

PS C:\>



```
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

PS C:\Users\manoj\Downloads\$env:windir\

Name                           Value
----                           --
PSVersion                      3.0
PSEdition                      PSDesiredVersion
PSCompatibleVersions            3.0, 2.0, 1.0
PowerShellVersion               3.0.0.0
OutputEncoding                  UTF8
PSRemotingProtocolVersion      2.0
PSDefaultParameterSet           None
PSModulePath                    C:\Windows\system32\WindowsPowerShell\v1.0\Modules\
```

# Windows PowerShell Foundations

for Microsoft IT Professionals

# Welcome!

- Introductions
- Break / Meal Times
- Housekeeping

# About This Course

- Utilizes *Learn Windows PowerShell*
- 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.

# Day 1

- Meeting Windows PowerShell
- Using the Help System
- Running Commands
- Working with Providers
- Connecting Commands in the Pipeline
- Extending the Shell
- Working with Objects

# Day 2

- The Pipeline, Deeper
- Formatting
- Filtering and Comparisons
- A Practical Example
- Remote Control
- Using Windows Management Instrumentation
- Multitasking with Background Jobs

# Day 3

- Working with Groups of Objects
- Security
- Variables
- Input and Output
- Sessions for Remote Control
- Turning a Command into a Script
- Improving the Parameterized Script
- Advanced Remoting Configuration
- Using Regular Expressions

# Labs

- Your class materials should include a printed lab guide for you to work from.
- This includes sample answers in the back. However, please avoid using those answers as hints or reminders – it's important that you ask *questions* if you're stuck.
- Labs require a Windows 7/8/10 or Windows Server 2012 computer; most labs can be accomplished on Windows 7/8/10 or Windows Server 2008 R2 as well, if PowerShell v3 or above is installed.
- Some classes may utilize additional labs to give you more hands-on experience.

# Lab Environment Notes

- Please ensure that PowerShell is not loading any extensions at start-up – we will cover how to check this shortly.
- For now, please avoid using any third-party editors or console applications – stick with what's built into Windows to ensure a consistent lab experience.

# By the Way...

- This course is about **PowerShell**, not **PowerPoint** – so the slides will be minimal.  
The majority of the instructor time is intended to be demonstration.
- You're welcome to follow along with demos – **however**, don't get so caught up in  
doing so that you **miss the demo**.
- The slides will mainly be used to present each module's agenda.

>

DAY 1



## Module 1

### Before We Begin

- PowerShell System Requirements
- Installing PowerShell v3

# Setting Expectations

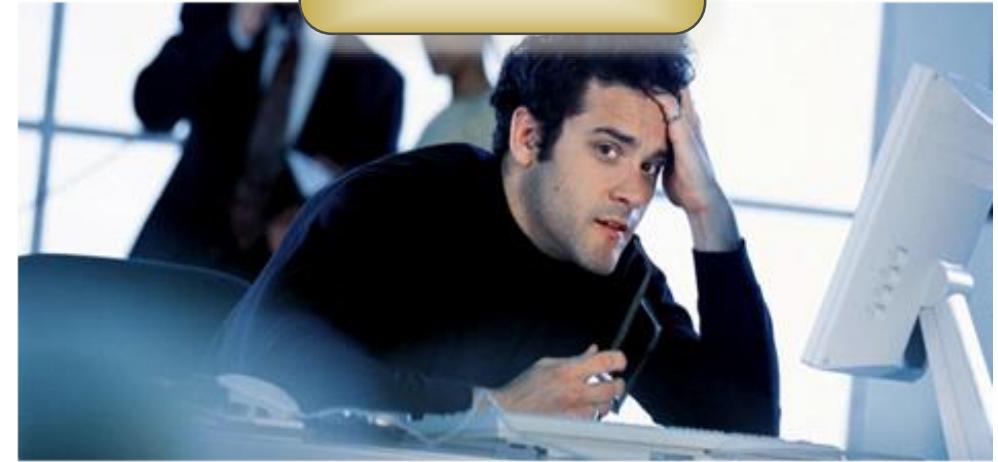
- **Target Audience**
  - Tailored for the IT pro that needs to improve management and automation
  - Fast paced for the real world
- **Suggested Prerequisites/Supporting Material**
  - Experience working as a Windows IT pro/Admin/Help Desk
  - Get answers in the forums at PowerShell.Org
  - Learn Windows PowerShell 3 in a Month of Lunches Book by Don Jones and Jeffery Hicks

# Module Overview

- The purpose for PowerShell
- Installing PowerShell – Windows Management Framework
- Launching PowerShell for the administrator
- Customize the shell for comfort
- Getting familiar with the shell

# The Purpose to PowerShell

YOU



THEM

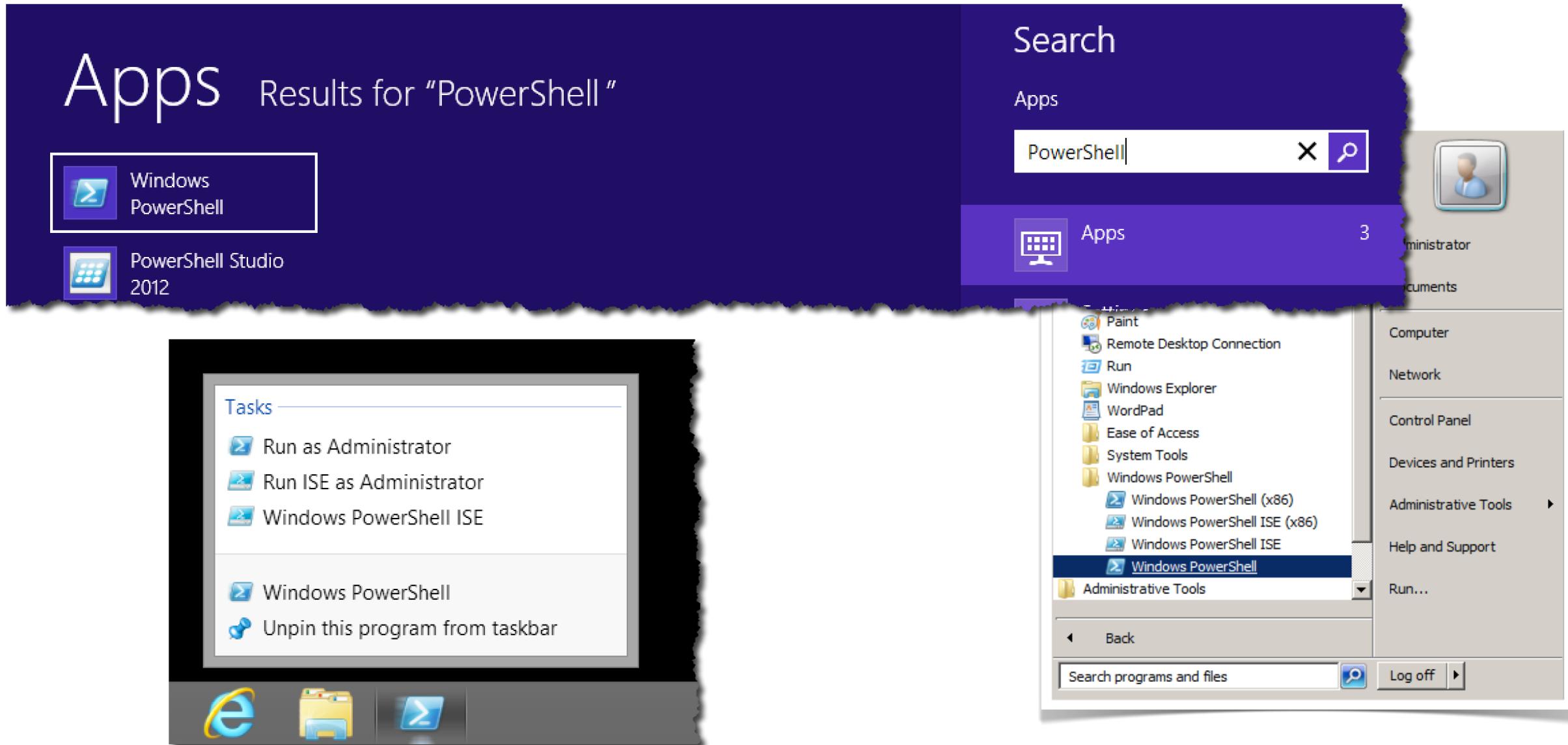


- Improved management and automation
- Manage real-time
- Manage large scale

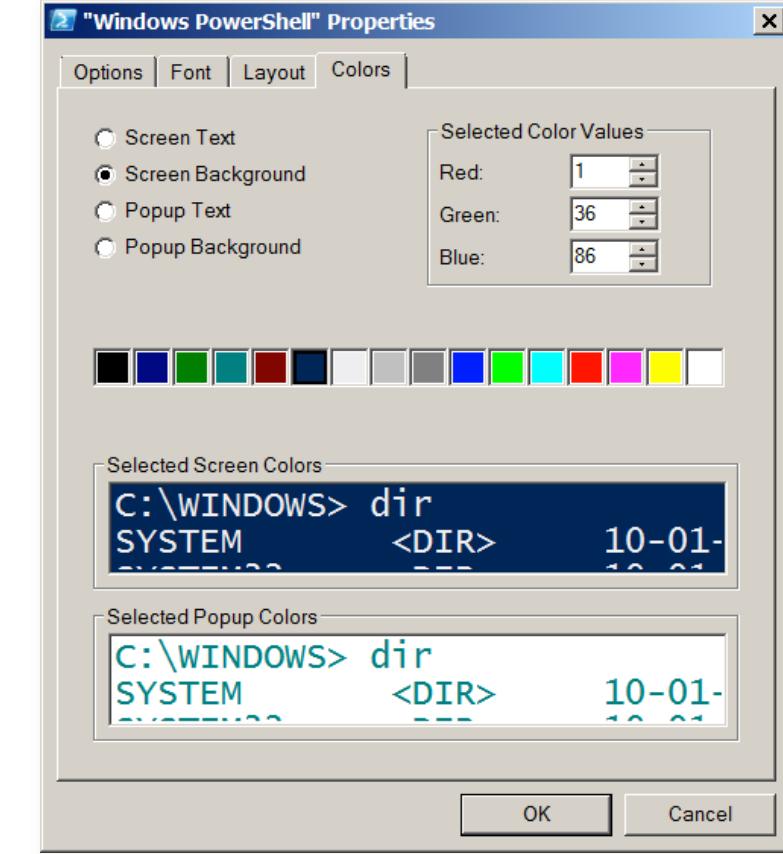
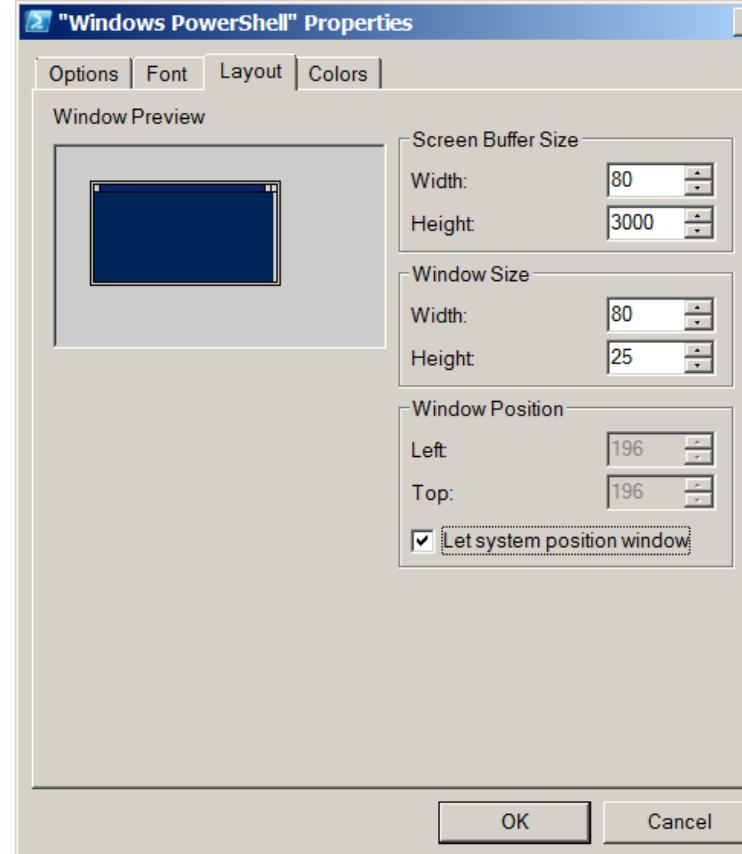
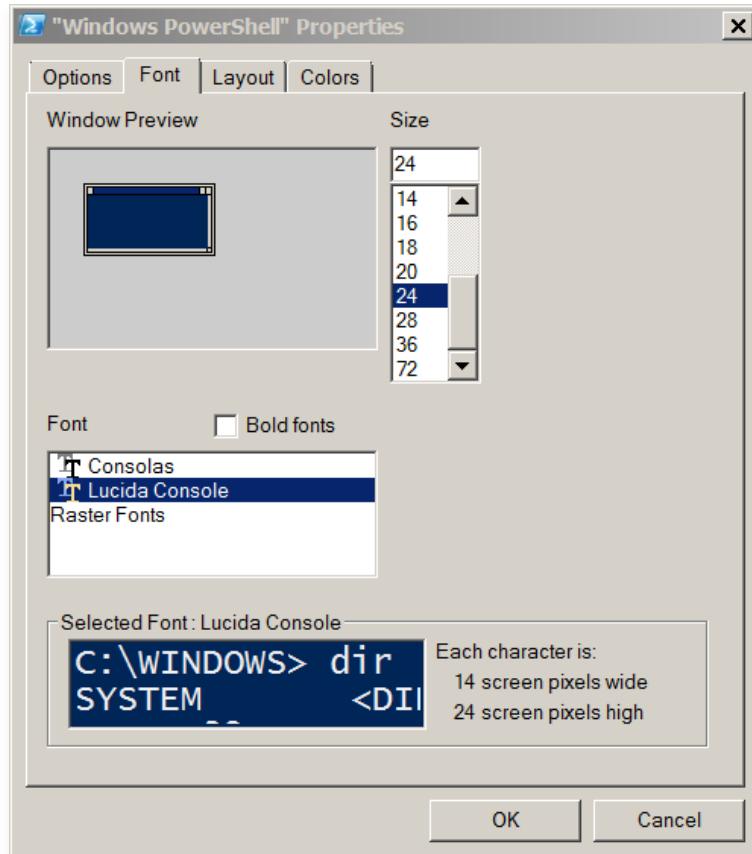
# Installing PowerShell – Windows Management Framework

- PowerShell V5 – Windows 10 and Server 2012
- PowerShell V3 – Windows 8 and Server 2012
- PowerShell V2 – Windows 7 and Server 2008
- Download the Windows Management Framework 3.0 at
- [http://www.microsoft.com/en-  
us/download/details.aspx?id=34595](http://www.microsoft.com/en-us/download/details.aspx?id=34595)
- Windows XP and Server 2003 can run V2

# Launching PowerShell for the Administrator

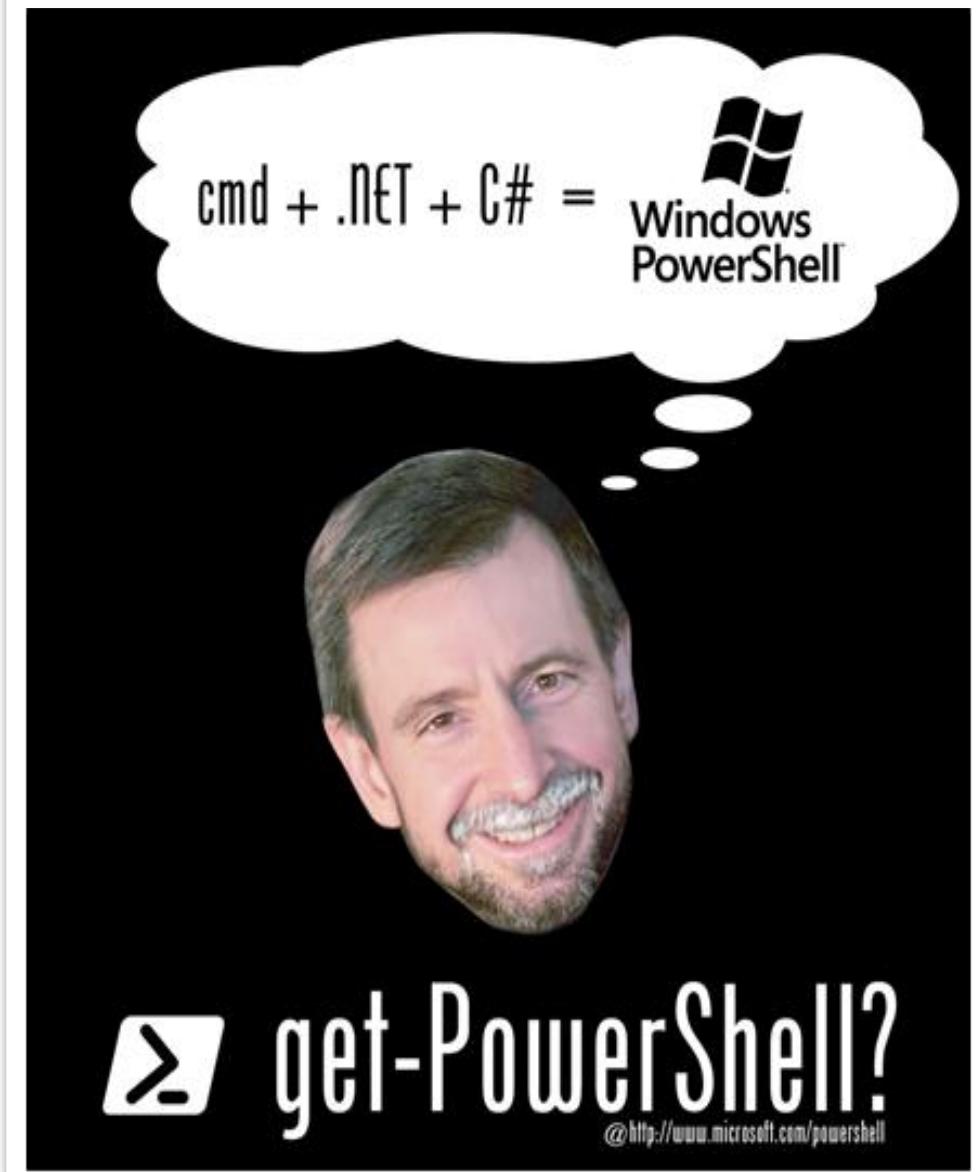


# Customize the shell for comfort



# Getting familiar with the shell

- Cmdlets : Verb – Noun
- Native commands work!
- Examples – Ping, IPConfig, calc, notepad, mspaint
- cls - Clear-Host
- cd - Set-Location
- dir, ls - Get-Childitem
- type, cat - Get-Content
- Copy, cp - Copy-item





## Module 2

Meeting Windows PowerShell

- Choose Your Weapon
  - The Console Window
  - The Integrated Scripting Environment
- It's Typing Class All Over Again!
- What Version is This?
- Common Points of Confusion
  - 32- and 64-bit
  - Running as Administrator
- Lab

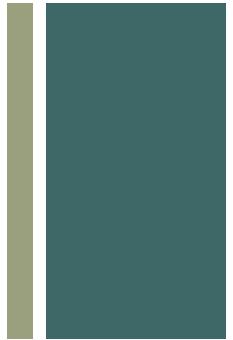


## Module 3

### Using the Help System

- The Help System: How You Discover Commands
- Updatable Help
- Asking for Help
- Using Help to Find Commands
- Interpreting the Help
  - Parameter Sets and Common Parameters
  - Optional and Mandatory Parameters
  - Positional Parameters
  - Parameter Values
  - Examples
- Accessing “About” Topics
- Accessing Online Help
- Lab

# Three Crucial Commands



□ Help





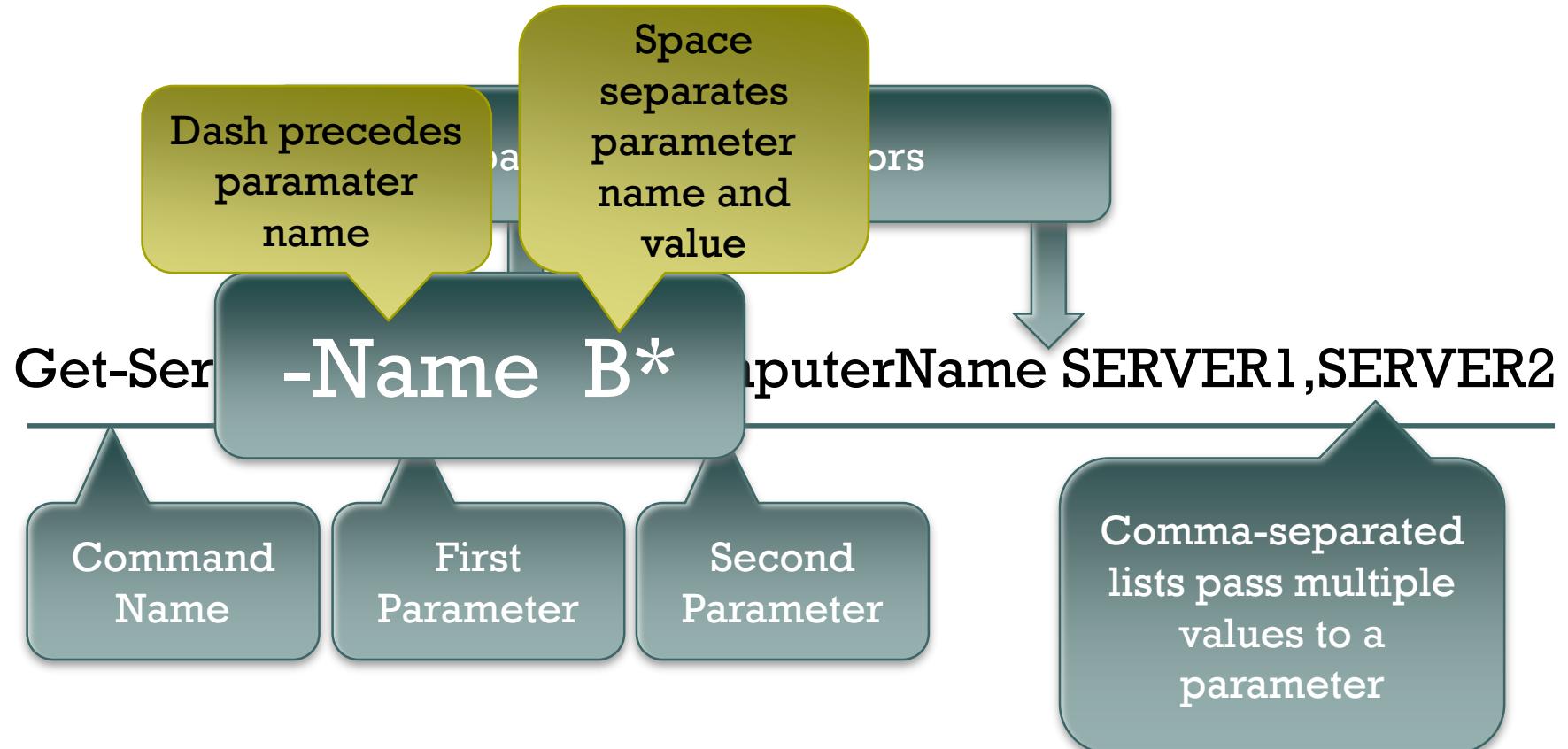
## Module 4

### Running Commands

- Not Scripting: Just Running Commands
- The Anatomy of a Command
- The Cmdlet Naming Convention
- Aliases: Nicknames for Commands
- Taking Shortcuts
  - Truncating Parameter Names
  - Parameter Name Aliases
  - Positional Parameters
- Cheating, a Bit: Show-Command
- Support for External Commands
- Dealing With Errors
- Common Points of Confusion
  - Typing Cmdlet Names
  - Typing Parameters
- Lab

&gt;

# Anatomy of a Command



>

# 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



## Module 5

### Working with Providers

- What are Providers?
- How the File System is Organized
- How the File System is Like Other Data Stores
- Navigating the File System
- Using Wildcards and Literal Paths
- Working with Other Providers
- **Lab**



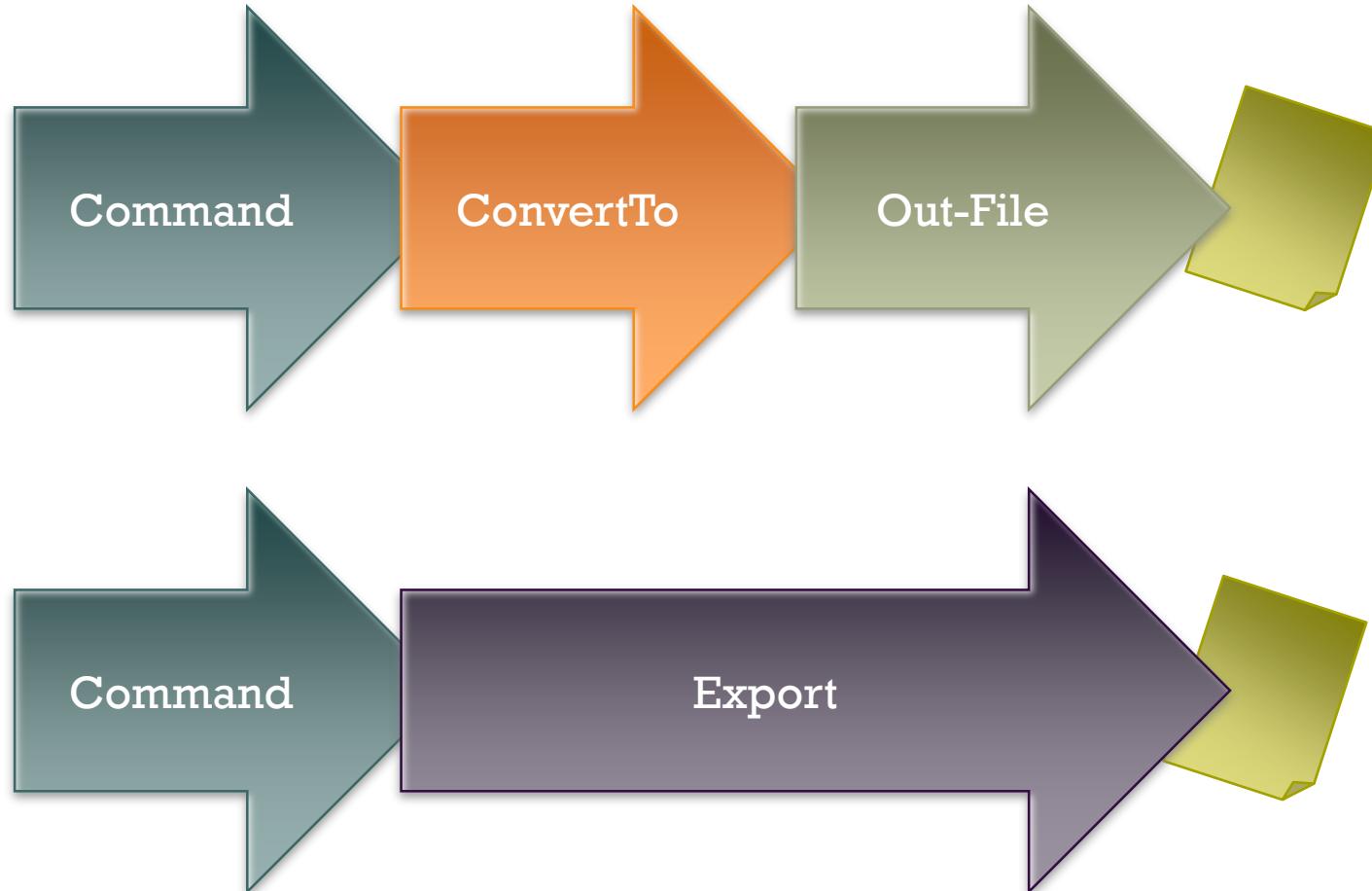
## Module 6

### The Pipeline: Connecting Commands

- Connect One Command to Another: Less Work For You!
- Exporting to a CSV or XML File
- Piping to a File or Printer
- Converting to HTML
- Using Cmdlets That Modify the System: Killing Processes and Stopping Services
- Common Points of Confusion
- **Lab**

>

# Export vs. ConvertTo



What's the pipeline and what does it do?

**Pipe character located above the Enter key**

Connects cmdlets to produce better results

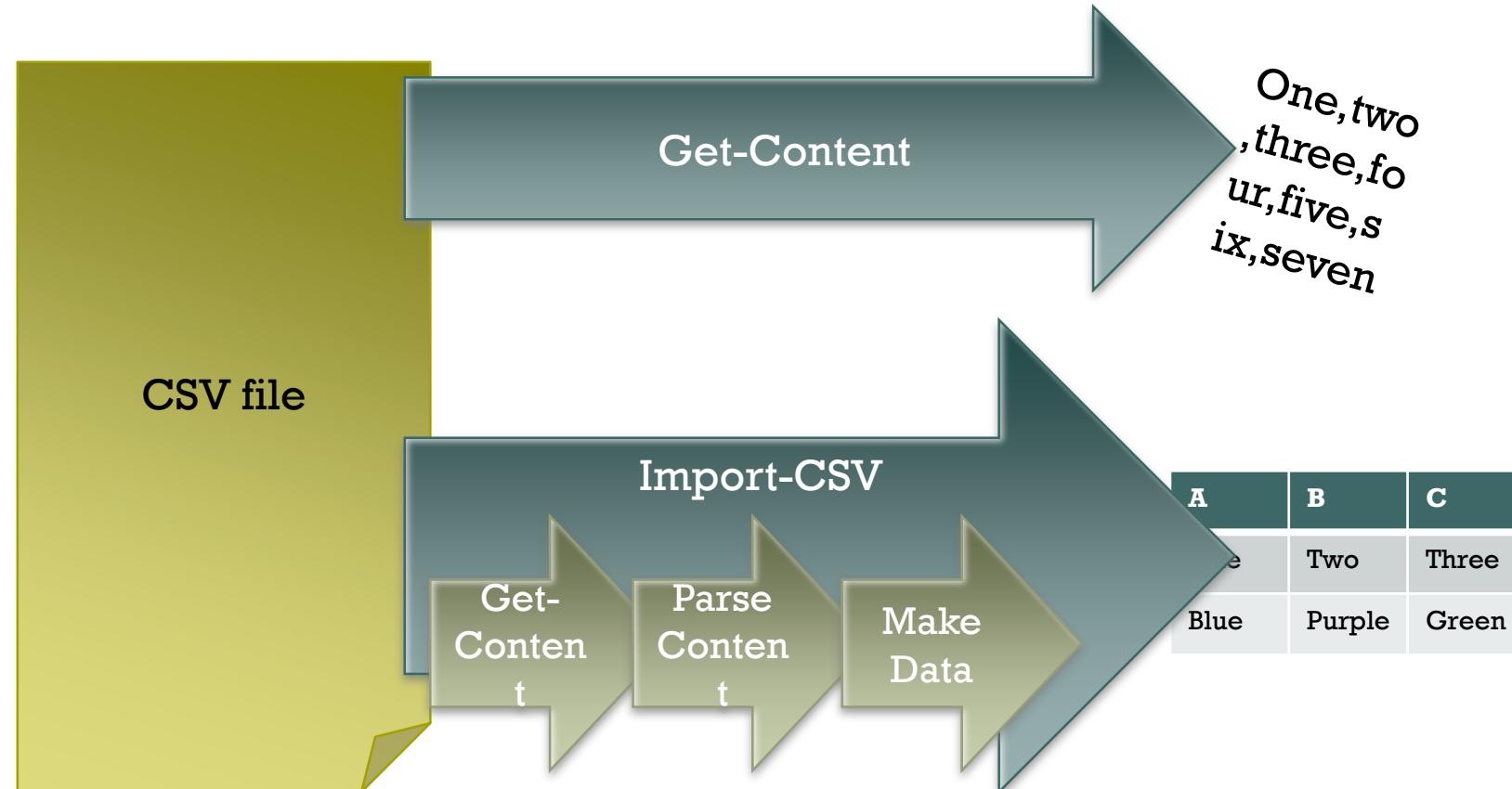


```
PS C:\> Get-Service | Select-Object name, status | Sort-Object name
```

Can be broken into several lines to increase readability

```
PS C:\> Get-Service |  
  >> Select-Object name, status |  
  >> Sort-Object name
```

# Get-Content vs. Import



&gt;

## Exporting/Importing CSV

The screenshot shows three windows illustrating the process of exporting and importing CSV files from Windows PowerShell.

- Administrator: Windows PowerShell**: A window titled "Administrator: Windows PowerShell" with the following command history:

```
PS C:\>
PS C:\> Get-Process | export-csv c:\Proc.csv
PS C:\> notepad c:\proc.csv
PS C:\>
```
- Proc.csv - Notepad**: A window titled "Proc.csv - Notepad" containing the contents of the exported CSV file:

```
#TYPE System.Diagnostics.Process
"__NounName", "Name", "Handles", "VM", "WS", "PM", "NPM", "Path", "Company", "CPU", "Fi
"Process", "coherenc e", "50", "19222528", "2859008", "864256", "5424", "C:\Program
"Process", "coherence", "50", "39010304", "2957312", "876544", "5296", "C:\Program F
"Process", "coherence", "40", "41709568", "2764800", "798720", "6000", "C:\Program F
"Process", "conhost", "89", "78626816", "7950336", "2347008", "8528", "C:\Windows\sv
"Pro
```
- Administrator: Windows PowerShell**: A window titled "Administrator: Windows PowerShell" with the following command history:

```
"Pro
"Pro
PS C:\> import-csv c:\proc.csv
```

The bottom window displays the imported CSV data as a table:

__NounName	: Process
Name	: coherence
Handles	: 52
VM	: 19222528
WS	: 2801664
PM	: 876544
NPM	: 5424
Path	: C:\Program Files (x86)\Parallels\Parallels Tools\s ervices\coherence.exe

&gt;

# Exporting/Importing XML

```
Administrator: Windows PowerShell
PS C:\> Get-Process | Export-clixml c:\ref.xml
PS C:\> _
```

```
Administrator: Windows PowerShell
PS C:\> Compare-Object -ReferenceObject (Import-clixml c:\ref.xml) -DifferenceObject (Get-Process) -Property name
name
-----
notepad
```

	sideIndicator
notepad	->

```
PS C:\>
```

## > Other files and printers

The screenshot displays three windows illustrating the use of PowerShell cmdlets to interact with services and their corresponding display names.

**Administrator: Windows PowerShell** window (top):

```
PS C:\>
PS C:\> Get-Service > C:\serv.txt
PS C:\> Get-Service | Out-File c:\serv2.txt
PS C:\> Get-Service | Out-Printer
PS C:\> Notepad c:\serv.txt
PS C:\> Notepad c:\serv2.txt
PS C:\>
```

**serv.txt - Notepad** window (middle-left):

Status	Name	DisplayName
-----	-----	-----
Running	ADWS	Active Directory Web Services
Stopped	AeLookupSVC	Application Experience
Stopped	ALG	Application Layer Gateway Service

**serv2.txt - Notepad** window (bottom-right):

Status	Name	DisplayName
-----	-----	-----
Running	ADWS	Active Directory Web Services
Stopped	AeLookupSVC	Application Experience
Stopped	ALG	Application Layer Gateway Service

## > Displaying information in a GUI

The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell" with the following command history:

```
PS C:\>
PS C:\> Get-process | Out-GridView
PS C:\> Get-Service | Out-GridView
PS C:\>
```

Below the PowerShell window are two separate "get-process | Out-GridView" and "get-service | Out-GridView" windows.

**get-process | Out-GridView** window content:

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
52	5	856	2,736	18	0.02	1,500	coherence
51	5	868	2,856	37	0.03	2,364	coherence
40	6	796	2,708	40	0.02	2,372	coherence
42	6	1,724	4,564	56	0.13	2,144	conhost
38	5	1,556	3,924	46	0.06	2,248	conhost
458	12	1,896	4,152	47	0.20	320	csrss
235	13	20,220	14,292	63	7.91	384	csrss
195	19	4,716	11,436	58	1.28	1,328	dfsrs
123	13	2,392	6,072	40	0.09	1,800	dfssvc
2,652	3,652	46,116	47,264	91	1.08	1,364	dns
70	7	1,704	4,824	53	0.05	2,664	dwm
667	48	30,760	46,040	197	3.27	2,696	explorer
0	0	0	24	0		0	Idle
103	14	3,104	5,444	40	0.11	1,392	ismserv
1,069	85	22,568	27,344	364	5.72	492	lsass
147	7	2,064	3,904	17	0.06	500	lsm

**get-service | Out-GridView** window content:

Status	Name	DisplayName
Running	ADWS	Active Directory Web Services
Stopped	AeL...	Application Experience
Stopped	ALG	Application Layer Gateway...
Stopped	App...	Application Identity
Stopped	App...	Application Information
Stopped	App...	Application Management
Stopped	asp...	ASP.NET State Service
Running	Aud...	Windows Audio Endpoint B...
Running	Aud...	Windows Audio

# > Making a webpage of information

```
Administrator: Windows PowerShell
PS C:\>
PS C:\> Get-Service | Export-Csv c:\serv.csv
PS C:\> Get-Service | ConvertTo-Csv | Out-File c:\serv.csv
PS C:\>
PS C:\> Get-Service | ConvertTo-Html -Property DisplayName, Status |
>> Out-File c:\serv.htm
>>
PS C:\> -
```

HTML TABLE - Windows Internet Explorer

The screenshot shows a Windows Internet Explorer window titled "HTML TABLE - Windows Internet Explorer". The address bar displays "C:\serv.htm". The main content area contains an HTML table with two columns: "Displayname" and "status". The table lists several services and their current status.

Displayname	status
Active Directory Web Services	Running
Application Experience	Stopped
Application Layer Gateway Service	Stopped
Application Identity	Stopped
Application Information	Stopped
Application Management	Stopped

## Cmdlets that kill

Stop-Process / kill  
Stop-service  
\$ConfirmPreference

- \$WhatIfPreference
- -Confirm
- -Whatif

```
PS C:\> Remove-Item c:\serv.htm -whatif
what if: Performing operation "Remove File" on Target "c:\serv.htm".
```

```
PS C:\> Remove-Item c:\serv.htm -Confirm
```

```
Confirm
Are you sure you want to perform this action?
Performing operation "Remove File" on Target "c:\serv.htm".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] suspend [?] Help
(default is "Y"):
```

PS C:\>



```
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

PS C:\Users\mano\Downloads\$env:temp\b1e

Name                           Value
----                           ---
PSVersion                      3.0
PSCultureName                  [1252, 2000, 9]
PSCompatibleVersions           {1.0, 2.0, 3.0}
PSRemotingProtocolVersion      2.2
PSDefaultVersion                3.0
PSReadLineVersion               1.0
PSHostUserInterfaceVersion     2.0
PSHostEnvironmentVersion        3.0
PSHostProcessVersion            3.0
```

PS C:\Users\mano\Downloads\\$env:temp\b1e >

# COMPLETE REVIEW LAB 1

## Based on Chapter 1-6



## Module 7

### Adding Commands

- How One Shell Can Do Everything
- About Product-Specific “Management Shells”
- Extensions: Finding and Adding Snap-Ins
- Extensions: Finding and Adding Modules
- Playing With a New Module
- Profile Scripts: Preloading Extensions When the Shell Starts
- Common Points of Confusion
- **Lab**

# Extending the Shell

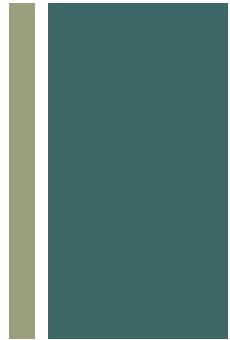
## The PSSnapin Way

- Find it:  
`Get-PSSnapin -registered`
- Load it:  
`Add-PSSnapin -Name name`
- Discover it:  
`Get-Command -PSSnapin name`

## The Module Way

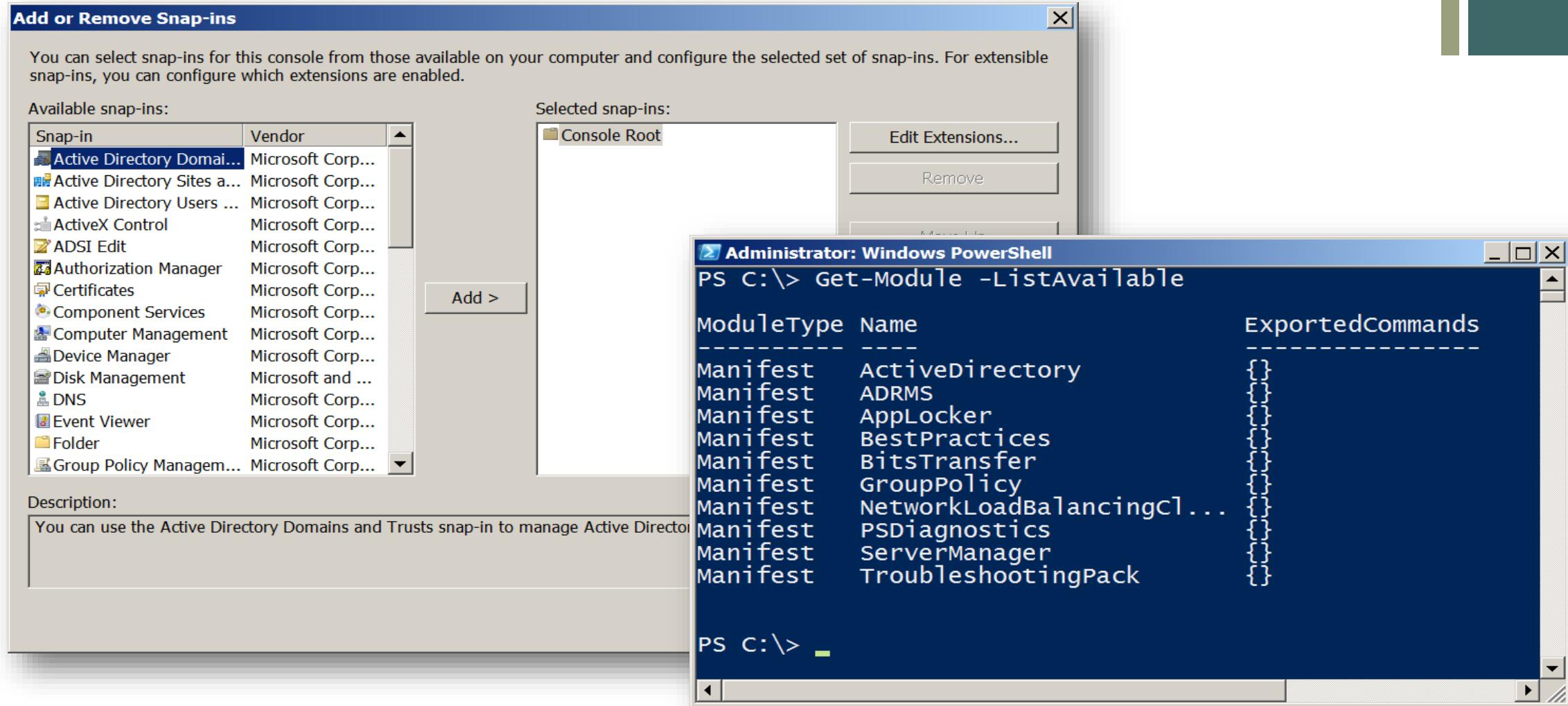
- Find it:  
`Get-Module -ListAvailable`
- Load it:  
`Import-Module -Name name`
- Discover it:  
`Get-Command -Module name`

# Three Crucial Commands



- Help
- Get-Command
-

# Like the MMC - One Shell does it all



# Finding & Adding Snap-ins

```
Administrator: Windows PowerShell
PS C:\> Get-PSSnapin -Registered
Name      : sqlServerCmdletsnapin100
PSVersion : 2.0
Description : This is a Powershell snap-in that includes various SQL Server cmdlets.

Name      : sqlServerProvidersnapin100
PSVersion : 2.0
Description : SQL Server Provider

PS C:\> Add-PSSnapin sql*
PS C:\> Get-Command -Module sql*
 CommandType      Name
 -----      -----
 Cmdlet      Convert-UrnToPath
 Cmdlet      Decode-SqlName
 Cmdlet      Encode-SqlName
 Cmdlet      Invoke-PolicyEvaluation
 Cmdlet      Invoke-Sqlcmd
                                         Definition
                                         -----
                                         Convert-UrnToPath [-urn] <String>...
                                         Decode-SqlName [-SqlName] <String...
                                         Encode-SqlName [-SqlName] <String...
                                         Invoke-PolicyEvaluation [-Policy]...
                                         Invoke-Sqlcmd [[-Query] <String>]...

PS C:\>
```

# Finding & Adding Modules

```
Administrator: Windows PowerShell
PS C:\>
PS C:\> Get-Module -ListAvailable
ModuleType Name
---- 
Manifest ActiveDirectory
Manifest ADRMS
Manifest AppLocker
Manifest BestPractices
Manifest BitsTransfer
Manifest GroupPolicy
Manifest NetworkLoadBalancingC...
Manifest PSDiagnostics
Manifest ServerManager
Manifest TroubleshootingPack

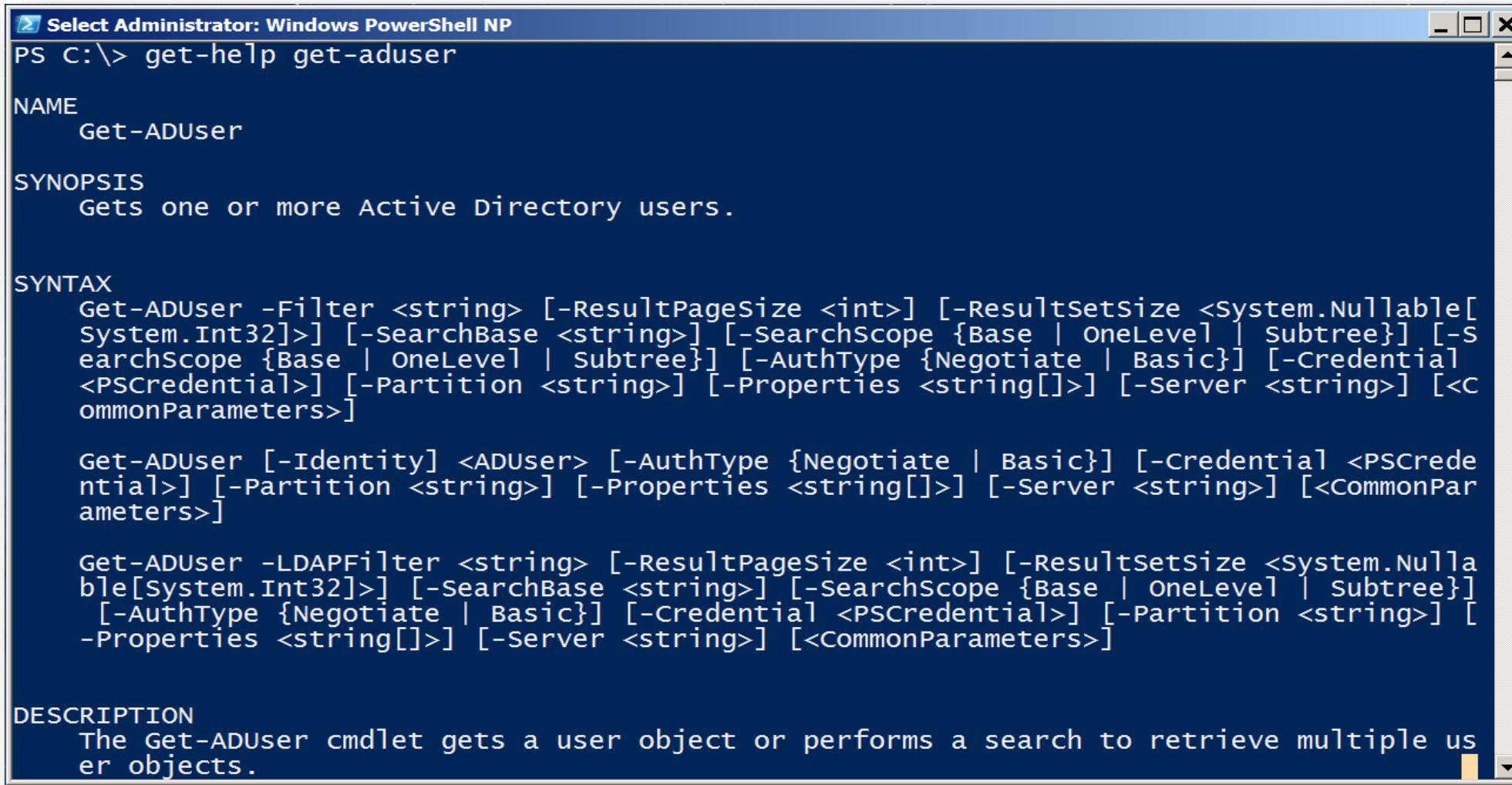
PS C:\> Import-Module act*
PS C:\> Get-Command -Module Act*
CommandType      Name
---- 
Cmdlet          Add-ADComputerServiceAccount
Cmdlet          Add-ADDomainControllerPasswo...
Cmdlet          Add-ADFineGrainedPasswordPol...
```

A red arrow points from the command `Get-Module -ListAvailable` in the first code block to the `ExportedCommands` column of the table in the second code block.

Two red arrows point from the command `Get-Command -Module Act*` in the third code block to the `Name` and `Definition` columns of the table in the fourth code block.

PowerShell V3  
dynamically imports  
modules when you  
use the cmdlet

# Discovering new commands



PS C:\> get-help get-aduser

**NAME**  
Get-ADUser

**SYNOPSIS**  
Gets one or more Active Directory users.

**SYNTAX**

```
Get-ADUser -Filter <string> [-ResultPageSize <int>] [-ResultsetSize <System.Nullable[System.Int32]>] [-SearchBase <string>] [-SearchScope {Base | OneLevel | Subtree}] [-SearchScope {Base | OneLevel | Subtree}] [-AuthType {Negotiate | Basic}] [-Credential <PSCredential>] [-Partition <string>] [-Properties <string[]>] [-Server <string>] [<CommonParameters>]
```

```
Get-ADUser [-Identity] <ADUser> [-AuthType {Negotiate | Basic}] [-Credential <PSCredential>] [-Partition <string>] [-Properties <string[]>] [-Server <string>] [<CommonParameters>]
```

```
Get-ADUser -LDAPFilter <string> [-ResultPageSize <int>] [-ResultsetSize <System.Nullable[System.Int32]>] [-SearchBase <string>] [-SearchScope {Base | OneLevel | Subtree}] [-AuthType {Negotiate | Basic}] [-Credential <PSCredential>] [-Partition <string>] [-Properties <string[]>] [-Server <string>] [<CommonParameters>]
```

**DESCRIPTION**  
The Get-ADUser cmdlet gets a user object or performs a search to retrieve multiple user objects.

# The real world of cmdlets

The image displays three overlapping Windows PowerShell windows, each showing a different cmdlet being used.

- Top Window:** Administrator: Windows PowerShell NP  
PS C:\>  
PS C:\> Import-Module ServerManager  
PS C:\> gcm -Module Server\*
- Middle Window:** Administrator: Windows PowerShell NP  
PS C:\> Get-WindowsFeature  
Display Name  
-----
  - [ ] Active Directory Certificate Services
  - [ ] Certification Authority
  - [ ] Certification Authority Web Enrollment
  - [ ] Online Responder
  - [ ] Network Devi
  - [ ] Certificate
  - [ ] Certificate
  - [X] Active Directory
  - [X] Active DirecName  
-----
  - AD-Certificate
  - ADCS-Cert-Authority
  - ADCS-Web-Enrollment
  - ADCS-Online-Cert
- Bottom Window:** Administrator: Windows PowerShell NP  
PS C:\> Add-WindowsFeature telnet-client, telnet-server -Restart  
Success Restart Needed Exit Code Feature Result  
-----
  - True No Success {Telnet Server, Telnet Client}PS C:\>



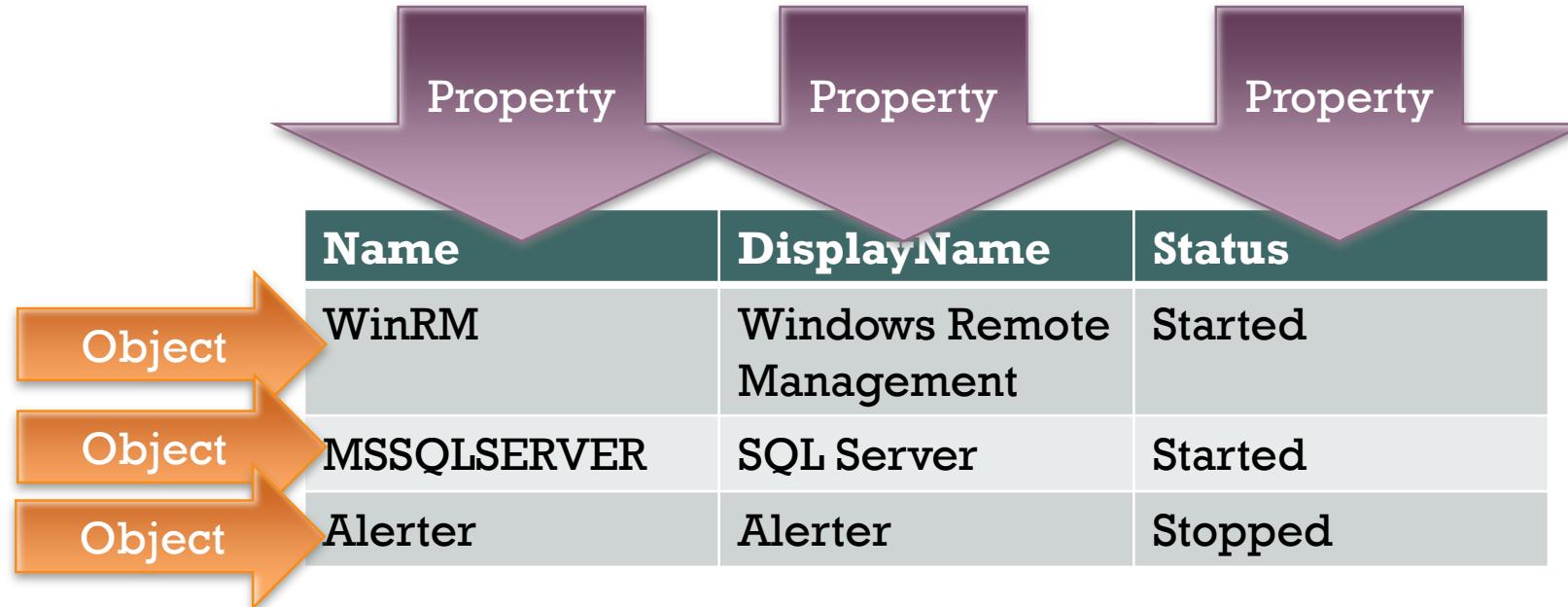
## Module 8

Objects:  
Just Data by Another Name

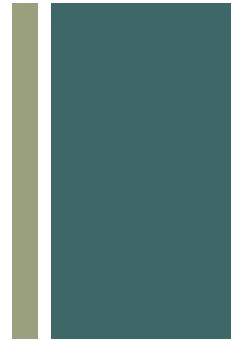
- What are Objects?
- Why PowerShell Uses Objects
- Discovering Objects: Get-Member
- Object Attributes, or “Properties”
- Object Actions, or “Methods”
- Sorting Objects
- Selecting the Properties You Want
- Objects Until the Very End
- Common Points of Confusion
- **Lab**

&gt;

# “Objects”



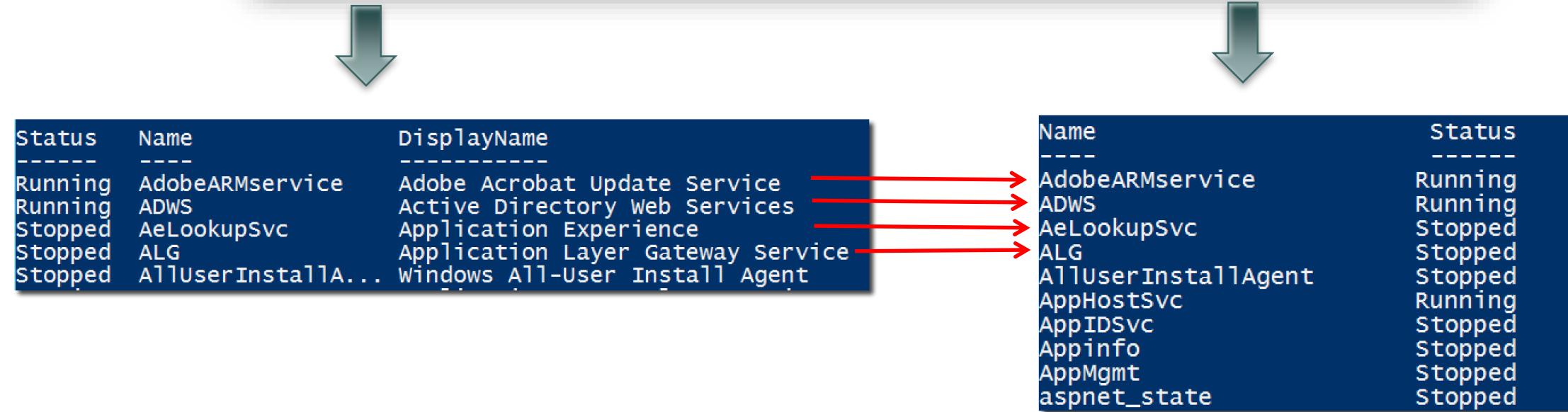
# Three Crucial Commands



- Help
- Get-Command
- Get-Member

# Object across the pipeline

```
PS C:\> Get-Service | select-object name, status
```



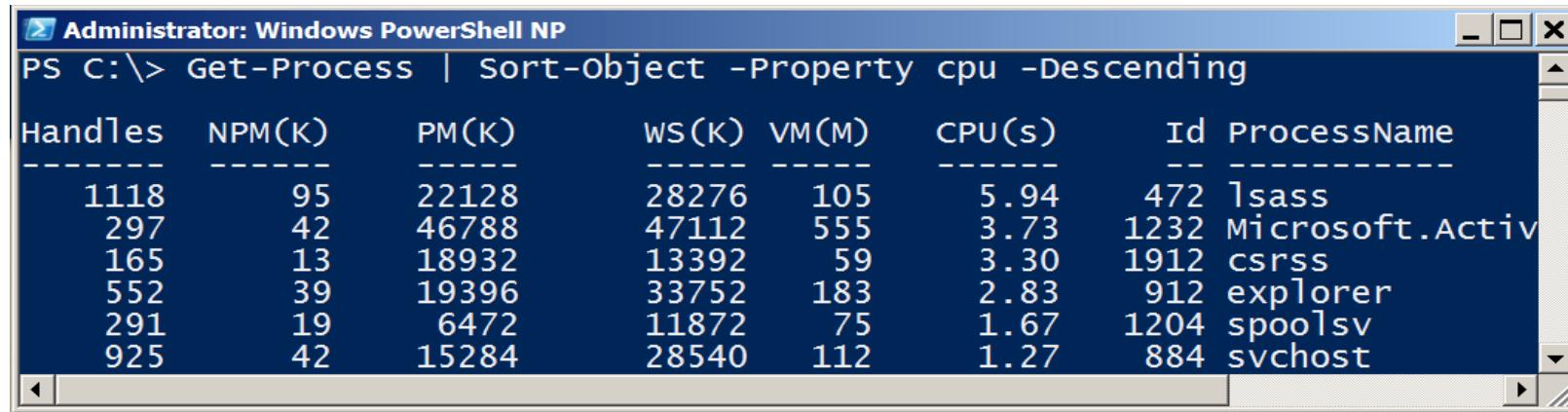
>

# Getting the information you need

- Get-Member (gm)
  - TypeName is a unique Windows assigned name
  - Displays the properties and methods of an object
  - Properties are potential columns of information
  - Methods are the potential actions that can be taken

Name	MemberType	Definition
Handles	AliasProperty	Handles = Handlec
Name	AliasProperty	Name = ProcessNam
NPM	AliasProperty	NPM = NonpagedSys
PM	AliasProperty	PM = PagedMemoryS
VM	AliasProperty	VM = VirtualMemor
WS	AliasProperty	WS = WorkingSet
Disposed	Event	System.EventHandl
ErrorDataReceived	Event	System.Diagnostic
Exited	Event	System.EventHandl
OutputDataReceived	Event	System.Diagnostic
BeginErrorReadLine	Method	System.Void Begin
BeginOutputReadLine	Method	System.Void Begin
CancelErrorRead	Method	System.Void Canc
CancelOutputRead	Method	System.Void Canc
Close	Method	System.Void Close
CloseMainWindow	Method	bool CloseMainWin
CreateObjRef	Method	System.Runtime.Re
Dispose	Method	System.Void Dispo

# Sorting Objects

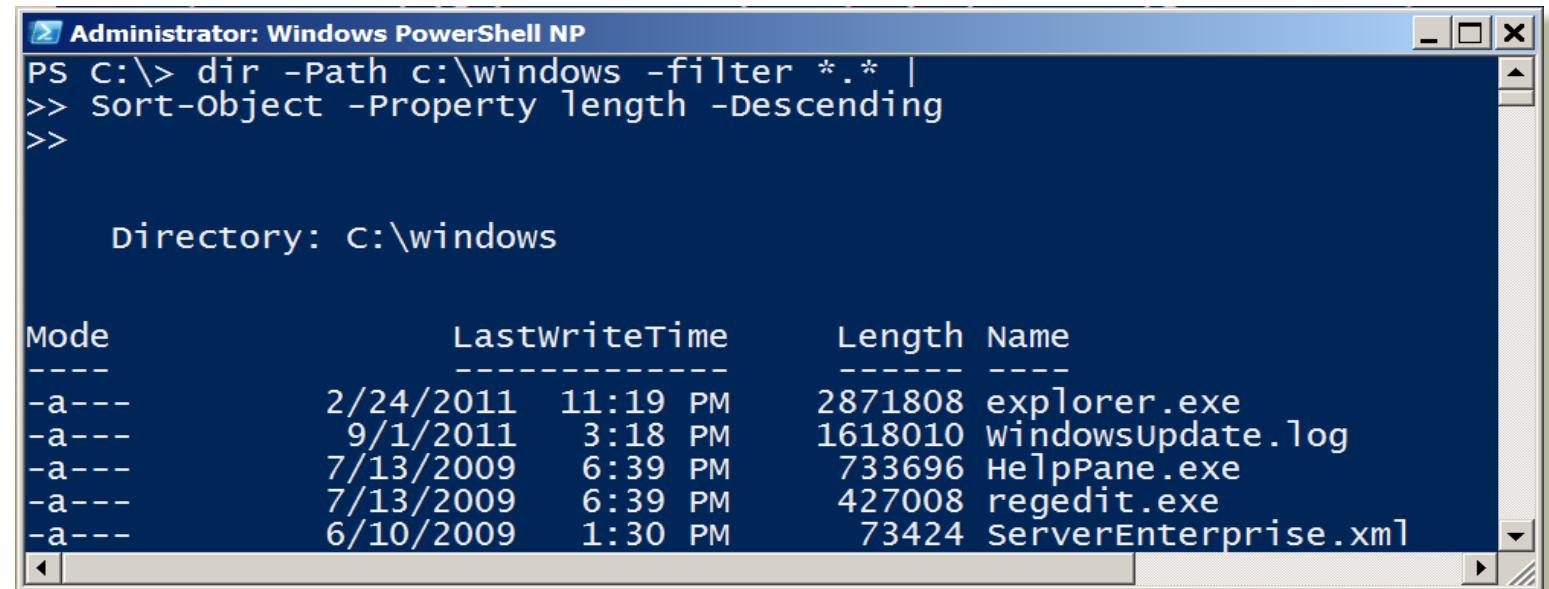


Administrator: Windows PowerShell NP

```
PS C:\> Get-Process | Sort-Object -Property cpu -Descending
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	ID	ProcessName
1118	95	22128	28276	105	5.94	472	lsass
297	42	46788	47112	555	3.73	1232	Microsoft.Activ
165	13	18932	13392	59	3.30	1912	csrss
552	39	19396	33752	183	2.83	912	explorer
291	19	6472	11872	75	1.67	1204	spoolsv
925	42	15284	28540	112	1.27	884	svchost

- Sort-Object sorts properties.
- Use Get-Member to see a list of properties



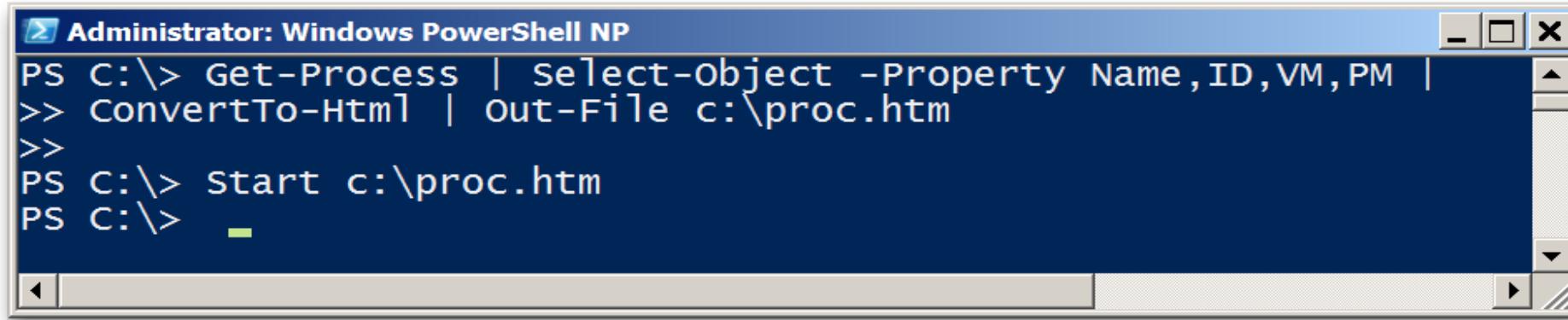
Administrator: Windows PowerShell NP

```
PS C:\> dir -Path c:\windows -filter *.* | Sort-Object -Property length -Descending
>>
```

Directory: C:\windows

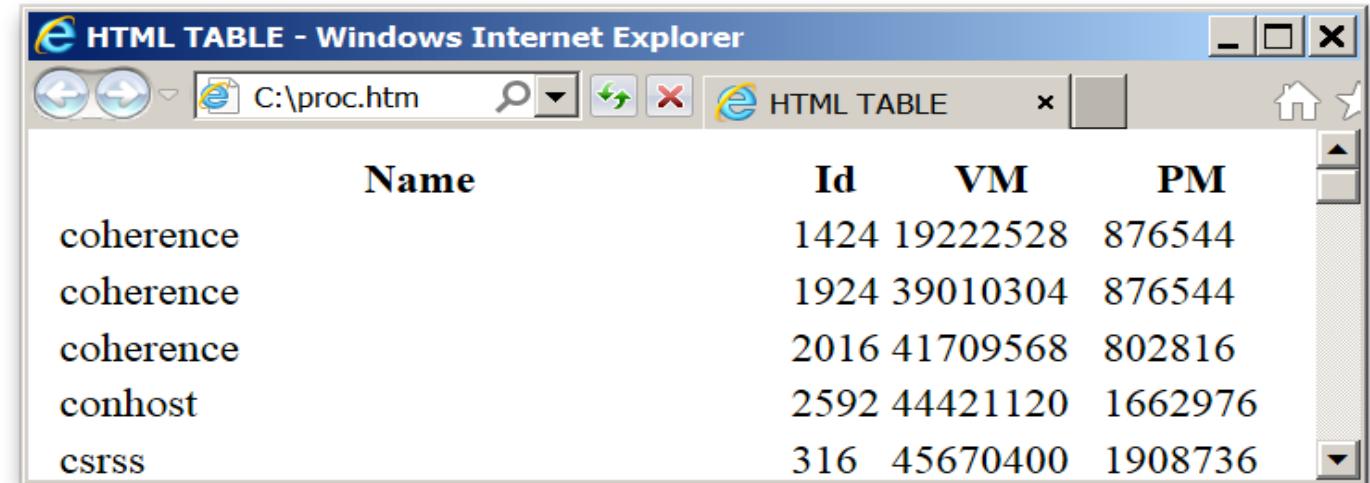
Mode	LastWriteTime	Length	Name
-a---	2/24/2011 11:19 PM	2871808	explorer.exe
-a---	9/1/2011 3:18 PM	1618010	windowsUpdate.log
-a---	7/13/2009 6:39 PM	733696	HelpPane.exe
-a---	7/13/2009 6:39 PM	427008	regedit.exe
-a---	6/10/2009 1:30 PM	73424	ServerEnterprise.xml

# Selecting Objects



```
Administrator: Windows PowerShell NP
PS C:\> Get-Process | Select-Object -Property Name, ID, VM, PM |
>> ConvertTo-Html | Out-File c:\proc.htm
>>
PS C:\> Start c:\proc.htm
PS C:\> -
```

- **Select-Object selects properties.**
- **Use Get-Member to list properties to select from.**
- **-first and -last restrict list of rows displayed.**



Name	Id	VM	PM
coherence	1424	19222528	876544
coherence	1924	39010304	876544
coherence	2016	41709568	802816
conhost	2592	44421120	1662976
csrss	316	45670400	1908736

# Custom Properties



Administrator: Windows PowerShell

```
PS C:\> Get-WmiObject win32_LogicalDisk -filter "deviceID='c:'" |  
    >> Select-Object -Property __Server,  
    >> @{n='FreeGB';e={$_.Freespace /1Gb -as [int]}} |  
    >> Format-Table -AutoSize  
    >>
```

__SERVER	FreeGB
AKITAWIN8	43

```
PS C:\> _
```

# Filter Object Out of the Pipeline

```
Administrator: Windows PowerShell
PS C:\> Get-Service | where-object -FilterScript { $_.status -eq 'Running' }

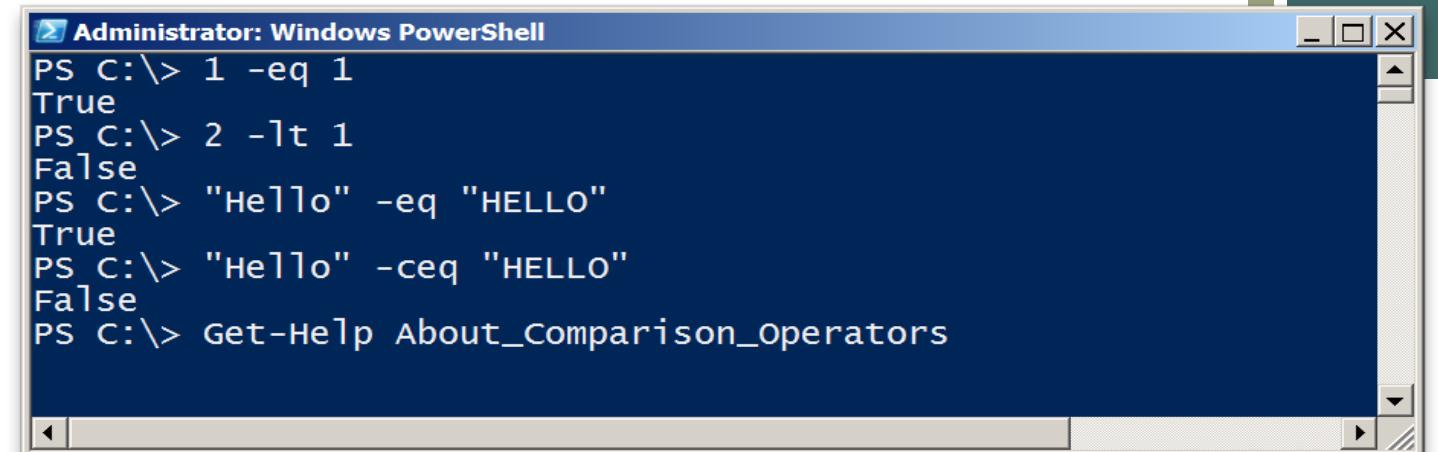
Status      Name          DisplayName
-----      --           -----
Running    ADWS          Active Directory Web Services
Running    AudioEndpointBu... Windows Audio Endpoint Builder
Running    Audiosrv       Windows Audio
Running    BFE           Base Filtering Engine
Running    BITS          Background Intelligent Transfer Ser...
Running    CryptSvc      Cryptographic Services
```

```
Administrator: Windows PowerShell
PS C:\> gsv | ?{$_.status -eq 'Running'}

Status      Name          DisplayName
-----      --           -----
Running    ADWS          Active Directory Web Services
Running    AudioEndpointBu... Windows Audio Endpoint Builder
Running    Audiosrv       Windows Audio
Running    BFE           Base Filtering Engine
Running    BITS          Background Intelligent Transfer Ser...
Running    CryptSvc      Cryptographic Services
```

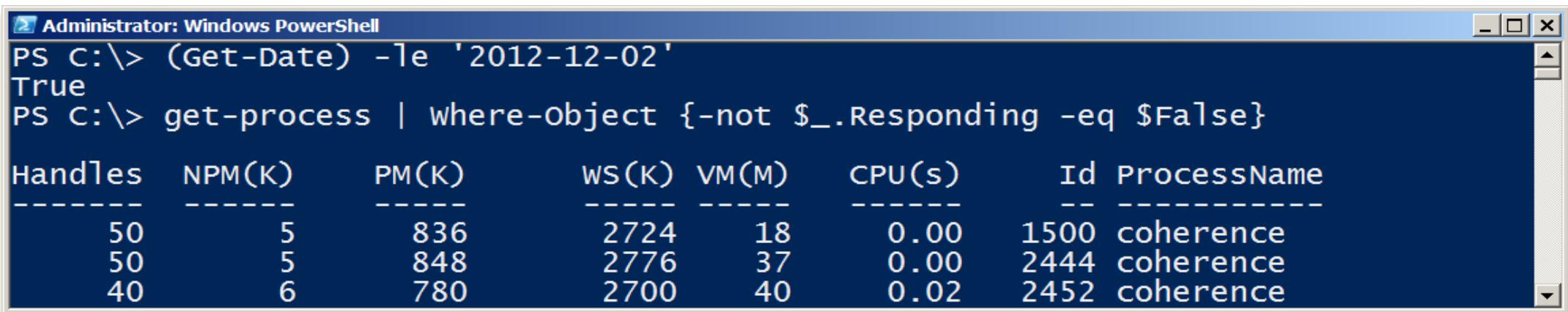
# Comparison Operators

- Comparison returns boolean True or False
- Comparison can be case-sensitive using 'c' prefix
- For complete description, see `About_Comparison`



A screenshot of a Windows PowerShell window titled "Administrator: Windows PowerShell". The window shows the following command and output:

```
PS C:\> 1 -eq 1
True
PS C:\> 2 -lt 1
False
PS C:\> "Hello" -eq "HELLO"
True
PS C:\> "Hello" -ceq "HELLO"
False
PS C:\> Get-Help About_Comparison_Operators
```



A screenshot of a Windows PowerShell window titled "Administrator: Windows PowerShell". The window shows the following commands and output:

```
PS C:\> (Get-Date) -le '2012-12-02'
True
PS C:\> get-process | where-object {-not $_.Responding -eq $False}
```

Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	ID	ProcessName
50	5	836	2724	18	0.00	1500	coherence
50	5	848	2776	37	0.00	2444	coherence
40	6	780	2700	40	0.02	2452	coherence

&gt;

# Methods – When no cmdlet exists

The image shows four nested Windows PowerShell windows, each titled "Administrator: Windows PowerShell".

- Top Window:** Shows the command: `PS C:\> Get-Service bits | foreach{$_ stop()}`. This command attempts to use the `stop()` method on a service object.
- Second Window:** Shows the command: `PS C:\> Get-WmiObject Win32_Service | gm`. This command uses the `gm` alias to get members of the `Win32_Service` class, revealing its type.
- Third Window:** Shows the command: `PS C:\> Get-WmiObject Win32_Service | gm -Name change | fl definition`. This command filters for the `change` method and displays its definition. The output shows the method's parameters: `Change(System.String DisplayName, System.String PathName, System.Byte ServiceType, System.Byte ErrorControl, System.String StartMode, System.Boolean DesktopInteract, System.String StartName, System.String StartPassword, System.String LoadOrderGroup, System.String[] LoadorderGroupDependencies)`.
- Bottom Window:** Shows a partial command: `PS C:\> Get-WmiObject win32_service -filter "name='bits'" | >> ForEach-Object -Process { >> $_.change($null,$null,$null,$null,$null,$null,"P@ssw0rd")}`. This command uses the `change` method on a filtered `win32_service` object, passing it the value "bits" and a password "P@ssw0rd".

>

DAY 2

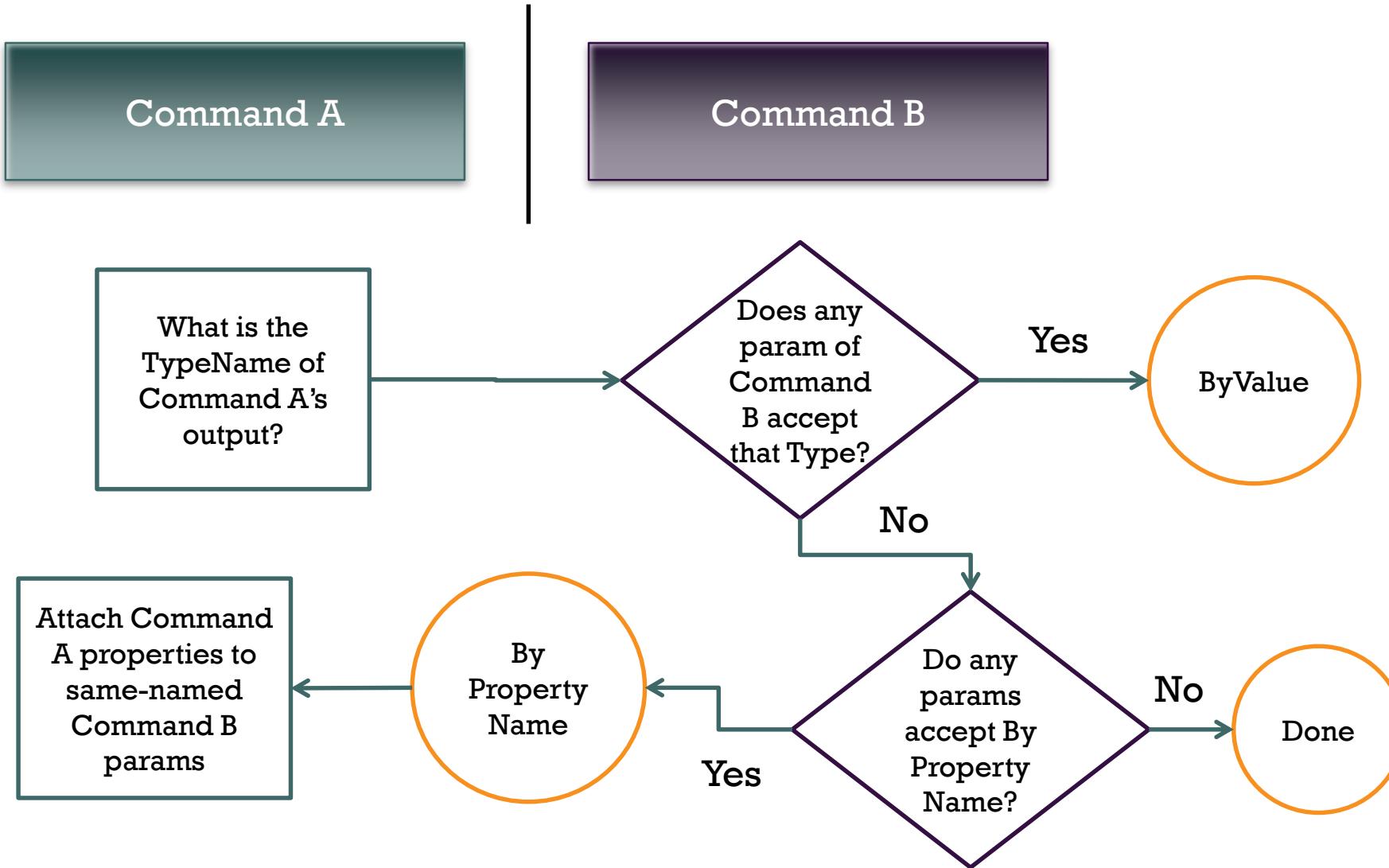


## Module 9

### The Pipeline, Deeper

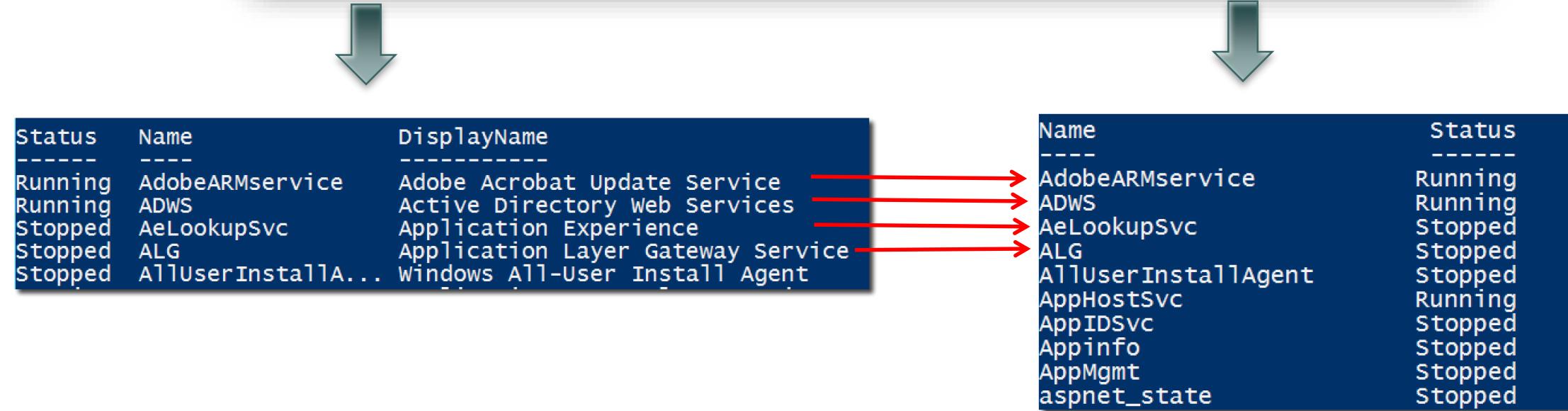
- The Pipeline: Enabling Power With Less Typing
- How PowerShell Passes Data Down the Pipeline
- Plan A: Pipeline Input ByValue
- Plan B: Pipeline Input ByPropertyName
- When Things Don't Line Up: Custom Properties
- Parenthetical Commands
- Extracting the Value from a Single Property
- **Lab**

# Pipeline Parameter Binding



# How the pipeline really works - The 4 step solution

```
PS C:\> Get-Service | select-object name, status
```



# ByValue

1. Get-Service passes ServiceController objects to the pipeline

```
PS C:\> get-service bits | gm
```

```
  TypeName: System.ServiceProcess.ServiceController
```

2. Does Stop-Service accept ServiceController Objects?

```
PS C:\> get-service bits | stop-service
```



3. Help Stop-Service -Full displays a parameter that accepts ServiceController ByValue

```
-InputObject <ServiceController[]>
```

Specifies ServiceController objects representing the services to be stopped. Enter a variable that contains the objects, or type a command or expression that gets the objects.

Required?	false
Position?	named
Default value	
Accept pipeline input?	true (ByValue)
Accept wildcard characters?	false

# ByPropertyName

```
Administrator: Windows PowerShell
PS C:\> Get-Process | Get-Member -MemberType Properties
>>

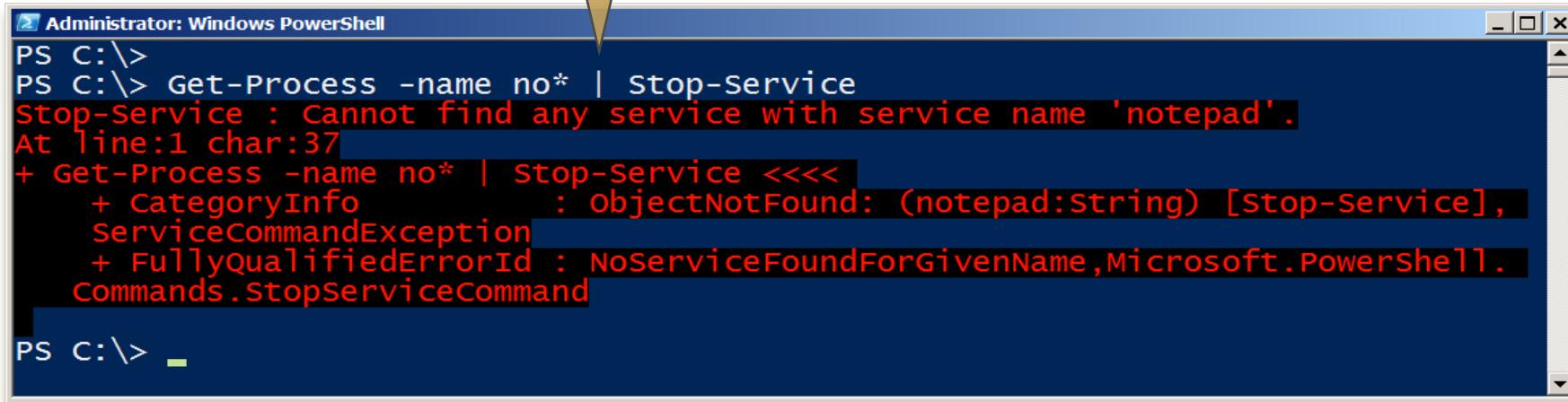
```

Name	MemberType
Handles	AliasProperty
<b>Name</b>	<b>AliasProperty</b>
NPM	AliasProperty
PM	AliasProperty
VM	AliasProperty
WS	AliasProperty
<u>__NounName</u>	NoteProperty
BasePriority	Property
Container	Property
EnableRaisingEvents	Property
ExitCode	Property
ExitTime	Property
Handle	Property
HandleCount	Property
HasExited	Property
<b>Id</b>	<b>Property</b>
<b>MachineName</b>	<b>Property</b>
MainModule	Property

-Name <string[]>	
Required?	false
Position?	1
Default value	
Accept pipeline input?	true (ByPropertyName)
Accept wildcard characters?	true
-Id <Int32[]>	
Required?	true
Position?	named
Default value	
Accept pipeline input?	true (ByPropertyName)
Accept wildcard characters?	false
-ComputerName <string[]>	
Required?	false
Position?	named
Default value	
Accept pipeline input?	true (ByPropertyName)
Accept wildcard characters?	false

# ByPropertyName

1. Get-Process is passing a “Process” Object



```
Administrator: Windows PowerShell
PS C:\>
PS C:\> Get-Process -name no* | Stop-Service
Stop-Service : Cannot find any service with service name 'notepad'.
At line:1 char:37
+ Get-Process -name no* | Stop-Service <<<
+ CategoryInfo          : ObjectNotFound: (notepad:String) [Stop-Service],
+ ServiceCommandException
+ FullyQualifiedErrorId : NoServiceFoundForGivenName,Microsoft.PowerShell.Commands.StopServiceCommand
PS C:\>
```

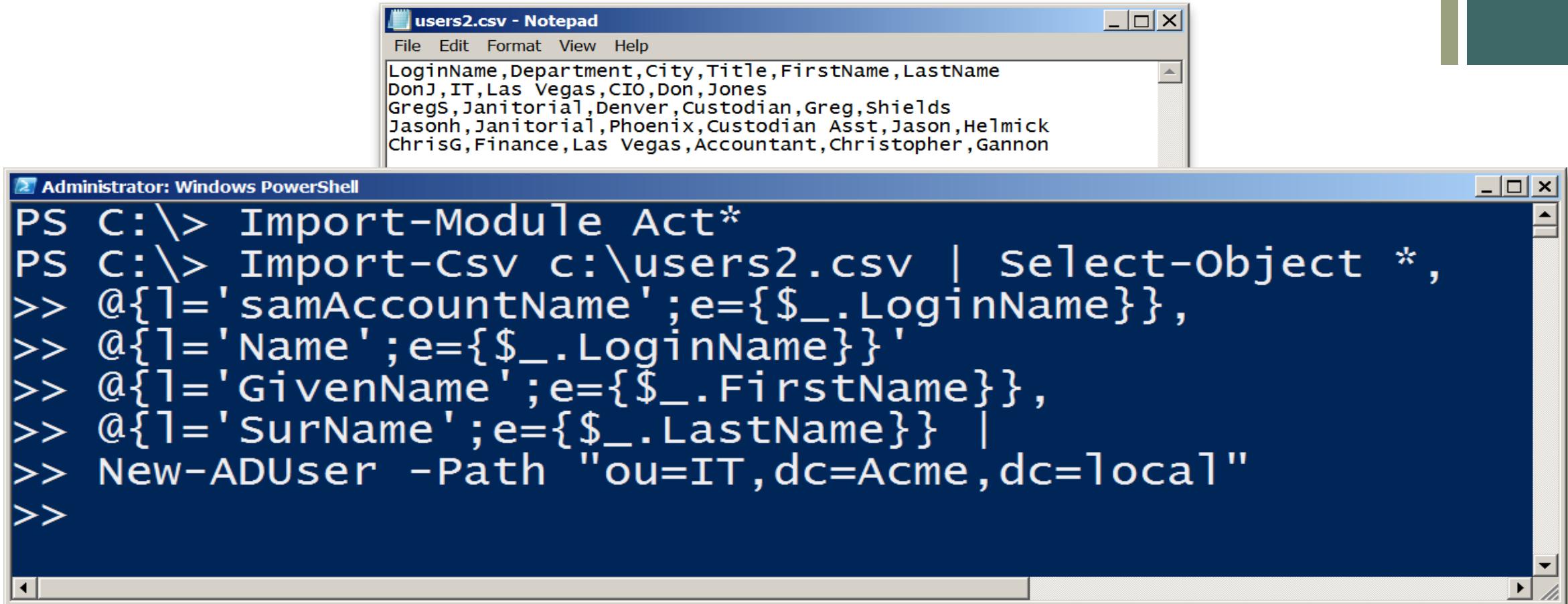
2. Stop-Service does not support accepting “Process” objects `ByValue`, so PowerShell checks what can be accepted `ByPropertyName`.

3. -Name does accept strings `ByPropertyName`, and the objects in the pipeline are labeled as a Name property

4. Stop-Service attempts to use the objects for its `-Name`, in this example, fails

<code>-Name &lt;string[]&gt;</code>	
Required?	true
Position?	1
Default value	
Accept pipeline input?	true ( <code>ByValue, ByPropertyName</code> )
Accept wildcard characters?	false

# What if my property doesn't match – Customize it!



The image shows a Windows desktop environment. At the top, there is a Notepad window titled "users2.csv - Notepad" containing the following CSV data:

	LoginName	Department	City	Title	FirstName	LastName
1	DonJ	IT	Las Vegas	CIO	Don	Jones
2	Gregs	Janitorial	Denver	Custodian	Greg	Shields
3	Jasonh	Janitorial	Phoenix	Custodian Asst	Jason	Helmick
4	ChrisG	Finance	Las Vegas	Accountant	Christopher	Gannon

Below the Notepad window is a PowerShell window titled "Administrator: Windows PowerShell" with the following command history:

```
PS C:\> Import-Module Act*
PS C:\> Import-Csv c:\users2.csv | Select-Object *,
>> @{'l='samAccountName';e={$_.LoginName}},
>> @{'l='Name';e={$_.LoginName}}'
>> @{'l='GivenName';e={$_.FirstName}},
>> @{'l='SurName';e={$_.LastName}} |
>> New-ADUser -Path "ou=IT,dc=Acme,dc=local"
>>
```

# The Parenthetical – when all else fails

1. I want to pass a list of computer names to Get-Service. Why does this fail?

Administrator: Windows PowerShell NP

```
PS C:\> Get-Content c:\names.txt | Get-Service
Get-Service : Cannot find any service with service name ''.
At Line:1 char:39
```

Administrator: Windows PowerShell NP

```
+ Get-Content c:\names.txt | gm
```

Name	MemberType	Definition
Clone	Method	System.Object Clone()

2. -Name and -InputObject accept pipeline input ByValue, not -Computername. -Name accepts text, and then causes the failure.

Administrator: Windows PowerShell NP

```
PS C:\> Get-Service -ComputerName (Get-Content c:\names.txt)
```

Status	Name	DisplayName
Running	ADWS	Active Directory Web Services
Running	ADWS	Active Directory Web Services
Stopped	AeLookupSvc	Application Experience

Parenthesis don't rely on binding and attach information directly to the desired parameter

# The Parenthetical – when all else fails

The image displays three separate Windows PowerShell windows, each showing a different command and its output. Red arrows point from annotations to specific parts of the commands.

- Top Window:** Shows the command `Get-service -ComputerName (Get-ADComputer -Filter *)`. A red arrow points to the parentheses around the pipeline operator (`|`), and a callout bubble says: "Returns a collection (table) of objects."
- Middle Window:** Shows the command `Get-service -ComputerName (Get-ADComputer -Filter * | Select -Property name)`. A red arrow points to the `-Property` parameter, and a callout bubble says: "Returns string contents".
- Bottom Window:** Shows the command `Get-service -ComputerName (Get-ADComputer -Filter * | Select -ExpandProperty Name)`. A red arrow points to the `-ExpandProperty` parameter.

**Output of the bottom window:**

Status	Name	DisplayName
Running	ADWS	Active Directory Web Services
Running	AeLookupSvc	Application Experience
Running	AeLookupSvc	Application Experience
Stopped	AeLookupSvc	Application Experience
Stopped	ALG	Application Layer Gateway Service
Stopped	ALG	Application Layer Gateway Service
Stopped	AppIDSvc	Application Identity
Stopped	AppIDSvc	Application Identity



## Module 10

Formatting –  
and Why it's Done on the Right

- Formatting: Making What You See Prettier
- About the Default Formatting
- Formatting Tables
- Formatting Lists
- Formatting Wide
- Custom Columns and List Entries
- Going Out: To a File, a Printer, or the Host
- Another Out: GridViews
- Common Points of Confusion
  - Always Format Right
  - One Object at a Time, Please
- Lab



## Module 11

### Filtering and Comparisons

- Making the Shell Give You Just What You Need
- Filter Left
- Comparison Operators
- Filtering Objects out of the Pipeline
- The Iterative Command-Line Model
- Common Points of Confusion
  - Filter Left, Please
  - When `$_` is Allowed
- Lab



## Module 12

A Practical Interlude

- Defining the Task
- Finding the Commands
- Learning to Use the Commands
- Tips for Teaching Yourself
- **Lab**



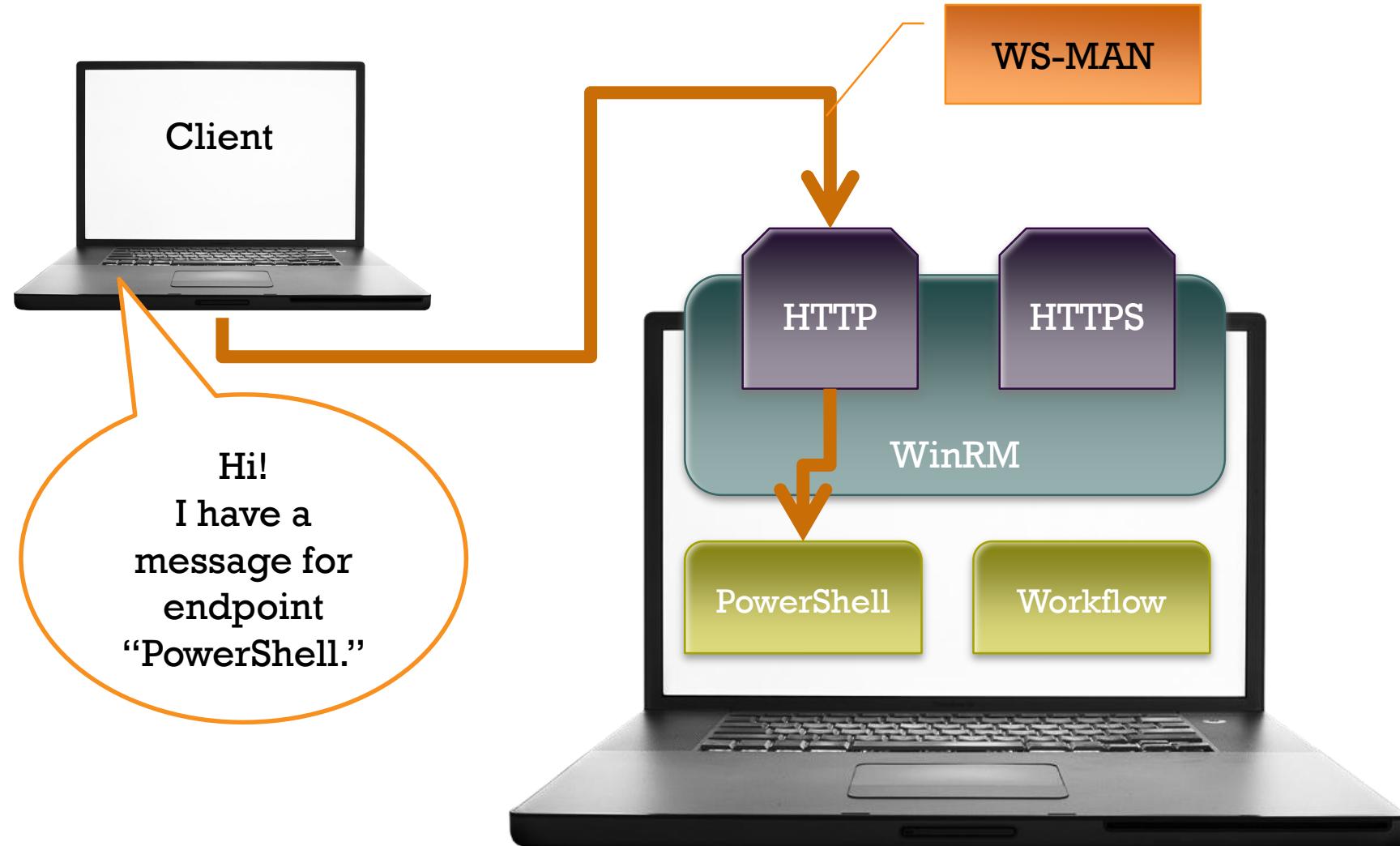
## Module 13

Remote Control:  
One on One, and One to Many

- The Idea Behind Remote PowerShell
- WinRM Overview
- Using Enter-PSSession and Exit-PSSession for One-to-one Remoting
- Using Invoke-Command for One-to-many Remoting
- Differences Between Remote and Local Commands
  - Invoke-Command vs -ComputerName
  - Local vs Remote Processing
  - Deserialized Objects
- But Wait, There's More
- Remoting Options
- Common Points of Confusion
- Lab

&gt;

# Remoting Architecture





## Module 14

### Using Windows Management Instrumentation

- WMI Essentials
- The Bad News About WMI
- Exploring WMI
- Choose Your Weapon: WMI or CIM
- Using Get-WmiObject
- Using Get-Ciminstance
- WMI Documentation
- Common Points of Confusion
- Lab

PS C:\>



```
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

PS C:\Users\mano\Downloads\$env:temp\b1e
Name                           Value
----                           --
PSVersion                      3.0
PSCultureName                  [1,0,2,0,-5]
PSCompatibleVersions            4.0,4.0.20395.17920
PSRemotingProtocolVersion      2.0
PSDefaultParameterSet           None
PSModulePath                   C:\Windows\system32\WindowsPowerShell\v1.0\Modules\

PS C:\Users\mano\Downloads\$env:temp\b1e>
```

# COMPLETE REVIEW LAB 2

Based on Chapter 1-14



## Module 15

### Multitasking with Background Jobs

- Making PowerShell Do Multiple Things at the Same Time
- Synchronous versus Asynchronous
- Creating a Local Job
  - WMI, as a Job
  - Remoting, as a Job
- Getting Job Results
- Working with Child Jobs
- Commands for Managing Jobs
- Scheduled Jobs
- Common Points of Confusion
- Lab

>

DAY 3



## Module 16

Working with Bunches of Objects,  
One at a Time

- Automation for Mass Management
- The Preferred Way: “Batch” Cmdlets
- The WMI Way: Invoking WMI Methods
- The Backup Plan: Enumerating Objects
- Common Points of Confusion
  - Which Way is the Right Way?
  - WMI Methods versus Cmdlets
  - Method Documentation
  - ForEach-Object Confusion
- Lab



## Module 17

Security Alert!

- Keeping the Shell Secure
- Windows PowerShell Security Goals
- Execution Policy and Code Signing
  - Execution Policy Settings
  - Digital Code Signing
- Other Security Measures
- Other Security Holes?
- Security Recommendations
- Lab

# PowerShell security goals

- Secured by default
- Prevents mistakes by unintentional admins and users
- No Script Execution
- .Psl associated with notepad
- Must type path to execute a script





# Execution Policy

- By default, PowerShell does not run scripts.
- Get/Set-ExecutionPolicy
  - Restricted
  - Unrestricted
- AllSigned
- RemoteSigned
- Bypass
- Undefined
- Can be set with Group Policy





## Module 18

Variables: A Place to Store Your Stuff

- Introduction to Variables
- Storing Values in Variables
- Fun Tricks with Quotes
- Storing Lots of Objects in a Variable
- More Tricks with Double Quotes
- Declaring a Variable's Type
- Commands for Working with Variables
- Variable Best Practices
- Common Points of Confusion
- **Lab**

# > Variables: A place to store stuff

- Use \$ to create and use variables
- Can contain letters, numbers, spaces and underscores
- Don't persist after Shell exits
- New-Variable
- Set-Variable
- Get-Variable
- Clear-Variable
- Remove-Variable
- Can force a type – [int]\$var

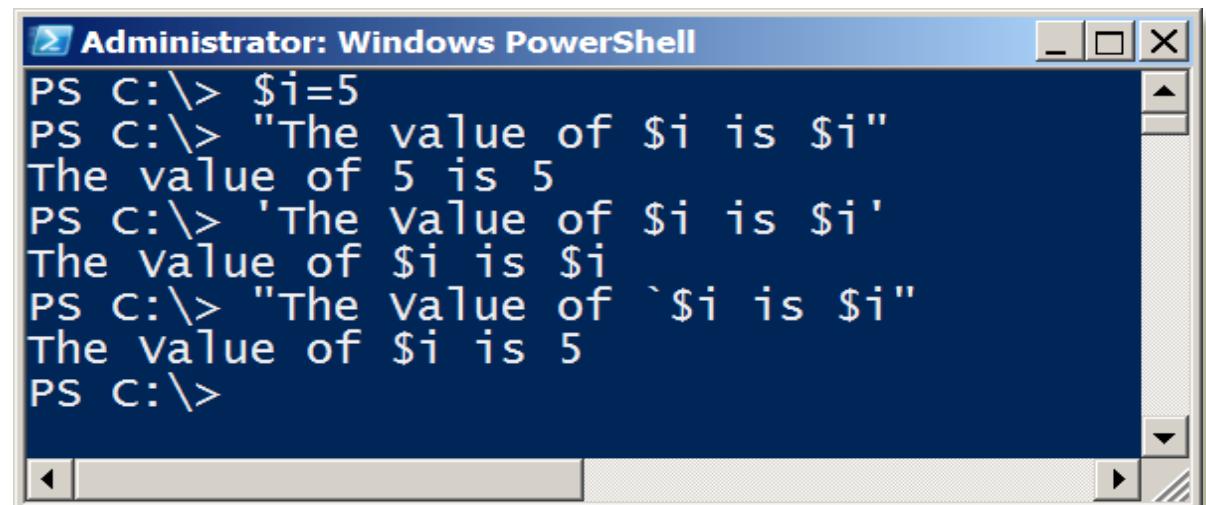
\$myvariable  
\${My Variable}

**Note: The \$ is not part of the variable name, it's a cue to access the contents of the variable**

Type	Description
[int]	32-bit signed integer
[long]	64-bit signed integer
[string]	Fixed-length string of Unicode characters
[char]	A Unicode 16-bit character
[byte]	An 8-bit unsigned character
[bool]	Boolean True/False value
[decimal]	An 128-bit decimal value
[single]	Single-precision 32-bit floating point number
[double]	Double-precision 64-bit floating point number
[datetime]	Date and Time
[array]	An array of values
[hashtable]	Hashtable object

## > Fun with Quotes

- Double Quotes resolve all variables
- Can use Sub-Expressions
- Single Quotes prevent substitution
- Get-Help About\_Quoting\_Rules
- Back-tick/Grave-Accent prevents individual substitution



The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". It displays the following command and output sequence:

```
PS C:\> $i=5
PS C:\> "The value of $i is $i"
The value of 5 is 5
PS C:\> 'The value of $i is $i'
The value of $i is $i
PS C:\> "The value of `\$i is \$i"
The value of $i is 5
PS C:\>
```





# Module 19

## Input and Output

- Prompting For, and Displaying, Information
  - Read-Host
  - Write-Host
  - Write-Output
  - Other Ways to Write
  - Lab

# > Getting and displaying input

The image displays three overlapping Windows PowerShell windows, each showing different command examples:

- Top Window:** Shows the use of `Read-Host` to prompt for a computer name.

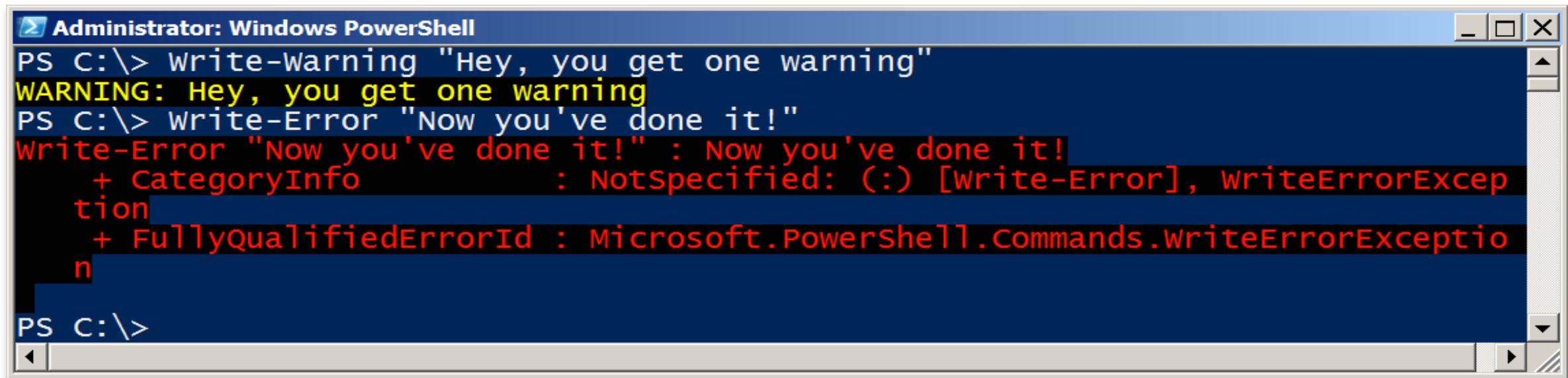
```
PS C:\> Read-Host "Enter a computer name"
Enter a computer name: Server1
Server1
PS C:\>
```
- Middle Window:** Shows color output and a parsing error.

```
PS C:\> Write-Host "PowerShell Rocks!" -ForegroundColor Yellow -BackgroundColor Red
PowerShell Rocks!
PS C:\> "Hello?" -foregroundcolor yellow
You must provide a value expression on the right-hand side of the '-f' operator
.
At Line:1 char:12
+ "Hello?" -f <<< orgroundcolor yellow
+ CategoryInfo          : ParserError: (:) [], ParentContainsErrorRecordException
+ FullyQualifiedErrorId : ExpectedValueExpression
```
- Bottom Window:** Shows filtering output based on string length.

```
PS C:\> write-output "Hello" | where-object {$_ .length -gt 10}
PS C:\>
PS C:\> write-Host "Hello" | where-object {$_ .length -gt 10}
Hello
PS C:\>
```

# > Other output for scripts and automation

- Write-Warning
  - \$Preference variables to know
  - Help about\_Preference\_Variables
- Write-Verbose
  - \$DebugPreference=SilentlyContinue
  - \$ErrorActionPreference=Continue
  - #VerbosePreference=SilentlyContinue
- Write-Debug
- Write-Error



The screenshot shows a Windows PowerShell window titled "Administrator: Windows PowerShell". The command PS C:\> write-warning "Hey, you get one warning" is run, followed by the output "WARNING: Hey, you get one warning". The command PS C:\> write-error "Now you've done it!" is run, followed by the output "write-Error "Now you've done it!" : Now you've done it! + CategoryInfo : NotSpecified: (:) [write-Error], writeErrorException + FullyQualifiedErrorId : Microsoft.PowerShell.Commands.writeErrorException". The PowerShell window has a blue header bar and a dark blue body.

```
Administrator: Windows PowerShell
PS C:\> write-warning "Hey, you get one warning"
WARNING: Hey, you get one warning
PS C:\> write-error "Now you've done it!"
write-Error "Now you've done it!" : Now you've done it!
+ CategoryInfo          : NotSpecified: (:) [write-Error], writeErrorException
+ FullyQualifiedErrorId : Microsoft.PowerShell.Commands.writeErrorException
PS C:\>
```

PS C:\>



```
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

PS C:\Users\mano\Downloads\$env:temp\b1e

Name                           Value
----                           ---
PSVersion                      3.0
PSCultureName                  [1252, 2000, 9]
PSCompatibleVersions           {1.0, 2.0, 3.0}
PSRemotingProtocolVersion      2.2.10240.6
PSDefaultParameterValues        {}
PSReadLineProtocolVersion      1.0
PSRemotingTransportVersion     2.0.0
PSHostUserAgent                1.3.0.1

PS C:\Users\mano\Downloads\$env:temp\b1e >
```

# COMPLETE REVIEW LAB 3

Based on Chapter 1-19

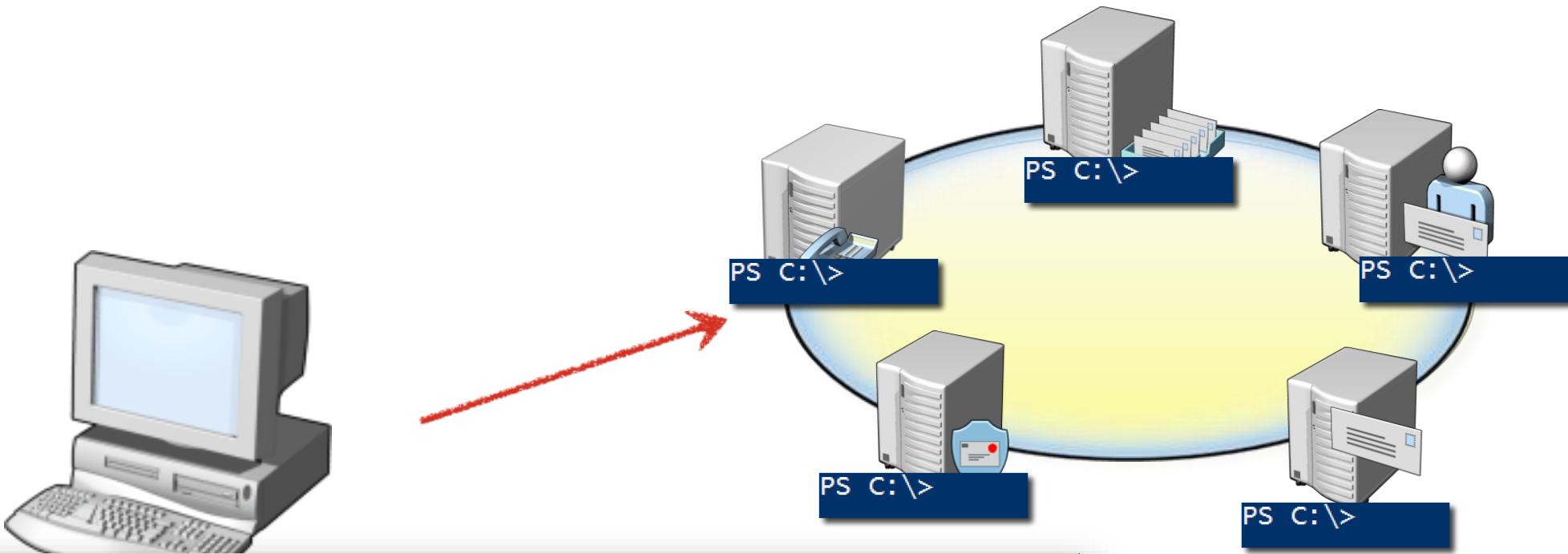


# Module 20

Sessions:  
Remote Control, with Less Work

- Making PowerShell Remoting a Bit Easier
- Creating and Using Reusable Sessions
- Using Sessions with Enter-PSSession
- Using Sessions with Invoke-Command
- Implicit Remoting: Importing a Session
- Disconnected Sessions
- **Lab**

# Overview of Remoting



```
Administrator: Windows PowerShell
PS C:\> Enter-PSSession server1
[server1]: PS C:\Users\Administrator.SERVER1\Documents> Get-Service

Status   Name           DisplayName
----   --   -----
Running  ADWS          Active Directory Web Services
Stopped AeLookupSvc    Application Experience
Stopped ALG           Application Layer Gateway Service
Stopped AppIDSvc      Application Identity
Stopped Appinfo        Application Information
Stopped AppMgmt        Application Management
Stopped aspnet_state   ASP.NET State Service
```

# Enable Remoting

PS WSMAN:\LocalHost>

```
Administrator: Windows PowerShell
PS C:\> Enable-PSRemoting
WinRM Quick configuration
Running command "Set-WSManQuickConfig"
management through WinRM service.
This includes:
  1. Starting or restarting (if already running) the WinRM service
  2. Setting the WinRM service type
  3. Creating a listener to accept requests
  4. Enabling firewall exception for WinRM
Do you want to continue?
[Y] Yes [A] Yes to All [N] No [L] No [default is "Y"] :y
```

The screenshot shows the Group Policy Management Editor window. The left pane displays the navigation tree under 'Computer Configuration/Policies/Administrative Templates/Windows Components/Windows Remote Management'. The 'WinRM Service' node is selected and highlighted with a blue border. A red arrow points from the PowerShell command line in the left panel to this selected node. The right pane shows a table of policy settings:

Setting	State
Allow automatic configuration of listeners	Not config...
Allow Basic authentication	Not config...
Allow CredSSP authentication	Not config...
Allow unencrypted traffic	Not config...
Specify channel binding token hardening I...	Not config...
Disallow Kerberos authentication	Not config...
Disallow Negotiate authentication	Not config...
Turn On Compatibility HTTP Listener	Not config...
Turn On Compatibility HTTPS Listener	Not config...

PowerShell Remoting  
is already enabled in  
Server 2012

Computer Configuration/Policies/Administrative  
Templates/Windows Components/Windows Remote Management

# One to One - Interactive

```
Administrator: Windows PowerShell
PS C:\> Enter-PSSession Server2.acme.local
[server2.acme.local]: PS C:\Users\administrator.ACME\Documents> cd\
[server2.acme.local]: PS C:\> Get-Service

Status      Name                DisplayName
----      ----
Running     AeLookupSvc        Application Experience
Stopped    AppMgmt            Application Management
Running     BFE                Base Filtering Engine
Running     BITS               Background Intelligent Transfer Ser...
Stopped    Browser             Computer Browser
Running     CertPropSvc       Certificate Propagation
Stopped    clr_optimizatio... Microsoft .NET Framework NGEN v2.0....
Stopped    COMSysApp          COM+ System Application
Running     CryptSvc           Cryptographic Services
Running     DcomLaunch         DCOM Server Process Launcher
Stopped    defragsvc          Disk Defragmenter
Running     Dhcp               DHCP Client
Running     Dnscache           DNS Client
Running     DPS                Diagnostic Policy Service
Stopped    EFS                Encrypting File System (EFS)
Running     eventlog           Windows Event Log
```

# One-To-Many

```
Administrator: Windows PowerShell
PS C:\>
PS C:\> Invoke-Command -ComputerName Server1, Server2, Win7 -ScriptBlock {
>> Get-EventLog -LogName Security -Newest 2 } |
>> Format-Table PsComputerName, EntryType, Source
>>



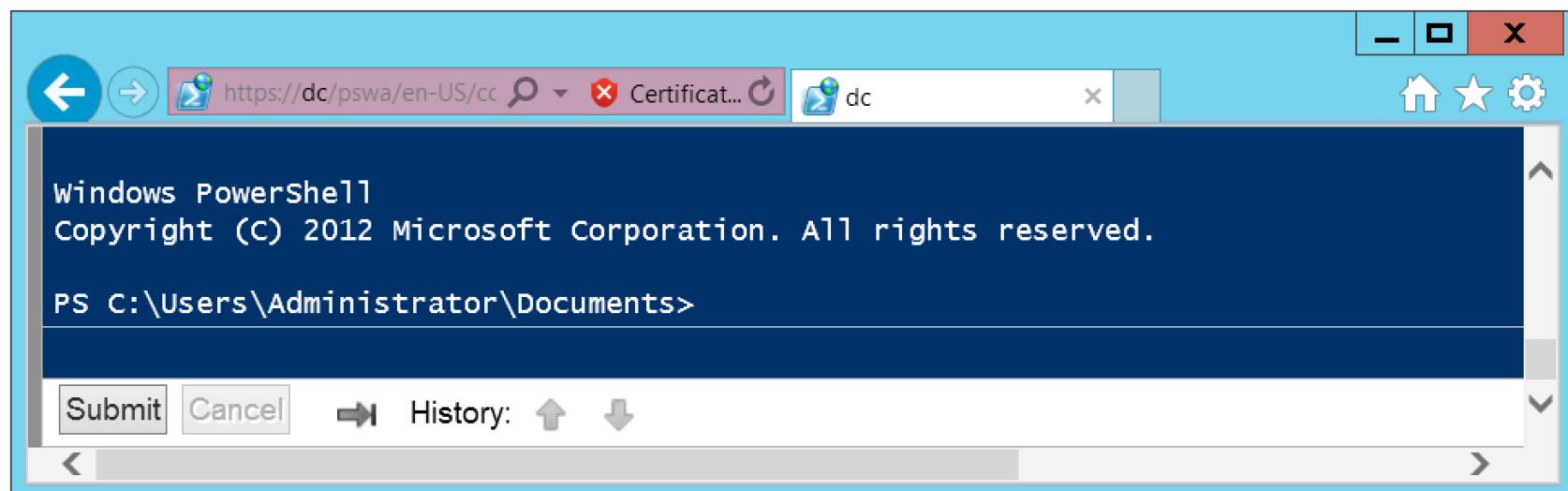
| PsComputerName | EntryType    | Source                    |
|----------------|--------------|---------------------------|
| server1        | SuccessAudit | Microsoft-windows-Secu... |
| server1        | SuccessAudit | Microsoft-windows-Secu... |
| server2        | SuccessAudit | Microsoft-windows-Secu... |
| server2        | SuccessAudit | Microsoft-windows-Secu... |
| win7           | SuccessAudit | Microsoft-windows-Secu... |
| win7           | SuccessAudit | Microsoft-windows-Secu... |


```

```
Administrator: Windows PowerShell
PS C:\>
PS C:\> Invoke-Command -UseSSL -Port 443 -ThrottleLimit 64 -ScriptBlock{
>> Get-Service} -ComputerName Server1, Server2 -credential (Get-credential)
>> -
```

# PowerShell Web Access

- PowerShell – Anywhere, anytime, on any device!
- Install-WindowsFeature –Name WindowsPowerShellWebAccess
- Get-Help \*Pswa\*
- Install-PswaWebApplication –UseTestCertificate
- # Use the –useTestCertificate for testing (Expires in 90 days)
- Add-PswaAuthorizationRule –userName <Domain\User | Computer\user> -ComputerName <Computer> -ConfigurationName AdminsOnly



# Not the end yet!

- More to come!
- Managing in scale and in real time!
- Automation and scripting!
- Great resource: Free!
- Secrets of PowerShell Remoting – Don Jones and Tobias Weltner
- <http://powershell.org/wp/powershell-books/>



## Module 21

You Call This Scripting?

- Not Programming... More Like Batch Files
- Making Commands Repeatable
- Parameterizing Commands
- Creating a Parameterized Script
- Documenting Your Script
- One Script, One Pipeline
- A Quick Look at Scope
- **Lab**

```
param(  
    [string]$computerName,  
    [string]$serviceName  
)  
Get-WmiObject -Class Win32_Service  
    -Filter "name='\$serviceName'"  
    -ComputerName $computerName
```

Parameter

Comma separates the two parameters

Backtick!

&gt;

# The new ISE

Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

PS C:\> Get-WmiObject -Class win32\_LogicalDisk -Filter "DeviceID='C:'" -ComputerName localhost

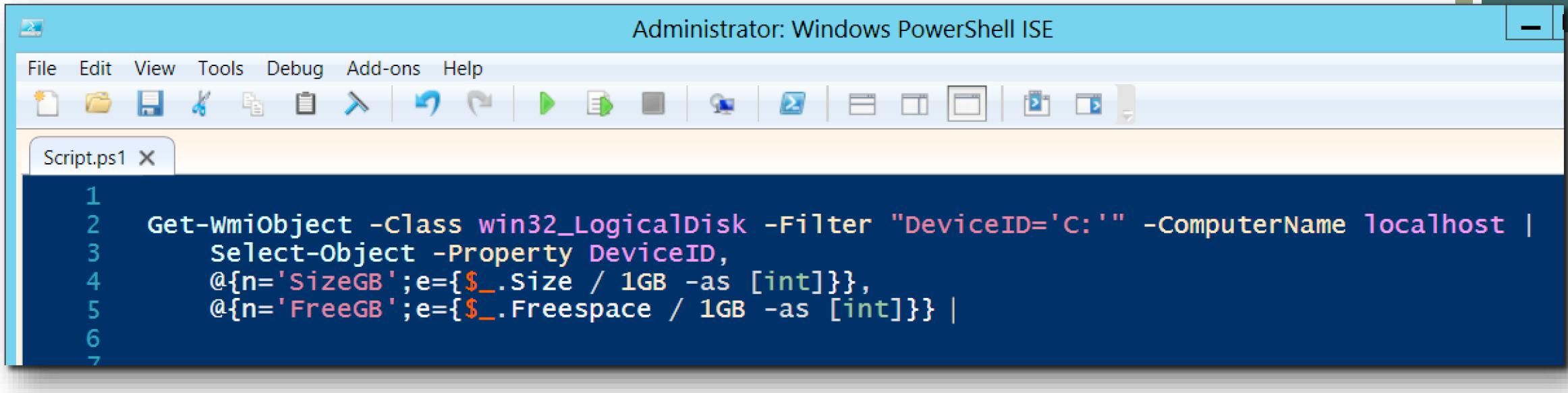
Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Script.ps1 X

```
1
2 Get-WmiObject -Class win32_LogicalDisk -Filter "DeviceID='C:'" -ComputerName localhost |
3     Select-Object -Property DeviceID,
4         @{n='SizeGB';e={$_.Size / 1GB -as [int]}},
5         @{n='FreeGB';e={$_.Freespace / 1GB -as [int]}} |
```

# Making commands repeatable



The screenshot shows the Windows PowerShell Integrated Scripting Environment (ISE) window titled "Administrator: Windows PowerShell ISE". The menu bar includes File, Edit, View, Tools, Debug, Add-ons, and Help. The toolbar contains various icons for file operations like Open, Save, and Run. A script named "Script.ps1" is open in the editor, displaying the following PowerShell command:

```
1
2 Get-WmiObject -Class win32_LogicalDisk -Filter "DeviceID='C:'" -ComputerName localhost |
3     Select-Object -Property DeviceID,
4     @{n='SizeGB';e={$_.Size / 1GB -as [int]}},
5     @{n='FreeGB';e={$_.Freespace / 1GB -as [int]}} |
```

>

# Adding parameters to your script

Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Script.ps1 Script1a.ps1 X Script2.ps1

```
1 Param(  
2     $ComputerName='localhost'  
3 )  
4  
5 Get-WmiObject -Class win32_LogicalDisk -Filter "DeviceID='C:'" -ComputerName $ComputerName |  
6     Select-Object -Property DeviceID,  
7     @{n='SizeGB';e={$_.Size / 1GB -as [int]}},  
8     @{n='FreeGB';e={$_.Freespace / 1GB -as [int]}}
```

PS C:\> .\Script1a.ps1 -| ComputerName [Object] ComputerName

## > Documenting your script

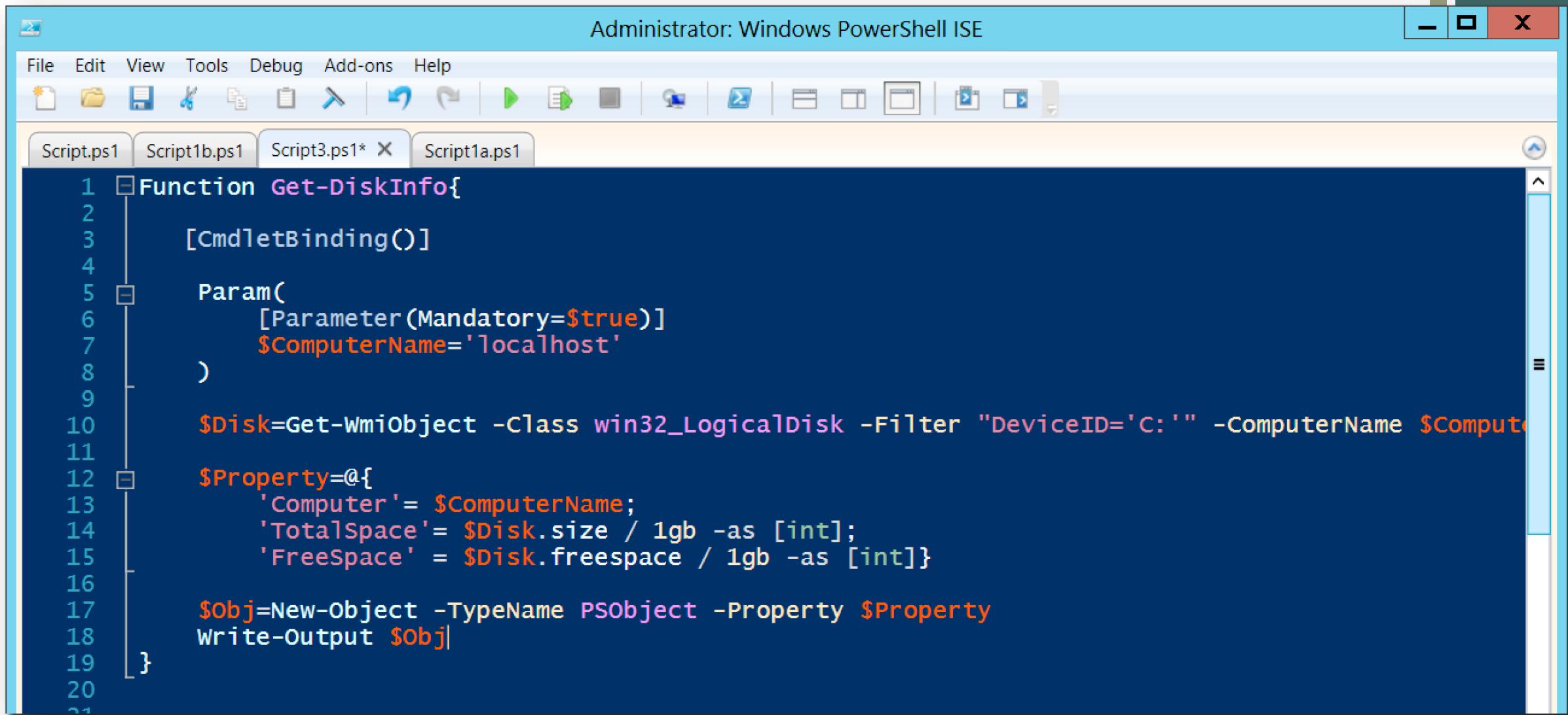
Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Script.ps1 Script1b.ps1 X Script2.ps1 Script1a.ps1

```
1 <#
2 .Synopsis
3 This gets the current disk size and freespace.
4 .Description
5 This is the long description
6 .Parameter ComputerName
7 This is the computer (local or remote) you want the diskspace from.
8 .Example
9 PS C:\> .\Script.ps1 -ComputerName <Computer>
10 Getting the diskspace of a remote computer
11 #>
12 Param(
13     $ComputerName='localhost'
14 )
15
16 Get-WmiObject -Class win32_LogicalDisk -Filter "DeviceID='C:'" -ComputerName
17     Select-Object -Property DeviceID,
18     @{n='SizeGB';e={$_.Size / 1GB -as [int]}},
19     @{n='FreeGB';e={$_.Freespace / 1GB -as [int]}}
```

# Turning your script into a tool for others



The screenshot shows the Windows PowerShell Integrated Scripting Environment (ISE) window titled "Administrator: Windows PowerShell ISE". The menu bar includes File, Edit, View, Tools, Debug, Add-ons, and Help. The toolbar contains various icons for file operations like Open, Save, and Run. The tabs at the top show "Script.ps1", "Script1b.ps1", "Script3.ps1\*", and "Script1a.ps1". The main code editor displays the following PowerShell script:

```
1 Function Get-DiskInfo{
2
3     [CmdletBinding()]
4
5     Param(
6         [Parameter(Mandatory=$true)]
7         $ComputerName='localhost'
8     )
9
10    $Disk=Get-WmiObject -Class win32_LogicalDisk -Filter "DeviceID='C:'" -ComputerName $ComputerName
11
12    $Property=@{
13        'Computer' = $ComputerName;
14        'TotalSpace' = $Disk.size / 1gb -as [int];
15        'FreeSpace' = $Disk.freespace / 1gb -as [int]}
16
17    $Obj=New-Object -TypeName PSObject -Property $Property
18    Write-Output $Obj
19
20 }
```



# Storing your tools in a module

Administrator: Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Script.ps1 Script1b.ps1 Script4.psm1\* Script1a.ps1

```
1 Function Set-DiskInfo {}
2 Function Remove-DiskInfo {}
3 Function Get-DiskInfo{
4     [CmdletBinding()]
5     Param(
6         [Parameter(Mandatory=$true)]
7         $ComputerName='localhost'
8     )
9     $Disk=Get-WmiObject -Class win32_LogicalDisk -Filter "DeviceID='C:'" -ComputerNa
10    $Property=@{
11        'Computer'= $ComputerName;
12        'TotalSpace'= $Disk.size / 1gb -as [int];
13        'FreeSpace' = $Disk.freespace / 1gb -as [int]}
14        $Obj=New-Object -TypeName PSObject -Property $Property
15        Write-Output $Obj
16    }
```



## Module 22

### Improving Your Parameterized Script

- Starting Point
- Getting PowerShell to do the Hard Work
- Making Parameters Mandatory
- Adding Parameter Aliases
- Validating Parameter Input
- Adding the Warm and Fuzzies with Verbose Output
- **Lab**

```
[CmdletBinding()]
param(
    [Parameter(Mandatory=$True)]
    [string]$computerName,
    [Parameter(Mandatory=$True)]
    [string]$serviceName
)
Get-WmiObject -Class Win32_Service
    -Filter "name='\$serviceName'"
    -ComputerName $computerName
```



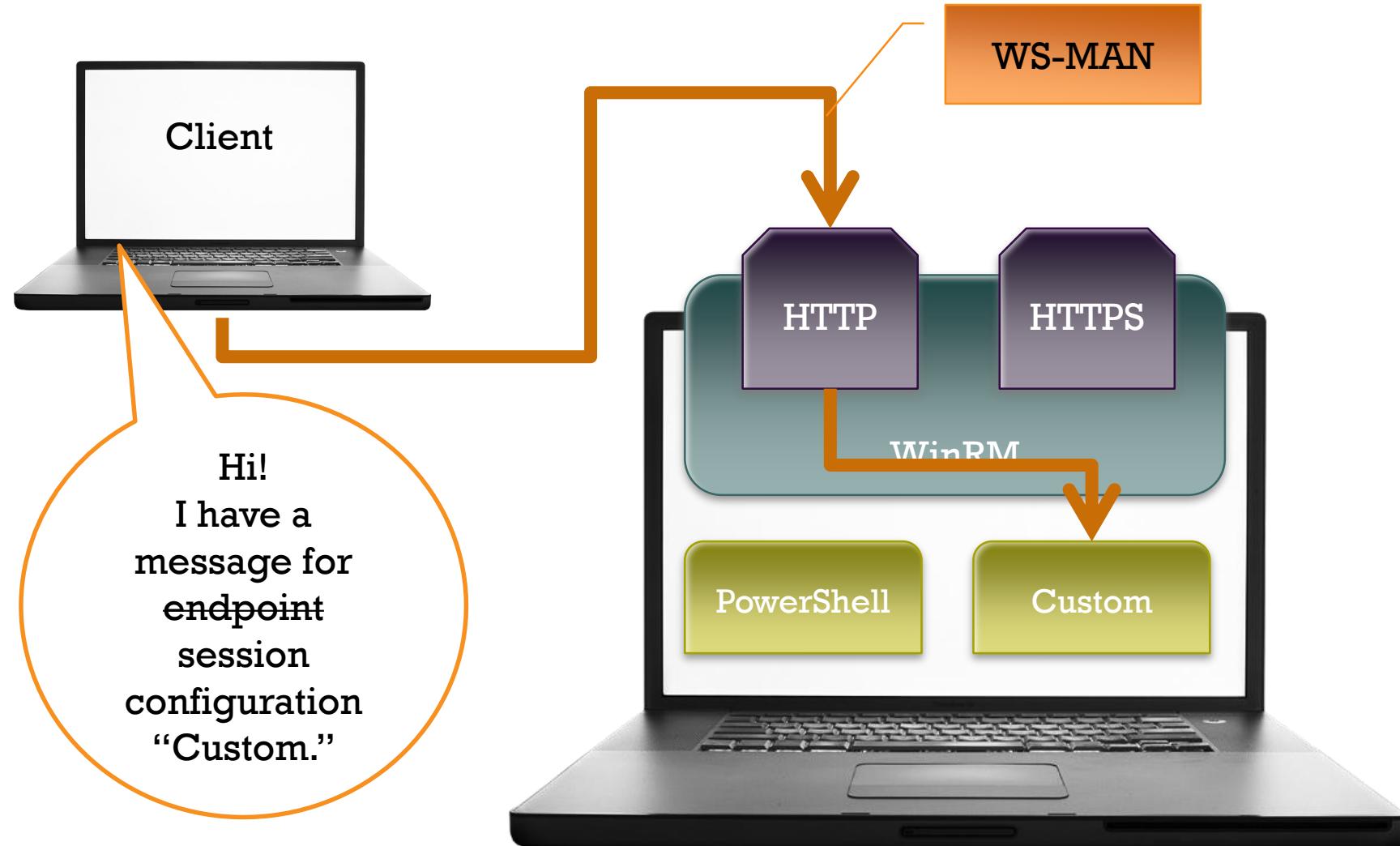
## Module 23

### Advanced Remoting Configuration

- Using Other Endpoints
- Creating Custom Endpoints
  - Creating the Session Configuration
  - Registering the Session
- Enabling Multi-Hop Remoting
- Digging Deep into Remoting Authentication
  - Defaults for Mutual Authentication
  - Mutual Authentication via SSL
  - Mutual Authentication via TrustedHosts
- Lab

&gt;

# Remoting Architecture





## Module 24

Using Regular Expressions to Parse Text Files

- The Purpose of Regular Expressions
- A RegEx Syntax Primer
- Using RegEx with -Match
- Using RegEx with Select-String
- **Lab**



## Module 25

Additional Tips, Tricks, and Techniques

- Profiles, Prompts and Colors: Customizing the Shell
  - PowerShell Profiles
  - Customizing the Prompt
  - Tweaking Colors
- More Operators: -as, -is, -replace, -join, -split
  - -as and -is
  - -replace
  - -join and -split
  - -contains and -in
- String Manipulation
- Date Manipulation
- Dealing with WMI Dates
- Setting Default Parameter Values
- Playing with Script Blocks