***Assignment-4***

***Name- Harsh Kumar***

***Email –*** [***hs150harsh@gmail.com***](mailto:hs150harsh@gmail.com)

***Introducing to Cmdlets***

cmdlet -- pronounced **command-let** -- is a small, ***lightweight command*** that is used in the ***Windows PowerShell environment***. A cmdlet typically exists as ***a small script*** that is intended to perform a ***single specific function*** such as coping files and changing directories. A cmdlet and its relevant parameters can be entered in a PowerShell command line for immediate execution or included as part of a longer PowerShell script that can be executed as desired.

***Cmdlet*** employ a ***verb***/***noun*** ***naming*** ***pattern*** that is designed to make each cmdlet easier to remember and read.

***Ex- Get-Process , Get-Service etc***.**>>>>>>>>>**

***A screenshot of a computer

AI-generated content may be incorrect.***Cmdlets are ***based on .NET classes*** and rely on the use of .NET objects. Thus, cmdlets can receive ***objects as input*** ***and deliver objects as output***, which can then feed the input of subsequent objects, enabling cmdlets to form a command pipeline.

***Key Characteristics of Cmdlets:***

* *Verb-Noun Naming*
* *Pipeline-Oriented*
* *Object****-****Oriented*
* *Built-in and Custom*

***The PowerShell Pipeline***

The PowerShell pipeline is a sequence of commands connected by the ***pipeline operator |.***

This ***operator allows*** the ***output of one command*** (a cmdlet, function, or script) to be ***passed as input to the next command*** in the pipeline. Unlike traditional command-line pipelines that handle text, PowerShell pipelines primarily work with objects.

A screenshot of a computer

AI-generated content may be incorrect.

***Functionality or how does pipelining work***

*Here’s some of the key concept of pipelining*

***Object-Based Flow***

***Real-Time Processing***

***Chaining Commands***

***Pipeline-Chain Operators***

***Key Cmdlets***

Here’s some of the key **PowerShell cmdlet**

* **Get-Process** – Shows all running processes on your system.
* **Get-Service** – Lists all services and their status.
* **Stop-Service** – Stops a running service.
* **Start-Service** – Starts a stopped service.
* **Get-EventLog** – Views events from Windows logs.
* **Get-Item** – Shows details of a specific file or folder.
* **Set-Location** – Changes the current working directory.
* **Copy-Item** – Copies a file or folder to another location.
* **Move-Item** – Moves a file or folder to another location.
* **Remove-Item** – Deletes a file or folder.
* **Rename-Item** – Renames a file or folder.

***WMI & PowerShell***

**Windows Management Instrumentation (WMI)** is a Microsoft framework designed to manage and monitor Windows-based operating systems. It provides a unified interface for accessing system information, managing devices, and automating administrative tasks. WMI is widely used by IT professionals and developers to interact with system components programmatically.

***Pipeline Filtering***

PowerShell's pipeline allows for sequential command execution, where the output of one command serves as the input for the next. Filtering and operators are crucial for manipulating data within the pipeline.

Ex:- Get-Process | Where-Object {$\_.CPU -gt 1}

A screenshot of a computer program

AI-generated content may be incorrect.

***Operators***

1. ***Conditional Operators***

PowerShell offers various comparison operators for use in filtering:

● -eq: equals

● -ne: not equals

● -gt: greater than

● -ge: greater than or equals

● -lt: less than

● -le: less than or equals

● -like: wildcard matching

● -notlike: not wildcard matching

● -match: regular expression matching

● -notmatch: not regular expression matching

1. ***Logical Operators***

Multiple conditions can be combined using logical operators:

● -and: Both conditions must be true.

● -or: At least one condition must be true.

● -not or !: Negates the condition.

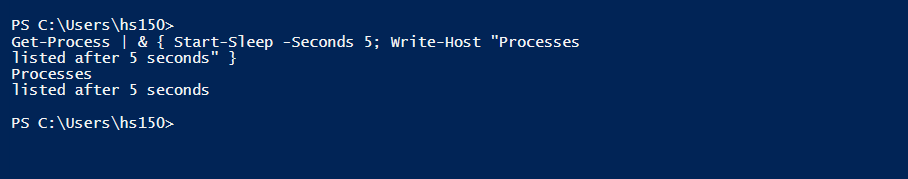
Ex:- Get-Process | Where-Object {$\_.CPU -gt 1 -and $\_.WorkingSet -gt 10MB}

A computer screen shot of a blue screen

AI-generated content may be incorrect.

PowerShell 7 introduced pipeline chain operators for conditional execution:

Ex:- Get-Process | & { Start-Sleep -Seconds 5; Write-Host "Processes listed after 5 seconds" }



***Input, Output & Formatting***

PowerShell handles input and output through several cmdlets and techniques, including reading from the console, writing to the console, and formatting output for readability.

***Input***

● **Read**-**Host**: This cmdlet prompts the user for input and stores it as a string.

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

● **Get**-**Content**: Reads the content of a file.

$content = Get-Content -Path "C:\example.log"

***Output***

* **Write**-**Host**: Displays output directly to the console.

Write-Host "This is a message."

* **Out**-**File**: Writes output to a file.

Get-Process | Out-File -FilePath "processes.txt"

***Formatting***

* **Format-Table**: Displays output in a table format.

Get-Process | Format-Table -Property Name, CPU, StartTime

* **Format-List**: Displays output as a list of properties.

Get-Service | Format-List -Property Name, Status, DisplayName

* **Format-Wide**: Displays output in a wide format, showing only one property per line.

Get-ChildItem | Format-Wide -Column 3

***String Formatting***

* **The** **-f format operator**: Allows for composite formatting, similar to string interpolation in other languages.

$name = "John" $age = 30 "My name is {0} and I am {1} years old" -f $name, $age

***Pipelines***

PowerShell uses pipelines (|) to pass the output of one command as input to the next. This is fundamental for processing and formatting data.

Get-Process | Sort-Object CPU -Descending | Select-Object -First 5 | Format-Table Name, CPU

***Scripting Overview***

**PowerShell scripting constructs** are fundamental elements that control the flow and logic of scripts. These constructs enable automation, decision-making, and repetition of tasks. Here's a breakdown of key scripting constructs in PowerShell:

* **Variables**: Used to store data values. They are defined using a $ prefix, e.g.,

$name = "John"

* **Arrays**: Ordered collections of items, accessed by index.  
  Example: $fruits = @("apple", "banana", "orange")
* **Operators**: Symbols that perform operations on values:
  + **Arithmetic**: +, -, \*, /, %
  + **Comparison**: -eq, -ne, -gt, -lt, -ge, -le
  + **Logical**: -and, -or, -not
* **Conditional Statements**:

**if, elseif, else**: Execute code blocks based on conditions.

**$age = 25**

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

**Write-Host "Adult"**

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

**Write-Host "Teenager"**

**} else {**

**Write-Host "Child"**

**}**

**switch: Efficiently handle multiple conditions.**

**$day = "Monday"**

**switch ($day) {**

**"Monday" { Write-Host "Start of the week" }**

**"Friday" { Write-Host "End of the week" }**

**default { Write-Host "Mid-week" }**

**}**

* **Looping Statements**:

**for: Iterate a specific number of times.**

**for ($i = 1; $i -le 5; $i++) {**

**Write-Host "Iteration: $i"**

**}**

**foreach: Iterate through a collection.**

**$colors = @("red", "green", "blue")**

**foreach ($color in $colors) {**

**Write-Host "Color: $color"**

**}**

**while: Repeat as long as a condition is true.**

**$count = 0**

**while ($count -lt 3) {**

**Write-Host "Count: $count"**

**$count++**

**}**

**do-while and do-until: Similar to while, but the condition is checked at the end**.

* **Functions**: Reusable blocks of code.

**function Greet {**

**param($name)**

**Write-Host "Hello, $name!"**

**}**

Greet -name "Alice"

* **Script Blocks**: Collections of statements treated as a single unit, often used with cmdlets like ForEach-Object.
* **Modules**: Groupings of PowerShell functionalities (functions, scripts, etc.) that can be imported and used in other scripts.
* **Error Handling**: Use try, catch, finally blocks to manage exceptions.

**try {**

**Get-Content "nonexistent\_file.txt" -ErrorAction Stop**

**} catch {**

**Write-Host "Error: $($\_.Exception.Message)"**

**} finally {**

**Write-Host "Cleanup actions"**

**}**

* **Comments**: Use # to add explanatory notes in your code.