PowerShell Piping and Filtering Tutorial

Table of Contents

Introduction to Piping

PowerShell's pipeline is one of its most powerful features. Unlike traditional command-line shells that pass text between commands, PowerShell passes .NET objects, preserving data structure and enabling rich manipulation.

What is Piping?

Piping sends the output of one cmdlet as input to another cmdlet using the | (pipe) operator. This creates a chain of commands that process data sequentially.

```powershell
# Basic syntax
Command1 | Command2 | Command3
```

Basic Piping Concepts

Object-Based Pipeline

PowerShell's pipeline works with objects, not just text:

```powershell
# Get processes and examine the object type
Get-Process | Get-Member

# Each process object has properties and methods
Get-Process | Select-Object Name, CPU, WorkingSet
```

Pipeline Variables

The current pipeline object is available as `$_` or `$PSItem`:

```powershell
Get-Process | Where-Object { $_.CPU -gt 100 }
```

Common Filtering Cmdlets

Essential Filtering Commands

Cmdlet	Purpose	Example
`Where-Object`	Filter objects based on conditions	`Get-Process | Where-Object CPU -gt 50`
`Select-Object`	Choose specific properties or objects	`Get-Process | Select-Object Name, CPU`
`Sort-Object`	Sort objects by properties	`Get-Process | Sort-Object CPU -Descending`
`Group-Object`	Group objects by property values	`Get-Process | Group-Object ProcessName`
`Measure-Object`	Calculate statistics	`Get-Process | Measure-Object CPU -Sum -Average`

Where-Object Filtering

Basic Syntax

```powershell
# Long form
Get-Process | Where-Object { $_.CPU -gt 50 }

# Short form (PowerShell 3.0+)
Get-Process | Where-Object CPU -gt 50

# Alias
Get-Process | ? CPU -gt 50
```

Comparison Operators

powershell

```powershell
# Equality
Get-Service | Where-Object Status -eq 'Running'

# Inequality
Get-Process | Where-Object Name -ne 'System'

# Greater than/Less than
Get-Process | Where-Object CPU -gt 100
Get-Process | Where-Object WorkingSet -lt 50MB

# Like/NotLike (supports wildcards)
Get-Process | Where-Object Name -like '*chrome*'
Get-Service | Where-Object DisplayName -notlike '*Microsoft*'

# Match/NotMatch (supports regex)
Get-Process | Where-Object Name -match '^s.*'
```

Multiple Conditions

powershell

```powershell
# AND condition
Get-Process | Where-Object { $_.CPU -gt 50 -and $_.WorkingSet -gt 100MB }

# OR condition
Get-Process | Where-Object { $_.Name -eq 'chrome' -or $_.Name -eq 'firefox' }

# Complex conditions
Get-Process | Where-Object {
    ($_.CPU -gt 50 -and $_.WorkingSet -gt 100MB) -or
    ($_.Name -like '*system*')
}
```

Select-Object for Data Shaping

Selecting Properties

powershell

```powershell
# Select specific properties
Get-Process | Select-Object Name, CPU, WorkingSet

# Select first/last objects
Get-Process | Sort-Object CPU -Descending | Select-Object -First 5
Get-EventLog System | Select-Object -Last 10

# Skip objects
Get-Process | Sort-Object Name | Select-Object -Skip 10 -First 5
```

Calculated Properties

powershell

```powershell
# Create calculated properties
Get-Process | Select-Object Name,
    @{Name='WorkingSetMB'; Expression={$_.WorkingSet / 1MB}},
    @{Name='CPUTime'; Expression={$_.TotalProcessorTime}}

# Using shortcuts
Get-Process | Select-Object Name,
    @{N='WSMB'; E={[math]::Round($_.WorkingSet / 1MB, 2)}},
    @{N='Status'; E={if($_.Responding){'OK'}else{'Not Responding'}}}
```

Unique Values

powershell

```powershell
# Get unique values
Get-Process | Select-Object ProcessName -Unique
Get-EventLog System | Select-Object EntryType -Unique
```

Sort-Object for Ordering

Basic Sorting

powershell

```powershell
# Sort by single property
Get-Process | Sort-Object Name
Get-Process | Sort-Object CPU -Descending

# Sort by multiple properties
Get-Process | Sort-Object ProcessName, CPU -Descending

# Mixed sort orders
Get-Process | Sort-Object @{Expression='ProcessName'; Descending=$false},
                          @{Expression='CPU'; Descending=$true}
```

Custom Sort Expressions

powershell

```powershell
# Sort by calculated values
Get-ChildItem | Sort-Object @{Expression={$_.Length}; Descending=$true}

# Sort by complex logic
Get-Process | Sort-Object @{
    Expression = {
        switch ($_.ProcessName) {
            'System' { 1 }
            'Idle' { 2 }
            default { 3 }
        }
    }
}, ProcessName
```

Group-Object for Categorization

Basic Grouping

powershell

```powershell
# Group by property
Get-Process | Group-Object ProcessName
Get-Service | Group-Object Status

# Group with counts
Get-EventLog System -Newest 100 | Group-Object EntryType |
    Select-Object Name, Count
```

Advanced Grouping

```powershell
# Group by calculated property
Get-Process | Group-Object @{Expression={
    if($_.WorkingSet -gt 100MB) {'High Memory'}
    elseif($_.WorkingSet -gt 50MB) {'Medium Memory'}
    else {'Low Memory'}
}}

# Group with custom formatting
Get-ChildItem | Group-Object @{Expression={$_.Extension.ToUpper()}} |
    Sort-Object Name |
    Format-Table Name, Count
```

Advanced Piping Techniques

ForEach-Object Processing

```powershell
# Process each object in the pipeline
Get-Process | ForEach-Object {
    [PSCustomObject]@{
        Name = $_.Name
        MemoryMB = [math]::Round($_.WorkingSet / 1MB, 2)
        Running = $_.Responding
    }
}

# Using the % alias
1..10 | % { $_ * $_ }  # Square each number
```

Tee-Object for Branching

```powershell
# Save intermediate results while continuing pipeline
Get-Process |
    Tee-Object -FilePath 'processes.txt' |
    Where-Object CPU -gt 50 |
    Sort-Object CPU -Descending
```

Out-GridView for Interactive Filtering

powershell

```powershell
# Open results in a GUI for interactive filtering
Get-Process | Out-GridView -PassThru | Stop-Process -WhatIf
Get-EventLog System -Newest 1000 | Out-GridView
```

Pipeline Parameter Binding

powershell

```powershell
# Understanding how objects bind to parameters
Get-Process chrome | Stop-Process -WhatIf  # ByValue binding
'chrome', 'notepad' | Get-Process          # ByPropertyName binding
```

Performance Considerations

Early Filtering

powershell

```powershell
# Good: Filter early in the pipeline
Get-WinEvent -LogName System | Where-Object Id -eq 1074 | Select-Object -First 10

# Better: Use cmdlet parameters when available
Get-WinEvent -LogName System -FilterHashtable @{ID=1074} | Select-Object -First 10
```

Efficient Object Selection

powershell

```powershell
# Avoid selecting unnecessary properties early
Get-Process | Select-Object Name, CPU | Where-Object CPU -gt 50

# Better: Filter first, then select
Get-Process | Where-Object CPU -gt 50 | Select-Object Name, CPU
```

Memory Management

powershell

```powershell
# For large datasets, consider streaming
Get-Content largefile.txt | Where-Object { $_ -match 'error' } |
    Select-Object -First 100
```

Real-World Examples

System Administration

powershell

```powershell
# Find high CPU processes
Get-Process |
    Where-Object CPU -gt 100 |
    Sort-Object CPU -Descending |
    Select-Object Name, CPU, WorkingSet, Id |
    Format-Table -AutoSize

# Check disk space
Get-WmiObject Win32_LogicalDisk |
    Where-Object DriveType -eq 3 |
    Select-Object DeviceID,
        @{N='SizeGB';E={[math]::Round($_.Size/1GB,2)}},
        @{N='FreeGB';E={[math]::Round($_.FreeSpace/1GB,2)}},
        @{N='%Free';E={[math]::Round(($_.FreeSpace/$_.Size)*100,1)}} |
    Where-Object '%Free' -lt 20
```

Log Analysis

powershell

```powershell
# Analyze Windows Event Log
Get-WinEvent -LogName System -MaxEvents 1000 |
    Where-Object LevelDisplayName -eq 'Error' |
    Group-Object Id |
    Sort-Object Count -Descending |
    Select-Object -First 10 |
    Format-Table Name, Count

# Find recent logon events
Get-WinEvent -LogName Security -FilterHashtable @{ID=4624} |
    Select-Object TimeCreated,
        @{N='User';E={$_.Properties[5].Value}},
        @{N='LogonType';E={$_.Properties[8].Value}} |
    Where-Object User -notlike '*$' |
    Sort-Object TimeCreated -Descending |
    Select-Object -First 20
```

File Management

powershell

```powershell
# Find large files
Get-ChildItem C:\ -Recurse -File -ErrorAction SilentlyContinue |
    Where-Object Length -gt 100MB |
    Sort-Object Length -Descending |
    Select-Object FullName,
        @{N='SizeMB';E={[math]::Round($_.Length/1MB,2)}} |
    Select-Object -First 20

# Group files by extension
Get-ChildItem -Path . -Recurse -File |
    Group-Object Extension |
    Sort-Object Count -Descending |
    Select-Object Name, Count,
        @{N='TotalSizeMB';E={
            ($_.Group | Measure-Object Length -Sum).Sum / 1MB
        }} |
    Format-Table -AutoSize
```

Network Analysis

powershell

```powershell
# Analyze network connections
Get-NetTCPConnection |
    Where-Object State -eq 'Established' |
    Group-Object RemotePort |
    Sort-Object Count -Descending |
    Select-Object Name, Count |
    Select-Object -First 10
```

Best Practices

1. **Filter Early**: Apply Where-Object as early as possible in the pipeline
2. **Use Cmdlet Parameters**: Prefer built-in filtering parameters over Where-Object when available
3. **Select Wisely**: Only select the properties you need to improve performance
4. **Understand Object Types**: Use Get-Member to understand what you're working with
5. **Test Incrementally**: Build complex pipelines step by step
6. **Use Aliases Judiciously**: While aliases like `?` and `%` are convenient, full cmdlet names are clearer in scripts

Common Pitfalls

- **Forgetting Object Types**: Remember you're working with objects, not text
- **Over-selecting**: Selecting too many properties can impact performance
- **Late Filtering**: Filtering at the end of a long pipeline wastes resources
- **Ignoring Errors**: Use -ErrorAction to handle errors in pipeline operations
- **Memory Issues**: Be careful with large datasets; consider streaming approaches

This tutorial provides a solid foundation for mastering PowerShell's piping and filtering capabilities. Practice these examples and experiment with your own data to become proficient with these powerful tools.