Microsoft

WorkshopPLUS - Windows PowerShell: Foundation Skills

Microsoft Services

Microsoft

Operators and Pipelining

Microsoft Services

Learnings Units covered in this Module

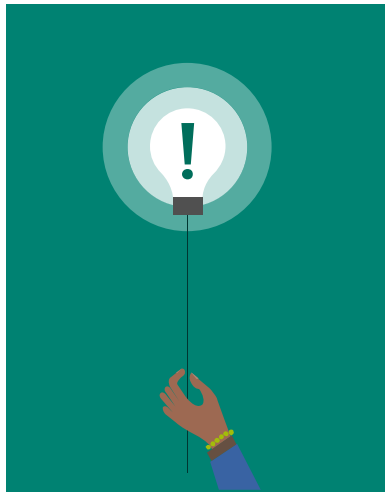
- Introduction to Operators
- Understanding the Windows PowerShell Pipeline
- Working with Pipelining
- Different types of Operators

Introduction to Operators

Objectives

After completing Introduction to Operators, you will be able to:

- Use operators to make code decisions



Comparison Operators

Comparison Operators

- Compare values
- Useful when testing conditions (If, Switch, Where-Object, etc.)
- Do not use = , > , < , ==, etc. to compare values
- Object type on left governs comparison

Comparison Operators - Basic

No Wildcards

Case-Insensitive Version

-eq	Equals	-ceq	-ieq
-ne	Not Equals	-cne	-ine
-gt	Greater Than	-cgt	-igt
-ge	Greater Than or Equal To	-cge	-ige
-lt	Less Than	-clt	-ilt
-le	Less Than or Equal To	-cle	-ile

Case-Sensitive Version

Basic Operators

```
PS C:\> 1 -eq 1
True

PS C:\> 1 -eq 2
False

PS C:\> 10 -gt 20
False

PS C:\> 10 -gt 5
True
```

Basic Operators

```
PS C:\> $true -eq ‘$false’
True

PS C:\> 'PowerShell' -gt 'CMDPrompt'
True

PS C:\> 'a' -lt 'aa'
True

PS C:\> $service = Get-Service bits
PS C:\> $service.Status -eq 'Running'
True
```

Demonstration

Operators

Comparison Operators - Wildcards

Case-Insensitive Version

-like	Equals with wildcards	-clike	-ilike
-notlike	Not Equals with wildcards	-cnotlike	-inotlike

Case-Sensitive Version

Allowed Wildcards	
*	Zero or any number of any chars
?	Exactly one of any char
[1az9]	Exactly one of given char(s)
[a-l]	Exactly one of range of given char(s)

-Like

```
PS C:\> 'Pear' -eq 'p*'
False

PS C:\> 'Pear' -like 'p*'
True

PS C:\> $Process = Get-Process -Name Sys*
PS C:\> $Process

Handles NPM(K)   PM(K) WS(K) VM(M) CPU(s) Id ProcessName
------- ------   ----- ----- ----- ------ -- -----------
    724      0    8972   628    13 51.33   4 System

PS C:\> $Process.Name -like '???????*'
False
```

Must be 7 chars followed by zero or more

Comparison Operators – Regular Expressions

- See Get-Help about_Regular_Expressions

-match	Regular Expression comparison	-cmatch	-imatch
-notmatch	Regular Expression NOT comparison	-cnotmatch	-inotmatch

Case-Sensitive Version

-Match

```
PS C:\> 'Digit 5 in this string' -match '\d'
True

PS C:\> 'hello there' -match '^there'
False

PS C:\> ‘Program Files' -match ‘files$’
True
```

\d - Digit

^ - Start of Text

$ - End of Text

See Get-Help about_Regular_Expressions for syntax

Comparison Operators – Array/Collection Containment

Always results in Boolean (True/False)

Case-Sensitive Version

Array on left, Singleton on Right			
-contains	Array Contains single	-ccontains	-icontains
-notcontains	Array not Contains single	-cnotcontains	-inotcontains

Case-insensitive Version

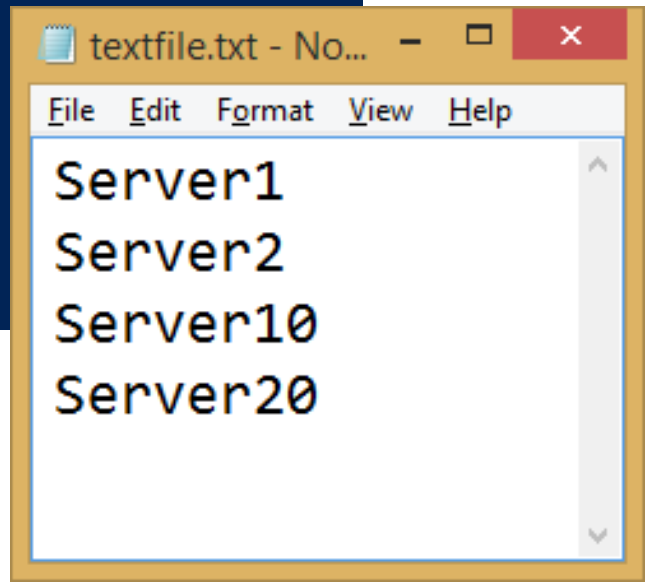
Singleton on left, Array on Right			
-in	Single in Array	-cin	-iin
-notin	Single not in Array	-cnotin	-inotin
-in, -notin, inotin introduced by PowerShell v3.0			

Array Containment

```
PS C:\> 1,2,3 -contains 2
True

PS C:\> "a","b","c" -notcontains “a"
False

PS C:\> 2 -in 1,2,3
True

PS C:\> "a" -notin "a","b","c"
False
```

Array Containment

```
PS C:\> (Get-Process).Name -contains
'Notepad'
True

PS C:\> $ServerList = Get-Content
.\textfile.txt
PS C:\> $ServerList -contains 'Server10'
True

PS C:\> 'Server20' -in $ServerList
True
```



Operator Case Sensitivity

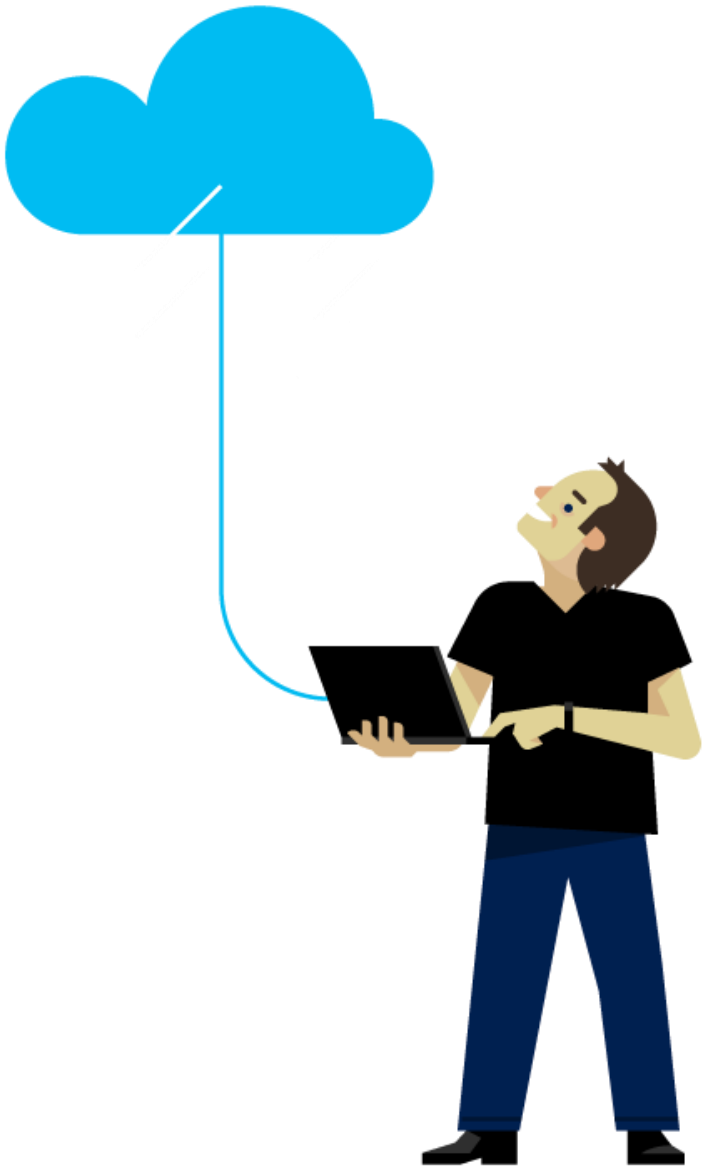
```
PS C:\> "abcd" -eq "ABCD"
True

PS C:\> "abcd" -like "ABC*"
True

PS C:\> "abcd" -match “ABCD$"
True
```

```
PS C:\> "abcd" -ceq "ABCD"
False

PS C:\> "abcd" -clike "ABC*"
False

PS C:\> "abcd" -cmatch “ABCD$"
False
```

Demonstration

Advanced Operators

Questions?



Logical Operators

Logical Operators

- Connect statements
- Compound conditions

Operator	Description
-and	TRUE only when both statements are TRUE.
-or	TRUE when either or both statements are TRUE.
-xor	TRUE only when one of the statements is TRUE and the other is FALSE.
-not or !	Negates the statement that follows it.

-and, –or, –xor, -not, !

```
PS C:\> (4 -lt 8) -and (5 -lt 10)
True

PS C:\> (4 -lt 8) -or (5 -lt 4)
True

PS C:\> (4 -lt 8) -xor (5 -lt 10)
False

PS C:\> -not (4 -lt 8)
False

PS C:\> !(Test-Path C:\windows)
False

PS C:\> (5 -lt 3 -or 4 -lt 8) -and (5 -lt 10 -and 20 -gt 10)
True
```

Demonstration

Logical operators

Range Operators

Range Operator

Numerical

```
PS C:\> 1..10
1
2
3
4
5
6
7
8
9
10
```

```
PS C:\> 11..20
11
12
13
14
15
16
17
18
19
20
```

```
PS C:\> 5..-4
5
4
3
2
1
0
-1
-2
-3
-4
```

```
PS C:\> -1..-9
-1
-2
-3
-4
-5
-6
-7
-8
-9
```

Range of Characters Operator

Alphabetical [<letter>-<letter>]

Used with –Like,-NotLike and Parameters that accept wildcards

```
PS C:\> Get-ChildItem C:\windows\System32\[a-d]*

    Directory: C:\windows\System32

Mode                LastWriteTime  Length Name
----                -------------  ------ ----
d----        22/08/2013  11:36 AM         AdvancedInstallers
d----        20/05/2014   8:22 AM         AppLocker
d----        19/10/2013   8:19 AM         Appmgmt
d----         8/04/2014   7:38 PM         Ar-SA
d----        19/10/2013  12:05 PM         BestPractices
...
```

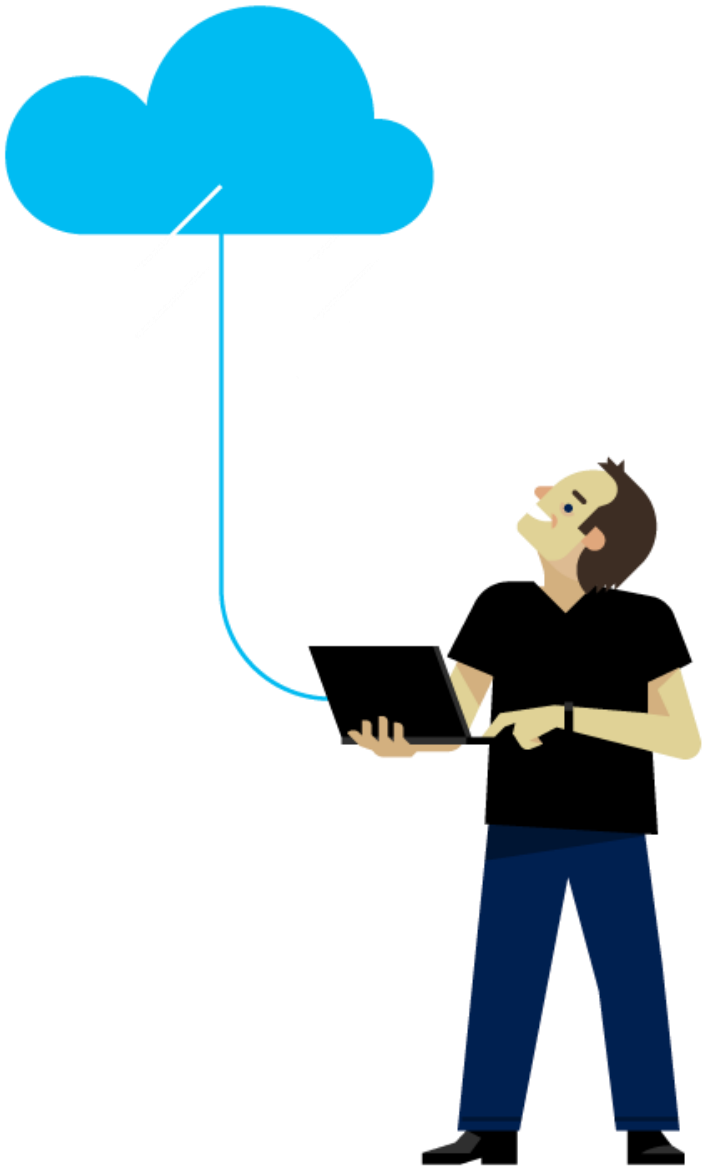
Specified Characters Operator

Alphabetical [<letter><letter>]

Used with –Like,-NotLike and Parameters that accept wildcards

```
PS C:\> Get-ChildItem C:\windows\System32\[jz]*

    Directory: C:\windows\System32

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----        23/08/2013   1:36 AM            ja-JP
d----        23/08/2013   1:36 AM            zh-CN
-a---        23/08/2013   5:12 AM     25600 jnwmon.dll
-a---        22/08/2013   9:03 PM    142848 joy.cpl
-a---        22/08/2013   9:01 PM    429568 zipfldr.dll
...
```

Questions?



Numeric Multipliers

Numeric Byte Multipliers

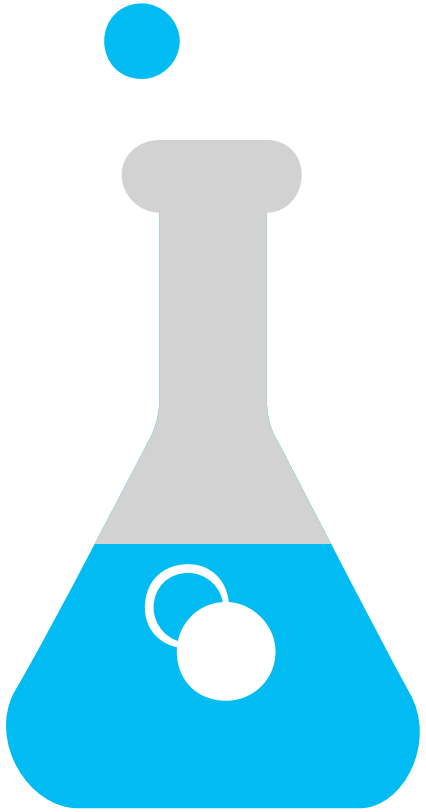
- Convenient byte multiples
- Commonly-used powers of 2
- Case-insensitive

Multiplier	Meaning	Example
kb	kilobyte (n * 1024)	`PS C:\> 2kb` `2048`
mb	megabyte (n * 1024 x 1024)	`PS C:\> 100mb` `104857600`
gb	gigabyte (n * 1024 x 1024 x 1024)	`PS C:\> 1.5gb` `1610612736`
tb	terabyte (n * 1024 x 1024 x 1024 x 1024)	`PS C:\> 1tb` `1099511627776`
pb	petabyte (n * 1024 x 1024 x 1024 x 1024 x 1024)	`PS C:\> 1pb` `1125899906842624`

Demonstration

Numeric Byte Multipliers

Introduction to Operators



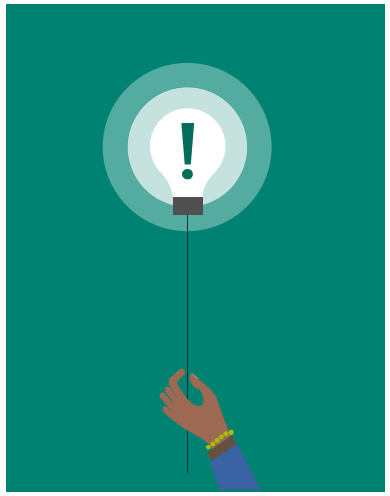
LAB

Understanding the Windows PowerShell Pipeline

Objectives

After completing Understanding the Windows PowerShell Pipeline, you will be able to:

- Work with the powershell Pipeline
- Understand how the basic commands interact with the Pipeline



What is a pipeline?

What is a Pipeline?

- Series of commands connected by the pipeline character.
  - Broken vertical bar |

- Passes OBJECT, not text, **Left** to **Right** starting with first command.
  - Each subsequent command takes its **Input** from the previous command's **Output**

- Allows Filtering, Formatting and Outputting

- Cmdlets are designed to be in a pipeline

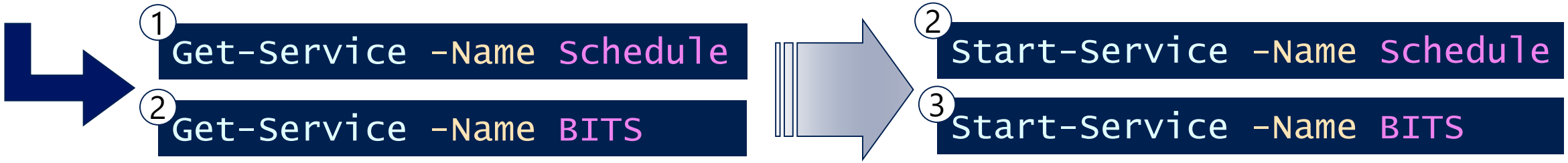
- Pipeline statements typically start with a "Get" command which introduces the objects to be used throughout the statement.

- Increases performance of operations by allowing simultaneous execution

Using the Pipeline

- Sends output from one command as input to another command
- Pipeline statements typically start with a "Get" command which introduces the objects to be used throughout the statement.
- Increases performance of operations by allowing simultaneous execution of each portion of pipeline statement.

```
PS C:\> Get-Service -Name Schedule , BITS | Start-Service
```



Order of Operations	
1. Get-Service –Name Schedule	
2. Start-Service –Name Schedule	2. Get-Service –Name Bits
3. Start-Service –Name Bits	

(Happens Simultaneously)

The “Get” Cmdlets

- Typically placed first in the pipeline
- Provides the input to be processed

Returns schedule and bits services

```
PS C:\> Get-Service -Name Schedule , BITS | Start-Service
```

Takes an action on the services

External Commands

- Can be used as input to the pipeline

External command

```
PS C:\> whoami.exe
Contoso\power

PS C:\> whoami.exe | Split-Path -Parent
contoso

PS C:\> whoami.exe | Split-Path -Leaf
power
```

Text File Input

- Text files provide input to be processed by the pipeline

Reads a text file

```
PS C:\> Get-Content .\services.txt | Get-Service
```

Takes an action on each line in the file

Pipeline Input

- Commands that get information are very suitable for use in the pipeline like the “Get-” commands
- Pipeline input can be any object
- PowerShell will bind properties from the input object on the left to the new command on the right

The “Get” Cmdlets

- Typically placed first in the pipeline
- Provides the input to be processed

Returns schedule and bits services

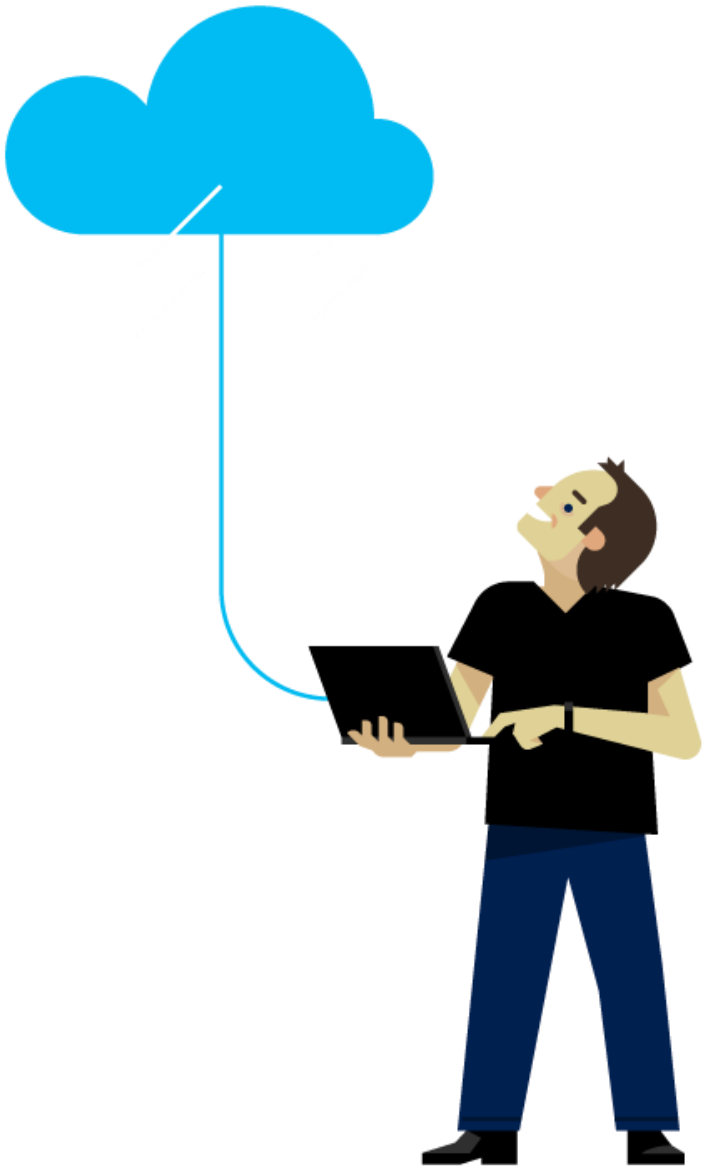
```
PS C:\> Get-Service -Name Schedule , BITS | Start-Service
```

Takes an action on the services

Demonstration

Pipeline Basics and Optimization

Questions?



Object Cmdlets

Object Cmdlets

Name	Description
Sort-Object	Sorts objects by property values
Select-Object	Selects object properties
Group-Object	Groups objects that contain the same value for specified properties
Measure-Object	Calculates numeric properties of objects, and the characters, words, and lines in string objects, such as text files
Compare-Object	Compares two sets of objects

Sort-Object and Select-Object

Get all processes, Sort by virtual memory then Select top 2

```
PS C:\> Get-Process | Sort-Object VM -Descending |
Select-Object -First 2

Handles  NPM(K)  PM(K)   WS(K)   VM(M)   CPU(s) Id   ProcessName
-------  ------  -----   -----   -----   ------ --   -----------
   1283      55  21020   30340    1237   477.78 304   svchost
   1926      44  285244  230112   1165   716.45 4124  livecomm
```

Group-Object

Get security event log then Group by entry type

```
PS C:\> Get-EventLog -LogName Security |
Group-Object EntryType

 Count Name          Group
----- ----          -----
18105 SuccessAudit  {System.Diagnostics.EventLogEntry,Sys...
   25 FailureAudit  {System.Diagnostics.EventLogEntry,Sys...
```

Measure-Object

Get files in c:\scripts then **Measure** their number (count) and total size (length) in bytes

```
PS C:\> Get-ChildItem C:\Scripts |
Measure-Object -Property Length -Sum

Count    : 2
Average  :
Sum      : 217837
Maximum  :
Minimum  :
Property : Length
```

Compare-Object

Comparing text files

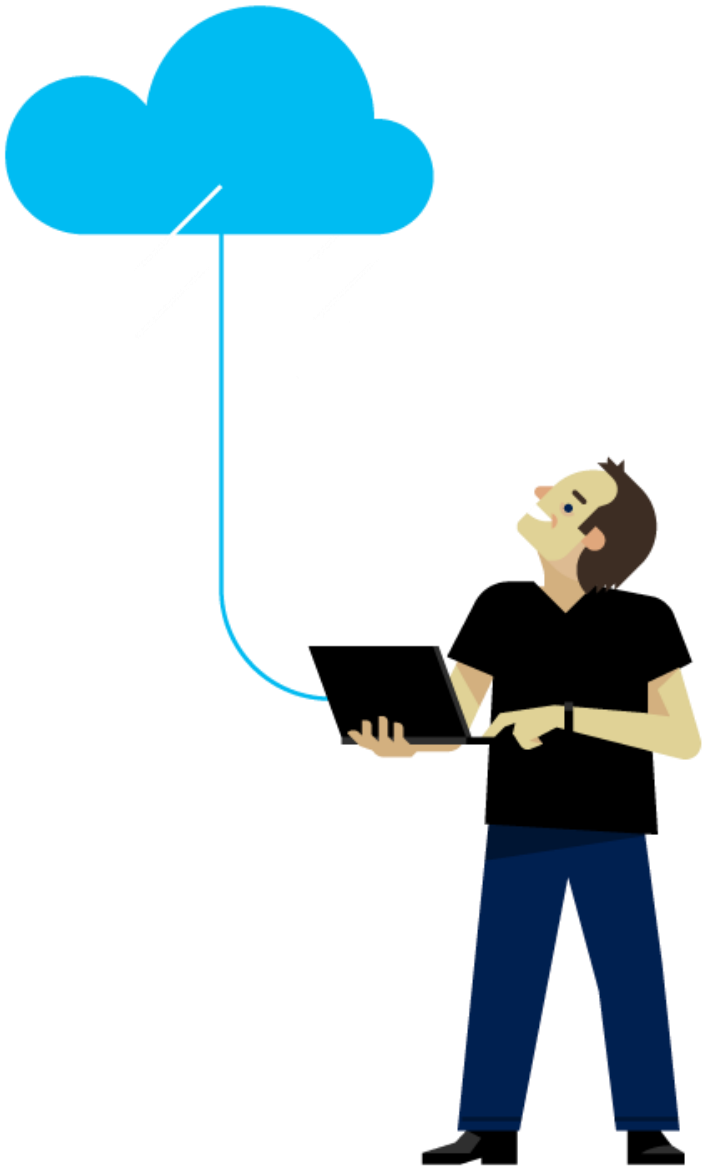
```
PS C:\> $ref = Get-Content -Path .\servers1.txt
PS C:\> Get-Content -Path .\servers2.txt | Compare-Object -ReferenceObject
$ref

InputObject SideIndicator
----------- -------------
Server3     =>
```

"Server3" is only in the difference variable (servers2.txt)

Demonstration

Object Cmdlets in the Pipeline

Questions?



Format Cmdlets

Format Cmdlets

- Convert pipeline objects into formatted output, typically for human consumption
- Should be last Cmdlet on the pipeline (only followed by Out-* Cmdlets)

Format-List	Format-Table	Format-Wide
• Key Parameters: • -Property * lists all properties	• Key Parameters: • -Autosize • -Wrap	• Key Parameters: • -Autosize • -Column

Format-List With Default Properties

```
PS C:\> Get-Process -Name powershell | Format-List

Id      : 6400
Handles : 472
CPU     : 0.78125
Name    : powershell
```

- Output is in list format
- Properties chosen are based on default formatting in PowerShell by object type

Format-List With Apecific Properties

```
PS C:\> Get-Process -Name powershell |
Format-List -Property Name, BasePriority,
PriorityClass

Name          : powershell
BasePriority  : 8
PriorityClass : Normal
```

- Output in list format
- Consists of specified properties

Format-Table With Default Properties

```
PS C:\> Get-Process | Format-Table

 Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     82       7     1308       1420    45     0.14   2308 armsvc
    195      13     2568       3440    94     3.78   1192 atieclxx
    110       6      852       1172    23     0.09    868 atiesrxx
    565      20     6384       7092   113    42.14   4308 BasisSync
    180      12     2276       2660    89     0.41   7744 BDAppHost
    142      11     1860       1768    76     0.14   7712 BDExtHost
    335      24    12120      14988   126     1.31   7772 BDRuntimeHost
    413      31     8128      10668   209     1.39   6636 BingDesktop
. . .
```

Format-Table With Specific Properties

```
PS C:\> Get-Process |
Format-Table -Property name,workingset,handles

Name                                WorkingSet                Handles
----                                ----------                -------
csrss                                   847872                    216
csrss                                   356352                     91
csrss                                 15646720                    183
dwm                                    7045120                    176
dwm                                   30498816                    201
explorer                              37539840                   1427
Idle                                      4096                      0
LogonUI                                6897664                    367
lsass                                  7622656                   1050
MsMpEng                               24444928                    528
powershell_ise                       144850944                    515
...
```

Format-Table With Specific Properties and -AutoSize

```
PS C:\> Get-Process |
Format-Table -Property name,workingset,handles -AutoSize

Name            WorkingSet Handles
----            ---------- -------
csrss               843776     216
csrss               356352      91
csrss             15523840     183
dwm                7045120     176
dwm               30691328     201
explorer          37486592    1421
Idle                 4096       0
LogonUI            6897664     367
lsass              7454720    1055
MsMpEng           22908928     527
powershell_ise   147017728     565
...
```

Format-Table With Specific Properties

```
PS C:\> Get-Process | Format-Table -Property Name,Path,WorkingSet

Name                Path                        WorkingSet
----                ----                        ----------
armsvc                                             1454080
atieclxx                                           3760128
atiesrxx                                           1200128
audiodg                                           11911168
BDAppHost           C:\Program Fil...              2736128
BDExtHost           C:\Program Fil...              1826816
BDRuntimeHost       C:\Program Fil...             15331328
BingDesktop         C:\Program Fil...             10981376
CCC                 C:\Pro                         5857280
...
```

Path truncated due to wide values

Format-Table With Auto Sized Columns

```
PS C:\> Get-Process | Format-Table -Property Name,Path,WorkingSet -AutoSize

Name            Path
----            ----
armsvc
atieclxx
atiesrxx
BDAppHost       C:\Program Files (x86)\Microsoft\BingDesktop\BDAppHost.exe
BDExtHost       C:\Program Files (x86)\Microsoft\BingDesktop\BDExtHost.exe
BDRuntimeHost C:\Program Files
(x86)\Microsoft\BingDesktop\BDRuntimeHost...
BingDesktop     C:\Program Files (x86)\Microsoft\BingDesktop\BingDesktop.exe
CCC             C:\Program Files (x86)\ATI Technologi   \ATI.ACE\Core-
Stati...
...
```

Path property truncation is minimized with –Autosize, but workingset column lost

Format-Table With Wrap

```
PS C:\> Get-Process |
Format-Table -Property Name,Path,WorkingSet -AutoSize -Wrap


Name            Path
----            ----
armsvc
atieclxx
atiesrxx
BDAppHost       C:\Program Files (x86)\Microsoft\BingDes
                ktop\BDAppHost.exe
BDExtHost       C:\Program Files (x86)\Microsoft\BingDes
                ktop\BDExtHost.exe
BDRuntimeHost C:\Program Files (x86)\Microsoft\BingDes
                ktop\BDRuntimeHost.exe
BingDesktop     C:\Program Files (x86)\Microsoft\BingDes
                ktop\BingDesktop.exe
```

Path property is line wrapped, workingset still lost in this case

Format-Table With Grouping

Processes are grouped by BasePriority (need to sort by groupby prop first)

```
PS C:\> Get-Process |
Sort-Object -Property BasePriority |
Format-Table -GroupBy BasePriority -Wrap -AutoSize

   BasePriority: 0

Handles NPM(K) PM(K) WS(K) VM(M) CPU(s) Id ProcessName
------- ------ ----- ----- ----- ------ -- -----------
      0      0     0    24     0         0 Idle
...
PS C:\> Get-EventLog -LogName Security | Group-Object EntryType | Format-
Table -AutoSize -Wrap

Count  Name          Group
-----  ----          -----
181027 SuccessAudit {System.Diagnostics.EventLogEntry}
    25 FailureAudit {System.Diagnostics.EventLogEntry}
```

Autosize minimizes data truncation, wrap eliminates it

Format-Wide – Default 2 Columns

```
PS C:\> Get-ChildItem | Format-Wide

    Directory: C:\

[PerfLogs]                          [Program Files]
[Program Files (x86)]               [PShell]
[Users]                             [Windows]
```

Output displayed in 2 columns by default

Format-Wide – Explicit Number of Columns

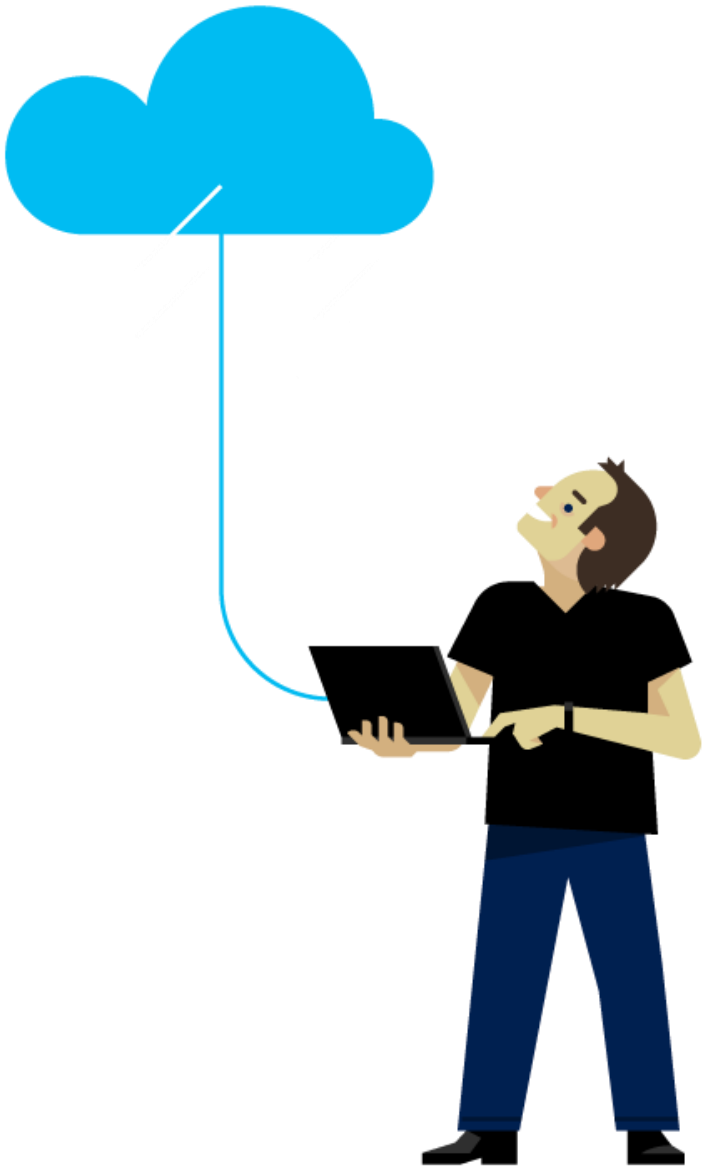
```
PS C:\> Get-ChildItem | Format-Wide -Column 3

    Directory: C:\

[PerfLogs]          [Program Files]          [Program Files (x86)]
[PShell]            [Users]                  [Windows]
```

Output displayed in 3 columns

Demonstration

Format Cmdlets

Questions?



Export and Import Cmdlets

Export Cmdlets

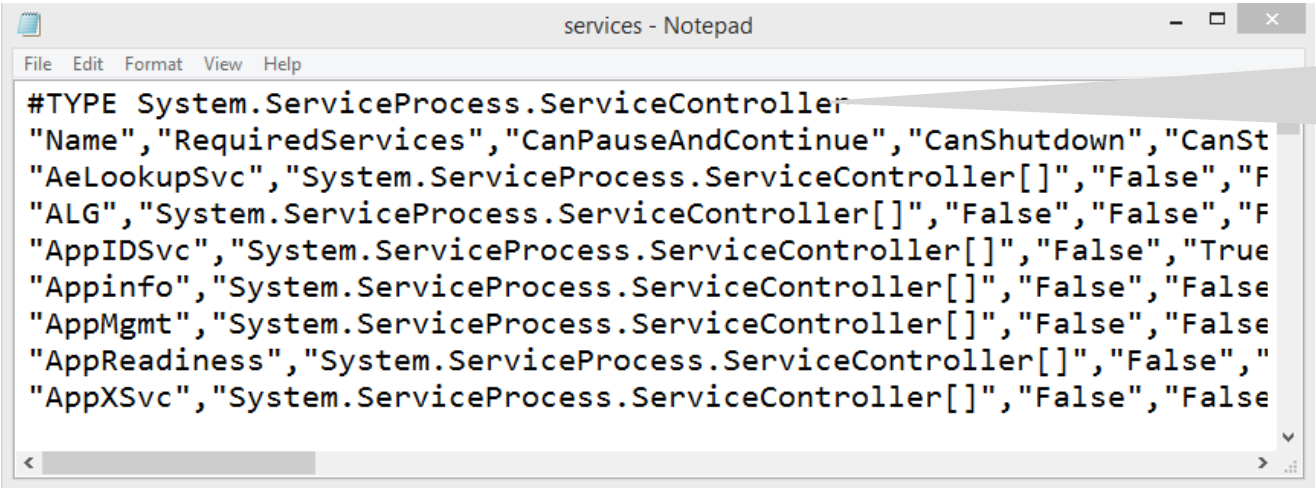
- Export objects to text file
- Provides structured information which can be imported
- Should be last cmdlet on the pipeline

Export-Csv	Export-CliXml

• Key Properties:
  • -Path
  • -Delimiter
  • -UseCulture
	• Key Properties
	  • -Path
	  • -Depth

Export-CSV

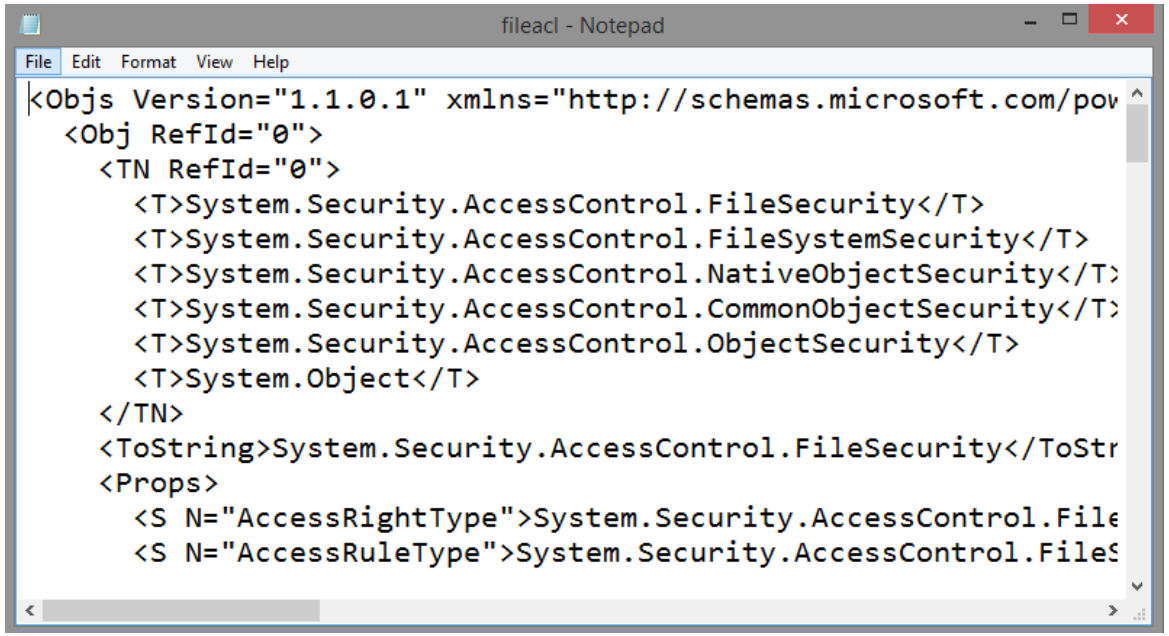
```
PS C:\> Get-Service | Export-Csv c:\services.csv

PS C:\> notepad.exe C:\services.csv
```



-NoTypeInformation parameter avoids this as 1st line

Export-Clixml

```
PS C:\> get-acl C:\Process.txt -Audit |
Export-Clixml -Path fileacl.xml

PS C:\> notepad .\fileacl.xml
```



Import Cmdlets

- Imports objects from text file
- Should be first cmdlet on the pipeline

Import-Csv	Import-CliXml
• Key Properties: • -Path • -Delimiter • -UseCulture	• Key Properties • -Path

Import-Csv

```
PS C:\> Import-Csv C:\usermailbox.csv | Select-Object mailbox

Mailbox
-------
administrator@contoso.com
dpark@contoso.com
kakers@contoso.com
```

usermailbox.csv - Notepad

File Edit Format View Help

```
UserName,Mailbox,Quota
administrator,administrator@contoso.com,1GB
DanPark,dpark@contoso.com,5GB
KimAkers,kakers@contoso.com,500MB
```

Import-CliXml

```
PS C:\> Import-Clixml -Path 'C:\temp\1.xml'
tekst

Handles  NPM(K)    PM(K)      WS(K)     CPU(s)     Id S
                                                     I
-------  ------    -----      -----     ------     -- -
    811      44    31340      68712       4,53   9452 2
```

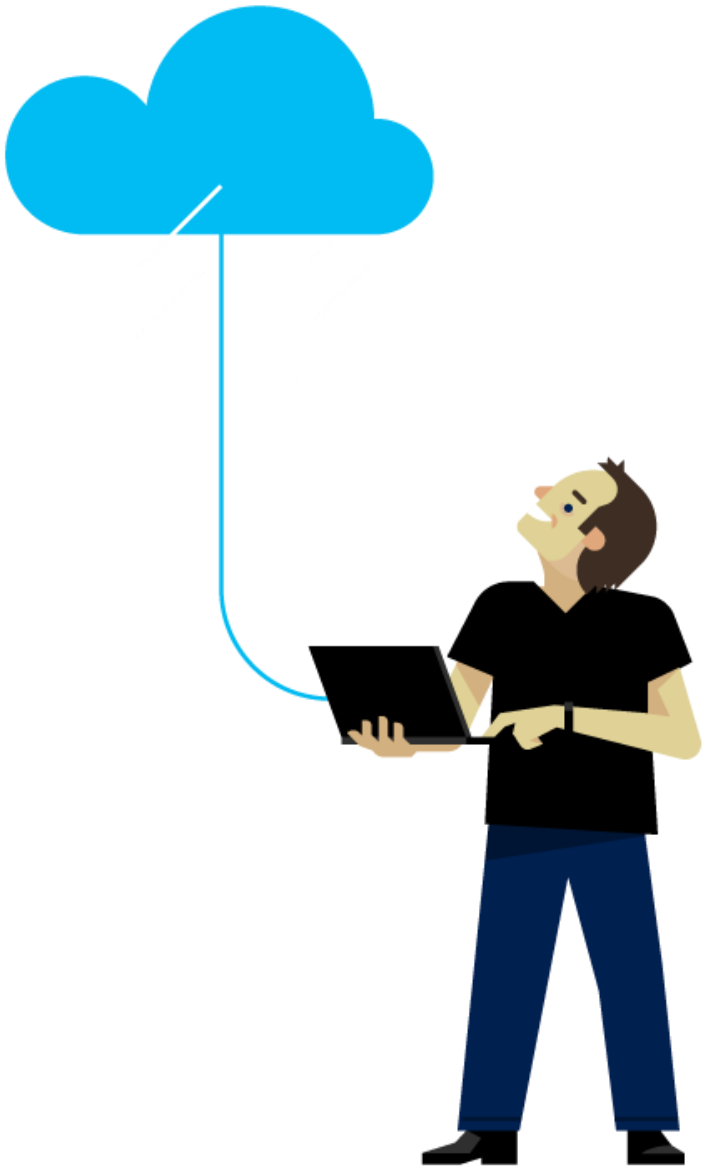
C:\temp\1.xml


```
<?xml version="1.0"?>
- <Objs xmlns="http://schemas.microsoft.com/powershell/2004/04" Version="1.1.0.1">
    <S>tekst</S>
  - <Obj RefId="0">
    - <TN RefId="0">
        <T>System.Diagnostics.Process</T>
        <T>System.ComponentModel.Component</T>
        <T>System.MarshalByRefObject</T>
        <T>System.Object</T>
      </TN>
      <ToString>System.Diagnostics.Process (ApplicationFrameHost)</ToString>
    - <Props>
        <I32 N="BasePriority">8</I32>
```

Demonstration

Import and Export cmdlets

Questions?



Out Cmdlets

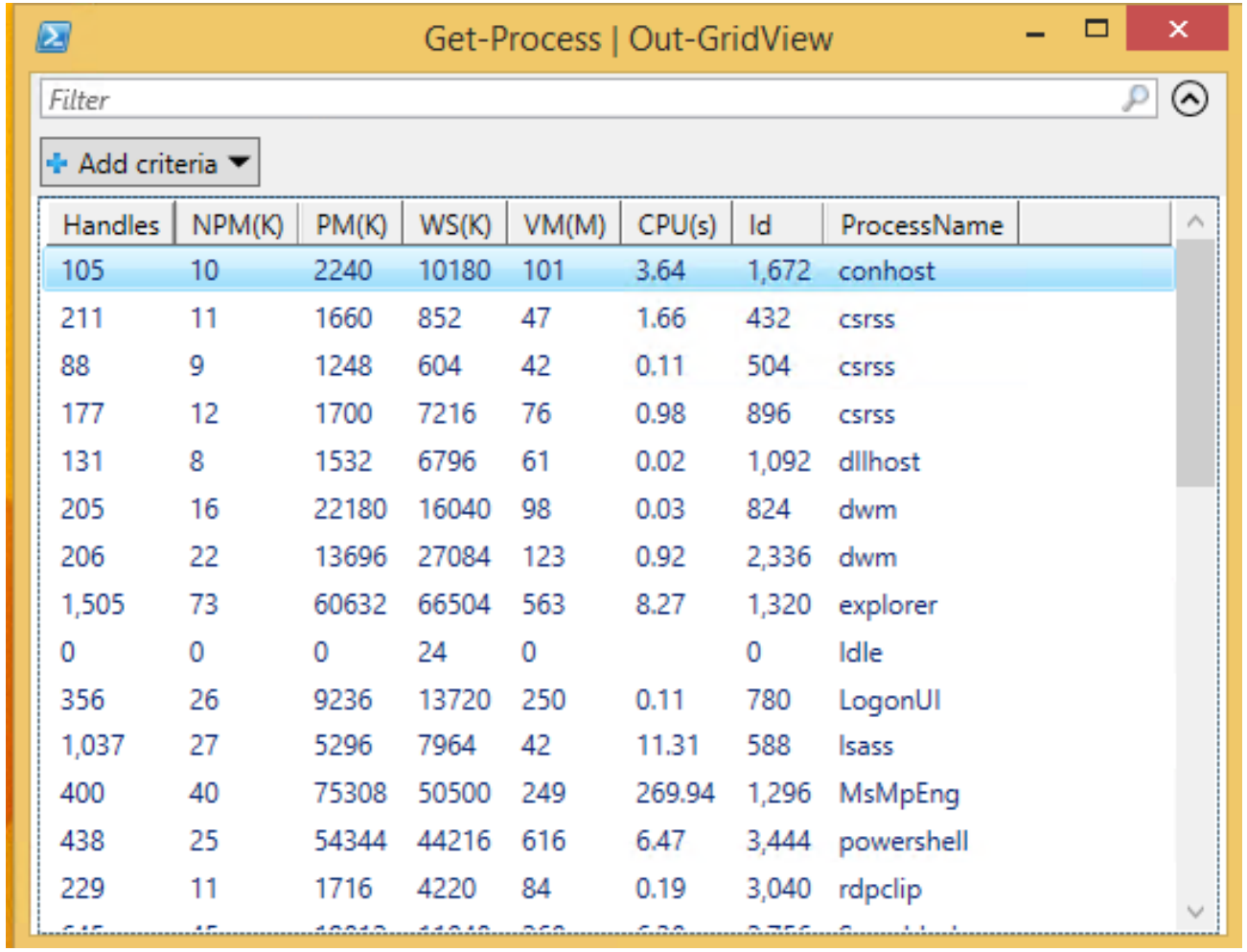
Out Cmdlets

Sends command output to a specified device

Name	Description
Out-Default	Sends output to default formatter and to default output cmdlet (Out-Host)
Out-File	Sends output to a file Append switch parameter Encoding parameter allows control of the character encoding
Out-GridView	Sends output to an interactive table in a separate GUI
Out-Host	Default Sends output to PowerShell host Paging switch parameter displays one page at a time
Out-Null	Deletes output instead of sending it down the pipeline
Out-Printer	Sends output to a printer
Out-String	Sends objects to the host as a series of strings

Out-Gridview

```
PS C:\> Get-Process | Out-GridView
```

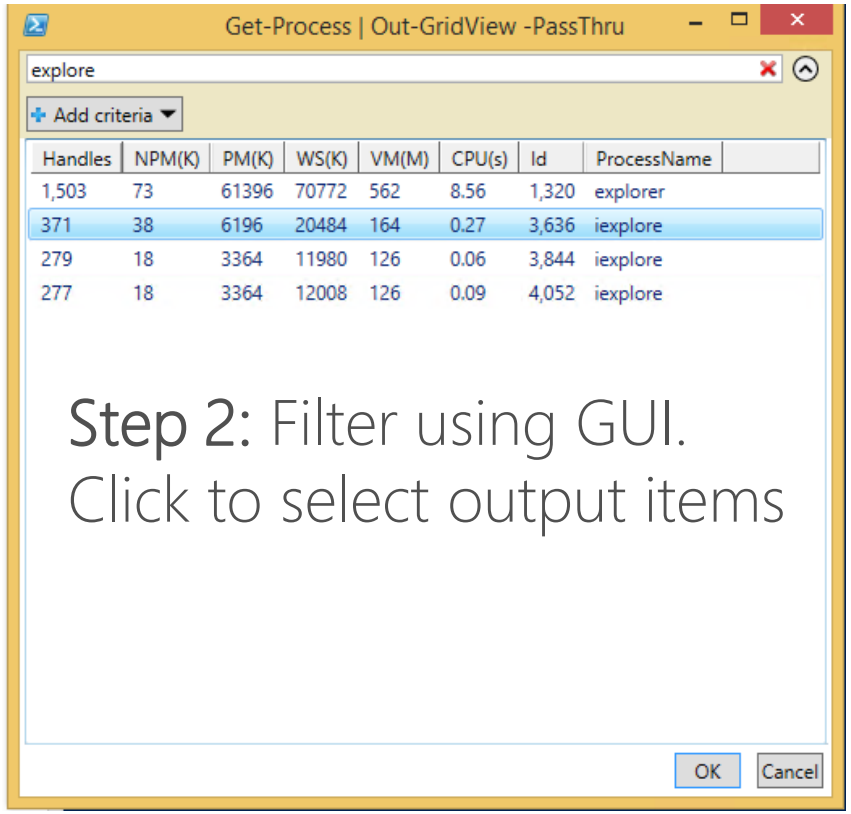


Handles	NPM(K)	PM(K)	WS(K)	VM(M)	CPU(s)	Id	ProcessName
105	10	2240	10180	101	3.64	1,672	conhost
211	11	1660	852	47	1.66	432	csrss
88	9	1248	604	42	0.11	504	csrss
177	12	1700	7216	76	0.98	896	csrss
131	8	1532	6796	61	0.02	1,092	dllhost
205	16	22180	16040	98	0.03	824	dwm
206	22	13696	27084	123	0.92	2,336	dwm
1,505	73	60632	66504	563	8.27	1,320	explorer
0	0	0	24	0		0	Idle
356	26	9236	13720	250	0.11	780	LogonUI
1,037	27	5296	7964	42	11.31	588	lsass
400	40	75308	50500	249	269.94	1,296	MsMpEng
438	25	54344	44216	616	6.47	3,444	powershell
229	11	1716	4220	84	0.19	3,040	rdpclip

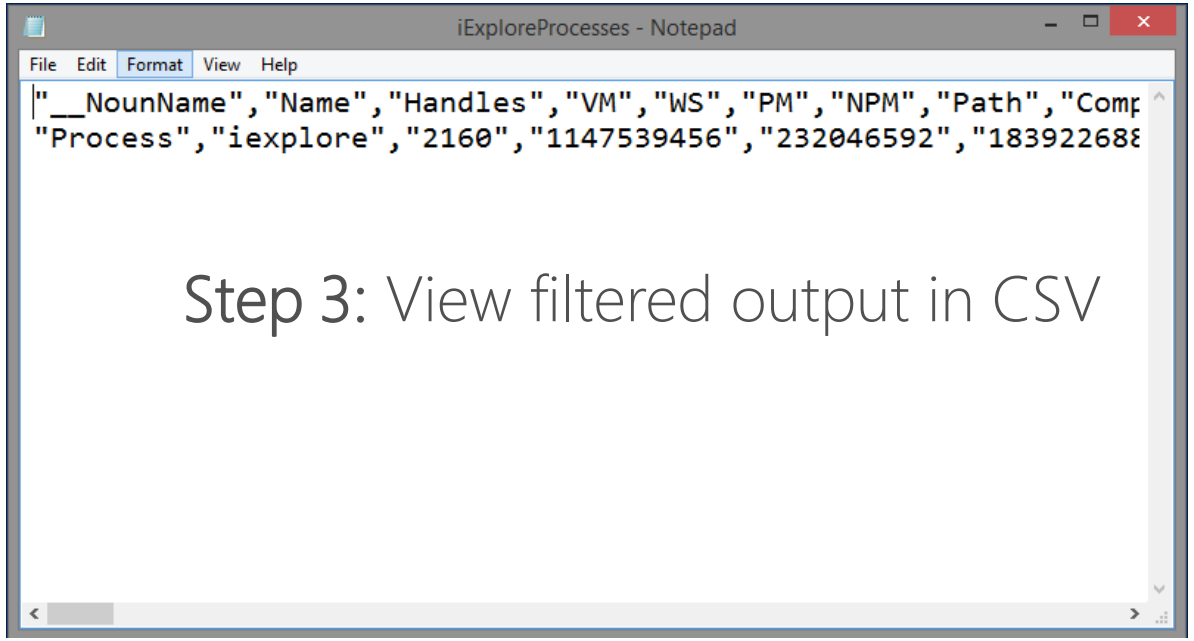
Out-Gridview with PassThru

Step 1: Send Get-Process output to Out-Gridview with PassThru switch parameter, followed by export to CSV

```
PS C:\> Get-Process | Out-GridView -PassThru |
Export-Csv c:\scripts\iExploreProcesses.csv -NoTypeInformation
```



Step 2: Filter using GUI. Click to select output items

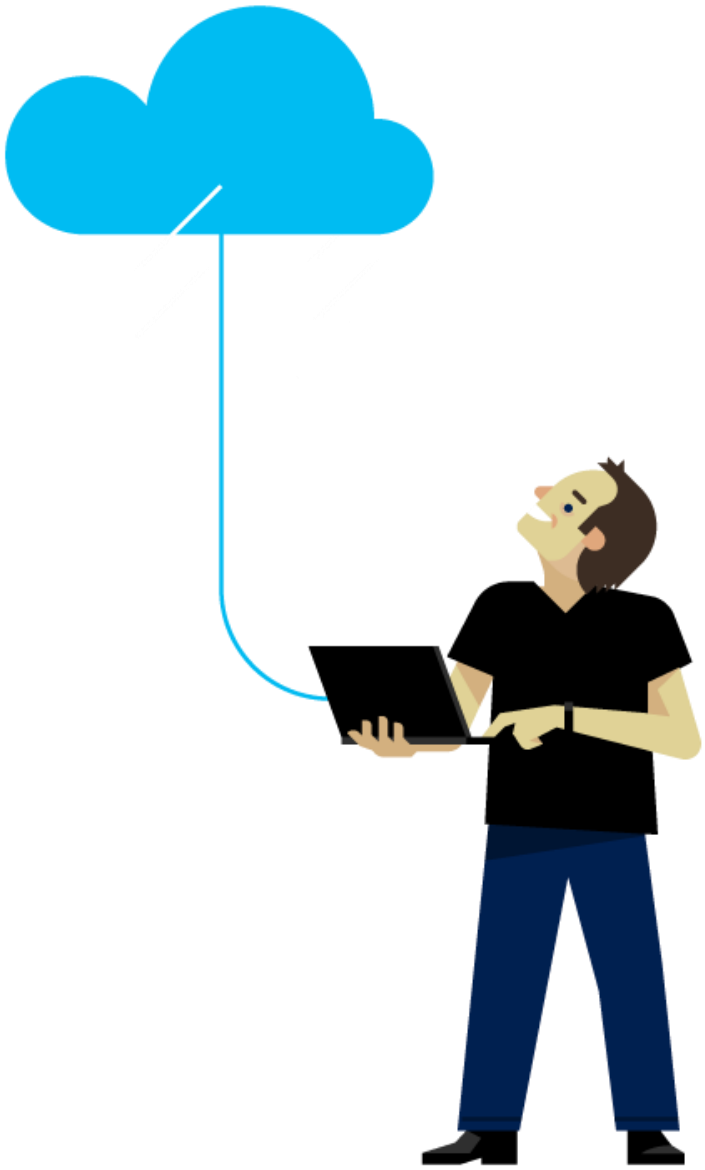


Step 3: View filtered output in CSV

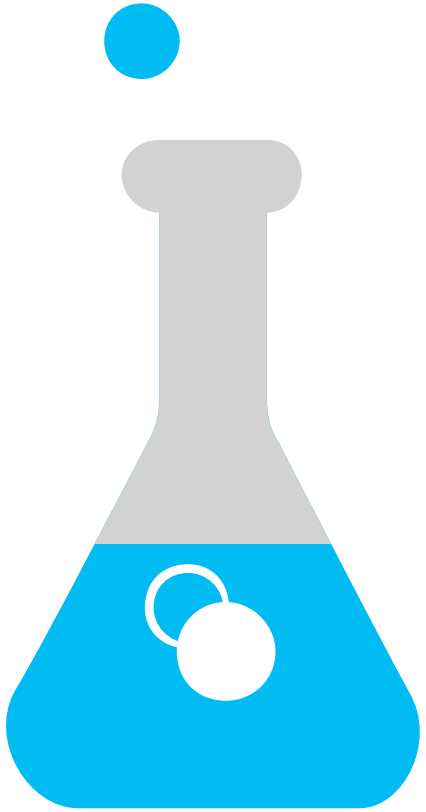
Demonstration

Out-* Cmdlets

Questions?



Understanding the Windows PowerShell Pipeline



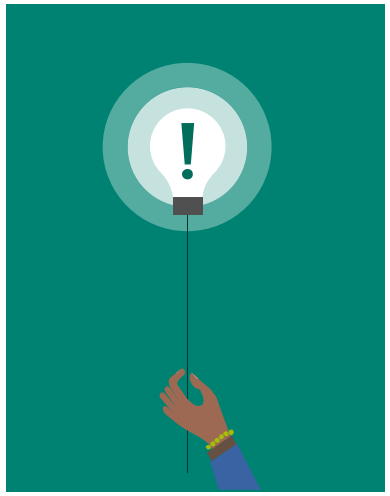
LAB

Working with Pipelining

Objectives

After completing Working with Pipelining, you will be able to:

- Work with advanced Pipeline objects



Pipeline Variable

Pipeline Variables

- When multiple objects are piped, PowerShell sends objects one at a time
- Built-in variables $_ and $PSItem represent current object on pipeline
- Used to perform an action on every object
- Use -PipelineVariable parameter to name your own variable on the pipeline
- Scoped only to current pipeline

Pipeline Variables - Examples

Default Pipeline variables:

Both lines produce the same result

```
PS C:\> Get-Process | Where-Object {$psitem.ws -gt 100MB}
PS C:\> Get-Process | Where-Object {$_.ws -gt 100MB}
```

User defined variables:

Storing cmdlet output in a user-defined

```
PS C:\>  Get-Process -PipelineVariable CurrentProcess |
Where-Object {$CurrentProcess.ws -gt 100MB}
```

Demonstration

Pipeline Variables

More Object Cmdlets

Other Object Cmdlets

Cmdlet	Usecase
ForEach-Object	Performs an operation against each item. Aliases: • % • ForEach
Where-Object	Filters objects in the pipeline Aliases: • ? • Where

ForEach-Object

Take an action on each object

```
PS C:\> Get-Service net* | ForEach-Object -process
{"Hello " + $_.Name}

Hello Netlogon
Hello Netman
Hello netprofm
Hello NetTcpPortSharing
```

Pipeline Filtering With Where-Object

```
Get-ChildItem -path c:\windows -filter *.ini |`
  Where-Object -filterscript {$_.length -lt 10kb} |`
    Sort-Object -property Length |`
      Format-Table -property name, length
```

Get-ChildItem –path c:\windows –filter *.ini

(FileInfo)
(*.ini)

Where-Object {$_.length -lt 10kb}

(FileInfo)
(*.ini)
(Length < 10240)

Sort-Object -property Length

(FileInfo)
(*.ini)
(Length < 10240)
(Sorted by length)

Format-Table -property name, length

(FileInfo)
(*.ini)
(Length < 10240)
(Sorted by length)
(Formatted in table)

Where-Object (Simple syntax)

- Simplified filtering syntax
- Syntax emulates natural language
- PowerShell v3.0+
- Note: Multiple filter conditions need full syntax

PowerShell v1.0+

```
PS C:\> Get-ChildItem | Where-Object { $_.Length -gt 1MB }
PS C:\> Get-ChildItem | Where-Object { $_.PSIsContainer }
PS C:\> Get-Service | Where-Object {$_.Status -eq "Running" -and $_.CanShutdown}
```

PowerShell v3.0+ (Single comparison operator only)

```
PS C:\> Get-ChildItem | Where-Object Length -gt 1MB
PS C:\> Get-ChildItem | Where PSIsContainer
PS C:\> Get-Service   | Where Status -eq Running
```

No compound conditions with simplified syntax

Filtering With Parameters vs. Where-Object

Filter output with Where-Object (~11 milliseconds)

```
PS C:\> Get-Process | Where-Object {$_.Name -match "net"}
```

vs.

Filter output with parameters (~4 milliseconds)

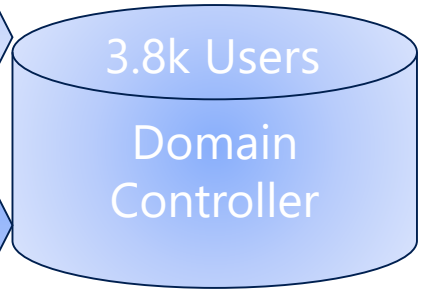
```
PS C:\> Get-Process -Name *net*
```

```
PS C:\> Get-ADUser -Filter * -Properties Surname | ?{$_.Surname -eq "Snover"}
```

Grab All Users as Objects and Perform Filtering (3.8k Users) ~2300 milliseconds

vs.

Filtering occurs on DC and matching objects are returned. (1 user) ~10 milliseconds




```
PS C:\> Get-ADUser -Filter {Surname -eq "Snover" }
```

PowerShell Proverb: If a Cmdlet has a –Filter there's a reason.

Automatic Member Enumeration

Retrieve single property from collection without using ForEach-Object

Single level

```
PS C:\> (Get-Process).ID
4300
8844
8812
```

Doesn't work in PowerShell v1.0 and v2.0

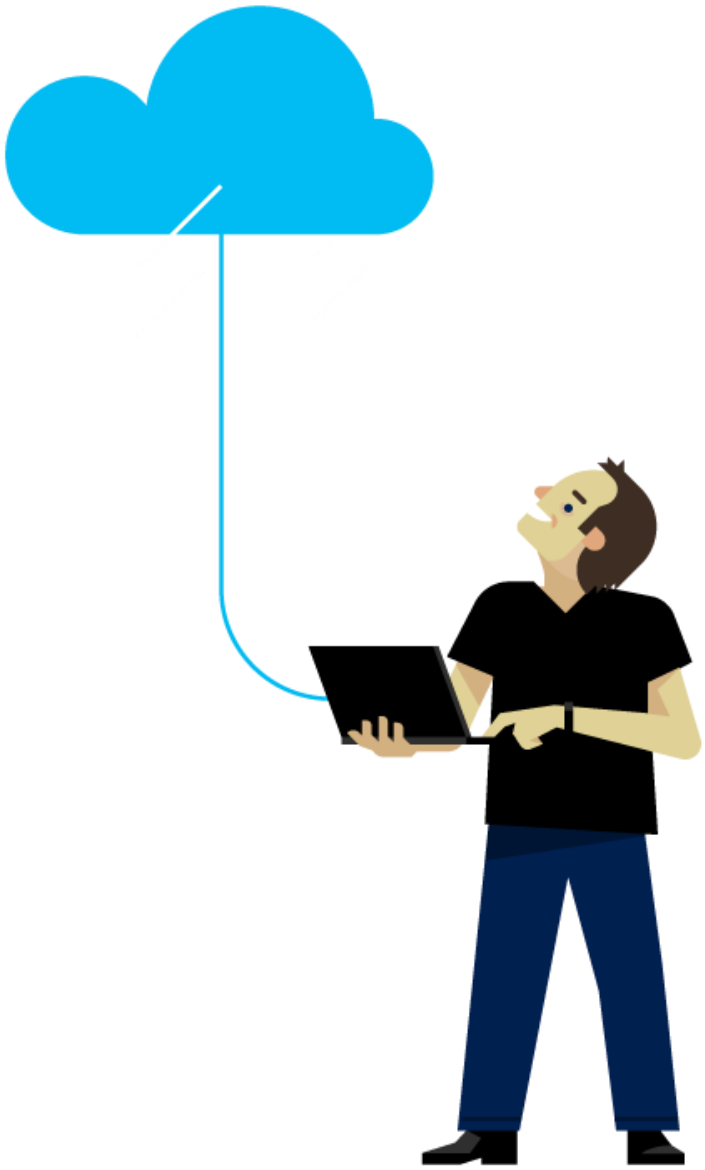
Multiple levels deep

```
PS C:\> (Get-EventLog -Log System).TimeWritten.DayOfWeek |
>> Group-Object

Count Name      Group
----- ----      -----
 4174 Tuesday   {Tuesday, Tuesday, Tuesday...}
 4349 Monday    {Monday, Monday, Monday...}
```

Demonstration

Foreach-Object and Where-Object

Questions?



Begin, Process and End Blocks

Foreach-Object - Anatomy

- Begin Block
  - Statements executed once, before first pipeline object
- Process Block
  - Statements executed for each pipeline object delivered
  - If called outside a pipeline context, block is executed exactly once
  - Default if unnamed
- End block
  - Statements executed once, after last pipeline object

Foreach-Object -Process Parameter

ForEach-Object often used with positional parameter in simple scenario

```
PS C:\>  Get-EventLog -LogName Application -Newest 5 |
ForEach-Object {$_.Message | Out-File -Filepath Events.txt -
Append}
```

Position 1 is -Process Parameter

```
PS C:\>  Get-EventLog -LogName Application -Newest 5 |
ForEach-Object -Process {$_.Message | Out-File Events.txt -
Append}
```

Parameter can be named

Begin, Process and End Parameters

ForEach-Object cmdlet supports Begin, Process, and End Parameters

Begin block → run once before any items are processed

```
PS C:\>  Get-EventLog -LogName Application -Newest 5 |
ForEach-Object
-Begin {Remove-Item .\Events.txt; Write-Host "Start" -ForegroundColor Red}
-Process {$_.Message | Out-File -Filepath Events.txt -Append}
-End {Write-Host "End" -ForegroundColor Green; notepad.exe Events.txt}
```

Begin, Process and End Parameters

ForEach-Object cmdlet supports Begin, Process, and End Parameters

Process block → run for each object on pipeline

```
PS C:\>  Get-EventLog -LogName Application -Newest 5 |
ForEach-Object
-Begin {Remove-Item .\Events.txt; Write-Host "Start" -ForegroundColor Red}
-Process {$_.Message | Out-File -Filepath Events.txt -Append}
-End {Write-Host "End" -ForegroundColor Green; notepad.exe Events.txt}
```

Begin, Process and End Parameters

ForEach-Object cmdlet supports Begin, Process, and End Parameters

End block → run once after all items have been processed

```
PS C:\>  Get-EventLog -LogName Application -Newest 5 |
ForEach-Object
-Begin {Remove-Item .\Events.txt; Write-Host "Start" -ForegroundColor Red}
-Process {$_.Message | Out-File -Filepath Events.txt -Append}
-End {Write-Host "End" -ForegroundColor Green; notepad.exe Events.txt}
```

Named Blocks in Functions/ScriptBlocks

- Statements can be in an unnamed block or in one or more named blocks
- Allows custom processing of collections coming from pipelines
- Can be defined in any order
- Use the Begin / Process / End blocks

Named Blocks

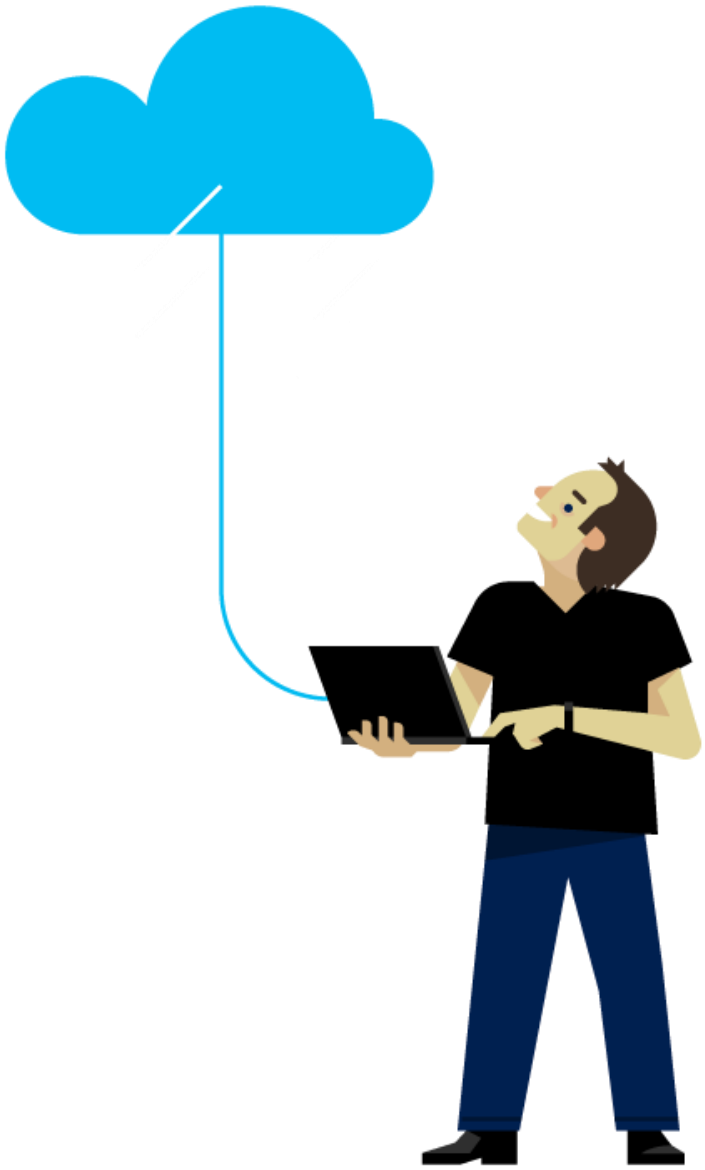
```
function My-Function
{
    Begin
    {
        Remove-Item .\Events.txt
        Write-Host "Start" -ForegroundColor Red
    }
    Process
    {
         $_.Message | Out-File -Filepath Events.txt -Append
    }
    End
    {
        Write-Host "End" -ForegroundColor Green
        notepad.exe Events.txt
    }
}
```

```
PS C:\> Get-EventLog -LogName Application -Newest 5 | My-Function
```

Demonstration

Begin Process End

Questions?



Two ways to accept pipeline input

Methods Of Accepting Parameter Pipeline Input

Cmdlet parameters can accept pipeline input in one of two ways:

- ByValue (Object Data Type)
- ByPropertyName (Object Property Name)

Cmdlet parameters may accept pipelined objects by value, by property name or both.

Does a Parameter Accept Pipeline Input?

```
PS C:\> Get-Help Restart-Computer -Parameter
ComputerName

 -ComputerName <String[]>

    Specifies one or more remote computers. The ...

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

Pipeline Input ByValue

For parameters that accept pipeline input ByValue, piped **objects** will bind:

- To a parameter of the same TYPE
- To a parameter that can be converted to the same TYPE

Pipeline Input ByValue

Restart-Computer ComputerName Parameter

```
PS C:\> Get-Help Restart-Computer -Parameter ComputerName

 -ComputerName <String[]>

    Specifies one or more remote computers. The default is ...

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

Pipe Computer names (strings) to Restart-Computer

```
PS C:\> 'MS','DC' | Restart-Computer -WhatIf

What if: Performing the operation "Restart the computer." on target "MS".
What if: Performing the operation "Restart the computer." on target “DC".
```

Pipeline Input ByPropertyName

For parameters that accept pipeline input ByPropertyName, piped objects **properties** will bind:

- To parameter(s) of the same name

Pipeline Input ByProperty

Restart-Computer ComputerName Parameter

```
PS C:\> Get-Help Restart-Computer -Parameter ComputerName

 -ComputerName <String[]>

 Specifies one or more remote computers. The default is ...

                    false
                    1
                    Local computer
                    True (ByValue, ByPropertyName)
                    false
```

ServerInfo.csv - Notepad

File Edit Format View Help

```
ComputerName,IPAddress,OSName
DC,192.168.1.1,'Windows Server 2016'
MS,192.168.1.3,'Windows Server 2016'
```

Ln 1, Col 1

```
PS C:\> Import-Csv .\ServerInfo.csv | Restart-Computer -WhatIf

What if: Performing the operation "Restart the computer." on target "DC".
What if: Performing the operation "Restart the computer." on target "MS".
```

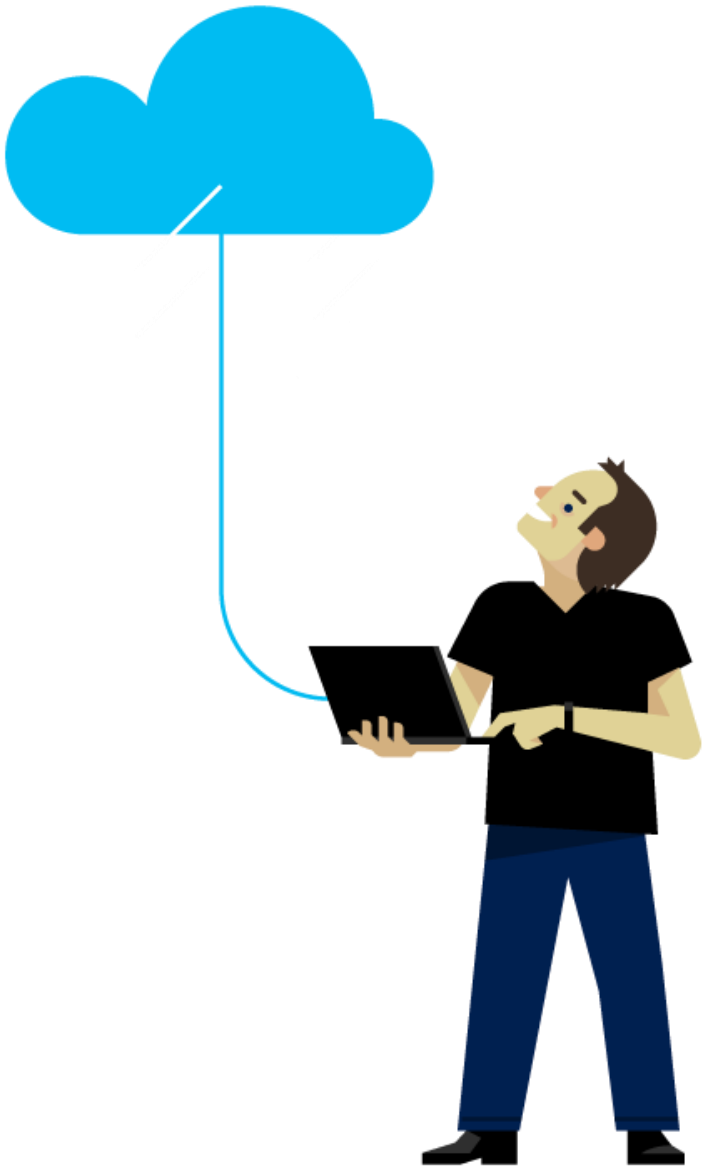
The Parameter Binding Steps

1. Bind all named parameters
2. Bind all positional parameters

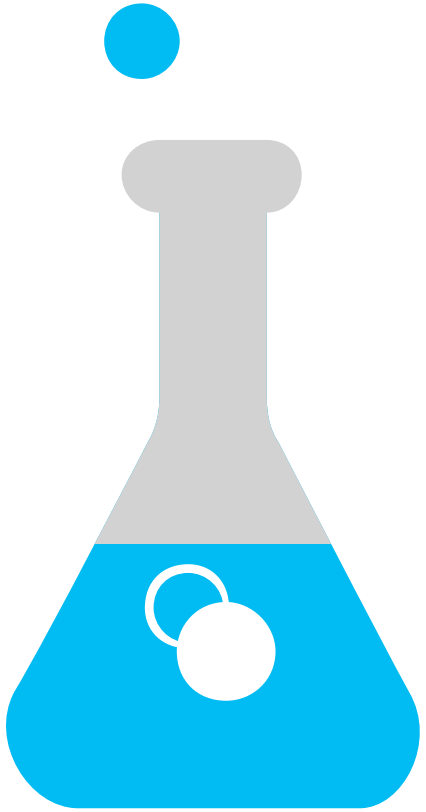
3. Bind from the pipeline **by value** with exact match
4. Bind from the pipeline **by value** with conversion

5. Bind from the pipeline **by name** with exact type match
6. Bind from the pipeline **by name** with type conversion

Questions?



Working With Pipelining



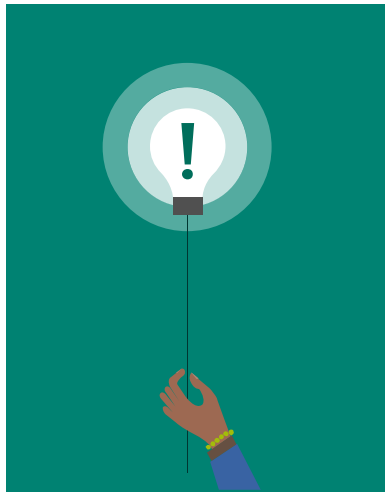
LAB

Different types of Operators

Objectives

After completing Different types of Operators, you will be able to:

- Work with advanced operators
- Work with text objects and manipulate them



Arithmetic Operators

Arithmetic Operators

- Arithmetic operators are mathematical functions that takes two operands and performs a calculation on them
- Arithmetic Operators work with more than only integer types

Arithmetic Operators

Operator	Description	Example(s)	Result(s)
+	Adds integers; concatenates strings, arrays, and hash tables	`6 + 2` `"file" + "name"`	8 filename
-	Subtracts values	`6 - 2`	4
-	Indicates negative value	`-6 + 2`	-4
*	Multiplies integers; copies strings and arrays	`6 * 2` `"ABC" * 3`	12 ABCABCABC
/	Divides values	`6/2`	3
%	Returns remainder of division (modulus)	`7%2`	1
-shl	Shift-left bitwise	`100 -shl 2`	400
-shr	Shift-right bitwise	`100 -shr 1`	50

100 in binary is 1100100
-shl shifts digits n chars left
400 in binary is 110010000

100 in binary is 1100100
-shr shifts digits n chars right
50 in binary is 110010

Rounding Integers

Integers get rounded to the closest whole number

If the value starts with 0.5 rounding happens to the nearest even number by default

Additional rounding options available if needed

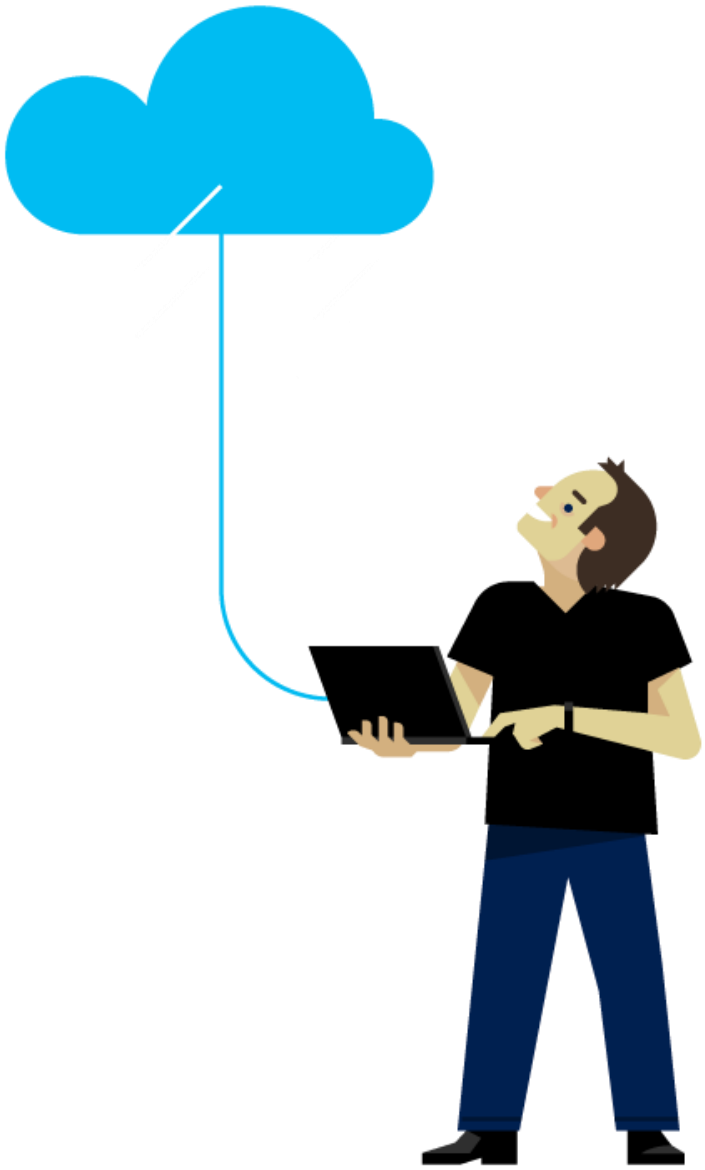
```
PS C:\>[int](5/2)
2
PS C:\>[int](7/2)
4
```

```
PS C:\>[int](12.5)
12
PS C:\>[int](13.5)
14
```

Demonstration

Arithmetic Operators

Questions?



Assignment Operators

Assignment Operators

- An assignment operator assigns a value to its left operand based on the value of its right operand.

Assignment Operators

Operator	Description	Example(s)
=	Sets variable	`$integer = 100`
+=	Increases variable	`$integer += 1`
-=	Decreases variable	`$integer -= 1`
*=	Multiplies variable	`$integer *= 2`
/=	Divides variable	`$integer /= 2`
%=	Divides variable and assigns remainder to variable	`$integer %= 3`
++	Unary Operator. Increases variable by 1	`$integer++`
--	Unary Operator. Decreases variable by 1	`$integer--`

All produce same result

```
$integer = $integer + 1
$integer += 1
$integer++
```

Unary Operators Pre And Post Processing

Pre processing will increment the value before being used

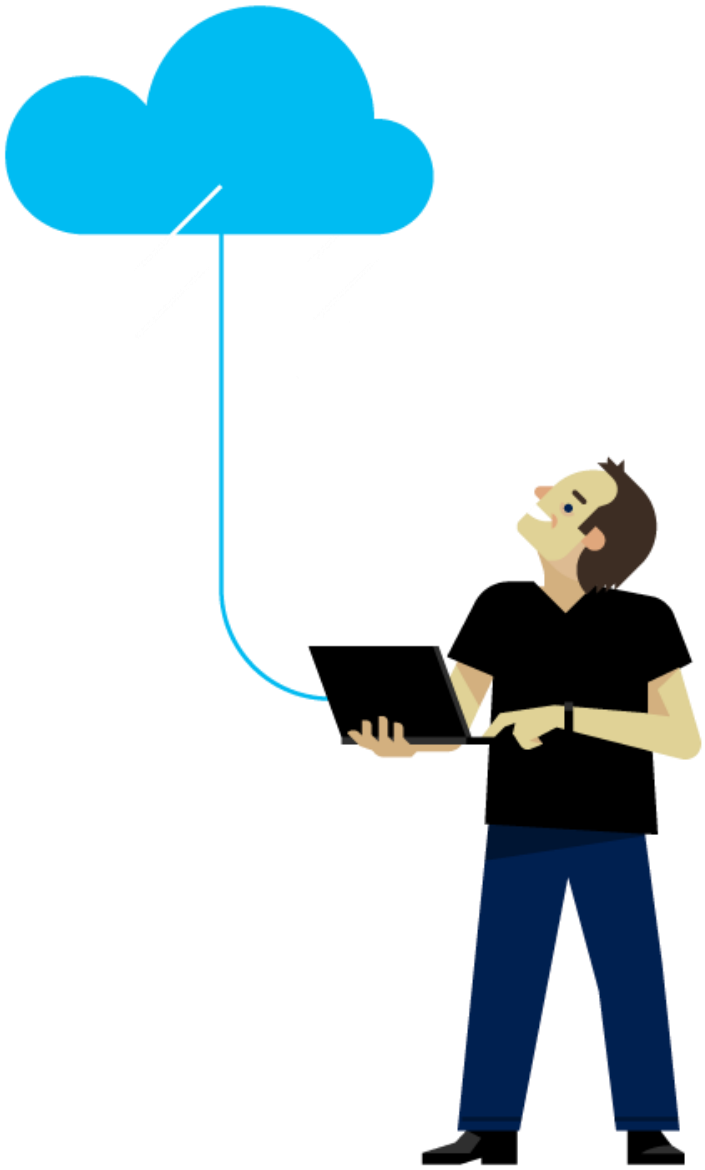
Post processing will only increment the value after being used

```
PS C:\> $1 = 5
PS C:\> $1
5
PS C:\> write-output -InputObject (++$1)
6
PS C:\> write-output -InputObject ($1++)
6
PS C:\> $1
7
```

Demonstration

Assignment Operators

Questions?



Binary Operators

Bitwise Operators

- Bitwise operators act on the binary format of a value at the level of their individual bits.
- Can be used to calculate if a bit is set. (Example User-account-control in AD)

Bitwise Operators

Operator	Description	Example(s)	Binary Format
-bAnd	Bitwise AND	`10 -band 3`	1010 (10) 0011 (3) -------- bAND 0010 (2)
-bOr	Bitwise OR (inclusive)	`10 -bor 3`	1010 (10) 0011 (3) --------- bOR 1011 (11)
-bXor	Bitwise OR (exclusive)	`10 -bxor 3`	1010 (10) 0011 (3) ------- bXOR 1001 (9)
-bNot	Bitwise NOT	`-bnot 10`	0000 1010 (10) ---------- bNOT 1111 0101 (-11)

Split, Join and replace Operators

Split, Join, and Replace Operators

- The Split, Join, and Replace operators are used to manipulate text strings

Split Operator

Description	Example(s)

Unary split operator:
-split <string>
 	
Note:
• Example splits on space as delimiter
• Only splits the first string
	`PS C:\> -split "1 a b"`
	`1`
	`a`
	`b`

Binary split operator:
<string> -split <delimiter>
 	
Note:
• Example splits on comma as delimiter
	`PS C:\> "1,a b" -split ","`
	`1`
	`a b`

The binary -Split operator uses a Regular Expression for the delimiter

Join Operator

Description	Example(s)
Unary join operator: -join <string[]>	`PS C:\> -join ("a", "b", "c")` `abc`

Binary join operator:
String[]> -
Join<Delimiter>
	`PS C:\> "Windows", "PowerShell", "4.0" -join [char]2`
	`Windows╚PowerShell╚4.0`
	
	`PS C:\> "How", "are", "you", "doing?" -join " "`
	`How are you doing?`

Replace Operator

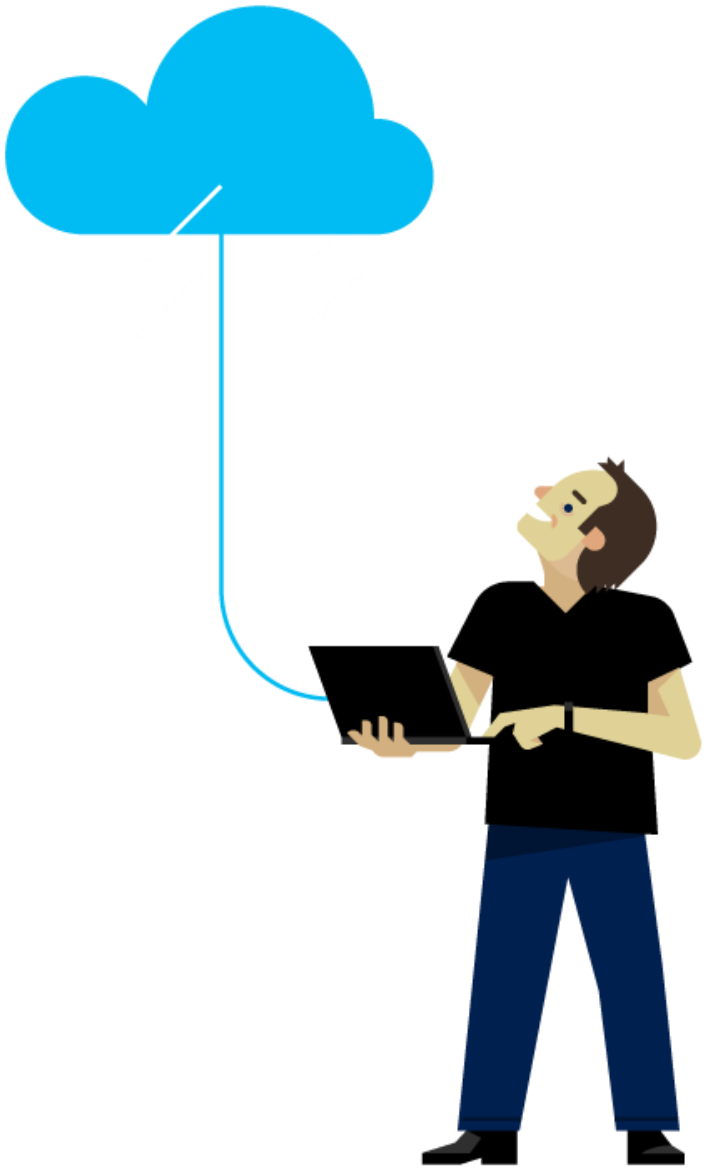
Description	Example(s)
Binary replace operator: <String[]> -Replace <Delimiter>	`PS C:\> "Windows PowerShell 4.0" -replace "4.0","5.0"` `Windows PowerShell 5.0`

The -Replace operator uses a Regular Expression for the delimiter

Demonstration

-Split , -Join and -Replace

Questions?



Format Operator

Format Operator (-f)

- Formats strings by using the format method of string objects
- Format string on the left side of the operator
- Objects to be formatted on the right side of the operator
- Format specifiers enable the value to take multiple forms

Format Operator – Index

Index Positions

Ordered using index

0 1 2 3 4

```
PS C:\> "{1}{0}{2}{0}{3}{0}{4}" -f "-","Text","separated","with","dash“

Text-separated-with-dash
```

{ index[,alignment][:formatString] } –f “string(s)” , “to be formatted”

Format Operator – Variations

```
PS C:\> $MyArray = 'Smith','John',123.456

PS C:\> “Custom Text" -f $MyArray
Custom Text

PS C:\> "First name is: {1}  Last name is: {0}" -f $MyArray
First name is: John  Last name is: Smith

PS C:\> "{2}" -f $MyArray
123.456

PS C:\> "Using a Format Specifier {2:N1}" -f $MyArray
Using a Format Specifier 123.5
```

Format Operator – Alignment

Forced Width
(negative-left justified, positive-right justified)

```
PS C:\> "{1}{0,5}{2}" -f "-","Text","separated","with","dash“

Text    -separated
```

{ index[,alignment][:formatString] } –f “string(s)” , “to be formatted”

Format Operator – FormatString

Format as percentage

```
PS C:\> "{0:p} {1:p} {2:p}" -f 0.3,0.56,0.99

30.00% 56.00% 99.00%
```

{ index[,alignment][:formatString] } –f “string(s)” , “to be formatted”

Format Operator – FormatString Cont.

Round number to 2 decimal places

```
PS C:\> "{0:n2}" -f 3.1415926

3.14
```

{ index[,alignment][:formatString] } –f “string(s)” , “to be formatted”

List of Valid Format Strings

Conversions:

Format use	Explanation
:c	Currency format (for the current culture)
:d	Decimal. leading zeroes are added to the beginning of the number if needed.
:e	Scientific (exp) notation
:f	Fixed point :f5 = fix to 5 places
:g	Most compact format, fixed or sci :g5 = 5 significant digits
:n	Number, includes culture separator for thousands 1,000.00
:p	Percentage
:r	Reversible Precision
:x	Hex format

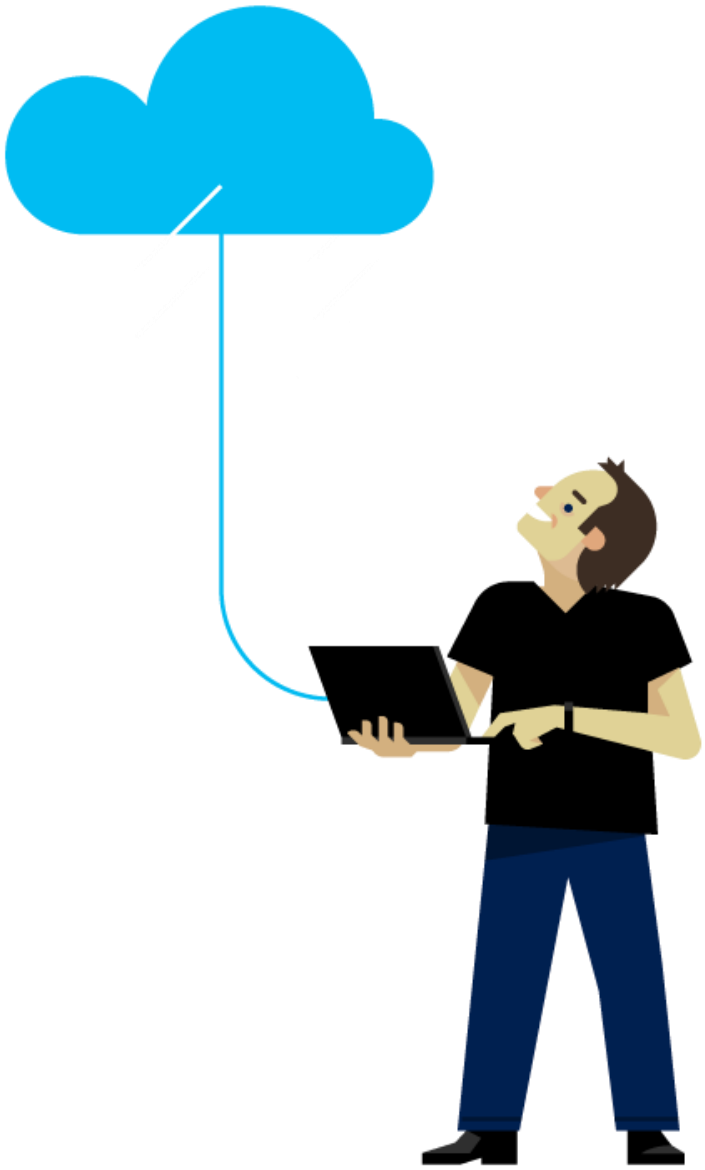
Date / Time:

Format use	Explanation
:hh :mm :ss	Convert a DateTime to a 2 digit Hour/minute/second "{0:hh}:{0:mm}"
:HH	Hour in 24 Hour format
:dd	Day of Month
:MM	Month of year
:ddd	Convert a DateTime to Day of the Week
:dddd	Full name of Day of Week
:yyyy	Full year
#	Digit Place Holder

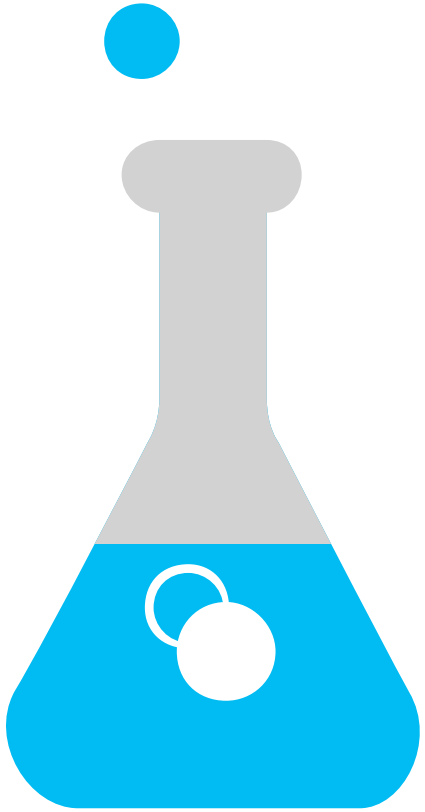
Demonstration

Formatting With -f

Questions?



Operator Types



LAB

Microsoft