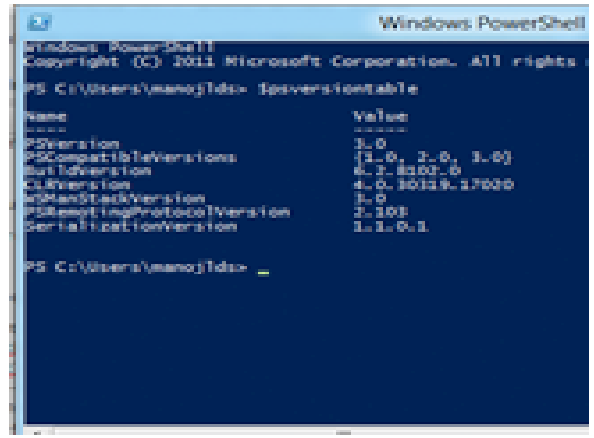
PS C:\>



Windows PowerShell Foundations

for Microsoft IT Professionals

Based on Learn Windows PowerShell in a Month of Lunches Book

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

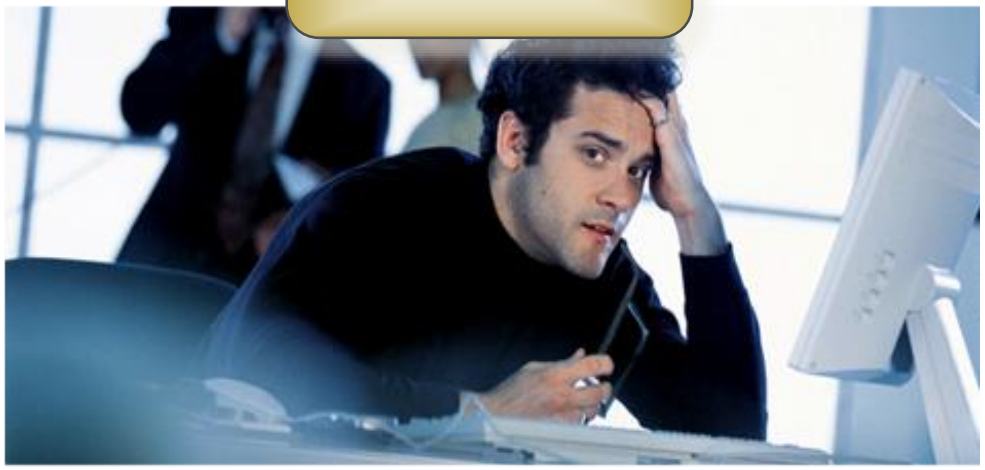
- Target Audience
  - Tailored for the IT pro that needs to improve management and automation
  - Fast paced for the real world
- Suggested Prerequisites/Supporting Material
  - Experience working as a Windows IT pro/Admin/Help Desk
  - Get answers in the forums at PowerShell.Org
  - Learn Windows PowerShell 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



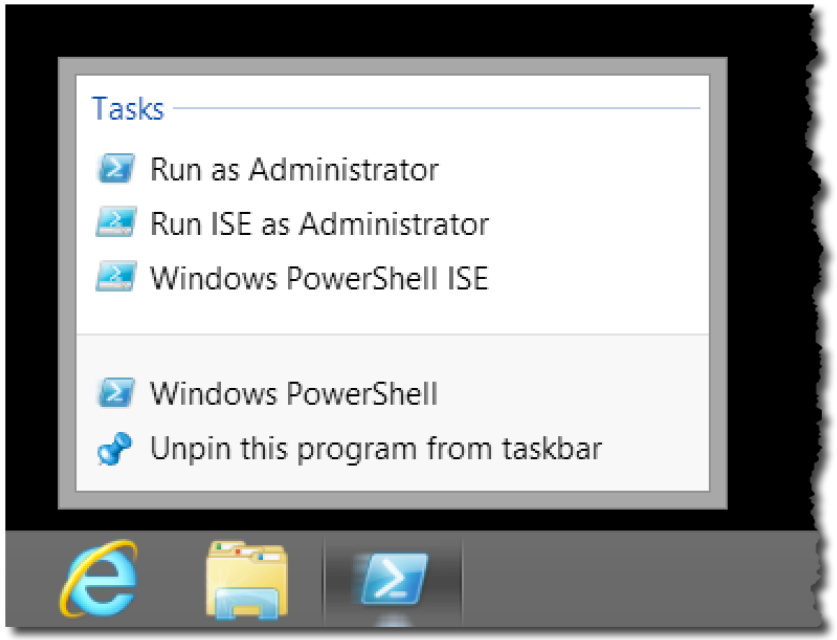
THEM

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

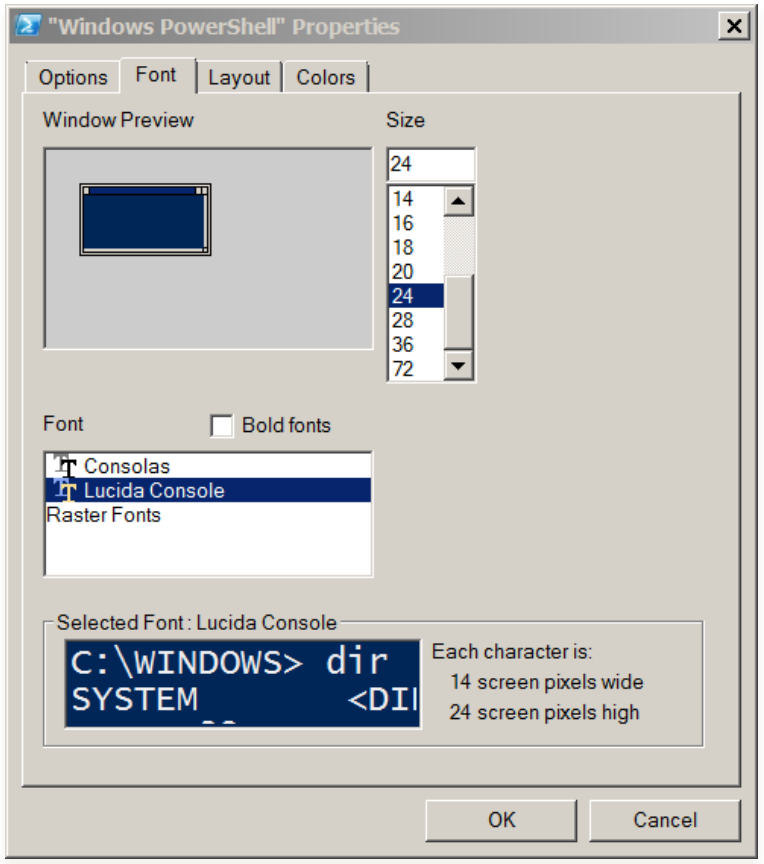
Installing PowerShell – Windows Management Framework

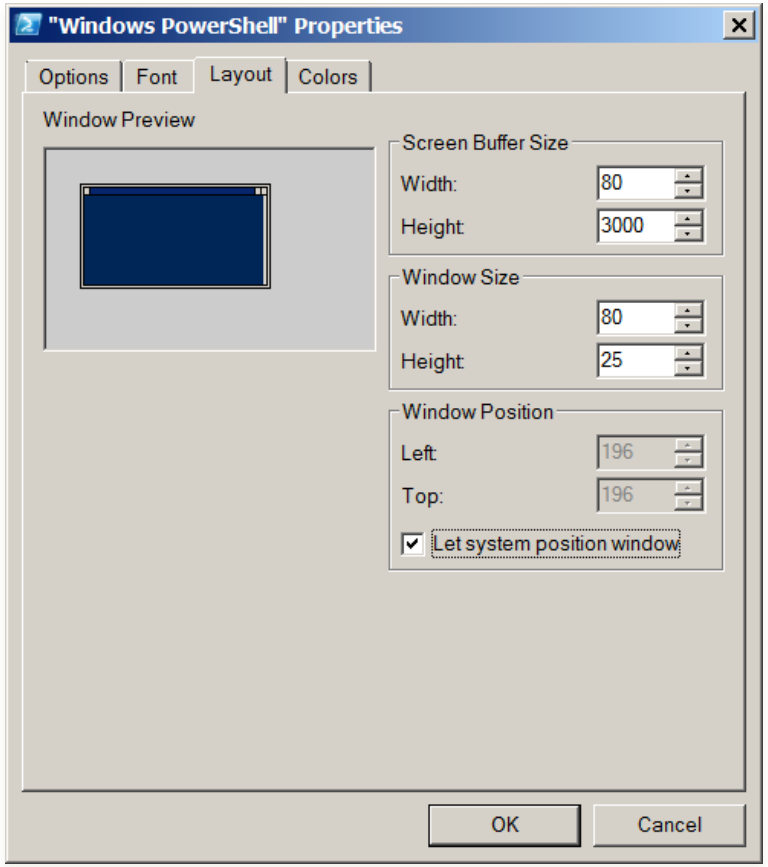
- PowerShell V5 – Windows 10 and Server 2012
- PowerShell V3 – Windows 8 and Server 2012
- PowerShell V2 – Windows 7 and Server 2008
- Download the Windows Management Framework 3.0 at
- http://www.microsoft.com/en-us/download/details.aspx?id=34595
- Windows XP and Server 2003 can run V2

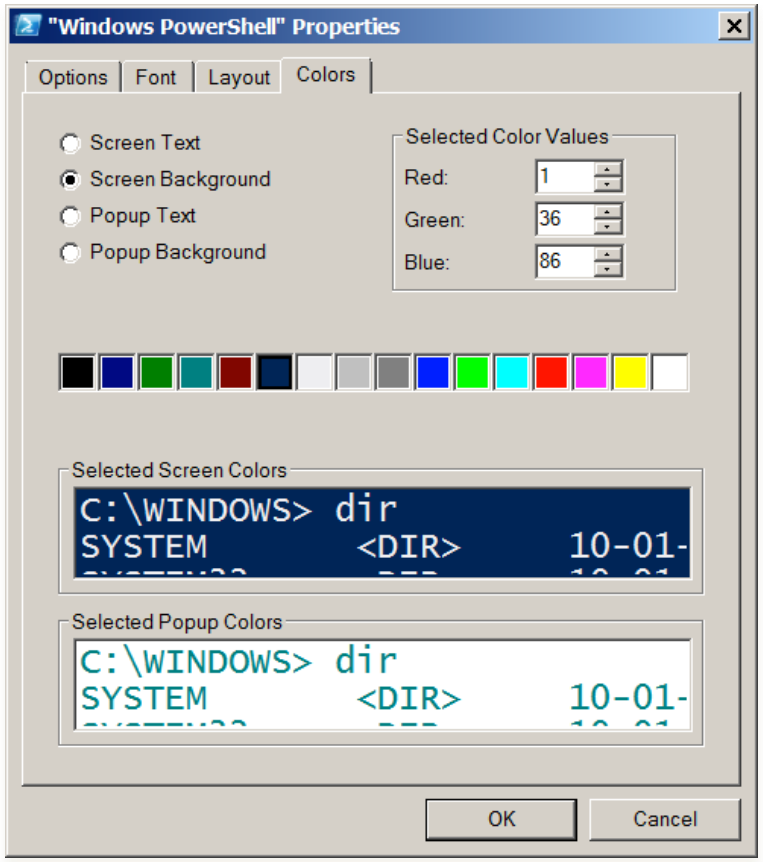
Launching PowerShell for the Administrator



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

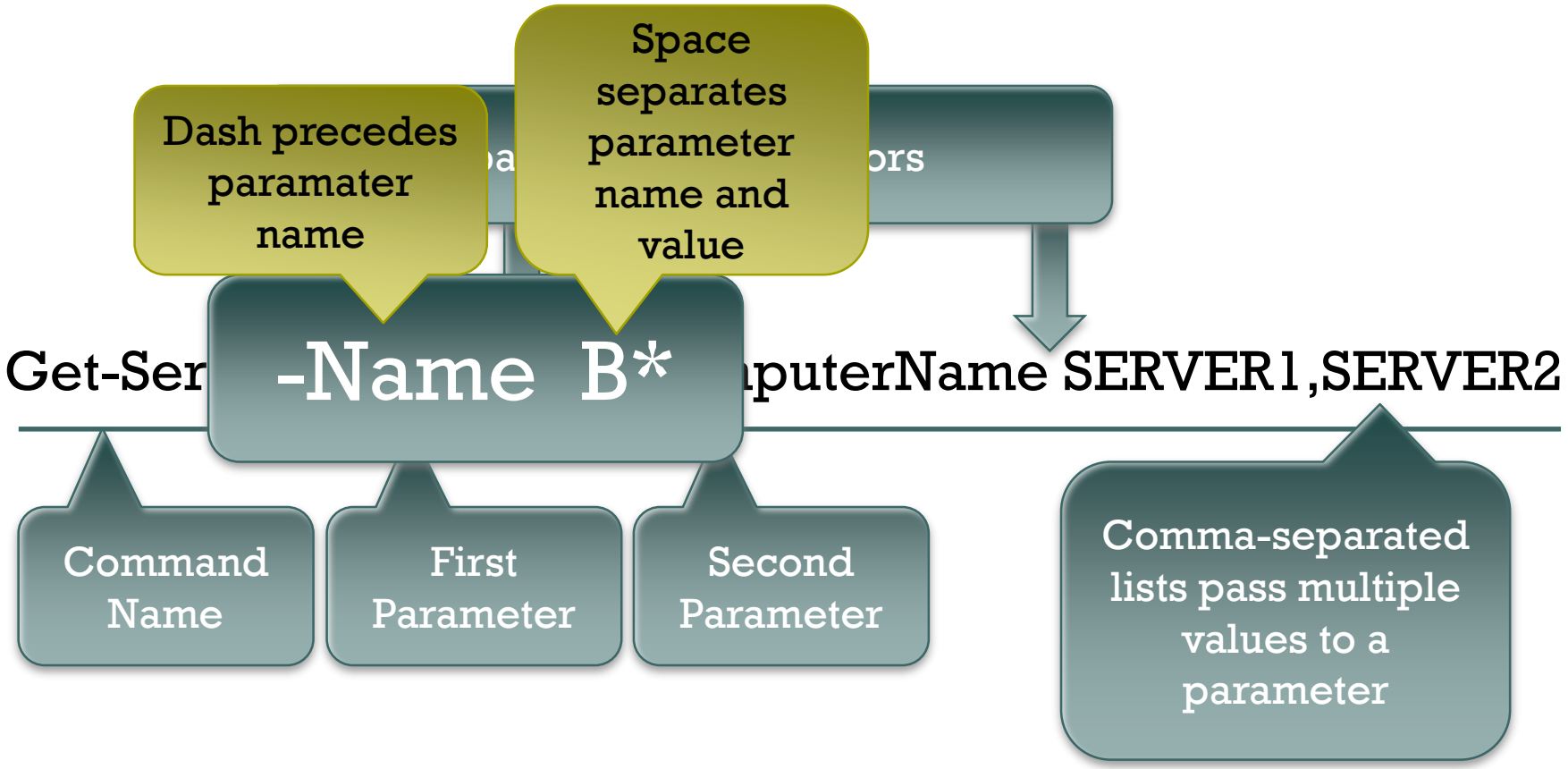
- Help
-
-

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**

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

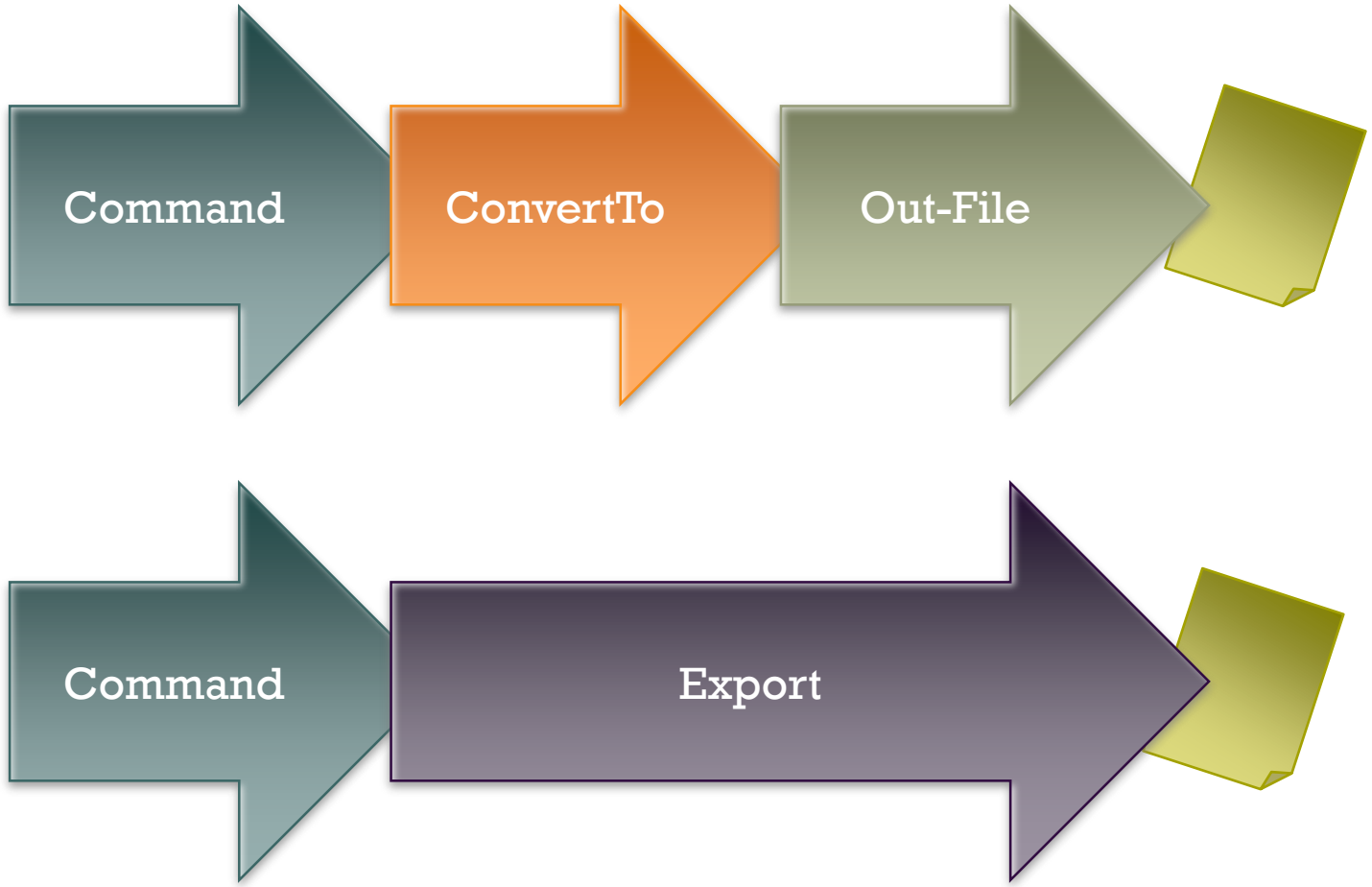
- What are Providers?
- How the File System is Organized
- How the File System is Like Other Data Stores
- Navigating the File System
- Using Wildcards and Literal Paths
- Working with Other Providers
- **Lab**

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 |

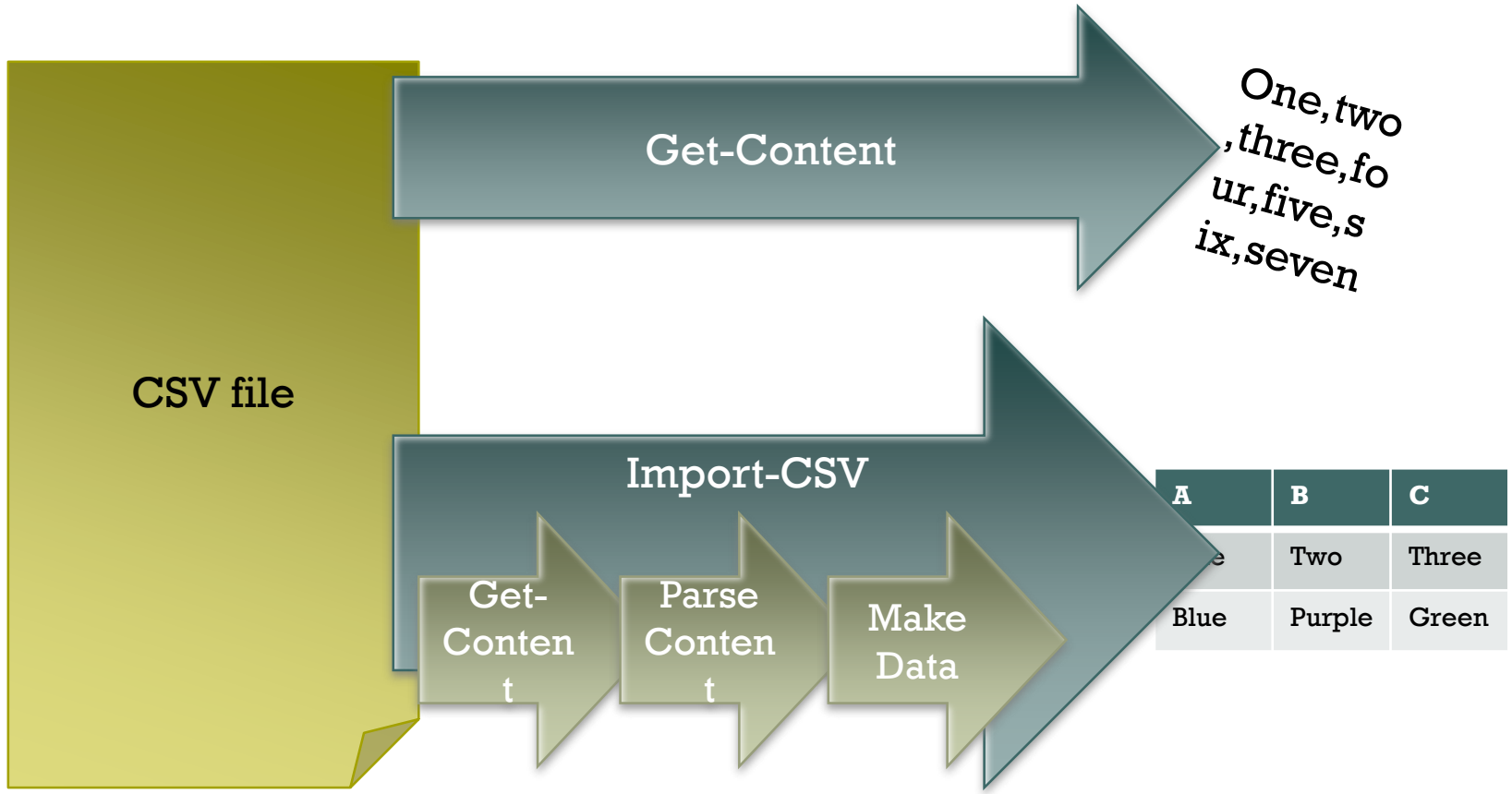
Connects cmdlets to produce better results

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

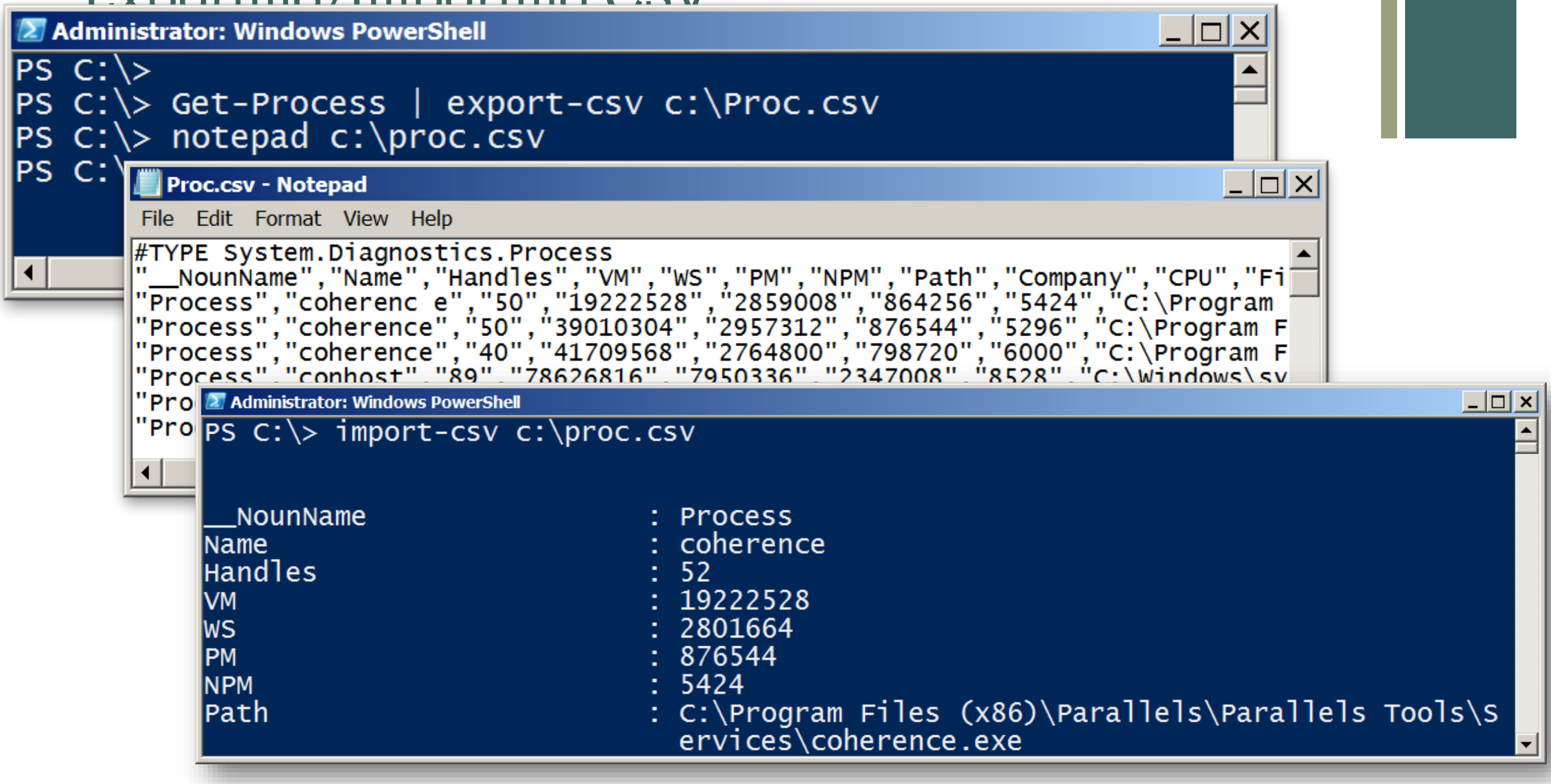
Can be broken into several lines to increase readability

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

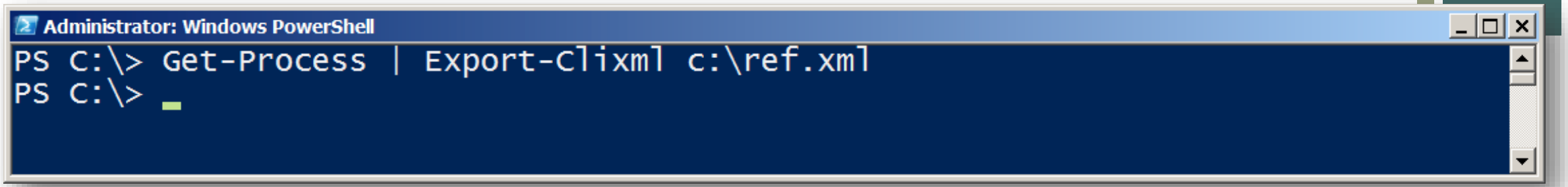
Get-Content vs. Import

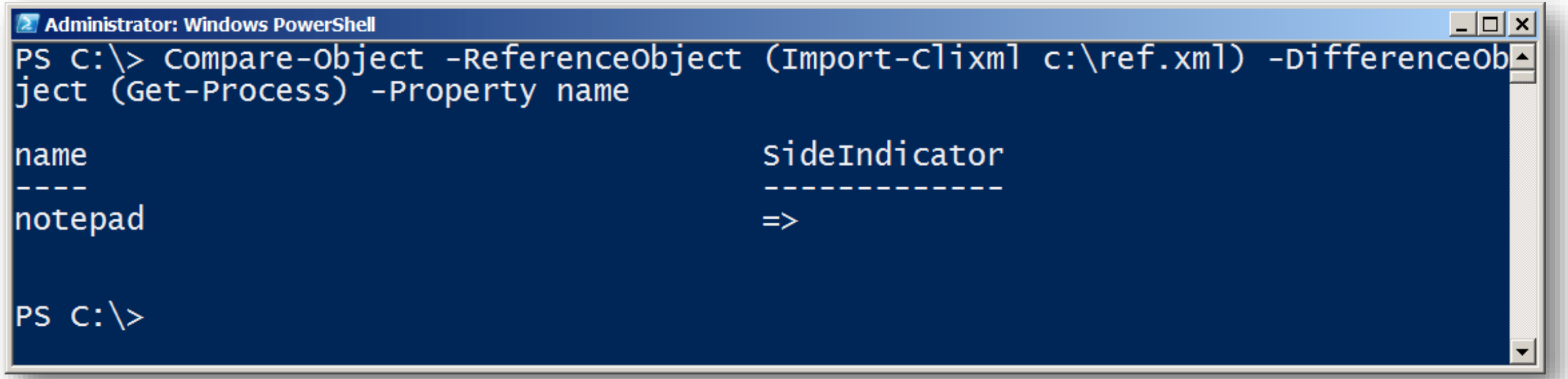


Exporting/Importing CSV

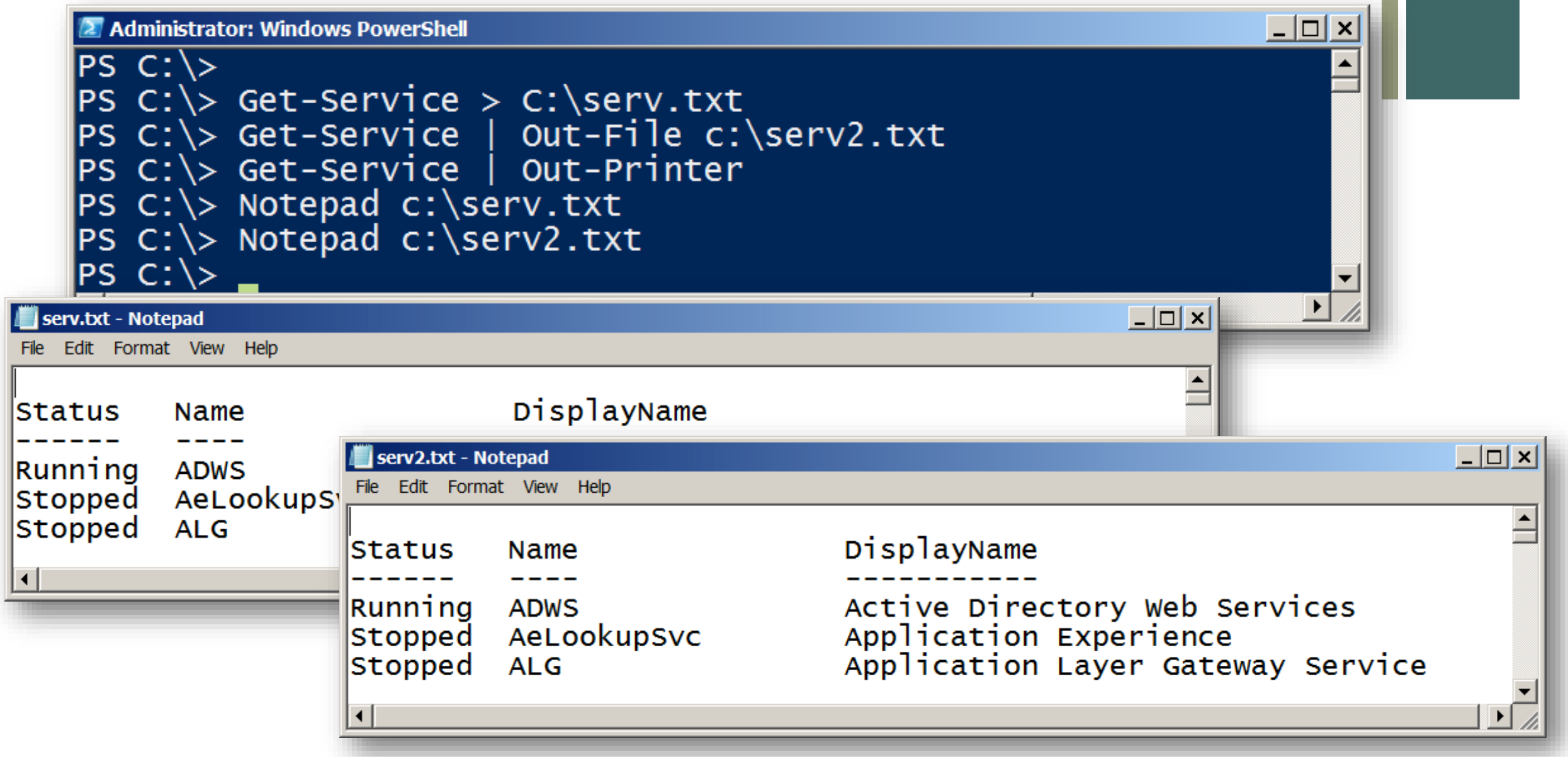


Exporting/Importing XML

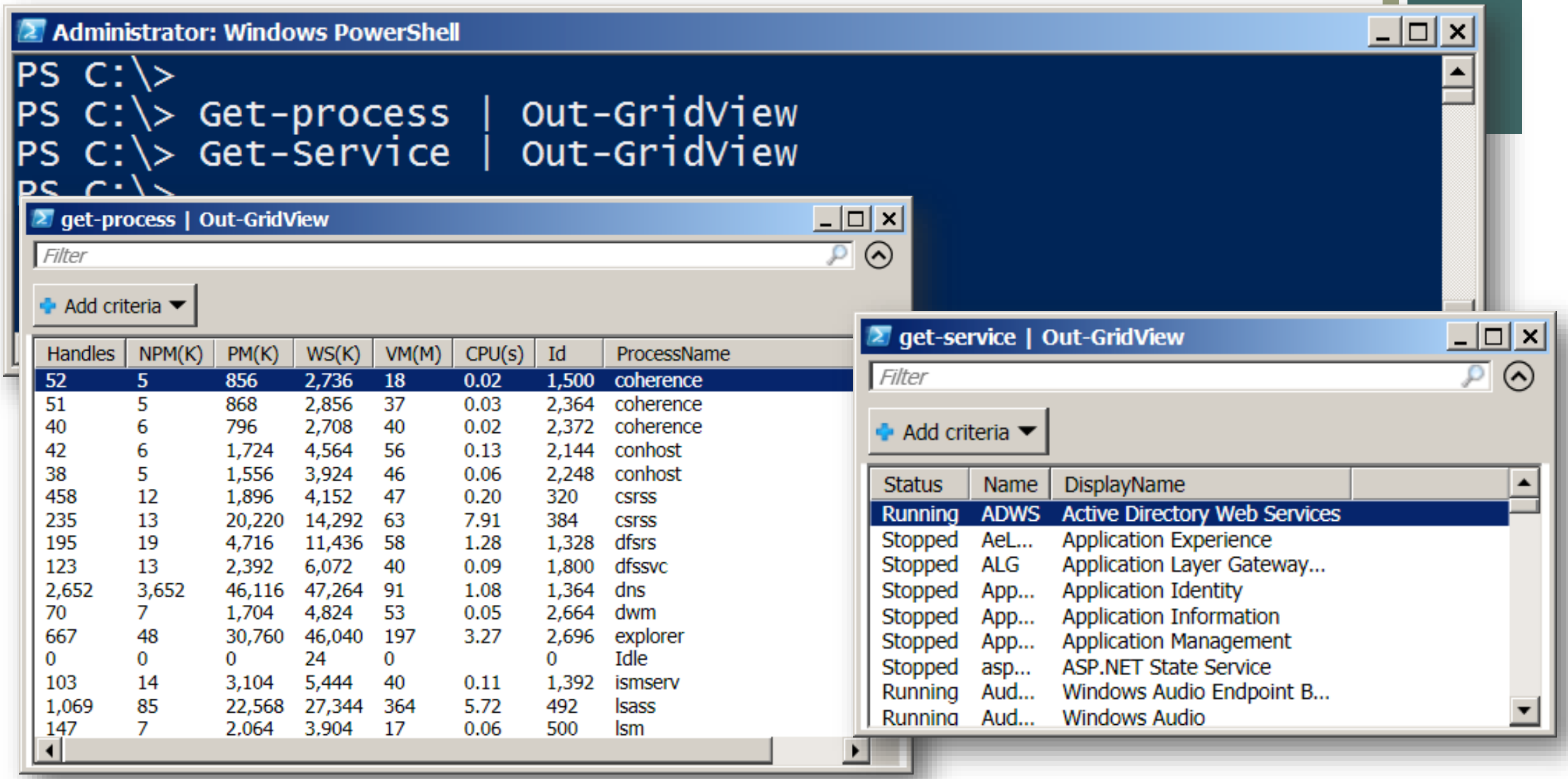




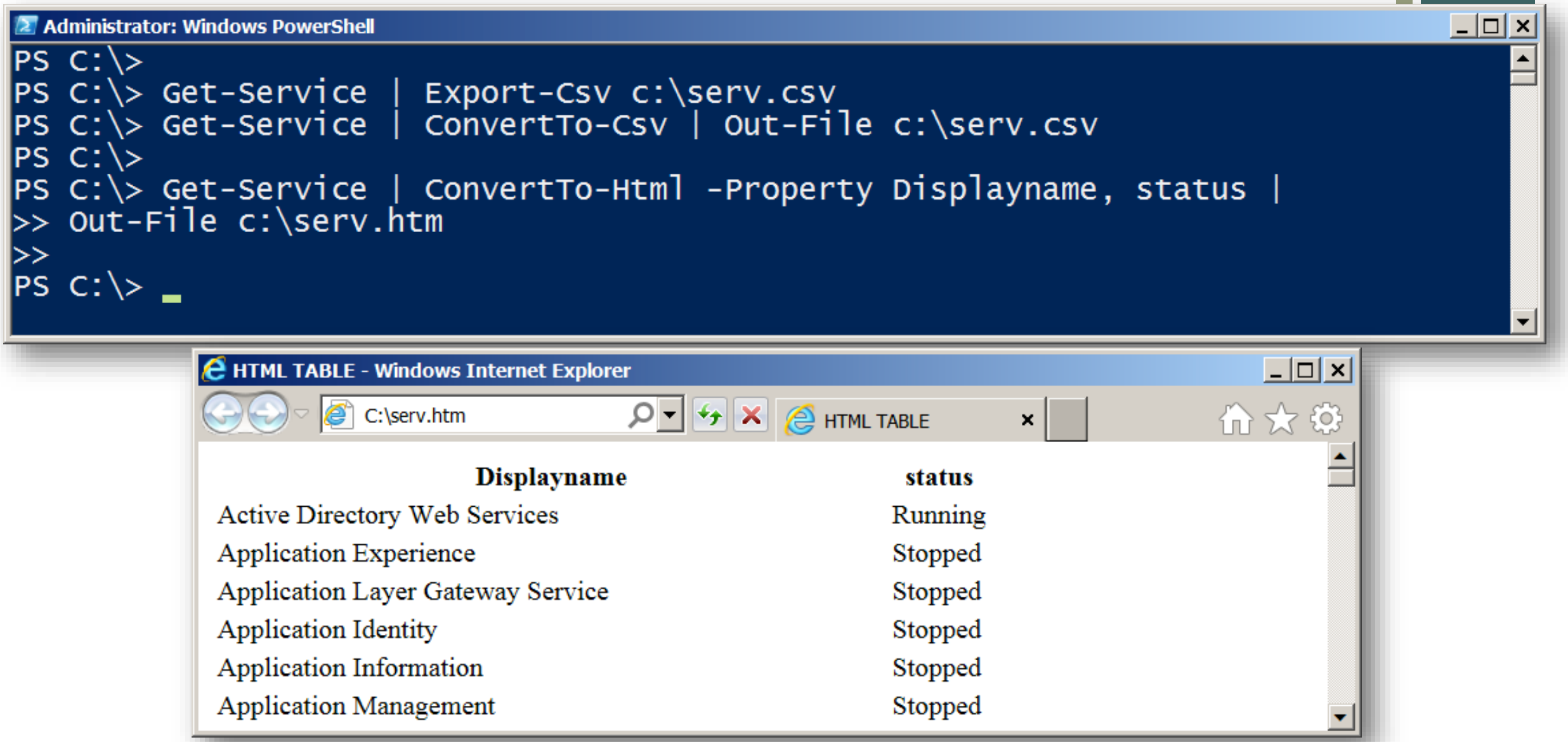
Other files and printers



Displaying information in a GUI



Making a webpage of information



Cmdlets that kill

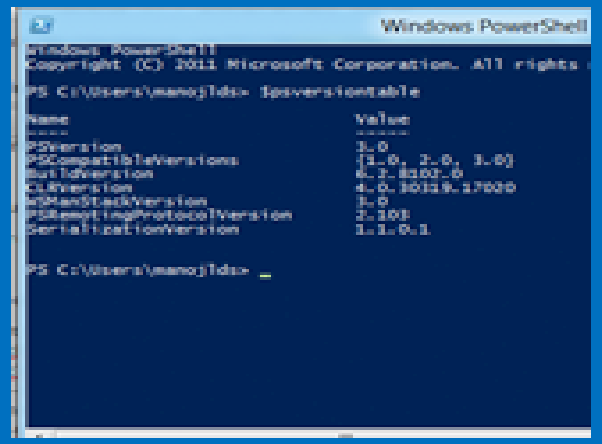
- $WhatIfPreference
- -Confirm
- -Whatif

Stop-Process / kill
Stop-service
$ConfirmPreference

```
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:\>



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*

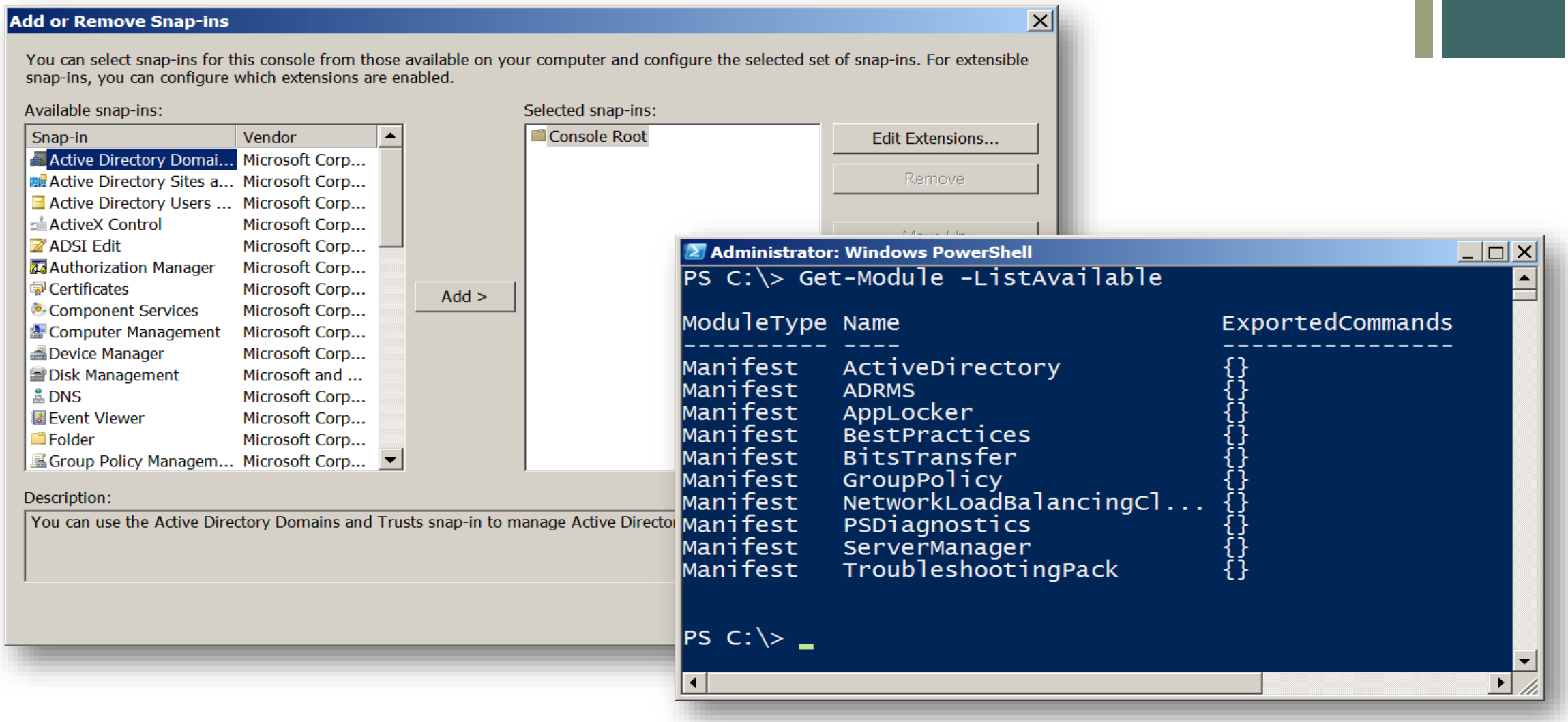
The Module Way

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

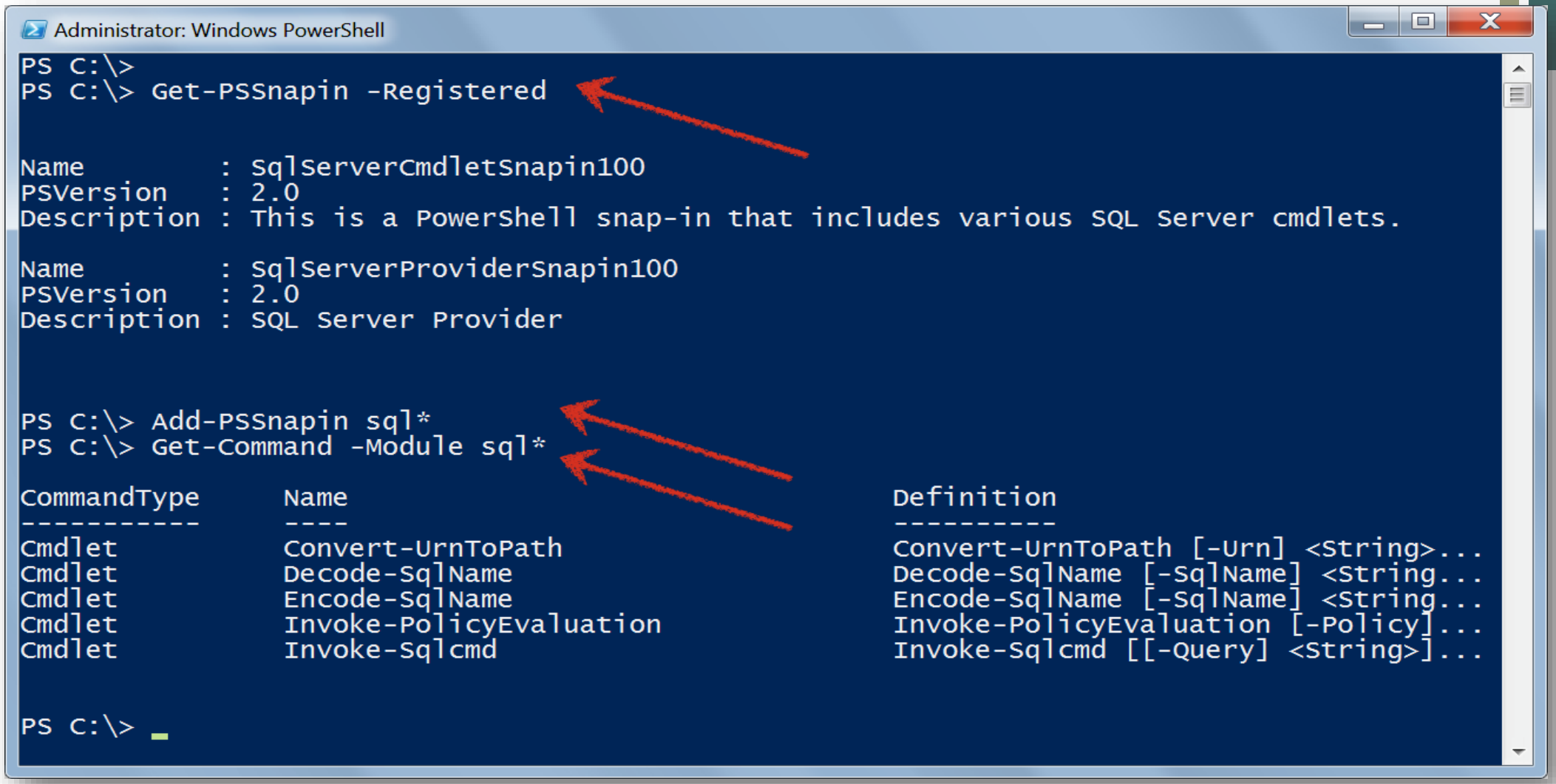
Three Crucial Commands

- Help
- Get-Command
-

Like the MMC - One Shell does it all

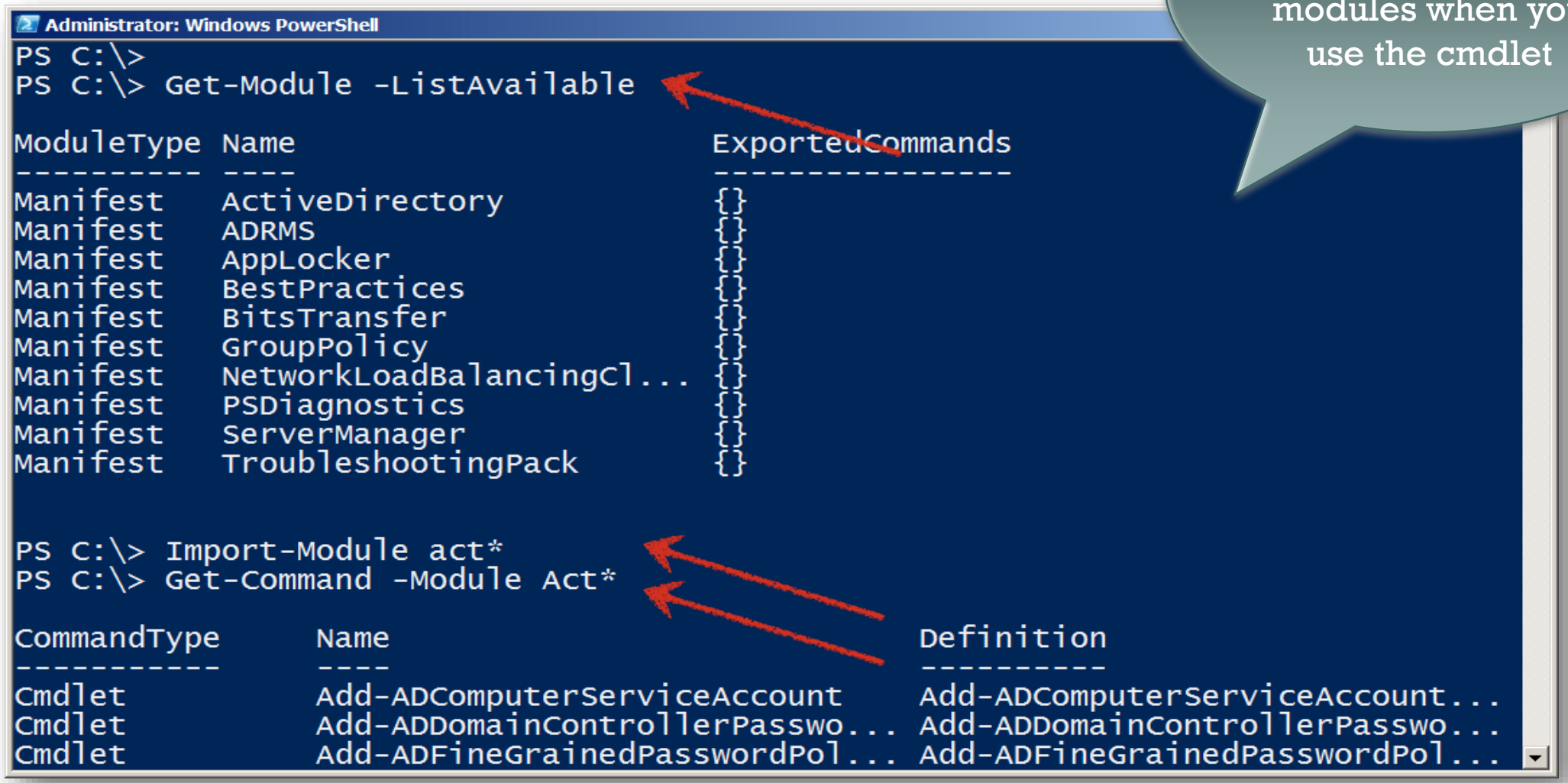


Finding & Adding Snap-ins

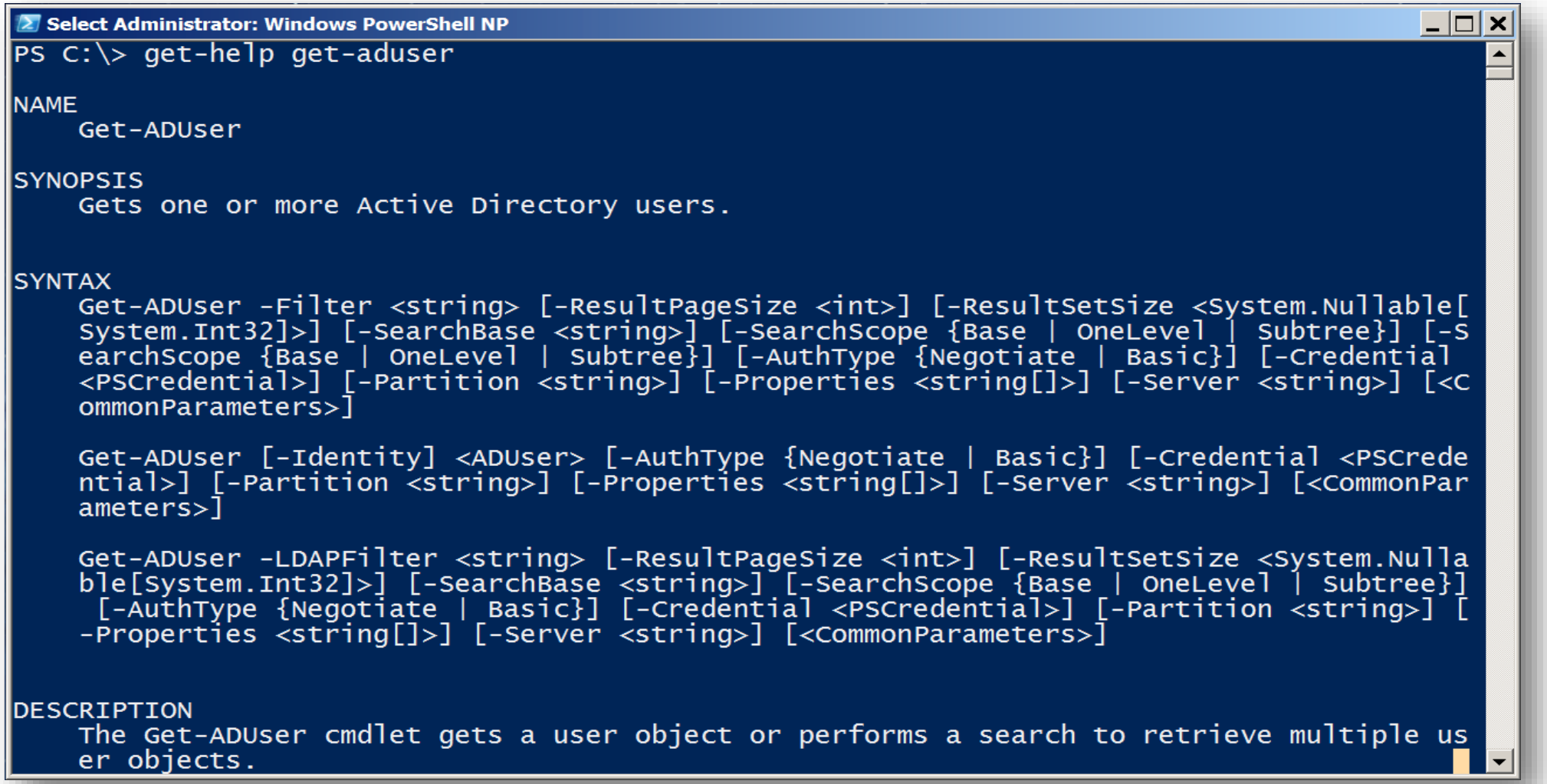


Finding & Adding Modules

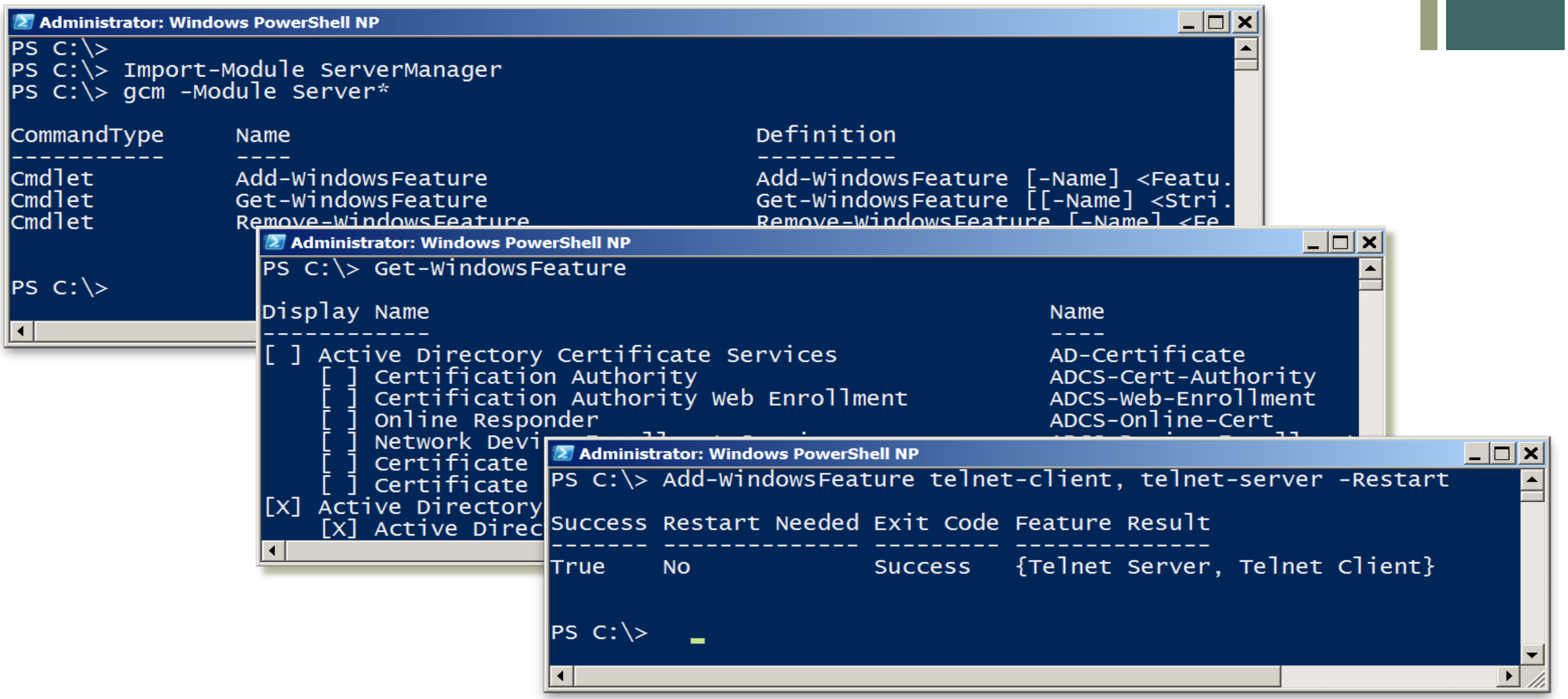
PowerShell V3 dynamically imports modules when you use the cmdlet



Discovering new commands



The real world of cmdlets

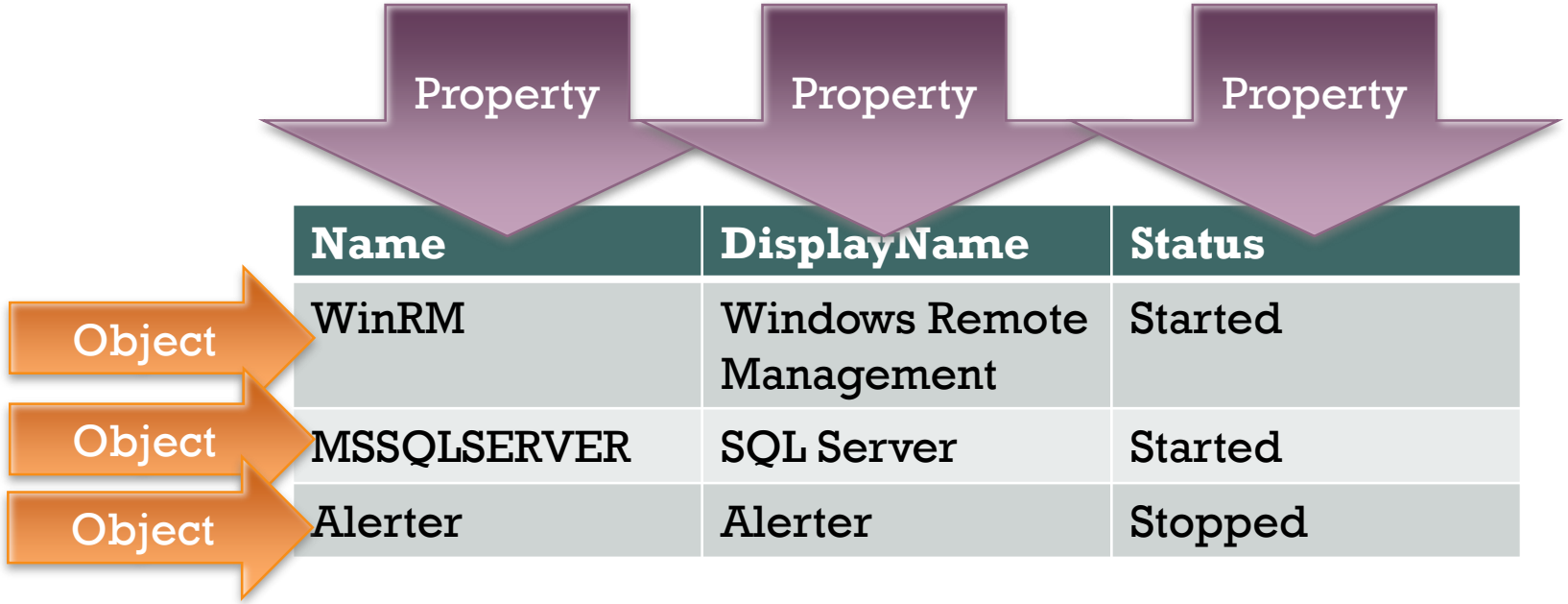


Module 8

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**

“Objects”



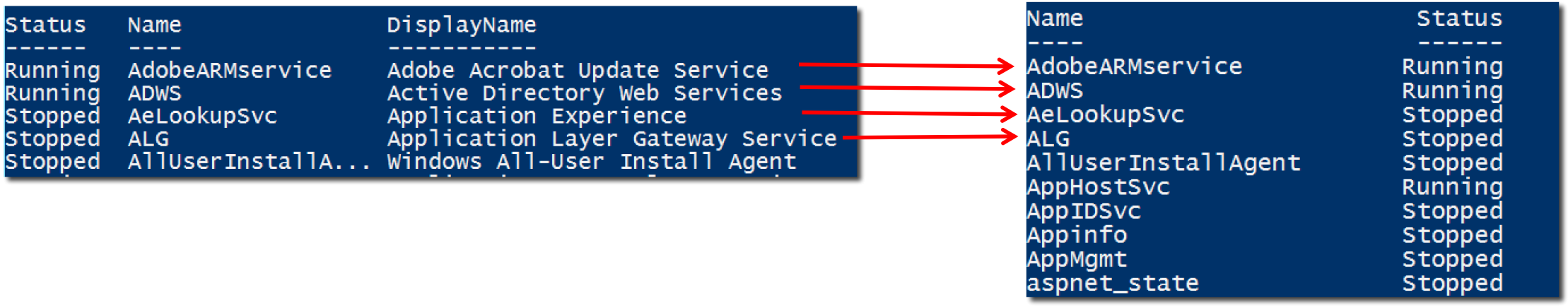
Name	DisplayName	Status
WinRM	Windows Remote Management	Started
MSSQLSERVER	SQL Server	Started
Alerter	Alerter	Stopped

Three Crucial Commands

- Help
- Get-Command
- Get-Member

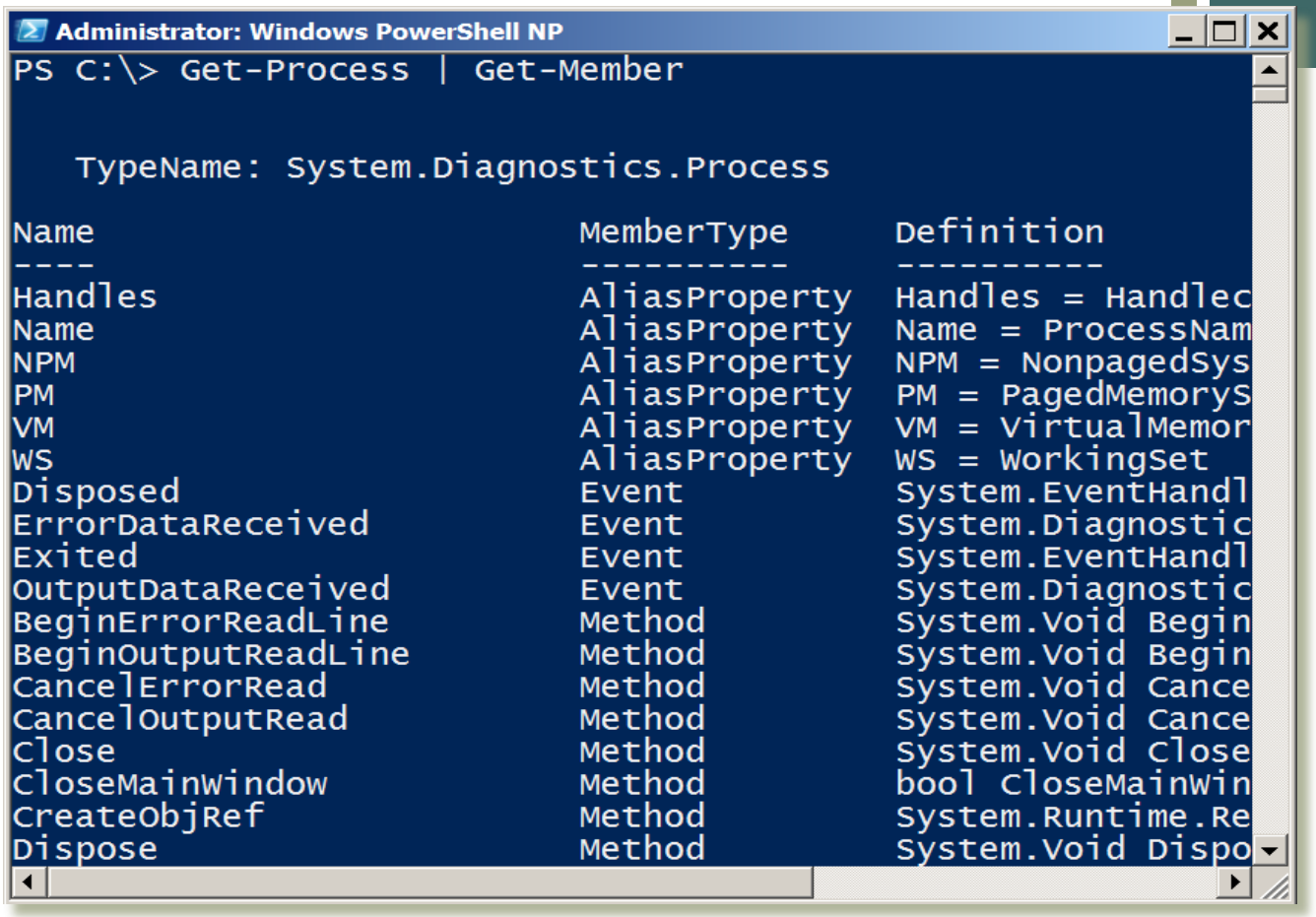
Object across the pipeline

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

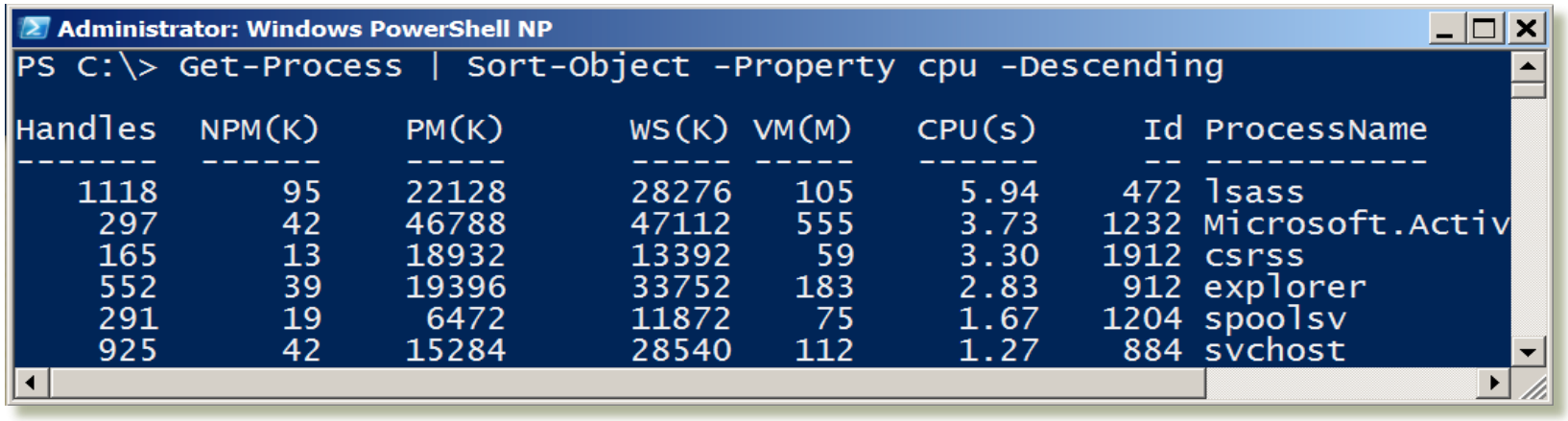


Getting the information you need

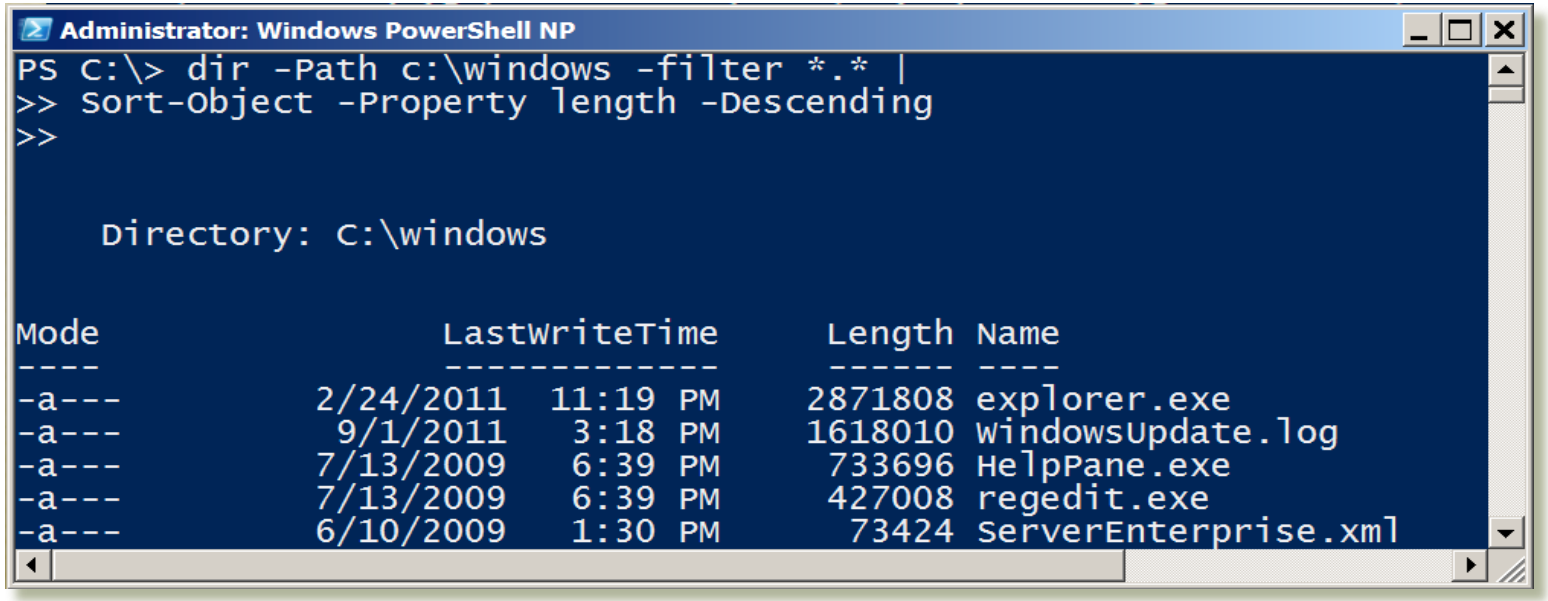
- Get-Member (gm)
- TypeName is a unique Windows assigned name
- Displays the properties and methods of an object
- Properties are potential columns of information
- Methods are the potential actions that can be taken



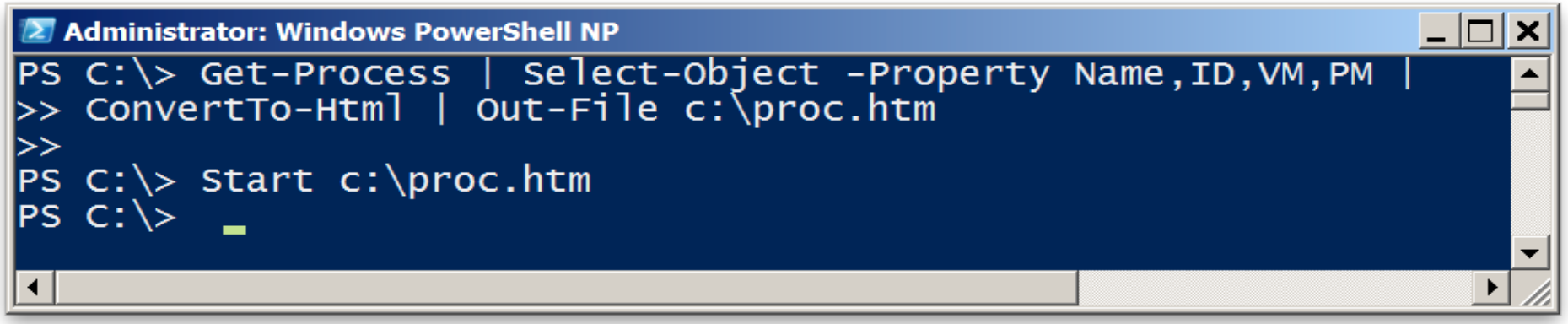
Sorting Objects



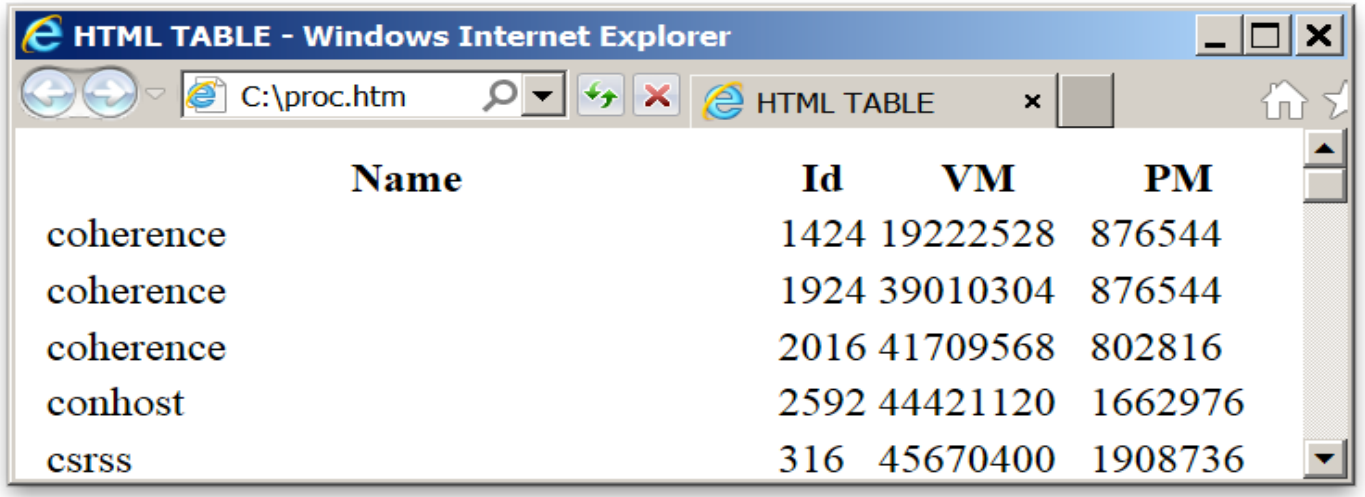
- Sort-Object sorts properties.
- Use Get-Member to see a list of properties



Selecting Objects

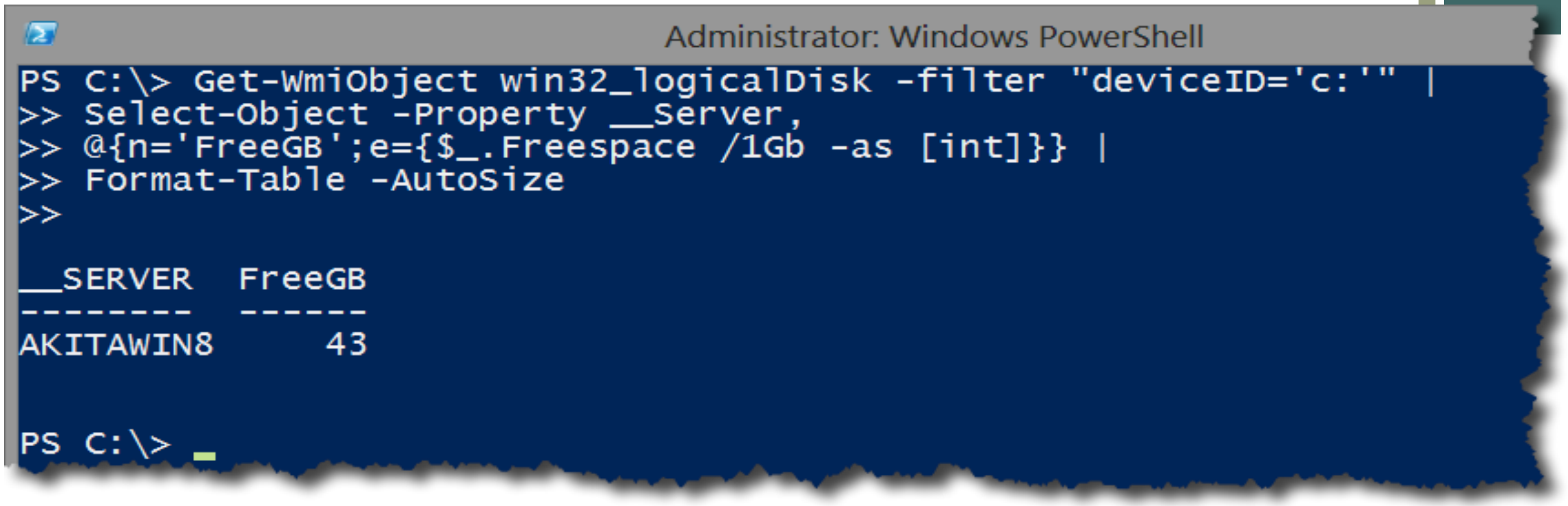


- Select-Object selects properties.
- Use Get-Member to list properties to select from.
- -first and -last restrict list of rows displayed.

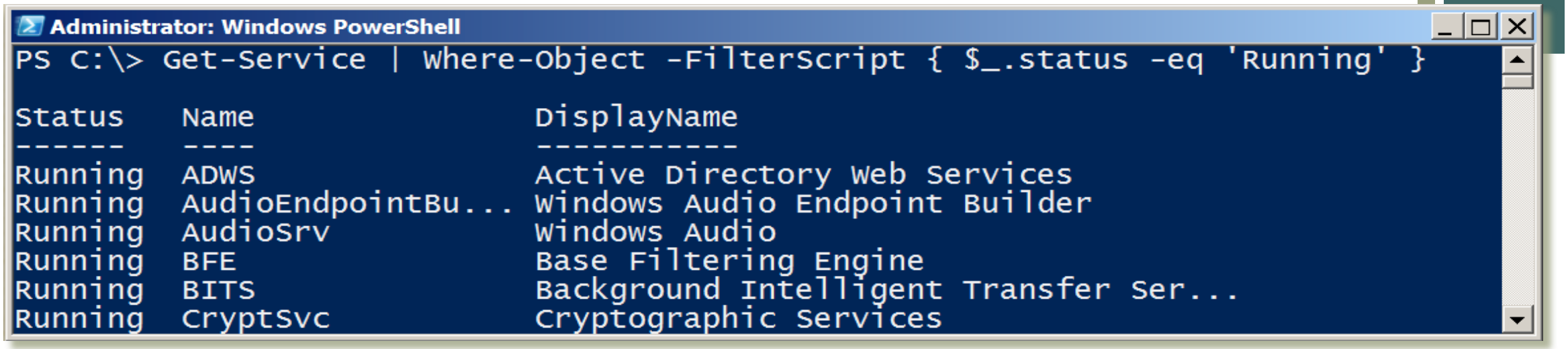


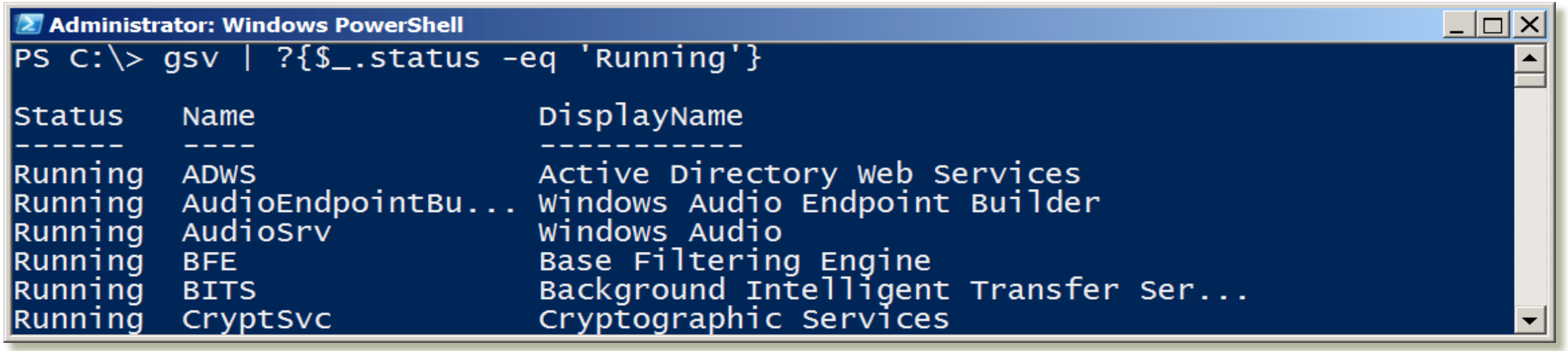
Name	Id	VM	PM
coherence	1424	19222528	876544
coherence	1924	39010304	876544
coherence	2016	41709568	802816
conhost	2592	44421120	1662976
csrss	316	45670400	1908736

Custom Properties



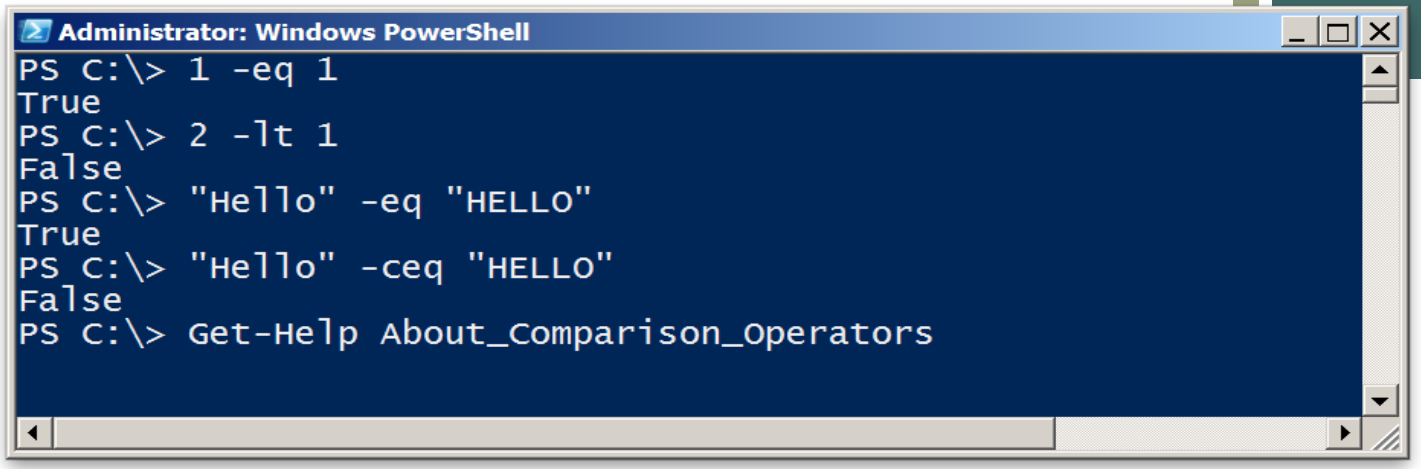
Filter Object Out of the Pipeline

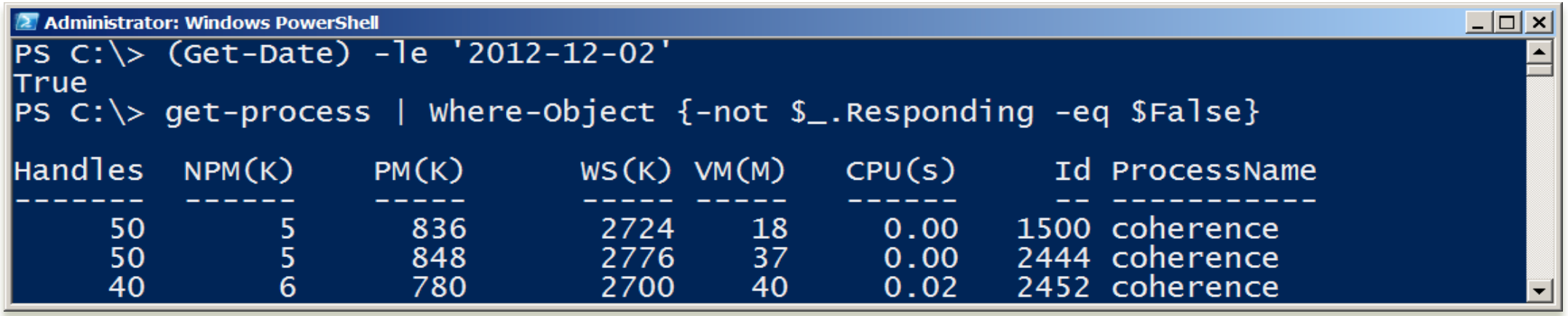




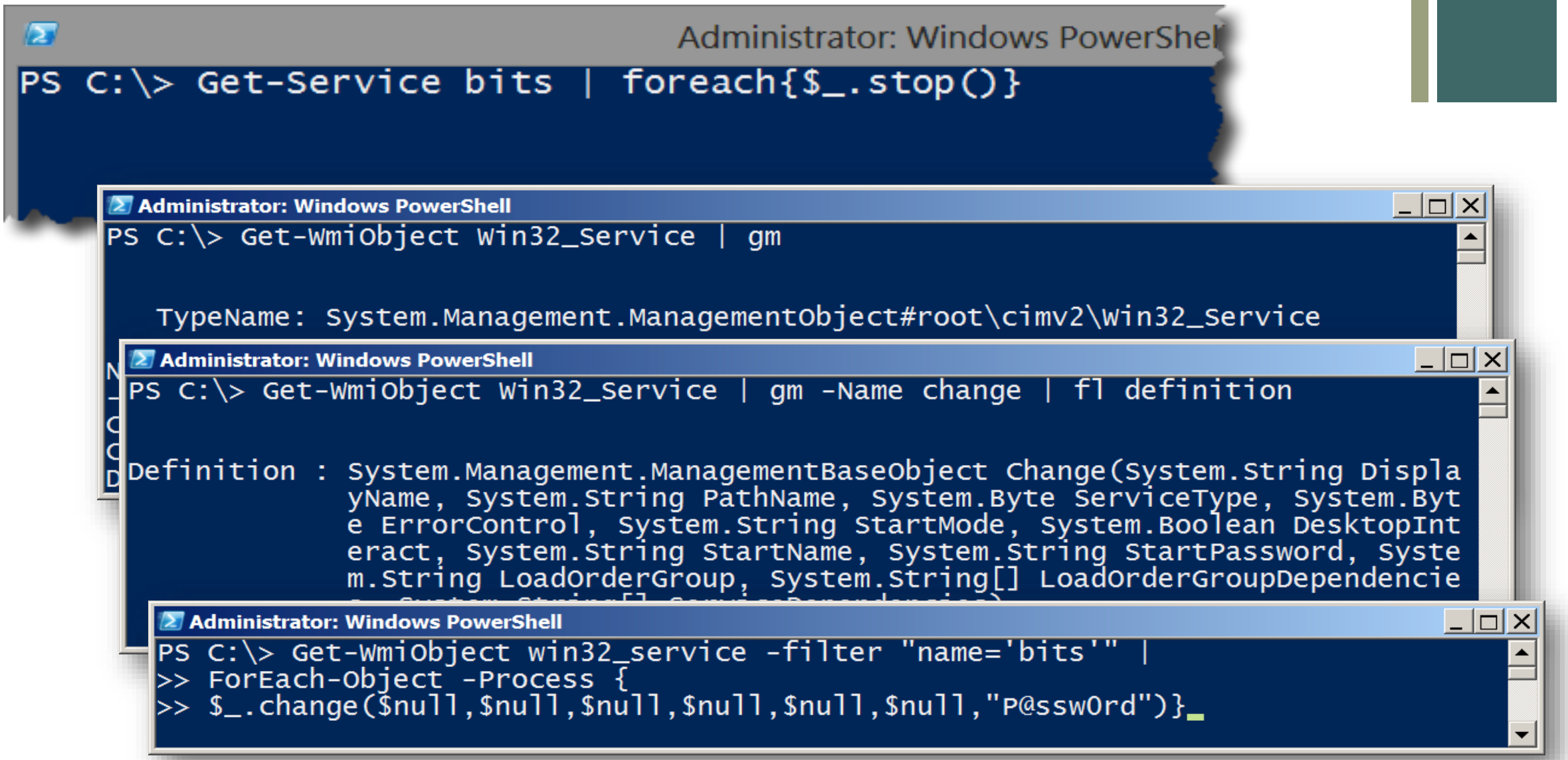
Comparison Operators

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





Methods – When no cmdlet exists



>

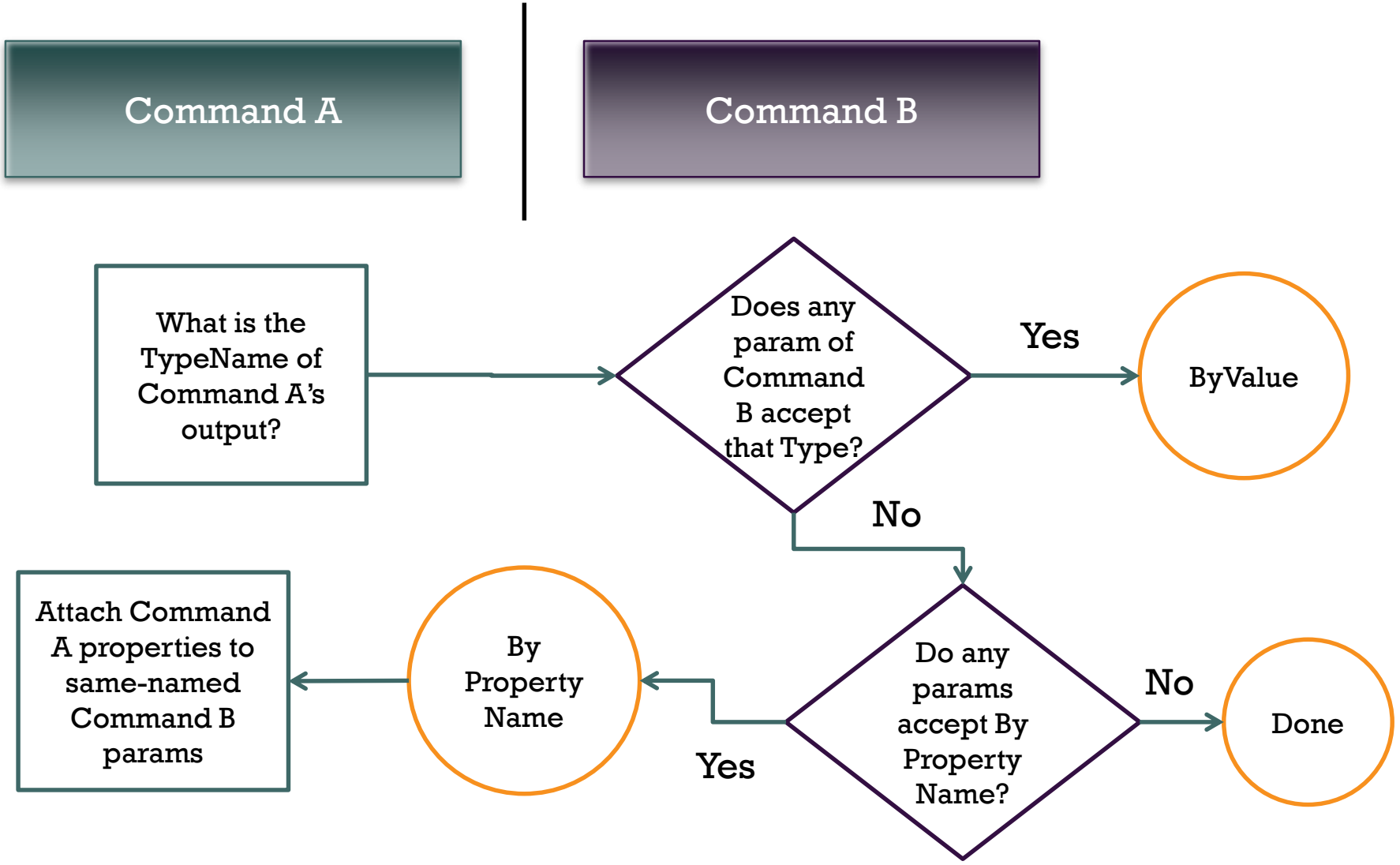
DAY 2

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
```

```
Status   Name               DisplayName
------   ----               -----------
Running  AdobeARMservice    Adobe Acrobat Update Service
Running  ADWS               Active Directory Web Services
Stopped  AeLookupSvc        Application Experience
Stopped  ALG                Application Layer Gateway Service
Stopped  AllUserInstallA... Windows All-User Install Agent
```

```
Name                           Status
----                           ------
AdobeARMservice                Running
ADWS                           Running
AeLookupSvc                    Stopped
ALG                            Stopped
AllUserInstallAgent            Stopped
AppHostSvc                     Running
AppIDSvc                       Stopped
Appinfo                        Stopped
AppMgmt                        Stopped
aspnet_state                   Stopped
```

ByValue

1. Get-Service passes ServiceController objects to the pipeline

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

   TypeName: System.ServiceProcess.ServiceController
```

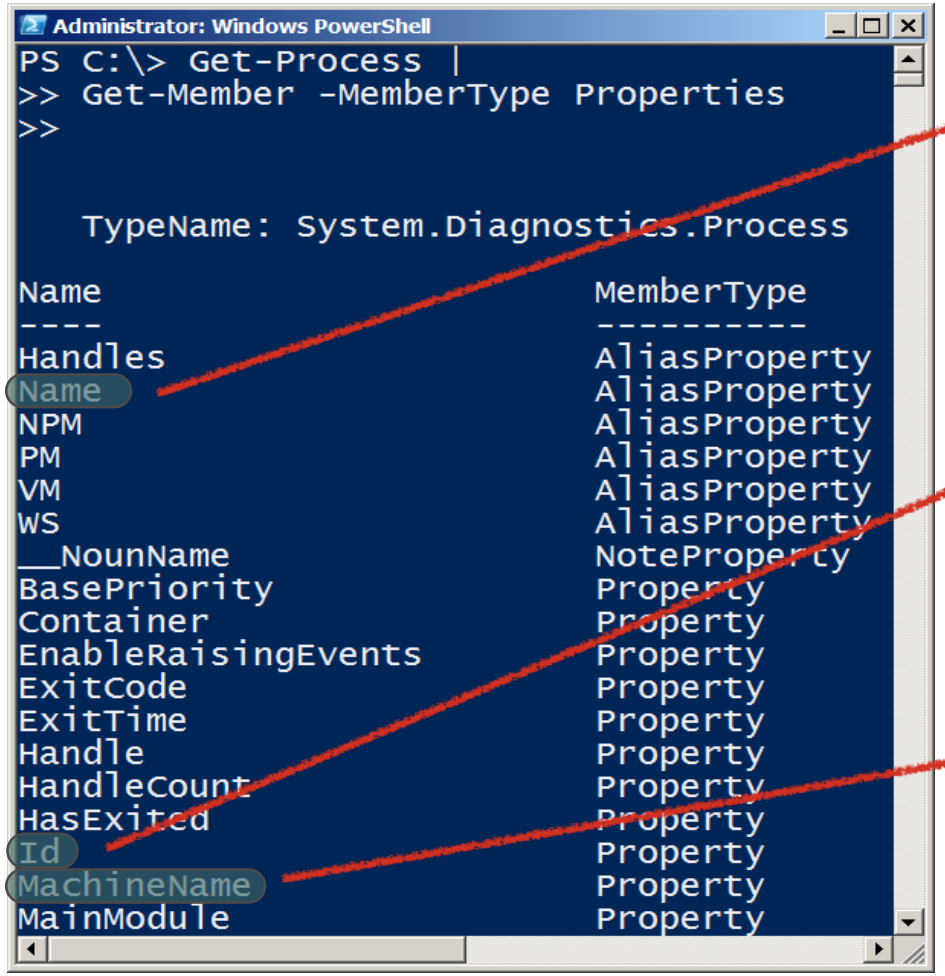
2. Does Stop-Service accept ServiceController Objects?

```
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 sto
    pped. 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




```
-Name <string[]>
```

```
Required?                    false
Position?                    1
Default value
Accept pipeline input?       true (ByPropertyName)
Accept wildcard characters?  true
```

```
-Id <Int32[]>
```

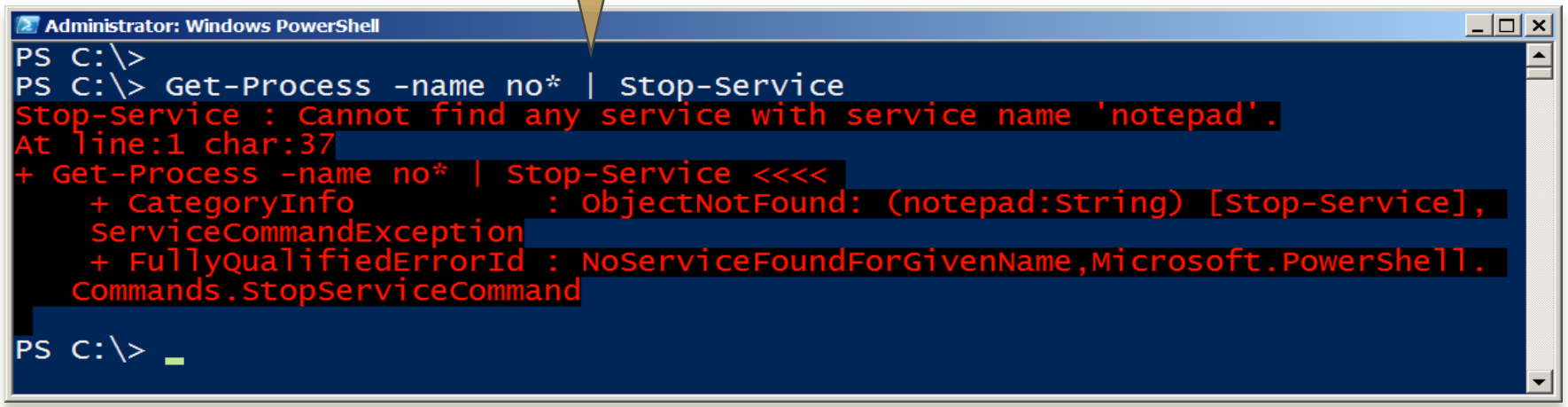
```
Required?                    true
Position?                    named
Default value
Accept pipeline input?       true (ByPropertyName)
Accept wildcard characters?  false
```

```
-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



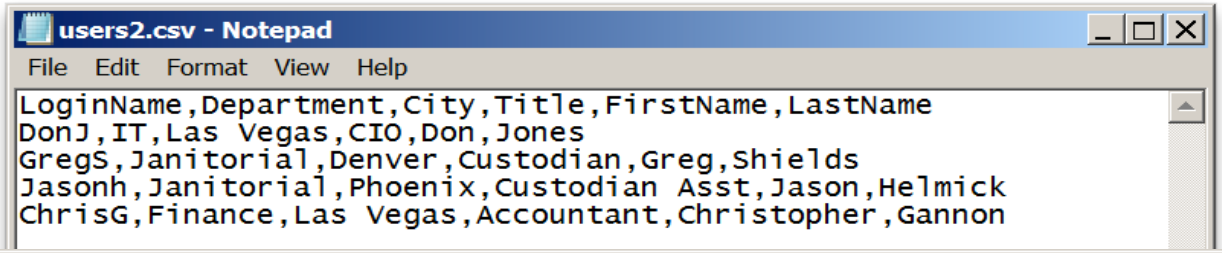
2. Stop-Service does not support accepting “Process” objects ByValue, so PowerShell checks what can be accepted ByPropertyName.

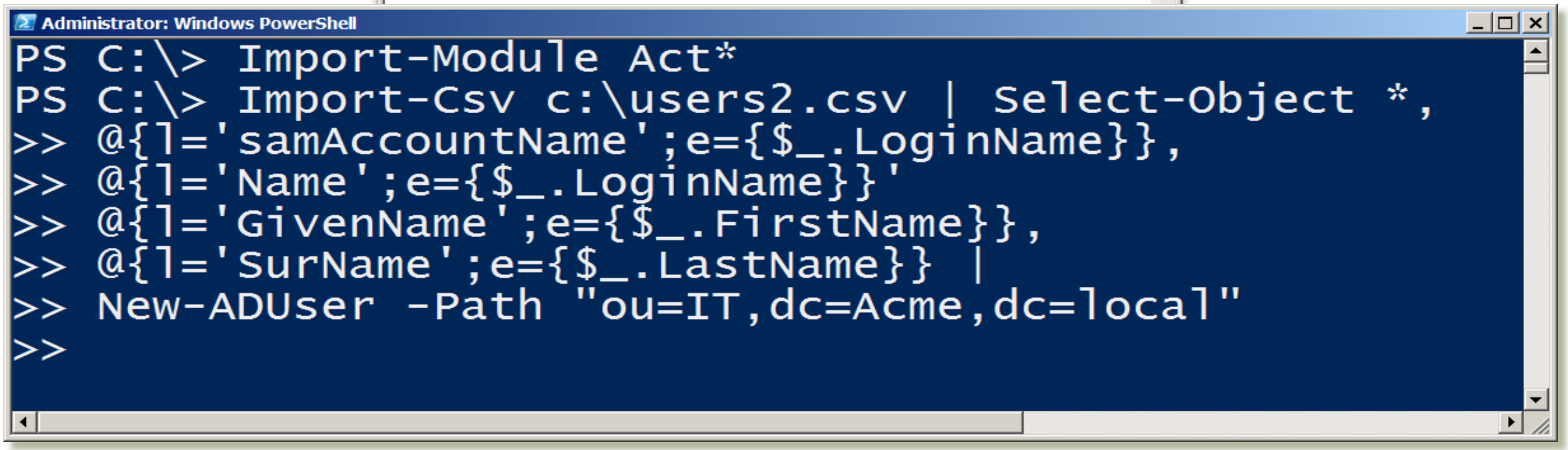
3. -Name does accept strings ByPropertyName, and the objects in the pipeline are labeled as a Name property

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

```
-Name <string[]>

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

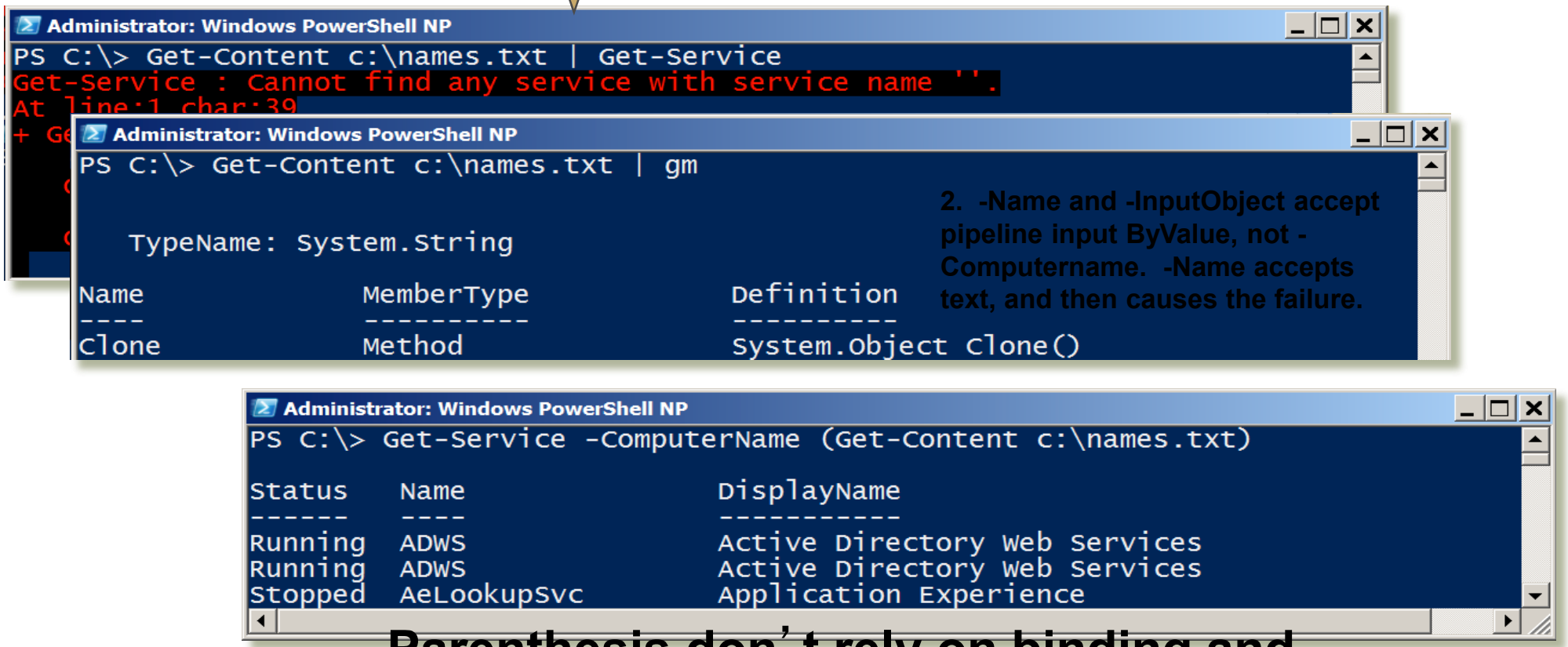
What if my property doesn't match – Customize it!





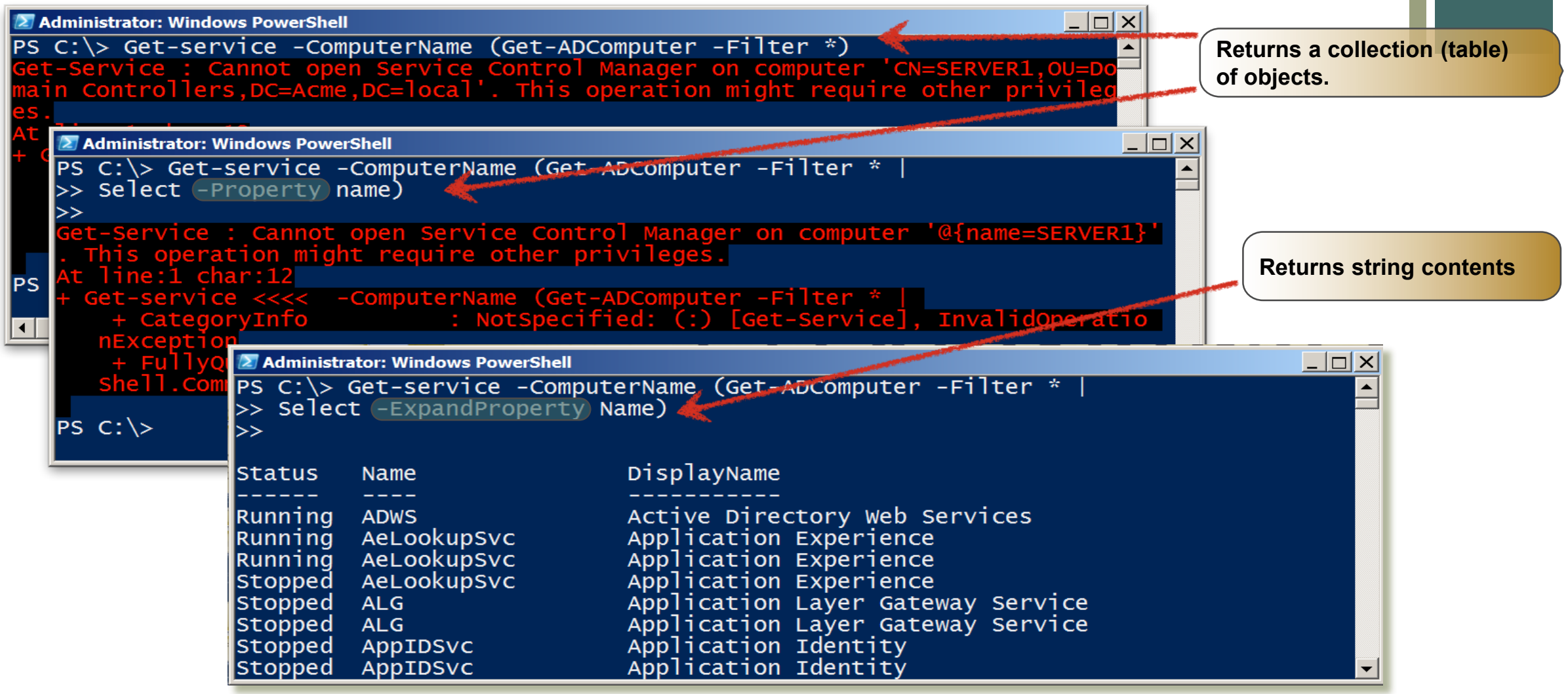
The Parenthetical – when all else fails

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



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

The Parenthetical – when all else fails



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

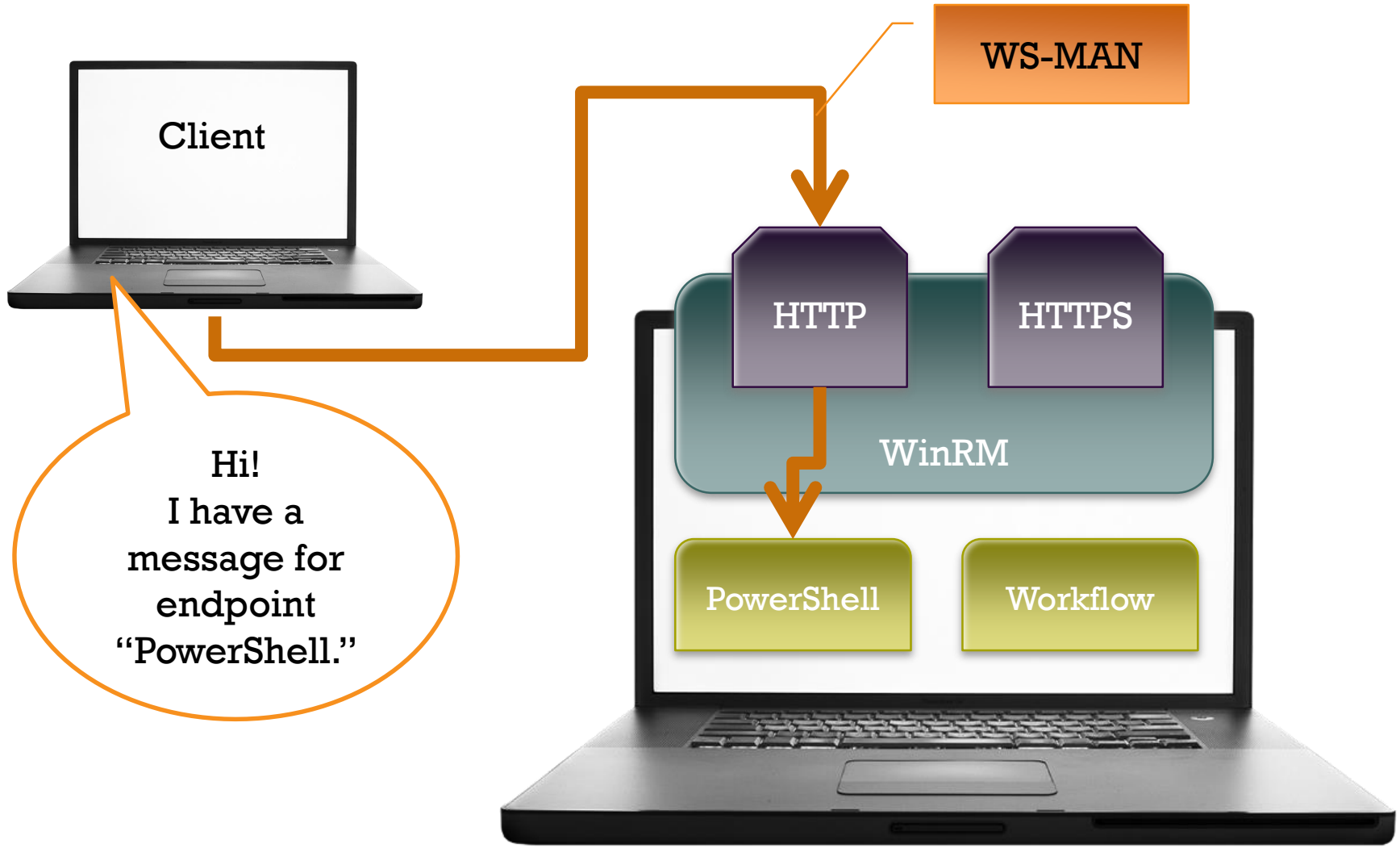
- Defining the Task
- Finding the Commands
- Learning to Use the Commands
- Tips for Teaching Yourself
- **Lab**

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**

Remoting Architecture

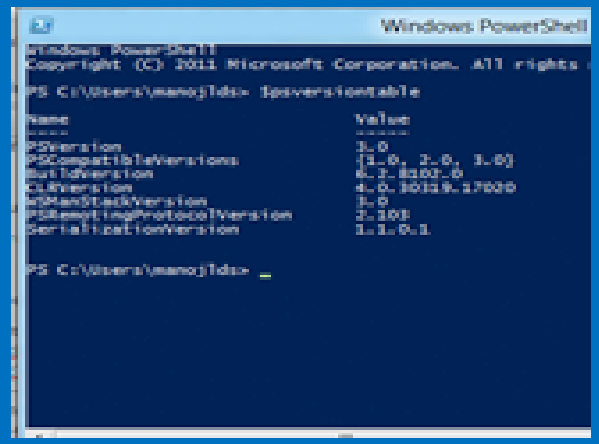


Module 14

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:\>



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
- .Ps1 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

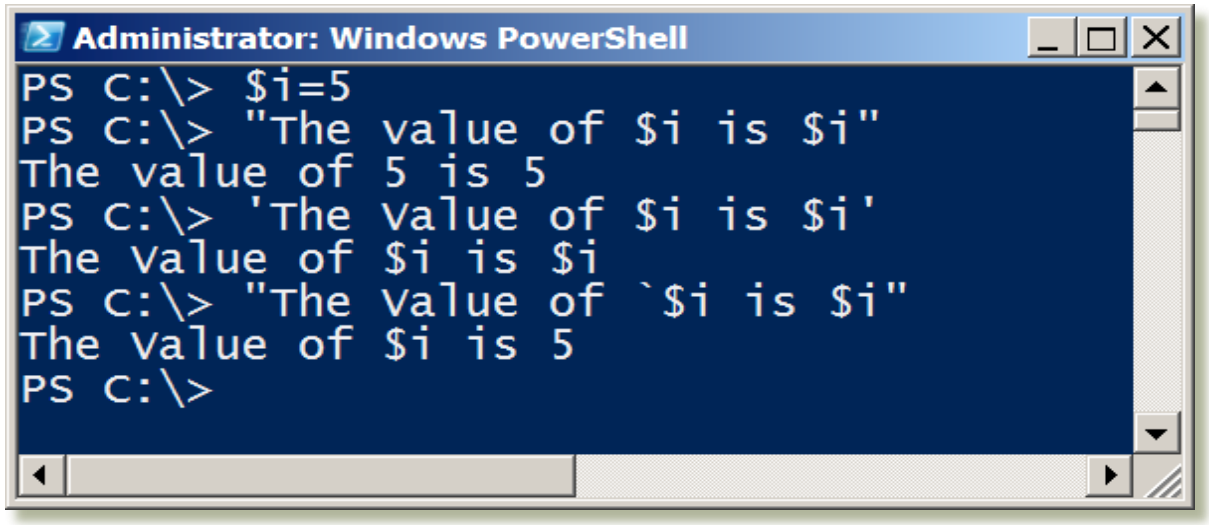
```
$myVariable
${My Variable}
```

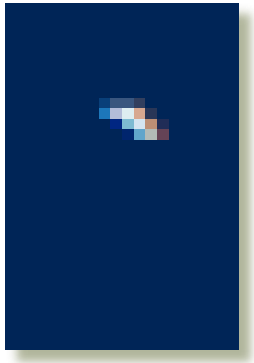
Note: The $ is not part of the variable name, it's a cue to access the contents of the variable

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



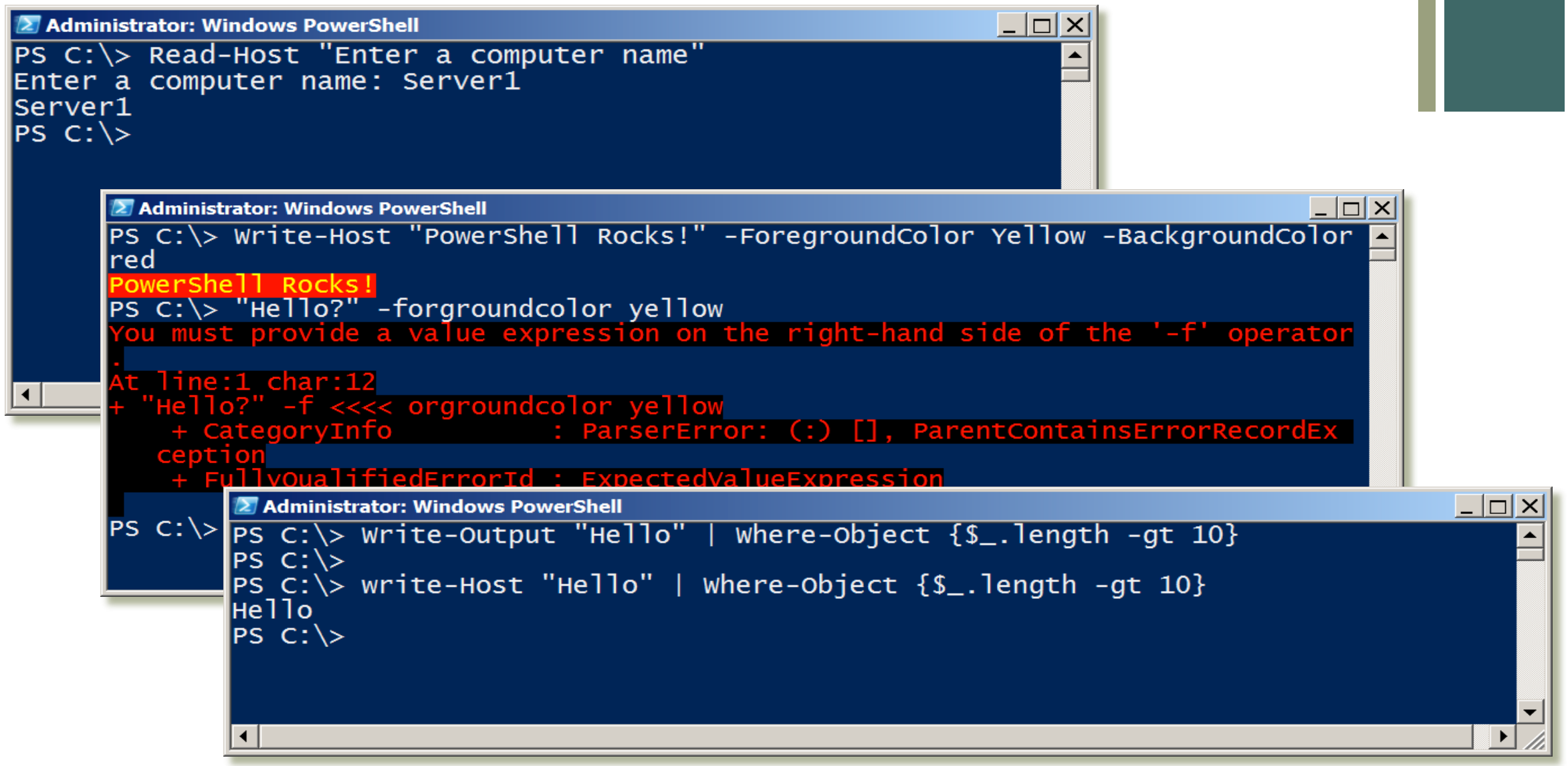


Module 19

Input and Output

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

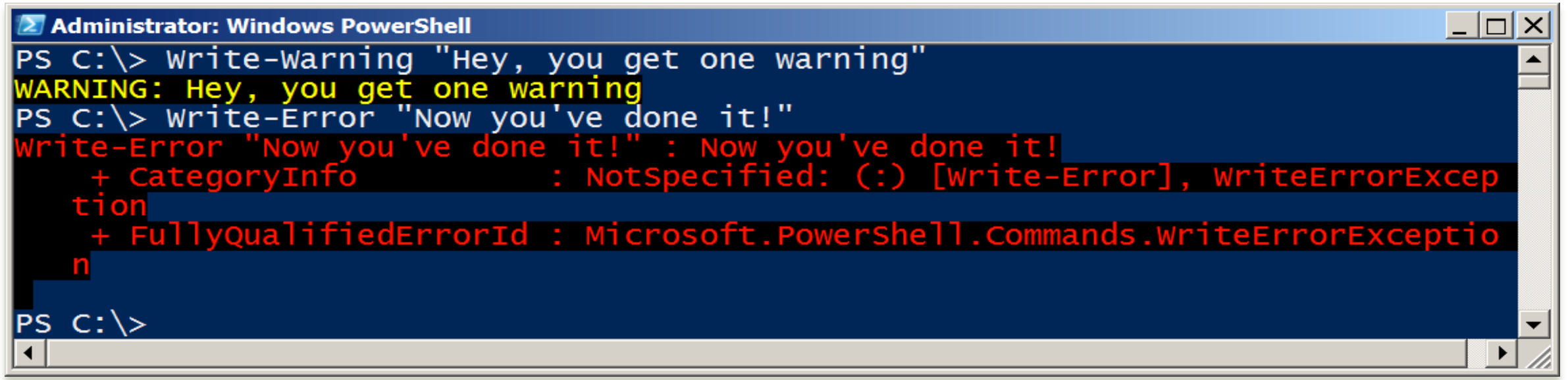
Getting and displaying input



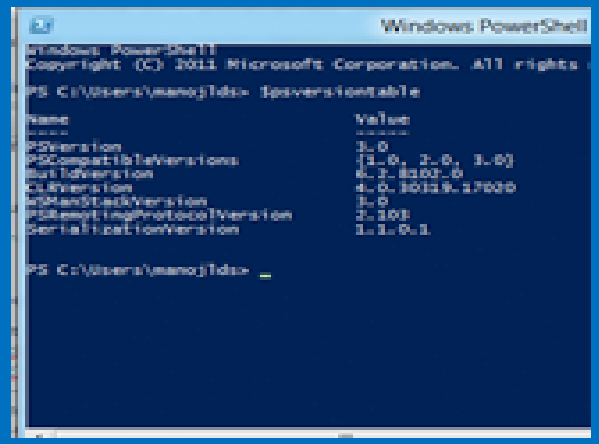
Other output for scripts and automation

- Write-Warning
- Write-Verbose
- Write-Debug
- Write-Error

- $Preference variables to know
- Help about_Preference_Variables
- $DebugPreference=SilentlyContinue
- $ErrorActionPreference=Continue
- #VerbosePreference=SilentlyContinue



PS C:\>



COMPLETE REVIEW LAB 3

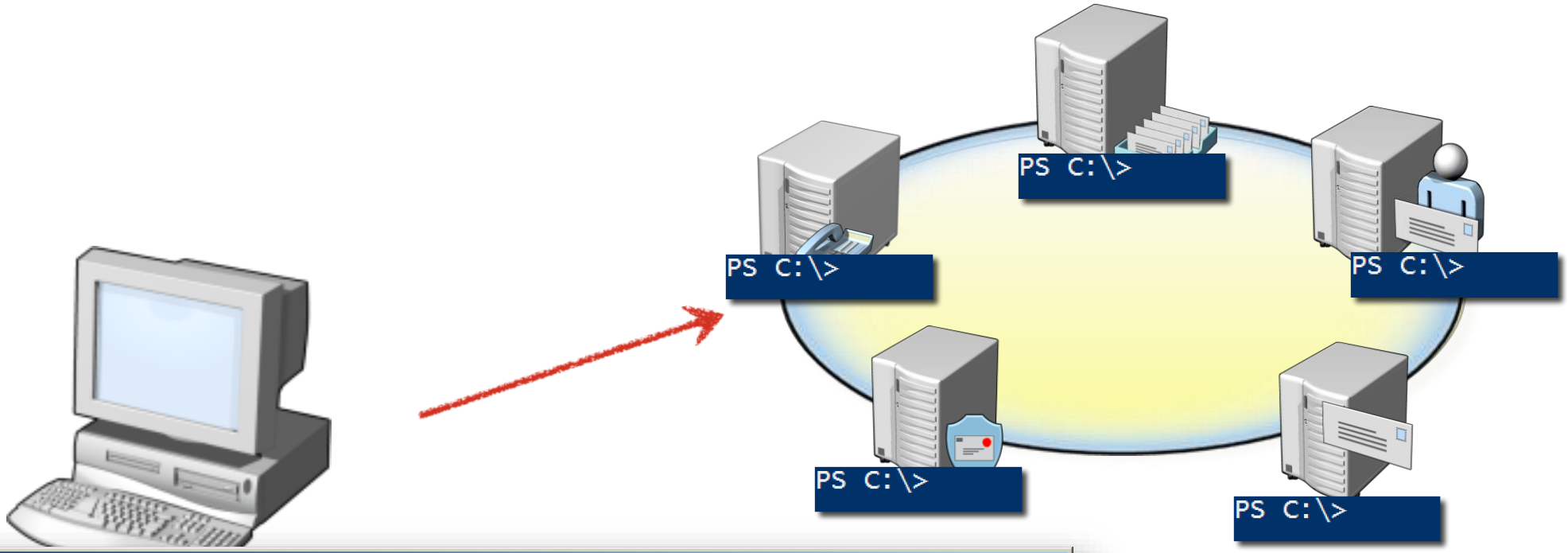
Based on Chapter 1-19

Module 20

Sessions:
Remote Control, with Less Work

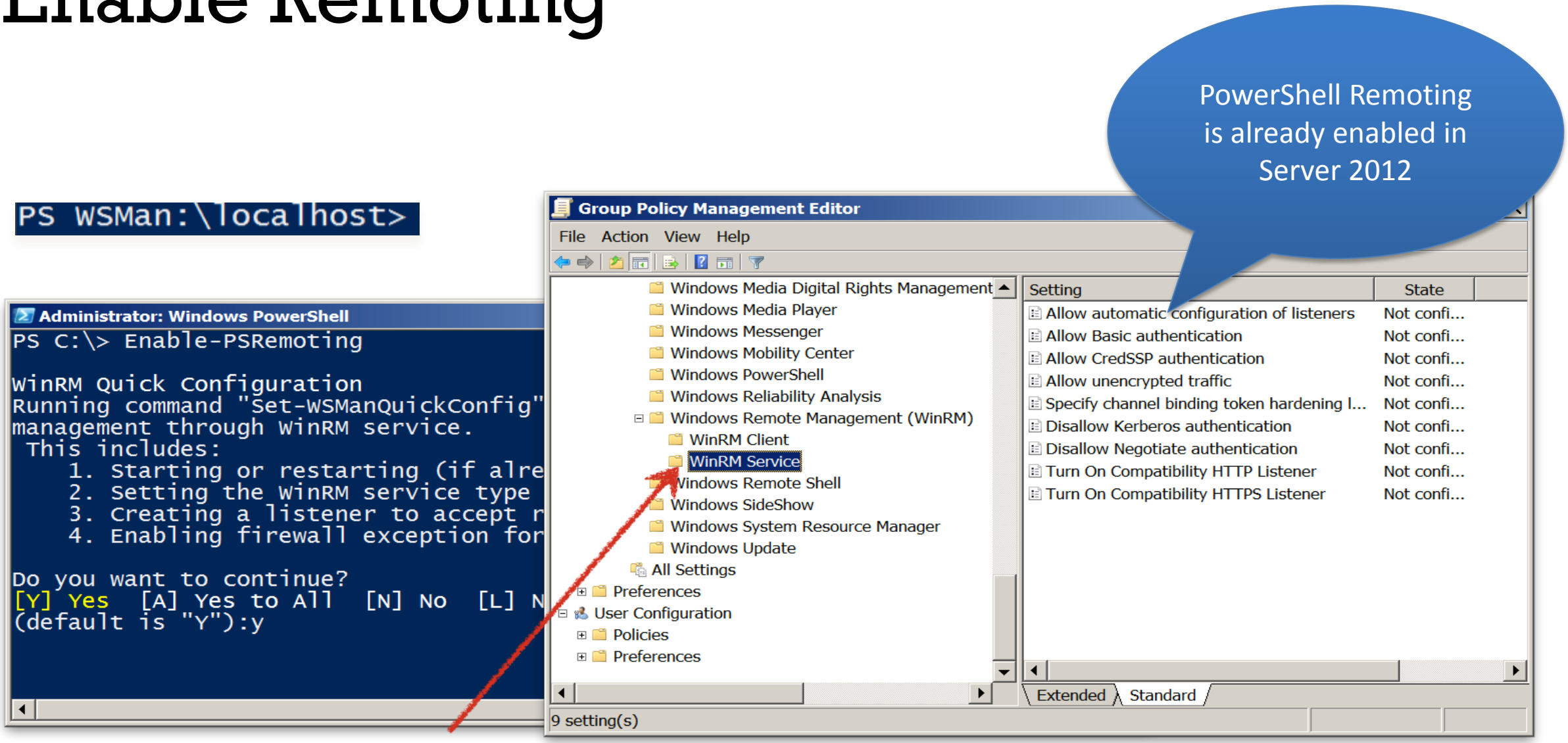
- Making PowerShell Remoting a Bit Easier
- Creating and Using Reusable Sessions
- Using Sessions with Enter-PSSession
- Using Sessions with Invoke-Command
- Implicit Remoting: Importing a Session
- Disconnected Sessions
- **Lab**

Overview of Remoting



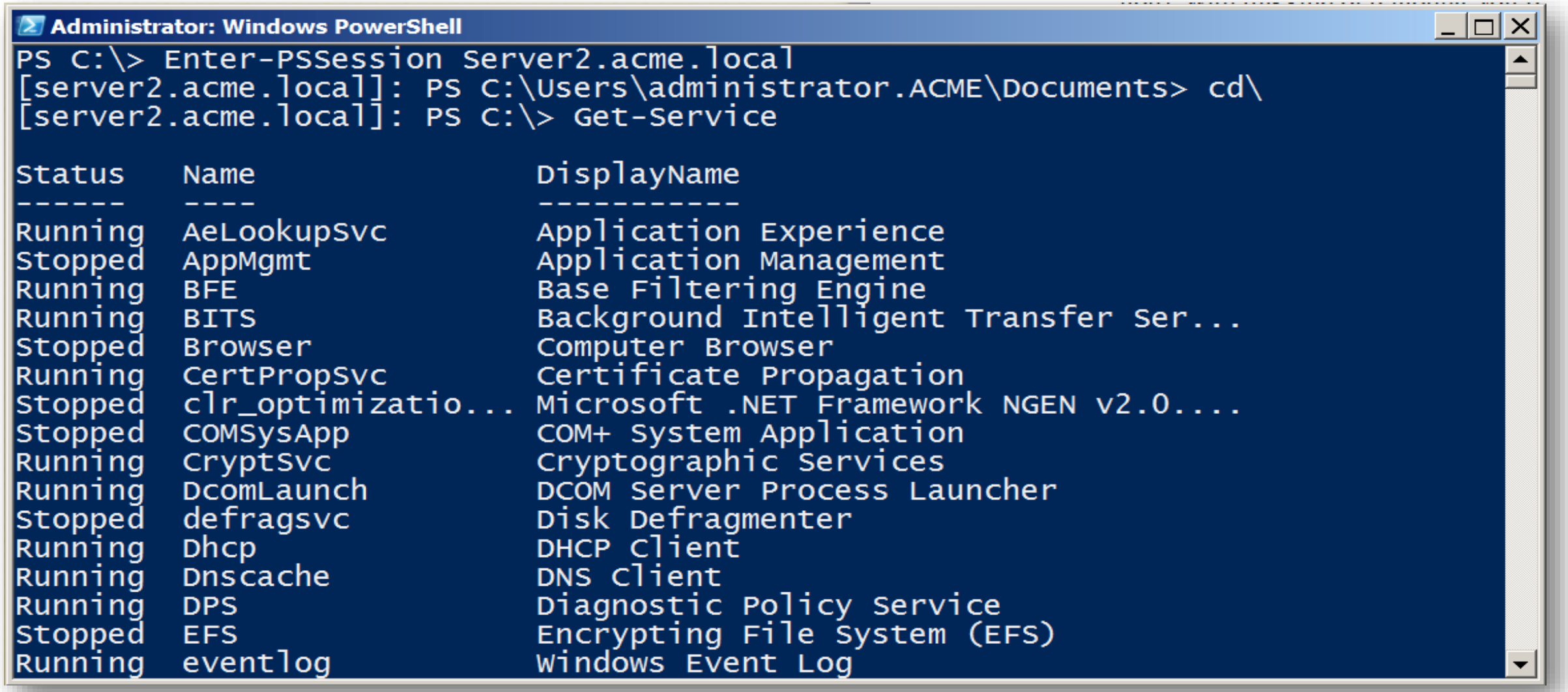

```
Administrator: Windows PowerShell
PS C:\> Enter-PSSession server1
[server1]: PS C:\Users\Administrator.SERVER1\Documents> Get-Service

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

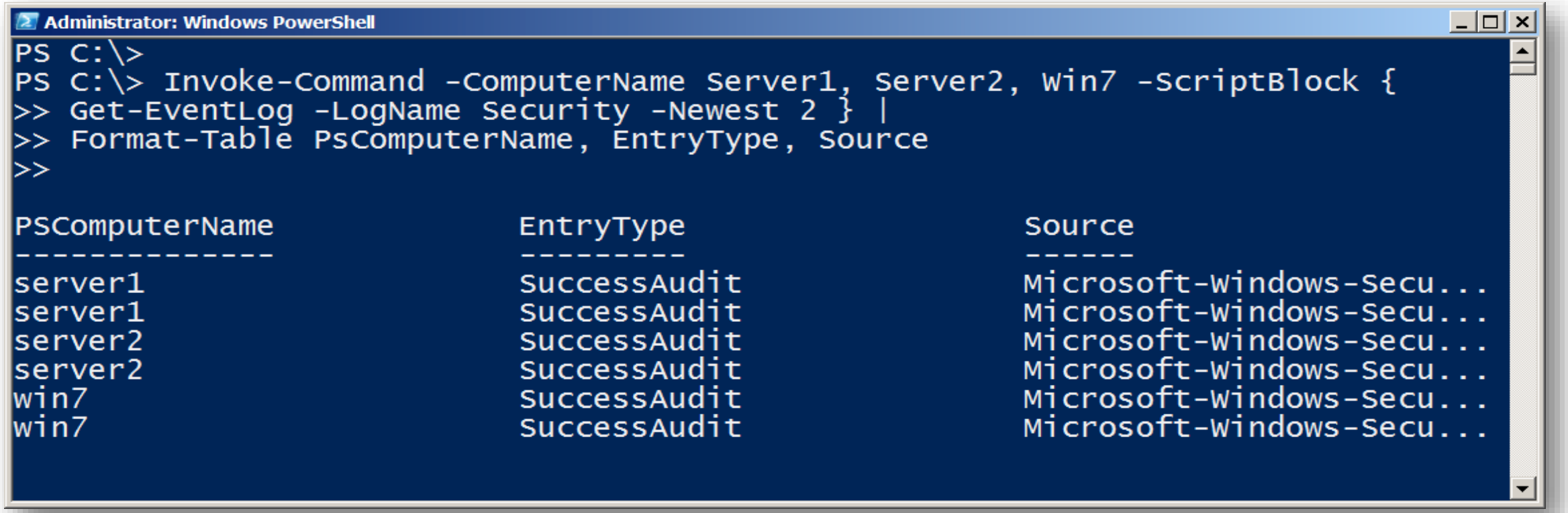


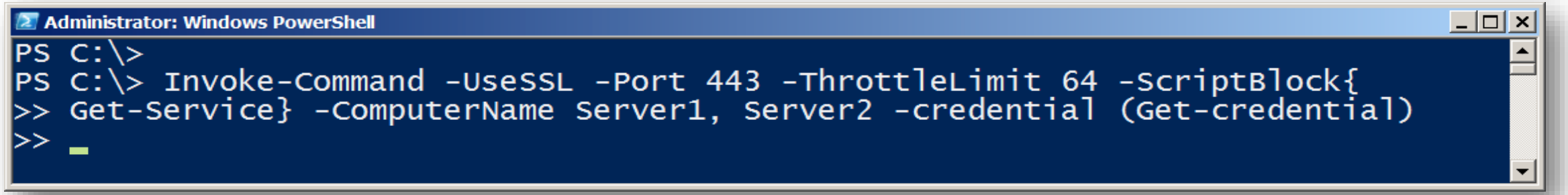
Computer Configuration/Policies/Administrative Templates/Windows Components/Windows Remote Management

One to One - Interactive



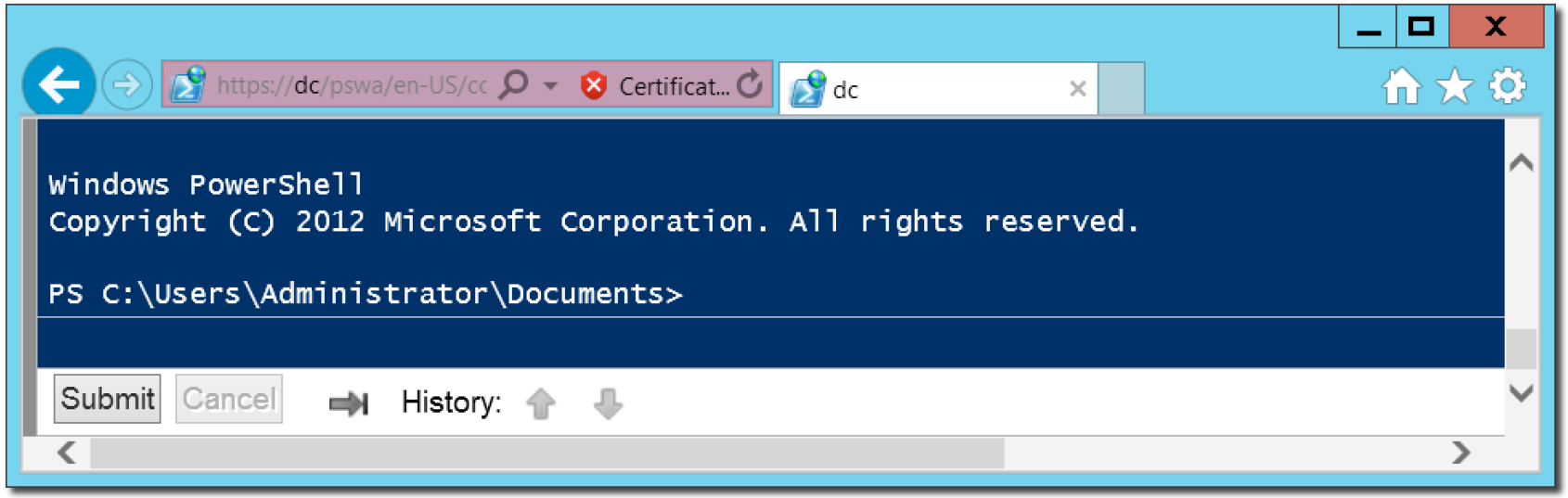
One-To-Many


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



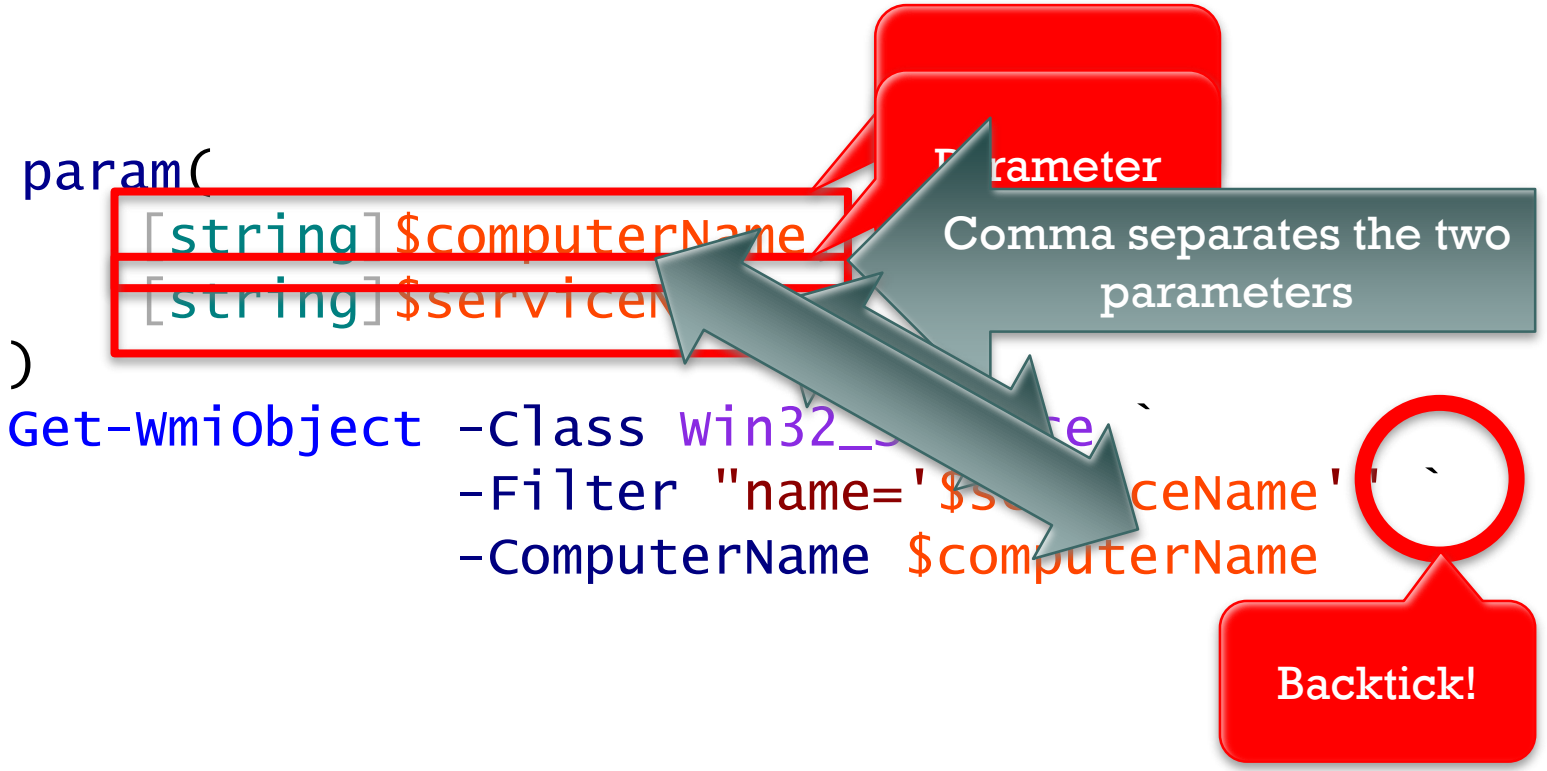
Not the end yet!

- 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/

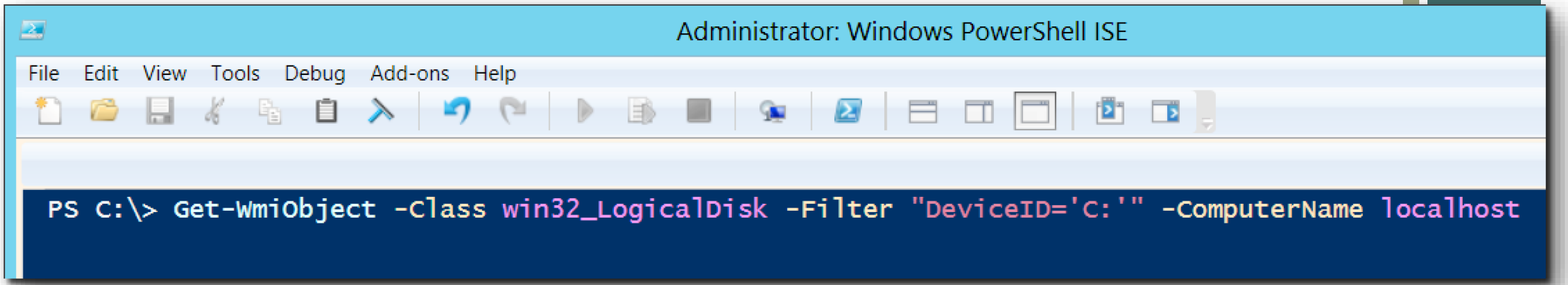
Module 21

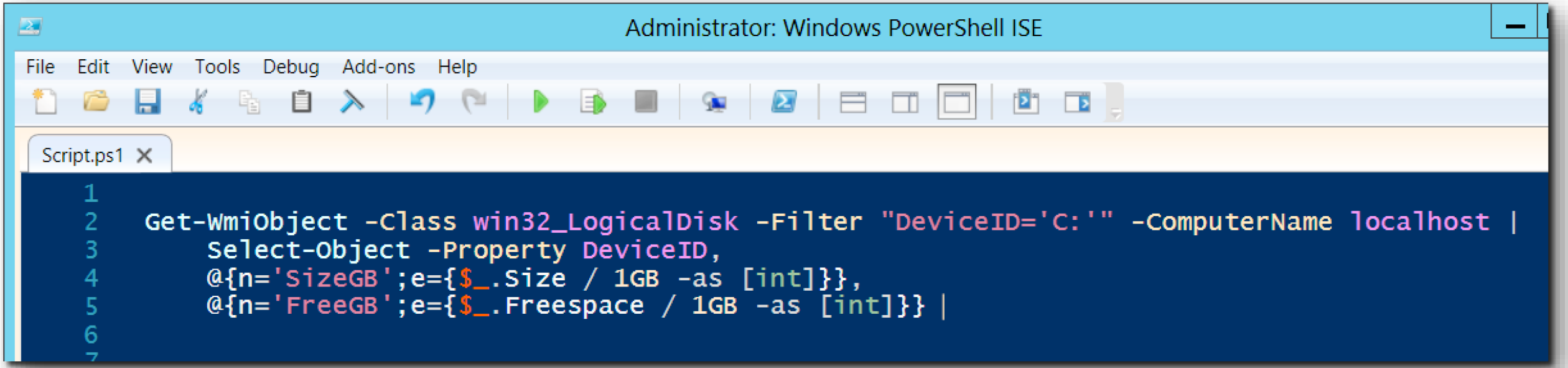
You Call This Scripting?

- 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**

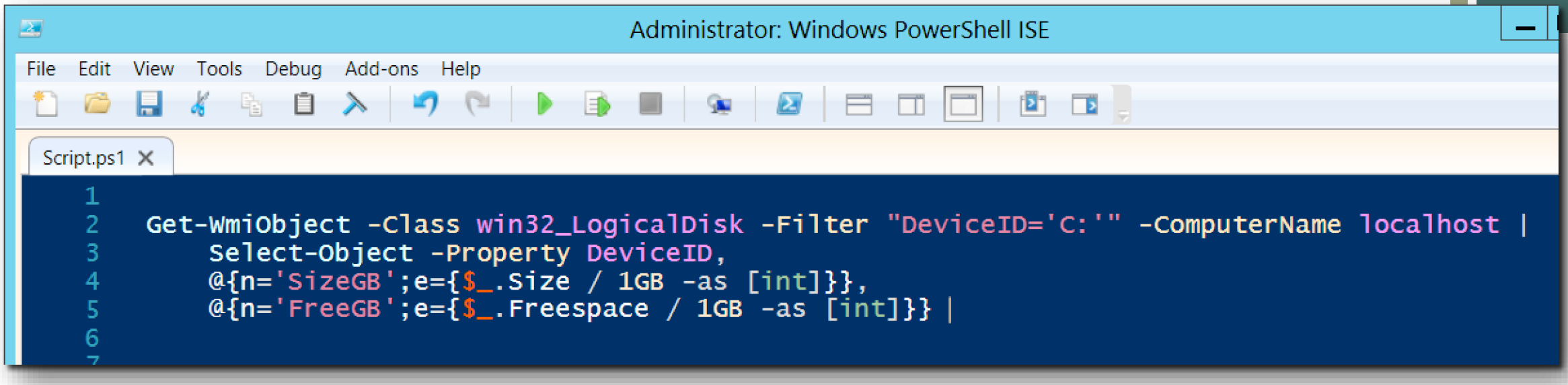

param(
[string]$computerName,
[string]$serviceN
)
Get-WmiObject -Class Win32_S e `
-Filter "name='$s ceName'" `
-ComputerName $computerName
rameter
Comma separates the two parameters
Backtick!

The new ISE

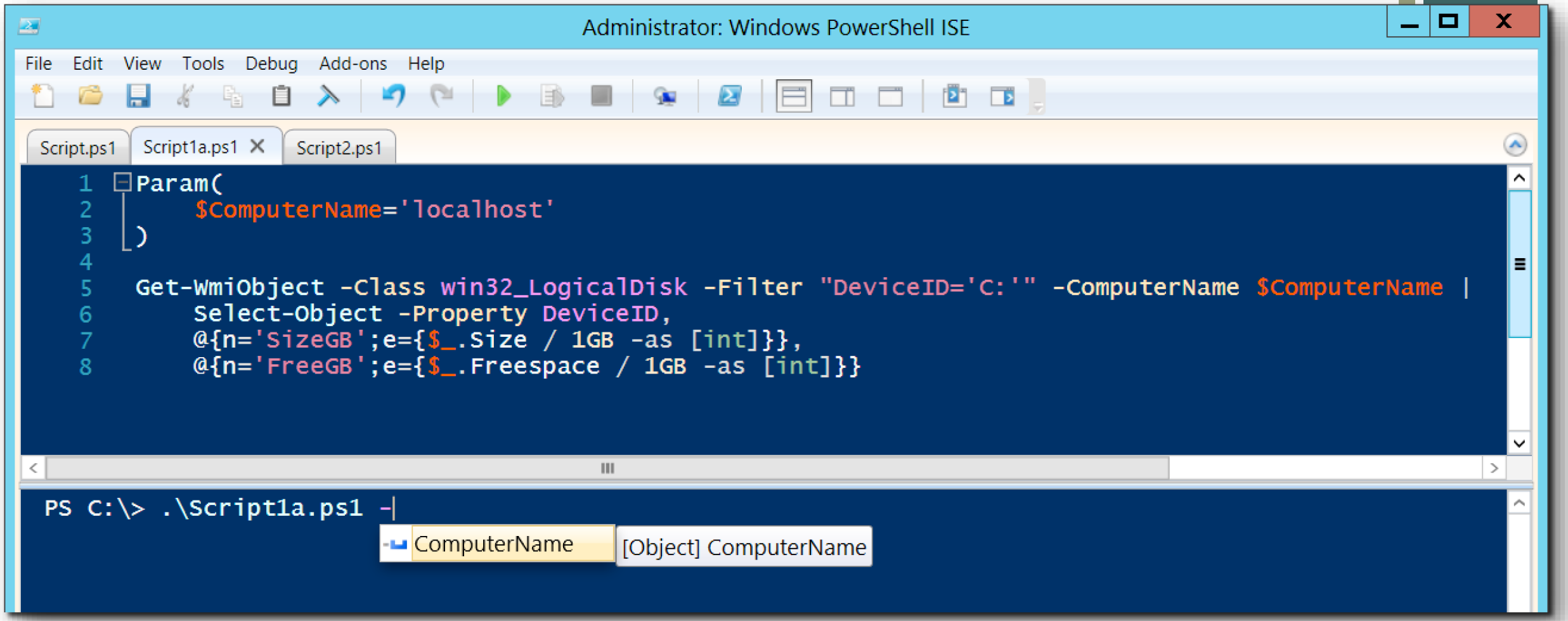

Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
PS C:\> Get-WmiObject -Class win32_LogicalDisk -Filter "DeviceID='C:'" -ComputerName localhost


Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
Script.ps1
Get-WmiObject -Class win32_LogicalDisk -Filter "DeviceID='C:'" -ComputerName localhost |
Select-Object -Property DeviceID,
@{n='SizeGB';e={$_.Size / 1GB -as [int]}},
@{n='FreeGB';e={$_.Freespace / 1GB -as [int]}} |

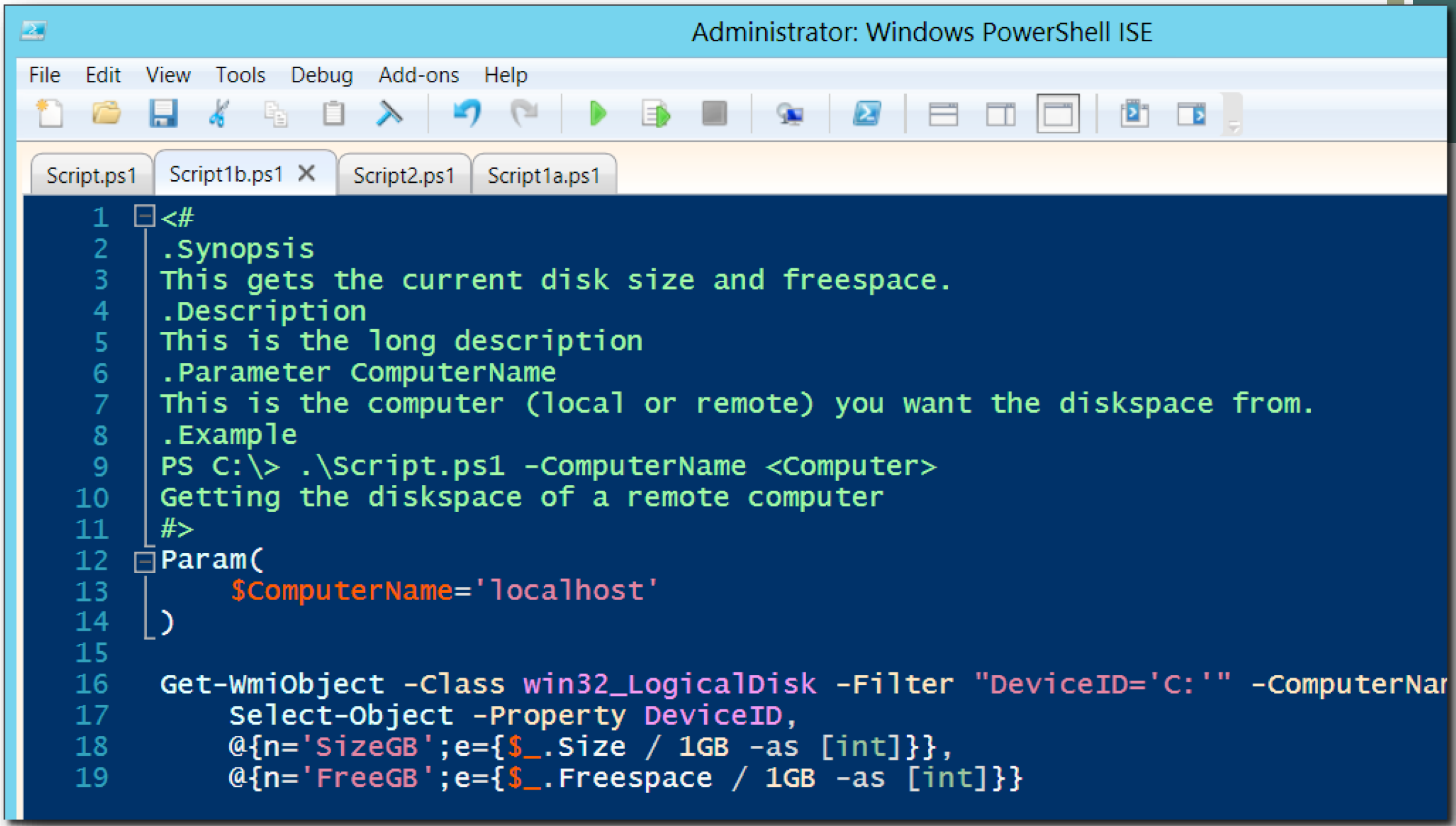
Making commands repeatable



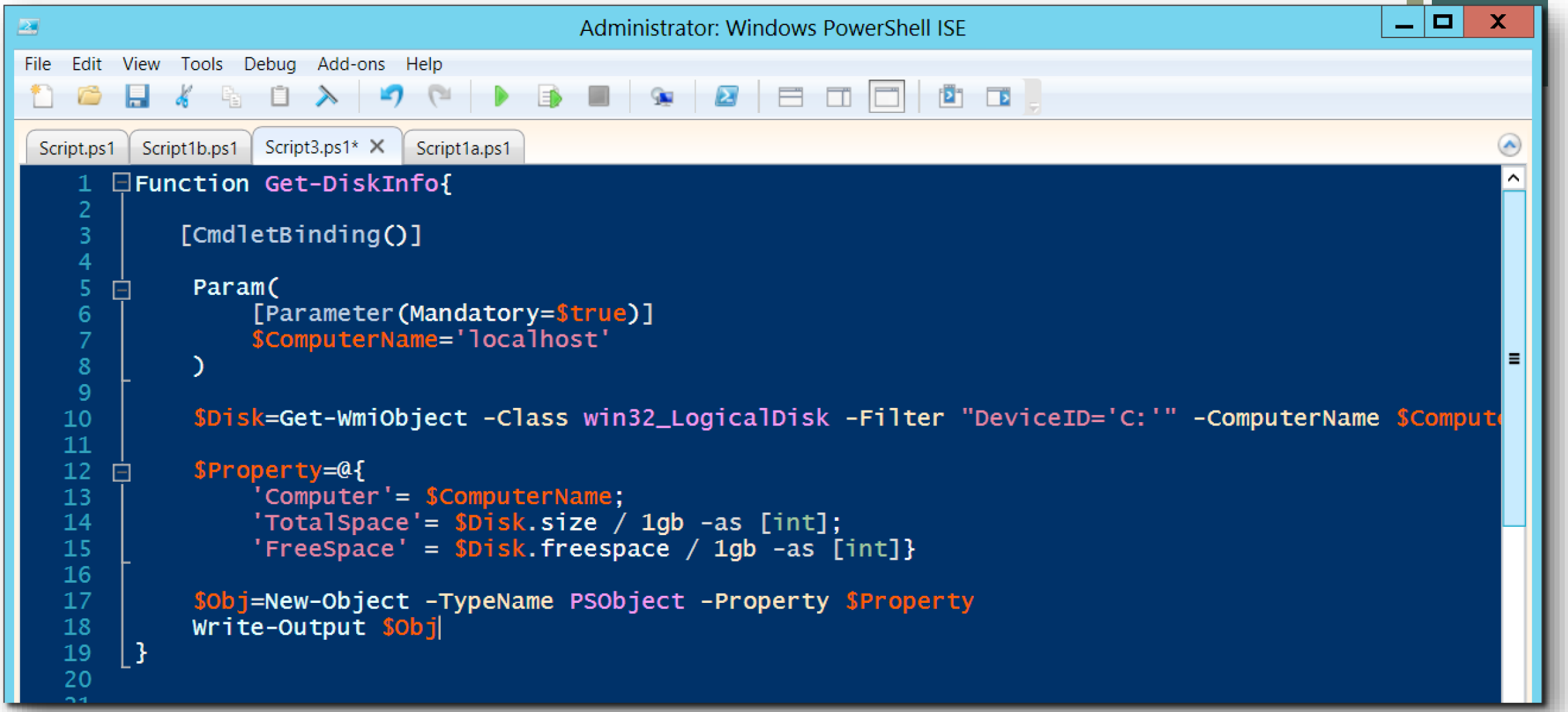
Adding parameters to your script



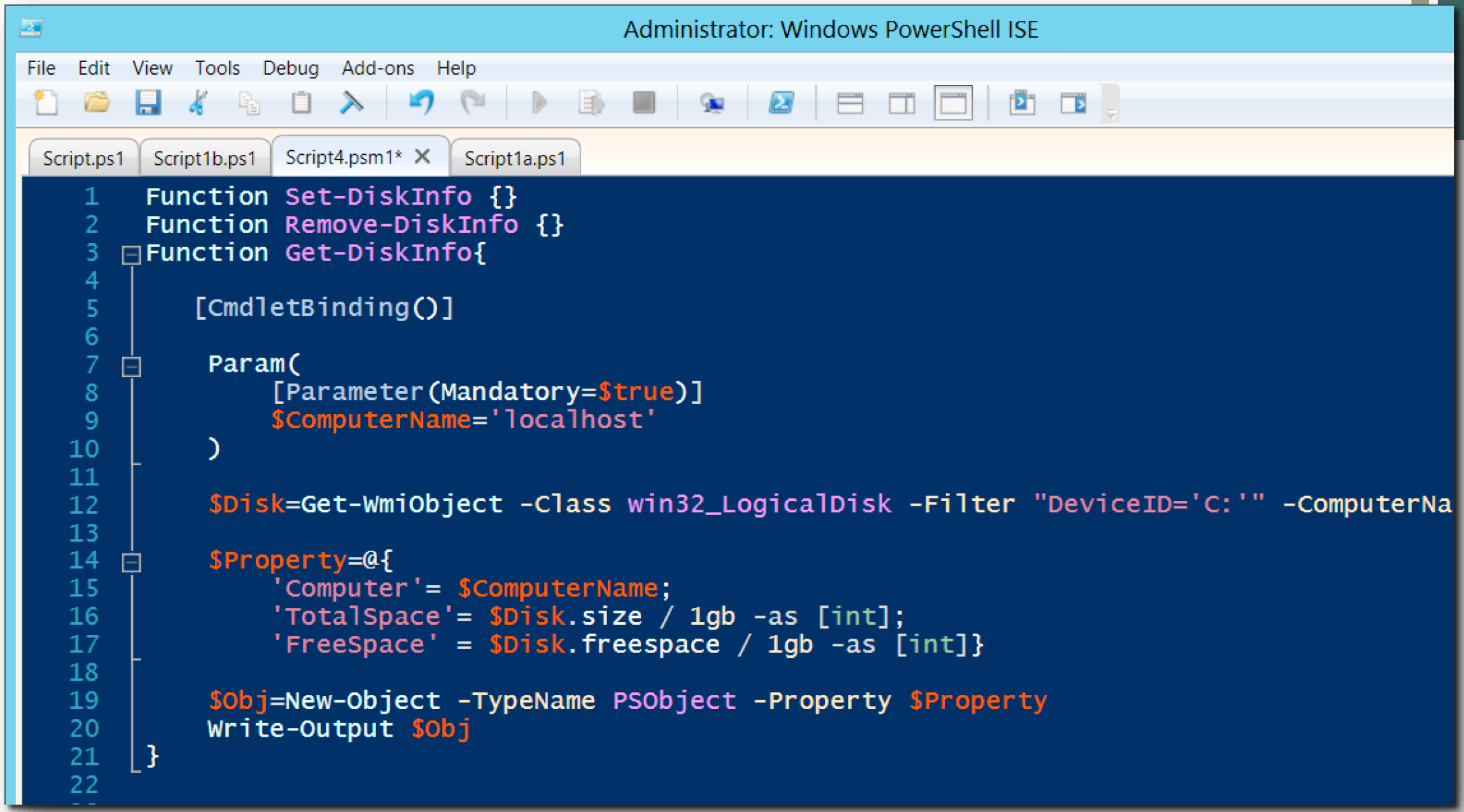
Documenting your script



Turning your script into a tool for others



Storing your tools in a module

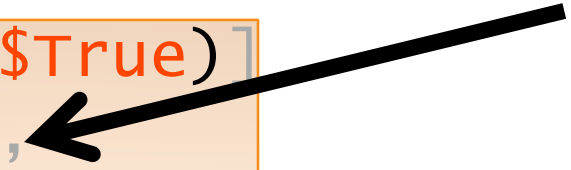


Module 22

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**

```powershell
[CmdletBinding()]
param(
    [Parameter(Mandatory=$True)]
    [string]$computerName,

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

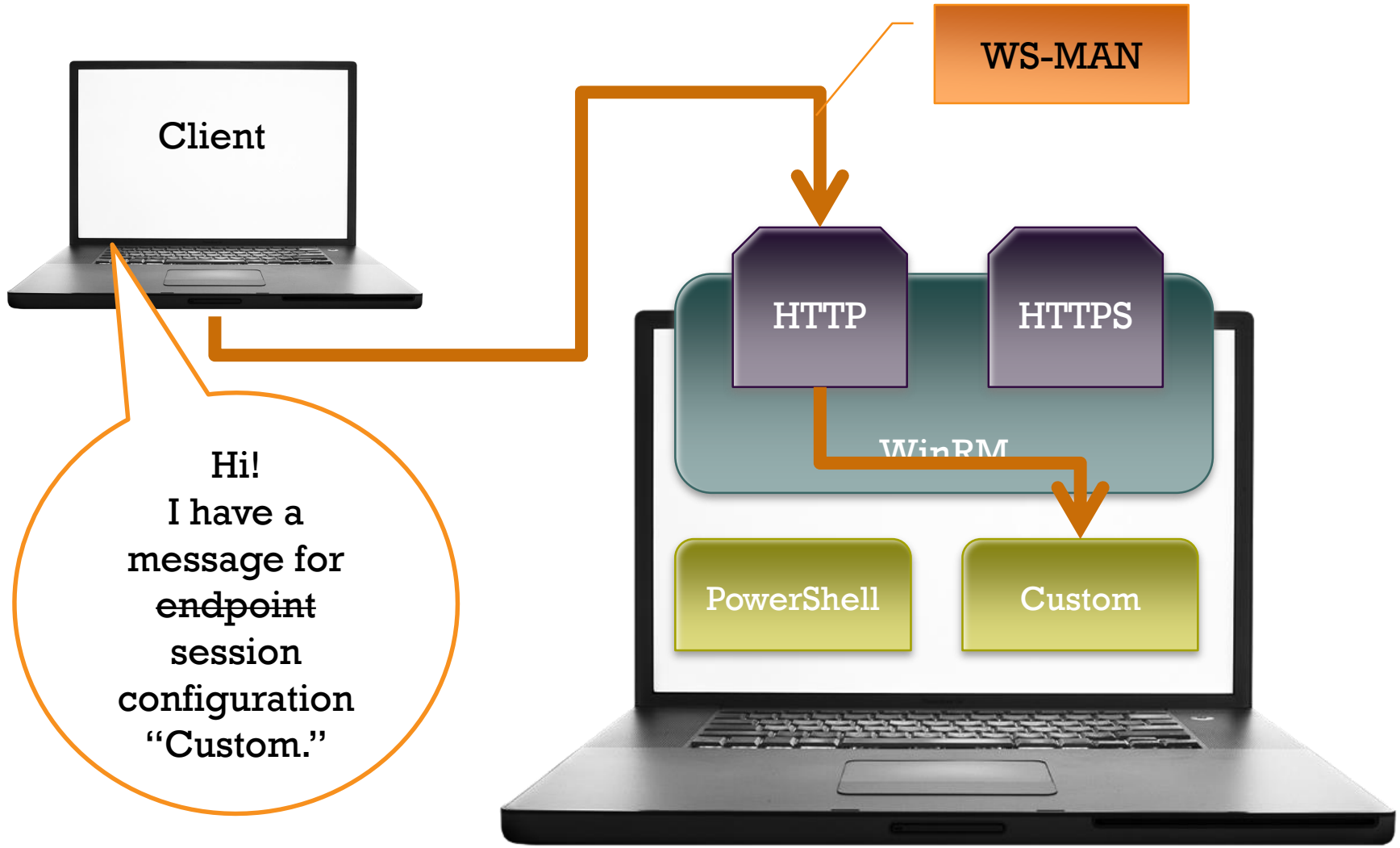


Module 23

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**

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