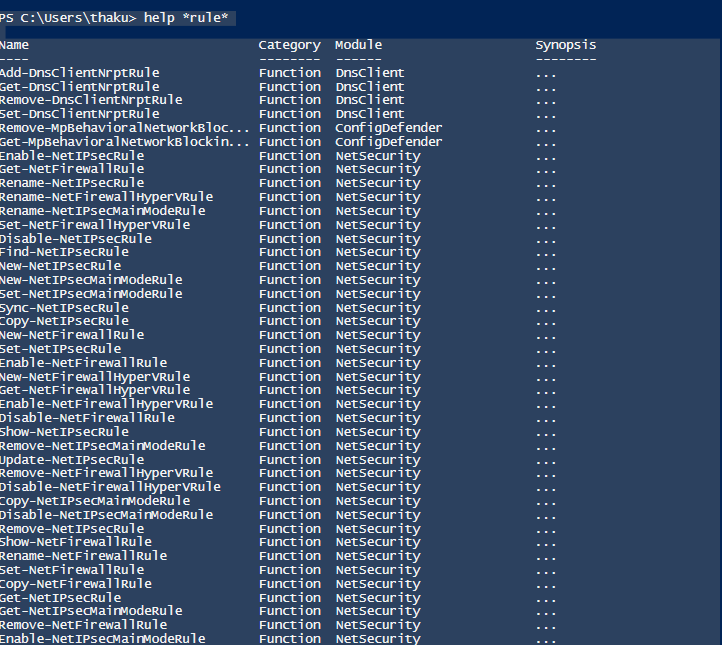
**Get-Hotfix | Select-Object –Property HotFixID, @{n='HotFixAge'; e={(New-TimeSpan –Start $PSItem.InstalledOn).Days}}, InstalledBy**

Gets all hotfixes, calculates the number of days since each was installed (HotFixAge), and displays this along with the hotfix ID and installer. It’s useful for identifying older updates that may no longer be relevant.



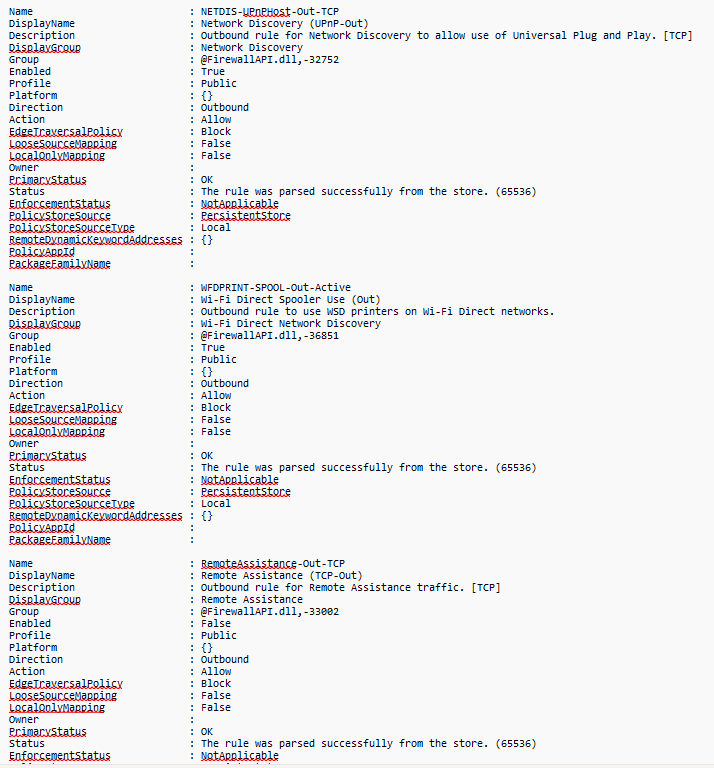
**help \*rule\***

Searches for and displays PowerShell help topics whose names contain the word “rule.” It’s mostly used to learn about available cmdlets related to rules, such as firewall rules, and see their descriptions, syntax, and usage examples.



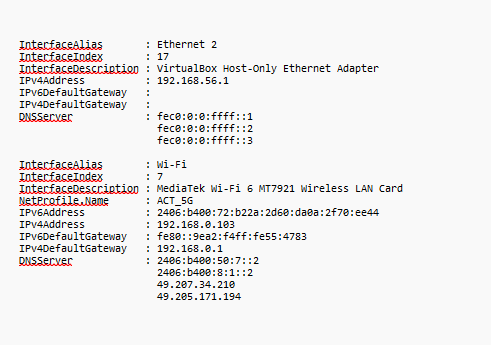
**Get-NetFirewallRule | Out-File D:\powershellcommands\netipconfigs -Append**

Retrieves all current Windows firewall rules configured on the system and appends the list to the file. This is useful for auditing firewall settings, creating backups, or comparing current rules against previous configurations.



**Get-NetIPConfiguration | Out-File D:\powershellcommands\netipconfigs -Append**

Gets the system’s current network configuration, including IP addresses, subnet masks, gateways, and DNS settings, and appends these details to the file for documentation, troubleshooting, or network change tracking purposes over time.

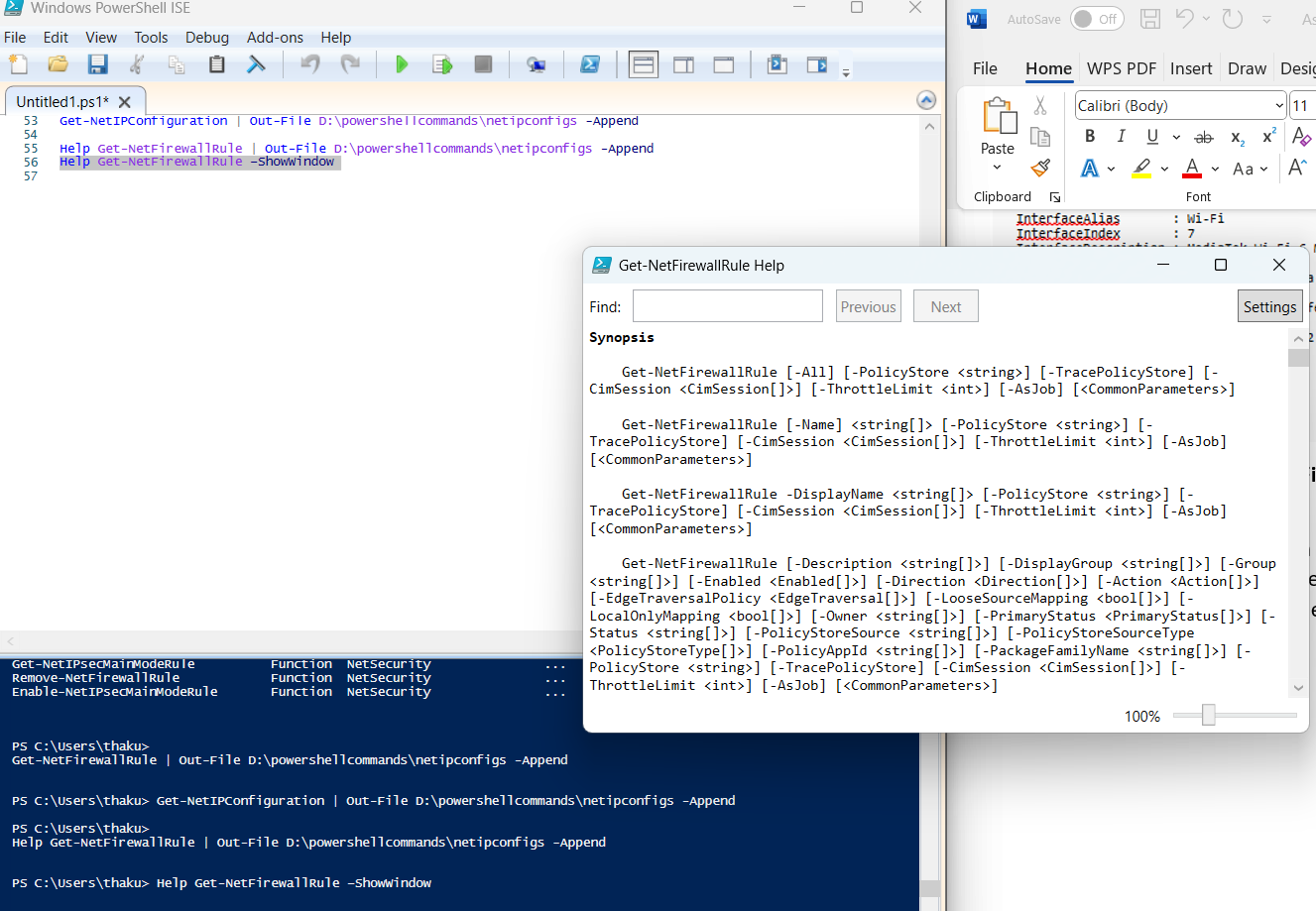


**Help Get-NetFirewallRule | Out-File D:\powershellcommands\netipconfigs -Append**

Retrieves the help documentation for the Get-NetFirewallRule cmdlet, including syntax and examples, then appends it to the file so you can read about the command’s usage offline without querying the system help evetime. 

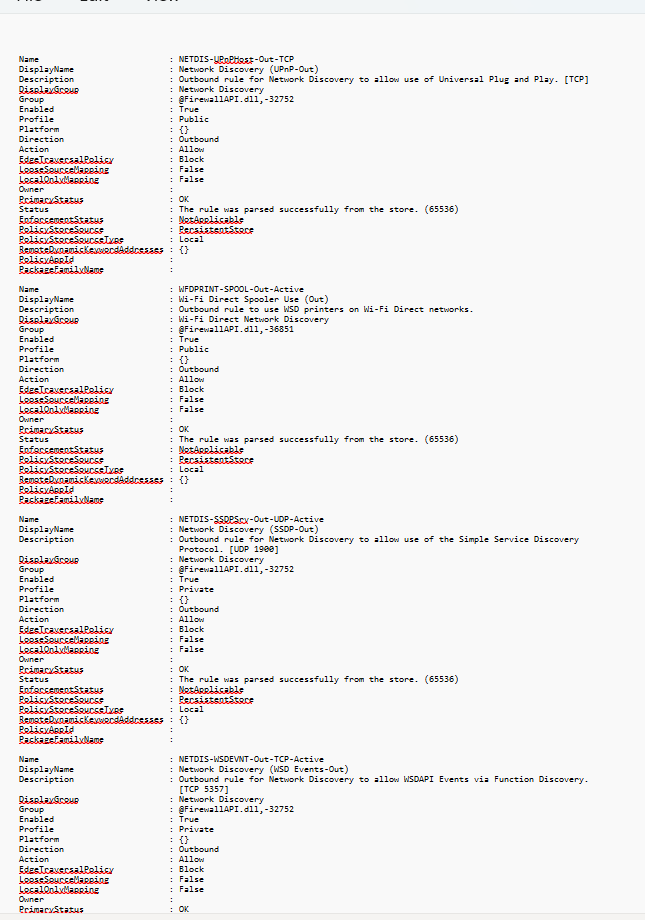
**Help Get-NetFirewallRule**

This command opens detailed help for Get-NetFirewallRule in a separate window. It provides descriptions, syntax, parameters, and examples to help users understand how to retrieve and manage Windows Firewall rules easily and interactively.



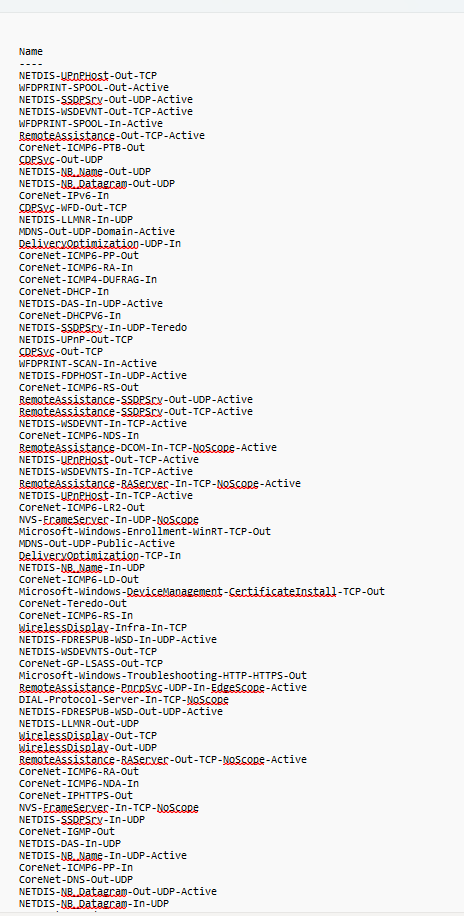
**Get-NetFirewallRule –Enabled True | Out-File D:\powershellcommands\a.txt -Append**

Gets only enabled Windows Firewall rules, then appends the list to the file a.txt without removing existing contents. Useful for logging currently active firewall rules for future review, security auditing, or network troubleshooting purposes.



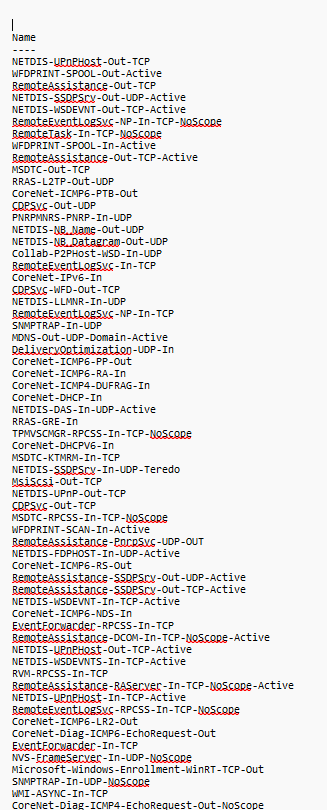
**Get-NetFirewallRule –Enabled True | Format-Table -wrap | Out-File D:\powershellcommands\b.txt -Append**

Retrieves all enabled firewall rules, formats them as a readable table with wrapped text for long fields, and appends the output to b.txt. Helps create a structured, easy-to-read list of active firewall rules for documentation.



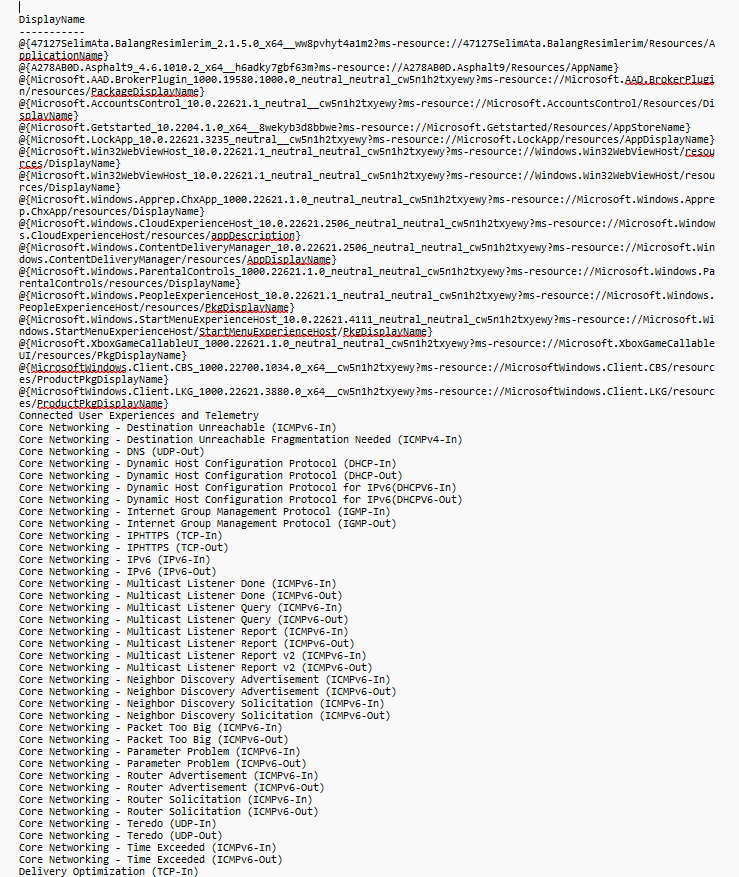
**Get-NetFirewallRule | Format-Table -wrap | Out-File D:\powershellcommands\abc.txt-Append**

Lists all firewall rules (enabled and disabled), formats them in a word‑friendly table with wrapped text, and appends this output to abc.txt. Useful for keeping a complete, well‑formatted firewall configuration record in a shareable document



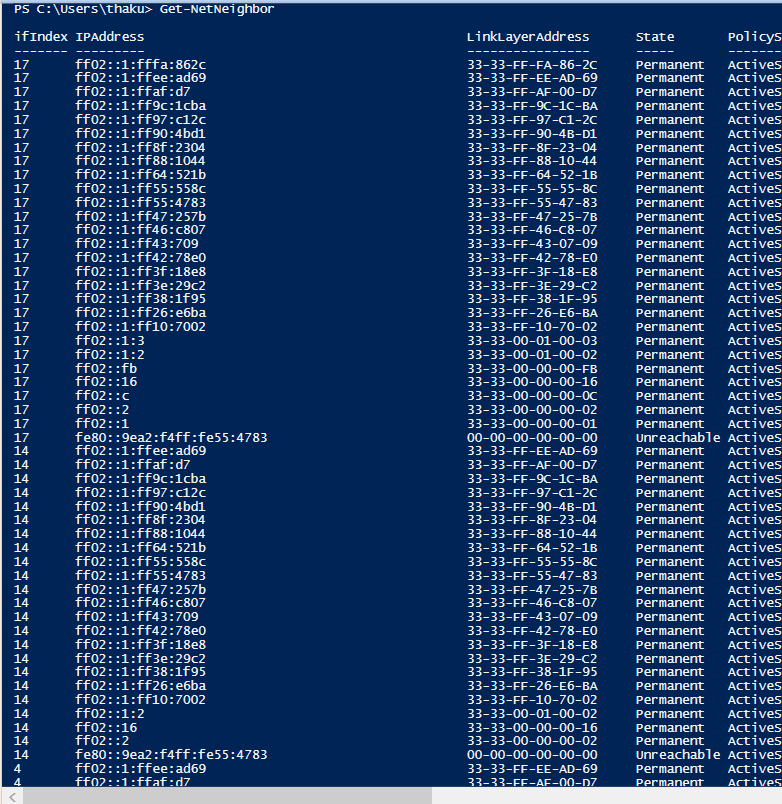
**Get-NetFirewallRule –Enabled True | Select-Object –Property DisplayName,Profile,Direction,Action | Sort-Object –Property Profile, DisplayName | Format-Table -wrap | Out-File D:\powershellcommands\ad.txt -Append**

This command lists all enabled firewall rules, selects their name, profile, direction, and action, sorts them by profile and name, formats them in a wrapped table, and appends this organized, readable output to the specified file for documentation.



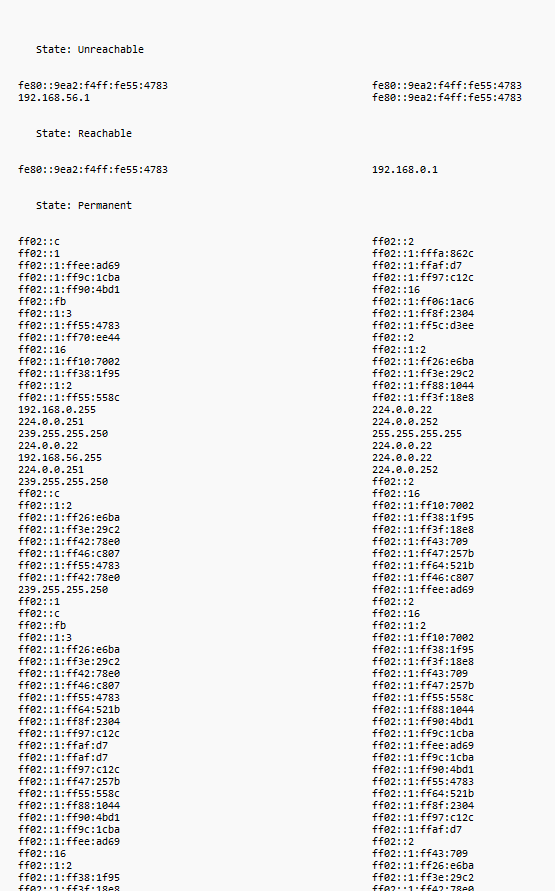
**Get-NetNeighbor**

This command retrieves the neighbor cache entries (ARP cache) on your system. It shows IP addresses and their corresponding physical (MAC) addresses for devices on the local network, useful for network troubleshooting and verifying device connections.



**Get-NetNeighbor | Sort-Object –Property State | Select-Object –Property IPAddress, State | Format-Wide -GroupBy State | Out-File D:\powershellcommands\cd.txt -Append**

This command lists all neighbor cache (ARP) entries, sorts them by connection state, selects only IP address and state, groups and displays them by state in a wide format, and appends the organized output to a specified file for reference.



**100 -gt 10**

**500 -le 100**

**'hello' -eq 'HELLO'**

**'hello' -ceq 'HELLO'**

**Output:-** PS C:\Users\thaku> 100 -gt 10

500 -le 100

'hello' -eq 'HELLO'

'hello' -ceq 'HELLO'

True

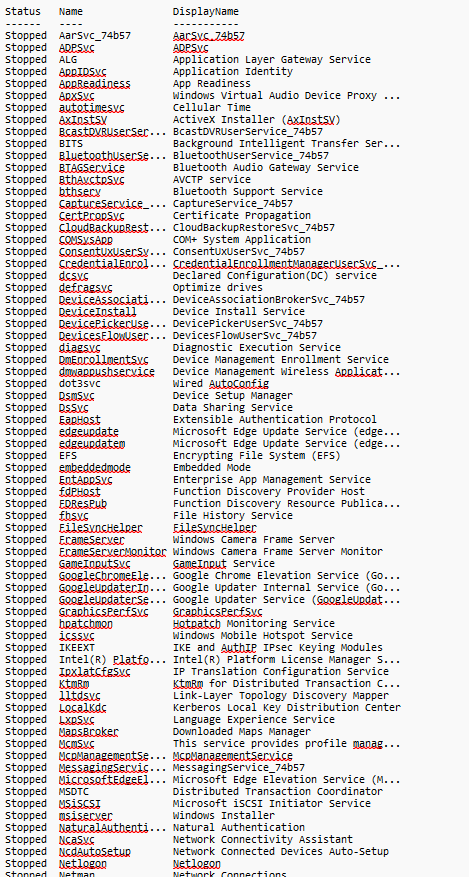
False

True

False

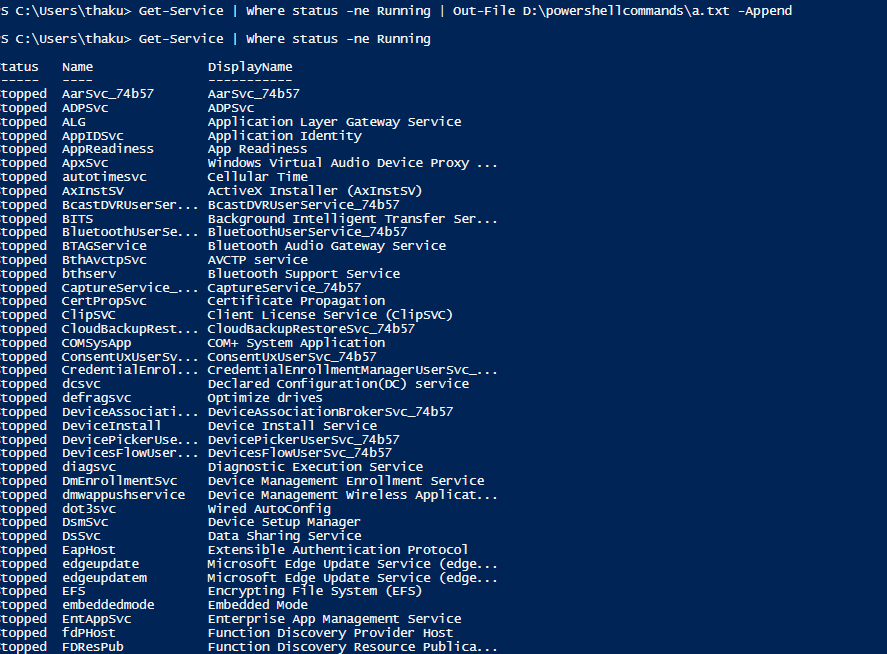
**Get-Service | Where status -ne Running | Out-File D:\powershellcommands\a.txt -Append**

This command finds all services that are not running, then appends this list of stopped or paused services to the specified text file, useful for keeping a log of non-active services for monitoring or troubleshooting.



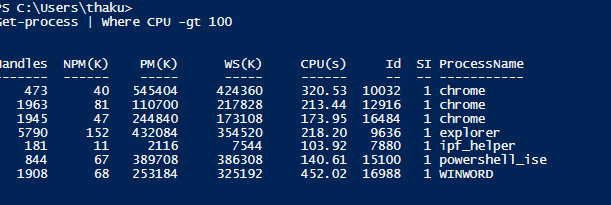
**Get-Service | Where status -ne Running**

Retrieves and shows all services on the system whose status is not "Running," effectively listing services that are stopped or paused. This helps identify which services aren’t currently active without saving the output.



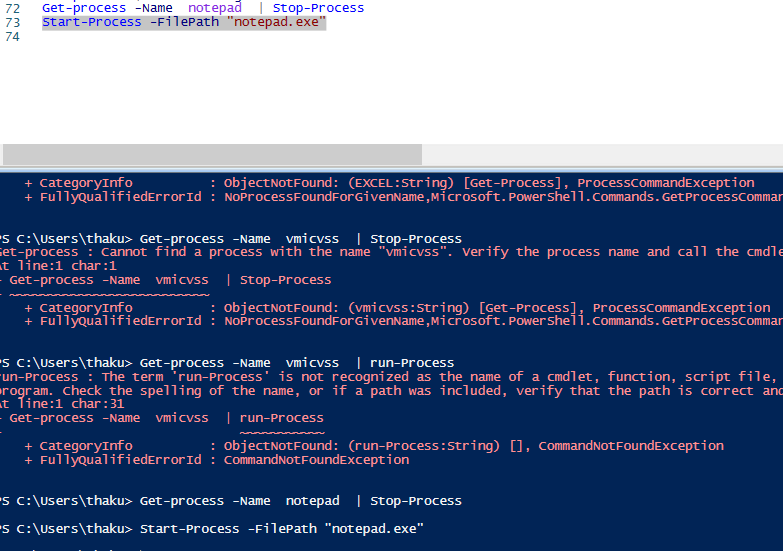
**Get-process | Where CPU -gt 100**

Lists all running processes that are using more than 100 CPU seconds (total processor time), helping to spot processes consuming high CPU resources that might affect system performance.



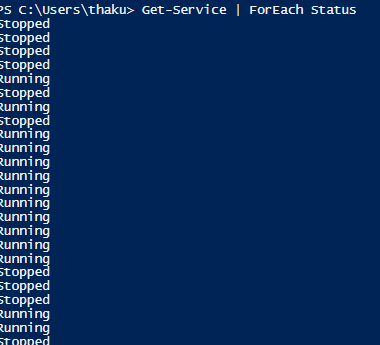
**Get-process -Name notepad | Stop-Process**

Finds all running processes with the name "notepad" and stops them (forcefully closes Excel). Useful to terminate stuck or unwanted Excel applications through PowerShell.



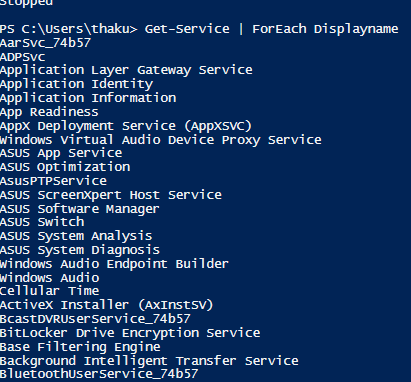
**Get-Service | ForEach Status**

It outputs the status (Running, Stopped, etc.) of each service individually.



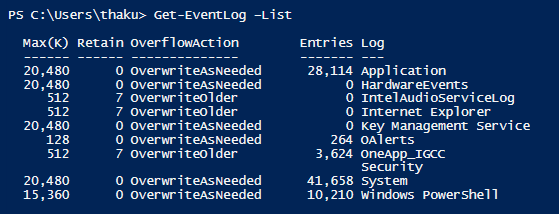
**Get-Service | ForEach Displayname**

This lists the friendly names of all services running or stopped on the system.



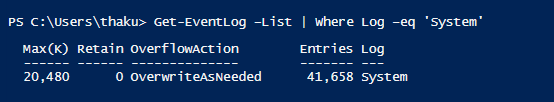
**Get-EventLog –List**

Shows a summary of all event logs present on the computer, including log names, sizes, and retention policies. This helps identify available logs for monitoring and troubleshooting Windows events.



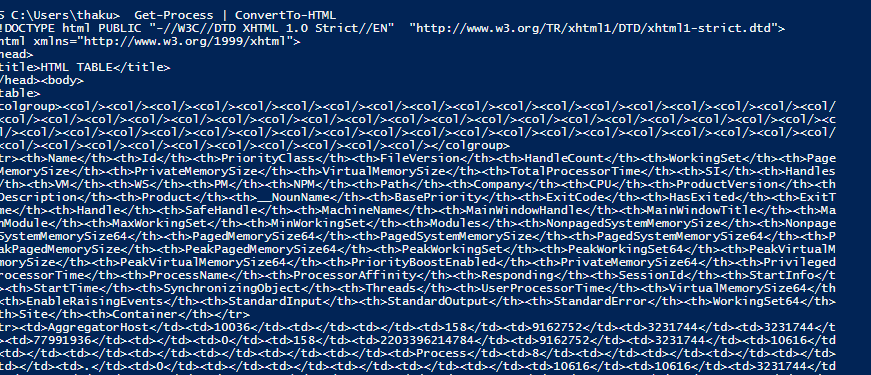
**Get-EventLog –List | Where Log –eq 'System'**

Filters the list of event logs to show only the "System" log, which records system-level events such as driver failures or hardware changes, useful for focused system troubleshooting.



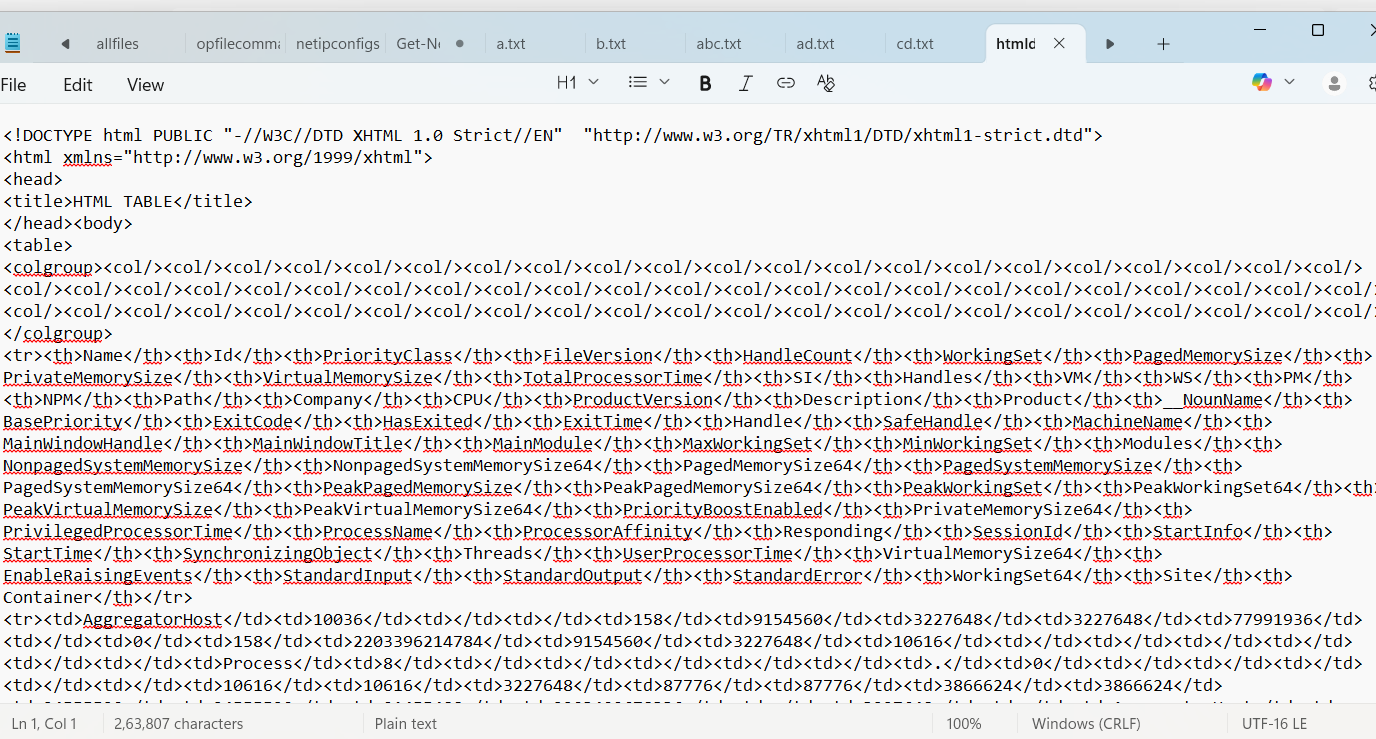
**Get-Process | ConvertTo-HTML**

It convert all process into html .

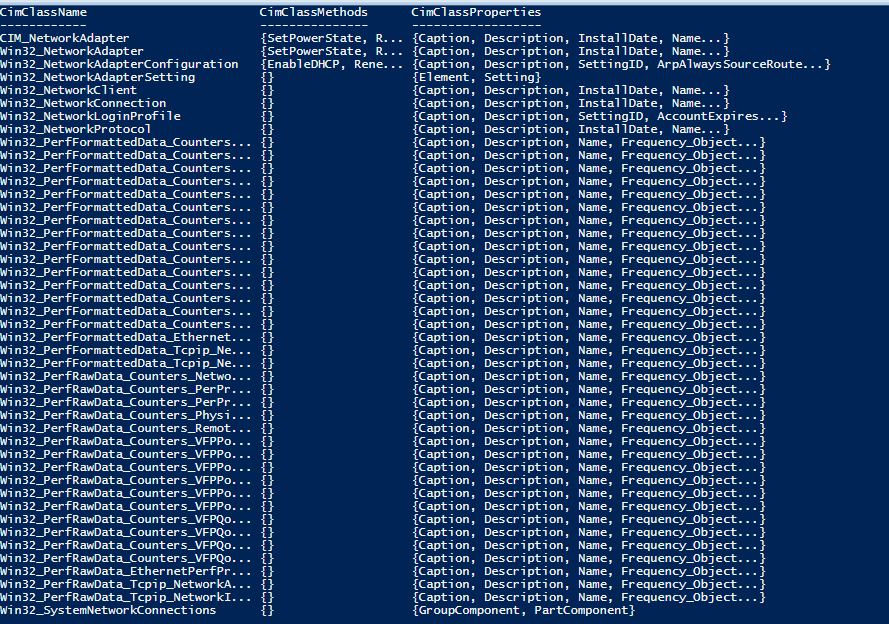


**Get-Process | ConvertTo-HTML | Out-File D:\powershellcommands\htmldata.txt**

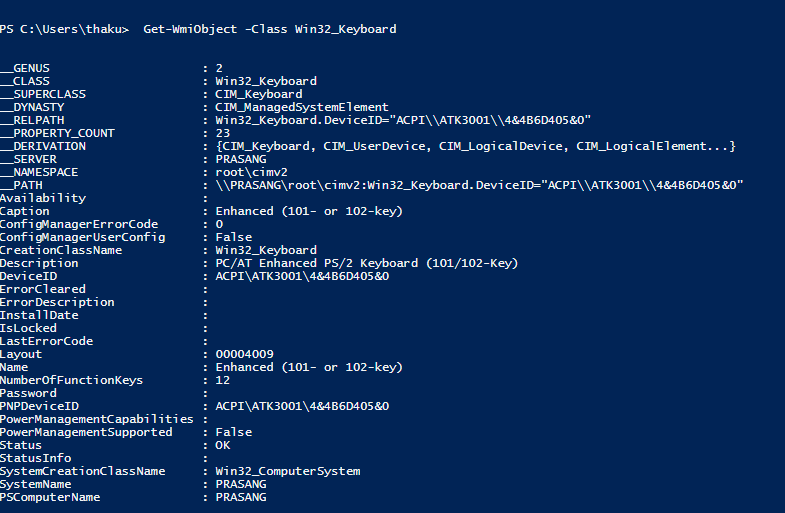
It coverts all process into html and then save it into htmldata.txt file .



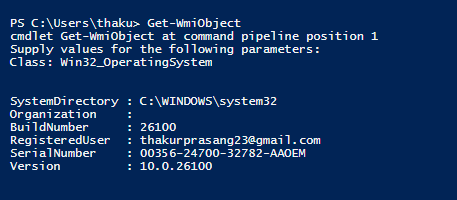
**Get-CimClass \*network\* | Sort-Object CimClassName** This command retrieves CIM classes related to "network" from the system management namespace and sorts them by class name. It helps explore available network-related configuration classes for monitoring, management, or automation purposes.



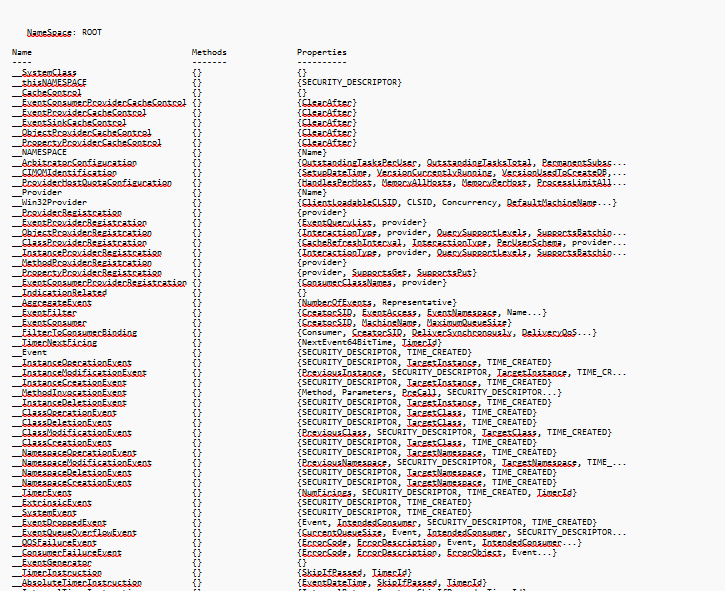
**Get-WmiObject -Class Win32\_Keyboard**  
This command fetches information about the computer's keyboard hardware using WMI. It returns properties like device ID, description, and status for each keyboard installed on the Windows system. Useful for hardware inventory or diagnostic tasks.



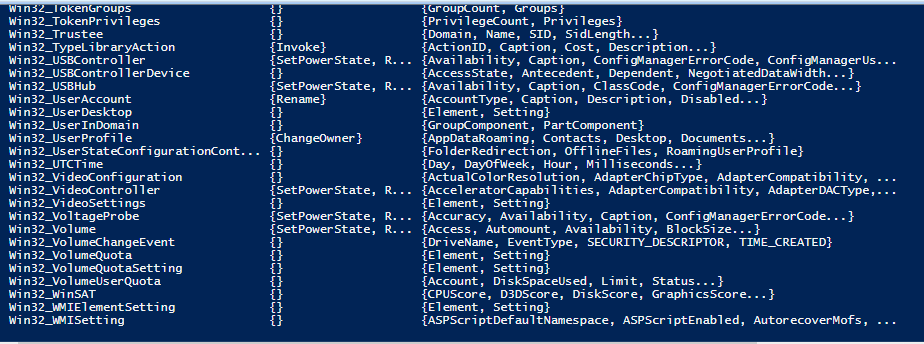
**Get-WmiObject**  
Running without parameters, this lists all WMI classes available in the default namespace (root\cimv2). It helps you discover which classes exist on the system for querying system, hardware, or configuration information.



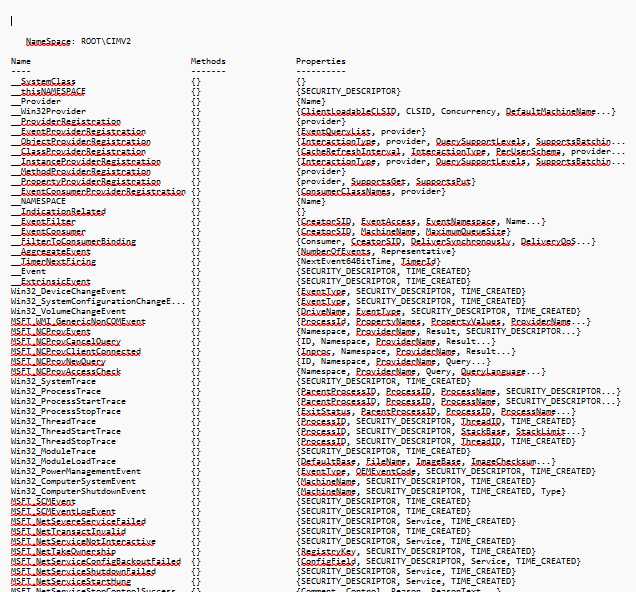
**Get-WmiObject -Namespace root -List | Out-File D:\powershellcommands\Namespace.txt -Append**  
This command lists all available WMI classes in the root namespace and appends them to the given file. It’s useful for documenting the base WMI namespace content without overwriting any existing file data.



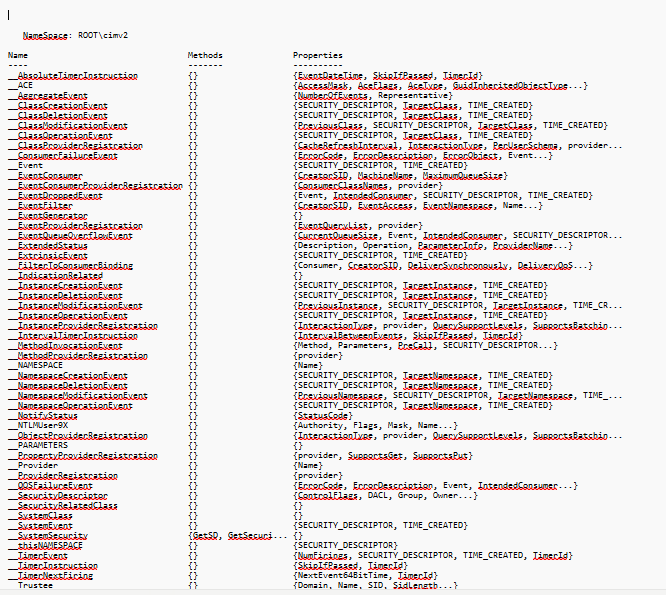
**Get-WmiObject -Namespace root\cimv2 -List | Sort Name**  
This lists all WMI classes in the root\cimv2 namespace, sorts them alphabetically by their name, and displays them in the console. It helps quickly locate specific classes while keeping the list neatly organized for reference.



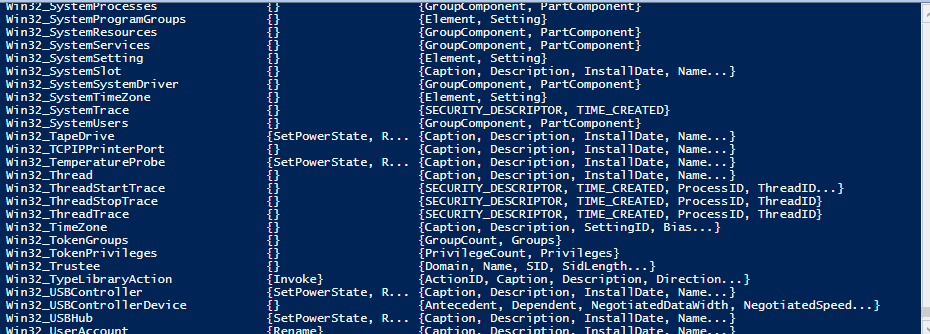
**Get-WmiObject -Namespace root\CIMv2 -List | Out-File D:\powershellcommands\cimv2.txt -Append**  
Retrieves all WMI classes from the root\CIMv2 namespace and appends them to the specified file. This is useful for saving a complete inventory of the CIMv2 namespace for documentation or offline analysis purposes.



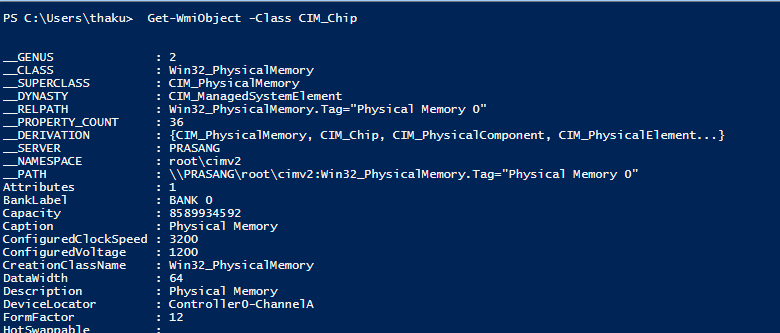
**Get-WmiObject -Namespace root\cimv2 -List | Sort Name | Out-File D:\powershellcommands\sortcimv2.txt -Append**  
Fetches all WMI classes from root\cimv2, sorts them by name alphabetically, and appends them into the specified file. This makes it easy to maintain an alphabetically arranged reference list of CIMv2 classes for later review or administrative tasks.



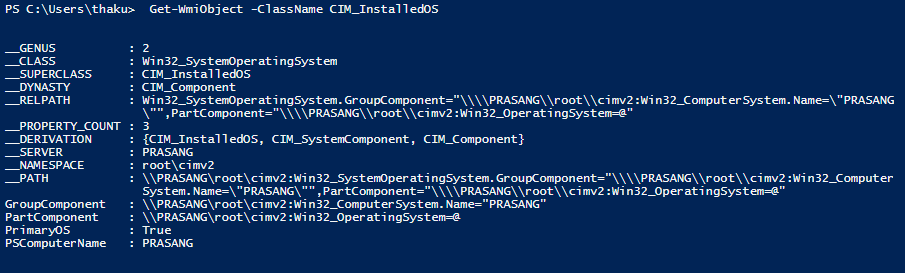
**Get-CimClass -Namespace root\CIMv2 | Sort CimClassName**  
This command retrieves all CIM classes available in the root\CIMv2 namespace and sorts them alphabetically by their class name. It helps explore system classes used for device and configuration management.



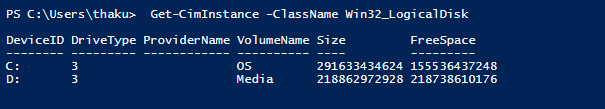
**Get-WmiObject -Class CIM\_Chip**  
This retrieves information about hardware chips on your system like processors and chipsets, using the WMI CIM\_Chip class. It provides details such as manufacturer, version, and device ID for chip-level hardware info.



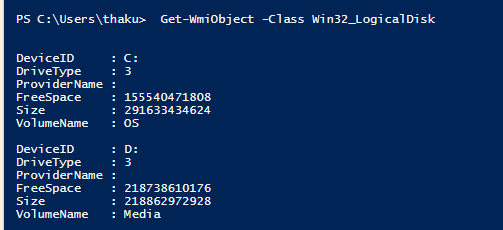
**Get-WmiObject -ClassName CIM\_InstalledOS**  
This gets information about the installed operating system on the computer by querying the WMI class CIM\_InstalledOS, including OS version, manufacturer, and installation details.



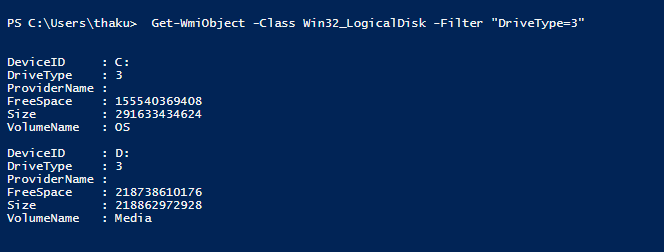
**Get-CimInstance -ClassName Win32\_LogicalDisk**  
Retrieves instances of all logical disks (like hard drives and partitions) on the system using CIM cmdlets. It provides data such as device ID, size, free space, and drive type.



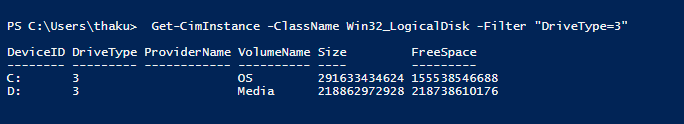
**Get-WmiObject -Class Win32\_LogicalDisk**  
Performs a similar query using WMI to get all logical disk instances with properties like disk size, free space, and drive letter. Useful for disk and storage management.



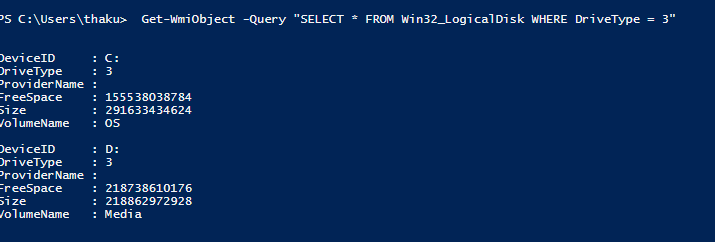
**Get-WmiObject -Class Win32\_LogicalDisk -Filter "DriveType=3"**  
This filters the logical disks retrieved to show only fixed local drives (DriveType=3). It excludes removable drives or network shares, showing only internal hard disks.



**Get-CimInstance -ClassName Win32\_LogicalDisk -Filter "DriveType=3"**Using CIM with a filter for DriveType=3” this command lists all local fixed drives on the system with details such as size and free space.

****

**Get-WmiObject -Query "SELECT \* FROM Win32\_LogicalDisk WHERE DriveType = 3"**  
Runs a WMI query to select all properties from Win32\_LogicalDisk where the drive type is fixed disk (3), equivalent to the previous filter commands.



**Get-CimInstance -Query "SELECT \* FROM Win32\_LogicalDisk WHERE DriveType = 3"**  
Executes a CIM query returning all fixed local disks with their properties, similar to the WMI query but using the newer CIM cmdlet interface.

