Microsoft

Pipeline Advanced

Learnings covered in this Unit

- Pipeline variables
- Filtering on the pipeline
- Looping elements in the pipeline
- Pipeline input

Pipeline Variable

Pipeline Variable Overview

- Represents the **current** object on the pipeline
- Used perform an action on **every** object
- Used with cmdlets like **Foreach-Object** and **Where-Object**
- **$_** and **$PSItem**
- Use **-PipelineVariable** parameter to name your own variable on pipeline
- Scoped only to **current** pipeline

Object Cmdlets

ForEach-Object

- Performs an **operation** against **each object** on the pipeline
- Aliases: **%** and **ForEach**

Where-Object

- **Filters** objects in pipeline using a **script block** to check **conditions**
- Aliases: **?** and **Where**

Where-Object

Where-Object Filtering

Script block needs to return **True** or **False**

$_ allows accessing **properties** or **methods**

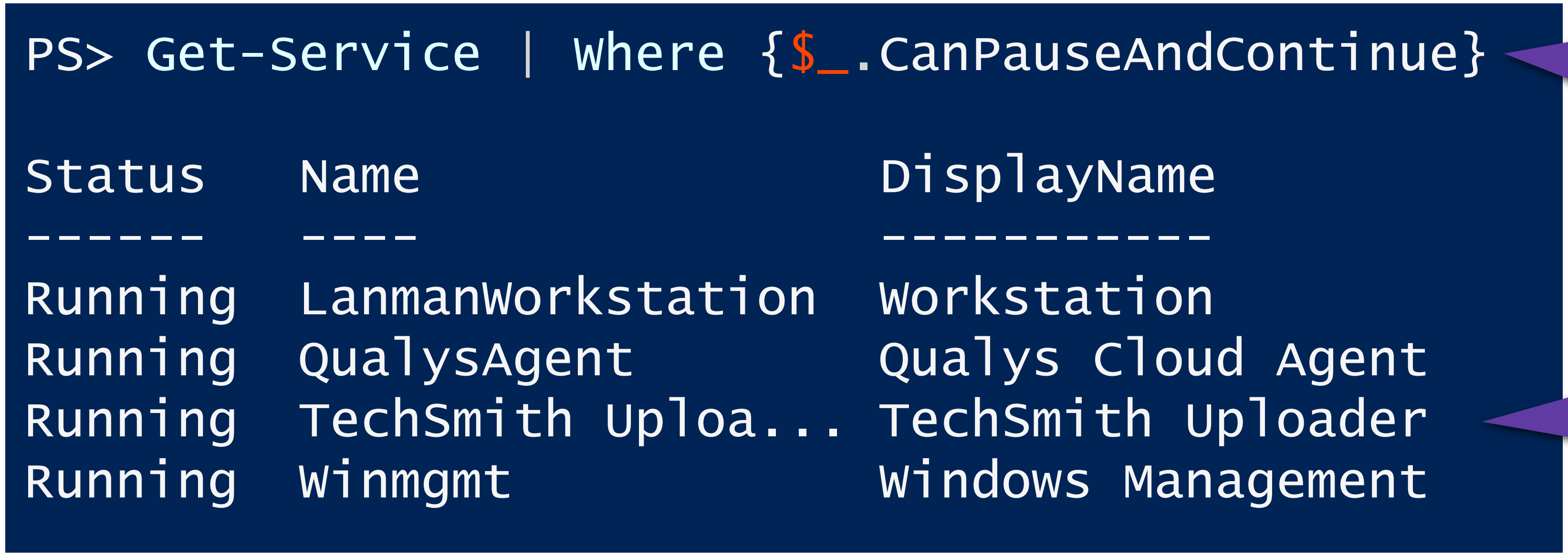
Comparison and **Logical Operators** are generally used

Any value except **$False**, **$Null**, and **0** considered True

Where-Object Basics

Filters objects on pipeline using a **script block** to check **conditions**

Aliases: **?** and **Where**



```
PS> Get-Service | Where {$_.CanPauseAndContinue}

Status   Name                DisplayName
------   ----                -----------
Running  LanmanWorkstation   Workstation
Running  QualysAgent         Qualys Cloud Agent
Running  TechSmith Uploa...  TechSmith Uploader
Running  Winmgmt             Windows Management
```

Boolean property is already **True** or **False**

4 services returned instead of all 300

Comparison Operators

	Case Insensitive	Case Sensitive
Equal	-eq	-ceq
Not Equal	-ne	-cne
Greater Than	-gt	-cgt
Greater Than or Equal To	-ge	-cge
Less Than	-lt	-clt
Less Than or Equal To	-le	-cle

No Wildcards

	Case Insensitive	Case Sensitive
Equal With Wildcard	-like	-clike
Not Equal With Wildcard	-notlike	-cnotlike

Wildcards

More comparison operators will appear in other sections

Basic Comparison Examples

```
PS> "This" -eq "That"
False

PS> "This" -eq "This"
True

PS> "This" -eq "Th*"
False

#Wildcard must be on right
PS> "This" -like "Th*"
True

PS> "This" -like "That"
False

PS> "This" -notlike "That"
True
```

```
PS> 5 -gt 3
True

PS> 5 -gt 5
False

PS> 5 -ge 5
True

#Case Insensitive
PS> "This" -eq "this"
True

#Case Sensitive
PS> "This" -ceq "this"
False
```

Where-Object Using Comparisons

Use pipeline variable: **$_** or **$PSItem**

Compare **properties** or **methods** output to other **values**

```
PS> Get-Service | Where-Object {$_.StartType -eq "Disabled"}

 Status   Name                DisplayName
------   ----                -----------
Stopped  AppVClient          Microsoft App-V Client
Stopped  NetTcpPortSharing   Net.Tcp Port Sharing Service
Stopped  RemoteAccess        Routing and Remote Access
Stopped  RemoteRegistry      Remote Registry
Stopped  shpamsvc            Shared PC Account Manager
Stopped  ssh-agent           OpenSSH Authentication Agent
Stopped  tzautoupdate        Auto Time Zone Updater
Stopped  UevAgentService     User Experience Virtualization Service
```

Logical Operators – Basic – 1

Join **multiple comparisons** together into **compound conditions**

Operator	Description
-and	TRUE only when both statements are TRUE
-or	TRUE when either or both statements are TRUE
-xor	TRUE only when one of the statements is TRUE and the other is FALSE
-not	Prepended - Toggles the statement TRUE to FALSE or vice versa
!	Same as -not

Logical Operators – Basic – 2

Join **multiple comparisons** together into **compound conditions**

Operator	Description
-and	TRUE only when **both statements** are TRUE
-or	TRUE when **either** or both statements are TRUE
-xor	TRUE only when **one** of the statements is **TRUE** and the **other** is **FALSE**
-not	**Prepended - Toggles** the statement TRUE to FALSE or vice versa
!	Same as -not

```
PS> ("This" -eq "This") -and ("That" -eq "That")
True

PS> ("This" -eq "This") -and ("That" -eq "NO GOOD")
False
```

Logical Operators – Basic – 3

Join **multiple comparisons** together into **compound conditions**

Operator	Description
-and	TRUE only when **both statements** are TRUE
-or	TRUE when **either** or both statements are TRUE
-xor	TRUE only when **one** of the statements is **TRUE** and the **other** is **FALSE**
-not	**Prepended - Toggles** the statement TRUE to FALSE or vice versa
!	Same as -not

```
PS> ("This" -eq "This") -or ("That" -eq "That")
True

PS> ("This" -eq "This") -or ("That" -eq "NO GOOD")
True
```

Logical Operators – Basic – 4

Join **multiple comparisons** together into **compound conditions**

Operator	Description
-and	TRUE only when **both statements** are TRUE
-or	TRUE when **either** or both statements are TRUE
-xor	TRUE only when **one** of the statements is **TRUE** and the **other** is **FALSE**
-not	**Prepended - Toggles** the statement TRUE to FALSE or vice versa
!	Same as -not

```
PS> ("This" -eq "This") -xor ("That" -eq "That")
False
PS> ("This" -eq "This") -xor ("That" -eq "NO GOOD")
True
PS> ("This" -eq "NO GOOD") -xor ("That" -eq "NO GOOD")
False
```

Logical Operators – Basic – 5

Join **multiple comparisons** together into **compound conditions**

Operator	Description
-and	TRUE only when **both statements** are TRUE
-or	TRUE when **either** or both statements are TRUE
-xor	TRUE only when **one** of the statements is **TRUE** and the **other** is **FALSE**
-not	**Prepended - Toggles** the statement TRUE to FALSE or vice versa
!	Same as -not

```
PS> -not("This" -eq "This")
False

PS> !("This" -eq "NO GOOD")
True
```

Where-Object Using Logical Operators

```
PS> Get-Service | Where-Object {$_.StartType -eq "Disabled"}

Status   Name                DisplayName
------   ----                -----------
Stopped  AppVClient          Microsoft App-V Client
Stopped  NetTcpPortSharing   Net.Tcp Port Sharing Service
Stopped  RemoteAccess        Routing and Remote Access
Stopped  RemoteRegistry      Remote Registry
Stopped  shpamsvc            Shared PC Account Manager
Stopped  ssh-agent           OpenSSH Authentication Agent
Stopped  tzautoupdate        Auto Time Zone Updater
Stopped  UevAgentService     User Experience Virtualization Service

PS> Get-Service |
       Where-Object {$_.StartType -eq "Disabled" -and $_.Name -like "r*"}

Status   Name                DisplayName
------   ----                -----------
Stopped  RemoteAccess        Routing and Remote Access
Stopped  RemoteRegistry      Remote Registry
```

Where-Object Simple Syntax

Shortcut for **simple** comparisons

PowerShell **v3.0**+

Compound conditions need **full syntax**

Full syntax

```
PS> Get-Service | Where-Object {$_.Status -eq "Running"}
```

Simple syntax

```
PS> Get-Service | Where Status -eq Running
```

Full syntax needed for compound conditions

```
PS> Get-Service | Where-Object {$_.Status -eq "Running" -and $_.CanStop}
```

Filtering with Parameters vs. Where-Object

If a cmdlet has a **parameter** to **filter** upon, it is usually **optimized**

Where-Object is a great **backup**, but always check the cmdlet's parameters first

Observable with **large data sets**, but negligible with small data sets

Filter output with Where-Object (~11 milliseconds)

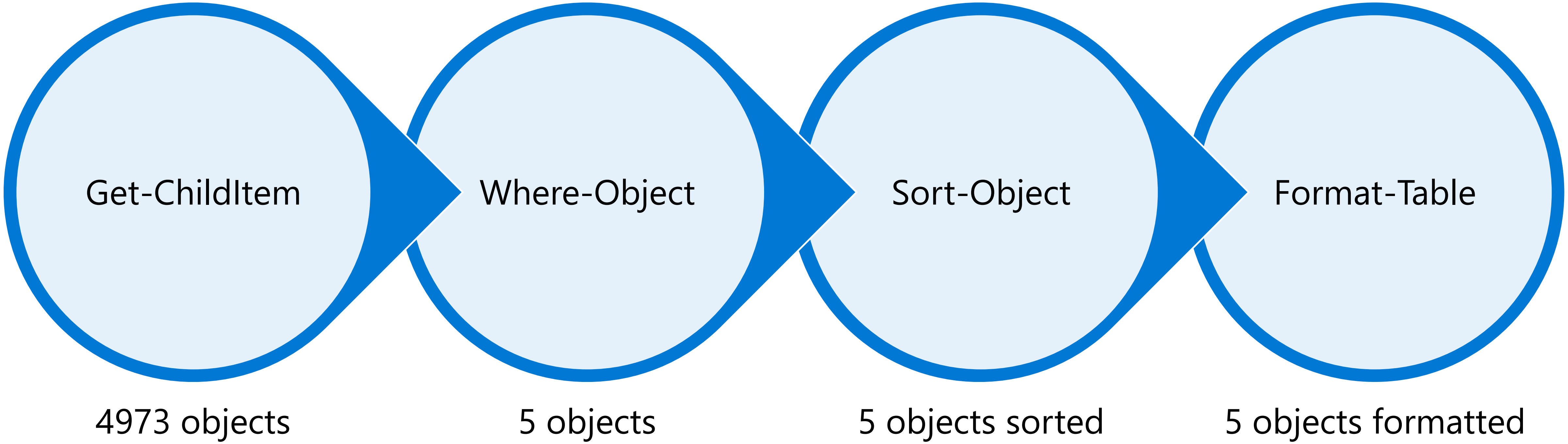
```
PS> Get-Process | Where-Object {$_.Name -eq "explorer"}
```

Filter output with parameters (~4 milliseconds)

```
PS> Get-Process -Name explorer
```

Piping

```
PS>  Get-ChildItem -Path C:\Windows\System32 |
       Where-Object Length -gt 50mb |
         Sort-Object Length |
             Format-Table Name,Length
```



Demonstration
Pipeline Variable
Where-Object
Operators

- Pipeline Variable
- Where-Object
- Operators

Foreach-Object

Foreach-Object Basics

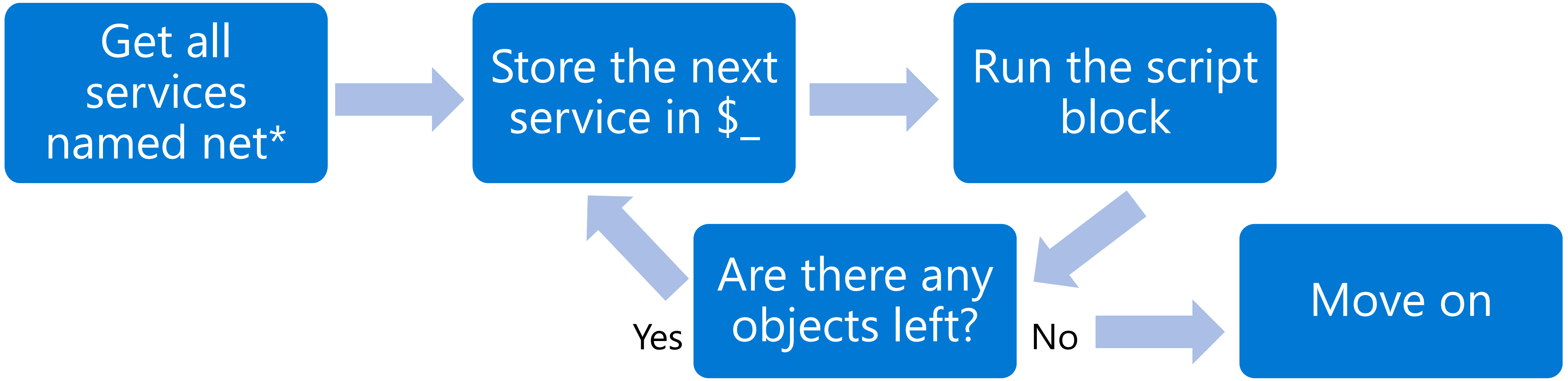
Performs an **action** to **every** object on the pipeline using a **script block**

Aliases: **%** and **Foreach**

Script block can perform **any amount** of code and be **saved** into a **variable**

$_ allows accessing **properties** or **methods**

Foreach-Object


Get all services named net*
Store the next service in $_
Run the script block
Are there any objects left?
Yes
No
Move on

```
PS> Get-Service net* | ForEach-Object {"Hello " + $_.Name}

Hello Netlogon
Hello Netman
Hello netprofm
Hello NetTcpPortSharing
```

Automatic Member Enumeration

Retrieve **single property** from collection **without** using ForEach-Object

```
PS> (Get-Process).ID
4300
8844
8812
```

Multiple levels deep

```
PS> (Get-EventLog -Log System).TimeWritten.DayOfWeek | Group-Object

Count Name      Group
----- ----      -----
 4174 Tuesday   {Tuesday, Tuesday, Tuesday...}
 4349 Monday    {Monday, Monday, Monday...}
```

Foreach-Object Example: Active Directory

The .. operator will return each integer between the two values

Each integer is passed through the pipeline to ForEach-Object

ForEach-Object will use the **$_** variable to represent each integer in the following commands

```
PS> 1..100 | ForEach-Object {
    New-ADUser  -Name User$_ `
      -Organization "contoso.com/Accounts" `
      -UserPrincipalName "User$_@contoso.com" `
      -emailaddress "User$_@contoso.com" `
      -ChangePasswordAtLogon $true
}
```

Demonstration
For-Each Object

- Foreach-Object
- Automatic Enumeration

Pipeline Processing with Foreach and Functions

Foreach-Object -Process Parameter

ForEach-Object is often used with a positional parameter in simple scenarios

Other parameters exist for specialized processing

```
PS C:\> Get-EventLog -LogName Application -Newest 5 |
ForEach-Object {$_.Message | Out-File -Filepath Events.txt -Append}
```

Position 1 is -Process Parameter

```
PS C:\> Get-EventLog -LogName Application -Newest 5 |
ForEach-Object -Process {$_.Message | Out-File Events.txt -Append}
```

-Process parameter can be named

Parameters – Begin

- ForEach-Object cmdlet supports Begin, Process, and End Parameters

- Begin block → run once before any items are processed
- Process block → run for each object on pipeline
- End block → run once after all items have been processed

```
PS C:\> Get-EventLog -LogName Application -Newest 5 |
ForEach-Object
-Begin {Remove-Item .\Events.txt; Write-Host "Start" -ForegroundColor Yellow}
-Process {$_.Message | Out-File -Filepath Events.txt -Append}
-End {Write-Host "End" -ForegroundColor Green; notepad.exe Events.txt}
```

Parameters – Process

- ForEach-Object cmdlet supports Begin, Process and End Parameters

- Begin block → run once before any items are processed
- Process block → run for each object on pipeline
- End block → run once after all items have been processed

```
PS C:\> Get-EventLog -LogName Application -Newest 5 |
ForEach-Object
-Begin {Remove-Item .\Events.txt; Write-Host "Start" -ForegroundColor Yellow}
-Process {$_.Message | Out-File -Filepath Events.txt -Append}
-End {Write-Host "End" -ForegroundColor Green; notepad.exe Events.txt}
```

Parameters – End

- ForEach-Object cmdlet supports Begin, Process and End Parameters

- Begin block → run once before any items are processed
- Process block → run for each object on pipeline
- End block → run once after all items have been processed

```
PS C:\> Get-EventLog -LogName Application -Newest 5 |
ForEach-Object
-Begin {Remove-Item .\Events.txt; Write-Host "Start" -ForegroundColor Yellow}
-Process {$_.Message | Out-File -Filepath Events.txt -Append}
-End {Write-Host "End" -ForegroundColor Green; notepad.exe Events.txt}
```

Named Blocks in Functions/ScriptBlocks

Optional named blocks in a function

- Allows for **processing** collections from the pipeline
- Can be defined in **any** order

Begin Block

- Statements executed **once**, **before** first pipeline object

Process Block

- Statements **executed** for **each** pipeline **object** delivered, leveraging **$_**
- If called outside a pipeline context, block is executed exactly once
- Becomes more common and **useful** with **Advanced Functions**

End block

- Statements executed **once**, **after** last pipeline object

Named Blocks in Function

```
function My-Function
{
    Begin
    {
        Remove-Item .\Events.txt
        Write-Host "Start" -ForegroundColor Red
    }
    Process
    {
        $_.Message | Out-File -Filepath Events.txt -Append
    }
    End
    {
        Write-Host "End" -ForegroundColor Green
        notepad.exe Events.txt
    }
}
```

```
PS> Get-EventLog -LogName Application -Newest 5 | My-Function
```

Demonstration Process

Begin process end

Pipeline Input

Methods Of Accepting Parameter Pipeline Input

By Value

- Attempted first
- Incoming **object** and **parameter** are of **same** data TYPE
- Incoming **object** can be **converted** to same data **TYPE** as the parameter

By Property Name

- Attempted if object **does not** come in by **value**
- Incoming object has a **property name** that matches the **parameter name** and is the **same** data TYPE

Cmdlet parameters may **accept** pipelined **objects** by value, by property name or **both**.

Does a Parameter Accept Pipeline Input?

```
PS> Get-Help Restart-Computer -Parameter ComputerName

 -ComputerName <String[]>

    Specifies one or more remote computers. The default is ...

    Required?                    false
    Position?                    1
    Default value                Local computer
    Accept pipeline input?       True (ByValue, ByPropertyName)
    Accept wildcard characters?  false
```

Pipeline Input ByValue

```
PS>  Get-Help Get-Timezone -Parameter name

-Name <String[]>
    Specifies, as a string array, the name or names of the time zones
that this cmdlet gets.

    Required?                    false
    Position?                    0
    Default value                None
    Accept pipeline input?       True (ByValue)
    Accept wildcard characters?  false
```

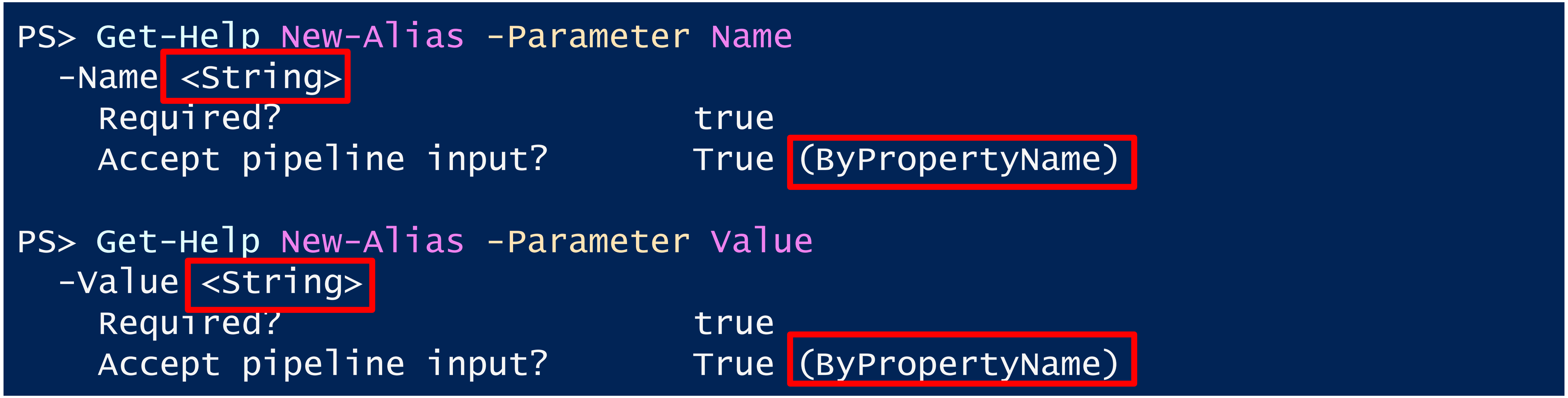
Strings

```
PS> "Eastern Standard Time", "Mountain Standard Time" | Get-TimeZone

PS> "Eastern Standard Time", "Mountain Standard Time" |
        ForEach-Object {Get-TimeZone -name $_}
```

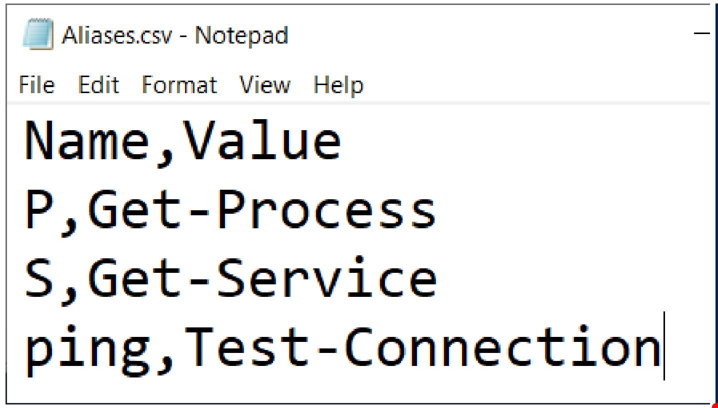
Same Results

Pipeline Input ByPropertyName

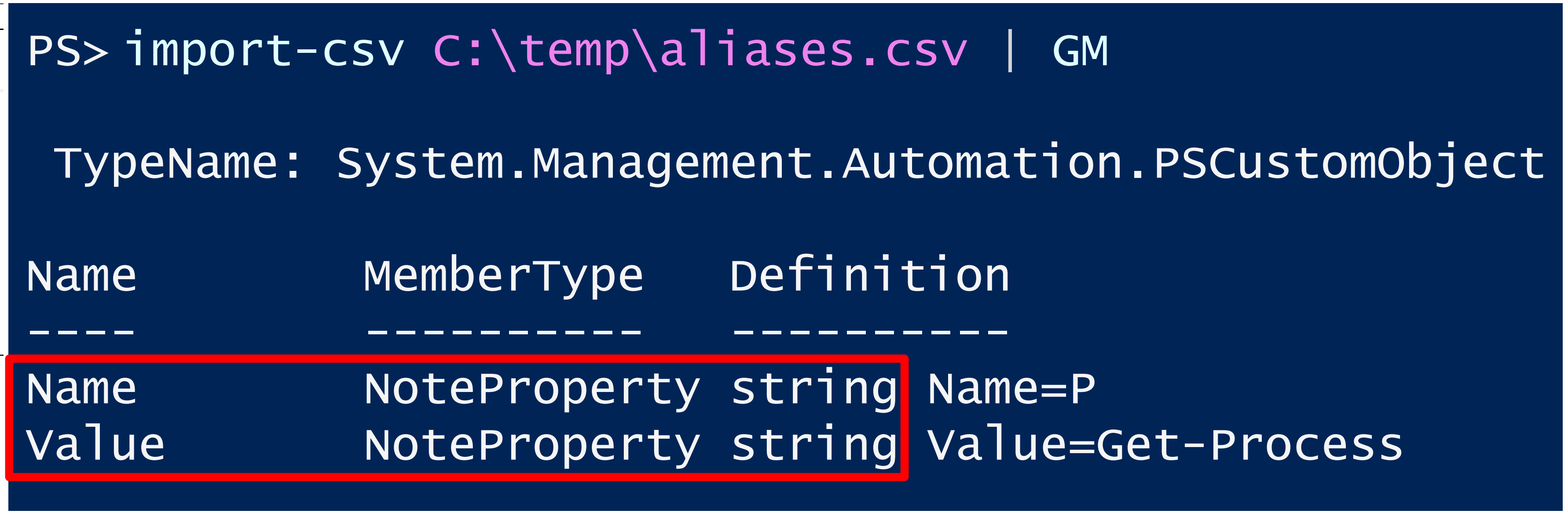


```
PS> Get-Help New-Alias -Parameter Name
  -Name <String>
    Required?                    true
    Accept pipeline input?       True (ByPropertyName)

PS> Get-Help New-Alias -Parameter Value
  -Value <String>
    Required?                    true
    Accept pipeline input?       True (ByPropertyName)
```



```
Name,Value
P,Get-Process
S,Get-Service
ping,Test-Connection
```

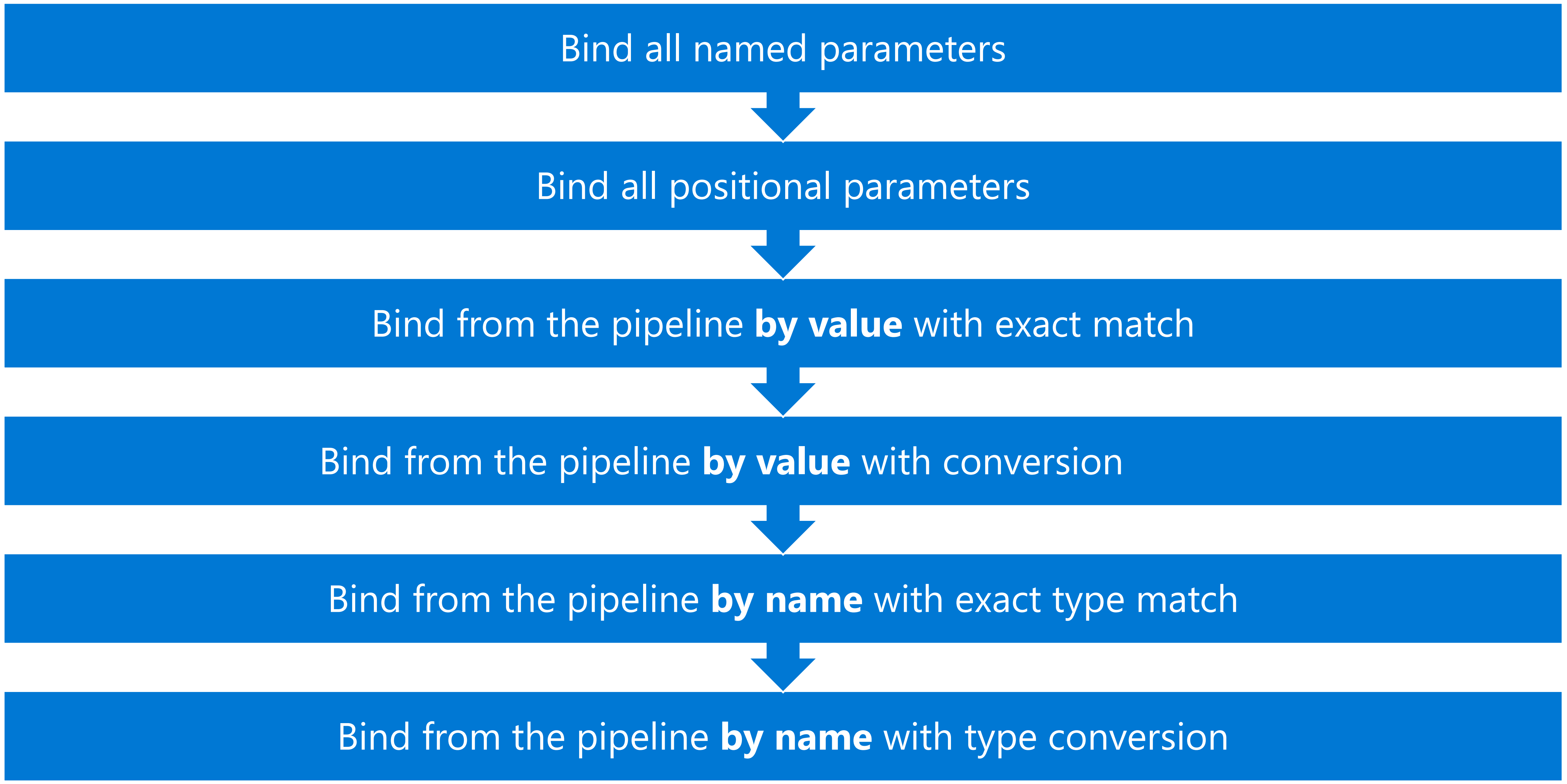


```
PS> import-csv C:\temp\aliases.csv | GM

 TypeName: System.Management.Automation.PSCustomObject

Name         MemberType   Definition
----         ----------   ----------
Name         NoteProperty string Name=P
Value        NoteProperty string Value=Get-Process
```

Parameter Binding Steps


Bind all named parameters
Bind all positional parameters
Bind from the pipeline **by value** with exact match
Bind from the pipeline **by value** with conversion
Bind from the pipeline **by name** with exact type match
Bind from the pipeline **by name** with type conversion

Demonstration
Pipeline Input

Pipeline Input

Lab 6:
Pipeline Advanced

45 minutes

LAB

Microsoft