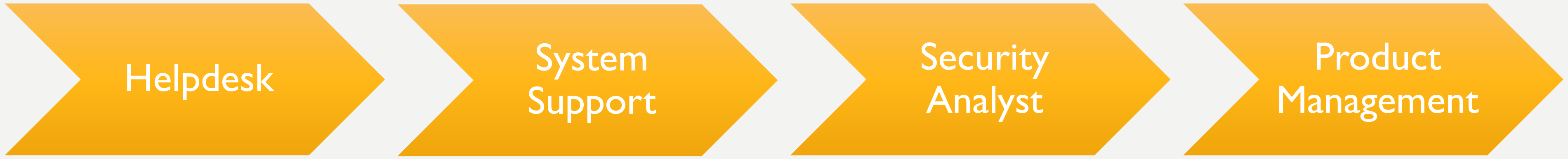
LEVEL UP WITH POWERSHELL

JOSH RICKARD

WELCOME!

- Introductions
- Break Times
- Housekeeping

JOSH RICKARD



ABOUT THIS COURSE

- Utilizes *Learn Windows PowerShell 3 in a Month of Lunches* (http://bit.ly/PSHv3Lunch)
- Focuses on *how to use PowerShell*, rather than on using PowerShell for any one specific product (e.g., Exchange or SharePoint)
- Provides the foundation skills you need to be *immediately effective* with PowerShell, and to self-teach whatever specific products you need to administer.

NOW YOU!

VERSIONS

POWERSHELL VERSIONS

- V2
  - WinXP+, Win2003+
- V3
  - Win7+, Win2008+
- V4
  - Win7+, Win2008R2+
- V5
  - Win10+. Win2012R2+

VERSION 2

- Windows XP and Later
- Windows Server 2003 or Later

- .NET Framework 2.0 (minimum)
- .NET Framework 3.5 (optimal)

VERSION 3

- Windows 7 and Later
- Windows Server 2008 and Later

- .NET Framework 4.0 (Full Install, NOT the client profile)

VERSION 4

- Windows 7 and Later
- Windows Server 2008R2 and Later

- .NET Framework 4.5

VERSION 5

- Windows 7 and Later
- Windows 2008R2 and Later

- .NET Framework 5.0

PowerShell V5

- Windows 7
- Windows 8.1
- Windows 10

64-BIT OS

- 4 different versions of PowerShell
  - Windows PowerShell (x86)
  - Windows PowerShell ISE (x86)
  - Windows PowerShell ISE
  - Windows PowerShell

ADDITIONAL FEATURES

- PowerShell Version features are dependent on a few things
  - The Host OS
  - The Feature you are wanting to use
    - I.e. Exchange, ActiveDirectory, etc.

- Native Version installed on OS will have more features than an older OS

POWERSHELL CONSOLE

POWERSHELL ISE

USING POWERSHELL

FINDING COMMANDS

- CmdLet
  - This is a keyword for ”Native Commands”
- Functions
  - This is a keyword for added functions
- Modules
  - This is a collection of Functions added to your PowerShell Session

GET-MODULE

- Get-Module –ListAvailable
  - –List all available Modules
- Import-module –Name TroubleshootingPack
  - –Loading it into memory
- You have to import the module everytime you load PowerShell
  - –Per Process/Per Window case

POWERSHELL V3

- Automatically loads Modules if referenced
- Even if you run
  - Get-Module
- And only a few return
- When you run Get-Command –Noun *computer*
- It will still possibly list out other modules that have not been loaded
- These will be loaded at runtime

- This is based on a PowerShell Module Path
  - $env:PSModulePath

POWERSHELL MODULE PATHS

- C:\Users\Administrator\Documents\WindowsPowerShell\Modules\
  - This location is for PowerShell modules that you have loaded or downloaded from the web
- C:\Windows\System32\WindowsPowerShell\v1.0\Modules\
  - This location is for native PowerShell modules, usually added by adding features or specific programs
    - i.e. Active Directory Users and Computers, Exchange CmdLets, etc.

THREE CRUCIAL COMMANDS

- ❑ Get-Command
- ❑
- ❑

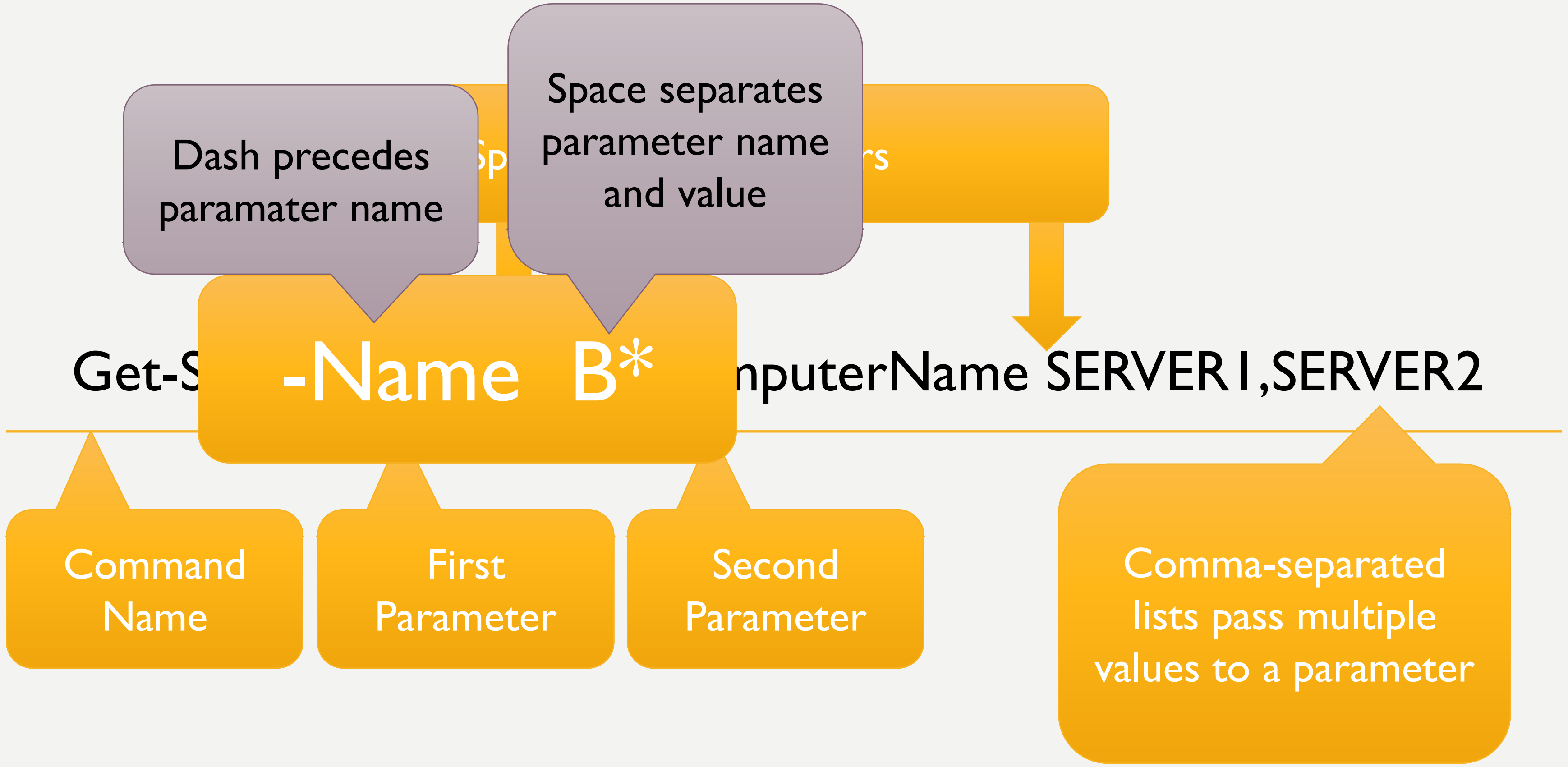
GET-COMMAND

- Get-Module –Module TroubleshootingPack
- Get-Command -Name *pack*
- Get-Command -Name *pack* -CommandType cmdlet,function
  - Only PowerShell Native stuff
- Get-Command –Name *log* -CommandType cmdlet, function

NAMING CONVENTIONS

- Verb-Noun (Singular Noun)
- Get-Comamnd –Name *service* -CommandType cmdlet,function
- Get-Command –Verb Get –Noun *service*

ANATOMY OF A COMMAND


Dash precedes paramater name
Space separates parameter name and value
Sp
rs
Get-S
-Name B*
nputerName SERVER1,SERVER2
Command Name
First Parameter
Second Parameter
Comma-separated lists pass multiple values to a parameter

QUESTION

- If you wanted to view a list of running processes, what would the Verb be?
- Get
- Get-Command –Verb Get –Noun *process*

TAB COMPLETION

- Get-Command -Noun netadapteren*

- Tab Completion

ALIAS'S

- Get-Command –CommandType Alias
  - % - Foreach-Object
  - ? – Where-Object
  - Cat – Get-Content
  - Cp – Copy-Item
  - Del – Remove-Item
  - Gci - Get-ChildItem
  - Ls – Get-ChildItem
  - Etc.

ANATOMY OF A COMMAND

Positional parameters can accept values without providing the parameter name – if you do so in the correct order!

```
gsv   B*  -Comp SERVER1,SERVER2
```

Alias is a "nickname" for the command

All parameter names can be truncated

FINDING HELP

SEO

- CmdLet
  - Add it to your web search
  - Microsoft made up this word to help with searching the internet for PowerShell help

WHERE DO I GET MODULES?

- Typically included in your additional software
  - i.e. Active Directory -> RSAT Tools

THREE CRUCIAL COMMANDS

- ❑ Get-Command
- ❑ Help
- ❑

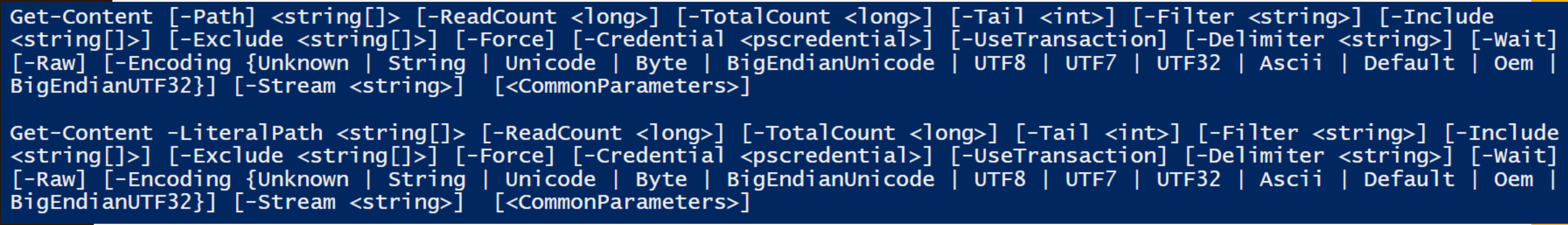
GET-HELP

- Get-Help –Name dir
  - Sometimes it will scroll past you
  - You can use help (alias) instead
    - This pipes the Get-Help info to the More command
    - Caveat, help will not always show you help unless you have loaded that module into memory first

HELP SWITCHES

- Get-Help dir
- Get-Help dir –full
- Get-Help dir –Examples
- Get-Help dir –Online
- Get-Help dir -ShowWindow

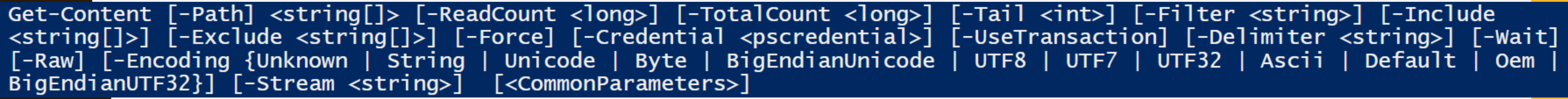
UNDERSTANDING HELP SYNTAX



```
Get-Content [-Path] <string[]> [-ReadCount <long>] [-TotalCount <long>] [-Tail <int>] [-Filter <string>] [-Include
<string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-UseTransaction] [-Delimiter <string>] [-Wait]
[-Raw] [-Encoding {Unknown | String | Unicode | Byte | BigEndianUnicode | UTF8 | UTF7 | UTF32 | Ascii | Default | Oem |
BigEndianUTF32}] [-Stream <string>]  [<CommonParameters>]

Get-Content -LiteralPath <string[]> [-ReadCount <long>] [-TotalCount <long>] [-Tail <int>] [-Filter <string>] [-Include
<string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-UseTransaction] [-Delimiter <string>] [-Wait]
[-Raw] [-Encoding {Unknown | String | Unicode | Byte | BigEndianUnicode | UTF8 | UTF7 | UTF32 | Ascii | Default | Oem |
BigEndianUTF32}] [-Stream <string>]  [<CommonParameters>]
```

- You cannot mix and match these parameter sets
- -Path only exists in the first parameter set
- -LiteralPath only exists in the second parameter set

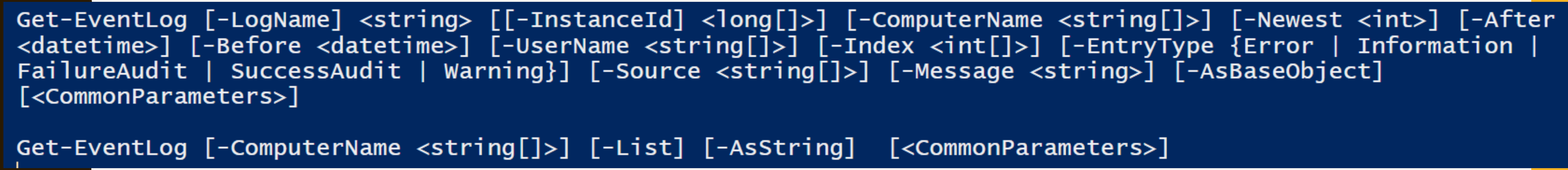
UNDERSTANDING PARAMETERS



```
Get-Content [-Path] <string[]> [-ReadCount <long>] [-TotalCount <long>] [-Tail <int>] [-Filter <string>] [-Include
<string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-UseTransaction] [-Delimiter <string>] [-Wait]
[-Raw] [-Encoding {Unknown | String | Unicode | Byte | BigEndianUnicode | UTF8 | UTF7 | UTF32 | Ascii | Default | Oem |
BigEndianUTF32}] [-Stream <string>]  [<CommonParameters>]
```

- [-Path] <string[]>
  - -Path is the name of the paramter. It will always start with a dash in front of it
  - The <string[]> is the value that this parameter must have
  - The square brackets surrounding the –Path indicates that it is a mandatory parameter
- [-InstanceId <long>]
  - The Square brackets surrounding this entire strings means that it is an optional parameter
- Another Example

UNDERSTANDING PARAMETERS



```
Get-EventLog [-LogName] <string> [[-InstanceId] <long[]>] [-ComputerName <string[]>] [-Newest <int>] [-After
<datetime>] [-Before <datetime>] [-UserName <string[]>] [-Index <int[]>] [-EntryType {Error | Information |
FailureAudit | SuccessAudit | Warning}] [-Source <string[]>] [-Message <string>] [-AsBaseObject]
[<CommonParameters>]

Get-EventLog [-ComputerName <string[]>] [-List] [-AsString]  [<CommonParameters>]
```

- 2 Parameter Sets
  - [[-InstanceId] <long[]>]
    - This indicates that this is an optional parameter
  - [-LogName] <string>
    - Notice, that this parameter does not have square brackets around the entire thing.
    - This means that it is a mandatory parameter
    - <string> is the value/data type of that value
- BUT! The "square brackets" around –LogName mean that the parameter it self is optional but not the value.
  - This is because of Positional Parameters

PARAMETER POSITIONING

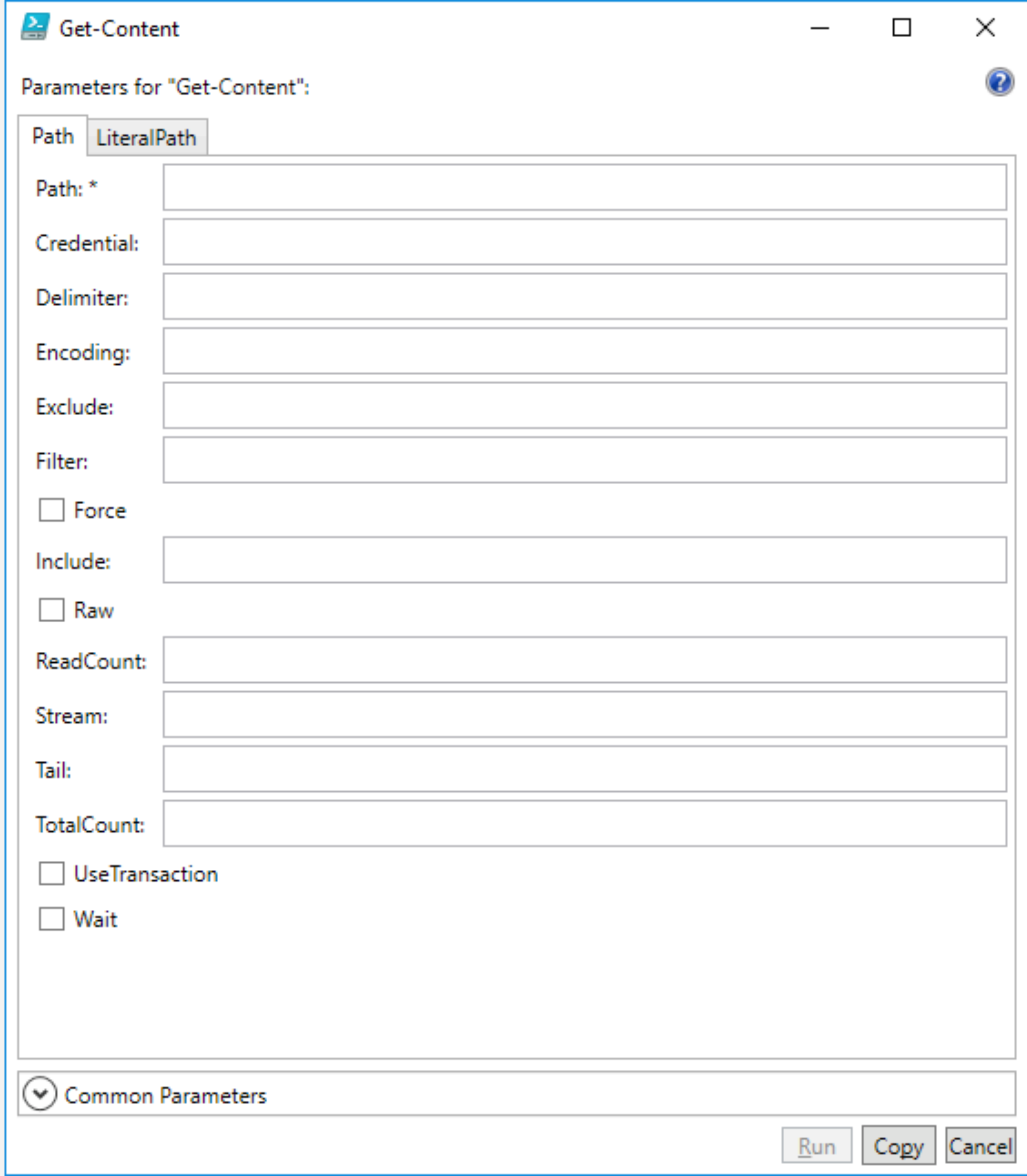
- [-LogName] <string>
  - Because –LogName is listed first, this means that it sits in Position 0

- Examples:
  - Get-EventLog –LogName Security –ComputerName ‘TEST-Computer’ ✓
  - Get-EventLog –LogName Security ‘TEST-Computer’ ✓
  - Get-EventLog Security –ComputerName ‘TEST-Computer’ ✓
  - Get-EventLog Security ‘TEST-Computer’ ✓
  - Get-EventLog ‘TEST-Computer’ Security ✗
  - Get-EventLog -ComputerName ‘Test-Computer’ Security ✗
  - Get-EventLog –ComputerName ‘TEST-Computer –LogName Security ✓

HELP FILES

- Update-Help
  - It's that simple. This will go out and make sure you have the most recent help/man files for each CmdLet.
- Save-Help
  - This will save your help files to a location, since Update-Help needs an internet connection
- No Internet connected machines
  - Save-Help –DestinationPath 'c:\some\path'
  - Update-Help –SourcePath 'C:\some\path'

- Show-Command –Name Get-EventLog
  - A GUI for filling out your parameters/values
  - It will have different windows for different parameter sets



Get-Content

Parameters for "Get-Content":

Path | LiteralPath

Path: *

Credential:

Delimiter:

Encoding:

Exclude:

Filter:

☐ Force

Include:

☐ Raw

ReadCount:

Stream:

Tail:

TotalCount:

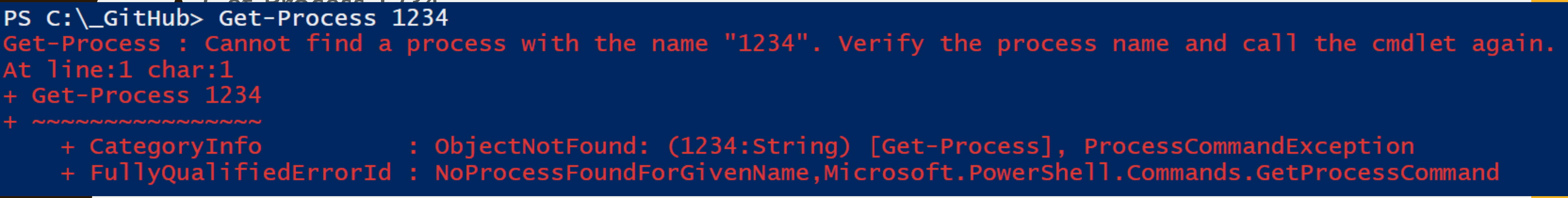
☐ UseTransaction

☐ Wait

Common Parameters

REVIEWING ERRORS

- Get-Process 1234

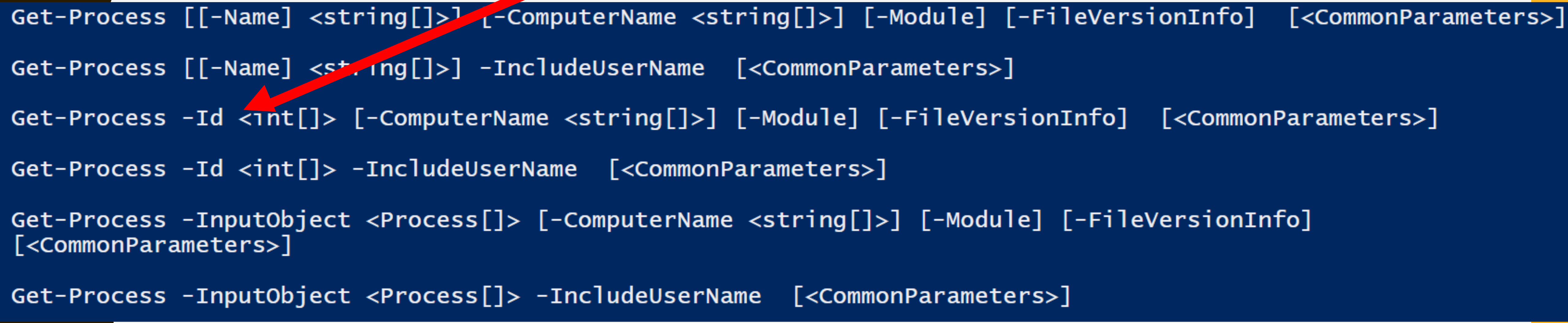


```
PS C:\_GitHub> Get-Process 1234
Get-Process : Cannot find a process with the name "1234". Verify the process name and call the cmdlet again.
At line:1 char:1
+ Get-Process 1234
+ ~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound: (1234:String) [Get-Process], ProcessCommandException
    + FullyQualifiedErrorId : NoProcessFoundForGivenName,Microsoft.PowerShell.Commands.GetProcessCommand
```

- **Cannot find a process with the name "1234". Verify the process name and call the cmdlet again.**

REVIEWING ERRORS

But it's in the first position?



```
Get-Process [[-Name] <string[]>] [-ComputerName <string[]>] [-Module] [-FileVersionInfo]  [<CommonParameters>]

Get-Process [[-Name] <string[]>] -IncludeUserName  [<CommonParameters>]

Get-Process -Id <int[]> [-ComputerName <string[]>] [-Module] [-FileVersionInfo]  [<CommonParameters>]

Get-Process -Id <int[]> -IncludeUserName  [<CommonParameters>]

Get-Process -InputObject <Process[]> [-ComputerName <string[]>] [-Module] [-FileVersionInfo]
[<CommonParameters>]

Get-Process -InputObject <Process[]> -IncludeUserName  [<CommonParameters>]
```

- Yes, but the –Id field is not positional! There are no square brackets around it!

PSPROVIDERS

PSPROVIDER'S

- Cd hkcu:
- Cd software

- Get-PSProvider
  - This is a list of data points/storage that PowerShell can interpret into a file system like behavior

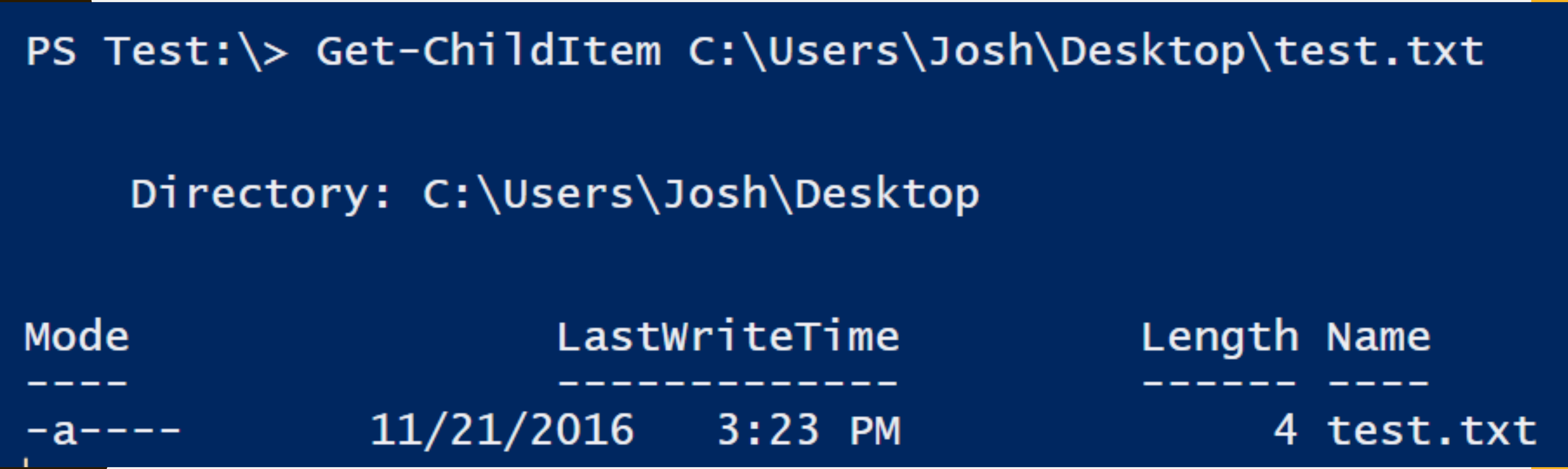
PSDRIVES

PSDRIVE’S

- Get-PSDrive
- Get-Command –Noun psdrive
- New-PSDrive –Name Test –PSProvider filesystem –Root $env:USERPROFILE

GETTING MORE DATA

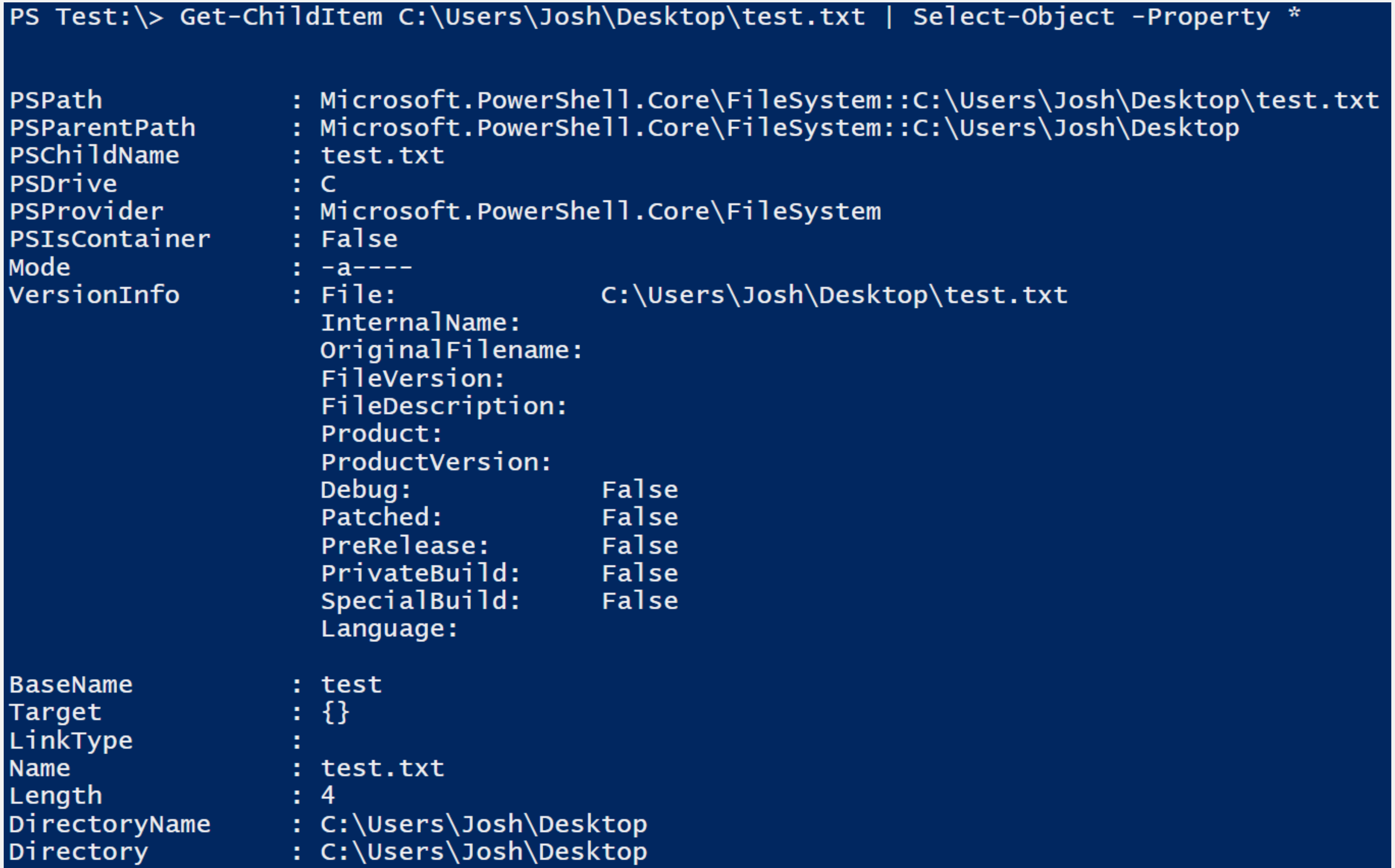
GET-CHILDITEM



```
PS Test:\> Get-ChildItem C:\Users\Josh\Desktop\test.txt


    Directory: C:\Users\Josh\Desktop


Mode                LastWriteTime         Length Name
----                -------------         ------ ----
-a----       11/21/2016   3:23 PM              4 test.txt
```

- But, that can't be it?

YOU'RE RIGHT!



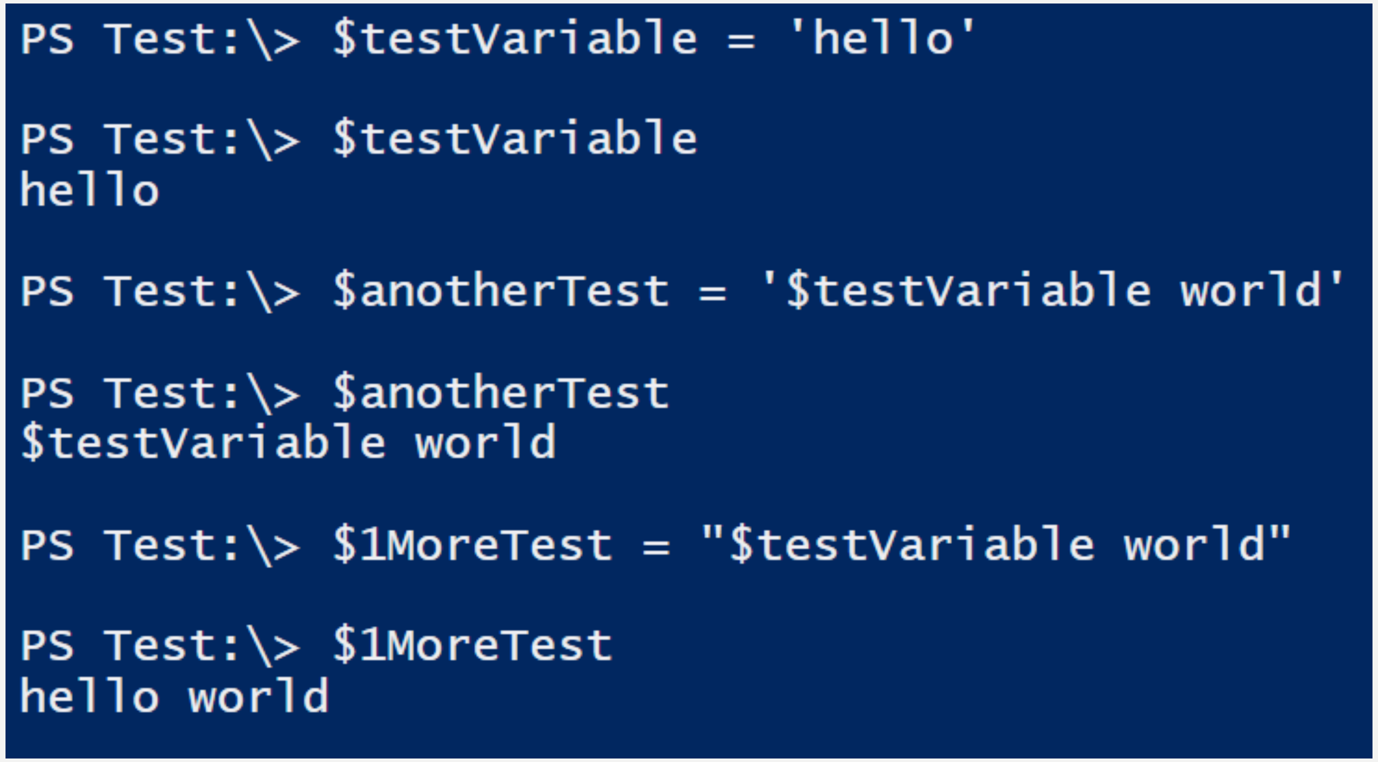
```
PS Test:\> Get-ChildItem C:\Users\Josh\Desktop\test.txt | Select-Object -Property *


PSPath            : Microsoft.PowerShell.Core\FileSystem::C:\Users\Josh\Desktop\test.txt
PSParentPath      : Microsoft.PowerShell.Core\FileSystem::C:\Users\Josh\Desktop
PSChildName       : test.txt
PSDrive           : C
PSProvider        : Microsoft.PowerShell.Core\FileSystem
PSIsContainer     : False
Mode              : -a----
VersionInfo       : File:             C:\Users\Josh\Desktop\test.txt
                    InternalName:
                    OriginalFilename:
                    FileVersion:
                    FileDescription:
                    Product:
                    ProductVersion:
                    Debug:            False
                    Patched:          False
                    PreRelease:       False
                    PrivateBuild:     False
                    SpecialBuild:     False
                    Language:

BaseName          : test
Target            : {}
LinkType          :
Name              : test.txt
Length            : 4
DirectoryName     : C:\Users\Josh\Desktop
Directory         : C:\Users\Josh\Desktop
```

VARIABLES & SYNTAX

VARIABLES

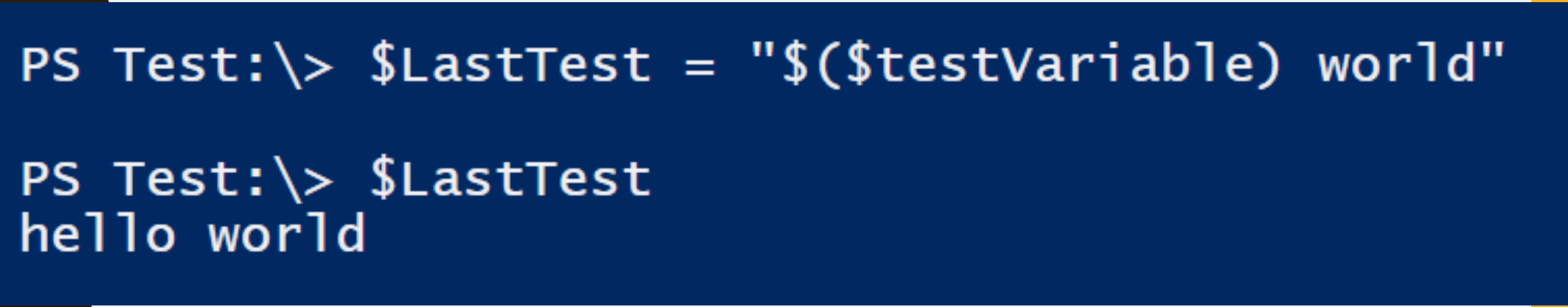
- All variables in PowerShell begin with a $
  - –$testvariable = ‘hello world’
  - –$test__Me = ‘goodbye world’
  - –${some long string can be a variable as well} = ‘hello again’
  - –${even long strings can be integers} = ‘42’

QUOTES



```
PS Test:\> $testVariable = 'hello'

PS Test:\> $testVariable
hello

PS Test:\> $anotherTest = '$testVariable world'

PS Test:\> $anotherTest
$testVariable world

PS Test:\> $1MoreTest = "$testVariable world"

PS Test:\> $1MoreTest
hello world
```

QUOTES

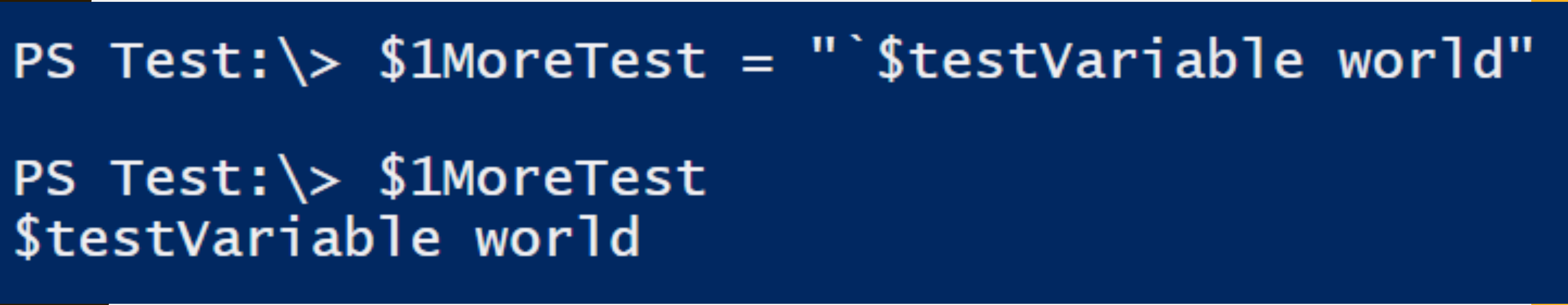


```
PS Test:\> $LastTest = "$($testVariable) world"

PS Test:\> $LastTest
hello world
```

- Everything inside the parentheses is considered literal

BACKTICKS

- In PowerShell, the ` (backtick) is an escape character

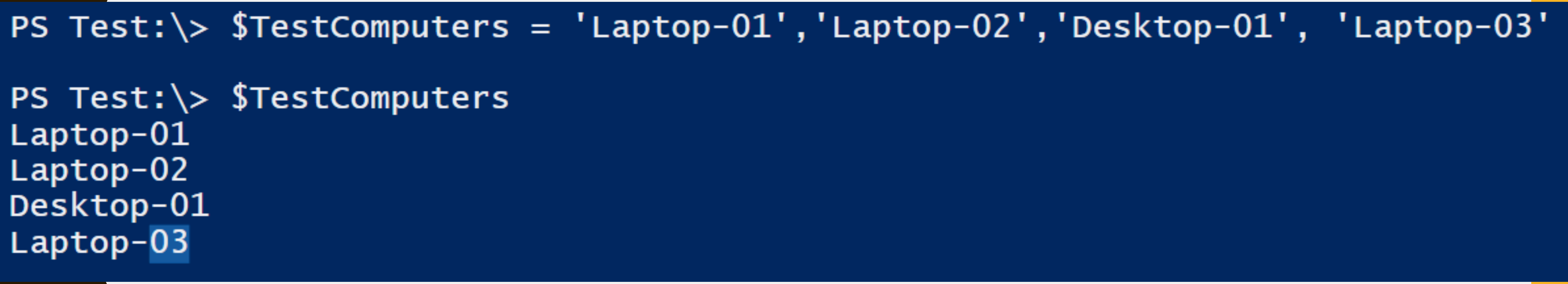


```
PS Test:\> $1MoreTest = "`$testVariable world"

PS Test:\> $1MoreTest
$testVariable world
```

UNDERSTANDIN G ARRAYS

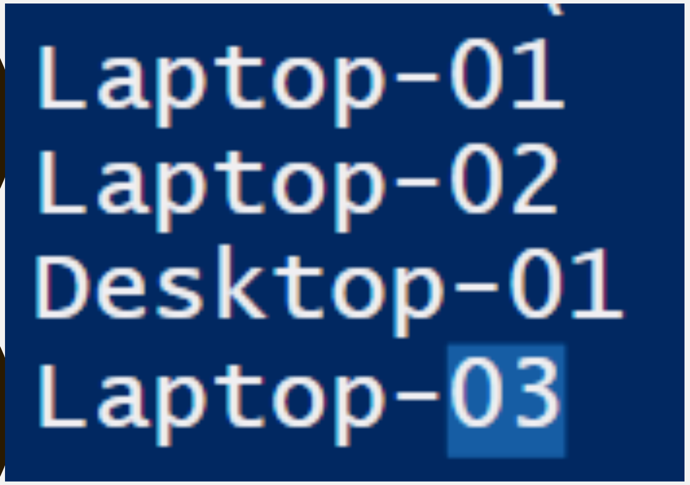
ARRAYS

- In “almost” every programming/scripting language, values start at 0



```
PS Test:\> $TestComputers = 'Laptop-01','Laptop-02','Desktop-01', 'Laptop-03'

PS Test:\> $TestComputers
Laptop-01
Laptop-02
Desktop-01
Laptop-03
```

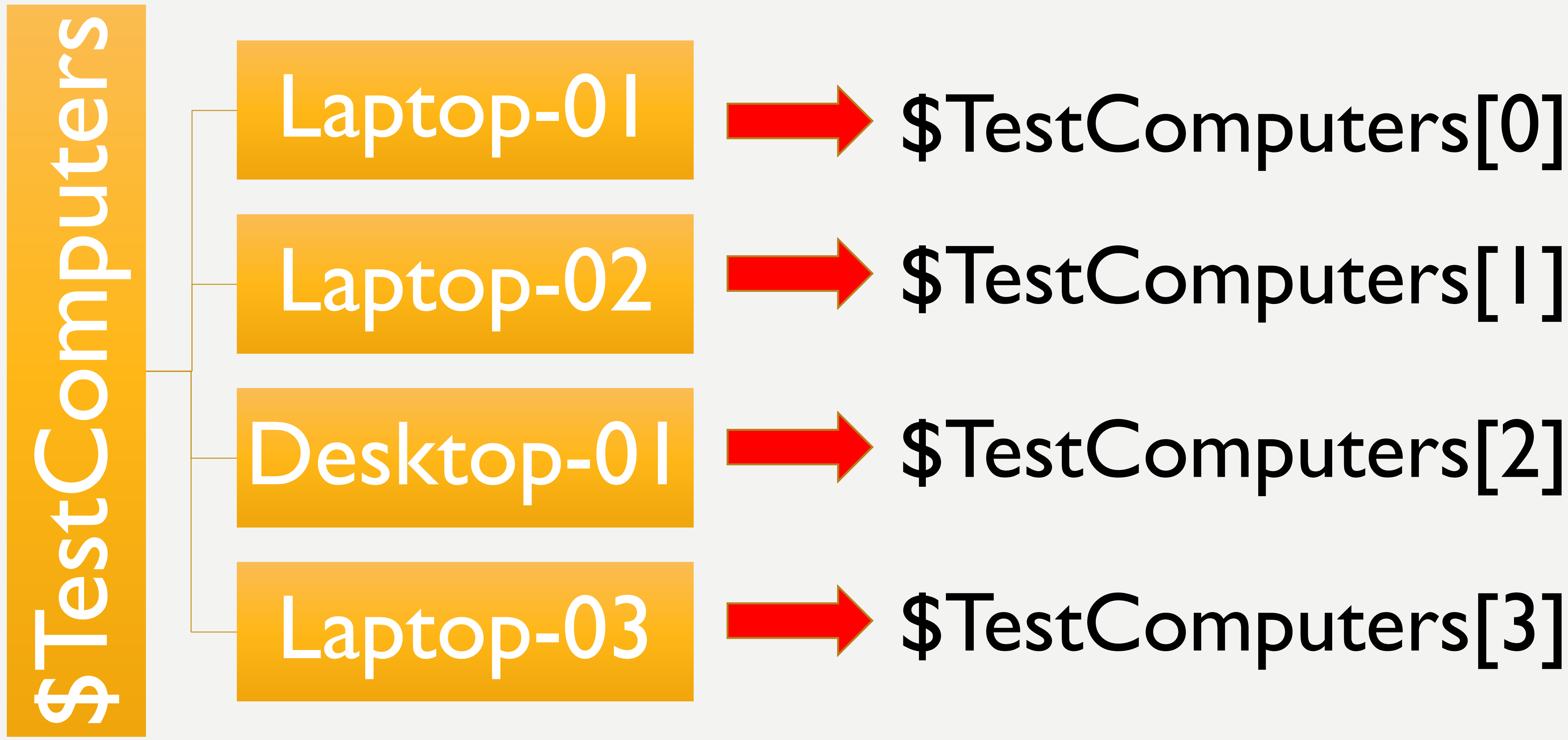
ARRAYS



```
Laptop-01
Laptop-02
Desktop-01
Laptop-03
```

- $TestComputers is an array of Objects
  - $TestComputers.count = 4
- To access the first computer
  - $TestComputers[0]
    - Laptop-01
- To access the last computer
  - $TestComputers[3]
    - Laptop-03

ARRAYS


$TestComputers
Laptop-01
Laptop-02
Desktop-01
Laptop-03
$TestComputers[0]
$TestComputers[1]
$TestComputers[2]
$TestComputers[3]

ARRAYS AND QUOTES

- Consider the following scenario: You are wanting to print out a specific object in an array. How would you do this?

- $PrintValue = “Looking at $TestComputers[0]” ✗
- $PrintValue

- $PrintValue = “Looking at $($TestComputers[0])” ✓
- $PrintValue

DATATYPES & OPERATORS

STRINGS

- $hello = ‘hello’
- $block = @”

This is a test of a

Multiple line string

“@

OPERATORS

- $a = 1
- $b = '3'
- $a + b = 4
- $b + a = 31

- PowerShell will always interpret the left most data type when evaluating the solution.
- The + operator can be used for both addition and concatenation

DATATYPES

- $input = Read-host “What’s the ultimate number?”

- Towel

- [int]$input = Read-Host “What’s the ultimate number?”

- Towel
- Error

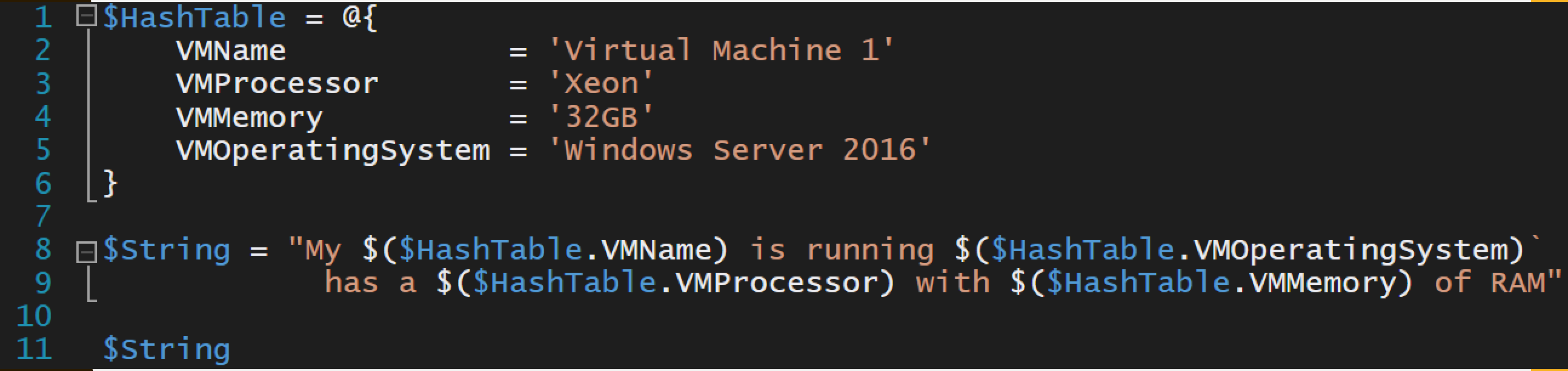
- 42
- No Error

HASHTABLES

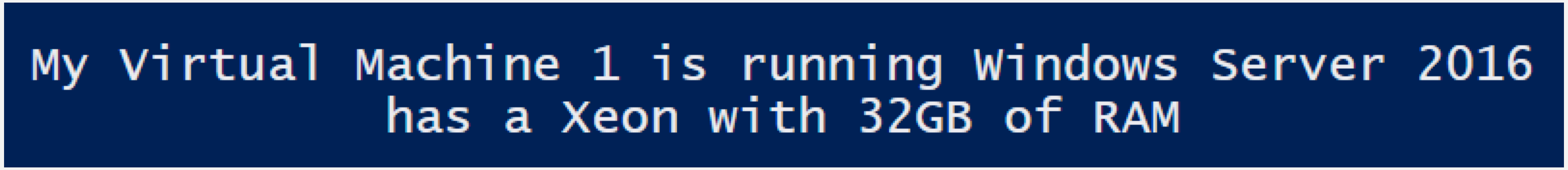
HASHTABLES

- Starts with @{}
- It’s basically a Key = Value pair inside the @{}
- Example:
- @{‘Key’=‘Value’}
- Also called a Dictionary table
- PowerShell = An automation scripting engine
- Example:
- @{‘PowerShell’=‘An automation scripting engine’}

HASHTABLES



```powershell
$HashTable = @{
    VMName              = 'Virtual Machine 1'
    VMProcessor         = 'Xeon'
    VMMemory            = '32GB'
    VMOperatingSystem = 'Windows Server 2016'
}

$String = "My $($HashTable.VMName) is running $($HashTable.VMOperatingSystem)`
            has a $($HashTable.VMProcessor) with $($HashTable.VMMemory) of RAM"

$String
```



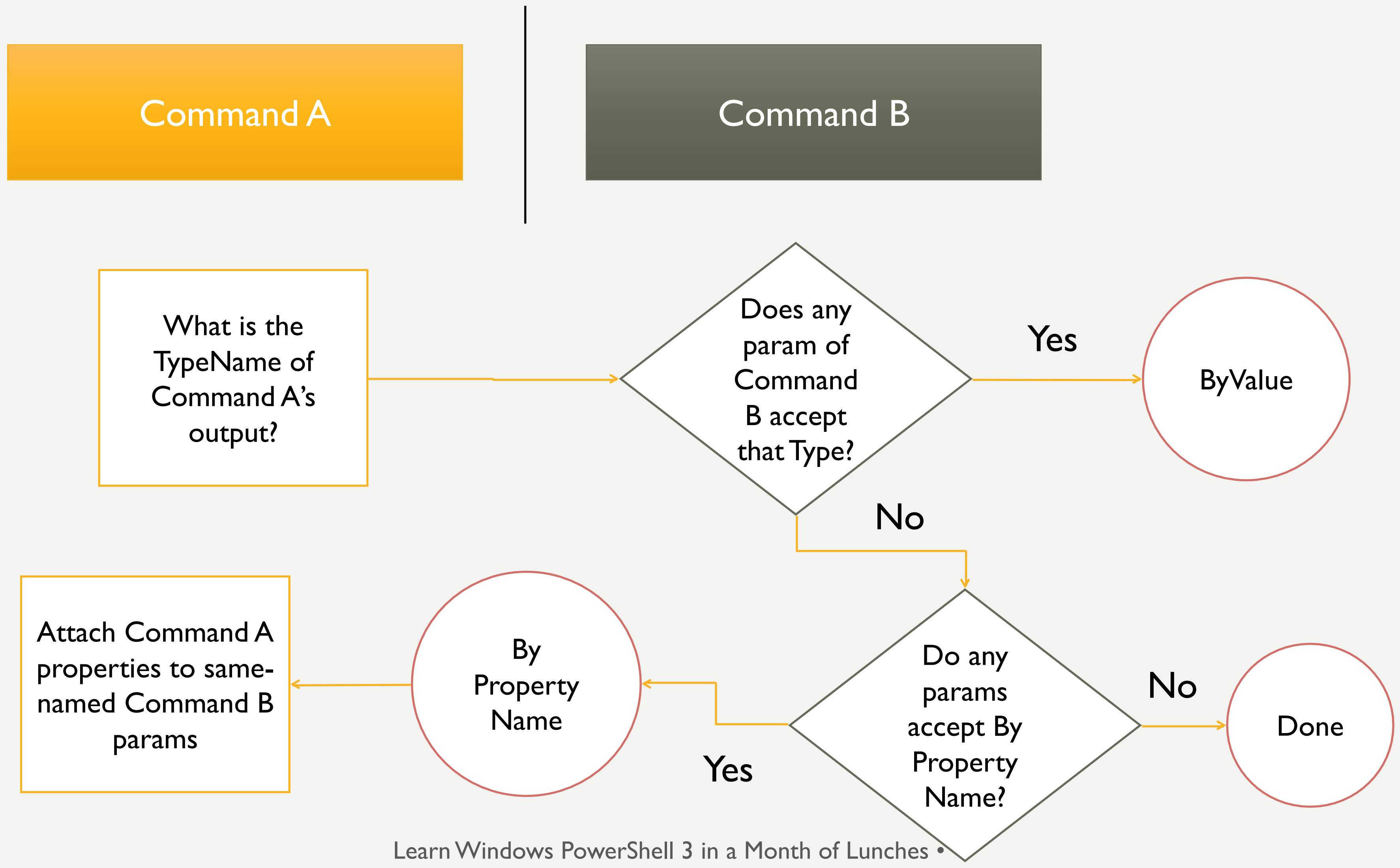
```
My Virtual Machine 1 is running Windows Server 2016
            has a Xeon with 32GB of RAM
```

THE PIPELINE

PIPELINE

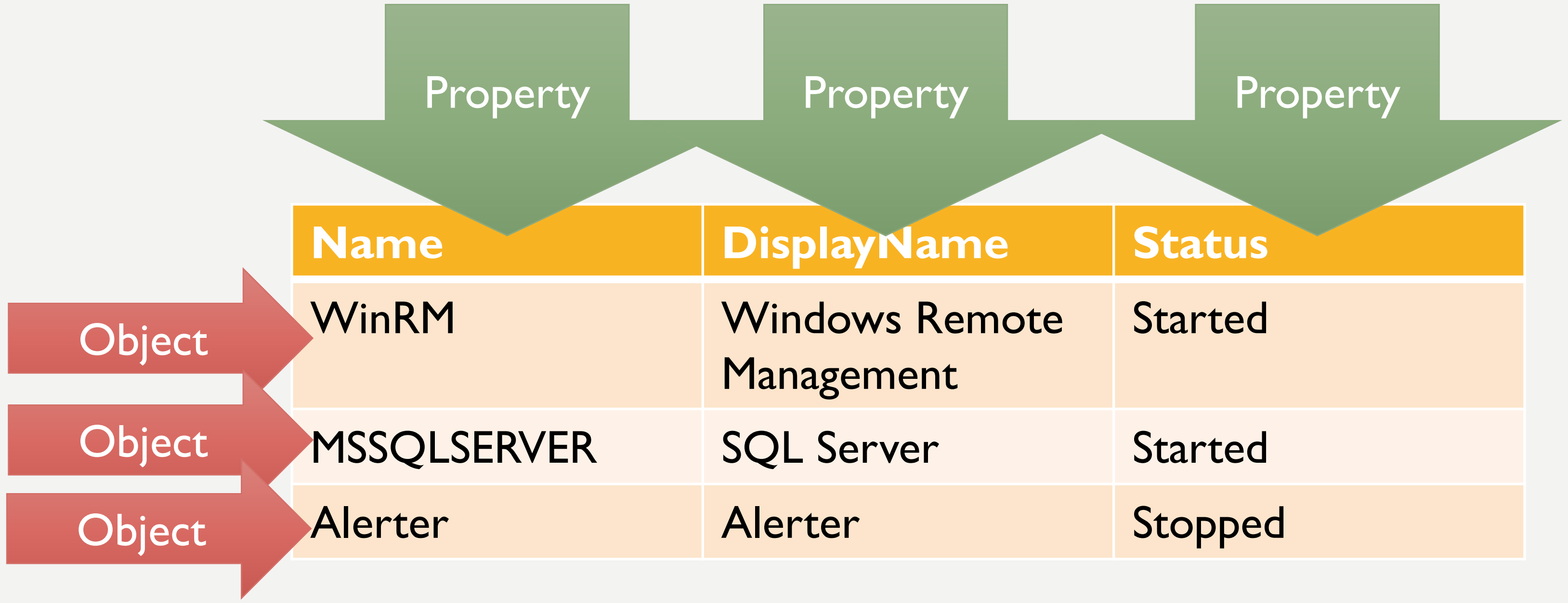
- Get-ChildItem C:\Windows\System32 | Where-Object { $_.BaseName -contains 'cmd' }

PIPELINE PARAMETER BINDING


Command A
Command B
What is the TypeName of Command A's output?
Does any param of Command B accept that Type?
Yes
ByValue
No
Do any params accept By Property Name?
No
Done
Yes
By Property Name
Attach Command A properties to same-named Command B params

POWERSHELL OBJECTS

"OBJECT" PROPERTIES



Name	DisplayName	Status
WinRM	Windows Remote Management	Started
MSSQLSERVER	SQL Server	Started
Alerter	Alerter	Stopped

“OBJECT” METHODS

~~WMI~~ CIM

BASIC FUNCTIONS