

# PowerShell Core Fundamentals

## WELCOME!



Allen R. Sanders  
Senior Technology Instructor





# Join Us in Making Learning Technology Easier



## Our mission...

Over 16 years ago, we embarked on a journey to improve the world by making learning technology easy and accessible to everyone.

## ...impacts everyone daily.

And it's working. Today, we're known for delivering customized tech learning programs that drive innovation and transform organizations.

In fact, when you talk on the phone, watch a movie, connect with friends on social media, drive a car, fly on a plane, shop online, and order a latte with your mobile app, you are experiencing the impact of our solutions.



Over The Past Few Decades, We've Provided

Over  
**62,300,000**  
expert-led learning hours

In 2019 Alone, We Provided

Training to over  
**13,500** engineers

Programs in  
**30** countries

Over **120** active trainers, with an average of over two decades of experience each.



## Technologies we cover

Develop  
Intelligence



React Native



AND MANY OTHER TRENDING TECHNOLOGIES



# World Class Practitioners



250 best selling books authored



EXPERT PRACTITIONERS

9+ years of training experience



SEASONED CONSULTANTS

150 speaking engagements at industry conferences



Over 62 million practitioner led training hours



ENGAGING INSTRUCTORS

Over 17 years of industry experience per instructor

125 certifications in leading technologies



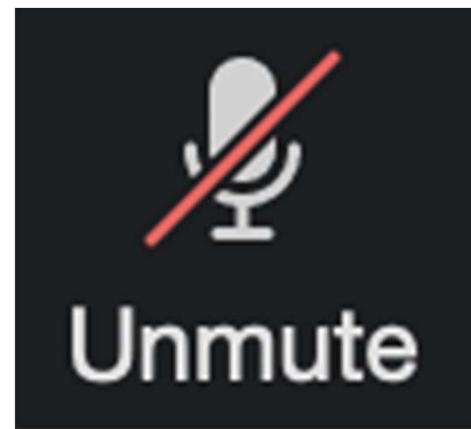
95% instructor satisfaction



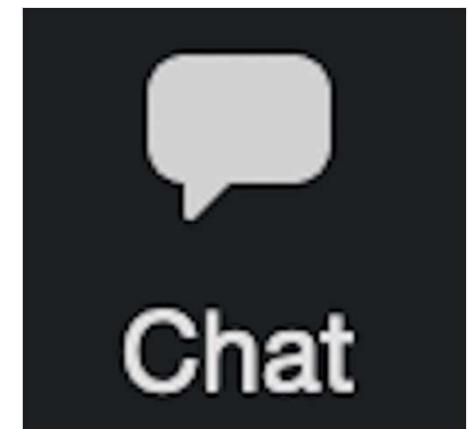
# Virtual Training Expectations for You



Arrive on time / return on time



Mute unless speaking



Use chat or ask  
questions verbally



# Virtual Training Expectations for Me



I pledge to:

- Make this as interesting and interactive as possible
- Ask questions in order to stimulate discussion
- Use whatever resources I have at hand to explain the material
- Try my best to manage verbal responses so that everyone who wants to speak can do so

Quick review of key Zoom features that may be helpful for our course



## Purpose



- Deepen understanding of PowerShell mechanics and automation options
- Explore various types of ad hoc scripting using PowerShell
- Explore building reusable scripts for execution and schedule
- Explore & practice remote management using PowerShell
- Applying foundation for integration with collaboration tool sets & services



# Objectives



Upon completion of this course, you should be able to:

- Utilize PowerShell for retrieving key information about a Windows system using various search and filter techniques
- Demonstrate the ability to effectively leverage PowerShell for automating specific tasks
- Combine PowerShell modules and cmdlets to create scripts for automating more sophisticated workflows
- Build scripts that automate collaboration functions and services, leveraging a strong PowerShell foundation developed throughout the course



# Let's Get to Know One Another



Tell me about you:

- Name
- Current role
- How long you've been at the company
- Any other development background?
- What's one thing you're hoping to get out of this course?

I'll tell you a little about me...

What are some automation tasks you are doing or want to be doing using PowerShell?



# Introduction





# What is PowerShell



From Microsoft docs:

## What is PowerShell?

05/22/2020 • 5 minutes to read • 

PowerShell is a cross-platform task automation and configuration management framework, consisting of a command-line shell and scripting language. Unlike most shells, which accept and return text, PowerShell is built on top of the .NET Common Language Runtime (CLR), and accepts and returns .NET objects. This fundamental change brings entirely new tools and methods for automation.



## PowerShell Features



- Output is object-based vs. scalar-based
- Command family is extensible
- Handles console input and display
- Supports ad hoc and named/reusable scripting



## PowerShell Features



- Supports “piping” for chaining together cmdlets for more complex tasks
- Rich querying and filtering capabilities
- Supports scheduled automation and remote management
- Provides modules for many, many types of admin workloads
- With latest versions, now supports Linux and macOS



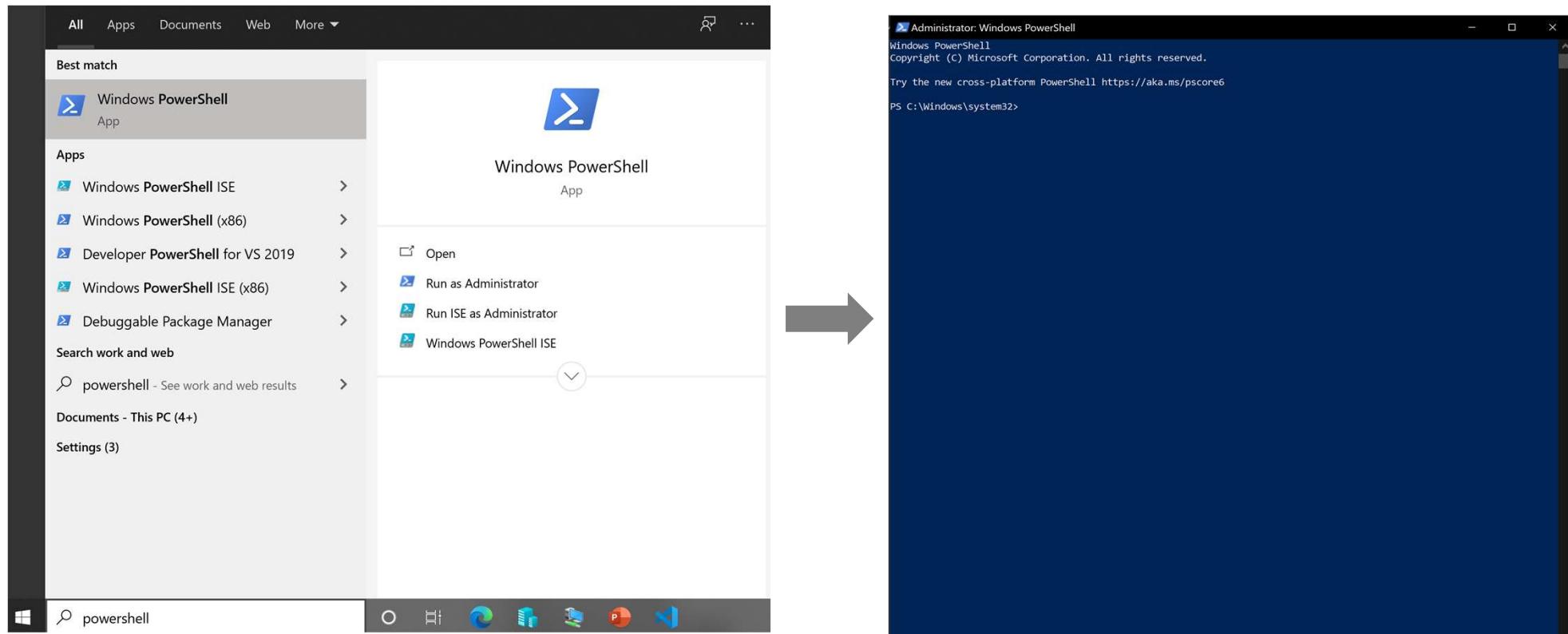
# PowerShell – Up and Running



- Since Windows 7 SP1 and Windows Server 2008 R2 SP1, installed by default (v. 5.1)
- For newer versions or other platforms (plus source code) -> <https://github.com/PowerShell/PowerShell>
- Supports but does not require an IDE
- To launch, click “Start” and enter “powershell” in search box
- Click “Run as Administrator”



# PowerShell – Up and Running





## PowerShell ISE



- Integrated Scripting Environment for PowerShell (similar to an IDE)
- Provides:
  - Development environment for building out new scripts
  - Quick access to help information for cmdlets
  - Interactive console for command execution
- Enables breakpoints and debugging of a script
- Supports selection of a specific section of code for targeted execution
- Will also need “Run as Administrator”



# Visual Studio Code



- Microsoft's recommended approach (with the PowerShell extension)
- Enables same/similar features but in a richer experience (IMO)
- Still provides:
  - Development environment for building out new scripts
  - Quick access to help information for cmdlets
  - Interactive console for command execution



## Visual Studio Code



- Enables breakpoints and debugging of a script
- Supports selection of a specific section of code for targeted execution
- Richer auto-complete and Intellisense for cmdlets
- Will also need “Run as Administrator”



# *Demo*



# PowerShell – The Lay of the Land





## Finding Help



- Universe of available PowerShell cmdlets is large
- Because of that, it will be difficult to remember everything about a command
- Also, you may need to find new commands not previously used
- As with anything, you can Google but there are better, more “PowerShell-y” ways to get help



# Get-Help



- Helps you quickly get to docs for another cmdlet (if available)
- Format is *Get-Help -Name <cmdlet name>*
- Displays as much information as it has access to about the command, command-line arguments, etc.

```
PS C:\Users\sysadmin\Downloads\PowerShell Scripts> Get-Help -Name New-Service

NAME
    New-Service

SYNTAX
    New-Service [-Name] <string> [-BinaryPathName] <string> [-DisplayName <string>] [-Description <string>] [-StartupType {Boot | System | Automatic | Manual | Disabled}] [-Credential <pscredential>] [-DependsOn <string[]>] [-WhatIf] [-Confirm] [<CommonParameters>]

ALIASES
    None

REMARKS
    Get-Help cannot find the Help files for this cmdlet on this computer. It is displaying only partial help.
    -- To download and install Help files for the module that includes this cmdlet, use Update-Help.
    -- To view the Help topic for this cmdlet online, type: "Get-Help New-Service -Online" or
        go to https://go.microsoft.com/fwlink/?LinkID=113359.
```



## Update-Help



- If `Get-Help` is unable to locate additional information about the requested cmdlet, try running `Update-Help`
- This will download and install the latest help files for PowerShell modules installed on your system
- After executing, try `Get-Help` again to see if additional detail is available



## Update-Help



- You may receive an error for some of the modules during update
- Could be:
  - A transient network issue (retry)
  - Help URL's provided in the manifest for the module are incorrect
- If you haven't done so yet, run *Update-Help* in PowerShell



## Get-Help - Redux



- After *Update-Help*, *Get-Help* might return additional detail and options for drilldown
- Additional flags are available for more info (detail increases with each)
  - *-Examples*
  - *-Detailed*
  - *-Full*
  - *-Online*

### REMARKS

To see the examples, type: "get-help New-Service -examples".  
For more information, type: "get-help New-Service -detailed".  
For technical information, type: "get-help New-Service -full".  
For online help, type: "get-help New-Service -online"



## Get-Command



- Like *Get-Help* but provides detail about the command (including syntax) vs. full documentation
- Use *Get-Command* with no parameters to get information about all available commands
- Use the *-Name* parameter to get information about a specific command



# Get-Command



- Returns type, name, version and source module/package
- Use `-Syntax` with a specific command to see information about command signature

```
PS C:\Users\user\source\repos\pshell> Get-Command -Name New-Service -Syntax

New-Service [-Name] <string> [-BinaryPathName] <string> [-DisplayName <string>] [-Description <string>] [-StartupType
<ServiceStartMode>] [-Credential <pscredential>] [-DependsOn <string[]>] [-WhatIf] [-Confirm] [<CommonParameters>]

PS C:\Users\user\source\repos\pshell> █
```



# Using Wildcards



- Many cmdlets support using wildcards for parameter values
- Asterisk (\*) matches zero or more occurrences of any character
- Question mark (?) matches exactly one character
- Left and right brackets ([ ]) surround a set of character options
- Wildcards can be combined

CommandType	Name	Version	Source
Cmdlet	New-AzureStorageAccount	5.3.1	Azure
Cmdlet	New-AzureStorageAccountSASToken	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageBlobSASToken	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageContainer	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageContainerSASToken	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageContainerStoredAccessPolicy	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageContext	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageDirectory	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageFileSASToken	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageKey	5.3.1	Azure
Cmdlet	New-AzureStorageQueue	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageQueueSASToken	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageQueueStoredAccessPolicy	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageShare	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageShareSASToken	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageShareStoredAccessPolicy	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageTable	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageTableSASToken	4.2.1	Azure.Storage
Cmdlet	New-AzureStorageTableStoredAccessPolicy	4.2.1	Azure.Storage



## Loose Parameter Name Matching



- PowerShell commands only require enough characters in a parameter name to uniquely match
- Names are also case-insensitive
- Can help speed up ad hoc scripting but harder to read
- Truncated parameter names not recommended for reusable scripts



# Loose Parameter Name Matching



```
PS C:\Users\{user}\source\repos\pshell> Get-Service -inc w* -exc w32*
```

Status	Name	DisplayName
Stopped	WaaSMedicSvc	Windows Update Medic Service
Stopped	WalletService	WalletService
Stopped	WarpJITSvc	WarpJITSvc
Stopped	wbengine	Block Level Backup Engine Service
Running	WbioSrvc	Windows Biometric Service
Running	Wcmsvc	Windows Connection Manager
Running	wcncsvc	Windows Connect Now - Config Registrar
Running	WdiServiceHost	Diagnostic Service Host
Running	WdiSystemHost	Diagnostic System Host
Stopped	WdnisSvc	Microsoft Defender Antivirus Network
Running	WebClient	WebClient
Stopped	Wecsvc	Windows Event Collector
Stopped	WEHOSTSVC	Windows Encryption Provider Host Se...
Stopped	wercpsupport	Problem Reports Control Panel Support
Stopped	WerSvc	Windows Error Reporting Service
Stopped	WFDSConMgrSvc	Wi-Fi Direct Services Connection Ma...
Stopped	WiaRpc	Still Image Acquisition Events
Stopped	WinDefend	Microsoft Defender Antivirus Service
Running	WinHttpAutoProx...	WinHTTP Web Proxy Auto-Discovery Se...
Running	Wimmgmt	Windows Management Instrumentation
Running	WinRM	Windows Remote Management (WS-Manag...
Stopped	wisvc	Windows Insider Service
Running	WlanSvc	WLAN AutoConfig
Stopped	wlidsvc	Microsoft Account Sign-in Assistant
Stopped	wlpasvc	Local Profile Assistant Service
Stopped	WManSvc	Windows Management Service
Running	wmiApSrv	WMI Performance Adapter
Stopped	WMPNetworkSvc	Windows Media Player Network Sharin...
Stopped	workfoldersvc	Work Folders
Stopped	WpcMonSvc	Parental Controls
Stopped	WPDDBusEnum	Portable Device Enumerator Service
Running	WpnService	Windows Push Notifications System S...
Running	WpnUserService_...	WpnUserService_54801
Running	wscsvc	Security Center
Running	WSearch	Windows Search
Running	wuauserv	Windows Update
Stopped	WwanSvc	WWAN AutoConfig



```
ell> Get-Service -inc w* -exc w32*
```



# Loose Parameter Name Matching



```
PS C:\Users\{user}\source\repos\pshell> Get-Service -in w* -exc w32*
Get-Service : Parameter cannot be processed because the parameter name 'in' is ambiguous. Possible matches include: -Include -InputObject -InformationAction
-InInformationVariable.
At line:1 char:13
+ Get-Service -in w* -exc w32*
+           ~~~
+ CategoryInfo          : InvalidArgument: (:) [Get-Service], ParameterBindingException
+ FullyQualifiedErrorId : AmbiguousParameter,Microsoft.PowerShell.Commands.GetServiceCommand
```



# Get-Member



- Displays underlying .NET type for an object
- Lists properties and methods available for an object
- Can be helpful for determining what's possible with a variable of a given type
- Results will be reflective of underlying .NET type

```
PS C:\Users\alansan\source\repos> Get-Command -Name Get-Member -Syntax

Get-Member [[-Name] <string[]>] [-InputObject <psobject>] [-MemberType <PSMemberTypes>] [-View <PSMemberViewTypes>] [-Static] [-Force] [<CommonParameters>]

PS C:\Users\alansan\source\repos> $var = 19

PS C:\Users\alansan\source\repos> Get-Member -InputObject $var

TypeName: System.Int32

Name      MemberType Definition
----      -----
CompareTo Method   int CompareTo(System.Object value), int CompareTo(int value), int IComparable.CompareTo(System.Object obj), int IComparable[int].CompareTo(int other)
Equals    Method   bool Equals(System.Object obj), bool Equals(int obj), bool IEquatable[int].Equals(int other)
GetHashCode Method  int GetHashCode()
GetType   Method   type GetType()
GetTypeCode Method  System.TypeCode GetTypeCode(), System.TypeCode IConvertible.GetTypeCode()
.ToBoolean Method  bool IConvertible.ToBoolean(System.IFormatProvider provider)
.ToByte    Method  byte IConvertible.ToByte(System.IFormatProvider provider)
.ToChar    Method  char IConvertible.ToChar(System.IFormatProvider provider)
.ToDateTime Method  datetime IConvertible.ToDateTime(System.IFormatProvider provider)
```



## Show-Command



- Launches a dialog for guided definition of a command
- User can set values for various parameters in the UI
- “Run” will execute with user-provided inputs
- “Copy” will copy for use in a script or scripting session
- Also, provides a link to help for the command



# Show-Command



Get-Service

Parameters for "Get-Service":

Default DisplayName InputObject

ComputerName: SERVER01

DependentServices

Exclude: w32\*

Include: w\*

Name:

RequiredServices

Common Parameters

Run Copy Cancel

Get-Service Help

Find: Previous Next Settings

**Synopsis**  
Gets the services on a local or remote computer.

**Description**  
The Get-Service cmdlet gets objects that represent the services on a local computer or on a remote computer, including running and stopped services.

You can direct this cmdlet to get only particular services by specifying the service name or the display name of the services, or you can pipe service objects to this cmdlet.

**Parameters**

-ComputerName <System.String[]>  
Gets the services running on the specified computers. The default is the local computer.

Required?	false
Position?	named
Default value	None

```
ell> Get-Service -ComputerName SERVER01 -Exclude w32* -Include w*
```



## -WhatIf and -Confirm



- Cmdlets that make changes often include two additional parameters
- *-WhatIf* provides info on what will happen without executing
- *-Confirm* allows user to verify intent to proceed
- There are ways to include the same for our own functions

```
PS C:\Users\user\source\repos\pshell> Get-ChildItem -Path C:\Users\user\Downloads\*.* -Recurse | Remove-Item -WhatIf

What if: Performing the operation "Remove File" on target "C:\Users\user\Downloads\2018090513375114894.pdf".
What if: Performing the operation "Remove File" on target "C:\Users\user\Downloads\File 1.txt".
What if: Performing the operation "Remove File" on target "C:\Users\user\Downloads\File 2.txt".
What if: Performing the operation "Remove File" on target "C:\Users\user\Downloads\File 3.txt".
What if: Performing the operation "Remove File" on target "C:\Users\user\Downloads\Image.bmp".
PS C:\Users\user\source\repos\pshell>
```

```
PS C:\Users\user\source\repos\pshell> Get-ChildItem -Path C:\Users\user\Downloads\*.* -Recurse | Remove-Item -Confirm

Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target "C:\Users\user\Downloads\2018090513375114894.pdf".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Yes"): n
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target "C:\Users\user\Downloads\File 1.txt".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Yes"): n
Confirm
Are you sure you want to perform this action?
Performing the operation "Remove File" on target "C:\Users\user\Downloads\File 2.txt".
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "Yes"): n
```



## Miscellaneous Info



- A backtick (`) can be used to span multiple lines with a command
- Multiple variables can be declared and initialized with same or different values in one line of code
- When working in PowerShell, Ctrl+Space auto-completes if able or shows available auto-completions (for selection)



# Miscellaneous Info



```
PS C:\Users\{user}\source\repos\pshell> Get-Service `>> -Include w*`>> -Exclude w32*

| Status  | Name           | DisplayName                               |
|---------|----------------|-------------------------------------------|
| Stopped | WaaSMedicSvc   | Windows Update Medic Service              |
| Stopped | WalletService  | WalletService                             |
| Stopped | WarpJITSvc     | WarpJITSvc                                |
| Stopped | wbengine       | Block Level Backup Engine Service         |
| Running | WbioSrvc       | Windows Biometric Service                 |
| Running | Wcmsvc         | Windows Connection Manager                |
| Running | wcnsvc         | Windows Connect Now - Config Registrar    |
| Running | WdiServiceHost | Diagnostic Service Host                   |
| Running | WdiSystemHost  | Diagnostic System Host                    |
| Stopped | WdNisSvc       | Microsoft Defender Antivirus Network...er |
| Running | WebClient      | WebClient                                 |


```

```
PS C:\Users\{user}\source\repos\pshell> $var1, $var2, $var3 = "A", 1, "C"  
PS C:\Users\{user}\source\repos\pshell> $var1  
A  
PS C:\Users\{user}\source\repos\pshell> $var2  
1  
PS C:\Users\{user}\source\repos\pshell> $var3  
C  
PS C:\Users\{user}\source\repos\pshell> $var4, $var5 = "D"  
PS C:\Users\{user}\source\repos\pshell> $var4  
D  
PS C:\Users\{user}\source\repos\pshell> $var5
```

```
PS C:\Users\{user}\source\repos\pshell> [System.MidpointRounding]::AwayFromZero  
AwayFromZero Equals GetName GetUnderlyingType IsDefined ReferenceEquals TryParse  
ToEven Format GetNames GetValues Parse ToObject  
  
static System.MidpointRounding AwayFromZero {get;}
```



# Variables & Data Types





# Declaring Variables in PowerShell



- To create a variable in PowerShell, you use \$ + variable name and assign an initial value
- PowerShell infers data type from the assigned value
- If you assign the variable to a different type, data type is silently changed



# Declaring Variables in PowerShell



```
PS C:\Users\ a_san\source\repos\pshell> $numHours = 12
PS C:\Users\ a_san\source\repos\pshell> $numHours
12
PS C:\Users\ a_san\source\repos\pshell> $numHours.GetType().FullName
System.Int32
PS C:\Users\ a_san\source\repos\pshell> $numHours = 13.8
PS C:\Users\ a_san\source\repos\pshell> $numHours
13.8
PS C:\Users\ a_san\source\repos\pshell> $numHours.GetType().FullName
System.Double
PS C:\Users\ a_san\source\repos\pshell> $numHours = "Eleven"
PS C:\Users\ a_san\source\repos\pshell> $numHours
Eleven
PS C:\Users\ a_san\source\repos\pshell> $numHours.GetType().FullName
System.String
PS C:\Users\ a_san\source\repos\pshell> █
```



# Declaring Variables in PowerShell



- There may be times when you want to set a specific type
- To do that, you use [*<typename>*]\$<variablename>
- Once declared with a specific type, the variable will keep that type for the current scope
- To recreate the same variable with a different type, first remove the existing one
- To delete a variable, you can use *Remove-Variable –Name <variablename>*



# Declaring Variables in PowerShell



```
PS C:\Users\{a_san}\source\repos\pshell> [int]$wholeNumbersOnly = 12
PS C:\Users\{a_san}\source\repos\pshell> $wholeNumbersOnly
12
PS C:\Users\{a_san}\source\repos\pshell> $wholeNumbersOnly.GetType().FullName
System.Int32
PS C:\Users\{a_san}\source\repos\pshell> $wholeNumbersOnly = 13.85
PS C:\Users\{a_san}\source\repos\pshell> $wholeNumbersOnly
14
PS C:\Users\{a_san}\source\repos\pshell> $wholeNumbersOnly.GetType().FullName
System.Int32
PS C:\Users\{a_san}\source\repos\pshell> $wholeNumbersOnly = "Eleven"
Cannot convert value "Eleven" to type "System.Int32". Error: "Input string was not in a correct format."
At line:1 char:1
+ $wholeNumbersOnly = "Eleven"
+ ~~~~~
+ CategoryInfo          : MetadataError: (:) [], ArgumentTransformationMetadataException
+ FullyQualifiedErrorId : RuntimeException

PS C:\Users\{a_san}\source\repos\pshell> Remove-Variable -Name wholeNumbersOnly
PS C:\Users\{a_san}\source\repos\pshell> $wholeNumbersOnly = "Eleven"
PS C:\Users\{a_san}\source\repos\pshell> $wholeNumbersOnly
Eleven
PS C:\Users\{a_san}\source\repos\pshell> █
```



# Declaring Variables in PowerShell



- Available types include .NET types defined in installed/imported modules
- .NET types may include namespace plus type name
- For standard types, you can use formal .NET type or simplified equivalent
- Explicit types improve readability and can help protect against errors



# Declaring Variables in PowerShell



```
PS C:\Users\A_San\source\repos\pshell> [System.Int32]$numVar1 = 12
PS C:\Users\A_San\source\repos\pshell> [int]$numVar2 = 13
PS C:\Users\A_San\source\repos\pshell> $numVar1.GetType().FullName
System.Int32
PS C:\Users\A_San\source\repos\pshell> $numVar2.GetType().FullName
System.Int32
PS C:\Users\A_San\source\repos\pshell> $dictionary = New-Object 'System.Collections.Generic.Dictionary[System.String, System.Object]]'
PS C:\Users\A_San\source\repos\pshell> $dictionary.Add("One", 1)
PS C:\Users\A_San\source\repos\pshell> $dictionary.Add("Two", 2)
PS C:\Users\A_San\source\repos\pshell> $dictionary

Key Value
--- ---
One     1
Two     2
```



# Declaring Variables in PowerShell



- Variables in PowerShell can also be used to store the results of a command for use later

```
PS C:\Users\{user}\source\repos\pshell> $serviceData = Get-Service -Name Wm*
PS C:\Users\{user}\source\repos\pshell> $serviceData

Status    Name            DisplayName
-----   --
Stopped   WManSvc        Windows Management Service
Stopped   wmiApSrv       WMI Performance Adapter
Stopped   WMPNetworkSvc  Windows Media Player Network Sharin...
```



# Automatic Variables in PowerShell



- Several pre-existing variables installed with PowerShell (akin to environment variables)
- Among other things:
  - Provide shortcut access to things like file paths
  - Used to represent certain “constant” values in PowerShell
- Conceptually, should be considered read-only



# Automatic Variables in PowerShell



- Path-like examples include:
  - `$HOME` – full path of user's home directory
  - `$PROFILE` – full path of PowerShell profile for current user; includes sub-properties to account for variations in user and host
  - `$PSHOME` – full path of installation directory for PowerShell
  - `$PWD` – full path of current directory
- Constant-like examples include:
  - `$true`
  - `$false`
  - `$null`
- `$PSVersionTable` provides info about PowerShell version for session



# Operators





# Arithmetic Operators



- PowerShell includes standard arithmetic operators

Operator	Description
+	Adds numbers and concatenates strings, arrays and hash tables
-	Subtracts numbers
-	Unary version used to negate a number
*	Multiples numbers or copies strings and arrays specified number of times
/	Divides numbers
%	Modulus – returns remainder of a division operation



# Additional Operators



- PowerShell also includes other operators

Operator	Description
-band	Bitwise AND
-bnot	Bitwise NOT
-bor	Bitwise OR (standard)
-bxor	Bitwise OR (exclusive)
-shl	Left shift operator for bits
-shr	Right shift operator for bits



# Operator Precedence



- When operators are combined, PowerShell evaluates in a specific order

Precedence	Operator	Description
1	()	Grouping parentheses; can be used with commands also
2	-	Unary negation
3	*, /, %	Multiplication, division and modulus
4	+,-	Addition and subtraction
5	-band, -bnot, -bor, -bxor, -shr, -shl	Bitwise operations



# Division and Rounding



- When the results of division are stored in an integer variable, PowerShell will round (if required)
- Follows most standard rounding rules
- If result includes .5, the number will be rounded to the nearest even integer
- The [Math] class can be used to modify rounding behavior

```
PS C:\Users\ a_san\source\repos\pshell> [int]$result1 = 5 / 2
PS C:\Users\ a_san\source\repos\pshell> [int]$result2 = 7 / 2
PS C:\Users\ a_san\source\repos\pshell> $result1
2
PS C:\Users\ a_san\source\repos\pshell> $result2
4
PS C:\Users\ a_san\source\repos\pshell> [int][Math]::Round(5 / 2, [MidpointRounding]::AwayFromZero)
3
PS C:\Users\ a_san\source\repos\pshell> [int][Math]::Round(7 / 2, [MidpointRounding]::AwayFromZero)
4
PS C:\Users\ a_san\source\repos\pshell> █
```



# Adding and Multiplying Non-Numeric Types



- When using + with strings, arrays, and hash tables, operands are concatenated
- When using \* with strings and arrays, the operand is repeated
- For hash tables, you can only add other hash tables to a hash table and the repeat operator is not supported
- When adding hash table to array, hash tables becomes array item
- With hash tables, attempts to add a hash table with duplicate key will fail



# Adding and Multiplying Non-Numeric Types



```
PS C:\Users\ a_san\source\repos\pshell> $partOne = "Hello"
PS C:\Users\ a_san\source\repos\pshell> $partTwo = "World!"
PS C:\Users\ a_san\source\repos\pshell> $partOne + " " + $partTwo
Hello World!
PS C:\Users\ a_san\source\repos\pshell> $array1 = 1, 2, 3
PS C:\Users\ a_san\source\repos\pshell> $array2 = "A", "B", "C"
PS C:\Users\ a_san\source\repos\pshell> $array1 + $array2
1
2
3
A
B
C
PS C:\Users\ a_san\source\repos\pshell> █
```



# Adding and Multiplying Non-Numeric Types



```
PS C:\Users\ a_san\source\repos\pshell> $repeat = "PowerShell"
PS C:\Users\ a_san\source\repos\pshell> $repeat * 3
PowerShellPowerShellPowerShell
PS C:\Users\ a_san\source\repos\pshell> $array = "A", 1, "C"
PS C:\Users\ a_san\source\repos\pshell> $array * 4
A
1
C
A
1
C
A
1
C
A
1
C
PS C:\Users\ a_san\source\repos\pshell>
```



# Adding and Multiplying Non-Numeric Types



```
PS C:\Users\ a_san\source\repos\pshell> $hash1 = @{ "One" = 1; "Two" = 2; "Three" = 3 }

PS C:\Users\ a_san\source\repos\pshell> $hash2 = @{ "Four" = 4; "Five" = 5 }

PS C:\Users\ a_san\source\repos\pshell> $hash1 + $hash2

Name          Value
----          -----
Four          4
Five          5
One           1
Three         3
Two           2

PS C:\Users\ a_san\source\repos\pshell> $hash3 = @{ "One" = 14; "Six" = 6 }

PS C:\Users\ a_san\source\repos\pshell> $hash1 + $hash3

Item has already been added. Key in dictionary: 'One'  Key being added: 'One'
At line:1 char:1
+ $hash1 + $hash3
+ ~~~~~
+ CategoryInfo          : OperationStopped: () [], ArgumentException
+ FullyQualifiedErrorId : System.ArgumentException
```



# Combining Different Types



- When combining values of different types (non-numeric with numeric), PowerShell targets type of leftmost operand
- PowerShell tries to convert all operands to that type before combining
- As a result, + and \* are not necessarily commutative when used this way

```
PS C:\Users\{user}\source\repos\pshell> 'string value' + 42
string value42
PS C:\Users\{user}\source\repos\pshell> 42 + 'string value'

Cannot convert value "string value" to type "System.Int32". Error: "Input string was not in a correct format."
At line:1 char:1
+ 42 + 'string value'
+ ~~~~~
+ CategoryInfo          : InvalidArgument: (:) [], RuntimeException
+ FullyQualifiedErrorId : InvalidCastFromStringToInteger
```



# Combining Different Types



- When combining values of different numeric types, PowerShell selects the type that minimizes precision loss
- If the result of the operation is too large for the largest type of the operands, PowerShell will automatically widen to accommodate
- This means that result may not match the type of any of the operands

```
PS C:\Users\{a_san}\source\repos\pshell> 29 * [Math]::PI
91.106186954104
PS C:\Users\{a_san}\source\repos\pshell> (29).GetType().FullName
System.Int32
PS C:\Users\{a_san}\source\repos\pshell> (29 * [Math]::PI).GetType().FullName
System.Double
PS C:\Users\{a_san}\source\repos\pshell> (1024MB).GetType().FullName
System.Double
PS C:\Users\{a_san}\source\repos\pshell> (1024MB * 1024MB).GetType().FullName
System.Double
PS C:\Users\{a_san}\source\repos\pshell> █
```



# Assignment Operators



- PowerShell includes various assignment operators (including shortcuts)

Operator	Description
=	Sets variable to specific value (lowest precedence)
+=	Adds value to variable and assigns result to variable; with non-numeric type, appends and assigns
-=	Subtracts value from variable and assigns result to variable
*=	Multiplies variable by a value and assigns result to variable; with non-numeric type, appends repeating occurrences and assigns
/=	Divides variable by a value and assigns result to variable
%=	Divides variable by a value and assigns remainder to variable
++	Increase value of variable by 1 (includes prefix and postfix versions)
--	Decreases value of variable by 1 (includes prefix and postfix versions)



# Assignment Operators



```
PS C:\Users\ a_san\source\repos\pshell> $cost = 19.99
```

```
PS C:\Users\ a_san\source\repos\pshell> $tax = 0.075
```

```
PS C:\Users\ a_san\source\repos\pshell> $cost += ($cost * $tax)
```

```
PS C:\Users\ a_san\source\repos\pshell> $cost
```

```
21.48925
```

```
PS C:\Users\ a_san\source\repos\pshell> "{0:C}" -f $cost
```

```
$21.49
```

```
PS C:\Users\ a_san\source\repos\pshell> █
```

```
PS C:\Users\ a_san\source\repos\pshell> $array3 = 13, -18
```

```
PS C:\Users\ a_san\source\repos\pshell> $array3 *= 2
```

```
PS C:\Users\ a_san\source\repos\pshell> $array3
```

```
13
```

```
-18
```

```
13
```

```
-18
```

```
PS C:\Users\ a_san\source\repos\pshell> █
```

```
PS C:\Users\ a_san\source\repos\pshell> $index = 0
```

```
PS C:\Users\ a_san\source\repos\pshell> $array1 = @(24, -8)
```

```
PS C:\Users\ a_san\source\repos\pshell> $array2 = @(-3, 14)
```

```
PS C:\Users\ a_san\source\repos\pshell> $array1[$index] = $array2[$index++]
```

```
PS C:\Users\ a_san\source\repos\pshell> $index, $array1, $array2
```

```
1
```

```
24
```

```
-3
```

```
-3
```

```
14
```

```
PS C:\Users\ a_san\source\repos\pshell> █
```



# Aliases





# What is an Alias in PowerShell?



- An alternate name or nickname for a command
- Can use the alias anywhere you would normally use the command
- There are predefined aliases, or you can create your own
- Aliases created by you only exist in current session (unless added to profile or exported for import in a new session)

```
PS C:\Users\ a_san\source\repos\pshell> dir
PS C:\Users\ a_san\source\repos\pshell> cd ..
PS C:\Users\ a_san\source\repos> dir

Directory: C:\Users\ a_san\source\repos

Mode                LastWriteTime         Length Name
----                -----          ----- 
d----   12/6/2020 1:13 PM            pshell
d----   12/6/2020 1:27 PM           pshell-core

PS C:\Users\ a_san\source\repos> ls

Directory: C:\Users\ a_san\source\repos

Mode                LastWriteTime         Length Name
----                -----          ----- 
d----   12/6/2020 1:13 PM            pshell
d----   12/6/2020 1:27 PM           pshell-core

PS C:\Users\ a_san\source\repos> █
```



# Get-Alias



- Lists all aliases available in current session
- Supports parameters for filtering and wildcards for searching

```
PS C:\Users\zhangsan\source\repos> Get-Command -Name Get-Alias -Syntax

Get-Alias [[-Name] <string[]>] [-Exclude <string[]>] [-Scope <string>] [<CommonParameters>]
Get-Alias [-Exclude <string[]>] [-Scope <string>] [-Definition <string[]>] [<CommonParameters>]

PS C:\Users\zhangsan\source\repos> Get-Alias -Name 'c*'

 CommandType      Name          Version   Source
 ----           ----          -----   -----
 Alias          cat -> Get-Content
 Alias          cd -> Set-Location
 Alias          CFS -> ConvertFrom-String
 Alias          chdir -> Set-Location
 Alias          clc -> Clear-Content
 Alias          clear -> Clear-Host
 Alias          clhy -> Clear-History
 Alias          cli -> Clear-Item
 Alias          clp -> Clear-ItemProperty
 Alias          cls -> Clear-Host
 Alias          clv -> Clear-Variable
 Alias          cnsn -> Connect-PSession
 Alias          compare -> Compare-Object
 Alias          copy -> Copy-Item
 Alias          cp -> Copy-Item
 Alias          cpi -> Copy-Item
 Alias          cpp -> Copy-ItemProperty
 Alias          curl -> Invoke-WebRequest
 Alias          cvpa -> Convert-Path
```



# New-Alias



- Creates a new custom alias or nickname for a specific command
- By default, exists only in current session
- Once defined, can be used in place of the command name

```
PS C:\Users\lisa\source\repos> New-Alias -Name ghelp -Value Get-Help
PS C:\Users\lisa\source\repos> Get-Alias -Name ghelp

 CommandType      Name          Version      Source
 -----          ----          -----      -----
 Alias           ghelp -> Get-Help

PS C:\Users\lisa\source\repos> ghelp New-Alias -Examples

NAME
    New-Alias

SYNOPSIS
    Creates a new alias.

----- Example 1: Create an alias for a cmdlet -----
PS C:\> New-Alias -Name "List" Get-ChildItem

This command creates an alias named List to represent the Get-ChildItem cmdlet.

----- Example 2: Create a read-only alias for a cmdlet -----
PS C:\> New-Alias -Name "W" -Value Get-WmiObject -Description "quick wmi alias" -Option ReadOnly
PS C:\> Get-Alias -Name "W" | Format-List *

This command creates an alias named W to represent the Get-WmiObject cmdlet. It creates a description, quick wmi alias, pipes it to Format-List to display all of the information about it.

PS C:\Users\lisa\source\repos>
```



# Set-Alias



- Creates or changes a custom alias
- Aliases can be used for user-defined functions as well

```
PS C:\Users\ a_san\source\repos> New-Alias -Name slog -Value Get-EventLog

PS C:\Users\ a_san\source\repos> slog -LogName System | Select-Object -First 3

Index Time          EntryType   Source           InstanceID Message
----- ----          -----      -----
13179 Dec 06 20:55 Information Service Control M... 1073748864 The start type of the Background Intelligent Transfer Service serv
13178 Dec 06 20:54 Warning    DCOM               10016 The description for Event ID '10016' in Source 'DCOM' cannot be fo
13177 Dec 06 20:51 Warning    DCOM               10016 The description for Event ID '10016' in Source 'DCOM' cannot be fo

PS C:\Users\ a_san\source\repos> function Get-SystemLog { Get-EventLog -LogName System }

PS C:\Users\ a_san\source\repos> Set-Alias -Name slog -Value Get-SystemLog

PS C:\Users\ a_san\source\repos> slog | Select-Object -First 3

Index Time          EntryType   Source           InstanceID Message
----- ----          -----      -----
13179 Dec 06 20:55 Information Service Control M... 1073748864 The start type of the Background Intelligent Transfer Service serv
13178 Dec 06 20:54 Warning    DCOM               10016 The description for Event ID '10016' in Source 'DCOM' cannot be fo
13177 Dec 06 20:51 Warning    DCOM               10016 The description for Event ID '10016' in Source 'DCOM' cannot be fo
```



# Exporting and Importing



- As mentioned, by default, custom aliases only exist in current session
- To make available for other sessions, can add *New-Alias* or *Set-Alias* to profile
- Alternatively, can export aliases to a file using *Export-Alias*
- Subsequently, can import into a new PowerShell session



# The Alias Provider



- We'll discuss PowerShell providers later
- For now, know that PowerShell includes a provider for defined aliases
- Provider allows us to reference available aliases as if contained in a drive

```
PS C:\Users\user\source\repos> Get-ChildItem -Path Alias: | Select-Object -First 10

CommandType      Name          Version   Source
----           ----          -----   -----
Alias           % -> ForEach-Object
Alias           ? -> Where-Object
Alias           ac -> Add-Content
Alias           asnp -> Add-PSSnapin
Alias           cat -> Get-Content
Alias           cd -> Set-Location
Alias           CFS -> ConvertFrom-String
Alias           chdir -> Set-Location
Alias           clc -> Clear-Content
Alias           clear -> Clear-Host
                                         3.1.0.0   Microsoft.PowerShell.Utility
```



# Arrays, ArrayLists and Hash Tables

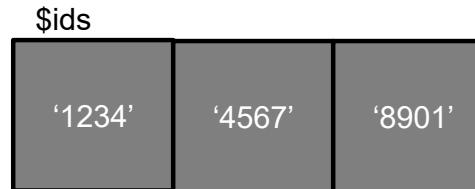




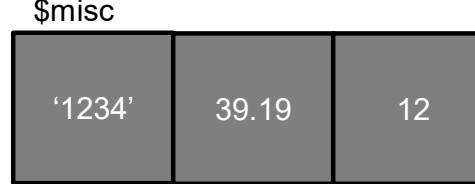
# Arrays

- Fixed-size data structures
- Used to store a collection of related values of the same type
- Stored contiguously in memory
- By default, stored as collection of objects (but can be defined with specific type)

```
$ids = @('1234', '4567', '8901')
```



```
$misc = @('1234', 39.19, 12)
```





# Defining Arrays



- To initialize an array variable, you can use @, ( ), and comma-separated values
- Alternatively, you can omit the @ and ( )

```
PS C:\Users\ a_san\source\repos> $array1 = @('A', 'B', 'C')
PS C:\Users\ a_san\source\repos> $array2 = 'D', 'E', 'F'
PS C:\Users\ a_san\source\repos> $array1.GetType().FullName
System.Object[]
PS C:\Users\ a_san\source\repos> $array2.GetType().FullName
System.Object[]
PS C:\Users\ a_san\source\repos> █
```



# Reading Array Elements



- Outputting an array variable to the console will list all values
- To read a specific element of the array, you use [ ] and a numeric index
- Index is 0-based (i.e., first element is [0] and last element is [length – 1])
- -1 will also return the last element
- ... can be used for the index to access a range of elements at once



# Reading Array Elements



```
PS C:\Users\ a_san\source\repos> $newArray = '1234', '4567', '8901'

PS C:\Users\ a_san\source\repos> $newArray

1234
4567
8901
PS C:\Users\ a_san\source\repos> $length = $newArray.Length

PS C:\Users\ a_san\source\repos> $newArray[0]

1234
PS C:\Users\ a_san\source\repos> $newArray[$length - 1]

8901
PS C:\Users\ a_san\source\repos> $newArray[-1]

8901
PS C:\Users\ a_san\source\repos> $newArray[1..2]

4567
8901
PS C:\Users\ a_san\source\repos> █
```



# Modifying Array Elements



- To change an array element, you can assign a new value to its index
- Data type must match type of elements in array
- Valid values for index are 0 to length – 1
- If assignment attempted for index out of valid range, an error occurs

```
PS C:\Users\ a_san\source\repos> $newArray  
1234  
4567  
8901  
PS C:\Users\ a_san\source\repos> $newArray[0] = '4321'  
PS C:\Users\ a_san\source\repos> $newArray[$newArray.Length - 1] = '1098'  
PS C:\Users\ a_san\source\repos> $newArray  
4321  
4567  
1098  
PS C:\Users\ a_san\source\repos> █
```

```
PS C:\Users\ a_san\source\repos> $newArray  
4321  
4567  
1098  
PS C:\Users\ a_san\source\repos> $newArray[$newArray.Length] = 'bad!'  
Index was outside the bounds of the array.  
At line:1 char:1  
+ $newArray[$newArray.Length] = 'bad!'  
+ ~~~~~~  
+ CategoryInfo          : OperationStopped: () [], IndexOutOfRangeException  
+ FullyQualifiedErrorId : System.IndexOutOfRangeException
```



# Adding Elements to an Array



- To add an element to an array, you use the + operator
- New value will be appended to end of array
- += can be used to append value and update an existing array
- + and += can be used with other arrays as well

```
PS C:\Users\ a_san\source\repos> $newArray  
4321  
4567  
1098  
PS C:\Users\ a_san\source\repos> $newArray += '0007'  
  
PS C:\Users\ a_san\source\repos> $newArray  
4321  
4567  
1098  
0007  
PS C:\Users\ a_san\source\repos> $newArray.Length  
  
4  
PS C:\Users\ a_san\source\repos> █
```

```
PS C:\Users\ a_san\source\repos> $newArray  
4321  
4567  
1098  
0007  
PS C:\Users\ a_san\source\repos> $newArray += '0003', '00004'  
  
PS C:\Users\ a_san\source\repos> $newArray  
4321  
4567  
1098  
0007  
0003  
00004  
PS C:\Users\ a_san\source\repos> $newArray.Length  
  
6  
PS C:\Users\ a_san\source\repos> █
```



# Removing Elements from an Array



- To remove an element from an array, you can't use the – operator
- Instead, you can assign the element the value `$null`
- The element will still exist, it will just have an empty value
- Will not be displayed in output but will be reflected in length

```
PS C:\Users\lisa\source\repos> $newArray  
4321  
4567  
1998  
0007  
0003  
0004  
PS C:\Users\lisa\source\repos> $newArray[3] = $null  
  
PS C:\Users\lisa\source\repos> $newArray  
4321  
4567  
1998  
0003  
0004  
PS C:\Users\lisa\source\repos> $newArray.Length  
6  
PS C:\Users\lisa\source\repos> █
```



- Arrays are fixed-size (i.e., `+=` was not changing existing array but creating a new array object to contain the result)
- With arrays of large size, this ongoing creation of new array objects to hold added elements will get costly
- ArrayLists provide an alternative data structure that can dynamically grow or shrink (not fixed-size)
- Performance and memory management is much more efficient with large collections



# Defining ArrayLists



- Created in a similar way as Arrays
- Cast as type `[System.Collections.ArrayList]`
- Because of cast, will require ( ) or @( ) to wrap values

```
PS C:\Users\ a_san\source\repos> $arrayList = [System.Collections.ArrayList]@('1234', '4567', '8901')

PS C:\Users\ a_san\source\repos> $arrayList

1234
4567
8901
PS C:\Users\ a_san\source\repos> $arrayList.Count

3
PS C:\Users\ a_san\source\repos> $arrayList.GetType().FullName

System.Collections.ArrayList
PS C:\Users\ a_san\source\repos> █
```



# Reading & Modifying ArrayList Elements



- Will use same syntax as was used with Arrays – [ ] and numeric index
- The key difference is that ArrayLists use *Count* instead of *Length* for # of elements



# Adding & Removing Elements in an ArrayList



- To add an element to an ArrayList, you use the *Add* method
- *Add* will return the index of the newly added element
- To remove, you use *Remove*
- *Remove* operates against the matched value, not an index
- Unlike Arrays, *Remove* for ArrayLists will truly reduce the size of the collection



# Adding & Removing Elements in an ArrayList



```
PS C:\Users\ a_san\source\repos> $arrayList  
  
1234  
4567  
8901  
PS C:\Users\ a_san\source\repos> $arrayList.Add('0007')  
  
3  
PS C:\Users\ a_san\source\repos> $arrayList.Add('0003')  
  
4  
PS C:\Users\ a_san\source\repos> $arrayList.Add('0004')  
  
5  
PS C:\Users\ a_san\source\repos> $arrayList  
  
1234  
4567  
8901  
0007  
0003  
0004  
PS C:\Users\ a_san\source\repos> $arrayList.Count  
  
6  
PS C:\Users\ a_san\source\repos> █
```

```
PS C:\Users\ a_san\source\repos> $arrayList  
  
1234  
4567  
8901  
0007  
0003  
0004  
PS C:\Users\ a_san\source\repos> $arrayList.Remove('0007')  
  
PS C:\Users\ a_san\source\repos> $arrayList  
  
1234  
4567  
8901  
0003  
0004  
PS C:\Users\ a_san\source\repos> $arrayList.Count  
  
5  
PS C:\Users\ a_san\source\repos> █
```



# Hashtables



- Aka dictionary – enables storage of a collection of key-value pairs
- Enables reference to elements by a key instead of numeric index
- Useful for storing related data together – e.g., counts by product number



# Defining Hashtables



- Created using @ and { } to wrap key-value combinations
- Each key-value combination can be defined on a separate line
- Alternatively, each key-value combination can be separated with a ;
- Order of elements not always in order added

```
PS C:\Users\ a_san\source\repos> $productCounts1 = @{
>> '12345' = 89
>> '67890' = 101
>> }

PS C:\Users\ a_san\source\repos> $productCounts2 = @{'00003' = 99; '0000032' = 3}

PS C:\Users\ a_san\source\repos> $productCounts = $productCounts1 + $productCounts2

PS C:\Users\ a_san\source\repos> $productCounts

Name          Value
----          -----
67890        101
12345        89
00003        99
0000032      3

PS C:\Users\ a_san\source\repos>
```



## Keys & Values



- Hashtables include a property for the collection of keys
- Hashtables also include a separate property for the collection of values
- Order of the 2 collections will correlate

```
PS C:\Users\ a_san\source\repos> $productCounts.Keys  
67890  
12345  
00003  
0000032  
PS C:\Users\ a_san\source\repos> $productCounts.Values  
101  
89  
99  
3  
PS C:\Users\ a_san\source\repos> $productCounts  
  
Name          Value  
----          ----  
67890          101  
12345          89  
00003          99  
0000032        3  
  
PS C:\Users\ a_san\source\repos> █
```



# Reading Hashtable Elements



- To reference elements of a hashtable, you can use [ ] notation or . notation
- In either case, the reference is to the key associated to the target element
- The *ContainsKey* method can be used to check for presence of a key

```
PS C:\Users\ a_san\source\repos> $productCounts

Name          Value
----          -----
67890         101
12345         89
00003         99
0000032       3

PS C:\Users\ a_san\source\repos> $productCounts['00003']

99
PS C:\Users\ a_san\source\repos> $productCounts.'00003'

99
PS C:\Users\ a_san\source\repos> $productCounts['99999']

PS C:\Users\ a_san\source\repos> $productCounts.Contains('0000032')

True
PS C:\Users\ a_san\source\repos> █
```



## Adding & Modifying Hashtable Elements



- To add a new element to hashtable, you can use the *Add* method
- Add accepts an argument for new key and value to be associated
- Alternatively, you can add a new element using the [<key>] notation
- Any attempts to add an element with a duplicate key will fail



# Adding & Modifying Hashtable Elements



```
PS C:\Users\{a_san}\source\repos> $productCounts.Add('983392', 1008)
PS C:\Users\{a_san}\source\repos> $productCounts['9993234'] = 11
PS C:\Users\{a_san}\source\repos> $productCounts

Name          Value
----          -----
12345          89
00003          99
983392         1008
9993234        11
0000032         3
67890          101

PS C:\Users\{a_san}\source\repos> $productCounts.Add('983392', 1000)
Exception calling "Add" with "2" argument(s): "Item has already been added. Key in dictionary: '983392'  Key being added: '983392'"
At line:1 char:1
+ $productCounts.Add('983392', 1000)
+ ~~~~~
+ CategoryInfo          : NotSpecified: () [], MethodInvocationException
+ FullyQualifiedErrorId : ArgumentException
PS C:\Users\{a_san}\source\repos> █
```



# Adding & Modifying Hashtable Elements



- To modify an existing element, you use the [] notation
- You can use ContainsKey first to confirm exists before updating
- Otherwise, [] notation will add a new element

```
PS C:\Users\ a_san\source\repos> if ($productCounts.ContainsKey('983392')) { $productCounts['983392'] = 1007 }

PS C:\Users\ a_san\source\repos> $productCounts

Name          Value
----          -----
12345          89
00003          99
983392         1007
9993234        11
0000032         3
67890          101

PS C:\Users\ a_san\source\repos> █
```



# Removing Elements from a Hashtable



- To remove an element from a hashtable, you use the *Remove* method
- *Remove* accepts an argument for the key to the element to be removed
- As with ArrayLists, *Remove* operates against the matched key value
- Attempts to remove a key that does not exist has no effect

```
PS C:\Users\ a_san\source\repos> $productCounts.Remove('9993234')

PS C:\Users\ a_san\source\repos> $productCounts

Name          Value
----          -----
12345          89
00003          99
983392         1007
0000032         3
67890          101

PS C:\Users\ a_san\source\repos> $productCounts.Count

5
PS C:\Users\ a_san\source\repos> $productCounts.Remove('9999999999')
```



# “Splatting” with Hashtables



- A hashtable can be used to encapsulate the parameters for a command
- The hashtable is passed as an argument to the command
- The command will map values in the hashtable to parameters on the command by name/key
- To pass the hashtable, use @<hashtable>

```
PS C:\Users\{user}\source\repos> $cmdArgs = @{
>>   Include = 'w*'
>>   Exclude = 'w32*'
>> }

PS C:\Users\{user}\source\repos> Get-Service @cmdArgs

  Status    Name            DisplayName
  -----    --name--          --displayname--
 Stopped  WaaSMedicSvc    Windows Update Medic Service
 Stopped  WalletService    WalletService
 Stopped  WarpJITSvc      WarpJITSvc
 Stopped  wbengine         Block Level Backup Engine Service
 Running   WbioSrvc        Windows Biometric Service
 Running   Wcmsvc          Windows Connection Manager
```



# Custom Objects





# Creating Custom Objects



- Uses the *PSCustomObject* type
- Enables the creation of user-defined structures with related properties
- Custom object will include common properties and methods alongside custom properties
- *Get-Member* can be used to view available members



# Creating Custom Objects



```
PS C:\Users\{a_san}\source\repos> $person = [PSCustomObject]@{
>> FirstName = 'Melissa'
>> LastName = 'Testing'
>> Age = 32
>> }

PS C:\Users\{a_san}\source\repos> $person.GetType().FullName

System.Management.Automation.PSCustomObject
PS C:\Users\{a_san}\source\repos> Get-Member -InputObject $person

TypeName: System.Management.Automation.PSCustomObject

Name      MemberType  Definition
----      -----      -----
Equals    Method     bool Equals(System.Object obj)
GetHashCode Method     int GetHashCode()
GetType   Method     type GetType()
ToString  Method     string ToString()
Age       NoteProperty int Age=32
FirstName NoteProperty string FirstName=Melissa
LastName  NoteProperty string LastName=Testing
```



# Referencing Properties of Custom Object



- Use . notation to reference user-defined properties
- Same notation used to reference common properties and methods

```
PS C:\Users\lisa\source\repos> Write-Host "$($person.FirstName) $($person.LastName) is $($person.Age) year(s) old..."  
Melissa Testing is 32 year(s) old...  
PS C:\Users\lisa\source\repos>
```



# Single Quotes vs. Double Quotes



- When working with strings, you have the option of single or double quotes
- If dealing with a simple string (static, no substitutions), either will work
- If you need to substitute runtime values for variable placeholders in strings, you must use double quotes
- This is what's known as string interpolation

```
PS C:\Users\lisa\source\repos> Write-Host '$($person.FirstName) $($person.LastName) is $($person.Age) year(s) old...'
 $($person.FirstName) $($person.LastName) is $($person.Age) year(s) old...
PS C:\Users\lisa\source\repos> Write-Host "($person.FirstName) ($person.LastName) is ($person.Age) year(s) old..."
(@{FirstName=Melissa; LastName=Testing; Age=32}.FirstName) (@{FirstName=Melissa; LastName=Testing; Age=32}.LastName) is (@{FirstName=Melissa; LastName=Testing; Age=32}.Age)
PS C:\Users\lisa\source\repos> Write-Host "$person.FirstName $person.LastName is $person.Age year(s) old..."
(@{FirstName=Melissa; LastName=Testing; Age=32}.FirstName @{{FirstName=Melissa; LastName=Testing; Age=32}.LastName} is @{{FirstName=Melissa; LastName=Testing; Age=32}.Age}
PS C:\Users\lisa\source\repos> Write-Host " $($person.FirstName) $($person.LastName) is $($person.Age) year(s) old..."

Melissa Testing is 32 year(s) old...
PS C:\Users\lisa\source\repos>
```



# The PowerShell Pipeline





# The Pipe Operator



- Using the PowerShell pipeline, we can chain commands together
- With the | operator between 2 commands, output from the first will act as input to the second
- Multiple commands can be piped together
- Last “link in the chain” pipes output to the console

```
PS C:\Users\{user}\source\repos> Get-Content -Path .\pshell\services.txt | Get-Service
```

Status	Name	DisplayName
Running	MSDTC	Distributed Transaction Coordinator
Running	WSearch	Windows Search
Running	WinRM	Windows Remote Management (WS-Manag...

```
PS C:\Users\{user}\source\repos> █
```



# The Pipe Operator



- Helps keep our code more concise
- Should probably limit chains to ~ 5 commands at most
- Literals and variables can be used as starting points for piping
- Key is that objects (rather than values) are piped between the commands

```
PS C:\Users\{user}\source\repos> 'WinRM', 'WSearch' | Get-Service | Start-Service  
PS C:\Users\{user}\source\repos> $services = 'MSDTC', 'WSearch', 'WinRM'  
PS C:\Users\{user}\source\repos> $services[0..1] | Get-Service | Start-Service  
PS C:\Users\{user}\source\repos> █
```



# Parameter Binding



- During piping, PowerShell leverages parameter binding
- Tries to match object passed in on pipe to parameters on target command
- For piping to be successful:
  - Pipeline support must be defined for at least one of the command's parameters
  - PowerShell must be able to bind according to a set of defined rules



# Parameter Binding



- Get-Help can be used to explore pipeline support for a command
- In definition of parameter, there will be an “Accept pipeline input?” entry
- If true, pipeline input is supported
- Will also indicate types of binding supported

Name parameter on *Get-Service* cmdlet

```
-Name <System.String[]>
    Specifies the service names of services to be retrieved. Wildcards are permitted. By default, this cmdlet gets all of the services on the computer.

    Required?          false
    Position?         0
    Default value     None
    Accept pipeline input?  True (ByPropertyName, ByValue)
    Accept wildcard characters? true
```



## ByValue



- Will match type of input to parameter
- Because match depends on type, only one parameter can be passed  
*ByValue*



## ByPropertyName



- Will look at object piped to command
- If input object has property with same name as parameter of command, command will use that property value

```
PS C:\Users\user\source\repos> $service = [PSCustomObject]@{Name = 'WinRM'; ComputerName = 'DESKTOP-QJENT2P'}

PS C:\Users\user\source\repos> $service | Get-Service

Status    Name          DisplayName
----    --
Running   WinRM        Windows Remote Management (WS-Manag...



PS C:\Users\user\source\repos>
```



# Processing Command Results





# Select-Object



- Command can be piped to *Select-Object* to project results to a specific set of fields
- Uses a comma-delimited list of field names for defining the set of fields on output
- Collection results can be limited using parameters (-First, -Last, -Unique, and -Skip)
- Results can be piped from and to other commands for additional processing

```
PS C:\Users\alok.san\source\repos> Get-Command -Name Select-Object -Syntax

Select-Object [[-Property] <Object[]>] [-InputObject <psobject>] [-ExcludeProperty <string[]>] [-ExpandProperty <string>] [-Unique] [-Last <int>] [-First <int>] [-Skip <int>] [-Wait] [<CommonParameters>]
Select-Object [[-Property] <Object[]>] [-InputObject <psobject>] [-ExcludeProperty <string[]>] [-ExpandProperty <string>] [-Unique] [-SkipLast <int>] [<CommonParameters>]
Select-Object [-InputObject <psobject>] [-Unique] [-Wait] [-Index <int[]>] [<CommonParameters>]
```



# Select-Object



```
PS C:\Users\alansan\source\repos> Get-Service | Select-Object -First 1 | Get-Member

TypeName: System.ServiceProcess.ServiceController

Name          MemberType  Definition
----          -----      -----
Name          AliasProperty  Name = ServiceName
RequiredServices AliasProperty  RequiredServices = ServicesDependedOn
Disposed       Event        System.EventHandler Disposed(System.Object, System.EventArgs)
Close         Method       void Close()
Continue      Method       void Continue()
CreateObjRef   Method      System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose        Method      void Dispose(), void IDisposable.Dispose()
Equals         Method      bool Equals(System.Object obj)
ExecuteCommand Method     void ExecuteCommand(int command)
GetHashCode    Method      int GetHashCode()
GetLifetimeService Method   System.Object GetLifetimeService()
GetType        Method      type GetType()
InitializeLifetimeService Method System.Object InitializeLifetimeService()
Pause          Method      void Pause()
Refresh        Method      void Refresh()
Start          Method      void Start(), void Start(string[] args)
Stop           Method      void Stop()
WaitForStatus  Method      void WaitForStatus(System.ServiceProcess.ServiceControllerStatus desiredStatus), void WaitForS
CanPauseAndContinue Property  bool CanPauseAndContinue {get;}
CanShutdown    Property  bool CanShutdown {get;}
CanStop        Property  bool CanStop {get;}
Container      Property  System.ComponentModel.IContainer Container {get;}
DependentServices Property  System.ServiceProcess.ServiceController[] DependentServices {get;}
DisplayName    Property  string DisplayName {get;set;}
MachineName   Property  string MachineName {get;set;}
ServiceHandle  Property  System.Runtime.InteropServices.SafeHandle ServiceHandle {get;}
ServiceName   Property  string ServiceName {get;set;}
ServicesDependedOn Property  System.ServiceProcess.ServiceController[] ServicesDependedOn {get;}
ServiceType    Property  System.ServiceProcess.ServiceType ServiceType {get;}
Site          Property  System.ComponentModel.ISite Site {get;set;}
StartType     Property  System.ServiceProcess.ServiceStartMode StartType {get;}
Status        Property  System.ServiceProcess.ServiceControllerStatus Status {get;}
ToString      ScriptMethod System.Object ToString();
```



# Select-Object



```
PS C:\Users\{user}\source\repos> Get-Service | Select-Object -First 5 -Property Name, DisplayName, MachineName, Status, DependentServices

Name          : AarSvc_54801
DisplayName   : AarSvc_54801
MachineName   : .
Status        : Stopped
DependentServices : {}

Name          : AJRouter
DisplayName   : AllJoyn Router Service
MachineName   : .
Status        : Stopped
DependentServices : {}

Name          : ALG
DisplayName   : Application Layer Gateway Service
MachineName   : .
Status        : Stopped
DependentServices : {}

Name          : Alienware Digital Delivery Services
DisplayName   : Alienware Digital Delivery Services
MachineName   : .
Status        : Running
DependentServices : {}

Name          : Alienware SupportAssist Remediation
DisplayName   : Alienware SupportAssist Remediation
MachineName   : .
Status        : Running
DependentServices : {}
```



# Select-Object



```
PS C:\Users\{user}\source\repos> Get-Content -Path .\pshell\services.txt | Get-Service

Status    Name          DisplayName
----      --           -----
Running   MSDTC         Distributed Transaction Coordinator
Running   WSearch        Windows Search
Running   WinRM         Windows Remote Management (WS-Manag...

PS C:\Users\{user}\source\repos> Get-Content -Path .\pshell\services.txt | Select-Object -Skip 1 | Get-Service

Status    Name          DisplayName
----      --           -----
Running   WSearch        Windows Search
Running   WinRM         Windows Remote Management (WS-Manag...

PS C:\Users\{user}\source\repos>
```



# Select-Object



```
PS C:\Users\{a_san}\source\repos> 'WSearch', 'WinRM' | Get-Service | Select-Object -Property Name, DisplayName, MachineName, Status, @{label='Num Dep Service'; expression={$_.DependentServices.Length}}
```

```
Name      : WSearch
DisplayName : Windows Search
MachineName : .
Status    : Running
Num Dep Service : 2

Name      : WinRM
DisplayName : Windows Remote Management (WS-Management)
MachineName : .
Status    : Running
Num Dep Service : 0

PS C:\Users\{a_san}\source\repos> █
```



# Select-Object



```
PS C:\Users\ a_san\source\repos> 'WSearch' | Get-Service | Select-Object -ExpandProperty DependentServices

Status    Name          DisplayName
----    --   -----
Stopped  workfolderssvc  Work Folders
Stopped  WMPNetworkSvc  Windows Media Player Network Sharin...
```

```
PS C:\Users\ a_san\source\repos> █
```



## Sort-Object



- Command can be piped to *Sort-Object* to sort results by property values
- Able to define property on which results should be sorted
- Default sort order is ascending but a “switch” parameter can be used to change sort order to descending



# Sort-Object



- By default, sorts are case-insensitive but “switch” parameter can be used to modify
- If intent is to sort by multiple properties, use a hashtable

```
PS C:\Users\ a_san\source\repos> Get-Command -Name Sort-Object -Syntax

Sort-Object [[-Property] <Object[]>] [-Descending] [-Unique] [-InputObject <psobject>] [-Culture <string>] [-CaseSensitive] [<CommonParameters>]

PS C:\Users\ a_san\source\repos> █
```



# Sort-Object



```
PS C:\Users\{a_san}\source\repos> Get-Process | Sort-Object -Property CPU -Descending | Select-Object -First 10
```

Handles	NPM(K)	PM(K)	WS(K)	CPU(s)	Id	SI	ProcessName
2720	127	1699696	415124	5,552.72	1828	1	dwm
827	83	693484	125536	4,302.44	6620	0	sqlservr
1660	57	673200	66548	4,135.41	16056	1	googledrivesync
8527	0	236	19612	3,291.47	4	0	System
2886	23	41916	41752	2,808.06	2796	0	svchost
1754	127	308976	253768	2,193.44	17196	1	Skype
3619	105	292376	373012	2,008.72	25068	1	POWERPNT
7759	916	210360	295440	1,906.78	5392	1	msedge
865	46	45584	50676	1,888.72	16072	0	XtuService
3831	122	392224	40048	1,870.55	5776	0	NortonSecurity

```
PS C:\Users\{a_san}\source\repos> []
```



## Where-Object



- Selects objects from a collection using different types of comparison against property values
- Includes a pretty feature-rich set of comparison operators (equals, like, not like, less than, greater than, etc.)
- Comparison operations also include case-sensitive versions (prepended with a “c”)



## Where-Object



- Comparison supports a couple of different formats:
  - `-Property Name -EQ -Value "win"`
  - `$_.Name -eq "win"`
- Piping to Where-Object commands can be chained to narrow the filter against a collection
- Alternatively, you can leverage combinations of –and and –or operators to build a multi-condition filter

Run `Get-Command -Name Where-Object -Syntax` to see options



# Where-Object



```
PS C:\Users\{user}\source\repos> Get-Service | Where-Object {$_.Status -eq 'Stopped'} | Select-Object -First 10
```

Status	Name	DisplayName
-----	-----	-----
Stopped	AarSvc_54801	AarSvc_54801
Stopped	AJRouter	AllJoyn Router Service
Stopped	ALG	Application Layer Gateway Service
Stopped	AppIDSvc	Application Identity
Stopped	AppMgmt	Application Management
Stopped	AppReadiness	App Readiness
Stopped	AppVClient	Microsoft App-V Client
Stopped	aspnet_state	ASP.NET State Service
Stopped	AssignedAccessM...	AssignedAccessManager Service
Stopped	autotimesvc	Cellular Time

```
PS C:\Users\{user}\source\repos> █
```



# Where-Object



```
PS C:\Users\{user}\source\repos> Get-Service | Where-Object Status -eq 'Stopped' | Select-Object -First 10
```

Status	Name	DisplayName
Stopped	AarSvc_54801	AarSvc_54801
Stopped	AJRouter	AllJoyn Router Service
Stopped	ALG	Application Layer Gateway Service
Stopped	AppIDSvc	Application Identity
Stopped	AppMgmt	Application Management
Stopped	AppReadiness	App Readiness
Stopped	AppVClient	Microsoft App-V Client
Stopped	aspnet_state	ASP.NET State Service
Stopped	AssignedAccessM...	AssignedAccessManager Service
Stopped	autotimesvc	Cellular Time

```
PS C:\Users\{user}\source\repos> █
```



# Where-Object



```
PS C:\Users\{user}\source\repos> Get-Module -ListAvailable | Where-Object {$_ .Name -notlike "Microsoft*" -and !$_.HelpInfoUri} | Select-Object -Property Name, HelpInfoUri
```

Name	HelpInfoUri
Azure	
Azure	
Azure.AnalysisServices	
Azure.Storage	
AzureRM	
AzureRM.AnalysisServices	
AzureRM.ApiManagement	
AzureRM.ApplicationInsights	
AzureRM.Automation	
AzureRM.Backup	
AzureRM.Batch	
AzureRM.Billing	
AzureRM.Cdn	
AzureRM.CognitiveServices	
AzureRM.Compute	
AzureRM.Consumption	
AzureRM.ContainerInstance	
AzureRM.ContainerRegistry	
AzureRM.DataFactories	
AzureRM.DataFactoryV2	
AzureRM.DataLakeAnalytics	
AzureRM.DataLakeStore	



# Where-Object



```
PS C:\Users\{a_san}\source\repos> Get-Module -ListAvailable | Where-Object {($_.Name -notlike "Microsoft*" -and $_.Name -notlike "Azure*") -and !$_.HelpInfoUri} | Select-Object -Property Name, HelpInfoUri

Name          HelpInfoUri
----          -----
Pester
DeliveryOptimization
PersistentMemory
PS.Diagnostics
StorageBusCache
SQLPS
Plaster
PowerShellEditorServices
PowerShellEditorServices.VSCode
PSScriptAnalyzer

PS C:\Users\{a_san}\source\repos>
```



## Measure-Object



- Calculates a set of measures against property value(s) of objects
- Can be used to count objects (optionally against a specific property)
- Also, for numeric collections supports min, max, average, sum and standard deviation
- For text objects, supports measurement of lines, words and characters



# Measure-Object



- By default, measure will return a list containing count only
- Using parameters of the command, you can enhance what's returned
- If you target a specific measure that is inconsistent with the type of data in the collection, you will receive an error
- Enables collection-level processing without using a loop

```
PS C:\Users\{a_san}\source\repos\pshell> Get-Command -Name Measure-Object -Syntax
Measure-Object [[-Property] <string[]>] [-InputObject <psobject>] [-Sum] [-Average] [-Maximum] [-Minimum] [<CommonParameters>]
Measure-Object [[-Property] <string[]>] [-InputObject <psobject>] [-Line] [-Word] [-Character] [-IgnoreWhiteSpace] [<CommonParameters>]
PS C:\Users\{a_san}\source\repos\pshell> █
```



# Measure-Object



```
PS C:\Users\ a_san\source\repos\pshell> $array = 1.3, 3.45, -18.99, 22.01  
PS C:\Users\ a_san\source\repos\pshell> $array | Measure-Object
```

```
Count      : 4  
Average    :  
Sum        :  
Maximum    :  
Minimum    :  
Property   :  
  
Count      : 4  
Average    :  
Sum        : 7.77  
Maximum    :  
Minimum    :  
Property   :
```

```
PS C:\Users\ a_san\source\repos\pshell> $array | Measure-Object -Sum
```

```
PS C:\Users\ a_san\source\repos\pshell> $array | Measure-Object -Sum -Average -Minimum
```

```
Count      : 4  
Average    : 1.9425  
Sum        : 7.77  
Maximum    :  
Minimum    : -18.99  
Property   :
```

```
PS C:\Users\ a_san\source\repos\pshell> █
```



# Measure-Object



```
PS C:\Users\ a_san\source\repos\pshell> 'The quick brown fox jumped over the lazy dog' | Measure-Object

Count      : 1
Average    :
Sum        :
Maximum   :
Minimum   :
Property  :

PS C:\Users\ a_san\source\repos\pshell> 'The quick brown fox jumped over the lazy dog' | Measure-Object -Word -Line -Character

Lines Words Characters Property
-----
 1      9       44

PS C:\Users\ a_san\source\repos\pshell>
```



# Measure-Object



```
PS C:\Users\{a_san}\source\repos\pshell> 'The quick brown fox jumped over the lazy dog' | Measure-Object -Sum
Measure-Object : Input object "The quick brown fox jumped over the lazy dog" is not numeric.
At line:1 char:58
+ 'The quick brown fox jumped over the lazy dog' | Measure-Object -Sum
+               ~~~~~~
+ CategoryInfo          : InvalidType: (The quick brown...er the lazy dog:String) [Measure-Object], PSInvalidOperationException
+ FullyQualifiedErrorId : NonNumericInputObject,Microsoft.PowerShell.Commands.MeasureObjectCommand

Count      : 1
Average    :
Sum        :
Maximum   :
Minimum   :
Property  :
```

```
PS C:\Users\{a_san}\source\repos\pshell> █
```



## ForEach-Object



- Allows execution of either a ScriptBlock or other statement against each element of a piped collection
- Statement supports gather of property value without changing its type
- Statement can also include execution of a method defined on the element
- Arguments to the method are also supported via *ArgumentList* parameter



# ForEach-Object



- For scripting logic to be run against each element, options include:
  - *Begin* – executed at the beginning of collection processing
  - *Process* – executed once for each element of the collection
  - *End* – executed at the end of collection processing
- *Process* supports an array of multiple ScriptBlocks to be executed in sequence

```
PS C:\Users\al_san\source\repos\pshell> Get-Command -Name ForEach-Object -Syntax
ForEach-Object [-Process] <scriptblock[]> [-InputObject <psobject>] [-Begin <scriptblock>] [-End <scriptblock>] [-RemainingScripts <scriptblock[]>] [-WhatIf] [-Confirm] [<CommonParameters>]
ForEach-Object [-MemberName] <string> [-InputObject <psobject>] [-ArgumentList <Object[]>] [-WhatIf] [-Confirm] [<CommonParameters>]
PS C:\Users\al_san\source\repos\pshell> █
```



# ForEach-Object



```
PS C:\Users\ a_san\source\repos\pshell> $total = 0
PS C:\Users\ a_san\source\repos\pshell> 1..100 | ForEach-Object -Begin {Write-Host 'Gathering sum'} -Process {$total += $_} -End {Write-Host 'Done with sum'}
Gathering sum
Done with sum
PS C:\Users\ a_san\source\repos\pshell> $total
5050
PS C:\Users\ a_san\source\repos\pshell> ■
```

```
PS C:\Users\ a_san\source\repos\pshell> 13.34342, 19.234234 | ForEach-Object -MemberName ToString -ArgumentList 'C'
$13.34
$19.23
PS C:\Users\ a_san\source\repos\pshell>
```



## PS Providers & PS Drives





- .NET programs providing access to specialized data stores
- Intended to facilitate view and management of different types of data
- The data appears in a “drive” – provides virtual path for a data type



# PSProviders



- Enables use of same commands used with files with other types of data
- Supports creation of your own or installation of those created by others
- *Get-PSPProvider* lists available providers

```
PS C:\Users\ a_san\source\repos\psshell> Get-PSPProvider

Name          Capabilities           Drives
----          -----
Registry      ShouldProcess, Transactions {HKLM, HKCU}
Alias         ShouldProcess          {Alias}
Environment   ShouldProcess          {Env}
FileSystem    Filter, ShouldProcess, Credentials {C, D}
Function      ShouldProcess          {Function}
Variable      ShouldProcess          {Variable}

PS C:\Users\ a_san\source\repos\psshell> █
```



## PSDrives



- Data store location that can be accessed like a file system drive
- Commands exist for navigating the drive and its hierarchy (if applicable)
- Supports creation of custom drives for targeted access



## PSDrives



- Reference to drive is via “<drivename>:” (like C: for files)
- *Get-PSDrive* can be used to list available drives

```
PS C:\Users\{a_san}\source\repos\pshell> Get-PSDrive

Name      Used (GB)   Free (GB) Provider      Root
----      -----      -----      -----
Alias          Alias
C           188.27    749.44 FileSystem   C:\\
Cert          Certificate
D           373.76    1489.25 FileSystem  D:\\
Env           Environment
Function        Function
HKCU          Registry   HKEY_CURRENT_USER
HKLM          Registry   HKEY_LOCAL_MACHINE
Variable        Variable
WSMan          WSMAN
```

```
PS C:\Users\{a_san}\source\repos\pshell> █
```



Drive Name	Brief Description
Alias	Provides access to available command aliases
C, D, etc.	Provides access to filesystem drives
Cert	Provides access to certificate store
Env	Provides access to environment variables
Function	Provides access to defined functions
HKCU, HKLM	Provides access to Registry keys
Variable	Provides access to defined variables
WSMan	Provides access to management configuration data (on local & remote systems)



```
PS C:\Users\{a_san}\source\repos\pshell> $myVar = 42
PS C:\Users\{a_san}\source\repos\pshell> Get-ChildItem -Path Variable:
Name          Value
----          -----
$              42
?              True
^              $myVar
args           {}
ConfirmPreference High
ConsoleFileName
DebugPreference SilentlyContinue
Error          {}
ErrorActionPreference Continue
ErrorView       NormalView
ExecutionContext System.Management.Automation.EngineIntrinsics
false          False
FormatEnumerationLimit 4
HOME           C:\Users\{a_san}
Host            System.Management.Automation.Internal.Host.InternalHost
InformationPreference SilentlyContinue
input           System.Collections.ArrayList+ArrayListEnumeratorSimple
MaximumAliasCount 4096
MaximumDriveCount 4096
MaximumErrorCount 256
MaximumFunctionCount 4096
MaximumHistoryCount 4096
MaximumVariableCount 4096
MyInvocation     System.Management.Automation.InvocationInfo
myVar           42
NestedPromptLevel 0
null            null
OutputEncoding   System.Text.ASCIIEncoding
PID             25216
PROFILE         C:\Users\{a_san}\OneDrive\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1
ProgressPreference Continue
```



```
PS C:\Users\ a_san\source\repos\pshell> Get-ChildItem -Path $HOME

Directory: C:\Users\ a_san

Mode                LastWriteTime      Length Name
----                -----          ---- 
d-----        11/21/2020  9:39 AM           .android
d-----        11/20/2020  5:50 PM           .dotnet
d-----        11/21/2020  9:28 AM           .nuget
d-----        11/21/2020  9:21 AM           .templateengine
d-----        12/7/2020   9:51 AM           .tobii
d-----        11/20/2020  2:05 PM           .vscode
d-r---        11/20/2020  11:02 AM          3D Objects
d-r---        11/20/2020  11:02 AM          Contacts
d-----        11/21/2020  11:44 AM          Documents
d-r---        12/7/2020   8:55 PM           Downloads
d-r---        11/20/2020  11:02 AM          Favorites
d-r---        11/21/2020  6:16 PM            Links
d-----        12/5/2020   11:17 AM          Microsoft
d-r---        11/20/2020  11:02 AM          Music
dar--l       12/5/2020   9:51 AM           OneDrive
d-----        11/21/2020  8:12 PM           Postman
d-r---        11/20/2020  11:02 AM          Saved Games
d-r---        11/20/2020  11:03 AM          Searches
d-----        11/20/2020  5:36 PM            source
d-r---        11/20/2020  12:01 PM           Videos
d-----        12/4/2020   4:08 PM           VM Images
-a---        11/21/2020  9:36 AM          16 .emulator_console_auth_token

PS C:\Users\ a_san\source\repos\pshell>
```



```
PS C:\Users\{a_san}\source\repos\psshell>
PS C:\Users\{a_san}\source\repos\psshell> Get-ChildItem -Path HKCU:\SOFTWARE\Microsoft\Windows
```

```
Hive: HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows
```

Name	Property
---	-----
AssignedAccessConfiguration	
CurrentVersion	
DWM	Composition : 1 ColorizationColor : 3293334087 ColorizationColorBalance : 89 ColorizationAfterglow : 3293334087 ColorizationAfterglowBalance : 10 ColorizationBlurBalance : 1 EnableWindowColorization : 0 ColorizationGlassAttribute : 1 AccentColor : 4282862156 ColorPrevalence : 0 EnableAeroPeek : 1 AlwaysHibernateThumbnails : 0
Shell	
ShellNoRoam	
TabletPC	
Windows Error Reporting	LastRateLimitedDumpGenerationTime : 13251842218783331
Winlogon	

```
PS C:\Users\{a_san}\source\repos\psshell> █
```



## Adding New Drives



- Using `New-PSDrive`, you can create a new drive “mapping”
- At creation, you specify name, associated provider, and drive root
- Can use to create your own aliases for common paths
- As with other entities, drive exists in current session only (by default)



# Adding New Drives



```
PS C:\Users\{a_san}\source\repos\pshell>
PS C:\Users\{a_san}\source\repos\pshell> New-PSDrive -Name git -PSProvider FileSystem -Root C:\Users\{a_san}\source\repos\
Name          Used (GB)    Free (GB) Provider      Root
----          -----       -----   FileSystem    C:\Users\{a_san}\source\repos\
git           0.00        749.44
```

```
PS C:\Users\{a_san}\source\repos\pshell> Get-PSDrive
Name          Used (GB)    Free (GB) Provider      Root
----          -----       -----   FileSystem    C:\_
Alias          Alias
C              188.27     749.44 FileSystem    C:\
Cert           Certificate \
D              373.76     1489.25 FileSystem   D:\
Env            Environment \
Function        Function \
git            0.00        749.44 FileSystem   C:\Users\{a_san}\source\repos\
HKCU           Registry    HKEY_CURRENT_USER
HKLM           Registry    HKEY_LOCAL_MACHINE
Variable        Variable \
WSMan          WSMAN
```

```
PS C:\Users\{a_san}\source\repos\pshell> Get-ChildItem -Path git:
Directory: C:\Users\{a_san}\source\repos

Mode          LastWriteTime      Length Name
----          -----           ----- 
d----         12/7/2020 6:41 PM      0 pshell
d----         12/6/2020 1:27 PM      0 pshell-core
d----         12/7/2020 7:04 PM      0 pshell-core-public

PS C:\Users\{a_san}\source\repos\pshell>
```



# Deleting Custom Drives



- Using *Remove-PSDrive*, you can delete a drive by name
- Cannot delete the drive while you are currently in the drive (i.e., Set-Location)

```
PS C:\Users\user\source\repos\pshell> Set-Location -Path git:  
PS git:> Remove-PSDrive -Name git  
Remove-PSDrive : Cannot remove drive 'git' because it is in use.  
At line:1 char:1  
+ Remove-PSDrive -Name git  
+ ~~~~~~  
+ CategoryInfo          : InvalidOperationException: () [Remove-PSDrive], PSInvalidOperationException  
+ FullyQualifiedErrorId : InvalidOperation,Microsoft.PowerShell.Commands.RemovePSDriveCommand  
PS git:> █
```



# Working with Files & Folders





## Test-Path

- Used to check whether all elements in a path exist
- Returns true if so; false, otherwise
- Can be used to test before executing other file operations

```
PS C:\Users\ a_san\source\repos\pshell>
PS C:\Users\ a_san\source\repos\pshell> Test-Path -Path $HOME
True
PS C:\Users\ a_san\source\repos\pshell> Test-Path -Path C:\I\Do\No\Exist\
False
PS C:\Users\ a_san\source\repos\pshell> █
```



# New-Item



- Can be used to create a new directory or folder
- For a file, can also optionally be used to set its value at creation time
- *-Path*, *-Name*, and *-ItemType* parameters are used to describe
- When creating file, all parts of target path must exist (i.e. folders)

```
PS C:\Users\al_San\source\repos\pshell>
PS C:\Users\al_San\source\repos\pshell> Get-Command -Name New-Item -Syntax

New-Item [-Path] <string[]> [-ItemType <string>] [-Value <Object>] [-Force] [-Credential <pscredential>] [-WhatIf] [-Confirm] [-UseTransaction] [<CommonParameters>]

New-Item [[-Path] <string[]>] -Name <string> [-ItemType <string>] [-Value <Object>] [-Force] [-Credential <pscredential>] [-WhatIf] [-Confirm] [-UseTransaction] [<CommonParameters>]

PS C:\Users\al_San\source\repos\pshell> █
```



# New-Item



```
PS C:\Users\{a_san}\source\repos\pshell>
PS C:\Users\{a_san}\source\repos\pshell> New-Item -Path . -Name 'new-folder' -ItemType Directory

Directory: C:\Users\{a_san}\source\repos\pshell

Mode                LastWriteTime         Length Name
----                -----          ----- 
d----    12/7/2020 11:25 PM            new-folder

PS C:\Users\{a_san}\source\repos\pshell> New-Item -Path .\new-folder\ -Name 'test.txt' -ItemType File

Directory: C:\Users\{a_san}\source\repos\pshell\new-folder

Mode                LastWriteTime         Length Name
----                -----          ----- 
-a----   12/7/2020 11:25 PM            0 test.txt

PS C:\Users\{a_san}\source\repos\pshell> New-Item -Path .\new-folder-2\ -Name 'test2.txt' -ItemType File
New-Item : Could not find a part of the path 'C:\Users\{a_san}\source\repos\pshell\new-folder-2\test2.txt'.
At line:1 char:1
+ New-Item -Path .\new-folder-2\ -Name 'test2.txt' -ItemType File
+ ~~~~~~
+     + CategoryInfo          : WriteError: (C:\Users\{a_san}\...der-2\test2.txt:String) [New-Item], DirectoryNotFoundException
+     + FullyQualifiedErrorId : NewItemIOError,Microsoft.PowerShell.Commands.NewItemCommand

PS C:\Users\{a_san}\source\repos\pshell>
```



# Set-Location



- *Set-Location* can be used to change to a specific path (like *cd*)
- In that location, you can navigate its hierarchy using *cd* or *Set-Location*
- Alternatively, you can use *-Path* parameter of other commands

```
PS git:\> ls

Directory: C:\Users\ a_san\source\repos

Mode                LastWriteTime     Length Name
----                -----          ---- 
d-----        12/7/2020   6:41 PM          pshell
d-----        12/6/2020   1:27 PM          pshell-core
d-----        12/7/2020   7:04 PM          pshell-core-public

PS git:\> cd .\pshell\
PS git:\pshell> cd ..
PS git:\> cd ..
PS git:\> █
```



## Get-ChildItem



- *Get-ChildItem* can be used to list contents of a drive or path
- *ls* and *dir* are aliases for the command
- Provides a *-Recurse* parameter to include sub-locations



# Get-ChildItem



- Provides a *-Force* parameter to list hidden items
- *-Exclude* can be used to exclude files by name
- *-Path* and *-Exclude* support wildcards

```
PS C:\Users\l_a_san\source\repos\pshell>
PS C:\Users\l_a_san\source\repos\pshell> Get-Command -Name Get-ChildItem -Syntax

Get-ChildItem [[-Path] <string[]>] [[-Filter] <string>] [-Include <string[]>] [-Exclude <string[]>] [-Recurse] [-Depth <uint32>] [-Force] [-Name] [-UseTransaction] [-Attributes <FlagsExpression[FileAttributes]>] [-Directory] [-File] [-Hidden] [-ReadOnly] [-System] [<CommonParameters>]

Get-ChildItem [[-Filter] <string>] -LiteralPath <string[]> [-Include <string[]>] [-Exclude <string[]>] [-Recurse] [-Depth <uint32>] [-Force] [-Name] [-UseTransaction] [-Attributes <FlagsExpression[FileAttributes]>] [-Directory] [-File] [-Hidden] [-ReadOnly] [-System] [<CommonParameters>]

PS C:\Users\l_a_san\source\repos\pshell> █
```



# Get-ChildItem



```
PS C:\Users\ a_san\source\repos\pshell>
PS C:\Users\ a_san\source\repos\pshell> Get-ChildItem -Path C:\Windows -Recurse -Force -Exclude *.dll | Select-Object -First 5
```

```
Directory: C:\Windows
```

Mode	LastWriteTime	Length	Name
d----	12/7/2019 4:50 AM		addons

```
Directory: C:\Windows\addons
```

Mode	LastWriteTime	Length	Name
-a---	4/18/2019 2:49 PM	802	FXSEXT.ecf

```
Directory: C:\Windows
```

Mode	LastWriteTime	Length	Name
d----	11/21/2020 9:17 AM		appcompat



# Set-Content



- Used for writing objects to a file
- Writes new content or replaces existing content
- Supports use of `-Value` parameter or can pipe content to the command
- Will create file if does not exist

```
PS C:\Users\{a_san}\source\repos\pshell>
PS C:\Users\{a_san}\source\repos\pshell> Get-Command -Name Set-Content -Syntax

Set-Content [-Path] <string[]> [-Value] <Object[]> [-PassThru] [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-WhatIf] [-Confirm] [-UseTransaction] [-NoNewline] [-Encoding <FileSystemCmdletProviderEncoding>] [-Stream <string>] [<CommonParameters>]

Set-Content [-Value] <Object[]> -LiteralPath <string[]> [-PassThru] [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-WhatIf] [-Confirm] [-UseTransaction] [-NoNewline] [-Encoding <FileSystemCmdletProviderEncoding>] [-Stream <string>] [<CommonParameters>]

PS C:\Users\{a_san}\source\repos\pshell> █
```



## Add-Content



- Used for appending objects to a file
- Does not replace existing content
- Supports use of `-Value` parameter or can pipe content to the command
- Will create file if does not exist

```
PS C:\Users\{user}\source\repos\pshell> Get-Command -Name Add-Content -Syntax

Add-Content [-Path] <string[]> [-Value] <Object[]> [-PassThru] [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-WhatIf] [-Confirm] [-UseTransaction] [-NoNewline] [-Encoding <FileSystemCmdletProviderEncoding>] [-Stream <string>] [<CommonParameters>]

Add-Content [-Value] <Object[]> -LiteralPath <string[]> [-PassThru] [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>] [-Force] [-Credential <pscredential>] [-WhatIf] [-Confirm] [-UseTransaction] [-NoNewline] [-Encoding <FileSystemCmdletProviderEncoding>] [-Stream <string>] [<CommonParameters>]

PS C:\Users\{user}\source\repos\pshell> █
```



# Get-Content



- Used for retrieving contents of file
- Content will be returned as an array with each line representing an element
- *Get-Content* is often used to pipe a list of things from a file to other commands

```
PS C:\Users\A_San\source\repos\pshell>
PS C:\Users\A_San\source\repos\pshell> Get-Command -Name Get-Content -Syntax

Get-Content [-Path] <string[]> [-ReadCount <long>] [-TotalCount <long>] [-Tail <int>] [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>]
[-Force] [-Credential <pscredential>] [-UseTransaction] [-Delimiter <string>] [-Wait] [-Raw] [-Encoding <FileSystemCmdletProviderEncoding>] [-Stream <string>] [<CommonParameters>]

Get-Content -LiteralPath <string[]> [-ReadCount <long>] [-TotalCount <long>] [-Tail <int>] [-Filter <string>] [-Include <string[]>] [-Exclude <string[]>]
[-Force] [-Credential <pscredential>] [-UseTransaction] [-Delimiter <string>] [-Wait] [-Raw] [-Encoding <FileSystemCmdletProviderEncoding>] [-Stream <string>] [<CommonParameters>]

PS C:\Users\A_San\source\repos\pshell>
```



# Get-Content



```
services.txt X
services.txt
1 MSDTC
2 WSearch
3 WinRM
```

```
PS C:\Users\{a_san}\source\repos\pshell>
PS C:\Users\{a_san}\source\repos\pshell> $contents = Get-Content -Path .\services.txt
PS C:\Users\{a_san}\source\repos\pshell> $contents
MSDTC
WSearch
WinRM
PS C:\Users\{a_san}\source\repos\pshell> $contents.Length
3
PS C:\Users\{a_san}\source\repos\pshell> $contents[1]
WSearch
PS C:\Users\{a_san}\source\repos\pshell> █
```



# CSV, HTML, and JSON





## Import-Csv



- Used for importing CSV file contents into memory
- Parses content into an array of PSCustomObjects
- Each header value is translated to a property on the object
- Row value for a given column header can be accessed using [ ] and . notation



# Import-Csv



- “Records” can be piped to other commands for filter, projection on import
- *-Delimiter* parameter allows specification of a different delimiter
- *-Header* parameter allows change to header identifiers

```
PS C:\Users\al_san\source\repos\pshell>
PS C:\Users\al_san\source\repos\pshell> Get-Command -Name Import-Csv -Syntax

Import-Csv [[-Path] <string[]>] [[-Delimiter] <char>] [-LiteralPath <string[]>] [-Header <string[]>] [-Encoding <string>] [<CommonParameters>]
Import-Csv [[-Path] <string[]>] -UseCulture [-LiteralPath <string[]>] [-Header <string[]>] [-Encoding <string>] [<CommonParameters>]
PS C:\Users\al_san\source\repos\pshell> █
```



# Import-Csv



A screenshot of a Windows PowerShell window titled "test.csv". The window shows a CSV file with four rows of data and a subsequent PowerShell session. The CSV file contains:

	FirstName	LastName	Age
1	"Joe"	"Smith"	42
2	"Melissa"	"Testing"	39
3			

The PowerShell session starts with the command `$people = Import-Csv -Path .\test.csv -Header "fname", "lname", "age"`. It then displays the contents of \$people as a table:

fname	lname	age
Joe	Smith	42
Melissa	Testing	39

Finally, the command `$people[1].lname` is run, outputting "Smith".



# Export-Csv



- Used for exporting data to a CSV file
- Can be used for storing projected results from another command
- Property names used as headers and property values become rows
- Useful for data reporting for available collections

```
PS C:\Users\al_san\source\repos\pshell>
PS C:\Users\al_san\source\repos\pshell> Get-Command -Name Export-Csv -Syntax
Export-Csv [[-Path] <string>] [[-Delimiter] <char>] -InputObject <psobject> [-LiteralPath <string>] [-Force] [-NoClobber] [-Encoding <string>] [-Append] [-NoTypeInformation] [-WhatIf] [-Confirm] [<CommonParameters>]
Export-Csv [[-Path] <string>] -InputObject <psobject> [-LiteralPath <string>] [-Force] [-NoClobber] [-Encoding <string>] [-Append] [-UseCulture] [-NoTypeInformation] [-WhatIf] [-Confirm] [<CommonParameters>]
PS C:\Users\al_san\source\repos\pshell> █
```



# Export-Csv



```
PS C:\Users\l_a_san\source\repos\pshell>
PS C:\Users\l_a_san\source\repos\pshell>
PS C:\Users\l_a_san\source\repos\pshell> Get-Process | Select-Object -Property Name,Company,Description |
>> Export-Csv -Path .\processes.csv -NoTypeInformation
PS C:\Users\l_a_san\source\repos\pshell>
```

```
processes.csv
processes.csv
1 "Name","Company","Description"
2 "AlienwareOn-ScreenDisplay","Alienware Corp.", "Alienware On-Screen Display"
3 "Amazon Music Helper","Amazon.com Services LLC", "Amazon Music Helper"
4 "ApplicationFrameHost", "Microsoft Corporation", "Application Frame Host"
5 "AppVShNotify", "Microsoft Corporation", "Microsoft Application Virtualization Client Shell Notifier"
6 "AppVShNotify", "Microsoft Corporation", "Microsoft Application Virtualization Client Shell Notifier"
7 "audiogd", "Microsoft Corporation", "Windows Audio Device Graph Isolation "
8 "AWCC", "Dell Technologies", "AWCC"
9 "AWCC.Background.Server", "Dell Technologies", "AWCC.Background.Server"
10 "AWCC.Service", "Dell Technologies", "AWCC.Service"
11 "Calculator", "Microsoft Corporation", "Calculator.exe"
12 "chrome", "Google LLC", "Google Chrome"
13 "chrome", "Google LLC", "Google Chrome"
14 "chrome", "Google LLC", "Google Chrome"
15 "chrome", "Google LLC", "Google Chrome"
16 "chrome", "Google LLC", "Google Chrome"
17 "chrome", "Google LLC", "Google Chrome"
18 "chrome", "Google LLC", "Google Chrome"
19 "chrome", "Google LLC", "Google Chrome"
20 "chrome", "Google LLC", "Google Chrome"
21 "chrome", "Google LLC", "Google Chrome"
22 "chrome", "Google LLC", "Google Chrome"
23 "chrome", "Google LLC", "Google Chrome"
```



# ConvertTo-Html



- Allows specification of fields to include on result (using *-Property*)
- Supports push to an output filename using > “<filepath>\<filename>.htm”
- Can be used to quickly provide results in a more visual appealing manner
- Can pipe output from another command for html display

```
PS C:\Users\ a_san\source\repos\pshell>
PS C:\Users\ a_san\source\repos\pshell> Get-Command -Name ConvertTo-Html -Syntax

ConvertTo-Html [[-Property] <Object[]> ] [[-Head] <string[]> ] [[-Title] <string> ] [[-Body] <string[]> ] [-InputObject <psobject> ] [-As <string> ] [-Css Uri <uri> ] [-PostContent <string[]> ] [-PreContent <string[]> ] [<CommonParameters>]

ConvertTo-Html [[-Property] <Object[]> ] [-InputObject <psobject> ] [-As <string> ] [-Fragment] [-PostContent <string[]> ] [-PreContent <string[]> ] [<CommonParameters>]

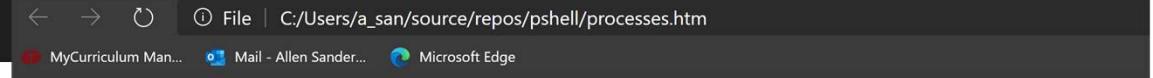
PS C:\Users\ a_san\source\repos\pshell> █
```



# ConvertTo-Html



```
PS C:\Users\{a_san}\source\repos\pshell>
PS C:\Users\{a_san}\source\repos\pshell> Get-Process | Select-Object -Property Name,Company,Description |
>> ConvertTo-Html -Property Name,Company,Description > '.\processes.htm'
PS C:\Users\{a_san}\source\repos\pshell>
```



Name	Company	Description
AlienwareOn-ScreenDisplay	Alienware Corp.	Alienware On-Screen Display
Amazon Music Helper	Amazon.com Services LLC	Amazon Music Helper
ApplicationFrameHost	Microsoft Corporation	Application Frame Host
AppVShNotify	Microsoft Corporation	Microsoft Application Virtualization Client Shell Notifier
AppVShNotify	Microsoft Corporation	Microsoft Application Virtualization Client Shell Notifier
audiogd	Microsoft Corporation	Windows Audio Device Graph Isolation
AWCC	Dell Technologies	AWCC
AWCC.Background.Server	Dell Technologies	AWCC.Background.Server
AWCC.Service	Dell Technologies	AWCC.Service
Calculator	Microsoft Corporation	Calculator.exe
chrome	Google LLC	Google Chrome
chrome	Google LLC	Google Chrome
chrome	Google LLC	Google Chrome
chrome	Google LLC	Google Chrome
chrome	Google LLC	Google Chrome
chrome	Google LLC	Google Chrome
chrome	Google LLC	Google Chrome
chrome	Google LLC	Google Chrome
chrome	Google LLC	Google Chrome



# ConvertFrom-Json



- Can be used to convert raw JSON as input into PowerShell objects
- Data will be converted to an array of custom objects
- Individual records and properties are available using [ ] and . notation

```
{-- people.json X
-- people.json > ...
1  [
2    {
3      "FirstName": "Joe",
4      "LastName": "Smith",
5      "Age": 42
6    },
7    {
8      "FirstName": "Melissa",
9      "LastName": "Testing",
10     "Age": 39
11   }
12 ]
```

```
PS C:\Users\A_San\source\repos\pshell> $jsonPeople = Get-Content -Path .\people.json -Raw | ConvertFrom-Json
PS C:\Users\A_San\source\repos\pshell> $jsonPeople
FirstName LastName Age
-----
Joe      Smith    42
Melissa Testing  39

PS C:\Users\A_San\source\repos\pshell> $jsonPeople[1].Age
39
PS C:\Users\A_San\source\repos\pshell> █
```



# ConvertTo-Json



- Can be used to pipe output from other commands to JSON
- Enables translation of one format (e.g., CSV) to JSON
- Provides a `-Compress` parameter for minimizing resulting text

```
processes.csv x
processes.csv
1 "Name", "Company", "Description"
2 "AlienwareOn-ScreenDisplay", "Alienware Corp.", "Alienware On-Screen Display"
3 "Amazon Music Helper", "Amazon.com Services LLC", "Amazon Music Helper"
4 "ApplicationFrameHost", "Microsoft Corporation", "Application Frame Host"
5 "AppVShNotify", "Microsoft Corporation", "Microsoft Application Virtualization Client Shell Notifier"
6 "AppVShNotify", "Microsoft Corporation", "Microsoft Application Virtualization Client Shell Notifier"
7 "audiodg", "Microsoft Corporation", "Windows Audio Device Graph Isolation "
8 "AWCC", "Dell Technologies", "AWCC"
9 "AWCC.Background.Server", "Dell Technologies", "AWCC.Background.Server"
10 "AWCC.Service", "Dell Technologies", "AWCC.Service"
11 "Calculator", "Microsoft Corporation", "Calculator.exe"
12 "chrome", "Google LLC", "Google Chrome"
13 "chrome", "Google LLC", "Google Chrome"
```

```
processes.json x
processes.json > ...
1 [{"Name": "AlienwareOn-ScreenDisplay", "Company": "Alienware Corp.", "Description": "A", "Description": "Alienware On-Screen Display"}, {"Name": "ApplicationFrameHost", "Company": "Microsoft Corporation", "Description": "Microsoft Application Virtualization Client Shell Notifier"}, {"Name": "audiodg", "Company": "Microsoft Corporation", "Description": "Windows Audio Device Graph Isolation"}, {"Name": "AWCC", "Company": "Dell Technologies", "Description": "AWCC"}, {"Name": "AWCC.Background.Server", "Company": "Dell Technologies", "Description": "AWCC.Background.Server"}, {"Name": "AWCC.Service", "Company": "Dell Technologies", "Description": "AWCC.Service"}, {"Name": "Calculator", "Company": "Microsoft Corporation", "Description": "Calculator"}, {"Name": "chrome", "Company": "Google LLC", "Description": "Google Chrome"}, {"Name": "Code", "Company": "Microsoft Corporation", "Description": "Code"}, {"Name": "Code", "Company": "Microsoft Corporation", "Description": "Visual Studio Code"}]
```

```
PS C:\Users\user\source\repos\pshell>
PS C:\Users\user\source\repos\pshell> Import-Csv .\processes.csv | ConvertTo-Json -Compress | Set-Content -Path .\processes.json
PS C:\Users\user\source\repos\pshell>
```



# Conditional Logic





# Comparison Operators



Operator	Description
-eq	Compares for equality
-ne	Compares for inequality
-gt	Checks for greater than
-ge	Checks for greater than or equal
-lt	Checks for less than
-le	Checks for less than or equal
-contains	Checks for contains

Case-sensitive versions also exist (prefixed with a “c”) – e.g. -ceq, -cne, etc.



# Logical Operators



Operator	Description
-and	Logical AND – True when both are True
-or	Logical OR – True when either is True (or both)
-xor	Logic Exclusive OR – True when only one is True
-not	Logical NOT – negates the statement that follows
!	Same as -not



# if Statement



- Allows us to check a condition (or combination or conditions)
- Multiple conditions can be combined as part of a single check using logical operators
- If evaluates to true, statement(s) defined in the corresponding branch execute

```
if (condition) {  
    # code to run when True  
}
```



## else Statement



- Allows us to define code to execute when condition evaluates to False
- Alternate branch logic

```
if (condition) {  
    # code to run when True  
} else {  
    # code to run when False  
}
```



## elseif Statement



- Allows definition of branches for checking additional conditions
- Best practice is to try and keep number of branches to a reasonable size

```
if (condition 1) {  
    # code to run when condition 1 True  
} else if (condition 2) {  
    # code to run when condition 2 True  
} else {  
    # code to run when all False  
}
```



# switch Statement



- Potential substitute for if-elseif when testing a single value
- Checks for matching value in branches and executes branch if match found
- If none found, executes default branch (if defined)
- *break* is used to make the conditions mutually exclusive

```
switch (expression) {  
    expressionvalue 1 {  
        # Code  
        break  
    }  
    expressionvalue 2 {  
        # Code  
        break  
    }  
    default {  
        # Code for no match  
    }  
}
```



Looping





## foreach Loop



- Performs an action for every object in a collection (usually an array)
- Useful when you need to iterate over the entire set
- Provides 3 different formats



## foreach Statement



- Separate looping structure that can be used in our PowerShell code
- Uses a variable to hold a copy of the current item during iteration
- Logic defined in the loop has access to the variable for processing as part of current iteration

```
foreach ($var in $array) {  
    # Logic to be executed on each iteration  
    # Has read-only access to $var  
}
```



# ForEach-Object Cmdlet



- What we looked at earlier as part of the piping discussion

```
PS C:\Users\ a_san\source\repos\pshell> Get-Command -Name ForEach-Object -Syntax
ForEach-Object [-Process] <scriptblock[]> [-InputObject <psobject>] [-Begin <scriptblock>] [-End <scriptblock>] [-RemainingScripts <scriptblock[]>] [-WhatIf] [-Confirm] [<CommonParameters>]
ForEach-Object [-MemberName] <string> [-InputObject <psobject>] [-ArgumentList <Object[]>] [-WhatIf] [-Confirm] [<CommonParameters>]
PS C:\Users\ a_san\source\repos\pshell> |
```

```
PS C:\Users\ a_san\source\repos\pshell> $total = 0
PS C:\Users\ a_san\source\repos\pshell> 1..100 | ForEach-Object -Begin {Write-Host 'Gathering sum'} -Process {$total += $_} -End {Write-Host 'Done with sum'}
Gathering sum
Done with sum
PS C:\Users\ a_san\source\repos\pshell> $total
5050
PS C:\Users\ a_san\source\repos\pshell> |
```

```
PS C:\Users\ a_san\source\repos\pshell> 13.34342, 19.234234 | ForEach-Object -MemberName ToString -ArgumentList 'C'
$13.34
$19.23
PS C:\Users\ a_san\source\repos\pshell>
```



## foreach() Method



- Exists on all arrays in PowerShell
- Provides same iterative functionality as other two
- Called as method on an array variable, executing the provided ScriptBlock parameter for each iteration
- Considerably faster than the other two

`$array.ForEach({ <script to be executed> })`



## for Loop



- Supports iteration for a predetermined number of occurrences
- Uses a numeric counter to track iterations against a target amount
- Requires an action to move the counter forward
- Also, helpful if access to current numeric position or access to more than one element is desired

```
for ($counter = 0; $counter -lt <#>; $counter++) {  
    # Access to $counter possible here  
    # Access to $array[$counter] or $array[$counter + 1], etc.  
}
```



## while Loop



- Executes a block of code while a condition is True
- Useful when the intended number of iterations is not known
- Requires a statement in the loop to move the loop to completion

```
$counter = 0
while ($counter -lt <#>) {
    # Access to $counter possible here
    $counter++ # For example
}
```



## while Loop – Alternate Version



```
$counter = 0
while ($true) {
    $counter++
    if ($counter -eq 10) {
        break
    }
}
```



## do/while Loop



- Executes a block of code while a condition is True
- Difference between this and *while* loop is that associated code block will be executed at least once with *do/while*

```
do {  
    # Code block  
    # Requires statement to move to completion  
} while (condition)
```



## do/until Loop



- Executes a block of code until a condition is True
- Inverse to *do/while*

```
do {  
    # Code block  
    # Requires statement to move to completion  
} until (condition) # Like while NOT true
```



# PowerShell Security





# Get-ExecutionPolicy



- Execution policy is implemented in PowerShell to protect against malicious scripting
- *Get-ExecutionPolicy* shows currently applied execution policy (if executed without params)
- If executed with the *-List* param, will show applied execution policies across various scopes



# Get-ExecutionPolicy



- Key options include:
  - *AllSigned* – All scripts (.ps1 files) must be signed by a trusted publisher (including scripts written on local computer); most stringent
  - *RemoteSigned* – All remotely downloaded scripts must be signed by a trusted publisher; scripts written on local computer can be executed without digital signature
  - *Restricted* – Prevents load of config files and script (.ps1 file) runs
  - *Unrestricted* – Loads config files and executes scripts; if unsigned downloaded from the Internet, will prompt for execution



# Set-ExecutionPolicy



- Enables update of execution policy (as required) for running scripts on a machine
- Includes a couple of key parameters:
  - *-ExecutionPolicy* for specifying target value (AllSigned, RemoteSigned, etc.)
  - *-Scope* for specifying scope of policy application



# Set-ExecutionPolicy



- Options include:
  - *MachinePolicy* – set by Group Policy for all users of the computer
  - *UserPolicy* – set by Group Policy for the current user of the computer
  - *Process* – affects only current PowerShell session
  - *CurrentUser* – affects only current user
  - *LocalMachine* – affects all users of the computer



# Digital-Signed Script



- *Set-AuthenticodeSignature* used to sign the script using trusted cert

```
# SIG # Begin signature block
# MIIEwYJKoZIhvcNAQcCoIEJDCCBCACAQExCzAJBgUrDgMCGgUAMGkGCisGAQQB
# gjcCAQSgWzBZMDQGCisGAQQBgjccAR4wJgIDAQABBAfzDtgWUsITrck0sYpfvNR
# AgEAAgEAAgEAAgEAMCEwCQYFKw4DAh0FAAQU6vQAn5sf2qIxQqwWUDwTZnJj
...snip...
# m5ugggI9MIICOTCCAaagAwIBAgIQyLeyGZcGA4ZOGqK7VF45GDAJBgUrDgMCHQUA
# Dxoj+2keS9sRR6XP1/ASs68LeF8o9cM=
# SIG # End signature block
```



## Get-Credential



- Many cmdlets allow specification of an alternate set of credentials at time of execution
- Useful if executing user or context is anonymous or not authorized to the target resources
- To use, look for a –Credential parameter on the command

```
PS C:\Users\user\source\repos\pshell> Get-Command -Name Get-Credential -Syntax

Get-Credential [-Credential] <pscredential> [<CommonParameters>]

Get-Credential [[-UserName] <string>] -Message <string> [<CommonParameters>]

PS C:\Users\user\source\repos\pshell> █
```



## Get-Credential



- There are mechanisms for building the credential in code as well

```
PS C:\Users\{user}\source\repos\pshell> New-PSSession -ComputerName WIN-KL9JE8KGF5I -Credential (Get-Credential)

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
User: administrator
Password for user administrator: *****

Id Name          ComputerName   ComputerType    State      ConfigurationName   Availability
-- --          -----          -----          -----      -----          -----
 1 WinRM1        WIN-KL9JE8KGF5I  RemoteMachine  Opened     Microsoft.PowerShell  Available

PS C:\Users\{user}\source\repos\pshell> Enter-PSSession -Id 1

[WIN-KL9JE8KGF5I]: PS C:\Users\Administrator\Documents>
```



# PowerShell Modules





## Finding Modules



- *Get-Module -ListAvailable* returns the list of modules that are installed and can be imported
- Modules can be installed in multiple locations
- Modules are made up of a .psm1 file (module code) and .psd1 (for manifest)
- To be useful, a module needs to have functions (which should be related)



# Finding Modules



- Possible installed module locations include:
  - System modules – C:\Windows\System32\WindowsPowerShell\1.0\Modules
  - All Users modules – C:\Program Files\WindowsPowerShell\Modules
  - Current User modules –C:\Users<userid>\Documents\WindowsPowerShell\Modules



# Importing Modules



- Installed modules must be imported to be used
- *Import-Module* can be used to import an installed module by name
- PowerShell supports auto-import – if module is installed and command in module is used, module will be automatically imported into session
- Can use *-Force* parameter with *Import-Module* to unload and reimport



# Installing Modules



- The PowerShell Gallery (<https://www.powershellgallery.com/>) is a good source for modules
- The PowerShellGet module provides commands for integrating with the Gallery



# Installing Modules



```
PS C:\Users\{a_san}\source\repos\pshell> Get-Command -Module PowerShellGet

CommandType      Name                                               Version   Source
----          ----
Function        Find-Command                                     1.0.0.1   PowerShellGet
Function        Find-DscResource                                1.0.0.1   PowerShellGet
Function        Find-Module                                     1.0.0.1   PowerShellGet
Function        Find-RoleCapability                            1.0.0.1   PowerShellGet
Function        Find-Script                                     1.0.0.1   PowerShellGet
Function        Get-InstalledModule                            1.0.0.1   PowerShellGet
Function        Get-InstalledScript                           1.0.0.1   PowerShellGet
Function        Get-PSRepository                             1.0.0.1   PowerShellGet
Function        Install-Module                                1.0.0.1   PowerShellGet
Function        Install-Script                                1.0.0.1   PowerShellGet
Function        New-ScriptFileInfo                           1.0.0.1   PowerShellGet
Function        Publish-Module                               1.0.0.1   PowerShellGet
Function        Publish-Script                                1.0.0.1   PowerShellGet
Function        Register-PSRepository                         1.0.0.1   PowerShellGet
Function        Save-Module                                    1.0.0.1   PowerShellGet
Function        Save-Script                                    1.0.0.1   PowerShellGet
Function        Set-PSRepository                            1.0.0.1   PowerShellGet
Function        Test-ScriptFileInfo                          1.0.0.1   PowerShellGet
Function        Uninstall-Module                            1.0.0.1   PowerShellGet
Function        Uninstall-Script                            1.0.0.1   PowerShellGet
Function        Unregister-PSRepository                      1.0.0.1   PowerShellGet
Function        Update-Module                                1.0.0.1   PowerShellGet
Function        Update-ModuleManifest                        1.0.0.1   PowerShellGet
Function        Update-Script                                1.0.0.1   PowerShellGet
Function        Update-ScriptFileInfo                      1.0.0.1   PowerShellGet

PS C:\Users\{a_san}\source\repos\pshell>
```



# Installing Modules



- To install a module from the Gallery, you can use *Find-Module* to search by name
- Results of *Find-Module* can be piped to *Install-Module* to install
- *-AllowClobber* can be used to override existing commands with same name

```
PS C:\Users\user\source\repos\pshell> Find-Module -Name VMware.PowerCLI | Install-Module

Untrusted repository
You are installing the modules from an untrusted repository. If you trust this repository, change its InstallationPolicy value by running the Set-PSRepository cmdlet. Are you sure you want to proceed? [Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "No"): a
[Y] Yes [A] Yes to All [N] No [L] No to All [S] Suspend [?] Help (default is "No"): a
PackageManagement\Install-Package : The following commands are already available on this system: 'Export-VM,Get-VM,Get-VMHost,Move-VM,New-VM,Remove-VM,Restart-VM,Set-VM,Set-VMHost,Start-VM,Stop-VM'. 'VMware.VimAutomation.Core' may override the existing commands. If you still want to install this module 'VMware.VimAutomation.Core', use -AllowClobber parameter.
At C:\Program Files\WindowsPowerShell\Modules\PowerShellGet\1.0.0.1\PSModule.psm1:1912 char:34
+ ...             $null = PackageManagement\Install-Package @PSBoundParameters
+ ~~~~~
    + CategoryInfo          : InvalidOperation: (Microsoft.Power...InstallPackage:InstallPackage) [Install-Package], Exception
    + FullyQualifiedErrorMessage : CommandAlreadyAvailable,Validate-ModuleCommandAlreadyAvailable,Microsoft.PowerShell.PackageManagement.Cmdlets.InstallPackage

PS C:\Users\user\source\repos\pshell>
```



# Uninstalling Modules



- Uninstall-Module can be used to remove a module from disk by name

```
PS C:\Users\{user}\source\repos\pshell> Uninstall-Module -Name VMware.PowerCLI
```



## Custom Modules



- PowerShell supports creation of your own modules
- Can be shared within organization or to the Gallery for public use
- Requires three pieces – a folder, a .psm1 file (module code) and a .psd1 file (module manifest)



## Custom Modules



- Files must be stored in folder with same name as module
- If placed in one of the common locations (System, All Users or Current User), PowerShell will import automatically
- Manifest file has a specific format



## Custom Modules



- *New-ModuleManifest* provides parameters for things like path, author, etc. and builds file in correct format
- To make functions defined in module available, need to set the *FunctionsToExport* key in manifest



# Functions





# What are Functions?



- Functions are a named piece of reusable code that performs a specific task
- Can accept parameters with validation (if required)
- Provide options for enabling pipeline support
- More advanced versions support parameter configuration to make the function look more like a cmdlet (-*WhatIf*, -*Confirm*, etc.)



## Functions vs. Cmdlets



- Sounds like cmdlets
- Difference lies in how created
- Cmdlets are written in another language, compiled and made available in PS
- Functions are written in PowerShell directly



## Defining Functions



- Simplest form includes key word *function*, a name and ScriptBlock
- Can be defined inline in the shell or (more likely) in a .ps1 script or module
- Once defined can be executed using its name



# Function Naming Convention



- By convention, PowerShell function names should follow the *Verb-Noun* format
- You can use *Get-Verb* to see the list of recommended verbs
- Noun typically matches the entity/context in which defined
- The following will group verbs by specific categories for view

```
> Get-Verb | Format-Table Verb, Group -GroupBy Group -AutoSize
```



# Function Parameters



- Adding parameters to functions allows variability
- Instead of doing the same thing every time called, parameters allow us to change based on runtime conditions
- Parameters are added to our functions using `param()`
- Supports multiple properties for defining/constraining the parameter's value



# Function Parameters



- Simple example

```
function Get-Area {  
    [CmdletBinding()]  
    param(  
        [Parameter()]  
        [decimal] $radius  
    )  
  
    $area = [Math]::PI * $radius * $radius  
    Write-Host "$radius has area $area"  
}
```



## Making Parameters Mandatory



- Add *Mandatory* keyword to parameter definition
- Attempts to call with out parameter will prompt user

```
function Get-Area {  
    [CmdletBinding( )]  
    param(  
        [Parameter(Mandatory)]  
        [decimal] $radius  
    )  
  
    $area = [Math]::PI * $radius * $radius  
    Write-Host "$radius has area $area"  
}
```



# Setting Parameter Defaults



- Accomplished by assigning value at parameter declaration
- Alternative to using *Mandatory*
- Still allows call with a specific value

```
function Get-Area {  
    [CmdletBinding()]  
    param(  
        [Parameter()]  
        [decimal] $radius = 1.5  
    )  
  
    $area = [Math]::PI * $radius * $radius  
    Write-Host "$radius has area $area"  
}
```



## Adding Parameter Validation



- Various validation attributes exist to validate parameter values
- If value provided on call violates, an error is thrown
- See <https://docs.microsoft.com/en-us/powershell/scripting/developer/cmdlet/validating-parameter-input>

```
function Get-Area {  
    [CmdletBinding( )]  
    param(  
        [Parameter( )]  
        [ValidateRange(0.0, 5.0)]  
        [decimal] $radius = 1.5  
    )  
  
    $area = [Math]::PI * $radius * $radius  
    Write-Host "$radius has area $area"  
}
```



## Adding Pipeline Support



- Adding *ValueFromPipeline* or *ValueFromPipelineByPropertyName* to parameter adds pipeline support
- *ValueFromPipeline* will match pipeline input by type
- *ValueFromPipelineByPropertyName* will match pipeline input by name
- Probably most useful for functions exported from a module



## Begin, End and Process



- As written so far, our function will only run once – even if multiple values are piped to it
- Logic to be executed once for each input object, should be wrapped in a *process* block
- A *begin* block can be used to execute logic before any piped elements are run
- An *end* block can be used to execute logic after all piped elements are run



*DEMO – Making available in module*



# Running Scripts Remotely





# Enabling Remoting



- Requires that a WinRM service is running with appropriate firewall ports open and access rights configured
- As of Server 2012, should be enabled by default
- You can run *Test-WSMan* to check setup

```
wsmid      : http://schemas.dmtf.org/wbem/wsman/identity/1/wsmanidentity.xsd
ProtocolVersion : http://schemas.dmtf.org/wbem/wsman/1/wsman.xsd
ProductVendor  : Microsoft Corporation
ProductVersion : OS: 0.0.0 SP: 0.0 Stack: 3.0
```

- If not setup, *Enable-PSRemoting* can be executed from an elevated console session to configure
- Once enabled, *Invoke-Command* can be used to execute remote commands



## Invoke-Command



- Enables connection to a remote system and execution of a ScriptBlock or script file

```
Invoke-Command -ScriptBlock { <script> } -ComputerName <computer name>  
Invoke-Command -FilePath <scriptfile> -ComputerName <computer name>
```

- For script file, script contents (not the script itself) executed on remote system and results returned to local
- *-ArgumentList* can be used to pass local values to remote system – referenced in script using \$args automatic variable (array)



## Invoke-Command



- *Invoke-Command* is used this way with ad hoc commands
- Connects, runs command and disconnects
- Behind-the-scenes, something called a session is used for the connection
- However, with ad hoc, the session is created, command executed, and session gets torn down



# Sessions



- PowerShell remoting leverages a session
- We can create a session remotely which opens a local session on the remote system
- We then connect to that session for routing remote commands



## Sessions



- Instead of ad hoc, you can create sessions, and then connect/disconnect
- Session maintains state until removed
- More efficient than ad hoc when multiple statements or scripts to be run



## New-PSSession



- Used to create a semi-permanent session on remote computer
- Uses name of machine for connection
- Requires local admin or member of Remote Management Users group on remote machine
- Also, requires that local and remote systems be in same AD domain



## New-PSSession



- If not in same AD domain, can add remote system to TrustedHosts collection
- Use Get-Item WSMan:\localhost\Client\TrustedHosts to check
- Use Set-Item WSMan:\localhost\Client\TrustedHosts –Value <hostname> to add



## New-PSSession



- Can use the `-Credential` parameter and `Get-Credential` cmdlet (or build credential in code) for access
- Adds session to set of available sessions configured on local system
- Can be referenced instead of `-ComputerName` to connect and execute



## Invoke-Command - Redux



- Enables connection to a remote system and execution of a ScriptBlock or script file

```
Invoke-Command -ScriptBlock { <script> } -Session <session ref>  
Invoke-Command -FilePath <scriptfile> -Session <session ref>
```

- Runs faster than ad hoc – no need to create and tear down for every command
- Also access to additional functionality (e.g., variables set in remote session persist)



## Interactive Sessions



- Enter-PSSession can be used to connect to a remote session interactively
- Command-line changes to reflect remote session
- Can be a good alternative to Remote Desktop Protocol (RDP)



## Disconnect & Reconnect – Session



- *Disconnect-PSSession* can be used to disconnect from a remote session (for potential reconnection later)
- *Connect-PSSession* can be used to reconnect
- NOTE: Disconnected sessions only work if local and remote machine have same PowerShell version
- Can use `$PSVersionTable` on both systems to check



## Removing Sessions



- *Remove-PSSession* can be used to delete a session
- Tears down session at remote computer, and, if exists, removes local references



## Remote Management



*DEMO*



## Scheduled Jobs





## What Are They?



- PowerShell scripts that can be triggered on a schedule or in response to events
- Run asynchronously in the background
- Used for regularly repeated tasks – maintenance, configuration, etc.



## What Are They?



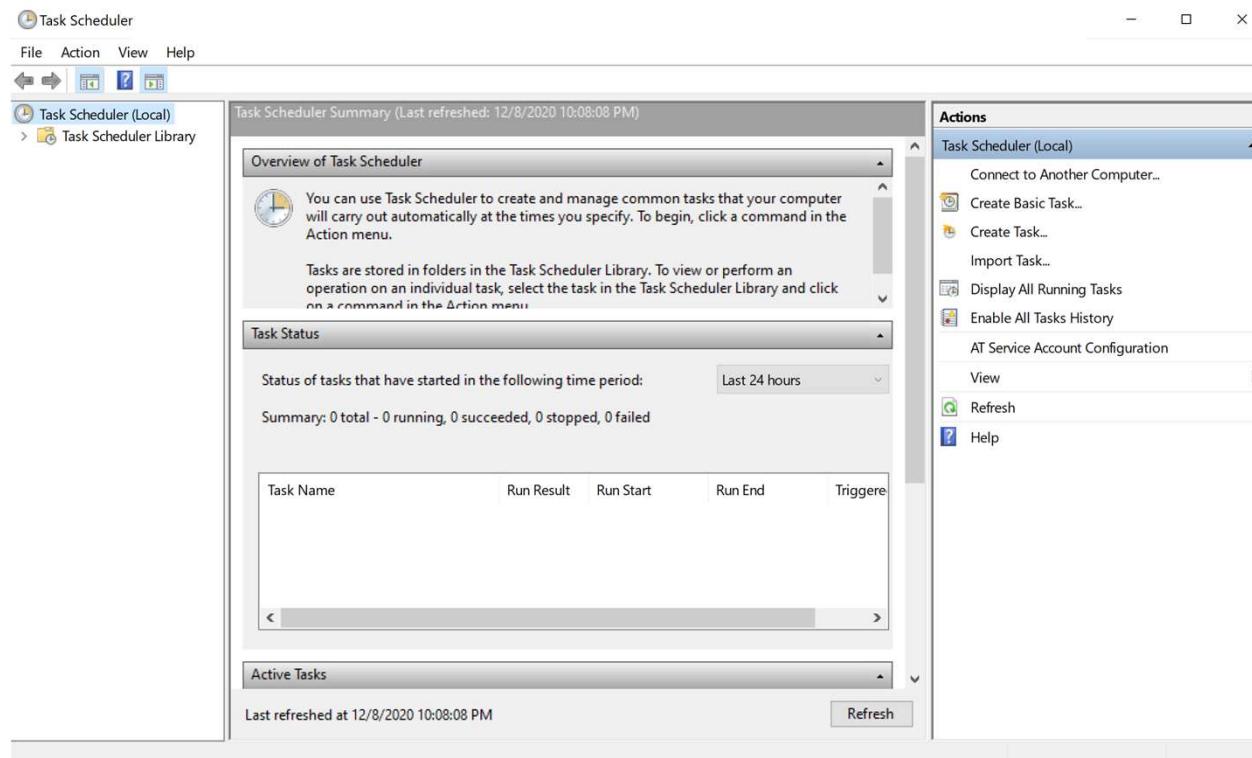
- Can be viewed and managed in Task Scheduler
- Jobs are saved to disk – can be enabled, disabled, executed
- Results of instance runs are saved to disk as well, including a log of job output



# Mirrors Task Scheduler

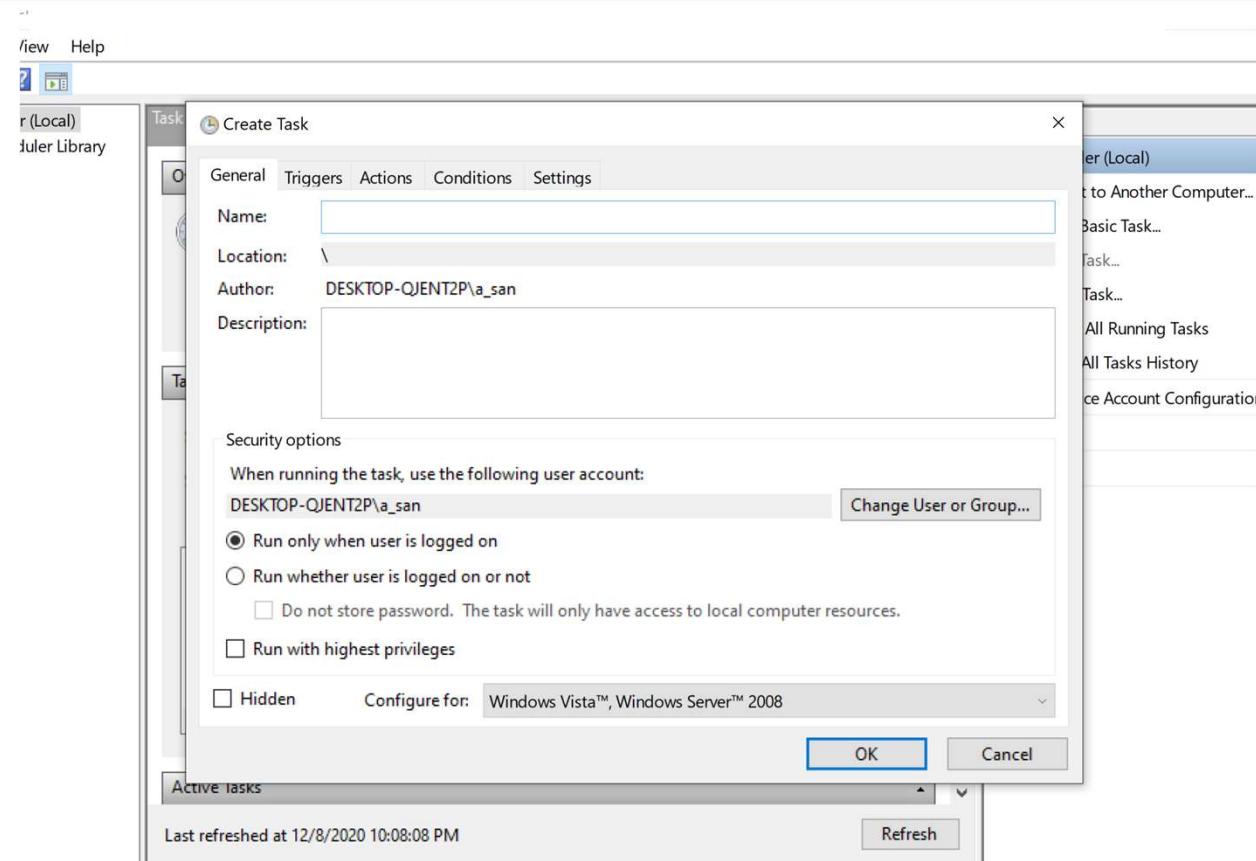


- But defined in PowerShell



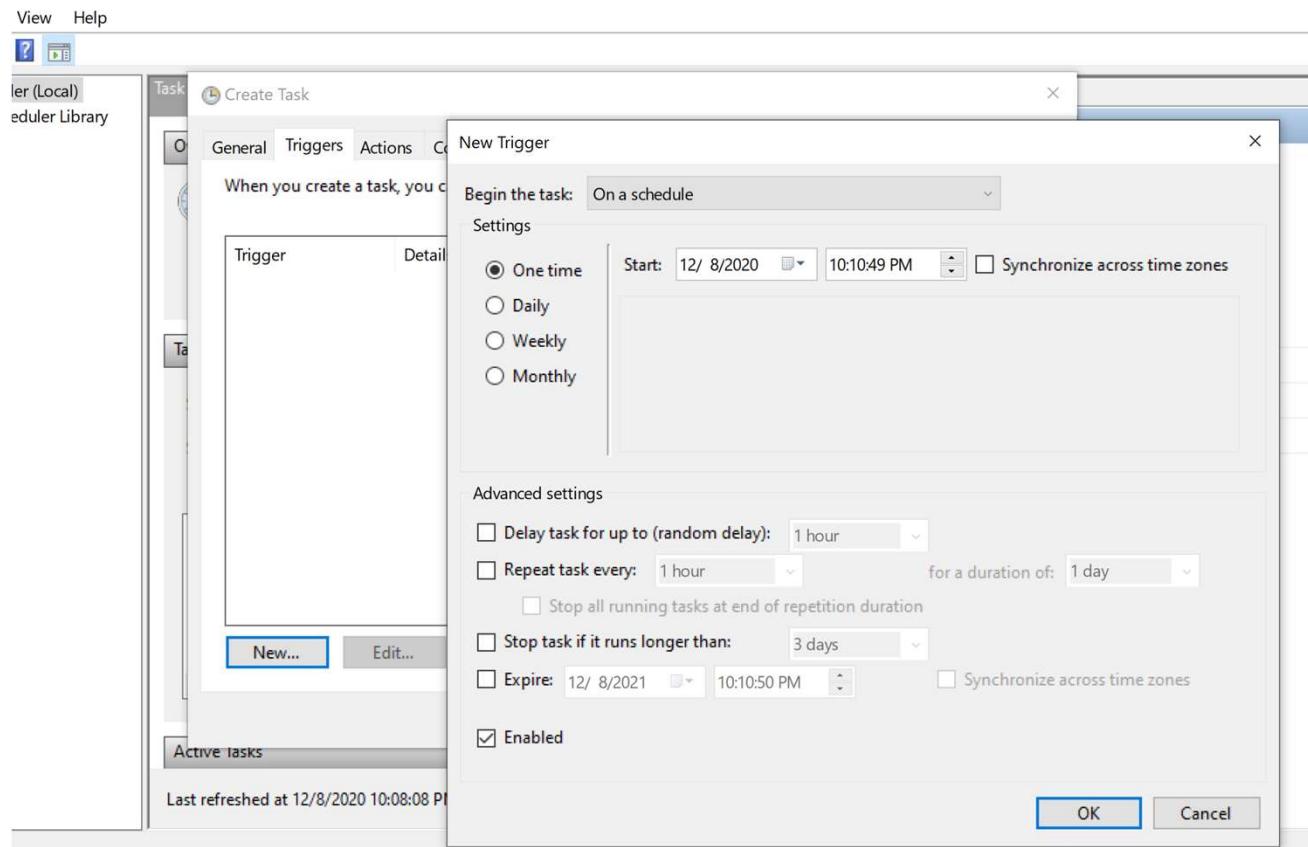


# Mirrors Task Scheduler





# Mirrors Task Scheduler





## Getting Info About Scheduled Jobs



- *Get-ScheduledJob* retrieves list of scheduled jobs on system
- *Get-JobTrigger* retrieves info about defined trigger for a job
- *Get-Job* retrieves info on all instances of scheduled job that have previously run



## Getting Info About Scheduled Jobs



- *Receive-Job* gets the results of the most recent run instance
- *Receive-Job* includes a `-Keep` parameter to leave in place as most recent
- Even without `-Keep`, job results are saved on disk until you delete or maximum # exceeded



# Creating New Scheduled Jobs



- *Register-ScheduledJob* used to define and register job for execution
- *-Trigger* parameter used to set job trigger
- *-ScheduledJobOption* used to define job options
- Job can be defined using *-ScriptBlock* or *-FilePath* (for script file)

```
PS C:\Users\{user}\source\repos\pshell> Get-Command -Name Register-ScheduledJob -Syntax

Register-ScheduledJob [-Name] <string> [-ScriptBlock] <scriptblock> [-Trigger <ScheduledJobTrigger[]>] [-InitializationScript <scriptblock>] [-RunAs32] [-Credential <pscredential>] [-Authentication <AuthenticationMechanism>] [-ScheduledJobOption <ScheduledJobOptions>] [-ArgumentList <Object[]>] [-MaxResultCount <int>] [-RunNow] [-RunEvery <timespan>] [-WhatIf] [-Confirm] [<CommonParameters>]

Register-ScheduledJob [-Name] <string> [-FilePath] <string> [-Trigger <ScheduledJobTrigger[]>] [-InitializationScript <scriptblock>] [-RunAs32] [-Credential <pscredential>] [-Authentication <AuthenticationMechanism>] [-ScheduledJobOption <ScheduledJobOptions>] [-ArgumentList <Object[]>] [-MaxResultCount <int>] [-RunNow] [-RunEvery <timespan>] [-WhatIf] [-Confirm] [<CommonParameters>]
```



# Defining Trigger



- *New-JobTrigger* used to define trigger
- Multiple versions available to support different types of schedules
- Versions also available for trigger on startup or logon

```
PS C:\Users\ a_san\source\repos\pshell> Get-Command -Name New-JobTrigger -Syntax

New-JobTrigger -Once -At <datetime> [-RandomDelay <timespan>] [-RepetitionInterval <timespan>] [-Repetitio
nDuration <timespan>] [-RepeatIndefinitely] [<CommonParameters>]

New-JobTrigger -Daily -At <datetime> [-DaysInterval <int>] [-RandomDelay <timespan>] [<CommonParameters>]

New-JobTrigger -Weekly -At <datetime> -DaysOfWeek <DayOfWeek[]> [-WeeksInterval <int>] [-RandomDelay <time
span>] [<CommonParameters>]

New-JobTrigger -AtStartup [-RandomDelay <timespan>] [<CommonParameters>]

New-JobTrigger -AtLogOn [-RandomDelay <timespan>] [-User <string>] [<CommonParameters>]

PS C:\Users\ a_san\source\repos\pshell> █
```



# PowerShell Use Cases





# Office 365 User Management



High-level walkthrough:

<https://docs.microsoft.com/en-us/microsoft-365/enterprise/manage-user-accounts-and-licenses-with-microsoft-365-powershell?view=o365-worldwide>



# Microsoft Teams



High-level walkthrough:

<https://docs.microsoft.com/en-us/MicrosoftTeams/teams-powershell-overview>

<https://github.com/cunninghamp/Install-MicrosoftTeams>

<https://docs.microsoft.com/en-us/powershell/module/teams/new-team?view=teams-ps>

<https://docs.microsoft.com/en-us/powershell/module/teams/new-teamchannel?view=teams-ps>



## Security & Compliance



High-level walkthrough:

<https://docs.microsoft.com/en-us/powershell/exchange/connect-to-scc-powershell?view=exchange-ps>



# Microsoft Graph API



High-level walkthrough:

<https://docs.microsoft.com/en-us/graph/overview>

<https://docs.microsoft.com/en-us/graph/powershell/get-started>



## Azure Automation



High-level walkthrough:

<https://docs.microsoft.com/en-us/azure/automation/automation-intro>

<https://docs.microsoft.com/en-us/azure/automation/automation-runbook-types>

*THANK YOU*



