Microsoft

WorkshopPLUS - Windows PowerShell: Foundation Skills

Microsoft Services

Microsoft

From Script Blocks to Scripts

Microsoft Services

Learning Units covered in this Module

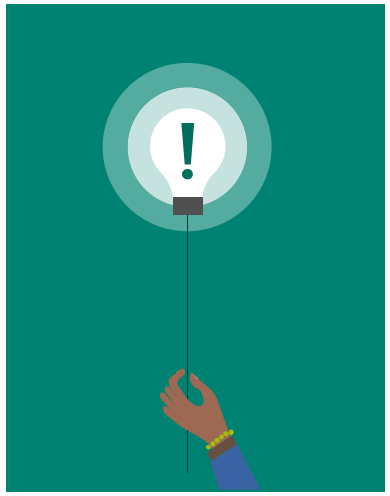
- Using Script Blocks
- Providers
- Scripts

Using Script Blocks

Objectives

After completing Using Script Blocks, you will be able to:

- Work with script blocks
- Work with functions
- Work with remoting



What is a Script Block?

What is a Script Block?

- A statement list in braces “{ }”
- Simplifies reuse of code for commands with script block parameters
- Can accept parameter values and return output
- Used by Cmdlets, Functions, Workflows and Desired State Configuration

Script Block - Examples

```
{<statement list>}
```

```
{
param ($parameter1,$parameterN)
<statement list>
}
```

```
PS C:\> $scriptblock = { param($test) write-host $test}
PS C:\> &$scriptblock "2"
2
```

Cmdlet with Script Block Parameter Argument

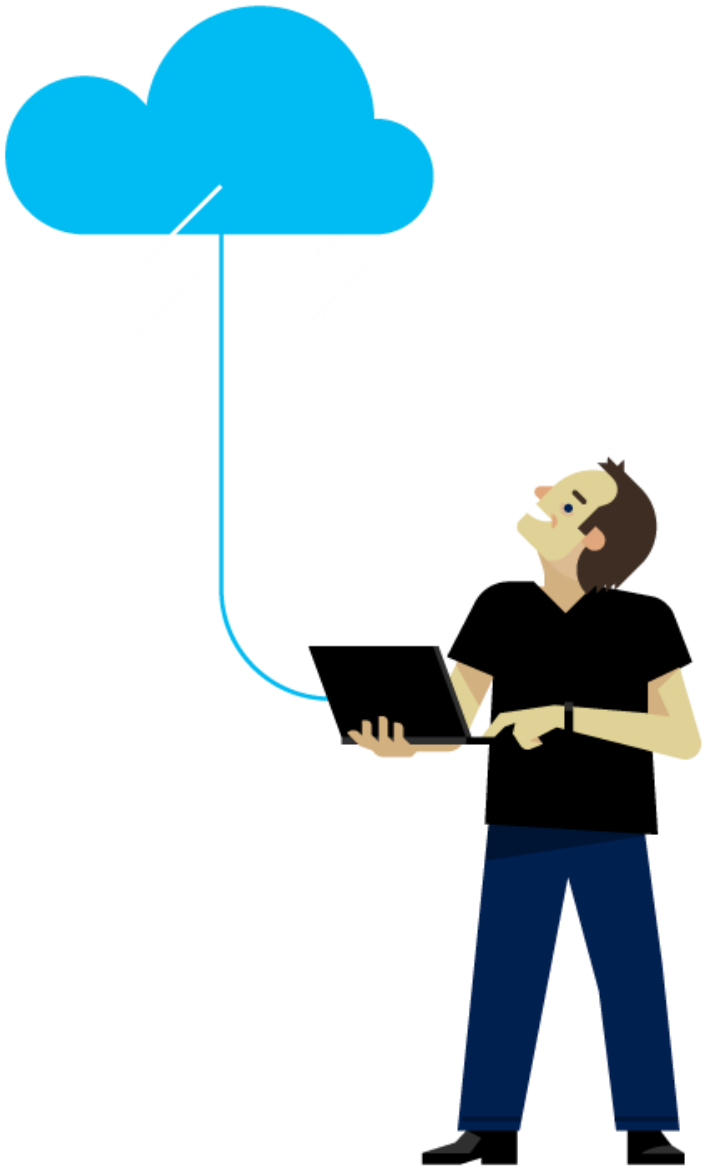
```
PS C:\> Invoke-Command -ScriptBlock {Get-Process} -ComputerName
>> DC, MS, WIN10

 Handles  NPM(K)  PM(K)  WS(K)  VM(M)   CPU(s)   Id    ProcessName
-------  ------  -----  -----  -----   ------   --    -----------
   4848      55  48316  64252    237 3,077.20  1840   CcmExec
     76       8   1948   7180     60     3.55  9356   conhost
    386      23   7768  18512    286     0.16 14092   csrss
...
```

```
PS C:\> Measure-Command -Expression {Get-Process}

Days              : 0
Minutes           : 0
Seconds           : 2
Milliseconds      : 933
Ticks             : 29332816
...
```

Demonstration

Script Blocks

Questions?



Functions Introduction

What is a Function?

- Named Script Block
- Reusable block of PowerShell code
- Reduces size of code and increases reliability
- Can accept parameter values and return output
- Advanced Functions behave like Cmdlets
- Can be created with help topics that can be used with Get-Help (like cmdlets)

What Does a Function Look Like?

1. Function Keyword
2. Function Name
3. Matching Open /Close Curly Braces
4. Statement List
5. Use Function

```
1 function 2 Do-Something
3 {
    4 Write-Host “Do Something”
3 }

5 Do-Something
```

Creating a Utility Function

A series of commands can be contained in a function

```
PS C:\> Get-Service -Name spooler -RequiredServices -ComputerName DC
```

```
function Get-ServiceInfo
{
    Get-Service -Name spooler -RequiredServices -ComputerName DC
}
```

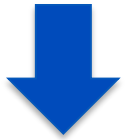
Run the function

```
PS C:\> Get-ServiceInfo
```

Parameters in a function

- Defined using param statement
- Used just like variables
- Passed in using Dash notation
- Can have advanced attributes

```
function Get-ServiceInfo
{
    Param ($svc, $computer)
    Get-Service -Name $svc -RequiredServices -ComputerName $computer
}
```

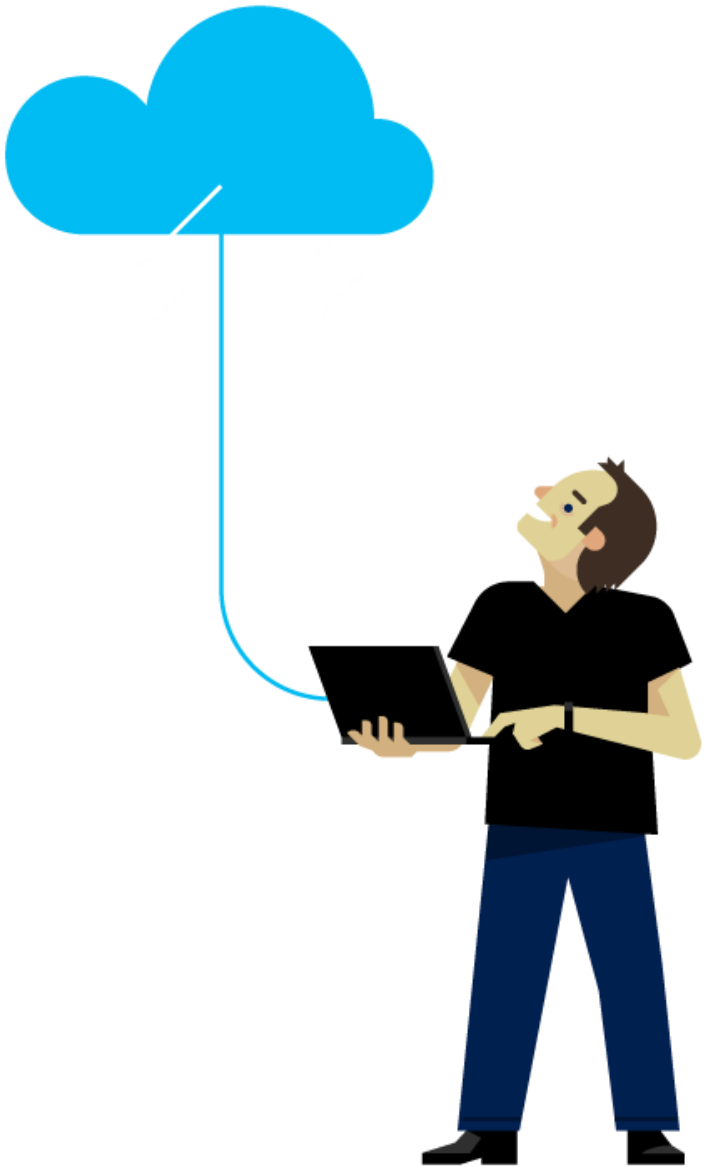


```
PS C:\> Get-ServiceInfo -svc spooler -computer localhost
```

Demonstration

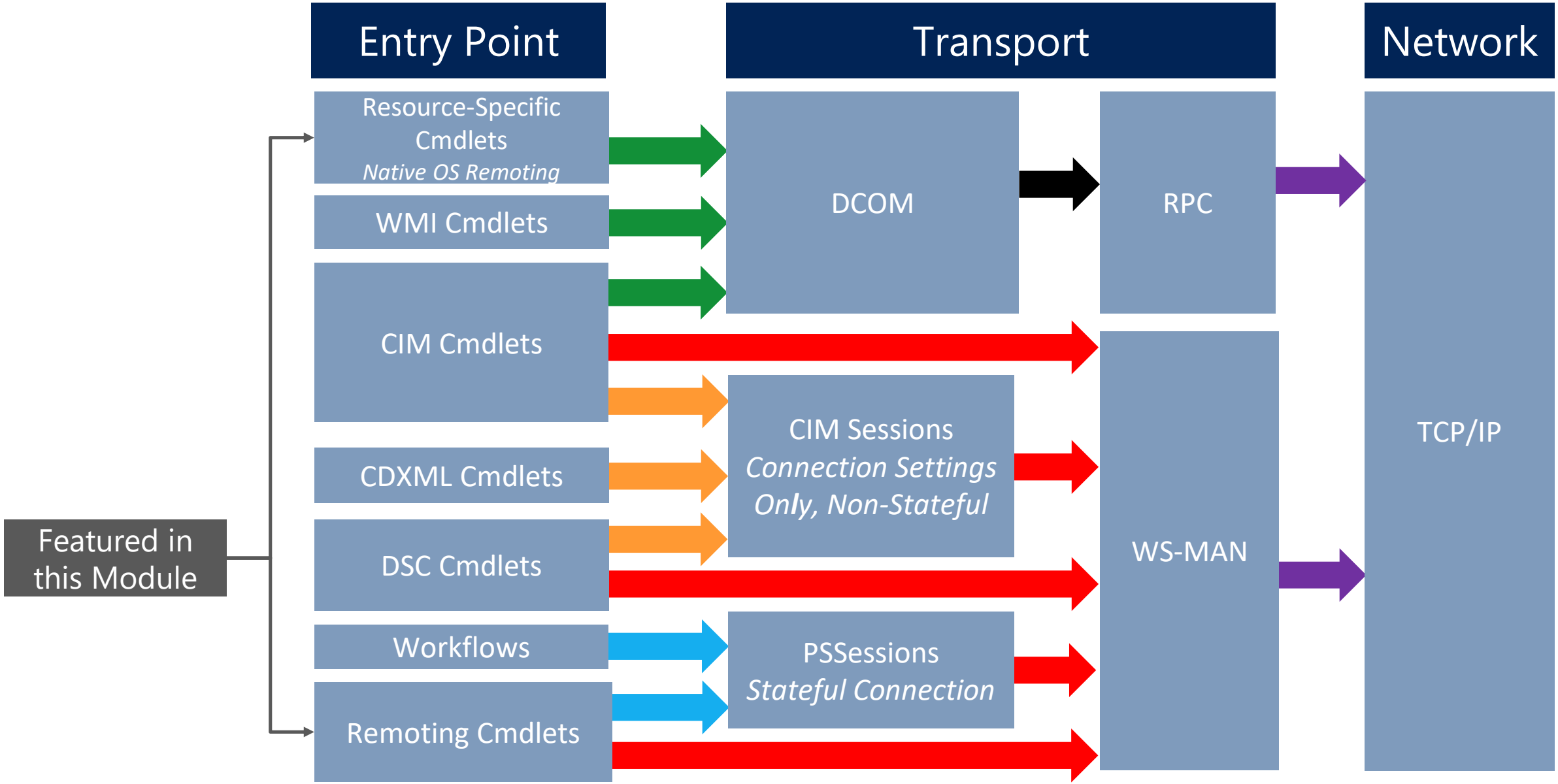
Functions 101

Questions?



Introduction to PowerShell Remoting

Various PowerShell Remote Administration Techniques



Native OS Remoting (-ComputerName Parameter)

- Typically Windows resource or action specific cmdlets
- Use built-in Windows services
- Target machines do not need PowerShell remoting

Examples:

```
PS C:\>  Get-Command -ParameterName ComputerName
...
```

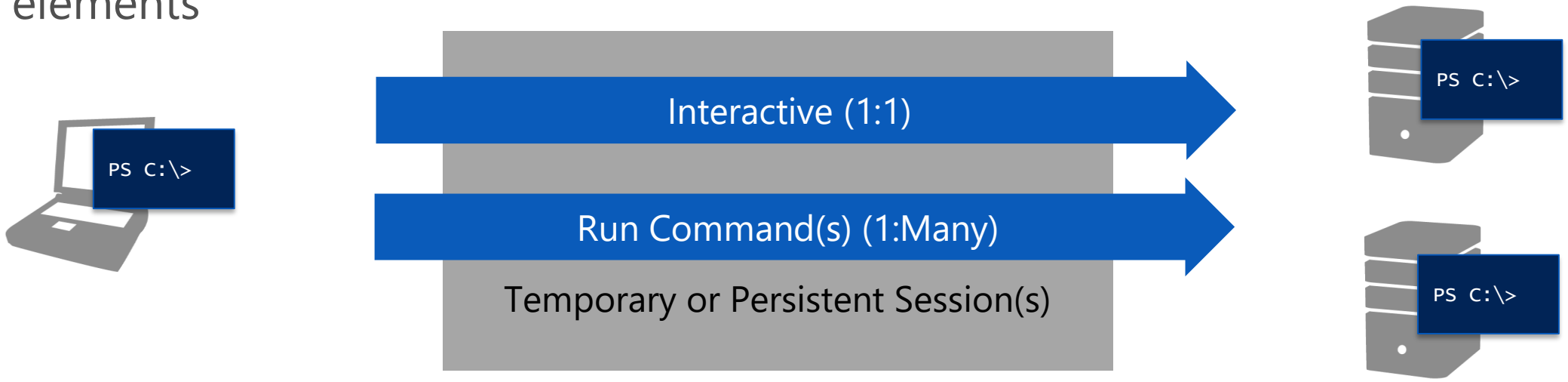
```
Get-Counter; Get-EventLog; Get-HotFix; Restart-Computer; Get-Process;
Get-Service; Stop-Computer; Test-Connection
```

What is PowerShell Remoting?

Introduced in Windows PowerShell 2.0 and enhanced in later versions

Windows PowerShell feature

- Interactively run command(s) with a single remote computer
- Run scriptblock or script on one or more remote computers
- Temporary or Persistent Sessions (connections)
- Destination can be restricted by limiting allowed commands and language elements



Requirements

Local and Remote Computers:

- PowerShell 2.0 or later (feature enhancements with newer versions)

Remoting must be enabled:

- Enabled by default on Windows Server 2012 Operating Systems (OS) and later
- Disabled by default on all Client and earlier Server OS's

Remote User Permissions:

- Must be a member of the Local Administrator group on the remote computer(s) (by default)

Enabling Remoting

- Use local command or Group Policy

Start PowerShell with the "Run as administrator" option

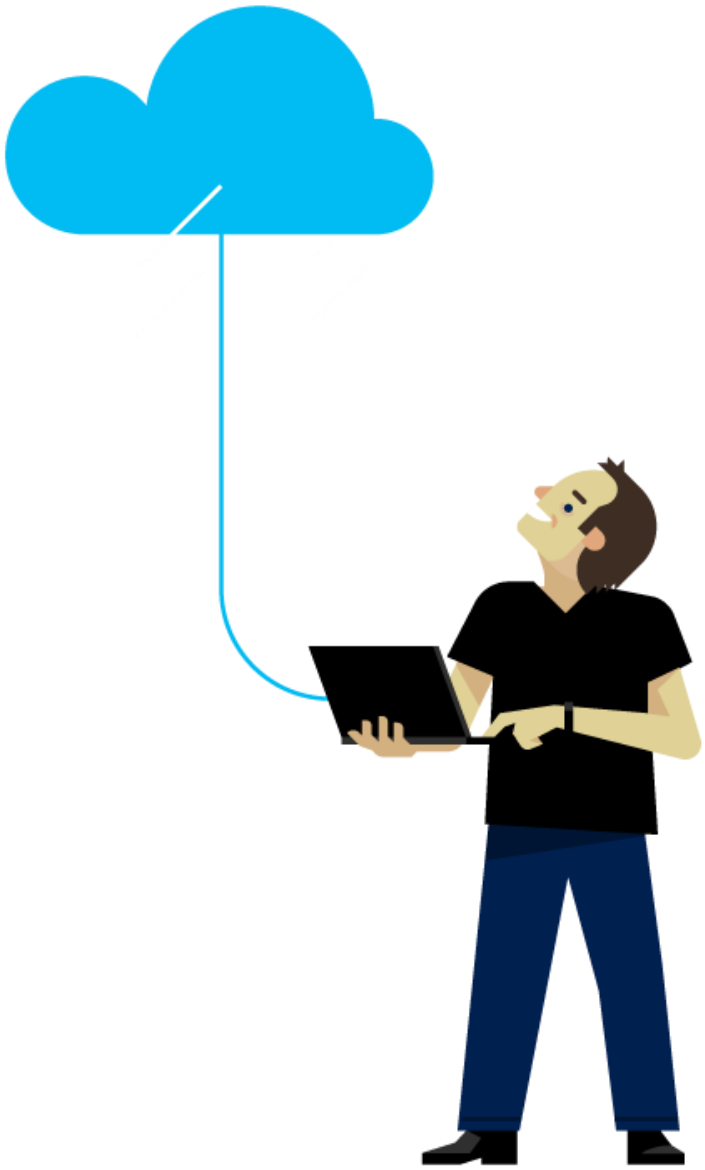
```
PS C:\> Enable-PSRemoting
```

Use group policy for bulk configuration

Demonstration

PowerShell Native OS Remoting

Questions?



Using PowerShell Remoting

Interactive Session

```
PS C:\> Enter-PSSession -ComputerName DC

[DC] PS C:\>

[DC] PS C:\> Hostname
DC

[DC] PS C:\> Exit-PSSession

PS C:\>
```

Remote Computer

Ending a session

Local Computer

Invoke a Command

```
PS C:\> Invoke-Command -ComputerName DC
>> -ScriptBlock {Get-Culture}

LCID      Name    DisplayName                 SComputerName
----      ----    -----------                 -------------
1033      en-US   English (United States)     DC
```

Invoke a Command (1:Many)

```
PS C:\> Invoke-Command -ComputerName DC, MS
>> -ScriptBlock {Get-Culture}

LCID        Name   DisplayName                PSComputerName
----        ----   -----------                ---------------
1033        en-US  English (United States)    DC
1033        en-US  English (United States)    MS
```

Use Alternate Credential

```
PS C:\> Invoke-Command -ComputerName DC -Credential
>> contoso\administrator -ScriptBlock {Get-Culture}

LCID      Name    DisplayName                PSComputerName
----      ----    -----------                --------------
1033      en-US   English (United States)    DC
```

Persistent Session (Repeat Use)

Step 1: Create a persistent session

```
PS C:\> New-PSSession -ComputerName DC -OutVariable ps

Id Name      ComputerName State   ConfigurationName     Availability
-- ----      ------------ -----   -----------------     ------------
1  Session1 DC           Opened Microsoft.PowerShell Available
```

Step 2: Use the session

```
PS C:\> Invoke-Command -Session $ps -ScriptBlock {Get-Culture}

LCID      Name    DisplayName           PSComputerName
----      ----    -----------           --------------
3081      en-AU   English (Australia)   DC
```

Persistent Session (Repeat Use) 1:Many

Step 1: Create persistent sessions

```
PS C:\> New-PSSession -ComputerName DC, MS
  -OutVariable ps

Id  Name       ComputerName  State   ConfigurationName      Availability
 -- ----       ------------  -----   -----------------      ------------
  1 Session1   DC            Opened  Microsoft.PowerShell      Available
  2 Session2   MS            Opened  Microsoft.PowerShell      Available
```

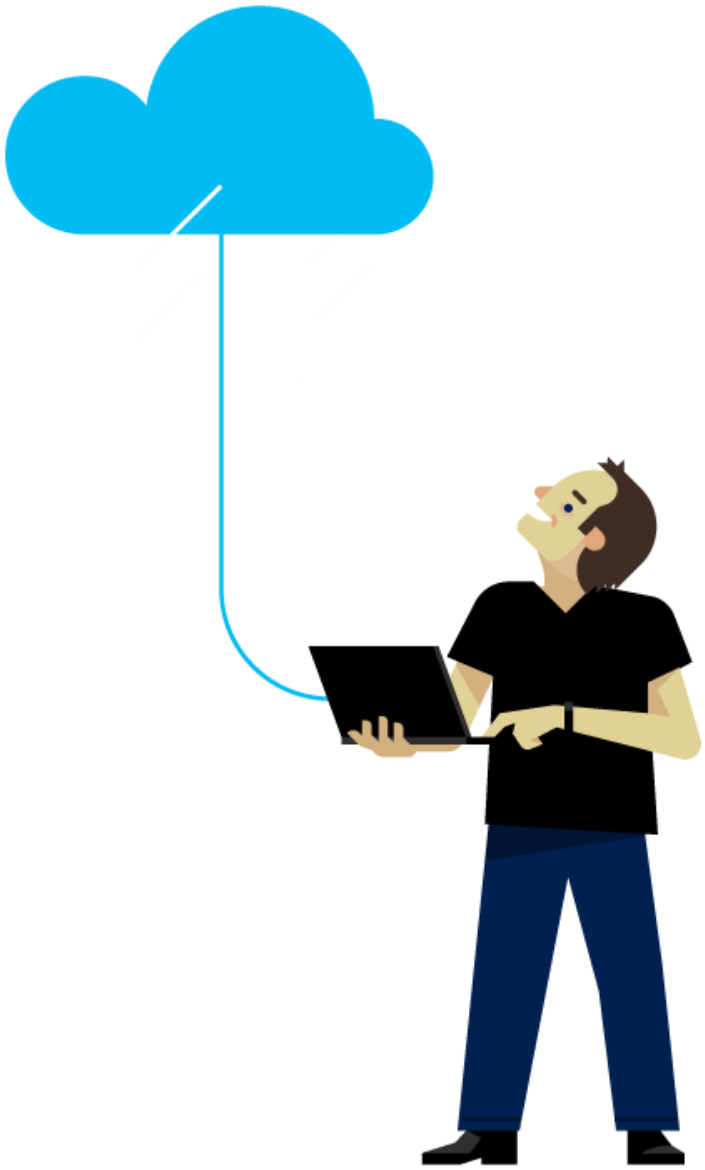
Step 2: Use the sessions

```
PS C:\> Invoke-Command -Session $ps -ScriptBlock {Get-Culture}

LCID           Name    DisplayName                PSComputerName
----           ----    -----------                --------------
1033           en-US   English (United States)    DC
1033           en-US   English (United States)    MS
```

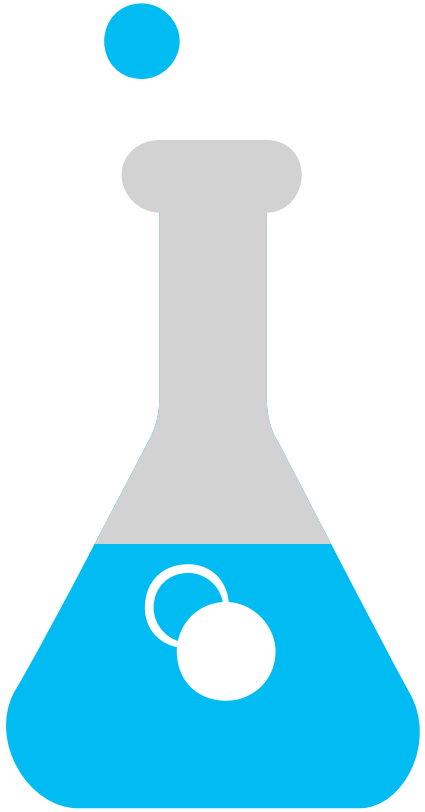
Demonstration

PowerShell Remoting

Questions?



Using Script Blocks



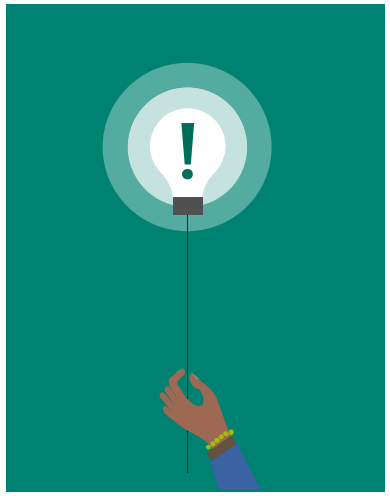
LAB

Providers

Objectives

After completing Providers, you will be able to:

- Work with PSproviders and PSdrives



What are Providers?

What are PowerShell Providers?

- Define the logic to access, navigate and edit a data store
- Functionally resemble a file system hierarchy
- Common interface to different data stores

Where to Get Providers

- PowerShell ships with built-in providers
- Providers can be imported via a module
- Examples of well-known imported providers:
  - Active Directory
  - SQL Server

What is a Powershell Drive?

- A specific entry-point to a data store surfaced by a provider
- Allows any data store to be exposed like a file system, as if it were a mounted drive
- Classic file system volume naming convention <Drive Name>:
- Consistent drive interaction with common Cmdlets

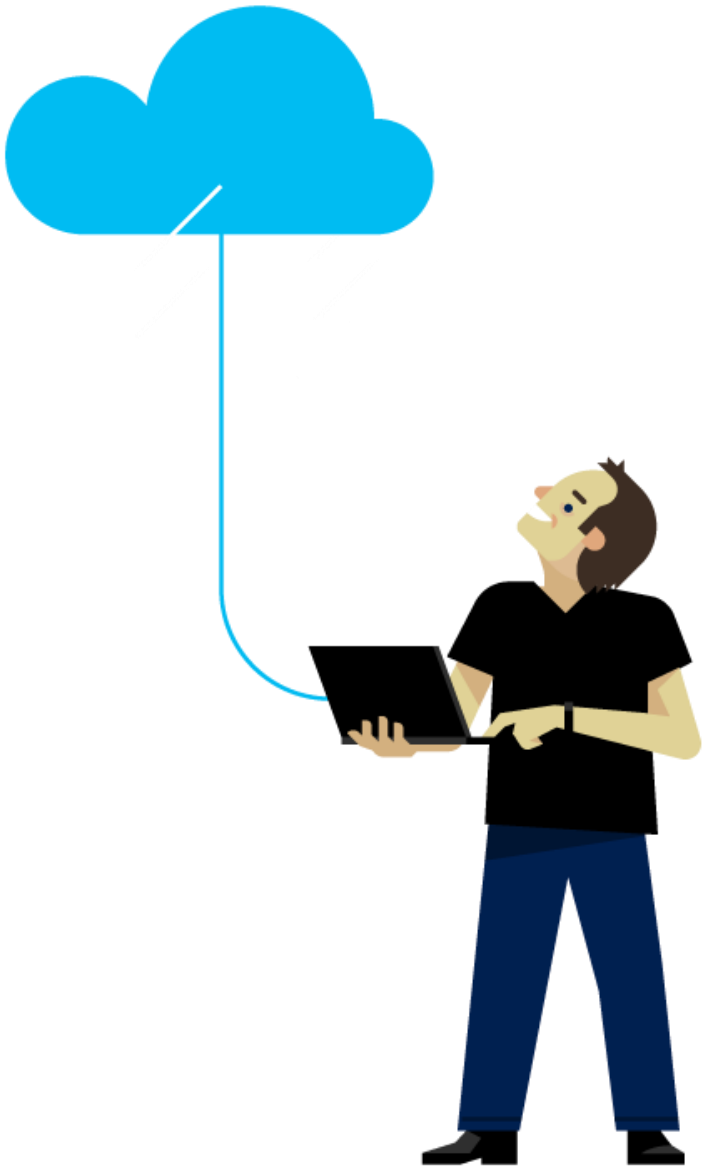
Built-in Providers

Provider	Drive	Data Store
Alias	Alias:	Windows PowerShell aliases
Certificate	Cert:	x509 certificates for digital signatures
Environment	Env:	Windows environment variables
FileSystem	C:, D:, etc.	File system drives, directories, and files
Function	Function:	Windows PowerShell functions
Registry	HKLM:, HKCU:	Windows registry
Variable	Variable:	Windows PowerShell variables
WSMan	WSMan:	WS-Management configuration information

Demonstration

PowerShell Providers and Drives

Questions?



Drive Cmdlets

PowerShell Drive Cmdlets

- Drive cmdlets can be used to create and remove access points into datastores managed by PSproviders

Get-PSDrive

Returns drives in current session

```
PS C:\> Get-PSDrive
Name        Used (GB) Free (GB) Provider      Root
----        ---------- ---------- ---------      ----
Alias                            Alias
C           206.81     16.00     FileSystem     C:\
Cert                             Certificate    \
Env                              Environment
Function                         Function
HKCU                             Registry
HKEY_CURRENT_USER
HKLM                             Registry
HKEY_LOCAL_MACHINE
……
```

New-PSDrive Remove-PSDrive

Creates a user-defined drive

```
PS C:\> New-PSDrive -Name HKCR -PSProvider Registry
 -Root HKEY_CLASSES_ROOT
```

Creates a user-defined drive (use only single letter name with persist)

```
PS C:\>  New-PSDrive -Name H -PSProvider FileSystem
 -Root \\MS\HomeShare -Persist
 -Credential (Get-Credential Contoso\DanPark)
```

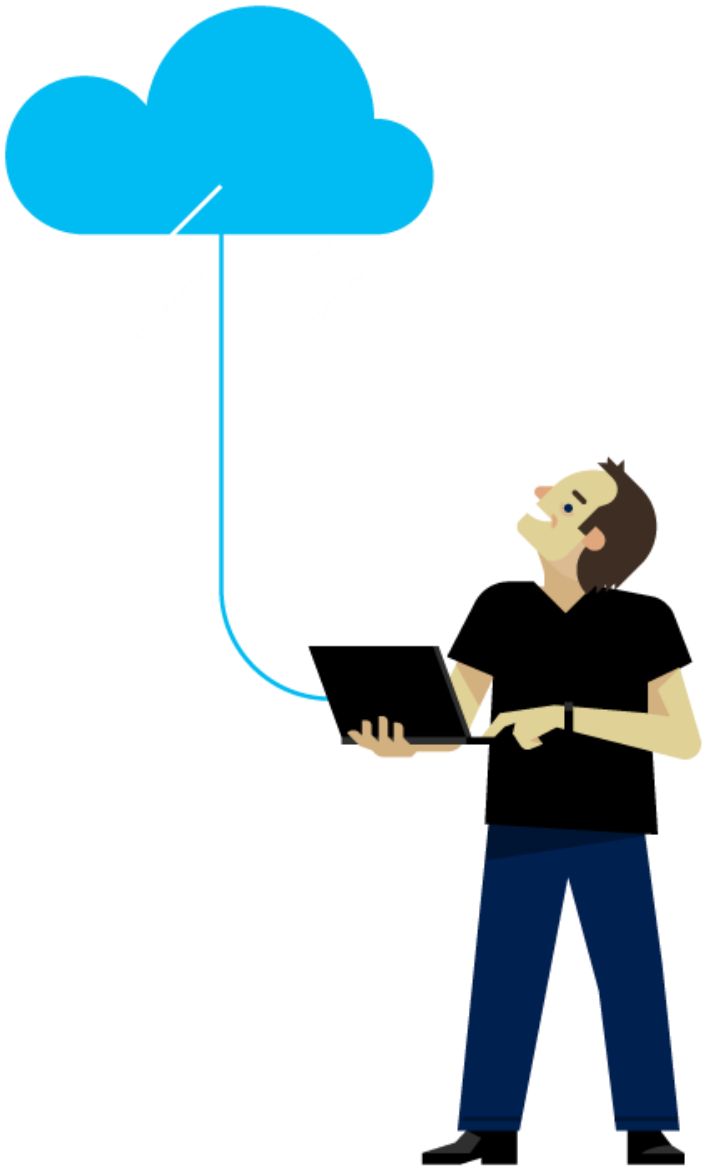
Removes a PowerShell drive (user or built-in)

```
PS C:\> Remove-PSDrive -Name HKCR
```

Demonstration

Creating PS Drives

Questions?



Item Cmdlets

Item Cmdlets

- Item cmdlets are used to read and manipulate path objects
- Item cmdlets cannot be used to manipulate sub properties like registry keys

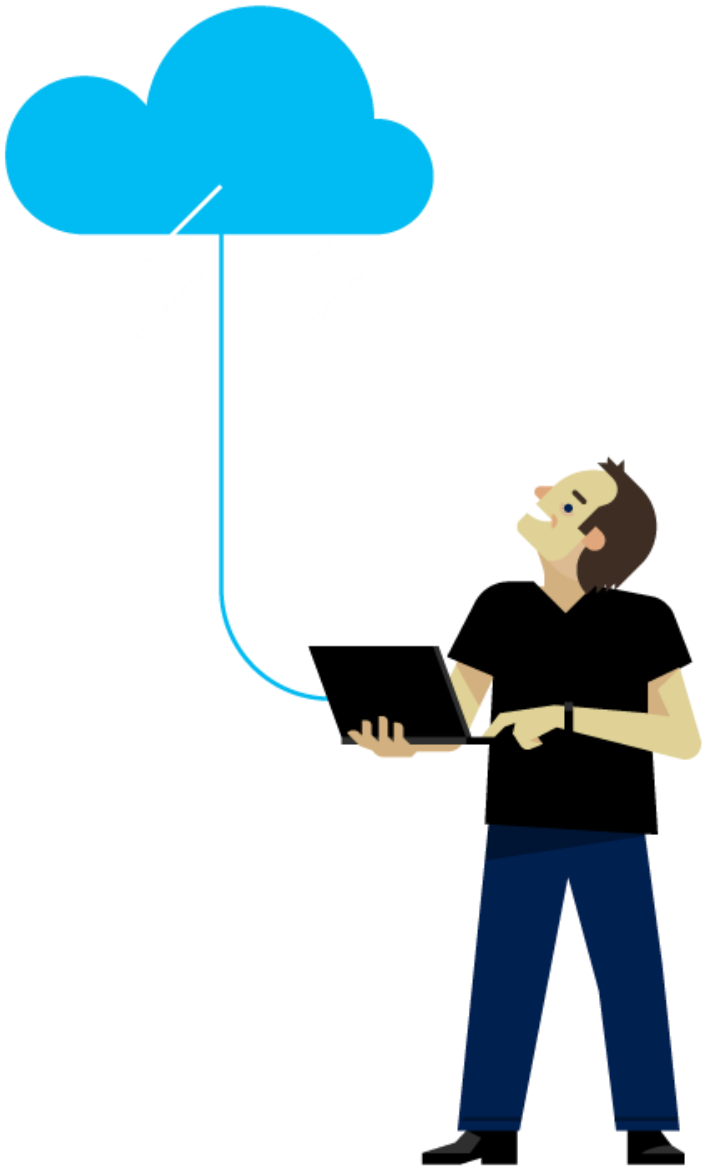
Item Cmdlets

Name	Example
Get-Item	`PS C:\> Get-Item c:\Windows`
Get-ChildItem	`PS C:\> Get-ChildItem -Path C:\Windows`
Copy-Item	`PS C:\> Copy-Item c:\Logs -Destination d:\Logs -Recurse`
Move-Item	`PS C:\> Move-Item HKLM:\software\A\* HKLM:\software`
Clear-Item	`PS C:\> Clear-Item HKLM:\Software\MyCompany -Confirm`

Item Cmdlets

Name	Example
Remove-Item	`PS C:\> Get-ChildItem * -Include *.mp3 -Recurse | Remove-Item`
Set-Item	`PS C:\> Set-Item -Path env:UserRole -Value Administrator`
Invoke-Item	`PS C:\> Invoke-Item “d:\Documents\Users.xls"`
New-Item	`PS C:\> New-Item -ItemType file -Path “d:\test.txt", "c:\Logs\test.log"`
Rename-Item	`PS C:\> Rename-Item HKLM:\Software\Company -NewName Marketing`

Questions?



ItemProperty Cmdlets

ItemProperty Cmdlets

- Item property cmdlets can read and manipulate sub properties at a path object location

ItemProperty Cmdlets

Name	Example
Get-ItemProperty	`PS C:\> Get-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\PowerShell\1`
Copy-ItemProperty	`PS C:\> Copy-ItemProperty -Path MyApp -Destination HKLM:\Software\MyAppRev2 -Name MyProperty`
Move-ItemProperty	`PS C:\> Move-ItemProperty HKLM:\Software\MyCompany\MyApp -Name Version -Destination HKLM:\Software\MyCompany\NewApp`
Clear-ItemProperty	`PS C:\> Clear-ItemProperty -Path HKLM:\Software\MyCompany\MyApp -Name Options`

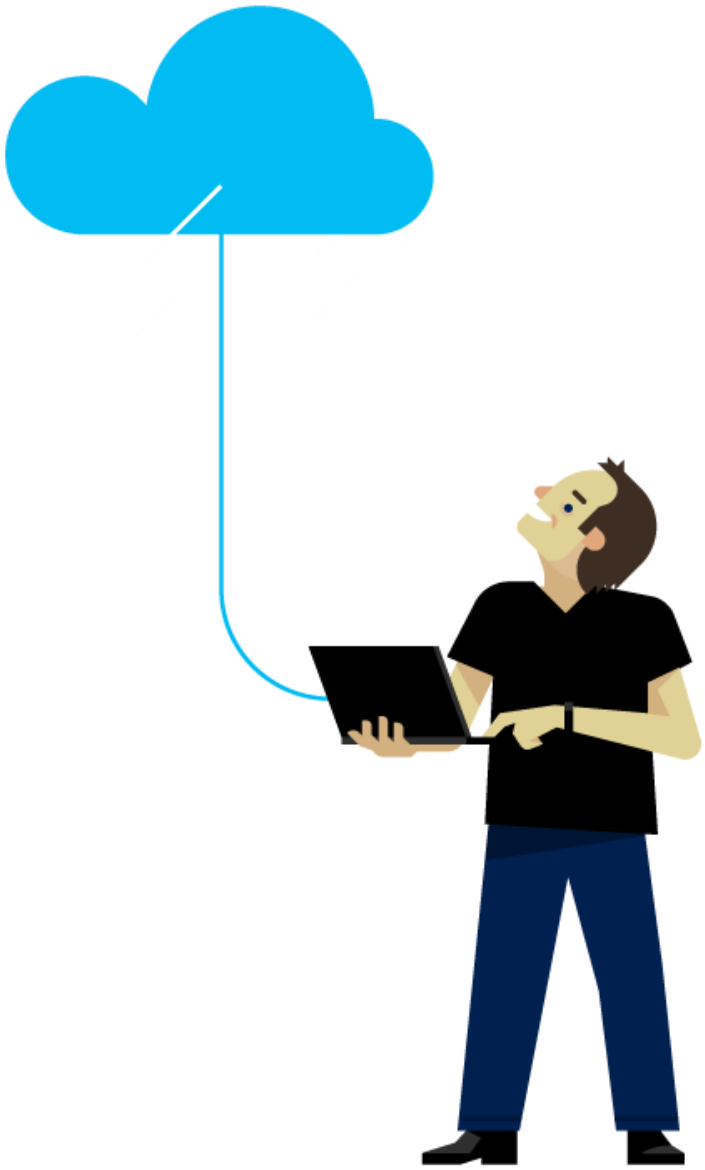
ItemProperty Cmdlets

Name	Example
Remove-ItemProperty	`PS C:\> Remove-ItemProperty -Path HKLM:\Software\MyApp -Name MyProperty`
Set-ItemProperty	`PS C:\> Get-ChildItem weekly.log | Set-ItemProperty -Name IsReadOnly -Value $true`
New-ItemProperty	`PS C:\> Get-Item -Path HKLM:\Software\MyCompany | New-ItemProperty -Name NoOfLocations -Value 3`
Rename-ItemProperty	`PS C:\> Rename-ItemProperty -Path HKLM:\Software\MyApp -Name config -NewName oldconfig`

Demonstration

Item and Itemproperty Cmdlets

Questions?



Content Cmdlets

Content Cmdlets

Name	Example

Get-Content	`PS C:\> Get-Content C:\Logs\Log060912.txt -TotalCount 50`
	`PS C:\> Get-Content Env:\CommonProgramFiles`
	`PS C:\> Get-Content Function:\Get-IseSnippet`

Add-Content	`PS C:\> Get-Content test.xml |` `Add-Content final.xml -Force -Encoding UTF8`
Clear-Content	`PS C:\> Clear-Content C:\Windows\Logs\bpa\Reports\*` `-Include 2013* -Exclude 2014*`
Set-Content	`PS C:\> Get-Date | Set-Content C:\Output\date.csv`

Demonstration

Content Cmdlets

Path Cmdlets

Path Cmdlets

- Path cmdlets can be used to manipulate path objects
- Path commands understand how a path object is buildup

Path Cmdlets

Name	Example

Test-Path	`PS C:\> Test-Path $pshome\PowerShell.exe -NewerThan "June 13, 2018“`
	
	`True`

Join-Path	`PS C:\> Join-Path -Path C: -ChildPath Temp -Resolve`
	
	`C:\Temp`

Split-Path	`PS C:\> Split-Path -Path 'C:\Program Files (x86)\Internet Explorer\iexplore.exe' -Leaf`
	
	`iexplore.exe`

Path Cmdlets

Name	Example
Convert-Path	`PS C:\> Convert-Path HKLM:\software\Microsoft` `HKEY_LOCAL_MACHINE\Software\Microsoft`

Resolve-Path	`PS C:\> Resolve-Path c:\prog* -Relative`
	`.\C:\Program Files`
	`.\C:\Program Files (x86)`

Location Cmdlets

Name	Example
Get-Location	`PS C:\> Get-Location` `C:\`
Set-Location	`PS C:\> Set-Location -Path HKLM:\SOFTWARE` `PS HKLM:\SOFTWARE>`

Variable Syntax – Access PSDrive Items

Drive	Examples:
Alias:	`PS C:\> $alias:dir` `Get-ChildItem`
Env:	`PS C:\> $Env:windir` `C:\windows`

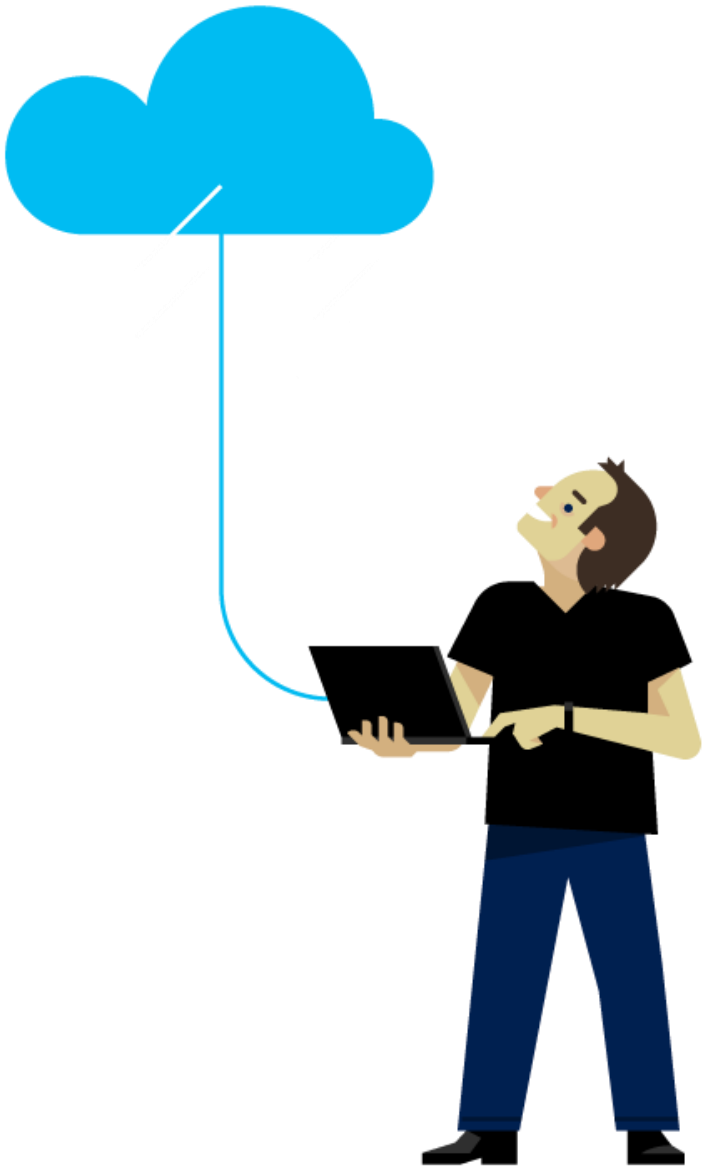
Function:	`PS C:\> $function:more`
	`param([string[]]$paths)`
	`{…}`

Variable:	`PS C:\> $variable:ref`
	`localhost`
	`server1`

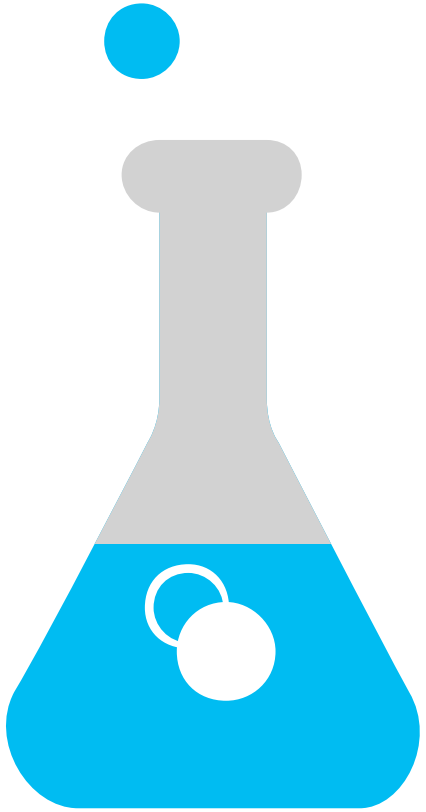
Demonstration

Path Cmdlets

Questions?



Providers



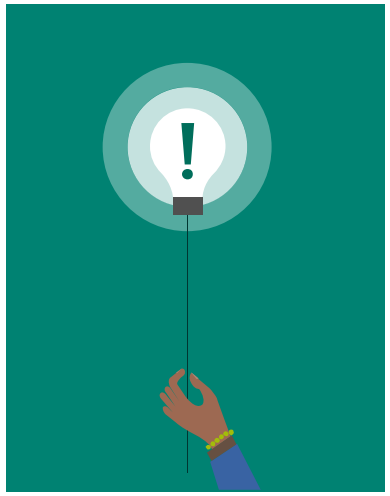
LAB

Scripts

Objectives

After completing Scripts, you will be able to:

- Understand how security policys prevent running scripts
- Understand how to create commenting in scripts and synopsis



What is a Script?

What is a Script?

- Reusable code
- Text file (.ps1) containing one or more PowerShell commands
- Simple ‘code packaging’ for distribution purposes
- Can also:
  - Be digitally signed for security
  - Take parameter values
  - Return values
  - Use the help syntax

Simple Script Example

Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Sample.ps1

```
1  Write-Host "Start of script" -ForegroundColor White -BackgroundColor Green
2  Write-Host "Display the % CPU Time utilization by the ISE" -ForegroundColor White -BackgroundColor Green
3  Get-Counter "\Process(powershell_ise)\% Processor Time"
4
5
6
7
8
9
10
11
12
```

```
PS C:\scripts> C:\scripts\Sample.ps1
Start of script
Display the % CPU Time utilization by the ISE

Timestamp                 CounterSamples
---------                 --------------
6/25/2018 11:22:39 AM     \\WIN10\process(powershell_ise)\% processor time :
                          14.0038797593275
```

Demonstration

PowerShell Scripts

Execution Policies

Execution Policy Levels

Restricted

- Default in 2008R2 and below.
- Scripts cannot be run
- PowerShell interactive-mode only

AllSigned

- Runs a script only if signed
- Signature must be trusted on local machine

RemoteSigned

- Default in 2012R2 and Beyond. (Recommended Minimum)
- Runs all local scripts
- Downloaded scripts must be signed by trusted source

Unrestricted

- All scripts from all sources can be run without signing

Execution Policy Scope

AD Group Policy – Computer

- Affects all users on targeted computer
- Edited through GPO Tools

AD Group Policy – User

- Affects users targeted only
- Edited through GPO Tools

Process

- Console or ISE Command-line Parameter (*c:\> powershell.exe –executionpolicy remotesigned)*
- Affects current PowerShell Host session only
- Lost upon exit of session (i.e. host process)

Registry – User

- Affects current user only
- Stored in HKCU registry subkey

Registry – Computer

- Affects all users on computer
- Stored in HKLM registry subkey (Admin access needed to change)

Highest Priority Wins

Script Execution

- Default Execution Policy (Remote Signed) prevents any scripts from running
- Must be changed to run any scripts
- Execution Policy is saved in the registry, and therefore only needs to be changed once per computer

Determine Which Execution Policy is in Effect

```
PS C:\> Get-ExecutionPolicy -List

Scope            ExecutionPolicy
-----            ---------------
MachinePolicy    Undefined
UserPolicy       Undefined
Process          Undefined
CurrentUser      Undefined
LocalMachine     RemoteSigned
```

Top most takes precedence

Effective Policy

Set Execution Policy - User

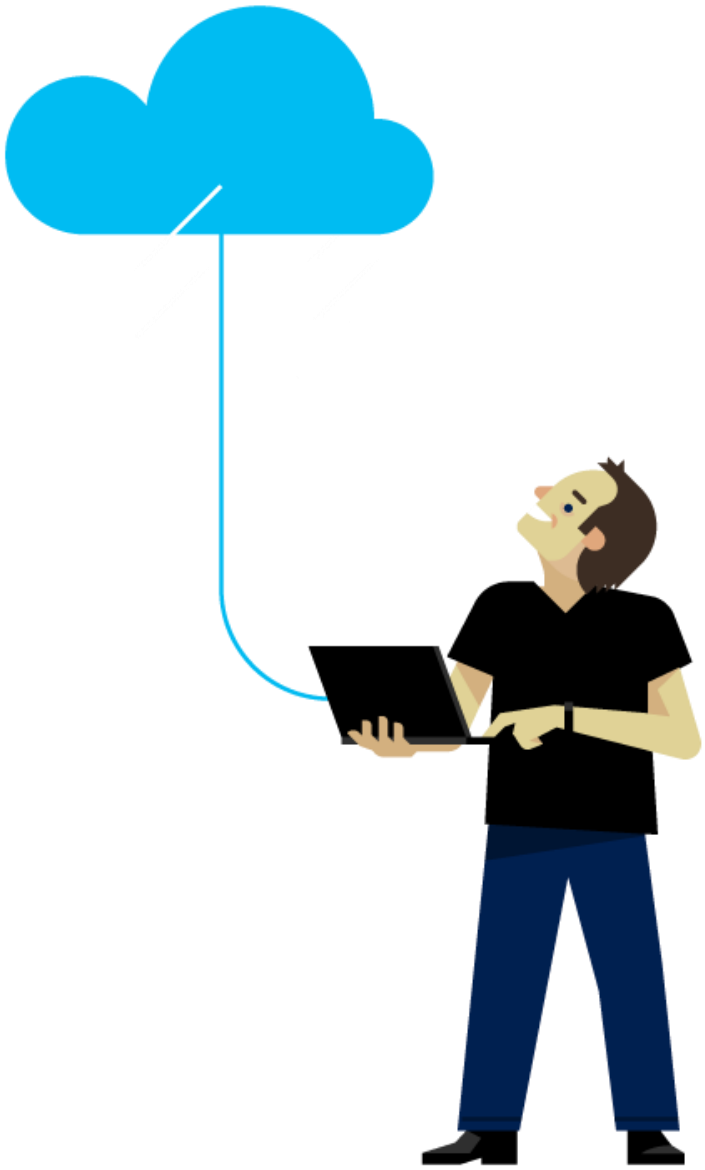
```
Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy
Unrestricted
```

Apply setting to current user only, default is current machine

Demonstration

Execution Policy

Questions?



Launching a script

Running a Script From the Shell

Full path and file name

```
PS C:\> c:\scripts\script.ps1
```

Script in current directory

```
PS C:\Scripts> .\script.ps1
```

Spaces in path (tab completion helps)

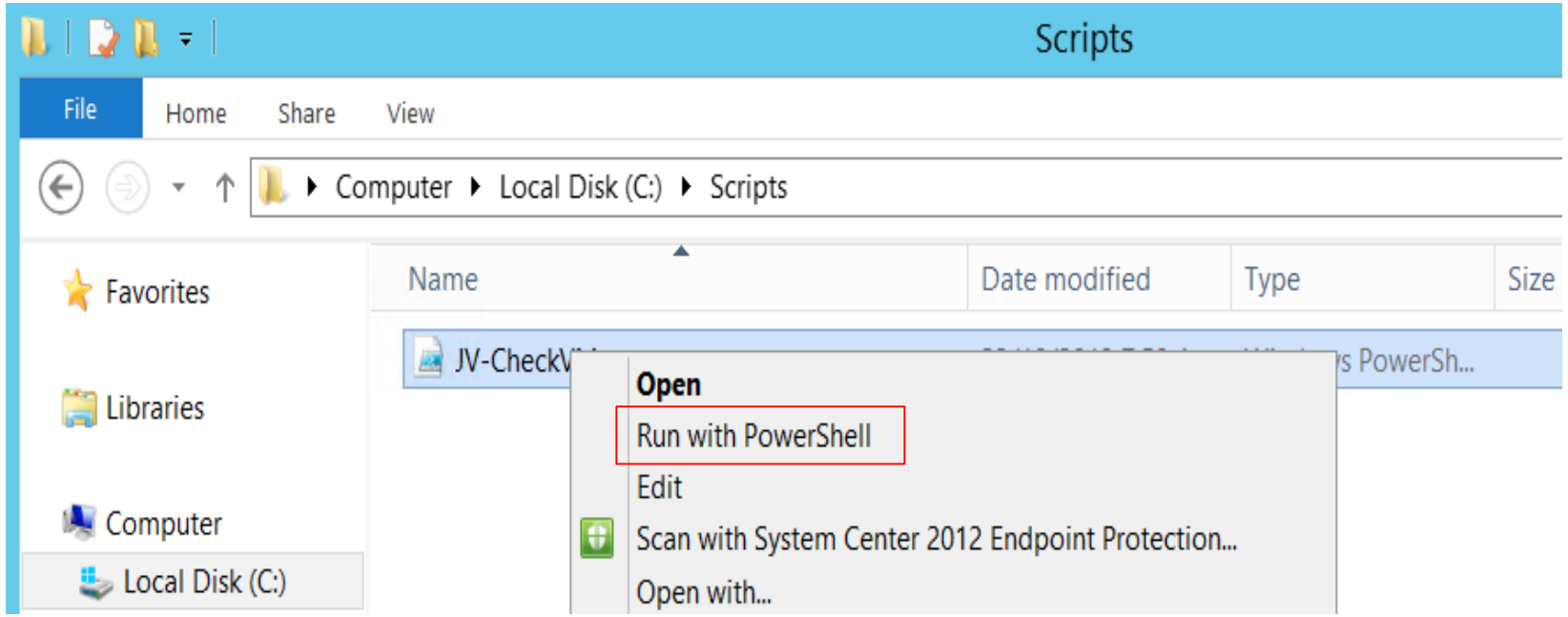
```
PS C:\> & "c:\scripts\my script.ps1"
```

Script is in environment path

```
PS C:\> Script.ps1
```

Running a Script With the Mouse

- Script files cannot be double clicked to run
- Run with PowerShell option:



- Right-click script
- Select Run with PowerShell

Launching a Script From Outside PowerShell (cmd.exe)

Optionally Keeps PowerShell Window open

Must be last parameter in command

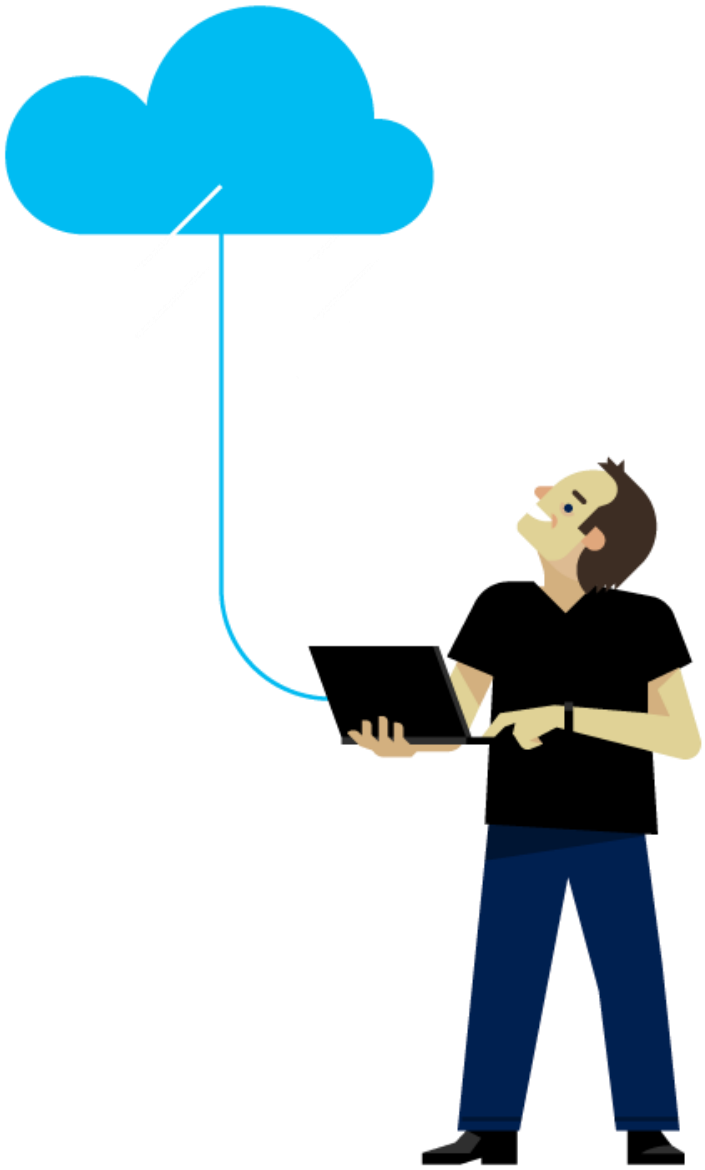
```
C:\> Powershell.exe -NoExit -File "c:\scripts\isecputime.ps1"
```

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

Start of script
Display the % CPU Time utilization by the ISE

Timestamp                 CounterSamples
---------                 --------------
6/25/2018 11:37:09 AM     \\win10\process(powershell_ise)\% processor time :
                          0
```

Demonstration

Starting Scripts

Questions?



Script signing

Script Signing

- Script signing validates the integrity of the script between signing and execution
- Execution policies can be used to enforce only signed scripts
- Certificate used should be of type: Code signing
- Public certificate should be used if script will be published

Script Signing

- Code signing certificate
- Trusted by computer where script will run

Step 1: Create a certificate variable

```
PS C:\> $cert = Get-ChildItem Cert:\CurrentUser\my\A4...
>> -CodeSigningCert
```

Step 2: Sign script

```
PS C:\> Set-AuthenticodeSignature .\ISECPUTime.ps1 $cert

   Directory: C:\Scripts

SignerCertificate                Status     Path
-----------------                ------     ----
A4..                             Valid      ISECPUTime.ps1
```

Script Before Signing

Sample.ps1 - Notepad

File Edit Format View Help

```
Write-Host "Start of script" -ForegroundColor White -BackgroundColor Green
Write-Host "Display the % CPU Time utilization by the ISE" -ForegroundColor White -BackgroundColor Green
Get-Counter "\Process(powershell_ise)\% Processor Time“
```

Ln 1, Col 1

Script Before Signing

Sample.ps1 - Notepad

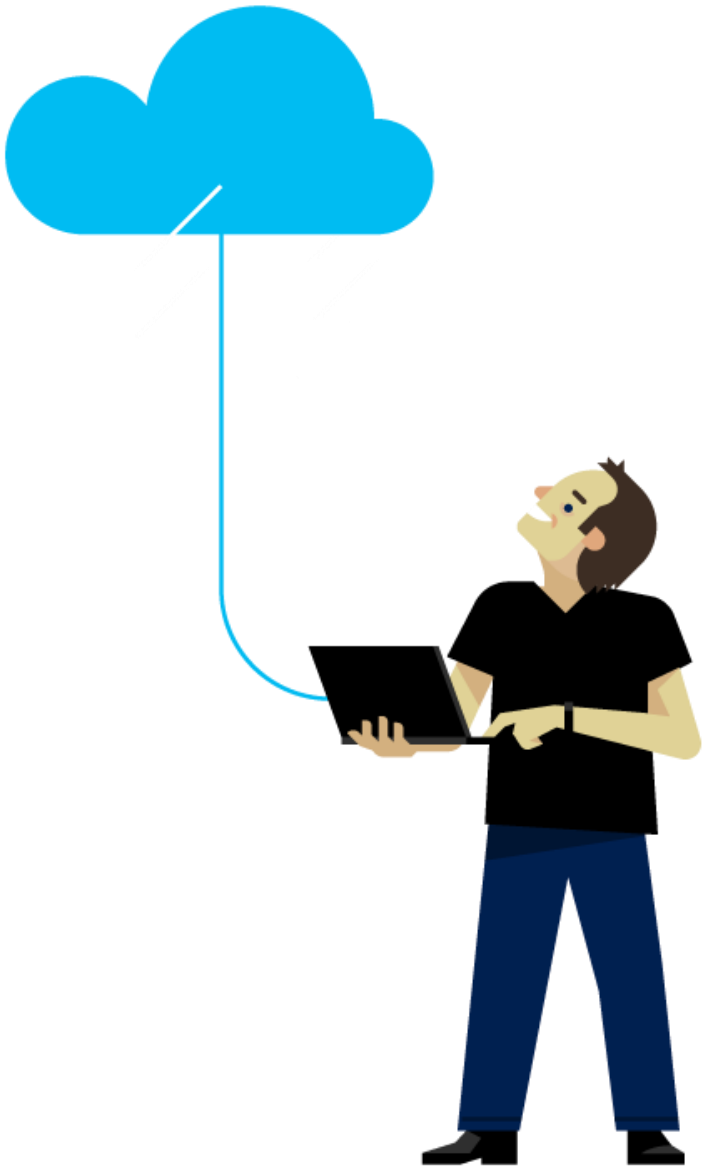
File Edit Format View Help

```
Write-Host "Start of script" -ForegroundColor White -BackgroundColor Green
Write-Host "Display the % CPU Time utilization by the ISE" -ForegroundColor White -BackgroundColor Green
Get-Counter "\Process(powershell_ise)\% Processor Time“
# SIG # Begin signature block
# mIIEMwYJKoZIhvcNAQcCoIIEJDCCBCACAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
# kjcCAQSgWsBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# agEAAsEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQU6vQAn5sf2qIxQqwWUDwTZnJj
# j5ufgfI9MIICOTCCAaagAwIBAgIQyLeyGZcGA4ZOGqK7VF45GDAJBgUrDgMCHQUA
# agEAAsEAAgEAAgEAAgEAMCEwCQYFKw4DAhoFAAQU6vQAn5sf2qIxQqwWUDwTZnJj
# kjcCAQSgWsBZMDQGCisGAQQBgjcCAR4wJgIDAQAABBAfzDtgWUsITrck0sYpfvNR
# SIG # End signature block
```

Ln 1, Col 1

Script signature block

Questions?



Single-line and Block Comments

Single-Line Comments

#	Comment character

Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Sample.ps1

```
1 param ($Computername)
2
3 #Testing connectivity to remote computers
4
5 $result = Test-Connection -ComputerName $Computername -Quiet -Count 1
6
7 Write-Host $result -ForegroundColor Green  # Inline comment
```

Block Comments

<#
#>

Block comment tags

Windows PowerShell ISE

File Edit View Tools Debug Add-ons Help

Sample.ps1

```
1  param ($Computername)
2
3  <# Testing connectivity to remote computers
4     Write Boolean output in Green
5  #>
6
7  $result = Test-Connection -ComputerName $Computername -Quiet -Count 1
8
9  Write-Host $result -ForegroundColor Green
```

Demonstration

Block Comments

The Param and Require Statements

Script Param Statement

- Must be first statement in script, except for comments
- Parameter values are available to commands in scripts

Param Statement

ScriptParamExample.ps1

```
1  param ($ComputerName)
2
3  $result = Test-Connection -ComputerName $ComputerName -Quiet -Count 1
4
5  Write-Host $result -ForegroundColor Green
```

Parameter

Parameter Value

```
PS C:\scripts> .\ScriptParamExample.ps1 -ComputerName localhost
True

PS C:\scripts> .\ScriptParamExample.ps1 -ComputerName DoesNotExist
False

PS C:\scripts>
```

Requires Statement

Special comment

Prevents script from running without required elements

Can only be used in scripts (not functions, cmdlets, etc)

Requires Option	Supported in PS Version
#Requires -Version <N>[.<n>]	2.0+
#Requires -PSSnapin <PSSnapin-Name> [-Version <N>[.<n>]]	2.0+
#Requires -ShellId <ShellId>	2.0+
#Requires -Modules { <Module-Name> | <Hashtable> }	3.0+
#Requires -RunAsAdministrator	4.0+

Version Requirement

- Prevents script from running on lower PowerShell versions
  - Script errors at start
  - Avoids unexpected errors from unsupported language, cmdlets, etc.
- Special comment tag: #Requires -Version <N>[.<n>]
- Get-help About_Requires

Sample Script

```
#requires -Version 3
Get-ChildItem c:\ -Hidden
```

Error when script runs within PowerShell v2

```
.\Test1.ps1 : The script 'Test1.ps1' cannot be run because it contained
a "#requires" statement at line 1 for Windows PowerShell version 3.0.
The version required by the script does not match the currently running
version of Windows PowerShell version 2.0.At line:1 char:12+
.\Test1.ps1 <<<<     + CategoryInfo          : ResourceUnavailable:
(Test1.ps1:String) [], ScriptRequiresException    +
FullyQualifiedErrorId : ScriptRequiresUnmatchedPSVersion
```

Administrator Requirement

- Script requires elevated user rights
  - Script errors at start indicating
  - Avoids unexpected errors in script
- Special comment tag: #Requires -RunAsAdministrator

Sample Script

```
#requires -RunAsAdministrator
Get-ChildItem c:\ -Hidden
```

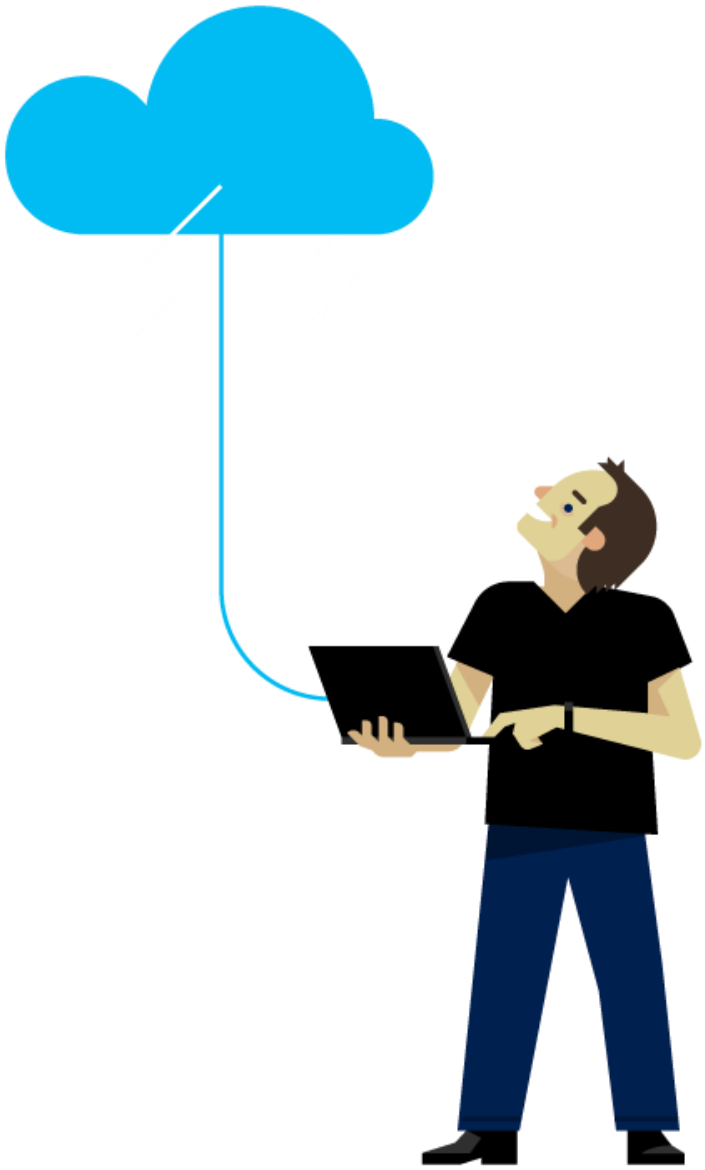
Error when script run from non-elevated session

```
.\RunAsAdminTest.ps1 : The script 'RunAsAdminTest.ps1' cannot be run
because it contains a "#requires" statement for
running as Administrator. The current Windows PowerShell session is not
running as Administrator. Start Windows
PowerShell by  using the Run as Administrator option, and then try
running the script again.
…
```

Demonstration

Param Statement and
Requires Statement

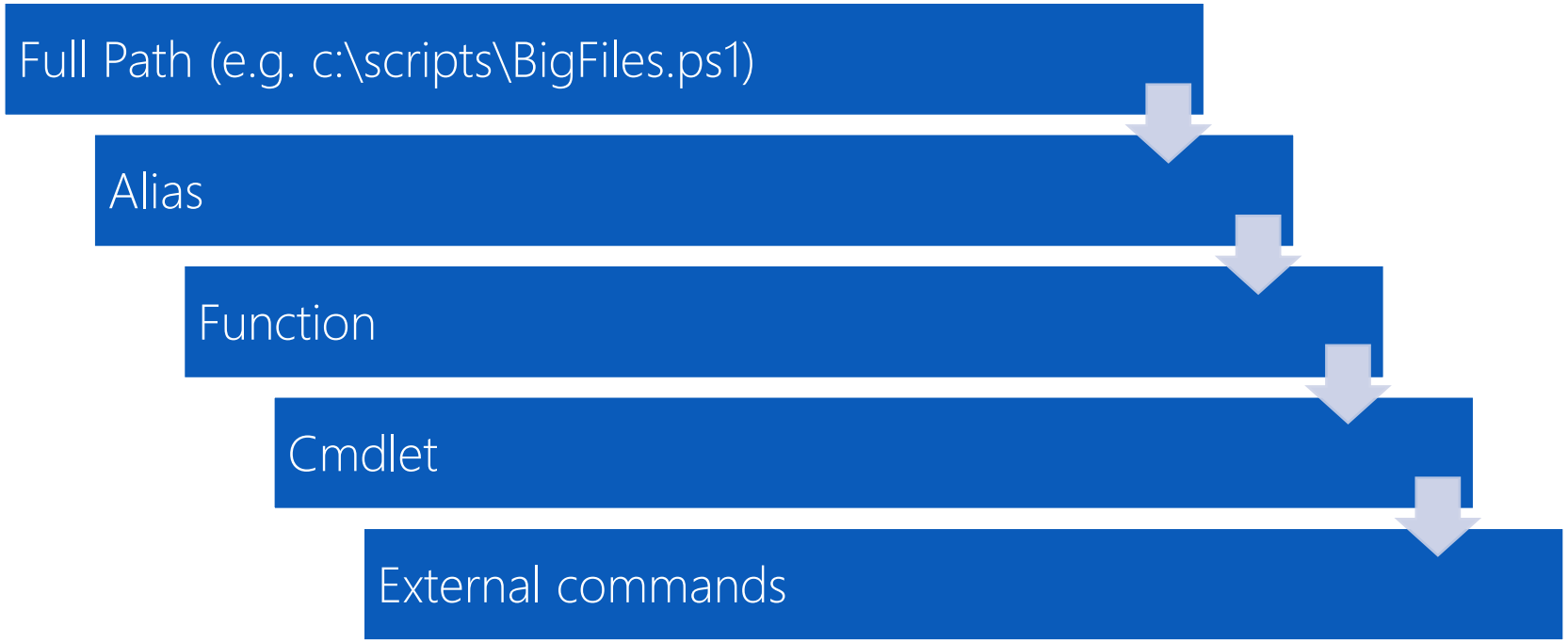
Questions?



Explore Command Precedence Rules

Command Lookup Precedence

- PowerShell rules that determine which command to run when there is more than one command with the same name



- Note: If the same type of command with the same name exists, PowerShell runs the command that was added to the session most recently

“Replace” Another Command

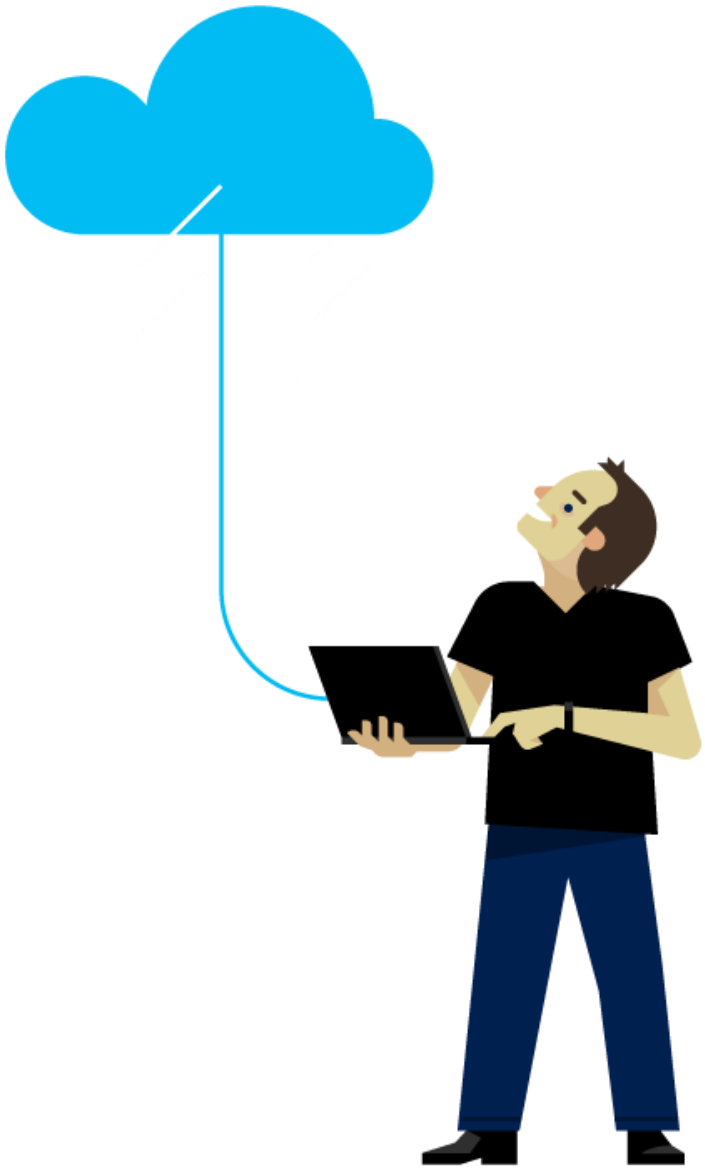
```
PS C:\scripts> ping MS

Pinging ms.contoso.local with 32 bytes of data:
Reply from 192.168.1.2: bytes=32 time<1ms TTL=64
Reply from 192.168.1.2: bytes=32 time<1ms TTL=64
Reply from 192.168.1.2: bytes=32 time<1ms TTL=64

Ping statistics for 192.168.1.2:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 0ms, Maximum = 0ms, Average = 0ms

PS C:\scripts> New-Alias -Name ping -Value Test-Connection
PS C:\scripts> ping MS

urce        Destination     IPV4Address      IPV6Address
----        -----------     -----------      -----------
WIN10       MS                  192.168.1.2
WIN10       MS            192.168.1.2
WIN10       MS                  192.168.1.2
```

Ping now cmdlet instead of external command

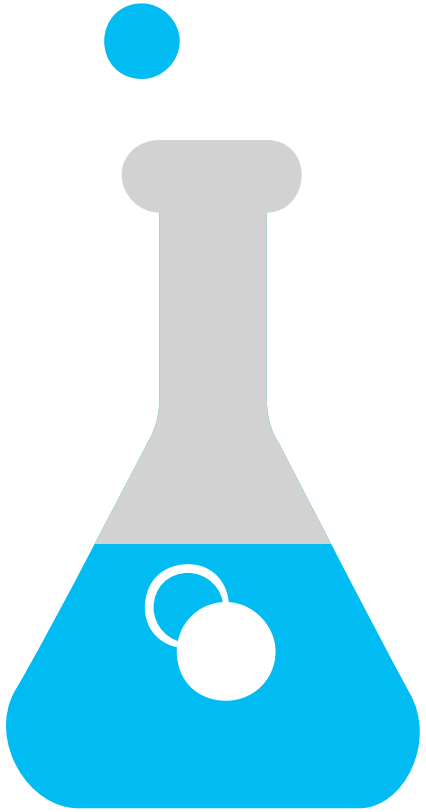
Module Qualify Command Name

```
#Run normal cmdlet
PS C:\> Get-Process system

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
   1427       0      140       4232    16               4 System


#Create function with same name
PS C:\> Function Get-Process {"This isn't Get-Process"}


#Command precedence runs function instead of cmdlet
PS C:\> Get-Process
This isn't Get-Process


#Module qualify command name
PS C:\> Microsoft.PowerShell.Management\Get-Process -Name system

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
   1427       0      140       4232    16               4 System
```

Questions?



Scripts



LAB

Microsoft