Using Common PowerShell Operators

Jeff Hicks
AUTHOR/TEACHER

@jeffhicks | https://jdhitsolutions.com

PowerShell Operators

PowerShell is about *doing*

PowerShell operators are critical parts of the syntax

You can use interactively at the console

More likely to use in PowerShell scripting

Use PowerShell Help

Read the help documentation

- Help about_*operators*

Many operators work in Windows PowerShell 5.1

Learning operators is very basic

Basic demonstrations are very simple

You need to try them on your own

Comparison Operators

Compare values

Result is True or False

PowerShell is not case-sensitive*

```
PS C:\> 5 -gt 2
True
PS C:\> 2 -ge 2
True
PS C:\> 5 -lt 2
False
PS C:\> 2 -le 5
True
```

◀ Greater than

◀ Greater than or equal to

◀ Less than

◀ Less than or equal to

```
PS C:\> $i = 7
PS C:\> $i -eq 7
True
PS C:\> $i -ne 7
False
PS C:\> "jeff" -eq "JEFF"
True
PS C:\> "jeff" -ceq "JEFF"
False
```

◄ This is the assignment operator

◄ Is $i equal to 7?

◄ Is $i not equal to 7

◄ String comparisons are **not** case-sensitive

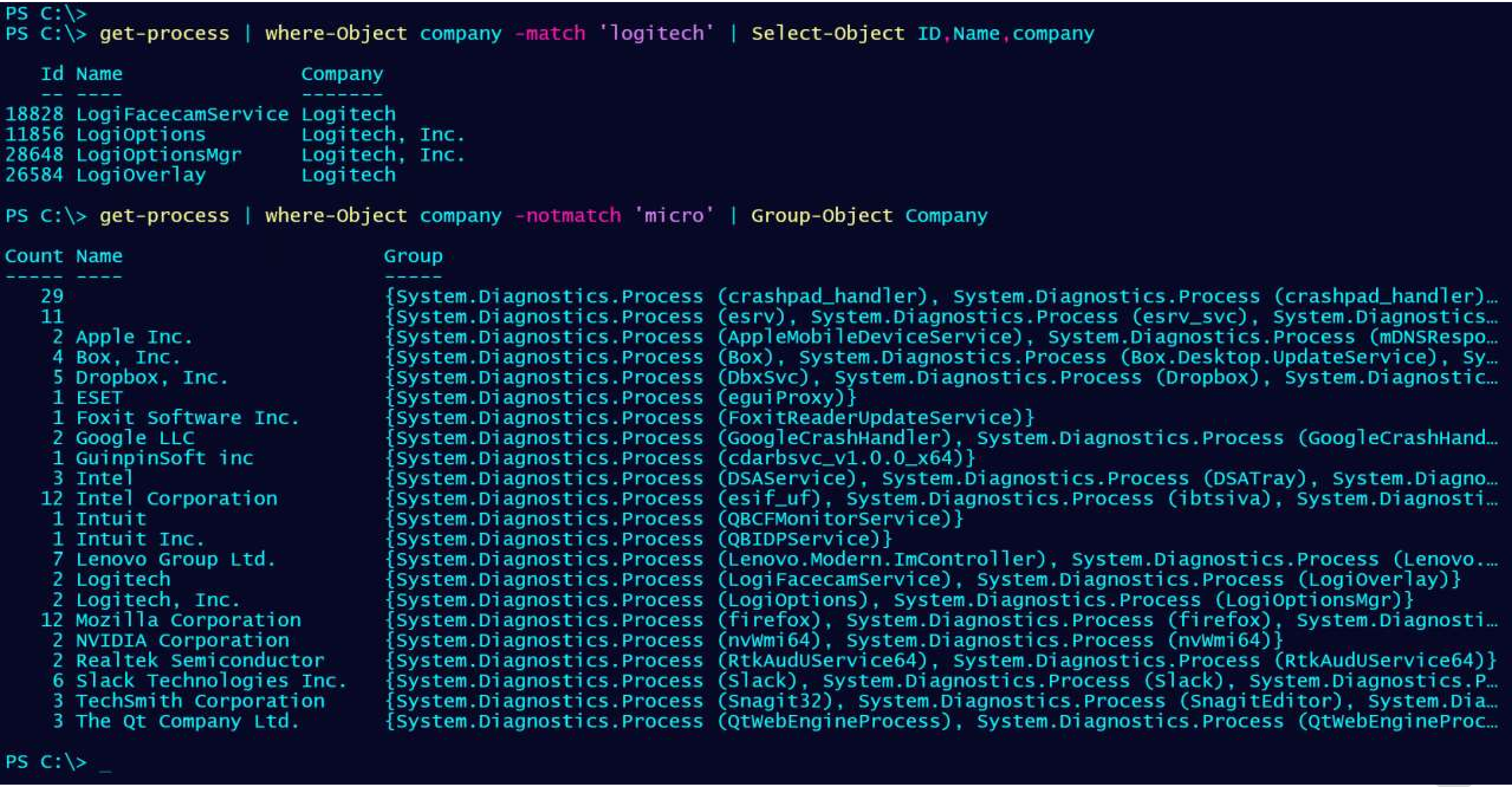
◄ But they can be

```
PS C:\> "foo" -like "f*"
True
PS C:\> "bar" -notlike "B*"
False
PS C:\> "bar" -clike "B*"
False
```

- Use wildcard comparisons
- Comparisons are not case-sensitive
- But they can be
  -cnotlike

```
PS C:\> "abc-1234" -match "\d+"
True
PS C:\> "1234" -notmatch "\d+"
False
```

- Use the regular expression operator
- This is an advanced topic
- Help About_Regular_Expressions
- Easy way to test a non-match



```
PS C:\>
PS C:\> get-process | where-Object company -match 'logitech' | Select-Object ID,Name,company

   Id Name               Company
   -- ----               -------
18828 LogiFacecamService Logitech
11856 LogiOptions        Logitech, Inc.
28648 LogiOptionsMgr     Logitech, Inc.
26584 LogiOverlay        Logitech

PS C:\> get-process | where-Object company -notmatch 'micro' | Group-Object Company

Count Name                      Group
----- ----                      -----
   29                           {System.Diagnostics.Process (crashpad_handler), System.Diagnostics.Process (crashpad_handler)…
   11                           {System.Diagnostics.Process (esrv), System.Diagnostics.Process (esrv_svc), System.Diagnostics…
    2 Apple Inc.                {System.Diagnostics.Process (AppleMobileDeviceService), System.Diagnostics.Process (mDNSRespo…
    4 Box, Inc.                 {System.Diagnostics.Process (Box), System.Diagnostics.Process (Box.Desktop.UpdateService), Sy…
    5 Dropbox, Inc.             {System.Diagnostics.Process (DbxSvc), System.Diagnostics.Process (Dropbox), System.Diagnostic…
    1 ESET                      {System.Diagnostics.Process (eguiProxy)}
    1 Foxit Software Inc.       {System.Diagnostics.Process (FoxitReaderUpdateService)}
    2 Google LLC                {System.Diagnostics.Process (GoogleCrashHandler), System.Diagnostics.Process (GoogleCrashHand…
    1 GuinpinSoft inc           {System.Diagnostics.Process (cdarbsvc_v1.0.0_x64)}
    3 Intel                     {System.Diagnostics.Process (DSAService), System.Diagnostics.Process (DSATray), System.Diagno…
   12 Intel Corporation         {System.Diagnostics.Process (esif_uf), System.Diagnostics.Process (ibtsiva), System.Diagnosti…
    1 Intuit                    {System.Diagnostics.Process (QBCFMonitorService)}
    1 Intuit Inc.               {System.Diagnostics.Process (QBIDPService)}
    7 Lenovo Group Ltd.         {System.Diagnostics.Process (Lenovo.Modern.ImController), System.Diagnostics.Process (Lenovo.…
    2 Logitech                  {System.Diagnostics.Process (LogiFacecamService), System.Diagnostics.Process (LogiOverlay)}
    2 Logitech, Inc.            {System.Diagnostics.Process (LogiOptions), System.Diagnostics.Process (LogiOptionsMgr)}
   12 Mozilla Corporation       {System.Diagnostics.Process (firefox), System.Diagnostics.Process (firefox), System.Diagnosti…
    2 NVIDIA Corporation        {System.Diagnostics.Process (nvWmi64), System.Diagnostics.Process (nvWmi64)}
    2 Realtek Semiconductor     {System.Diagnostics.Process (RtkAudUService64), System.Diagnostics.Process (RtkAudUService64)}
    6 Slack Technologies Inc.   {System.Diagnostics.Process (Slack), System.Diagnostics.Process (Slack), System.Diagnostics.P…
    3 TechSmith Corporation     {System.Diagnostics.Process (Snagit32), System.Diagnostics.Process (SnagitEditor), System.Dia…
    3 The Qt Company Ltd.       {System.Diagnostics.Process (QtWebEngineProcess), System.Diagnostics.Process (QtWebEngineProc…

PS C:\> _
PS C:\> 5+8
13

PS C:\> 9/3
3

PS C:\> 2*3*4
24

PS C:\> 10-6
4

PS C:\> (((5+3)/2)*7)-1
27
```

◀ Math operators are the ones you've always used

◀ Addition

◀ Division

◀ Multiplication

◀ Subtraction

◀ Control precedence

```
PS C:\> 1..5
1
2
3
4
5
PS C:\> 10..7
10
9
8
7
```

◄ Range operator
◄ Get numbers from beginning to end

◄ Reverse

```
PS C:\> (1..4),(7..10)
1
2
3
4
7
8
9
10
```

◀ This is technically 2 ranges

```
PS C:\> 97..105 |
>> foreach-object {[Char]$_}
>>
a
b
c
d
e
f
g
h
i
```

- Doesn’t work for letters
- But you can use “raw” [Char] values
- 65..90 = A-Z
- This is advanced stuff

```
PS C:\> $i = 4
```

Logical Operators

Sometimes you have complex expressions

```
PS C:\> $i = 4
PS C:\> ($i -le 10) -AND ($PSVersionTable.PSVersion.Major -ge 7)
```

Logical Operators

Sometimes you have complex expressions

This expression AND that expression must be BOTH be True

```
PS C:\> $i = 4
PS C:\> ($i -le 10) -AND ($PSVersionTable.PSVersion.Major -ge 7)
True
```

Logical Operators

Sometimes you have complex expressions

This expression AND that expression must be BOTH be True

The expression result is True

```
PS C:\> $i = 20
PS C:\> ($i -le 10) -AND ($PSVersionTable.PSVersion.Major -ge 7)
False
```

Logical Operators

One expression is False so the entire expression is False

```
PS C:\> $i = 20
PS C:\> ($i -le 10) -OR ($PSVersionTable.PSVersion.Major -ge 7)
True
```

Logical Operators

If either expression is True, the entire expression is True

```
PS C:\> $i = 20
PS C:\> $name = "jeff"
PS C:\> ($i -ge 20) -AND (($name -eq "Jeff") -OR $IsLinux)
True
```

Logical Operators

Combine expressions

Parentheses very helpful

```
PS C:\> $i = 20
PS C:\> $name = "jeff"
PS C:\> ($i -ge 20) -AND (($name -eq "Jeff") -OR $IsLinux)
True
```

Logical Operators

Combine expressions

Parentheses very helpful

```
PS C:\> Test-Path c:\windows\notepad.exe
True
```

Logical Operators

Normal result

```
PS C:\> Test-Path c:\windows\notepad.exe
True
PS C:\> -Not (Test-Path c:\windows\notepad.exe)
False
```

Logical Operators

Reverse the Boolean

```
PS C:\> Test-Path c:\windows\notepad.exe
True
PS C:\> -Not (Test-Path c:\windows\notepad.exe)
False
PS C:\> !(Test-Path c:\windows\notepad.exe)
False
```

Logical Operators

Reverse the Boolean

You can also use !

Expect to use more often in scripting

```
PS C:\> get-process -id $pid && 100
```

Chain Operator &&

Introduced in PowerShell 7

If the left-side expression is successful

Invoke the right-side expression

```
PS C:\> get-process -id $pid && 100

 NPM(K)    PM(M)      WS(M)     CPU(s)      Id  SI ProcessName
 ------    -----      -----     ------      --  -- -----------
    117   153.23     206.56       9.16   18944   1 pwsh
100
```

Chain Operator &&

May be more useful in building a script chain

```
PS C:\> get-process -id 99999 && 100
Get-Process: Cannot find a process with the process identifier 99999.
```

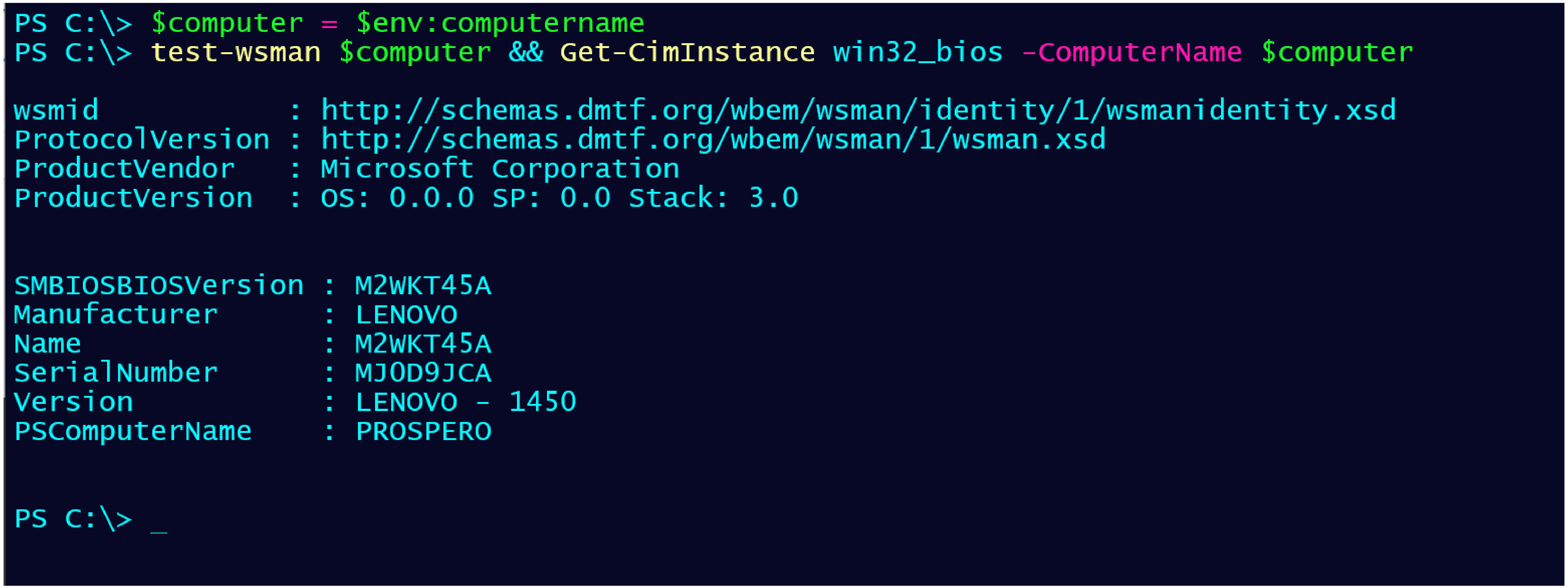
Chain Operator &&

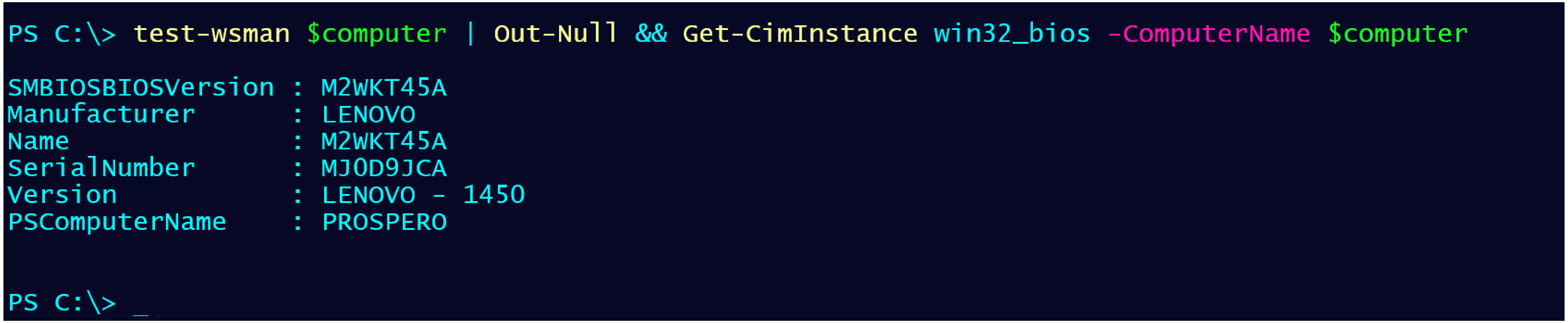
The first command fails so the second never runs

```
PS C:\> get-process -id 99999 || 100
Get-Process: Cannot find a process with the process identifier 99999.
100
```

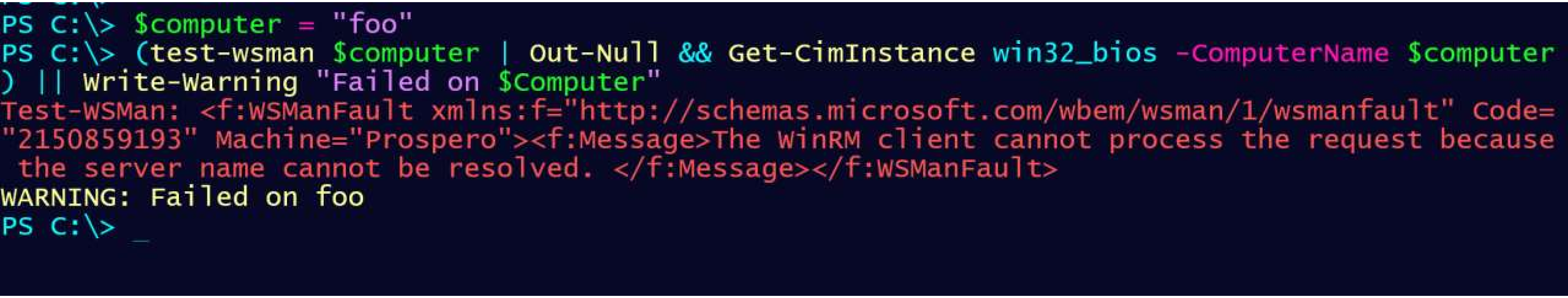
Chain Operator ||

If the first expression fails, then invoke the second


```
PS C:\> $computer = $env:computername
PS C:\> test-wsman $computer && Get-CimInstance win32_bios -ComputerName $computer

wsmid           : http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
ProtocolVersion : http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
ProductVendor   : Microsoft Corporation
ProductVersion  : OS: 0.0.0 SP: 0.0 Stack: 3.0


SMBIOSBIOSVersion : M2WKT45A
Manufacturer      : LENOVO
Name              : M2WKT45A
SerialNumber      : MJ0D9JCA
Version           : LENOVO - 1450
PSComputerName    : PROSPERO


PS C:\> _
```



```
PS C:\> test-wsman $computer | Out-Null && Get-CimInstance win32_bios -ComputerName $computer

SMBIOSBIOSVersion : M2WKT45A
Manufacturer      : LENOVO
Name              : M2WKT45A
SerialNumber      : MJ0D9JCA
Version           : LENOVO - 1450
PSComputerName    : PROSPERO


PS C:\> _
```



```
PS C:\> $computer = "foo"
PS C:\> (test-wsman $computer | Out-Null && Get-CimInstance win32_bios -ComputerName $computer
) || Write-Warning "Failed on $Computer"
Test-WSMan: <f:WSManFault xmlns:f="http://schemas.microsoft.com/wbem/wsman/1/wsmanfault" Code=
"2150859193" Machine="Prospero"><f:Message>The WinRM client cannot process the request because
 the server name cannot be resolved. </f:Message></f:WSManFault>
WARNING: Failed on foo
PS C:\> _
PS C:\> help about_if
```

Ternary Operator

An alternative to If/Else

<condition to test> ? <True code> : <False code>

```
PS C:\> If (2 -ge 1) {"yes"} else {"no"}
yes
```

Ternary Operator

```
PS C:\> 2 -ge 1 ? "yes" : "no"
yes
```

Ternary Operator

```
PS C:\> (Get-Process notepad) ? (Stop-Process -name Notepad) : (Write-
Warning "Notepad is not running")
```

Ternary Operator

Wrap expressions in ()

```
PS C:\> (Get-Process notepad) ? (Stop-Process -name Notepad) : (Write-
Warning "Notepad is not running")

Get-Process: Cannot find a process with the name "notepad". Verify the
process name and call the cmdlet again.

WARNING: Notepad is not running
```

Ternary Operator

Wrap expressions in ()

Convenient for short If/Else logic

```
PS C:\> $isWindows ? (gcim win32_operatingsystem) : (lsb_release -d)
```

Ternary Operator

gcim is an alias for Get-CimInstance

lsb_release is a Linux command

```
PS C:\> $n = $null
```

Null-Coalescing ??

PowerShell 7 includes operators that make it easy to work with null values

```
PS C:\> $n = $null
PS C:\> $n ?? (Write-Warning "the variable is null")
```

Null-Coalescing ??

PowerShell 7 includes operators that make it easy to work with null values

If the left side of ?? is null, evaluate the right side

```
PS C:\> $n = $null
PS C:\> $n ?? (Write-Warning "the variable is null")
WARNING: the variable is null
```

Null-Coalescing ??

PowerShell 7 includes operators that make it easy to work with null values

If the left side of ?? is null, evaluate the right side

```
PS C:\> $v = $PSEdition ?? ("unknown")
PS C:\> $v
Core
```

Null-Coalescing ??

If the left side is NOT null, the right side is never used

```
PS C:\> $n = $null
PS C:\> $n ??= "foo"
```

Null-Coalescing Assignment ??=

Assign the value of the right hand to the left hand, if the left hand is Null

```
PS C:\> $n = $null
PS C:\> $n ??= "foo"
PS C:\> $n
foo
```

Null-Coalescing Assignment ??=

Assign the value of the right hand to the left hand, if the left hand is Null

$n is null so the right-side value is assigned to $n

```
PS C:\> $n = $null
PS C:\> $n ??= "foo"
PS C:\> $n
foo
PS C:\> $n ??= "bar"
PS C:\> $n
```

Null-Coalescing Assignment ??=

Assign the value of the right hand to the left hand, if the left hand is Null

$n is null so the right-side value is assigned to $n

What is the value of $n now?

```
PS C:\> $n = $null
PS C:\> $n ??= "foo"
PS C:\> $n
foo
PS C:\> $n ??= "bar"
PS C:\> $n
foo
```

Null-Coalescing Assignment ??=

Assign the value of the right hand to the left hand, if the left hand is Null

$n is null so the right-side value is assigned to $n

What is the value of $n now?

```
PS C:\> $p = Get-Process | Where-Object {$_.WorkingSet -gt 1GB}
PS C:\> $p ??= 0
PS C:\> $p
0
```

Null-Coalescing Assignment ??=

There are no matching processes so $p gets assigned a value of 0

```
PS C:\> $p = Get-Process -id $pid
```

Null-Conditional Operator ?.

Applies when accessing a member of an object

```
PS C:\> $p = Get-Process -id $pid
PS C:\> $p.ToString()
```

Null-Conditional Operator ?.

Applies when accessing a member of an object

```
PS C:\> $p = Get-Process -id $pid
PS C:\> $p.ToString()
System.Diagnostics.Process (pwsh)
```

Null-Conditional Operator ?.

Applies when accessing a member of an object

This is the expected result because $p is not null

```
PS C:\> $p = $null
```

Null-Conditional Operator ?.

But if $p was null?

Maybe you forgot to assign a value

Maybe some external process failed to create it

```
PS C:\> $p = $null
PS C:\> $p.ToString()
InvalidOperation: You cannot call a method on a null-valued expression.
```

Null-Conditional Operator ?.

An error is thrown

This is not necessarily a bad thing

```
PS C:\> ${p}?.ToString()
PS C:\>
```

Null-Conditional Operator ?.

Use the ?. Operator

The {} is required to isolate the variable name

‘?’ is technically legal for variable names

```
PS C:\> $var = ${p}?.ToString() ?? "method failed"
PS C:\> $var
```

Combining Operators

If the left-side of ?? is Null, then evaluate the right-side

PowerShell writes the result of ?? to the pipeline

```
PS C:\> $var = ${p}?.ToString() ?? "method failed"
PS C:\> $var

method failed
PS C:\>
```

Combining Operators

Which is the message

You are more likely to use the Null-related operators in PowerShell scripting

Key Points

Operators are key elements of the PowerShell language

You can use them interactively in the console

Many newer operators are designed to be easy to use in pipelined expressions

Some operators have scripting alternatives which might be easier to understand

Group with parentheses for clarity

Read the help!