Microsoft

WorkshopPLUS - Windows PowerShell: Foundation Skills

Microsoft Services

Microsoft

Windows PowerShell Basics

Microsoft Services

Learning Units covered in this Module

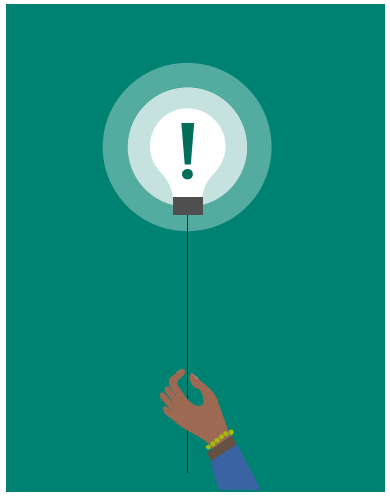
- Introduction to Windows PowerShell
- Introduction to Commands

Introduction to Windows PowerShell

Objectives

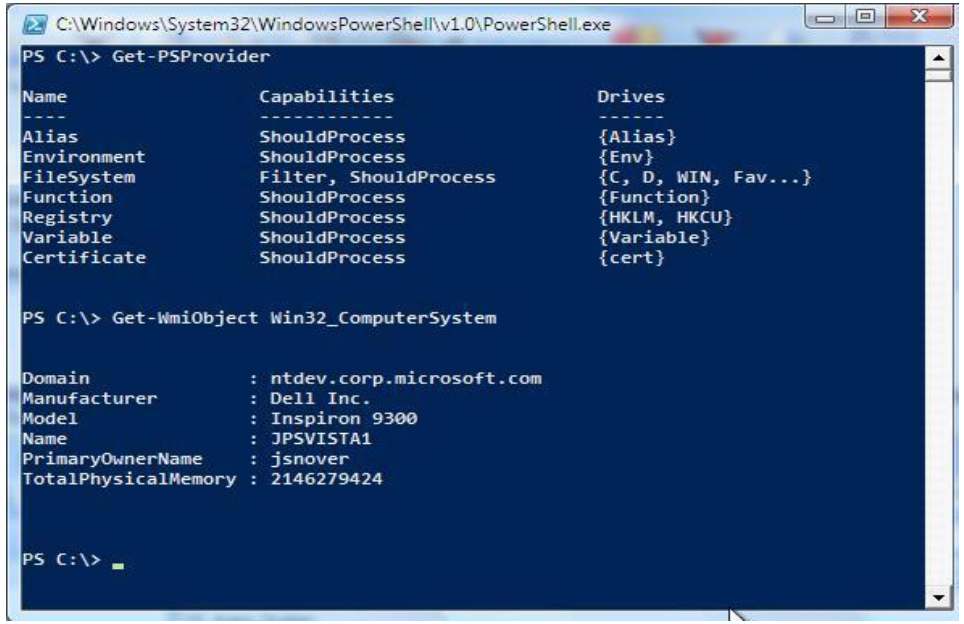
After completing Introduction to Windows PowerShell, you will be able to:

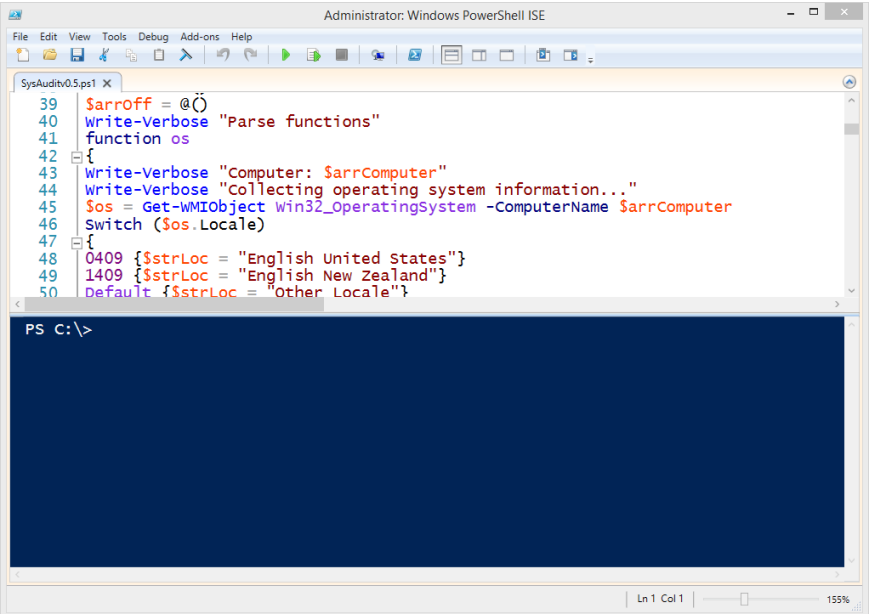
- -Understand the system requirements for PowerShell
- -Understand the PowerShell shell
- -Understand the Integrated Scripting Environment



PowerShell Introduction

What is PowerShell?





Management Shell:

- Automation Engine
- Scripting Language

Development Framework:

- Integrated Scripting Environment
- PowerShell Embedded in Host Applications

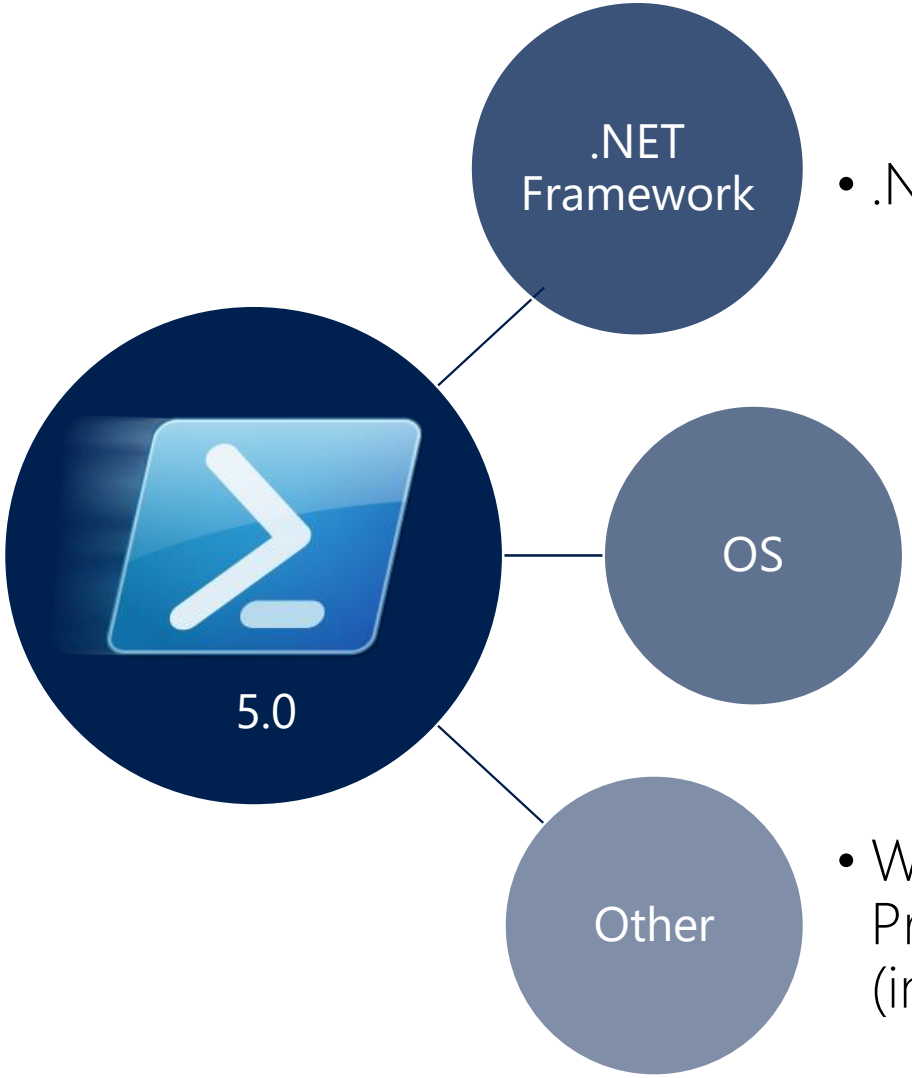
PowerShell Evolution

Code Name: Monad	1.0	2.0	3.0	4.0	5.0	6.0 Core
2005	2006	2008	2012	2013	2015	2018
	130 cmdlets	230 cmdlets Backward-Compatible Integrated Shell Environment (ISE) Remoting	>2,300 cmdlets Backward-Compatible WinPE Web Access Enhanced ISE Workflow	>2,300 cmdlets Backward-Compatible Desired State Configuration (DSC)	Package Management PowerShell Get CMS module Remote debugging DSC additions	Cross platform: Windows Linux Mac OS Docker support SSH remoting Pipeline commands in background

PowerShell Default Availability

Windows PowerShell is a Windows feature	
Windows PowerShell 5.0	• Windows 10 • Windows Server 2016 / Windows Server 2019
Windows PowerShell 4.0	• Windows 8.1 • Windows Server 2012R2
Windows PowerShell 3.0	• Windows 8 • Windows Server 2012
Windows PowerShell 2.0	• Windows 7 • Windows Server 2008 R2
Windows PowerShell 1.0	• Windows Server 2008

System Requirements

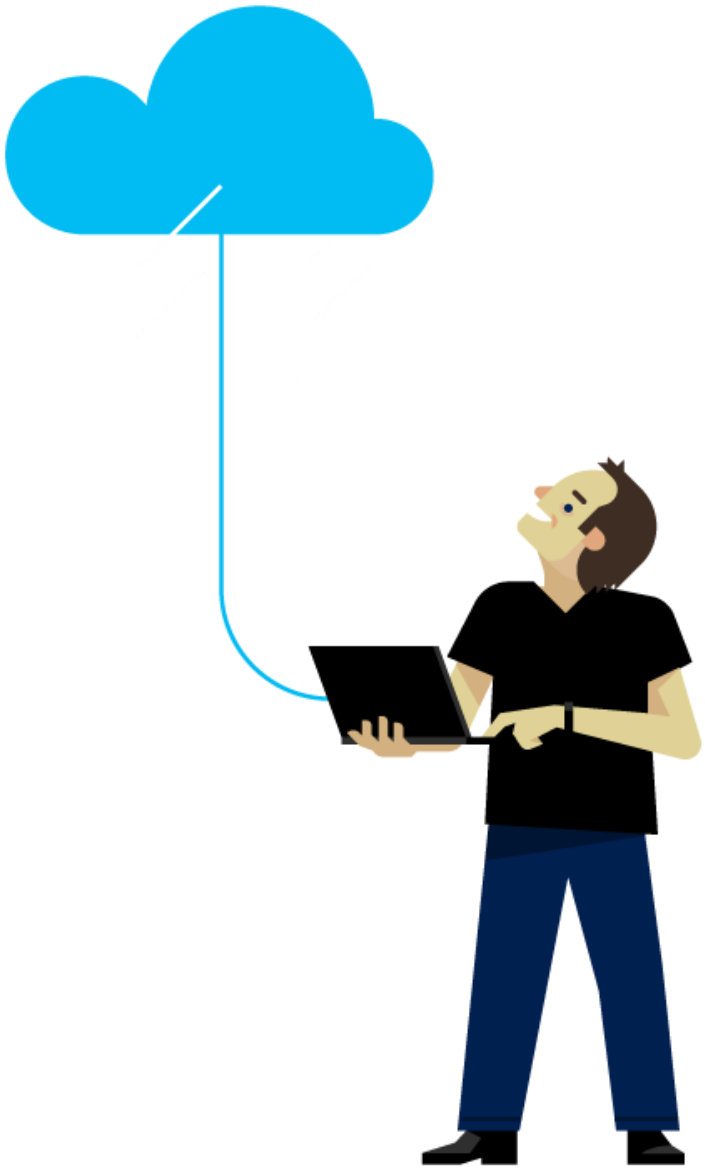


- .NET Framework v4.5

- Windows Server 2016
- Windows 10
- Windows Server 2012 or 2012 R2
- Windows 8.1
- Windows Server 2008 R2 SP1
- Windows 7 SP1

- WMI, WinRM and Server Manager CIM Provider
  (included in Windows or WMF)

Questions?



The PowerShell shell

Command-Line Interface (CLI)

- Interactive mode
- Simple commands to interact with applications and the operating system
- Handy shortcut keys: HOME, END, Arrows, CTRL+arrows, CTRL + Space

Windows PowerShell

```
PS C:\> Get-Process

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
     83       7     1112       4224    45             2296 armsvc
    184      12     2248       7664    92             1228 atieclxx
    109       6      776       3460    23             1012 atiesrxx
    461      19     5056      16564   110     0.80   6020 BasisSync
    182      13     2236       8200    89     0.11   4808 BDAppHost
    142      11     1948       7076    76     0.13   4452 BDExtHost
    315      22    11332      20060   122     0.67   6784 BDRuntimeHost
    402      30     7780      26080   207     0.63   6320 BingDesktop
   1101      89    91144       7712   859     7.02   6504 CCC
    890      43    19424      49124   158            5668 CcmExec
     51       6      904       4068    53     0.00   3420 conhost
    107      10     3232      10084   101     1.06   4216 conhost
    376      24    45200      55396   365     0.58   7356 CSISYNCCLIENT
    416      15     1824       4200    50              616 csrss
    507      28     2300       6208   134              728 csrss
    129       5      984       3988    18             2380 dasHost
```

PS Readline

Can be installed on versions of PowerShell 3.0 and greater, but is now built in to Windows 10 / Server 2016 and within Windows Management Framework (WMF) 5.0 and above.

PS Readline adds some distinct features to the console:

- Syntax coloring
- Simple syntax error notification
- Multi-line experience
- Customizable key bindings
- Cmd and emacs modes (preview)
- Bash style completion
- history search (CTRL-R)
- PowerShell token based "word" movement and kill
- Undo/redo
- Automatic saving of history across live sessions
- "Menu" completion via Ctrl + Space

Commandline IntelliSense and History

IntelliSence:

- Dynamically suggests code and provides help as you type
- Keyboard-based tab completion
- CTRL + Space provides menu-based completion

History:

- Preserved between hosts
- Use Get-history and recall
- History searches via CTRL+R & CTRL +S

History

Use “Get-history” to view you command history

Invoke a command from the history

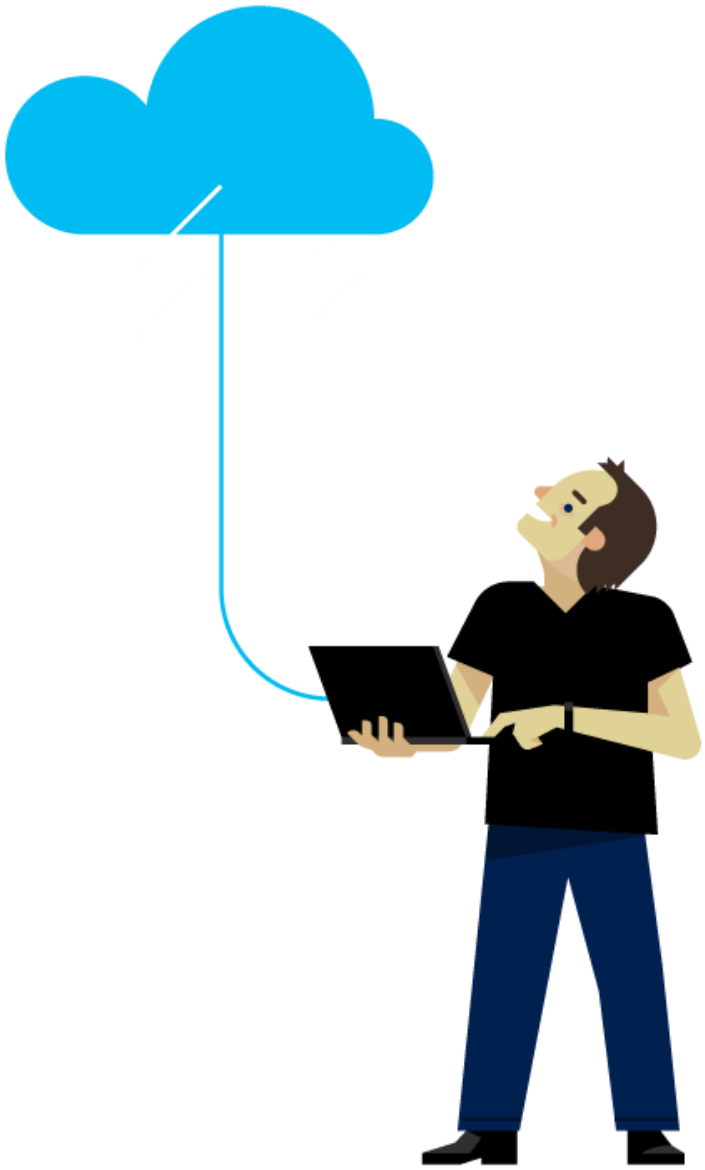
Windows PowerShell

```
PS C:\> Get-History

  Id CommandLine
  -- -----------
   1 get-help
   2 get-command
   3 get-host
   4 Get-Module


PS C:\> Invoke-History -Id 4
Get-Module

ModuleType Version    Name
---------- -------    ----
Manifest   3.1.0.0    Microsoft.PowerShell.Management
Manifest   3.1.0.0    Microsoft.PowerShell.Utility
Script     1.2        PSReadline
```

Demonstration

PS Readline

Questions?

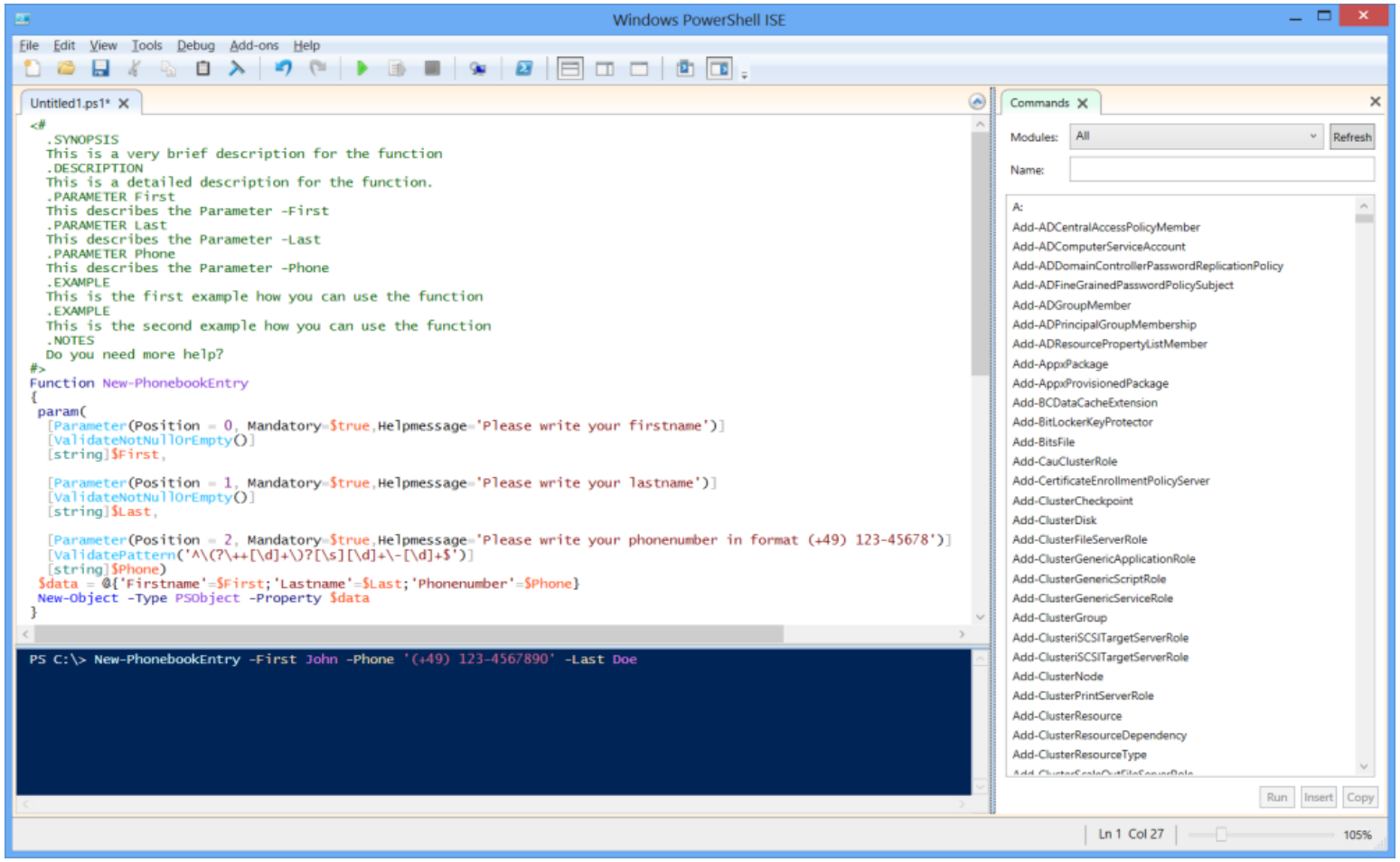


Interactive Scripting Environment (ISE) - Basics

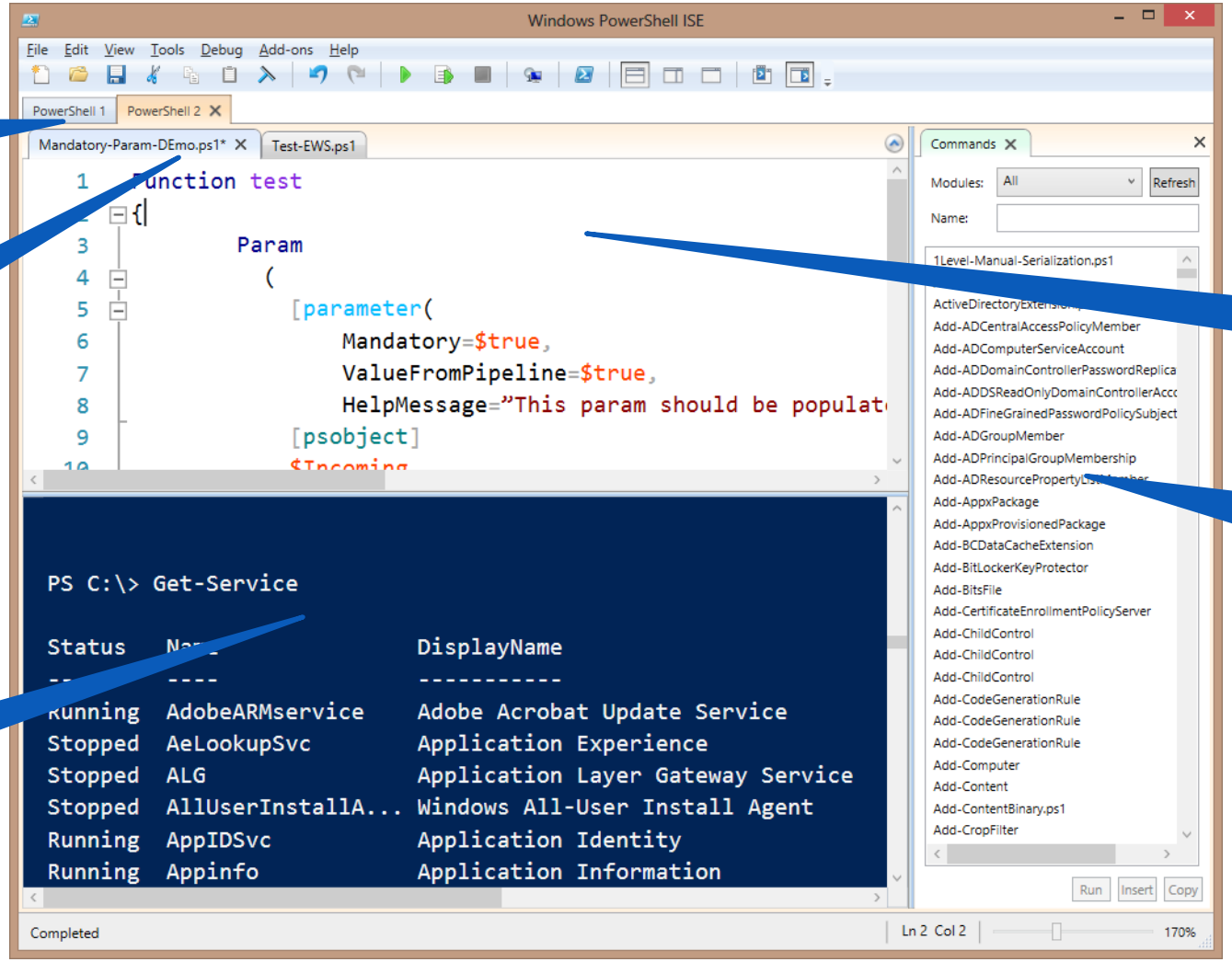
Integrated Scripting Environment (ISE)

ISE can be used as:

- Development Tool
- Graphical Editor
- Execution of code
- Debugging
- PowerShell remoting



Anatomy of the ISE



PowerShell tabs

Scripts open within a tab

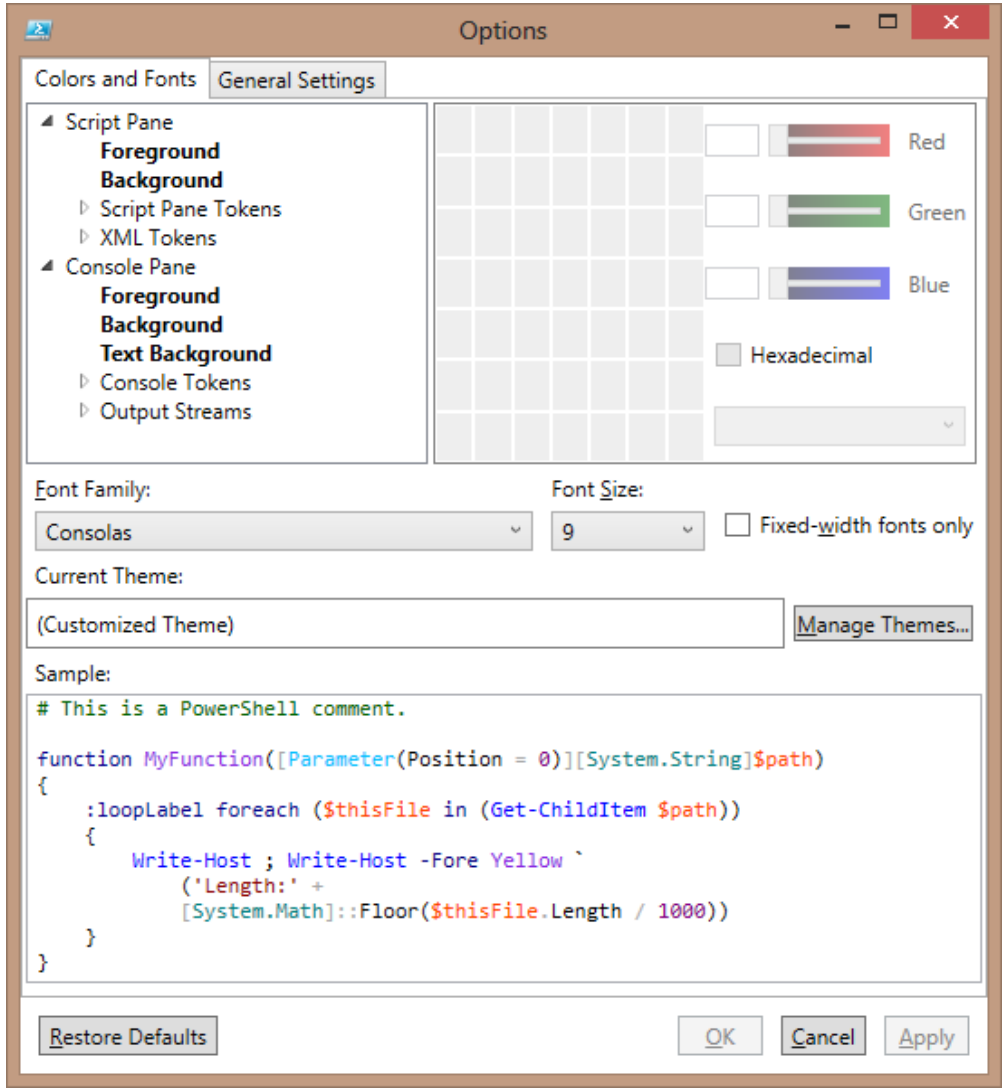
Console pane

Script pane

Show-Command add-on

Syntax Color Highlighting

- ISE includes enhanced syntax highlighting
- Color highlighting is automatic and customizable
- Tools-Options window (shown), includes detailed token, stream, and console colorization settings
- Themes to make common color sets easy to use



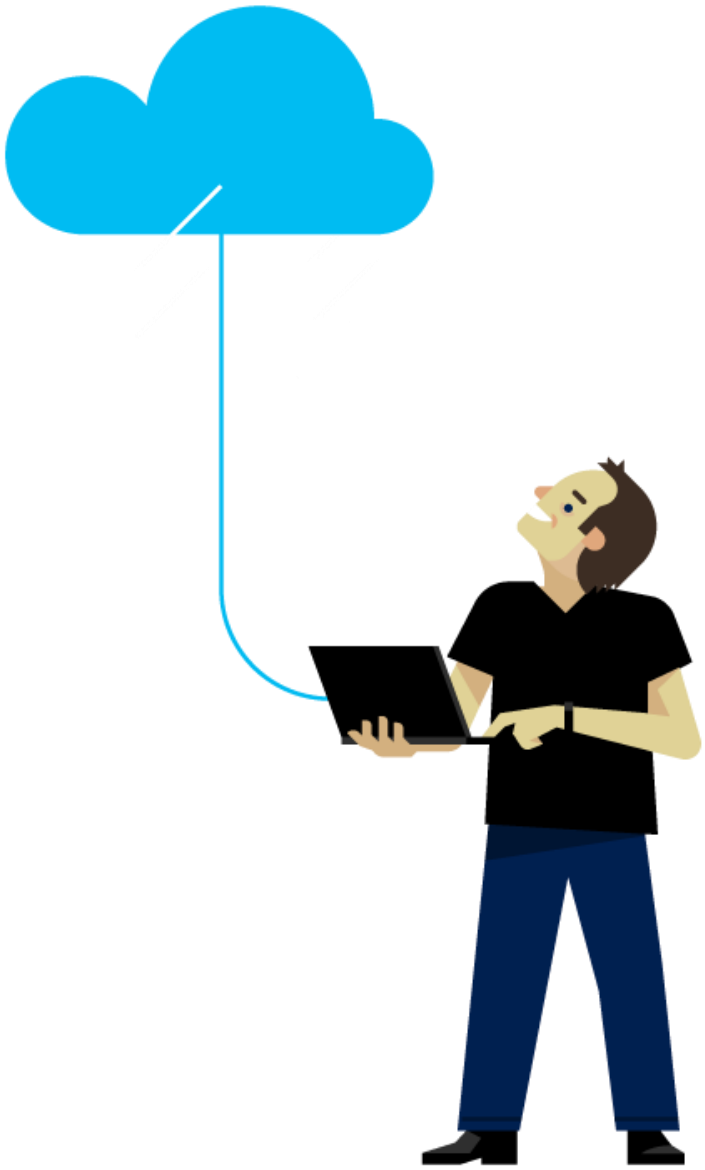
ISE Compiled Add-Ons

- Compiled add-ons allow for rich functionality to be created, such a **variable watch window**
- Compiled add-ons are **WPF-based controls**
- The built-in **Show-Command Add-On** is a good example of a compiled add-on
- Look for compiled add-ons coming from the PowerShell community and informally from Microsoft

Demonstration

Integrated Scripting Environment (ISE)

Questions?



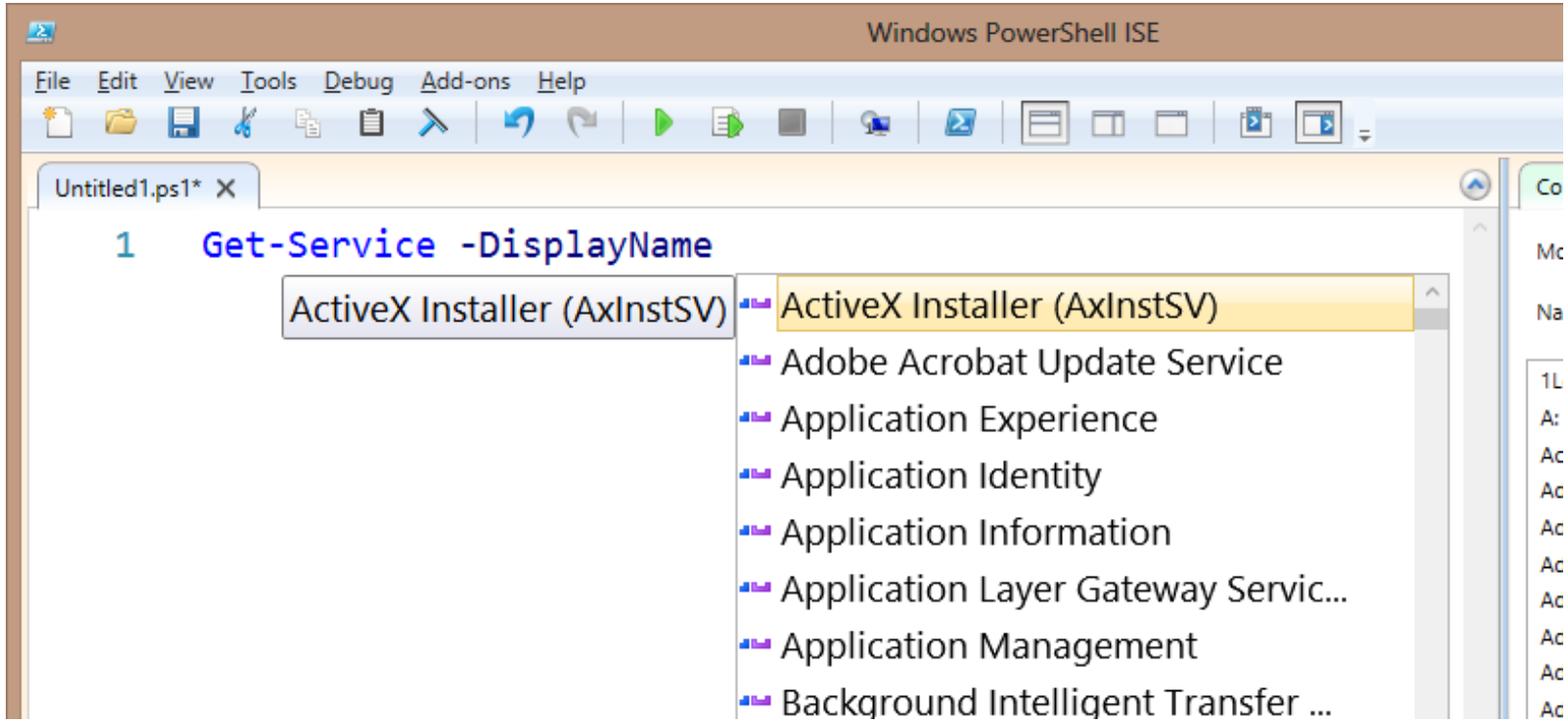
Interactive Scripting Environment (ISE) - Features

ISE IntelliSense

- Similar to Visual Studio IntelliSense
- Dynamically suggests code and provides help as you type
- Keyboard-based tab completion still works
- Mouse or keyboard can be used to leverage IntelliSense popups
- Works in Script and Command Pane

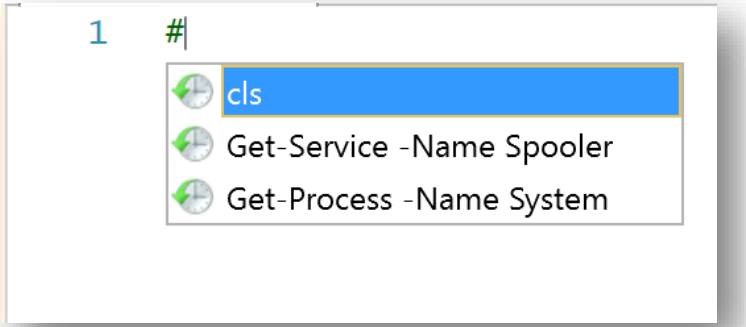
IntelliSense Parameter Arguments

Some cmdlet parameters display parameter values



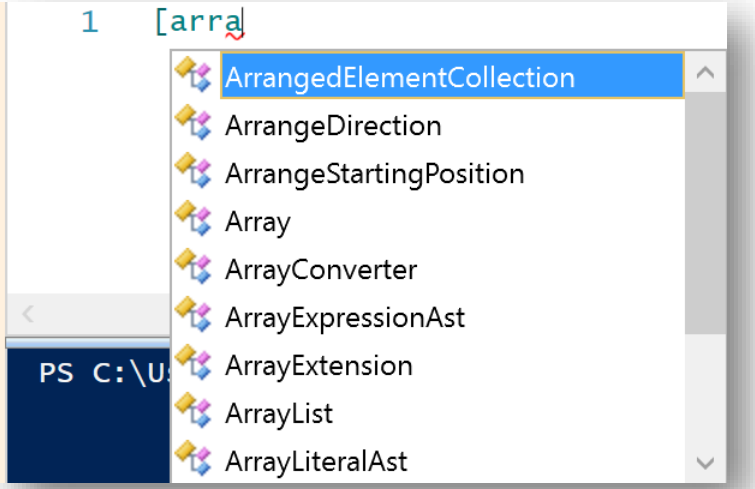
History

- Typing # followed by Ctrl + Space shows your command history at a glance.



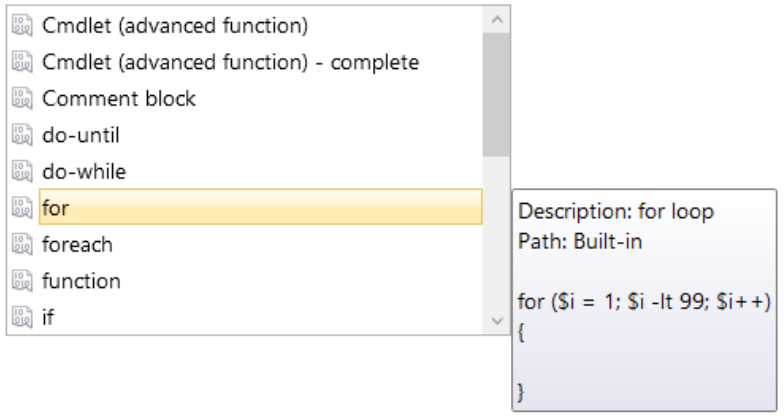
Types and Namespaces

- Typing "**[arra**" followed by **Ctrl** + **Space** shows types and namespaces in drop-down.



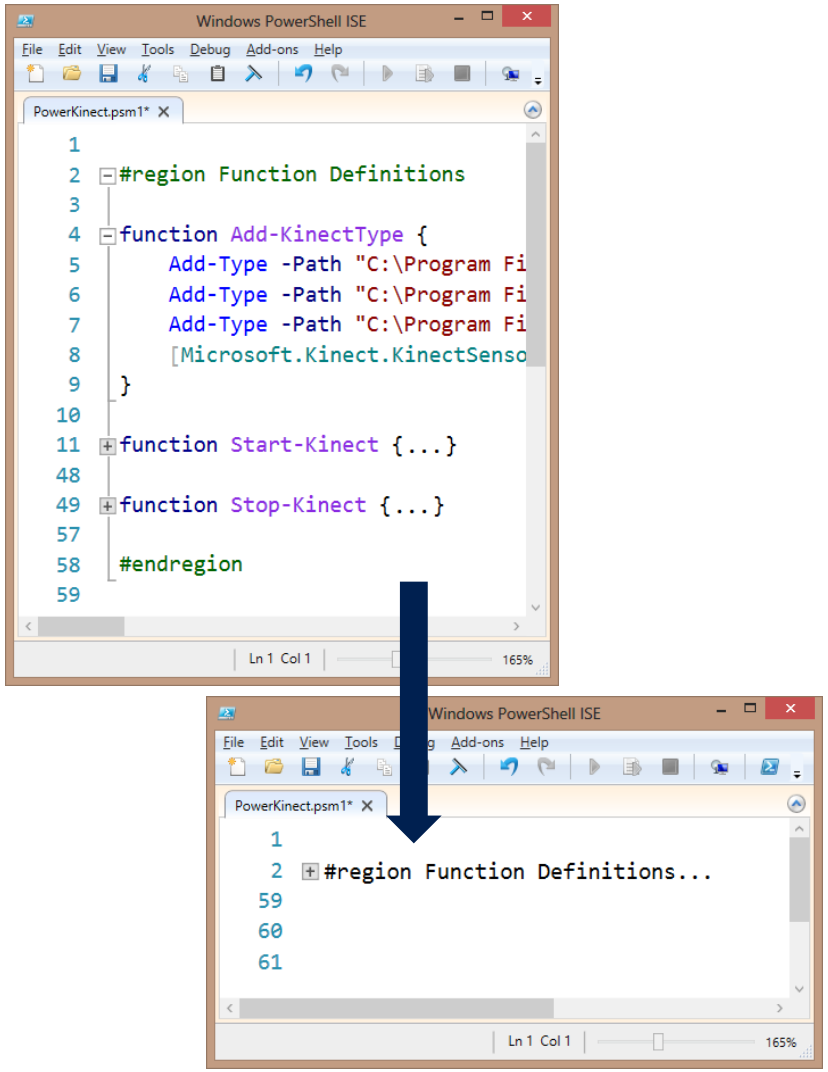
ISE Code Snippets

- Pre-created blocks of code
- Inserted at cursor
- Three kinds of snippets:
  - Default (included with ISEv3/4)
  - User-Defined
  - Module-Based
- Keyboard Shortcut - Ctrl-J
- Edit Menu, “Start Snippets”



Collapsable Code

- Functions, Script block, Parenthesis, Quotes, etc. can be collapsed spanning multiple lines
- Manual code regions allow collapsing between any two lines
  - #region – Begin a Region
  - #endregion – End a Region
  - Optional text following the #region tag can help with code documentation
  - When all regions are collapsed they should tell the story of the script like a book index



Brace Matching

ISE script pane will highlight matching braces when cursor positioned outside

Works with: { Curly Braces }, (Parentheses), [Square Brackets]

Find paired braces with ISE menu: Edit – Go to Match (CTRL-])

```
1   Get-Process |
2       ForEach-Object `
3       {$_.Threads | ForEach-Object{$TotalThreadCount++}}
4
```

Note: Subtle grey highlighting when cursor in front of curly brace

```
1   Get-Process |
2       ForEach-Object `
3       {$_.Threads | ForEach-Object{$TotalThreadCount++}}
4
```

Auto-Save and Crash Recovery

- Auto-Save
  - ISE automatically saves scripts to ‘alternate location’
  - Default save interval is 2 minutes
  - Interval is editable via menu and object model
- Crash Recovery
  - Uses alternate auto-saved files to restore un-saved scripts

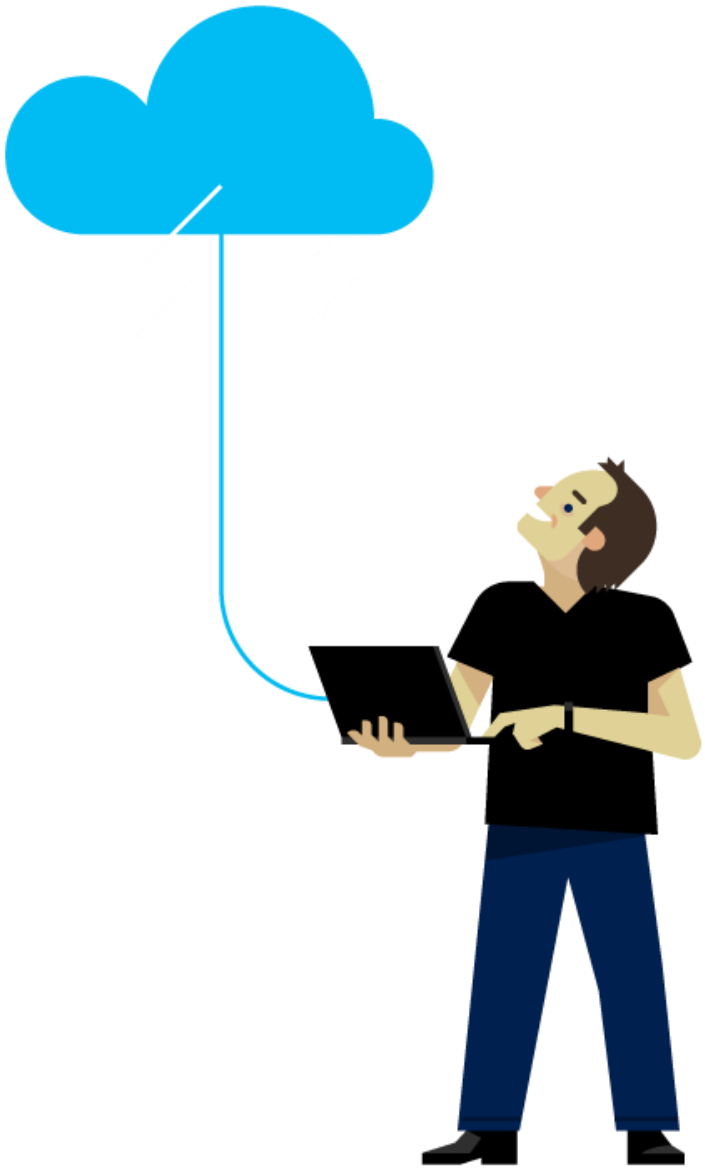
Rich Copy to and from the ISE

- Clipboard includes colors, font, size, etc.
- Excellent feature for sharing code in email, pptx, docx, etc.

Demonstration

ISE Features

Questions?

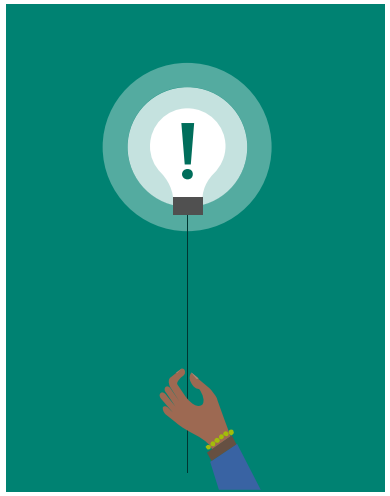


Introduction to Commands

Objectives

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

- Search and discover commands
- Understand how to get help
- Work with aliases



External Commands

External Commands

- Traditional command line tools like sc.exe, netsh.exe, reg.exe can still be used in PowerShell
- These commands run in a separate process
- Commands can be found using “Get-Command”
- Difficult to discover due to no standard naming convention or syntax
- Output of the commands is in a plain text array or lines

Searching for External Commands

- External commands can be found using “Get-Command”

```
PS C:\> Get-Command -Name *.exe

CommandType     Name                                               Version    Source
-----------     -------                                            ---------  --------
Application     AgentService.exe                                   10.0.17... C:\WINDOWS\system32\AgentService.exe
Application     aitstatic.exe                                      10.0.17... C:\WINDOWS\system32\aitstatic.exe
Application     alg.exe                                            10.0.17... C:\WINDOWS\system32\alg.exe
Application     AppHostRegistrationVerifier.exe                    10.0.17... C:\WINDOWS\system32\AppHostRegistratio...
```

External Commands VS Native PowerShell

- Lots of native command have a PowerShell equivalent

CMD	PowerShell
Ping Loopback –a –n 1	Test-Connection –count 1
Copy c:\SomeFile.txt d:\somefile.txt /Y	Copy-Item c:\SomeFile.txt d:\SomeFile -Force
Netsh interface ip show config	Get-NetIPAddress

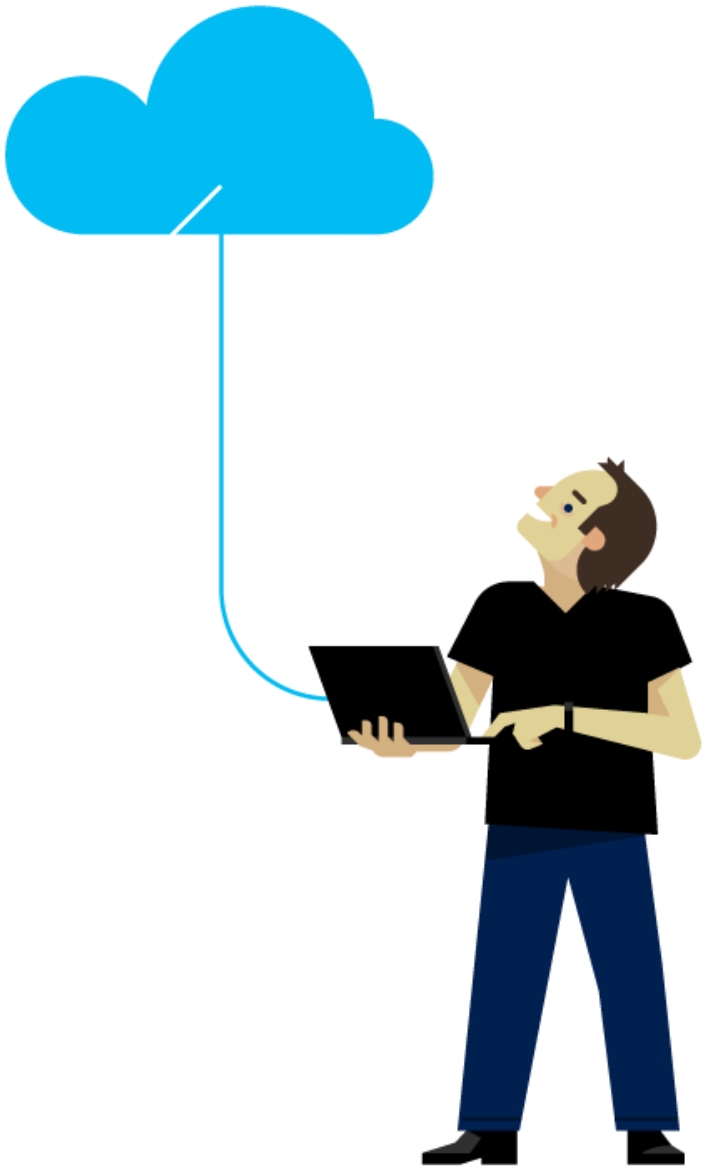
- Data is returned in an unstructured array of text strings

```
PS C:\> $result = netsh interface ip show config
PS C:\> $result.GetType().basetype.name
Array
PS C:\> $result[2..3]
    DHCP enabled:                Yes
    IP Address:                  192.168.3.101
PS C:\>
```

Demonstration

External Commands

Questions?

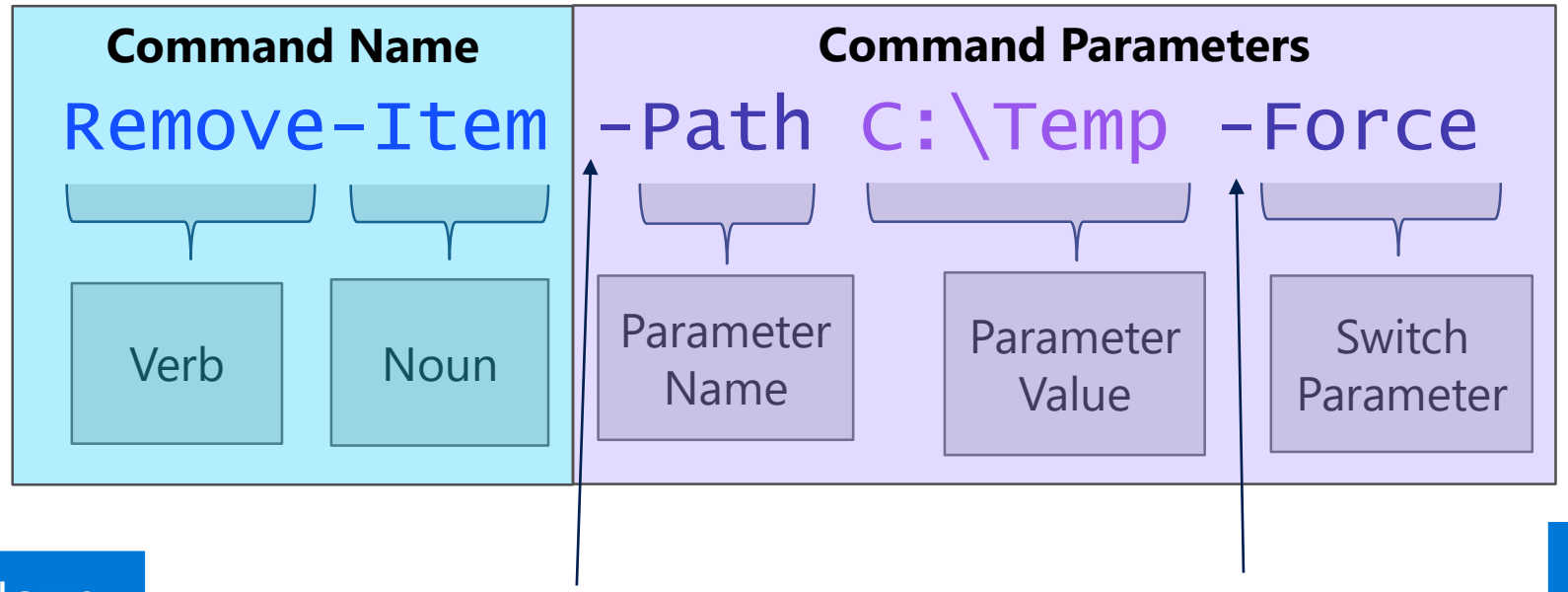


PowerShell commands

What is a Cmdlet?

Parameters to control Cmdlet behaviour

Native PowerShell command



Verb-Noun Naming

Dashes Precede all Parameter Names

Does not Launch in Separate Process

Cmdlet Examples

```
PS C:\> Get-Process

 Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
      83       7     1084       4124    45     0.09   7784 armsvc
     179      13     1892       8216    89     0.66   6540 BDAppHost
     143      12     1840       7320    76     0.22  11148 BDExtHost
...
```

```
PS C:\> Restart-Service -Name Spooler -Verbose

VERBOSE: Performing the operation "Restart-Service" on target "Print
Spooler (Spooler)".
```

```
 PS C:\> Test-Connection -ComputerName 2012R2-MS -Count 1 -Quiet
True
```

Get-Command

- Discover Commands (cmdlets, functions, scripts, aliases)
- Can show command syntax
- Can also discover external commands (.exe, .cpl, .msc)

Get-Command

```
PS C:\> Get-Command

CommandType     Name                    Definition
------------    ----                    -----------
Cmdlet          Add-Content             Add-Content [-Path] String[]...
Cmdlet          Add-History             Add-History [[-InputObject] ...
Cmdlet          Add-Member              Add-Member [-MemberType] <PS...
Function        Clear-Host              $space = New-Object System.A...
Alias           dir -> Get-Chil...
...
```

Wildcard in Name

```
PS C:\>  Get-Command -Name *user*

CommandType     Name
-----------     ----
Function        UpdateDefaultPreferencesWi...
Cmdlet          Get-WinUserLanguageList
Cmdlet          New-WinUserLanguageList
Cmdlet          Set-WinUserLanguageList
Cmdlet          Test-UserGroupMembership
Application     DsmUserTask.exe
Application     quser.exe
Application     UserAccountBroker.exe
Application     UserAccountControlSettings...
Application     userinit.exe
```

List Cmdlets by Verb

```
PS C:\> Get-Command -Verb Get

CommandType       Name                          ModuleName
-----------       ----                          -----------
Alias             Get-GPPermissions             GroupPolicy
Alias             Get-ProvisionedAppxPackage    Dism
Function          Get-AppBackgroundTask         AppBackgroundTask
...
```

List Cmdlets by Noun

```
PS C:\> Get-Command -Noun Service

CommandType     Name              ModuleName
-----------     ----              -----------
Cmdlet          Get-Service       Microsoft.PowerShell.Management
Cmdlet          New-Service       Microsoft.PowerShell.Management
Cmdlet          Restart-Service   Microsoft.PowerShell.Management
Cmdlet          Resume-Service    Microsoft.PowerShell.Management
...
```

List Cmdlets Only

```
PS C:\> Get-Command -CommandType Cmdlet

CommandType   Name                                  ModuleName
------------  ----                                  -----------
Cmdlet        Add-ADCentralAccessPolicyMember       ActiveDirectory
Cmdlet        Add-ADComputerServiceAccount          ActiveDirectory
...
```

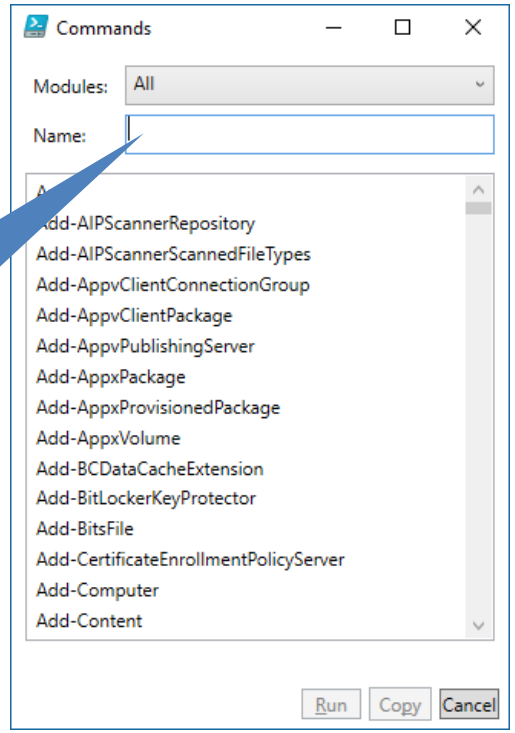
Single Command

```
PS C:\>  Get-Command -Name dir


CommandType     Name                                       ModuleName
-----------     ----                                       -----------
Alias           dir -> Get-ChildItem
```

Show-Command

- Show-Command cmdlet launches GUI Command Browser
- Populate Parameters and Insert or Execute

```
PS C:\> Show-Command
```

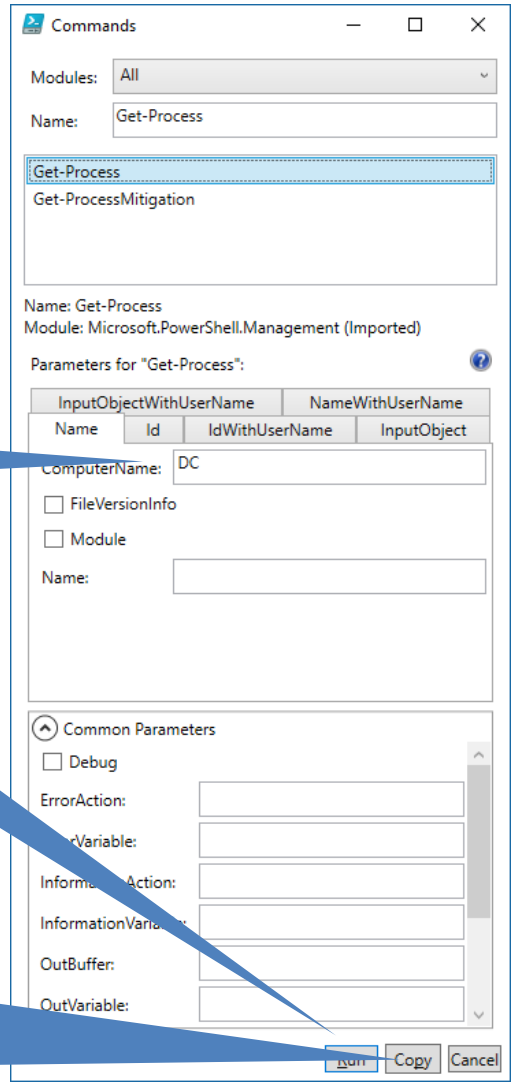


Start Typing Command Name and/or click on command in list

Fill in Parameters

Execute Command Directly

Insert Command with Parameters Populated

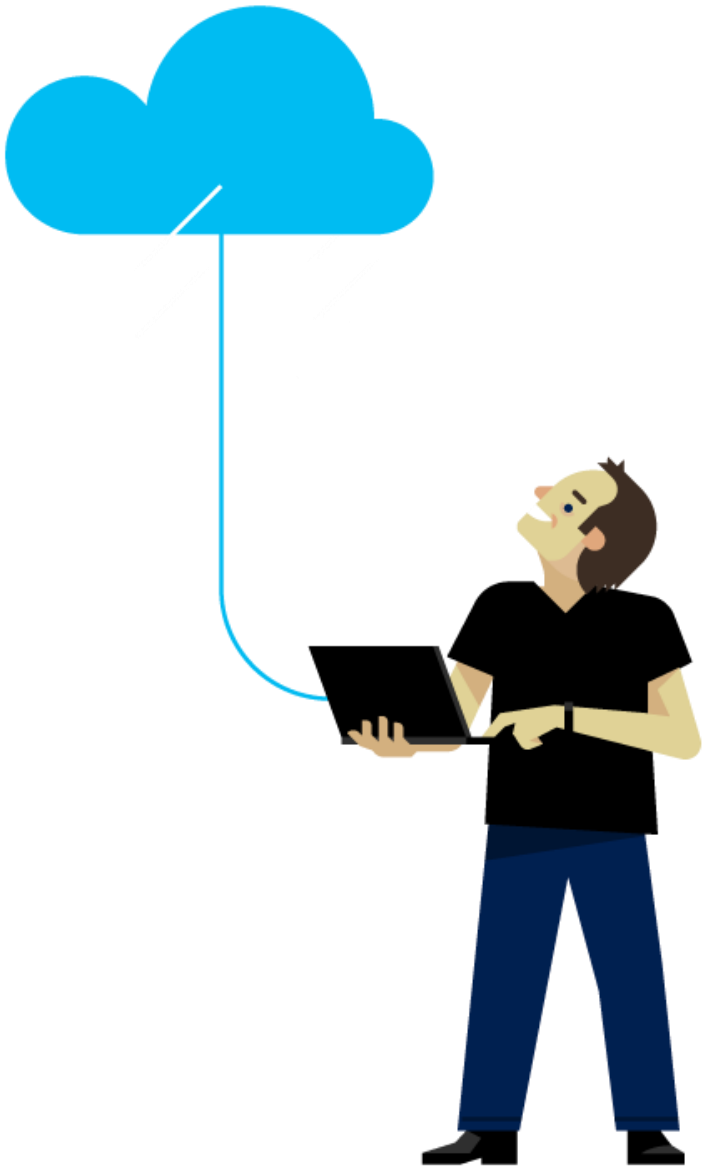



```
PS C:\> Get-Process -ComputerName DC
-Name system -ErrorAction SilentlyContinue
```

Demonstration

Get-Command

Questions?



PowerShell Cmdlet Syntax

List Cmdlet Syntax with Get-Command

```
PS C:\> Get-Command Get-WinEvent -Syntax

Get-WinEvent [[-LogName] <string[]>] [-MaxEvents <long>]
[-ComputerName <string>] [-Credential <pscredential>]
[-FilterXPath <string>] [-Force] [-Oldest] [<CommonParameters>]

Get-WinEvent [-ListLog] <string[]> [-ComputerName <string>]
[-Credential <pscredential>] [-Force] [<CommonParameters>]

Get-WinEvent [-ListProvider] <string[]> [-ComputerName <string>]
[-Credential <pscredential>] [<CommonParameters>]

...
```

Cmdlet Syntax

Syntax Definition

```
<Command-Name> -<Required Parameter Name> <Required Parameter Value>
[-<Optional Parameter Name> <Optional Parameter Value>]
[-<Optional Switch Parameters>]
[-<Optional Parameter Name>] <Required Parameter Value>
<Multiple Parameter Value>[]
```

Syntax Sample

```
PS C:\>  Get-Command -Name Add-Computer -Syntax

Add-Computer [-DomainName] <string> -Credential <pscredential> [-ComputerName
<string[]>] [-LocalCredential
<pscredential>] [-UnjoinDomainCredential <pscredential>] [-OUPath <string>] [-
Server <string>] [-Unsecure] [-Options
<JoinOptions>] [-Restart] [-PassThru] [-NewName <string>] [-Force] [-WhatIf] [-
Confirm] [<CommonParameters>]
```

Syntax Legend

<verb-noun>	Command name
-<parameter>	Required parameter name
<value>	Required parameter value
[-<> <>]	Optional parameter and/or value
[-<>] <value>	Required value, Parameter name is optional
<value[ ]>	Multiple parameter values

Cmdlet Syntax - Command Name

Syntax Definition

```
<Command-Name> -<Required Parameter Name> <Required Parameter Value>
[-<Optional Parameter Name> <Optional Parameter Value>]
[-<Optional Switch Parameters>]
[-<Optional Parameter Name>] <Required Parameter Value>
<Multiple Parameter Values>[]
```

Syntax Sample

```
PS C:\> Get-Command -Name Add-Computer -Syntax

Add-Computer [-DomainName] <string> -Credential <pscredential> [-ComputerName
<string[]>] [-LocalCredential
<pscredential>] [-UnjoinDomainCredential <pscredential>] [-OUPath <string>] [-
Server <string>] [-Unsecure] [-Options
<JoinOptions>] [-Restart] [-PassThru] [-NewName <string>] [-Force] [-WhatIf] [-
Confirm] [<CommonParameters>]
```

Cmdlet Syntax - Required Parameter

Syntax Definition

```
<Command-Name> -<Required Parameter Name> <Required Parameter Value>
[-<Optional Parameter Name> <Optional Parameter Value>]
[-<Optional Switch Parameters>]
[-<Optional Parameter Name>] <Required Parameter Value>
<Multiple Parameter Values>[]
```

Syntax Sample

```
PS C:\> Get-Command -Name Add-Computer -Syntax

Add-Computer [-DomainName] <string> -Credential <pscredential> [-ComputerName
<string[]>] [-LocalCredential
<pscredential>] [-UnjoinDomainCredential <pscredential>] [-OUPath <string>] [-
Server <string>] [-Unsecure] [-Options
<JoinOptions>] [-Restart] [-PassThru] [-NewName <string>] [-Force] [-WhatIf] [-
Confirm] [<CommonParameters>]
```

Cmdlet Syntax - Optional Parameter and Value

Syntax Definition

```
<Command-Name> -<Required Parameter Name> <Required Parameter Value>
[-<Optional Parameter Name> <Optional Parameter Value>]
[-<Optional Switch Parameters>]
[-<Optional Parameter Name>] <Required Parameter Value>
<Multiple Parameter Values>[]
```

Syntax Sample

```
PS C:\> Get-Command -Name Add-Computer -Syntax

Add-Computer [-DomainName] <string> -Credential <pscredential> [-ComputerName
<string[]>] [-LocalCredential
<pscredential>] [-UnjoinDomainCredential <pscredential>] [-OUPath <string>] [-
Server <string>] [-Unsecure] [-Options
<JoinOptions>] [-Restart] [-PassThru] [-NewName <string>] [-Force] [-WhatIf] [-
Confirm] [<CommonParameters>]
```

Cmdlet Syntax - Switch Parameter

Syntax Definition

```
<Command-Name> -<Required Parameter Name> <Required Parameter Value>
[-<Optional Parameter Name> <Optional Parameter Value>]
[-<Optional Switch Parameters>]
[-<Optional Parameter Name>] <Required Parameter Value>
<Multiple Parameter Values>[]
```

Syntax Sample

```
PS C:\> Get-Command -Name Add-Computer -Syntax

Add-Computer [-DomainName] <string> -Credential <pscredential> [-ComputerName
<string[]>] [-LocalCredential
<pscredential>] [-UnjoinDomainCredential <pscredential>] [-OUPath <string>] [-
Server <string>] [-Unsecure] [-Options
<JoinOptions>] [-Restart] [-PassThru] [-NewName <string>] [-Force] [-WhatIf] [-
Confirm] [<CommonParameters>]
```

Cmdlet Syntax - Optional Parameter, Required Value

Syntax Definition

```
<Command-Name> -<Required Parameter Name> <Required Parameter Value>
[-<Optional Parameter Name> <Optional Parameter Value>]
[-<Optional Switch Parameters>]
[-<Optional Parameter Name>] <Required Parameter Value>
<Multiple Parameter Values>[]
```

Syntax Sample

```
PS C:\> Get-Command -Name Add-Computer -Syntax

Add-Computer [-DomainName] <string> -Credential <pscredential> [-ComputerName
<string[]>] [-LocalCredential
<pscredential>] [-UnjoinDomainCredential <pscredential>] [-OUPath <string>] [-
Server <string>] [-Unsecure] [-Options
<JoinOptions>] [-Restart] [-PassThru] [-NewName <string>] [-Force] [-WhatIf] [-
Confirm] [<CommonParameters>]
```

Cmdlet Syntax - Multiple Parameter Values

Syntax Definition

```
<Command-Name> -<Required Parameter Name> <Required Parameter Value>
[-<Optional Parameter Name> <Optional Parameter Value>]
[-<Optional Switch Parameters>]
[-<Optional Parameter Name>] <Required Parameter Value>
<Multiple Parameter Values>[]
```

Syntax Sample

```
PS C:\> Get-Command -Name Add-Computer -Syntax

Add-Computer [-DomainName] <string> -Credential <pscredential> [-ComputerName
<string[]>] [-LocalCredential
<pscredential>] [-UnjoinDomainCredential <pscredential>] [-OUPath <string>] [-
Server <string>] [-Unsecure] [-Options
<JoinOptions>] [-Restart] [-PassThru] [-NewName <string>] [-Force] [-WhatIf] [-
Confirm] [<CommonParameters>]
```

Demonstration

Cmdlet Syntax

Cmdlet Syntax Diagram- Parameter Sets

```
PS C:\>  Get-Command -Name Stop-Process -Syntax

Stop-Process [-Id] <int[]> [-PassThru] [-Force] [-WhatIf] [-Confirm]
[<CommonParameters>]

Stop-Process -Name <string[]> [-PassThru] [-Force] [-WhatIf] [-Confirm]
[<CommonParameters>]

Stop-Process [-InputObject] <Process[]> [-PassThru] [-Force] [-WhatIf]
[-Confirm] [<CommonParameters>]
```

- Note: ‘Id’, ‘Name’ and ‘InputObject’ parameters cannot be used together and are required (value only for ‘-Id’ & ‘-InputObject’) in their respective parameter set

Positional vs. Named Parameters

- Parameters that can be passed without using their ParameterName will have square brackets around just their ParameterName and not their Value.

```
PS C:\>  Get-Command -Name Stop-Process -Syntax

Stop-Process [-Id] <int[]> [-PassThru] [-Force] [-WhatIf] [-Confirm]
[<CommonParameters>]

Stop-Process -Name <string[]> [-PassThru] [-Force] [-WhatIf] [-Confirm]
[<CommonParameters>]

Stop-Process [-InputObject] <Process[]> [-PassThru] [-Force] [-WhatIf] [-
Confirm] [<CommonParameters>]
```

```
PS C:\> Stop-Process -Id 8660
```

vs

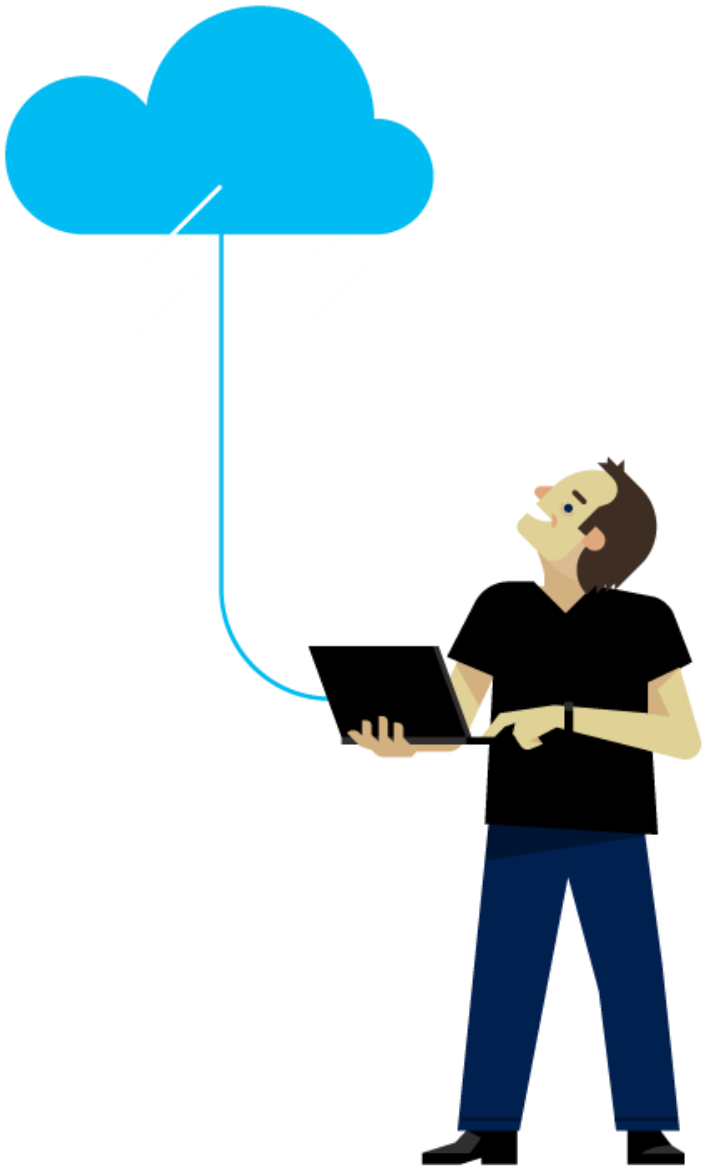
```
PS C:\> Stop-Process 8660
```

- Most Cmdlets will only allow 1 or 2 positional parameters to prevent ambiguity issues.
- PowerShell must be able to determine which parameter set to use.
- Use named parameters in scripts

Demonstration

Parameter Sets & Positional
VS Named Parameters

Questions?



Cmdlet Common Parameters

Common Parameters

- Parameters automatically available with any Cmdlet
- Implemented by PowerShell not Cmdlet developer
- Override system defaults or preferences

Common Parameters (with alias in parenthesis)

-Debug (db)	Displays programmer-level detail
-ErrorAction (ea)	Determines how cmdlet responds to errors
-ErrorVariable (ev)	Stores error messages in a specified variable
-OutVariable (ov)	Stores output in a specified variable
-OutBuffer (ob)	Determines number of output objects to accumulate in a buffer
-PipelineVariable (pv)	Stores value of current pipeline element as a variable
-Verbose (vb)	Displays detailed information
-WarningAction (wa)	Determines how cmdlet responds to warnings
-WarningVariable (wv)	Stores warnings in a specified variable

Common Parameters - Verbose

```
PS C:\> Restart-Service -Name Netlogon
PS C:\>
```

Common Parameter

```
PS C:\> Restart-Service -Name Netlogon -Verbose
 VERBOSE: Performing the operation "Restart-Service" on target
"Netlogon (Netlogon)".

PS C:\>
```

Common Parameters - ErrorAction

```
PS C:\> Get-Process Netlogon

Get-Process : Cannot find a process with the name "Netlogon".
Verify the process name and call the cmdlet again.
At line:1 char:1
+ Get-Process Netlogon
+ ~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : ObjectNotFound:
(Netlogon:String) [Get-Process], ProcessCommandException

PS C:\>
```

```
PS C:\> Get-Process Netlogon -ErrorAction SilentlyContinue
PS C:\>
```

Common Parameter

Error Action

Common Parameters - Outvariable

```
PS C:\> Get-FileHash .\iExploreProcesses.csv -OutVariable csvhash
```

Common Parameter

Variable Name

Use the variable to retrieve the command output

```
PS C:\> $csvhash

Algorithm       Hash            Path
-----------     ----            ----
SHA256          6A78…           C:\iExploreProcesses.csv
```

Note: $ prefix denotes a variable in PowerShell

Outvariable - Exceptions

- Select-object stops pipeline processing resulting in outvariable not containing expected result

```
PS C:\> Get-ChildItem -Path c:\ -OutVariable c | Select-Object -
First 3
PS C:\> $c.count
3
```

Demonstration

Common Parameters

Risk Mitigation Parameters

- Many cmdlets also offer risk mitigation parameters
- Typically when the cmdlet changes the system or application

-WhatIf (wi)	Displays message describing the effect of the command, instead of executing the command
-Confirm (cf)	Prompts for confirmation before executing command

-WhatIf Parameter in Action

```
PS C:\> Stop-Process -Name * -WhatIf

What if: Performing the operation "Stop-Process" on target "AcroRd32 (8160)".
What if: Performing the operation "Stop-Process" on target "AcroRd32 (12756)".
What if: Performing the operation "Stop-Process" on target "armsvc (2468)".
What if: Performing the operation "Stop-Process" on target "atieclxx (3220)".
What if: Performing the operation "Stop-Process" on target "atiesrxx (780)".
What if: Performing the operation "Stop-Process" on target "audiodg (9576)".
...
```

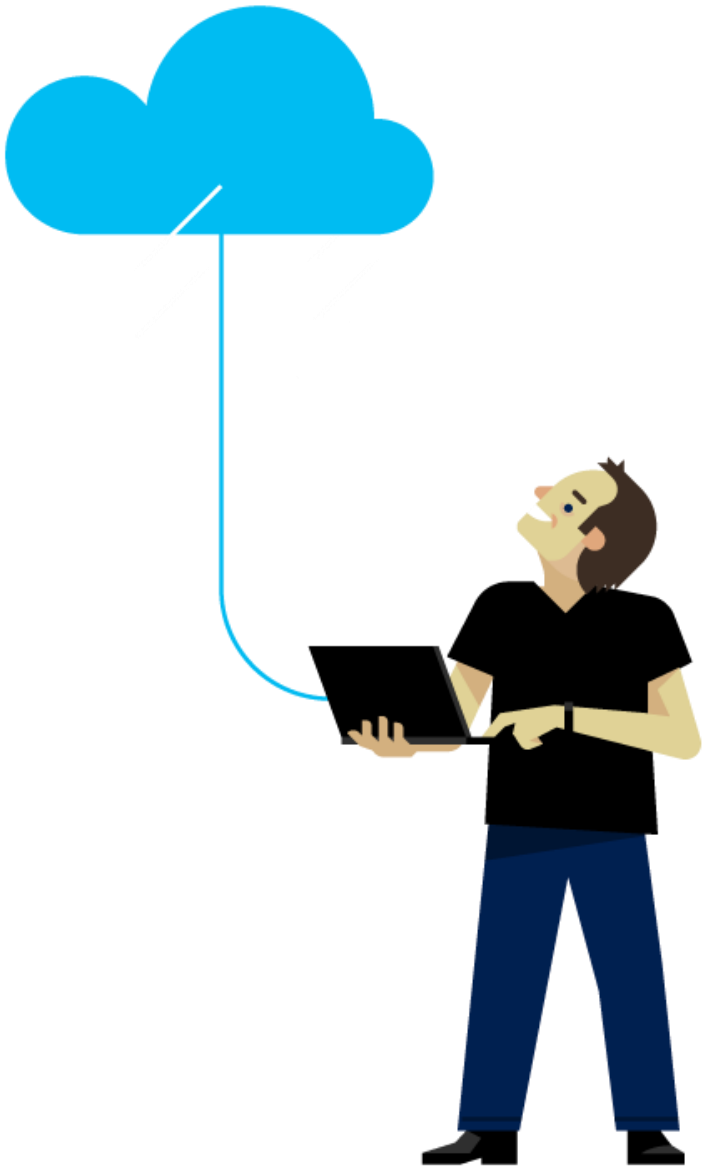
-Confirm Parameter in Action

```
PS C:\> Get-Service | Stop-Service -Confirm

Confirm
Are you sure you want to perform this action?
Performing the operation "Stop-Service" on target "AllJoyn Router Service
(AJRouter)".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default
is "Y"):
```

Demonstration

Risk Mitigation Parameters

Questions?



Command Termination and Line Continuation

Termination Characters

- To complete a command, use either a:
  - Newline character (enter) , or a
  - Semi-colon

- Semi-colon can be used to execute more than one statement on a single line
- Mostly there to support people coming from other programming languages
- Avoid doing this often, as it makes code harder to read and manage

Command Termination Character

Semi-colon command termination

```
PS C:\>  Get-Service BITS ; Get-Process System


Status   Name                DisplayName
------   ----                -----------
Running  BITS                Background Intelligent Transfer Ser...

Id      : 4
Handles : 1308
CPU     : 1213.59375
Name    : System
```

Line Continuation

- When a statement is not syntactically complete and there is a newline character, PowerShell enters line continuation

```
>>
```

- Complete the syntax and include an empty line to finish the statement and execute

Line Continuation

- The following things cause line continuation
  - {},( ), [ ], “ “, ‘ ‘ -> Incomplete Matching Pairs
  - @““@,@”@ -> Incomplete here strings
  - | -> Empty Pipes

- Ctrl-C to break out and abort statement and line continuation
  - Useful when line continuation is accidental (Ctrl-C followed by Up-Arrow gets you back)

- You can manually trigger a continued line with the backtick (grave accent)

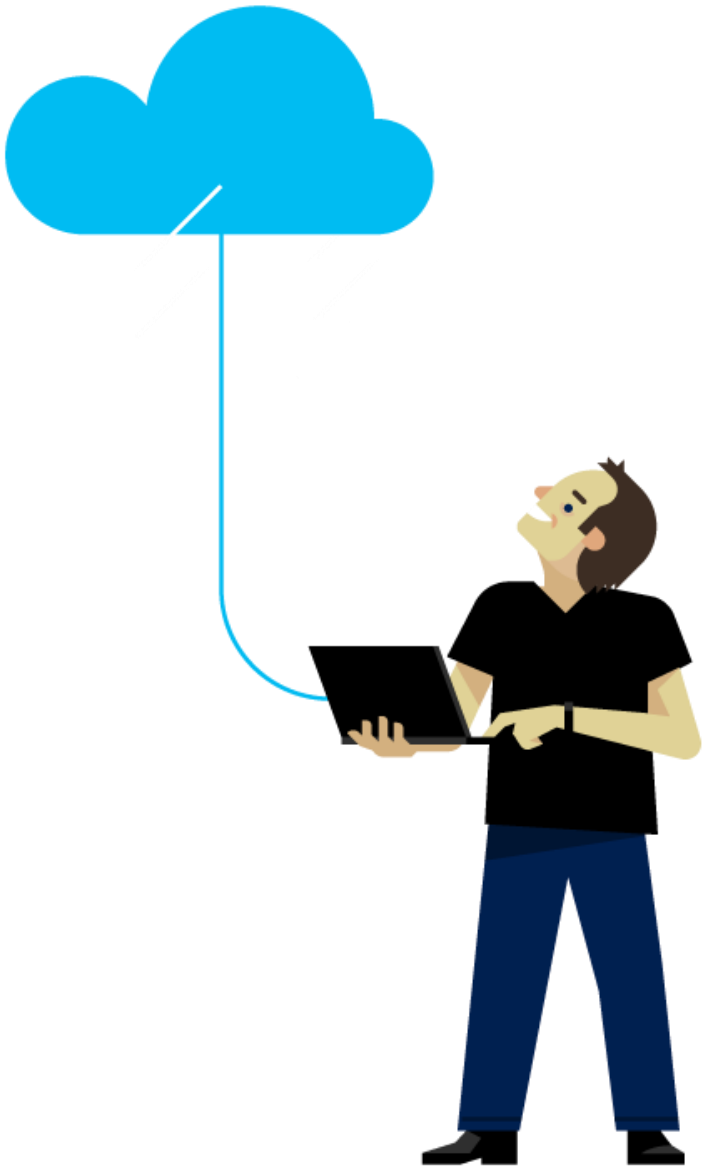
Line Continuation - Example

```
PS C:\> "This is a multi-line
>> string that continues
>> on several lines
>> until the syntax is completed"
>>

This is a multi-line
string that continues
on several lines
until the syntax is completed

PS C:\>
```

Demonstration

Command Termination and Line Continuation

Questions?



Built-in Aliases

What is a Alias

- PowerShell provides short names for frequently used cmdlets
- Can be created/changed by User
- Use Parameter Names as you normally would, or pass them positionally

```
PS C:\> Get-ChildItem -Path C:\ -Recurse
```

ls

```
PS C:\> ls -Path C:\ -Recurse
```

dir

```
PS C:\> dir -Path C:\ -Recurse
```

gci

```
PS C:\> gci -Path C:\ -Recurse
```

Built-in Aliases

- Created and maintained by PowerShell
  - **Don`t change the build in Aliases**
- Ease of PowerShell adoption for Windows cmd.exe and *Nix administrators
- Saves time when typing interactive commands

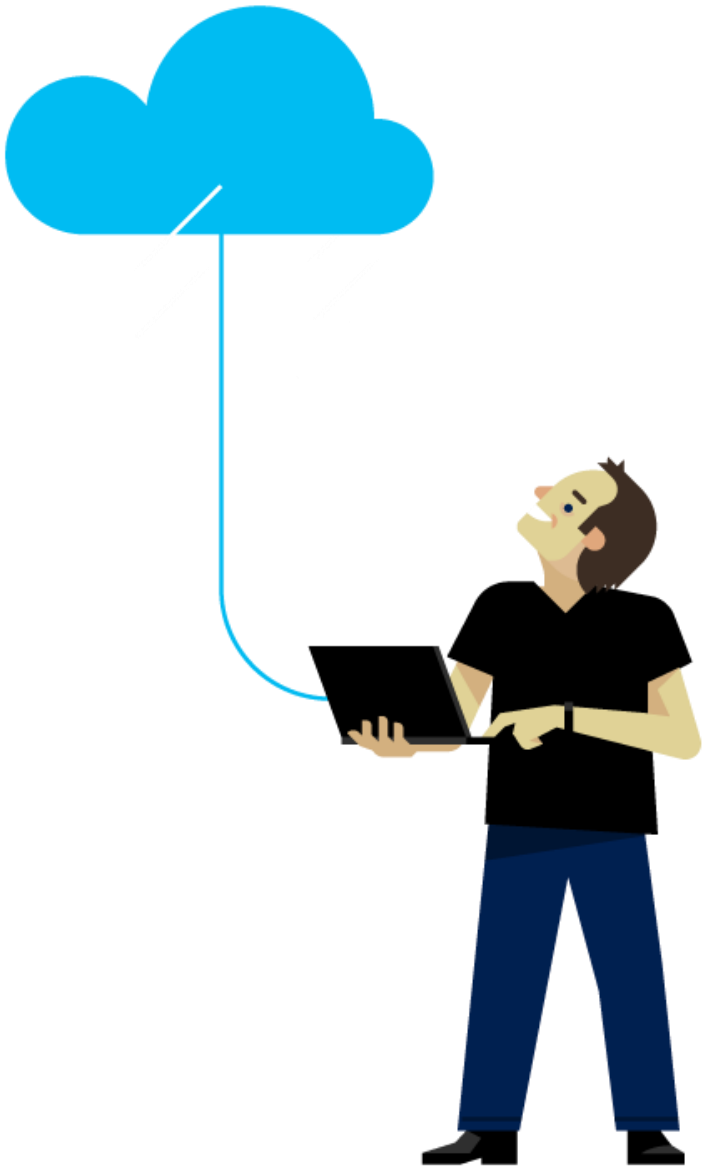
Listing all Aliases

```
PS C:\> Get-Alias

CommandType     Name                    ModuleName
-----------     ----                    -----------
Alias           % -> ForEach-Object
Alias           ? -> Where-Object
Alias           ac -> Add-Content
Alias           asnp -> Add-PSSnapin
Alias           cat -> Get-Content
Alias           cd -> Set-Location
Alias           chdir -> Set-Location
...
```

Demonstration

Built-in Aliases

Questions?



User-defined Aliases

User-Defined Aliases

- Created and maintained by the user
- Are not maintained when the window or script closes
- Saves time when typing interactive commands
- Should not be used in scripts

Alias Cmdlets

Name	Example
Get-Alias	`PS C:\> Get-Alias -Name dir`
Export-Alias	`PS C:\> Export-Alias -Path C:\Aliases.txt`
Import Alias	`PS C:\> Import-Alias -Path C:\Aliases.txt`
New-Alias	`PS C:\> New-Alias -Name gs -Value Get-Service`
Set-Alias	`PS C:\> Set-Alias -Name write -Value Write-Host`

Creating a Custom Alias

New Alias (list) for Get-ChildItem cmdlet

```
PS C:\> New-Alias -Name list -Value Get-ChildItem
```

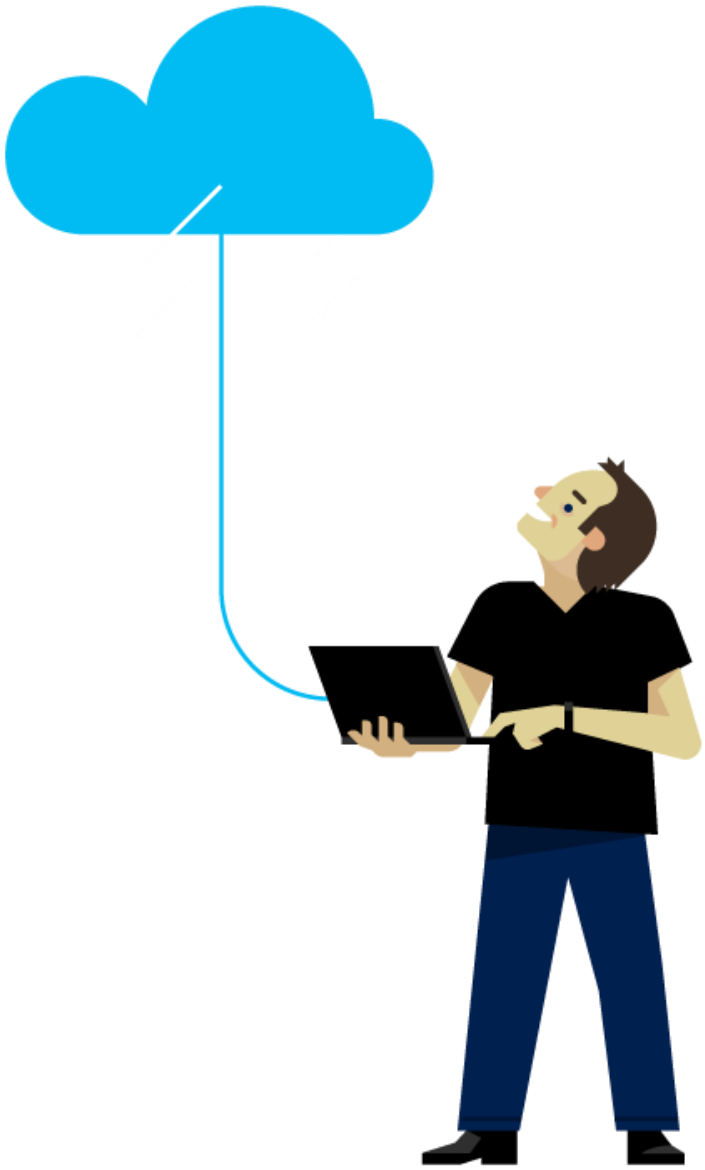
Using New Alias (list)

```
PS C:\> list

    Directory: C:\

Mode                LastWriteTime     Length Name
----                -------------     ------ ----
d----                5/09/2013   1:40 PM     Intel
d-r--               21/10/2013   1:31 PM     Program Files
d-r--               10/12/2013  10:26 AM     Program Files (x86)
d----                1/12/2013   1:32 PM     Scripts
```

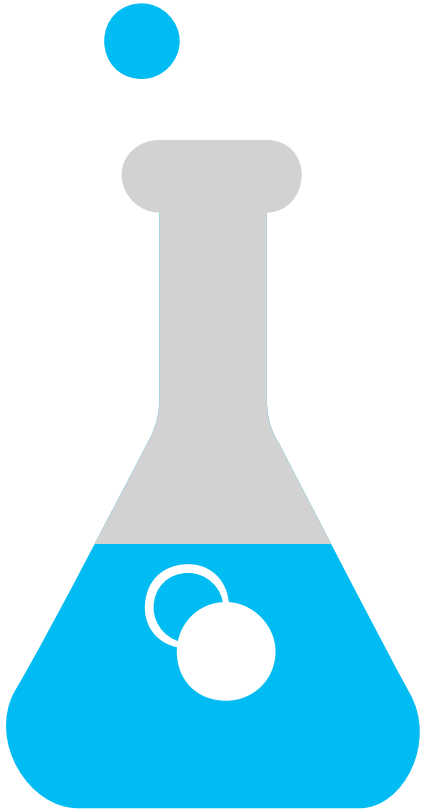
Demonstration

User Defined Aliases

Questions?



Introduction to Commands



LAB

Microsoft