PR

PowerShell for Responders

Student Guide

DC3 Cyber Training Academy

The Academy is accredited by the Commission of the Council on Occupational Education (COE).

COE is a national accrediting body dedicated to ensuring quality and integrity in career and technical education.

[v.1901]

CONTENTS

COURSE INTRODUCTION

After completing the PowerShell for Responders (PR) course, students will be able to design and employ scripts using the fundamentals of PowerShell scripting language.

This course teaches the fundamentals of the PowerShell scripting language. Students learn basic syntax and the script structure, including functions, variables, statements and expressions. Using PowerShell, students write programs using standard language infrastructure for the purpose of performing live forensics and/or system state modification.

LEARNING OBJECTIVES *After completing this course, students will be able to:*
Examine the capabilities of command line administration
Express command line and PowerShell scripting concepts
Execute commands to accomplish administrative tasks
Design scripts using branches, loops and functions
Describe the basics of the Common Information Model (CIM)
Employ WMIC in accessing WMI objects
Explain the process of risk analysis

MODULE 1

Introduction to Command Line Administration

This module will introduce students to basic Command Line functionality. It will also prepare students to use the CMD and PowerShell's Integrated Scripting Environment throughout the course. Understanding the key differences between the two will create a solid foundation for responders in the scripting environment.

OBJECTIVES *After completing this module, students will be able to:*
Duplicate the command line's capability to interact with its environment
Verify command (CMD) capabilities
Implement CMD commands
Compare and contrast the capabilities of PowerShell and CMD
Express the use of PowerShell's Integrated Scripting Environment (ISE)

Lesson 1

Command Line Introduction

This lesson introduces Command Line basics.

Command Line Basics

The Windows Command Prompt (CMD) is similar to shells in *nix. It exists as its own entity within the Windows system and executes predefined commands. It is similar to the File Explorer, but CMD also includes administrative commands. When the user execute commands with CMD, they are applied inside the current directory.

CMD has a wide range of functions. It can retrieve system and network data, modify system attributes, and modify the system's state. It is also able to execute multiple commands inside a batch file (or "script").

Lesson 2

Command Line Capabilities

This lesson will discuss the command line's capabilities. Also, it will go into detail regarding its use of processes.

Using Command Line Functions

In terms of real-time functionality, CMD can:

- Create and terminate processes
- Start and stop services
- Run basic scripts to automate basic administrative tasks
- Access Windows Management Instrumentation, or WMI (more on this later)
- Shut down/reboot system

However, CMD is not able to:

- Interchange content with other commands
- Display select content precisely per user specifications
- Run simple scripts that retrieve complex, explicit data
- Receive all parameters that are input by specification rather than order
- Natively run networking commands; e.g., HTTP, SSH
- Standardize options across commands, where applicable
- Retrieve a full list of available commands
- Create new commands without creating external programs

Lesson 3

Command Line Commands

This lesson discusses the commands that a user would employ in CMD.

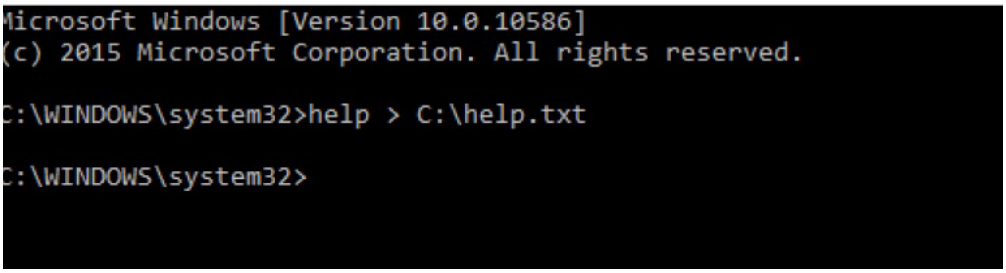
Basic Command Background

The command prompt (cmd.exe) is the interface for the command line. It is an executable that is embedded in most Windows operating systems and it can be accessed by typing "cmd" in the search bar of the start menu.

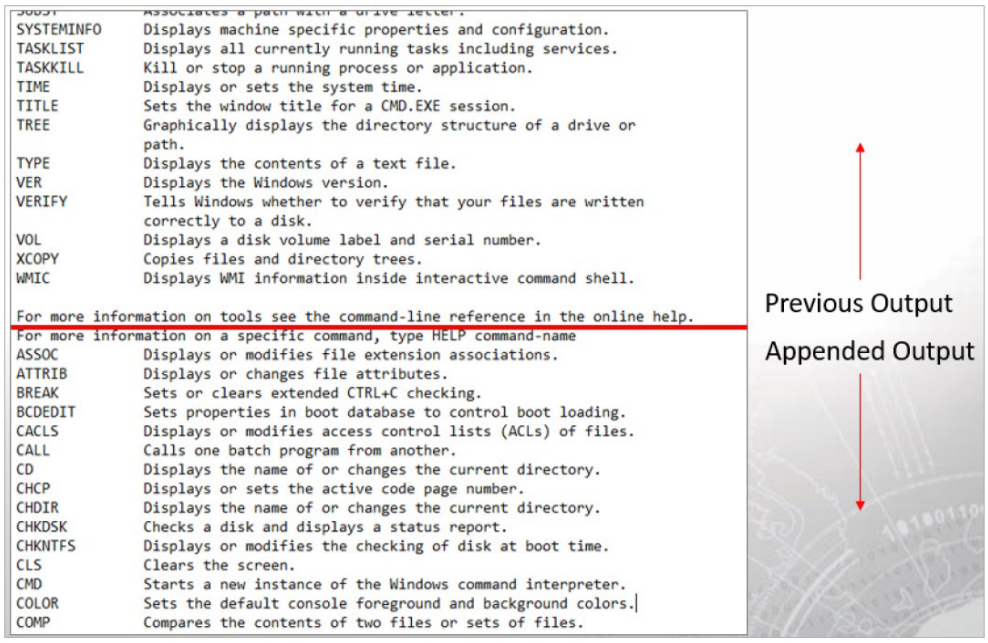
It is useful to know that all CMD commands are case-insensitive and entering "HELP" will produce the same list of command as "help." Variables also can be used in CMD by surrounding the variable name with percentage signs ("%"). For example, Windows 7 has a system variable named %SYSTEMROOT% that contains 'C:\Windows' by default.

Output Redirection

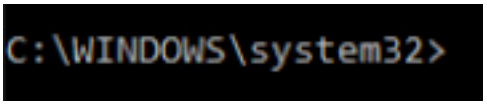
If users need to send input from a command to a specified file in CMD, they can use output redirection by placing a ">" after the command and entering the name and location of the target file. If you send data to a file that already exists via output redirection, you will overwrite the file. In the case below, a user redirects the output of the help command to a text file by entering the command `help > C:\help.txt:`



You may also add (or "append") the output from a command to the end of a file (instead of overwriting the entire file. Entering the command `help >> C:\help.txt` creates the screen below:



Users can run CMD commands within the current working directory. If you wish to work in a different directory, you will have to shift directories within CMD. The graphic below indicates that `C:\Windows\System32` is the current CMD directory, and all commands will be executed within this directory.



CMD can use two types of paths to reach a file: an absolute path and a relative path. Relative paths build upon the current working directory. If a filepath does not begin with a root directory, CMD considers it a relative filepath. For example, if you ask CMD to create a text file using a filepath of System32 and the working directory is C:\Windows, CMD will create a .txt file in `C:\Windows\System32`.

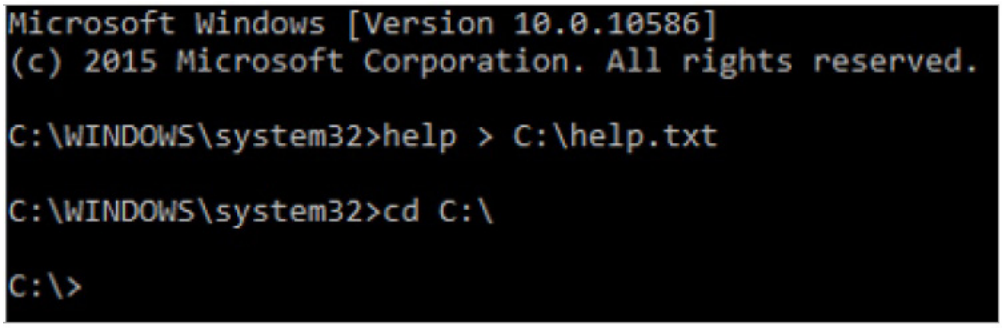
In contrast, absolute paths take you directly to the file's location. Whenever a filepath begins with a root path ("C:\"), CMD considers that path to be absolute. For example, `C:\Windows\System32` would refer to the System32 directory, regardless of the current working directory of CMD.

Typical Commands

These are typical commands that a user would use quite frequently within CMD. In CMD, the up and down arrow keys allow you to reuse previously entered commands. Furthermore, the Tab key allows you to complete partially typed folder/file names.

Commands

Change Directory	
Commands	**Description**
`CD <arguments>`	[C]hange [D]irectory – changes your current working directory.
Examples	
`cd C:\`	– changes the root directory of the C: drive
`cd ..`	– changes to the parent directory



Clear Screen	
Commands	**Description**
CLS	[CL]ear [S]creen – clears the screen.

Delete	
Commands	**Description**
`DEL <arguments>`	[DEL]ete – deletes one or more files.
Examples	
`del x.txt`	– deletes the file x.txt
`del C:\*.jpg`	– deletes all files ending in .jpg in the C:\ directory. Here, the asterisk ("*") is used as a wildcard character representing all .jpg files with filenames (i.e. all of them)

Unless there is an error, CMD will delete the specified files without returning a response.

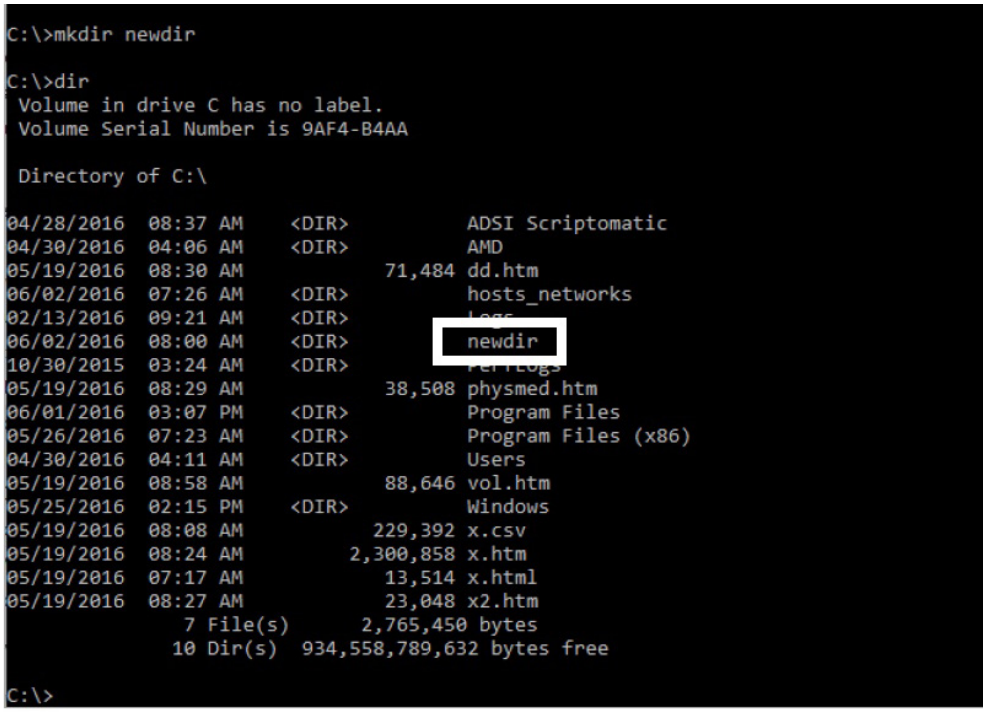
Directory Listing	
Commands	**Description**
`DIR <option args>`	[DIR]ectory listing – list directory contents

The directory listing command displays the current directory by default (`dir`). It can also display the contents of a user-specified directory (`dir c:\`).

```
C:\>del help.txt

C:\>dir
 Volume in drive C has no label.
 Volume Serial Number is 9AF4-B4AA

 Directory of C:\

04/28/2016  08:37 AM    <DIR>          ADSI Scriptomatic
04/30/2016  04:06 AM    <DIR>          AMD
05/19/2016  08:30 AM            71,484 dd.htm
06/02/2016  07:26 AM    <DIR>          hosts_networks
02/13/2016  09:21 AM    <DIR>          Logs
10/30/2015  03:24 AM    <DIR>          PerfLogs
05/19/2016  08:29 AM            38,508 physmed.htm
06/01/2016  03:07 PM    <DIR>          Program Files
05/26/2016  07:23 AM    <DIR>          Program Files (x86)
04/30/2016  04:11 AM    <DIR>          Users
05/19/2016  08:58 AM            88,646 vol.htm
05/25/2016  02:15 PM    <DIR>          Windows
05/19/2016  08:08 AM           229,392 x.csv
05/19/2016  08:24 AM         2,300,858 x.htm
05/19/2016  07:17 AM            13,514 x.html
05/19/2016  08:27 AM            23,048 x2.htm
               7 File(s)      2,765,450 bytes
               9 Dir(s)  934,558,789,632 bytes free

C:\>
```

Make Directory	
Commands	**Description**
`MKDIR <args>`	[M]a[K]e [DIR]ectory – create an empty directory
Examples	
`mkdir test`	– create the `test` directory in the current directory
`mkdir test\new\dir`	– create three nested directories `test\new\dir` in the current directory

The `mkdir` command is used to create an empty directory. It is possible to create a directory or nested directories within your current working directory.

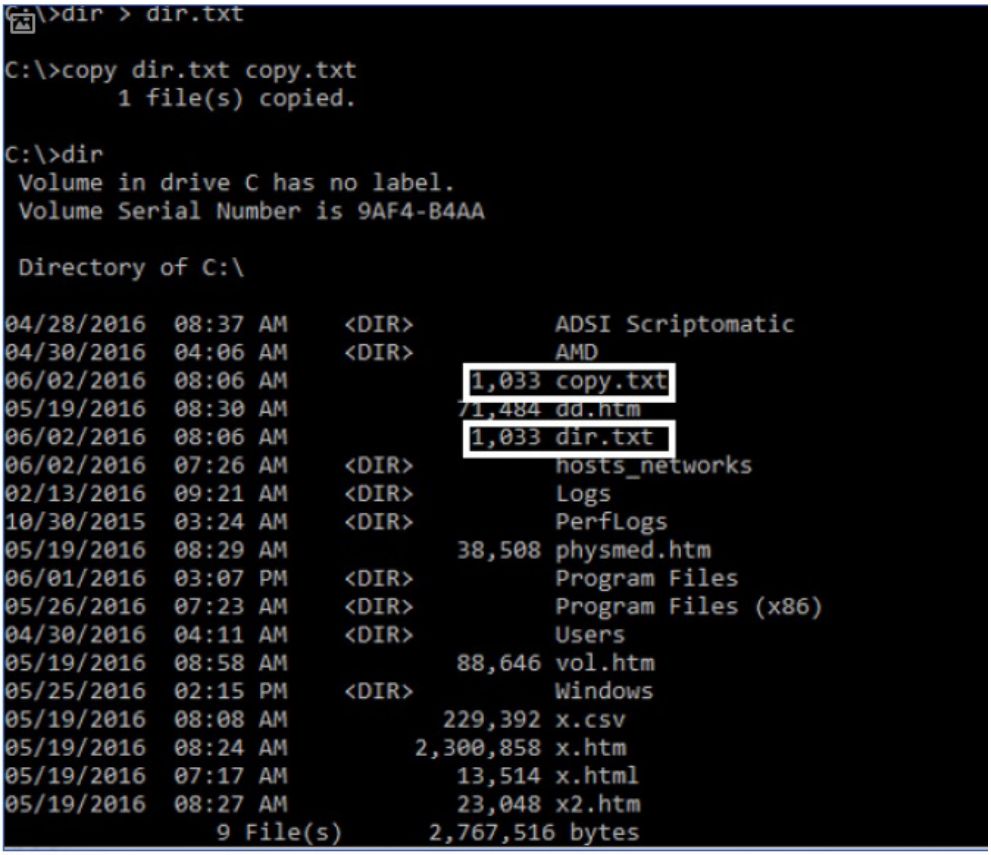


Remove Directory	
Commands	**Description**
`RMDIR <args>`	[R]e[M]ove [DIR]ectory – delete the specified directory
Examples	
`rmdir test`	– removes the test directory if test is empty
`rmdir /S tes`	– removes the test directory even if test is not empty

The `rmdir` is the opposite of the `mkdir` command, in that it will delete an entire directory. You can use arguments to specify which directory to remove, and under what conditions the directory may be removed.

```
          10 Dir(s)  934,558,789,632 bytes free

C:\>rmdir newdir

C:\>dir
 Volume in drive C has no label.
 Volume Serial Number is 9AF4-B4AA

 Directory of C:\

04/28/2016  08:37 AM    <DIR>          ADSI Scriptomatic
04/30/2016  04:06 AM    <DIR>          AMD
05/19/2016  08:30 AM            71,484 dd.htm
06/02/2016  07:26 AM    <DIR>          hosts_networks
02/13/2016  09:21 AM    <DIR>          Logs
10/30/2015  03:24 AM    <DIR>          PerfLogs
05/19/2016  08:29 AM            38,508 physmed.htm
06/01/2016  03:07 PM    <DIR>          Program Files
05/26/2016  07:23 AM    <DIR>          Program Files (x86)
04/30/2016  04:11 AM    <DIR>          Users
05/19/2016  08:58 AM            88,646 vol.htm
05/25/2016  02:15 PM    <DIR>          Windows
05/19/2016  08:08 AM           229,392 x.csv
05/19/2016  08:24 AM         2,300,858 x.htm
05/19/2016  07:17 AM            13,514 x.html
05/19/2016  08:27 AM            23,048 x2.htm
               7 File(s)      2,765,450 bytes
               9 Dir(s)  934,558,244,864 bytes free

C:\>
```

Copy File	
Commands	**Description**
`COPY <args>`	COPY – copy files to a new file/location
Examples	
`copy current.jpg new.jpg`	– copies `current.jpg` to new.jpg in the same directory
`copy current.jpg test\new.jpg`	– copies `current.jpg` into the test folder with the new name `new.jpg`



Move File	
Commands	**Description**
`MOVE <args>`	MOVE – moves a file to a new file/location
Examples	
`move current.jpg new.jpg`	– rename the file `current.jpg` to `new.jpg`
`move current.jpg test\`	– move `current.jpg` into the test folder

```
               9 Dir(s)  936,693,649,408 bytes free

C:\>move copy.txt dir.txt
Overwrite C:\dir.txt? (Yes/No/All): y
        1 file(s) moved.

C:\>dir
 Volume in drive C has no label.
 Volume Serial Number is 9AF4-B4AA

 Directory of C:\

04/28/2016  08:37 AM    <DIR>          ADSI Scriptomatic
04/30/2016  04:06 AM    <DIR>          AMD
05/19/2016  08:30 AM            71,484 dd.htm
06/02/2016  08:06 AM             1,033 dir.txt
06/02/2016  07:26 AM    <DIR>          hosts_networks
02/13/2016  09:21 AM    <DIR>          Logs
10/30/2015  03:24 AM    <DIR>          PerfLogs
05/19/2016  08:29 AM            38,508 physmed.htm
06/01/2016  03:07 PM    <DIR>          Program Files
05/26/2016  07:23 AM    <DIR>          Program Files (x86)
04/30/2016  04:11 AM    <DIR>          Users
05/19/2016  08:58 AM            88,646 vol.htm
05/25/2016  02:15 PM    <DIR>          Windows
05/19/2016  08:08 AM           229,392 x.csv
05/19/2016  08:24 AM         2,300,858 x.htm
05/19/2016  07:17 AM            13,514 x.html
05/19/2016  08:27 AM            23,048 x2.htm
               8 File(s)      2,766,483 bytes
```

Commands	Description
ECHO <args>	ECHO – print text to the screen

The echo function can be used to print variables and their values to the screen, which means that it is very useful when users need to debug large commands and large scripts.

```
C:\>echo Print text or variables: %systemroot%
Print text or variables: C:\WINDOWS

C:\>
```

Batch File

Batch files consist of multiple commands, which are executed from the top of the file down to the bottom of the file. For example, try making a text file called "`example.bat`", and include the following in that file:

```
cd C:\Windows\System32
dir
echo This is an example batch file
```

Lesson 4

PowerShell Capabilities

This lesson will describe the basic capabilities of PowerShell. It will prepare the student to begin working in PowerShell and provide insight into the differences and similarities between CMD and PowerShell.

What is PowerShell?

PowerShell is a tool that comes by default with Microsoft Windows. It has been available to users since Microsoft Windows Server 2008 and Windows 7. PowerShell is a powerful tool that has the abilities of CMD with the added benefits of additional reach and ease of use. It works well with one-line commands, just like CMD. For scripting purposes, a ".ps1" file is almost a necessity. It is similar to a batch file, but more granular and powerful. Single-line scripts are a viable alternative to creating .ps1 files.

PowerShell vs. Command

PowerShell can perform all CMD functions and handle many other useful tasks. New PowerShell commands can be created with C#. Most responders will find that PowerShell's ability to execute commands on a remote computer will be their most crucial function, because it will allow them to use scripts and commands on another system remotely. PowerShell can handle package management and also can manage all installed applications on a given system.

PowerShell's Integrated Scripting Environment (ISE) will play a key role in creating PowerShell scripts. It can be found in the Windows Start Menu by searching for "PowerShell_ISE". The scripting environment provides suggestions for incomplete commands and the "Tab" auto complete feature allows the user to automatically finish commands. Pressing Tab multiple times cycles through multiple command options. Similar to most integrated development environments, the ISE uses color codes to differentiate between code such as variables, commands, etc.

Exercise 1-1, Syntactical Exercise: CD and Variables

Exercise 1-2, Practical Exercise: CMD Navigation and Data Gathering for Extraction

Exercise 1-3, Practical Exercise: Batch File Scripting

MODULE 2

Pseudocode

This module will focus on how and why scripting occurs. Students will be using pseudocode and basic scripting functions throughout this module.

OBJECTIVES *After completing this module, students will be able to:*
Indicate the concepts of scripting automation
Interpret conditional control flow
Demonstrate looping control flow
Compose pseudocode using conditionals and loops

Lesson 1

Programming Technology and Concepts

This lesson describes basic programming technologies and concepts within PowerShell. It also discusses the syntax and operations associated with the basics of programming. PowerShell users can automate processes without requiring any user input.

What is a Command?

A program is a set of instructions that can be understood by computer software and all programs are designed to accomplish a goal or set of goals. The programmer that creates a program will start with a task and from there, they define the task's requirements and decide how to accomplish the task. There are many ways to achieve the same task in computer code. Some approaches may be shorter or more efficient, where others may require more steps. The programmer will then describe how they wish to accomplish the task and translate their approach into a computer language.

Basic Programming Operations

Most programs are able to do the these basic functions:

- Retrieve or input external information
- Output or store information externally
- Store information in memory (internally)
- Perform arithmetic
- Make decision about what to do next
- Do something a number of times

The operations in the list above employ control flow in three different ways. Control flow is the approach that a computer takes when it executes and evaluates individual statements, commands or functions of a program. A loop occurs when a set of instructions are executed multiple times until certain conditions are met or if it is designed to run indefinitely. The three types of command flow are:

- Sequential
    - Execute operations one after another
- Selection – IF and ELSE
    - Select a set of operations to execute from multiple sets
    - Use a comparison to select an option
        * Due to the comparison evaluating a condition, these are also called conditional statements, or simply conditionals
- Loop – WHILE
    - Repeat a set of operations a number of times
    - Stop looping based on a comparison

Lesson 2

Logic and Problem Solving

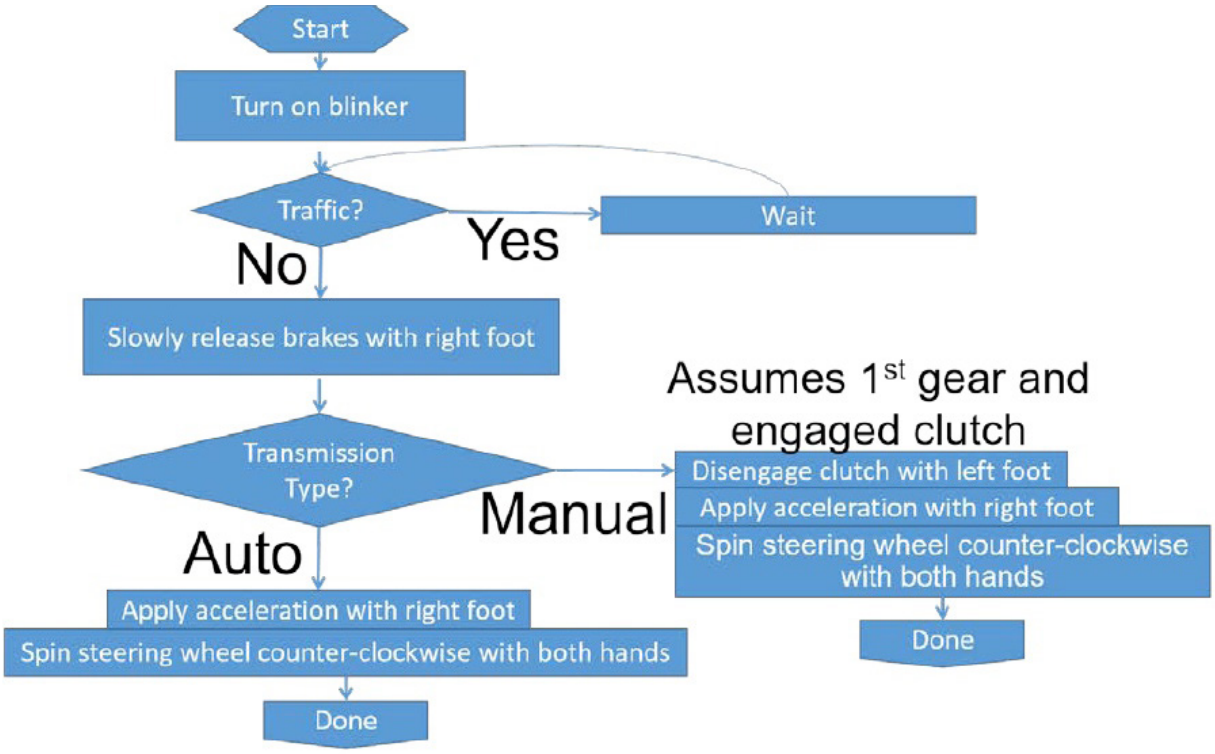
This lesson helps students understand how logic is used within programming and problem solving. It demonstrates how a programmer would outline a program based on a logical flow.

Starting the Programming Process with Logic

Before you start to write a PowerShell script, you must develop a plan. The best methods for documenting a plan are with flowcharts or pseudocode. The next page has flowchart and pseudocode plans that explain how to turn left in a car at a stop sign. This task assumes that the brakes (not including the emergency brake) are already applied.

A flowchart is a diagram using defined symbols that will allow a reader to see the exact path and output of a process. Pseudocode is an outline of your program and how it will run.

Programmer's flowcharts tend to be very detailed and structured. Visio is a modern tool that is widely used to create elaborate flowcharts.



Pseudocode is another method of documenting a program plan. It is flexible, closer to standard English than flowcharts, is more compact and provides better readability. Here is an example of using pseudocode to write directions to the airport.

```
Turn on blinker
While there's traffic:
      Wait
Release brakes slowly with right foot
If the transmission type is automatic:
      Apply acceleration with the right foot AND
      Spin the steering wheel counter-clockwise with
      both hands
Else if the transmission type is manual:
      Disengage clutch with left foot AND
      Apply acceleration with the right foot AND
      Spin the steering wheel counter-clockwise with
      both hands
```

Pseudocode is a way of outlining software designs without having to sacrifice detail.

- **Exercise 2-1, Syntactical Exercise: Pseudocoding**
- **Exercise 2-2, Syntactical Exercise: Pseudocoding**
- **Exercise 2-3, Syntactical Exercise: Pseudocoding**
- **Exercise 2-4, Practical Exercise: Pseudocoding**
- **Exercise 2-5, Syntactical Exercise: Pseudocoding**

MODULE 3

PowerShell Cmdlets and Syntax

This module will describe PowerShell's syntax and examine specific PowerShell cmdlets. It will also prepare the user for the execution of remote administration through PowerShell.

OBJECTIVES *After completing this module, students will be able to:*
Describe the cmdlet verb-noun structure
Identify PowerShell syntax
Construct PowerShell cmdlets
Write PowerShell scripts
Alter PowerShell commands for remote administration

Lesson 1

PowerShell Syntax

This lesson will discuss the basic syntax of PowerShell's cmdlets and explain how to use them correctly.

Syntax and Verbs

PowerShell syntax is based on a specific type of commands that are named *cmdlets* (pronounced "command-lets"). A cmdlet is a two-word command that calls on one or more built-in scripts to grab data of a specified type and then display the data in a logical format. Cmdlets use a verb-noun syntax. Here are a few examples:

- Get-Help
- Start-Process
- Resume-Service

Typical verb actions are as follows:

Verb	Action performed
Get	Displays or retrieves data
Set	Inputs or changes data
New	Creates a new item or object
Remove	Deletes data
Add	Appends data

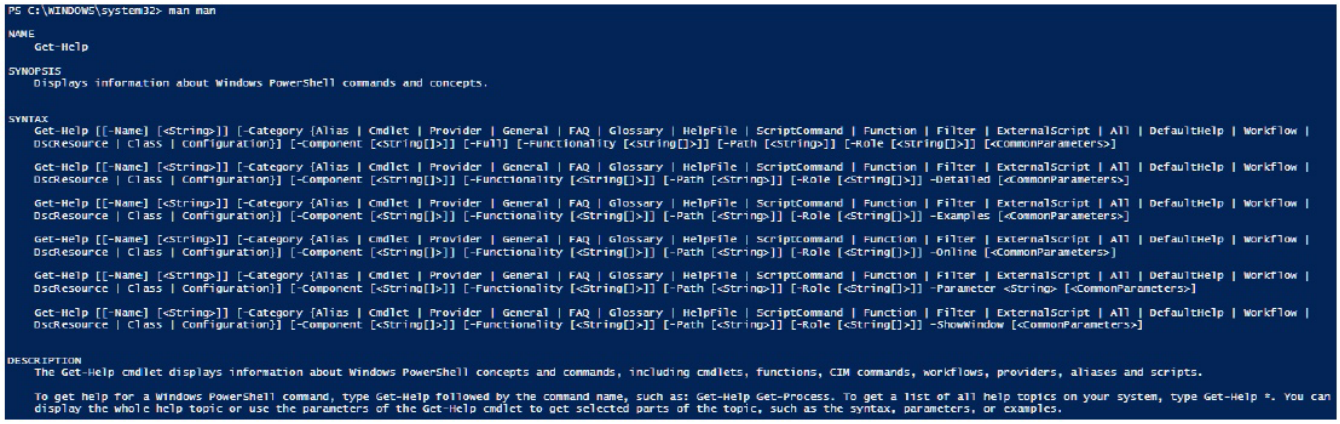
These verbs and commands are all case-insensitive.

For additional syntax help, there are three PowerShell cmdlets that provide the user with information about other cmdlets. These are as follows:

- Get-Help
- Get-Alias
- Get-Command

Get-Help

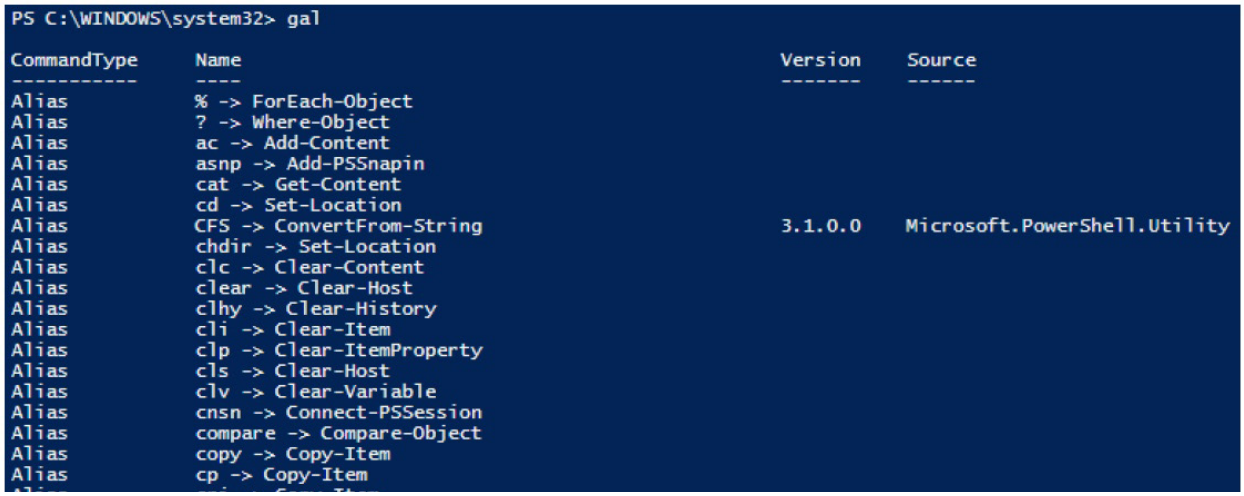
`Get-Help` provides information on PowerShell cmdlets. The Get-Help cmdlet can list optional parameters, required parameters, what type of input to use in parameters, and other technical information regarding cmdlets. Get-Help is called if you enter "man" in PowerShell (i.e. the command to retrieve manual pages in Linux). The primary syntax for the Get-Help cmdlet is simple: `Get-Help <CMDLET>`



(You can even retrieve syntax help for Get-Help via Get-Help.)

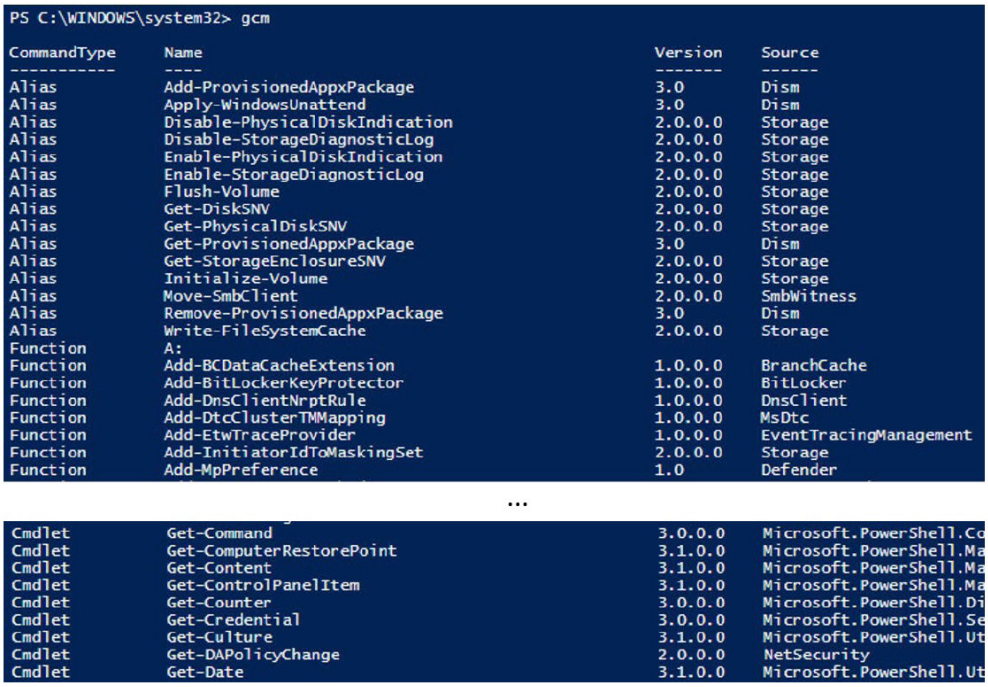
Alias

An alias in PowerShell is text that can be used to invoke a cmdlet. Normally, the alias text is an abbreviated version of a long cmdlet. You have already used an alias if you followed along on your computer, because "man" is just an alias for "Get-Help". To retrieve a list of all aliases within PowerShell, use the cmdlet `Get-Alias`, or its alias `gal`.



Get-Command

To see what PowerShell can do, use Get-Command to list all available cmdlets.



```
PS C:\WINDOWS\system32> gcm

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           Add-ProvisionedAppxPackage                         3.0        Dism
Alias           Apply-WindowsUnattend                              3.0        Dism
Alias           Disable-PhysicalDiskIndication                     2.0.0.0    Storage
Alias           Disable-StorageDiagnosticLog                       2.0.0.0    Storage
Alias           Enable-PhysicalDiskIndication                      2.0.0.0    Storage
Alias           Enable-StorageDiagnosticLog                        2.0.0.0    Storage
Alias           Flush-Volume                                       2.0.0.0    Storage
Alias           Get-DiskSNV                                        2.0.0.0    Storage
Alias           Get-PhysicalDiskSNV                                2.0.0.0    Storage
Alias           Get-ProvisionedAppxPackage                         3.0        Dism
Alias           Get-StorageEnclosureSNV                            2.0.0.0    Storage
Alias           Initialize-Volume                                  2.0.0.0    Storage
Alias           Move-SmbClient                                     2.0.0.0    SmbWitness
Alias           Remove-ProvisionedAppxPackage                      3.0        Dism
Alias           Write-FileSystemCache                              2.0.0.0    Storage
Function        A:
Function        Add-BCDataCacheExtension                           1.0.0.0    BranchCache
Function        Add-BitLockerKeyProtector                          1.0.0.0    BitLocker
Function        Add-DnsClientNrptRule                              1.0.0.0    DnsClient
Function        Add-DtcClusterTMMapping                            1.0.0.0    MsDtc
Function        Add-EtwTraceProvider                               1.0.0.0    EventTracingManagement
Function        Add-InitiatorIdToMaskingSet                        2.0.0.0    Storage
Function        Add-MpPreference                                   1.0        Defender
                                                                   ...
Cmdlet          Get-Command                                        3.0.0.0    Microsoft.PowerShell.Co
Cmdlet          Get-ComputerRestorePoint                           3.1.0.0    Microsoft.PowerShell.Ma
Cmdlet          Get-Content                                        3.1.0.0    Microsoft.PowerShell.Ma
Cmdlet          Get-ControlPanelItem                               3.1.0.0    Microsoft.PowerShell.Ma
Cmdlet          Get-Counter                                        3.0.0.0    Microsoft.PowerShell.Di
Cmdlet          Get-Credential                                     3.0.0.0    Microsoft.PowerShell.Se
Cmdlet          Get-Culture                                        3.1.0.0    Microsoft.PowerShell.Ut
Cmdlet          Get-DAPolicyChange                                 2.0.0.0    NetSecurity
Cmdlet          Get-Date                                           3.1.0.0    Microsoft.PowerShell.Ut
```

As a last note on PowerShell syntax, we'll add that variables are prefaced with a "$". For example, $test is a PowerShell variable called "test."

Lesson 2

PowerShell Setup

This lesson will show students how to configure PowerShell's ability to define what scripts are authorized to run on the software (i.e. execution policies)

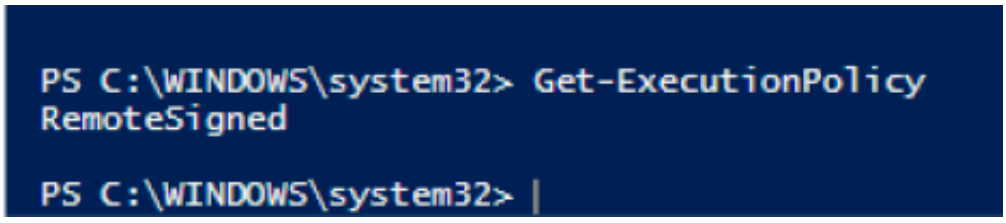
PowerShell Setup

Execution Policies

PowerShell has five types of execution policies. To set the execution policy for PowerShell, you must run the following as an administrator:

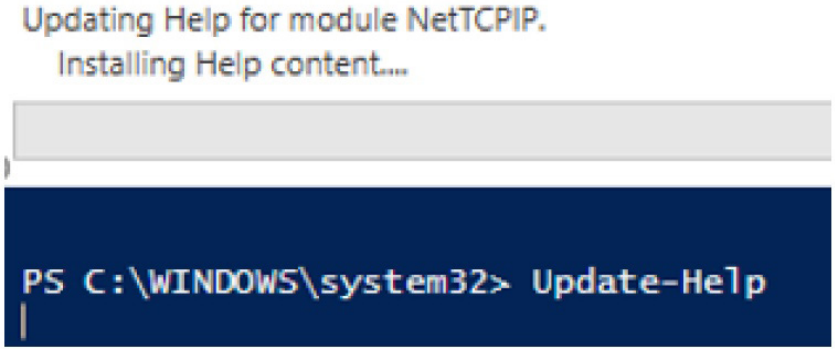
```
Set-ExecutionPolicy-<NameOfExecutionPolicy>
```

Execution Policies	Description
Restricted	This is the default execution policy for PowerShell. No scripts can be called to run. Select Microsoft-signed scripts can be run by the system.
AllSigned	Only scripts signed by a trusted publisher can be run.
RemoteSigned	If the script was downloaded, it must be signed by a trusted publisher to run.
Unrestricted	Unsigned script can be run; provides warning prior to running scripts and configuration files from the Internet.
Bypass	Nothing is blocked, and no warnings or prompts are provided. Does not check Execution Policy before running scripts.



Update Help

When you are running PowerShell, you need to update the Get-Help cmdlet. To update help, simply run Update-Help as an administrator.



Lesson 3

PowerShell Remote Execution

This lesson will show students how to run cmdlets remotely using PowerShell.

Remote PowerShell Administration

PowerShell responders must be able to contact other computers and run commands remotely. To do so, the user must establish a remote connection. To create a remote connection, do the following:

1. Run the following PowerShell cmdlet on the remote system as an administrator: `Enable-PSRemoting`. This cmdlet configures the remote computer to receive remote PS cmdlets. For this to work, the machine must be on the home or work network.

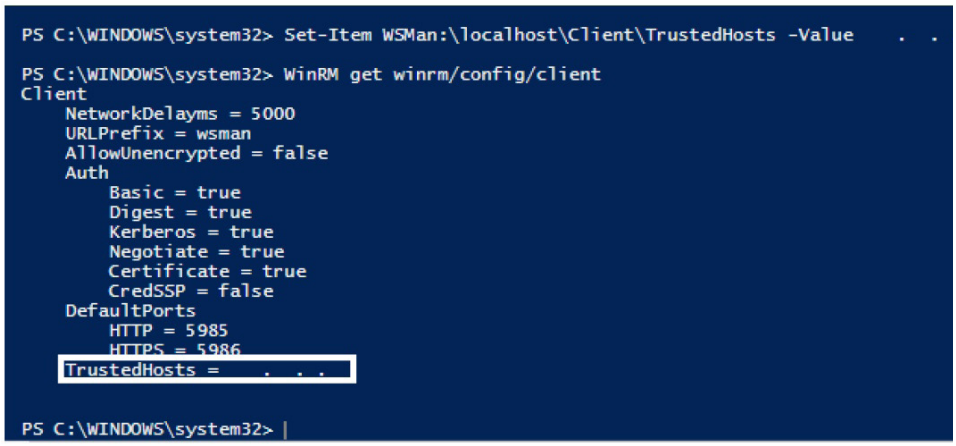
2. Run the `New-PSSession` cmdlet on the remote machine to create a persistent "remote" session on localhost (the same machine which `Enable-PSRemoting` was initiated). Attempt to run this cmdlet on the remote host. If this cmdlet is run successfully, remoting is verified.

```
PS C:\Users\Administrator> New-PSSession

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

PS C:\Users\Administrator>
```

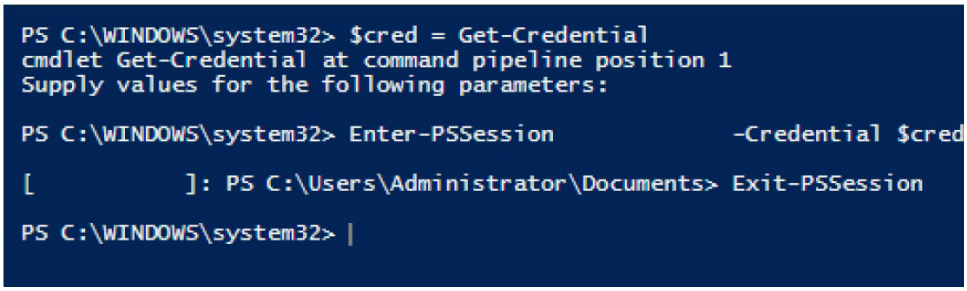
3. Ensure that PowerShell is being run as an administrator on the local machine.

4. Configure your local machine so it trusts specified hosts to receive sensitive credentials. To do so, enter the following cmdlet: `Set-Item WSMan:\localhost\Client\TrustedHosts -Value X.X.X.X` (where x.x.x.x is replaced with the remote IP). When prompted to allow this change, type "yes" or "y" (case-insensitive).

5. Use the following cmdlet to verify that the remote connection was established: `WinRM get winrm/config/client`



6. Send login credentials to the remote machine with the following command: `$cred = Get-Credential`. From there, you may input the username and password of the remote user. To enter the session, use the command `Enter-PSSession X.X.X.X -Credential $cred`.



Lesson 4

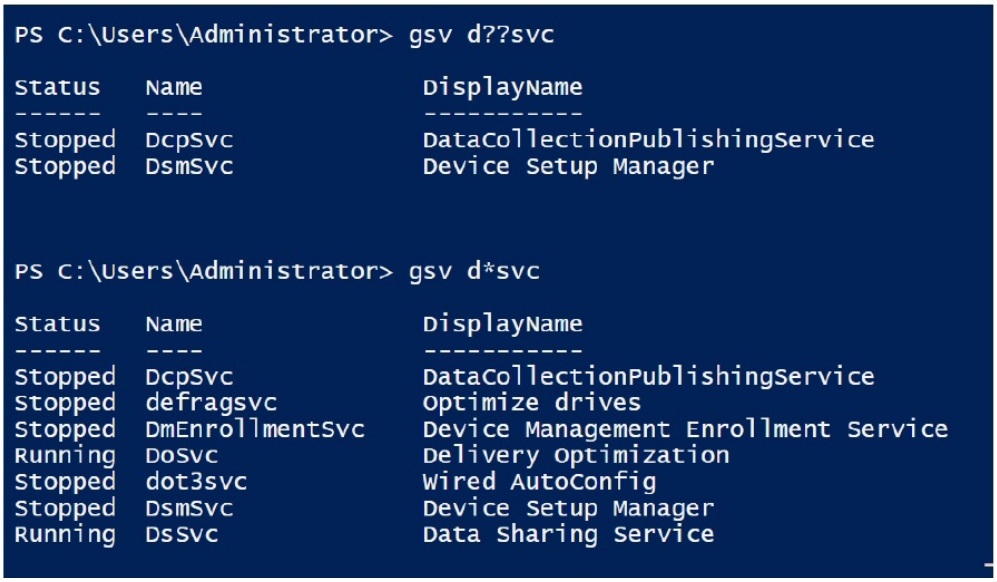
PowerShell Commands and Cmdlets

As stated in the previous module, PowerShell has more commands than CMD. Examine the output from the native PowerShell cmdlet Get-Service, which outputs a list of available services and their states:

```
[           ]: PS C:\Users\Administrator\Documents> Get-Service

Status   Name               DisplayName
------   ----               -----------
Running  AeLookupSvc        Application Experience
Stopped  ALG                Application Layer Gateway Service
Stopped  AppIDSvc           Application Identity
Stopped  Appinfo            Application Information
Stopped  AppMgmt            Application Management
Running  AudioEndpointBu... Windows Audio Endpoint Builder
Running  AudioSrv           Windows Audio
Stopped  AxInstSV           ActiveX Installer (AxInstSV)
Stopped  BDESVC             BitLocker Drive Encryption Service
Running  BFE                Base Filtering Engine
Stopped  BITS               Background Intelligent Transfer Ser...
Stopped  Browser            Computer Browser
Stopped  bthserv            Bluetooth Support Service
Stopped  CertPropSvc        Certificate Propagation
Stopped  clr_optimizatio... Microsoft .NET Framework NGEN v2.0....
Stopped  clr_optimizatio... Microsoft .NET Framework NGEN v2.0....
Running  COMSysApp          COM+ System Application
Running  CryptSvc           Cryptographic Services
Running  CscService         Offline Files
Running  DcomLaunch         DCOM Server Process Launcher
Running  defragsvc          Disk Defragmenter
Running  Dhcp               DHCP Client
Running  Dnscache           DNS Client
Stopped  dot3svc            Wired AutoConfig
Running  DPS                Diagnostic Policy Service
Stopped  EapHost            Extensible Authentication Protocol
Stopped  EFS                Encrypting File System (EFS)
Stopped  ehRecvr            Windows Media Center Receiver Service
Stopped  ehSched            Windows Media Center Scheduler Service
```

There are special characters used in PowerShell that can help you search for or filter certain data. One example is the asterisk (aka "star", or "splat"): "*". In PowerShell, the asterisk can serve as a wildcard character, which means that it is a character used to represent any other character. Also, the asterisk represents zero or more of any character. For example, "*" could represent "" (nothing), "`aaa`", or "`something entirely too long`". In PowerShell, if you entered `Get-Service Win*,` the command will retrieve all services starting with Win and ending with anything (including any services simply named `Win`).

Another special character used for searching and filtering is the question mark: "`?`". The question mark is another type of PowerShell wildcard—one that represents only a single character. For example, the command `Get-Service d??svc` would retrieve all services starting with the character *d*, followed by any two characters, and ending with the characters *svc*. As shown below, searching with wildcards may find data that you were not looking for, which is an important part of wildcard searches.



As the user, you can search for and through lists of cmdlets. For example, `Get-Command *service*` outputs all commands, aliases, functions, filters, scripts and applications that are installed on the computer with the word service in their name.

Exercise 3-1, Syntactical Exercise: Get-Command and Wildcards

PowerShell also uses special characters to denote variables. As a note, some special characters will interfere with running CMD and PowerShell commands properly, so make sure you are using the special character correctly. The dollar sign, "`$`", denotes a variable in PowerShell. For example, if you want to create a variable named "`x`" with a value of 10, the command would look like the following: `$x = 10`. To output the value of x, you could simply run the following: `echo $x`. In addition, a dollar sign followed by an underscore, "`$_`" is used to indicate a special variable in PowerShell, and we will discuss that later in the module.

```
[            ]: PS C:\Users\Administrator\Documents> $x = 10

[            ]: PS C:\Users\Administrator\Documents> echo $x
10
```

WhatIf

`WhatIf` is a special option available on some cmdlets. The user can set the `WhatIf` option to verify the intended action of a cmdlet without affecting the environment (like a dry run). Once the intended result is verified, the `WhatIf` option can be removed, which allows the cmdlet to execute normally.

Also, `WhatIf` is always active if the `WhatIf` environmental variable is set to true.

$WhatIfPreference = $true

1. Run `dir variable:\` to be certain (lists all variables)
2. If the `WhatIfPreference` is set to `True` then the action `Stop-Computer` will only be simulated

```
PSVersionTable                  {CLRVersion, BuildVersion, PSVersion, WSManStackVersion...}
PWD                             C:\Users\Administrator\Documents
ReportErrorShowExceptionClass   0
ReportErrorShowInnerException   0
ReportErrorShowSource           1
ReportErrorShowStackTrace       0
ShellId                         Microsoft.PowerShell
StackTrace                         at System.Management.Automation.FlowControlNode.Execute(A
true                            True
VerbosePreference               SilentlyContinue
WarningPreference               Continue
WhatifPreference                True


[            ]: PS C:\Users\Administrator\Documents> Stop-Computer
What if: Performing operation "Stop-Computer" on Target "localhost (WIN-JQ6S8LTTJHO)".

[            ]: PS C:\Users\Administrator\Documents> |
```

Exercise 3-2, Syntactical Exercise: Get-Command, Wildcards, Service Manipulation, and WhatIf

Read-Host

The `Read-Host` cmdlet receives input from the command line. For example, the cmdlet `Read-Host "Enter a number"` will ask the user to enter a number. When any character or string of characters is entered, the number or character(s) will be written to the screen. You may also have the user's input sent to a variable. For example, `$x = Read-host "Enter a number"` will do the same as the previous command, but will assign the input to the variable "x". To test this, you can then type $x and it should output the same string that was entered by the user.

NOTE: Any user input provided to `Read-Host` will be input as a string of characters, which means that PowerShell will not interpret numbers as numbers, but instead as simply characters, or an extension of letters that don't hold any inherent value.

```
PS C:\WINDOWS\system32> Read-Host "Enter a number"
Enter a number: 3
3

PS C:\WINDOWS\system32> $x = Read-Host "Enter a numbe
Enter a number: 4

PS C:\WINDOWS\system32> $x
4
```

The previous paragraph leads us to consider datatypes, which define how PowerShell handles your data. Datatypes can be altered in PowerShell by using the following syntax: `[DATATYPE]$VARIABLE`, where `DATATYPE` is chosen from the list below, and `VARIABLE` is the variable whose datatype you wish to modify.

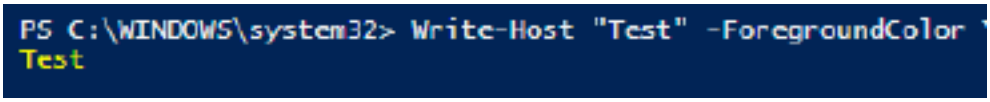
Datatypes

Datatype	Description
`int`	Short for integer, meaning a whole number E.g.: 5, 6, 7 #NOT: 5.5, 3.14, 7¾
`string`	Uses text as text, and nothing more E.g.: "Test", "Hello, World!", "Fifty-six" #"56" would be read as text, too
`float`	Short for floating point number, meaning a decimal point number E.g.: 5.5, 3.14, 7.75 #Also compatible with: 5, 6, 7

Exercise 3-3, Syntactical Exercise: Variables, Casting, and User Input

Write-Host

`Write-Host` outputs text to the command line. For example, try the following: `Write-Host "Test" –ForegroundColor Yellow`. This color is useful for warnings, errors, or highlighting items of importance with PowerShell.

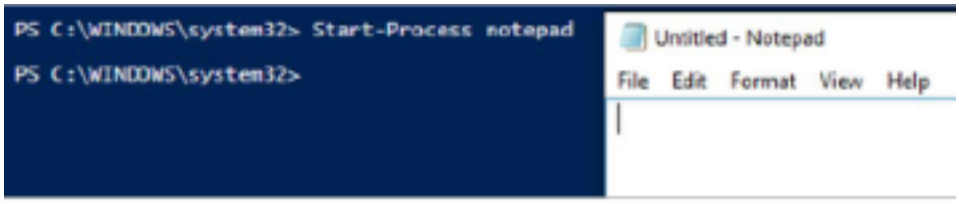


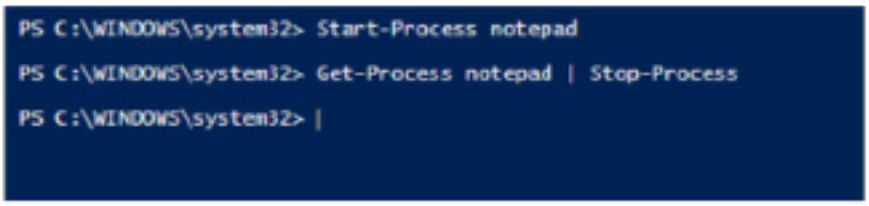
| - Piping

PowerShell's pipe character enables the output of one command to be used as the input of another. After commands are connected with pipe characters, they are called a "pipeline." When users are connecting commands via pipes, they are "piping" commands and chained commands can be referred to as "piped." Let's create an example in PowerShell:

1. Run the command `Start-Process notepad` to open Notepad.

2. Run the command `Get-Process notepad | Stop-Process`. You are asking PowerShell to retrieve the process of Notepad and pipe that process into the `Stop-Process` cmdlet, which stops the retrieved process. It close the Notepad application (along with any other instances of Notepad open; be careful).





Get-Member

`The Get-Member` cmdlet enumerates an object or cmdlet's properties and methods. It allows the user to know what they can or cannot access, understand which properties and methods are available, and allows the user to write a script or command without having to memorize every object model.

```
PS C:\WINDOWS\system32> Get-Service | gm


   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 desiredS
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();
```

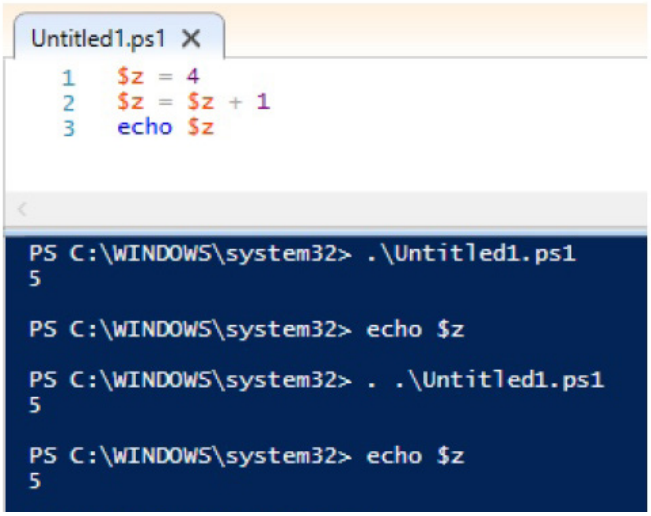
As shown in the graphic on the previous page, `Get-Service | gm` shows the member type and definition of the services on the machine. The member type called "Property," as well as its aliased counterpart "AliasProperty," contain information about the object to which they belong.

. (Dot or Period)

PowerShell allows users to access object members, such as properties or methods, with the period. The period is placed after a variable name with the property following (as follows): `$VARIABLE.PROPERTY.` To see how PowerShell uses the period, run the following: `$Service = Get-Service WinRM`. This command stores the output of `Get-Service WinRM` into `$Service`. To access the property `DisplayName of $Service`, run the following: `$Service.DisplayName.`

Another use of the period in PowerShell is called *dot sourcing*. Dot sourcing adds functions, aliases, and variables of a script to the current PowerShell script or session. As you can see, this allows a user to call any variable from the script after it finishes. For example, if the command below (which uses dot sourcing) is executed, the outcome is displayed in the following graphic:

```
. .\PSScript.ps1
```



( ) – Grouping Expression

The parentheses are special in PowerShell. They are used for different tasks in different contexts:

1. Parentheses are used to surround a command and act as a variable containing the output of the enclosed command. In this way, members (properties and methods) of the enclosed command can be accessed as a variable that holds the command's output object.

   To demonstrate how parentheses are used in this context, we'll browse the members of the `Get-Command` output using the following command: `gcm Get-Service | gm.` Next, we'll pick a member of interest from the output. As an example, we'll use the property `CommandType`, but you can try one of your choosing. Finally, we'll run the following: `(gcm Get-Service).CommandType.`

```
PS C:\Users\Administrator\Desktop> gcm Get-Service | gm


   TypeName: System.Management.Automation.CmdletInfo

Name                 MemberType     Definition
----                 ----------     ----------
Equals               Method         bool Equals(System.Object obj)
GetHashCode          Method         int GetHashCode()
GetType              Method         type GetType()
ResolveParameter     Method         System.Management.Automation.Param
ToString             Method         string ToString()
CommandType          Property       System.Management.Automation.Comma
DefaultParameterSet  Property       string DefaultParameterSet {get;}
Definition           Property       string Definition {get;}
```

```
PS C:\Users\Administrator\Desktop> (gcm Get-Service).CommandType
Cmdlet
```

2. Parentheses can also be used for mathematical purposes in PowerShell. In a mathematical string, precedence is given to cmdlets within the parentheses as follows:

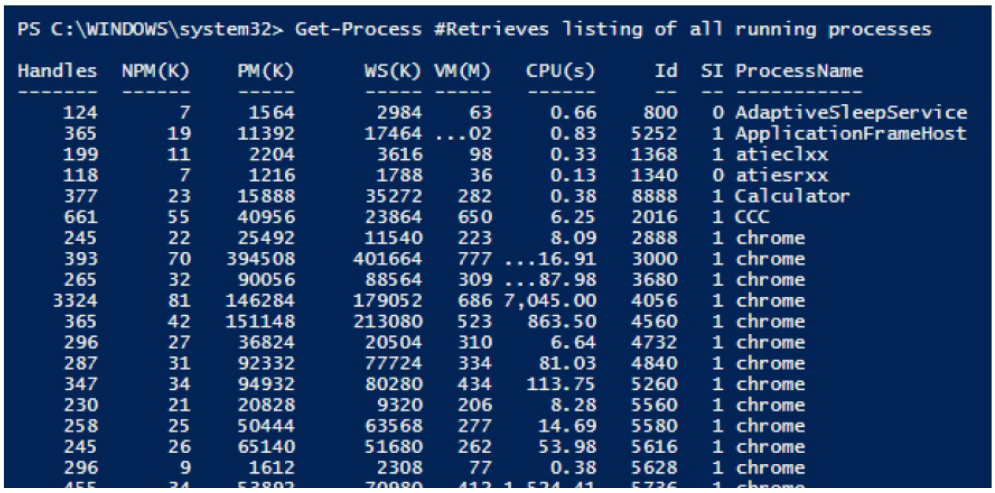
```
$x = 4
($x +3) * 5
echo $x+2
echo ($x+2)
```

```
PS C:\WINDOWS\system32> $x = 4

PS C:\WINDOWS\system32> ($x + 3) * 5
35

PS C:\WINDOWS\system32> echo $x + 2
4
+
2

PS C:\WINDOWS\system32> echo ($x + 2)
6

PS C:\WINDOWS\system32>
```

As shown in the output, the grouping allows the user to specify the order in which things are completed by the system.

(Hash or Number Sign) - Comment Output

When "#" is used in PowerShell scripts, it specifies that everything following the hash sign should not be interpreted as a command until the line has ended. The hash sign allows comments to be placed after commands and is most useful in scripts. For example, `Get-Process #Retrieves listing of all running processes` will run the cmdlet `Get-Process`, but also will allow a user to see the comment placed after the "#" without PowerShell attempting to execute it as part of the command.



```
PS C:\WINDOWS\system32> Get-Process #Retrieves listing of all running processes

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id  SI ProcessName
-------  ------    -----      ----- -----   ------     --  -- -----------
    124       7     1564       2984    63     0.66    800   0 AdaptiveSleepService
    365      19    11392      17464 ...02     0.83   5252   1 ApplicationFrameHost
    199      11     2204       3616    98     0.33   1368   1 atieclxx
    118       7     1216       1788    36     0.13   1340   0 atiesrxx
    377      23    15888      35272   282     0.38   8888   1 Calculator
    661      55    40956      23864   650     6.25   2016   1 CCC
    245      22    25492      11540   223     8.09   2888   1 chrome
    393      70   394508     401664   777 ...16.91   3000   1 chrome
    265      32    90056      88564   309 ...87.98   3680   1 chrome
   3324      81   146284     179052   686 7,045.00   4056   1 chrome
    365      42   151148     213080   523   863.50   4560   1 chrome
    296      27    36824      20504   310     6.64   4732   1 chrome
    287      31    92332      77724   334    81.03   4840   1 chrome
    347      34    94932      80280   434   113.75   5260   1 chrome
    230      21    20828       9320   206     8.28   5560   1 chrome
    258      25    50444      63568   277    14.69   5580   1 chrome
    245      26    65140      51680   262    53.98   5616   1 chrome
    296       9     1612       2308    77     0.38   5628   1 chrome
```

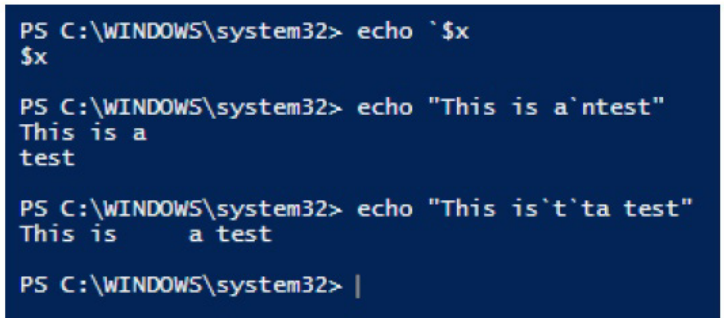
` (Backtick)- Escape Character

(Note: Backtick, not apostrophe)

When you use the backtick (" ` ") or backquote in PowerShell, the characters that follow it are printed outside the normal PowerShell context. For example the command `echo ``$x` will print the characters "$x" instead of the contents of the variable $x. The escape character forces PowerShell to treat the "$" as a character instead of a PowerShell function. Additionally, the backtick has special uses, including:

`` `n – Newline character ``
Try: echo `"This is a`ntest"`

`` `t – Tab ``
Try: echo `"This is`t`ta test"`

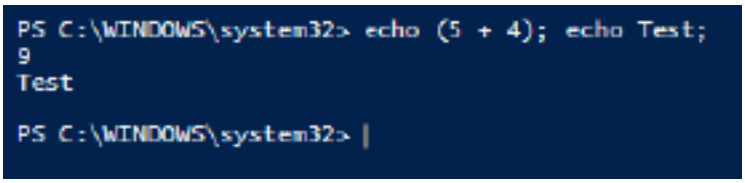


Also, the backtick is used with specific characters that have special uses relating to formatting.

; - Denotes end of command

This allows separation of commands, both in single and multi-line scripts.

```
echo (5 + 4); echo Test;
```



Additional Special Characters

The following are additional special characters to be aware of:

@

::

,

{}

[]

These characters have special meaning in PowerShell, so avoid using them outside of the proper context and do not use them with a backtick or else you will encounter unexpected results.

For any CMD commands that use PS special characters in their arguments, try: `CMD.EXE /C “echo `Run ;legacy @cmds {here}!"` This method passes the command to CMD without parsing the special characters.

```
PS C:\WINDOWS\system32> CMD.EXE /C "echo `Run ;legacy @cmds {here}!"
Run ;legacy @cmds {here}!
```

Get-Package

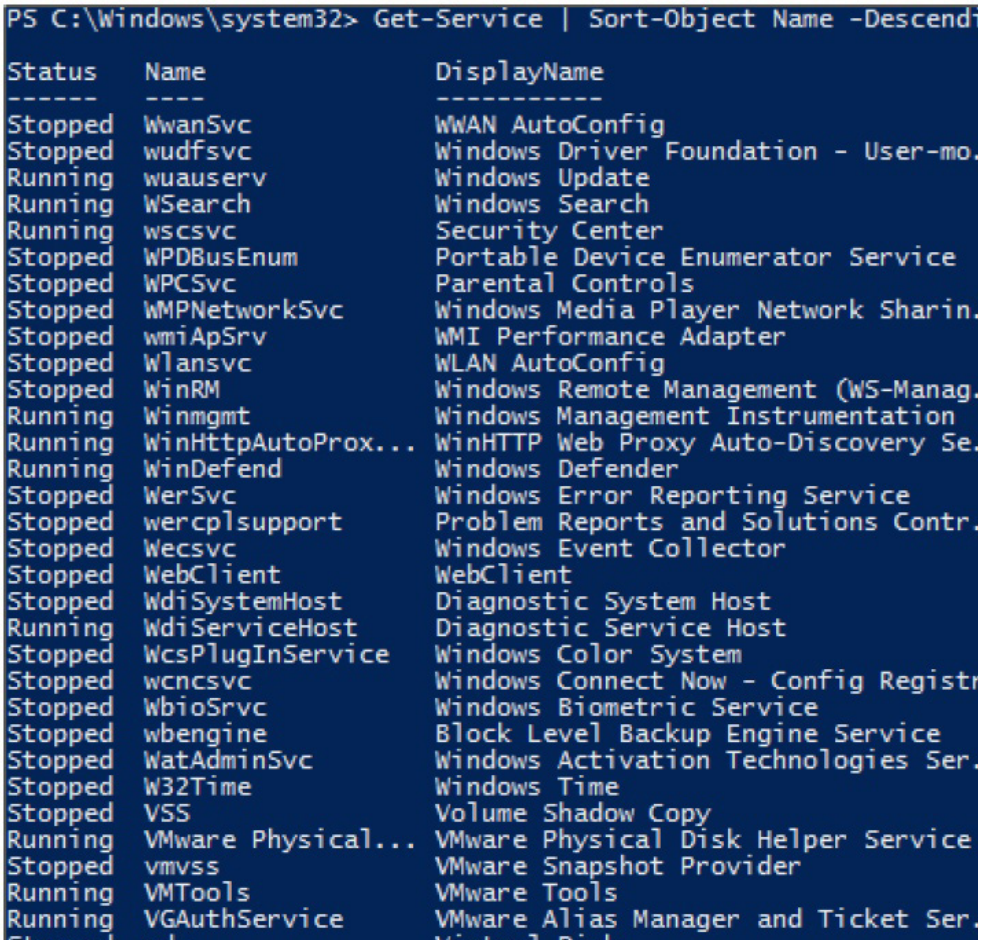
`Get-Package` is a useful cmdlet that retrieves a list of installed software and Windows updates on the system.

```
PS C:\WINDOWS\system32> Get-Package

Name                           Version          Source                           Summary
----                           -------          ------                           -------
Microsoft Office 365 ProPlu... 16.0.6001.1078
WinRAR 5.31 (64-bit)           5.31.0
AMD FirePro Settings           2015.1129.230...
Office 16 Click-to-Run Exte... 16.0.6001.1078   c:\program files (x86)\micr...
Office 16 Click-to-Run Loca... 16.0.6001.1078   c:\program files (x86)\micr...
Office 16 Click-to-Run Lice... 16.0.6001.1078   c:\program files (x86)\micr...
Microsoft .NET Framework 4.... 4.5.50932        C:\ProgramData\Package Cach...
Microsoft SQL Server 2016 T... 13.0.1100.286    C:\ProgramData\Package Cach...
Microsoft SQL Server Compac... 4.0.8876.1       C:\Program Files (x86)\Micr...
Microsoft .NET Framework 4.... 4.5.51209        C:\ProgramData\Package Cach...
Microsoft Visual Studio 201... 14.0.23107       C:\ProgramData\Package Cach...
Catalyst Control Center Nex... 2015.1129.230... C:\AMD\WU-CCC2\ccc2_install...
Microsoft SQL Server 2016 T... 13.0.12000.52     C:\ProgramData\Package Cach...
Microsoft Visual Studio 201... 14.0.23107       C:\ProgramData\Package Cach...
MSBuild/NuGet Integration 1... 14.0.25123       C:\ProgramData\Package Cach...
Catalyst Control Center Nex... 2015.1129.230... C:\AMD\WU-CCC2\ccc2_install...
IIS Express Application Com...
Microsoft Visual Studio Com... 14.0.23107       C:\ProgramData\Package Cach...
Visual C++ IDE Base Resourc... 14.0.25123       C:\ProgramData\Package Cach...
Microsoft Visual Studio 201... 14.0.25123       C:\ProgramData\Package Cach...
Blend for Visual Studio SDK... 3.0.40218.0      C:\ProgramData\Package Cach...
Microsoft .NET Framework 4.... 4.6.1055         C:\ProgramData\Package Cach...
```

Sort-Object

This cmdlet sorts objects based on arguments, such as sorting in descending or ascending order. For example:

```
Get-Service | Sort-Object Name –Descending
```



```
PS C:\Windows\system32> Get-Service | Sort-Object Name -Descend

Status   Name               DisplayName
------   ----               -----------
Stopped  WwanSvc            WWAN AutoConfig
Stopped  wudfsvc            Windows Driver Foundation - User-mo.
Running  wuauserv           Windows Update
Running  WSearch            Windows Search
Running  wscsvc             Security Center
Stopped  WPDBusEnum         Portable Device Enumerator Service
Stopped  WPCSvc             Parental Controls
Stopped  WMPNetworkSvc      Windows Media Player Network Sharin.
Stopped  wmiApSrv           WMI Performance Adapter
Stopped  Wlansvc            WLAN AutoConfig
Stopped  WinRM              Windows Remote Management (WS-Manag.
Running  Winmgmt            Windows Management Instrumentation
Running  WinHttpAutoProx... WinHTTP Web Proxy Auto-Discovery Se.
Running  WinDefend          Windows Defender
Stopped  WerSvc             Windows Error Reporting Service
Stopped  wercplsupport      Problem Reports and Solutions Contr.
Stopped  Wecsvc             Windows Event Collector
Stopped  WebClient          WebClient
Stopped  WdiSystemHost      Diagnostic System Host
Running  WdiServiceHost     Diagnostic Service Host
Stopped  WcsPlugInService   Windows Color System
Stopped  wcncsvc            Windows Connect Now - Config Registr
Stopped  WbioSrvc           Windows Biometric Service
Stopped  wbengine           Block Level Backup Engine Service
Stopped  WatAdminSvc        Windows Activation Technologies Ser.
Stopped  W32Time            Windows Time
Stopped  VSS                Volume Shadow Copy
Running  VMware Physical... VMware Physical Disk Helper Service
Stopped  vmvss              VMware Snapshot Provider
Running  VMTools            VMware Tools
Running  VGAuthService      VMware Alias Manager and Ticket Ser.
```

Exercise 3-4, Syntactical Exercise: Sorting

Select-Object

The `Select-Object` cmdlet retrieves objects (columns) from a collection based on specified criteria. The user is to specify the criteria after the cmdlet.

```
Get-Process | Select-Object Name, CPU
```

This script will output only the `CPU` and `Name` columns from the `Get-Process` output.

```
PS C:\Windows\system32> Get-Process | Select-Object Name, CPU

Name                    CPU
----                    ---
CodeMeter               3.9312252
CodeMeterCC             0.9204059
conhost                 0.4524029
csrss                   3.4632222
csrss                   2.7456176
dinotify                0.0156001
dwm                     6.9108443
EnCase                  1.1388073
EnCase                  1.3104084
enhkey.dll              0
explorer                14.0556901
hasplms                 2.6988173
Idle
lsass                   1.2324079
lsm                     0.0468003
msdtc                   0.0624004
notepad                 0.0312002
powershell              0.5148033
SearchIndexer           1.7940115
services                0.9984064
smss                    0.0468003
spoolsv                 0.0780005
svchost                 1.6068103
svchost                 0.4212027
svchost                 0.7488048
svchost                 0.1404009
svchost                 1.0296066
svchost                 28344.2116925
svchost                 2.1684139
```

```
Get-Process | Sort-Object CPU | Select-Object –last 3
```

```
PS C:\Windows\system32> Get-Process | Sort-Object CPU | Select-Object -last 3

Handles  NPM(K)    PM(K)      WS(K) VM(M)   CPU(s)     Id ProcessName
-------  ------    -----      ----- -----   ------     -- -----------
    374      49    74756      47580   257    14.45   2496 svchost
    250      18    46052      48180   310   215.19   2176 TrustedInstaller
   2133    1253   705556     392720  1157 ...50.67    952 svchost
```

To demonstrate the customization options available with `Select-Object`, the above script outputs three processes with the longest CPU time in seconds.

If, ElseIf, Else – If Statements

Like other programming languages, PowerShell uses `if` statements to put specific restrictions or contingencies on how a script runs. The following are comparison operators that can be used in PowerShell `if` statements:

Operator	Definition
- lt	Less than
- le	Less than or equal to
- gt	Greater than
- ge	Greater than or equal to
- eq	Equal to
- ne	Not equal to
- contains	Determ ine elements in a group
- notcontains	Determine excluded elements in a group

Using the cmdlets and operators above, we can write the following:

```
if (5 -gt 10) {"5 > 10"} else {"5 <= 10"}
```

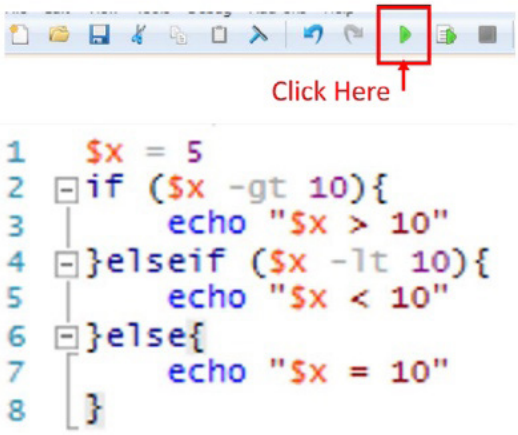
If the condition `(5 -gt 10)` is `True`, the code block (script contained within the braces) that follows will be executed. If the condition is `False`, PowerShell will skip the `if` code block and will run the next command. If the next command is an `else` statement, the `else` code block will be run. Perhaps the most important item to note is that, when using `if` statements, up to one block of code will be run. No block of code needs to be run when using a single `if` statement, but one will run if an `else` statement is involved. The example above produces the following output:

```
PS C:\Windows\system32> if (5 -gt 10) {"5 > 10"} else {"5 <= 10"}
5 <= 10
```

The final component of the `if` statement is the `elseif` statement, which is simply another `if` statement, or another condition to check, before going to the `else` script block. Again, up to one script block will run when using `if`, `elseif`, and `else` statements, and if no `else` is used, the `if` or `elseif` script block may not run at all.

If you write `if` statements in PowerShell ISE's script pane, you can write your script line by line, which will improve its readability. Try to predict what the output of the following script will look like. Then, try running it yourself.

```
1.$x = 5
2.  if ($x -gt 10){
3.  echo “"$x > 10”"
4.} elseif ($x -lt 10){
5.  echo “"$x < 10”"
6. } else {
7.  echo “"$x = 10”"
8.}
```



Try changing the value of the `$x` variable in the last example, and seeing which script block runs as a result. For example, you might change `$x` to the following values:

- `$x = 10`
- `$x = 100`
- `$x = 1`

`$x = 10`

```
PS C:\> $x = 10
if ($x -gt 10){
    echo "$x > 10"
} elseif ($x -lt 10){
    echo "$x < 10"
} else {
    echo "$x = 10"
}

10 = 10
```

`$x = 100`

```
PS C:\> $x = 100
if ($x -gt 10){
    echo "$x > 10"
} elseif ($x -lt 10){
    echo "$x < 10"
} else {
    echo "$x = 10"
}

100 > 10
```

`$x = 1`

```
PS C:\> $x = 1
if ($x -gt 10){
    echo "$x > 10"
} elseif ($x -lt 10){
    echo "$x < 10"
} else {
    echo "$x = 10"
}

1 < 10
```

Exercise 3-5, Syntactical Exercise: If Statements, Else Statements, and Scripting

Where-Object

The Where-Object cmdlet will search each row of data for a specified condition and return that row if the condition is met. Think of Where-Object as a filter.

```
Get-Service | Where-Object {$_.status –eq "Running"}
```

`$_` is a variable that holds special meaning in PowerShell. It holds a single object taken from the pipeline. In the example above, `$_` holds one service at a time. The status of each service is then checked against the `Where-Object` condition. If the condition is true, that service is retained, and is output to the screen. Examine the next screenshot or try it for yourself.

```
PS C:\Windows\system32> Get-Service | Where-Object {$_.status -eq "Running"}

Status   Name               DisplayName
------   ----               -----------
Running  Appinfo            Application Information
Running  AudioEndpointBu... Windows Audio Endpoint Builder
Running  AudioSrv           Windows Audio
Running  BFE                Base Filtering Engine
Running  BITS               Background Intelligent Transfer Ser...
Running  bthserv            Bluetooth Support Service
Running  CodeMeter.exe      CodeMeter Runtime Server
Running  CryptSvc           Cryptographic Services
Running  CscService         Offline Files
Running  DcomLaunch         DCOM Server Process Launcher
Running  Dhcp               DHCP Client
Running  DiagTrack          Diagnostics Tracking Service
Running  Dnscache           DNS Client
Running  DPS                Diagnostic Policy Service
Running  eventlog           Windows Event Log
Running  EventSystem        COM+ Event System
Running  FDResPub           Function Discovery Resource Publica...
Running  FontCache          Windows Font Cache Service
Running  gpsvc              Group Policy Client
Running  hasplms            Sentinel LDK License Manager
Running  iphlpsvc           IP Helper
Running  LanmanServer       Server
Running  LanmanWorkstation  Workstation
Running  lmhosts            TCP/IP NetBIOS Helper
Running  MpsSvc             Windows Firewall
Running  MSDTC              Distributed Transaction Coordinator
Running  Netman             Network Connections
Running  netprofm           Network List Service
Running  NlaSvc             Network Location Awareness
```

Byte Conversions

Because data sizes are measured in different units in different contexts, PowerShell allows you to refer to one kilobyte as `1KB`, 5 megabytes as `5MB`, or 20 gigabytes as `20GB`, and so on. Within PowerShell, byte measurements are in powers of 10 and PowerShell operates in bytes by default. The syntax of byte conversions is as follows: `[NUMBER][BYTE ID]`. The byte identifiers are:

- `KB` - Kilobyte
- `MB` – Megabyte
- `GB` – Gigabyte
- `TB` – Terabyte

Demonstrate byte conversion by entering the following command in PowerShell :

```
$x = 1KB; echo $x;
```

Exercise 3-6, Practical Exercise: Scripting, Where-Object, and Byte Conversions

Compare-Object

This cmdlet compares a collection of PowerShell objects against another collection of objects and detects differences between the two, independent of order. For example, Compare-Object will not show any differences between one collection containing all the current processes sorted by WorkingSet and another collection containing the same processes sorted by Name.

```
Compare-Object (Get-Process | Sort-Object WorkingSet)
(Get-Process | Sort-Object Name)
```

The user specifies what collection will be referenced. By default, the comparison will only output the differences between the two collections, while indicating which collection has different objects. Here are two options that allow comparisons to show matching objects instead.

`- IncludeEqual` – Instead of only outputting differences, the output includes the objects that are equal

`- ExcludeDifferent` – Excludes the objects that are different between the two collections

- Useful when combined with `-IncludeEqual` when looking for objects that match in two collections

For an example in PowerShell, try the following commands, one at a time:

```
$a = Get-Process
Start-process notepad
$b = Get-Process
Compare-Object $a $b
```

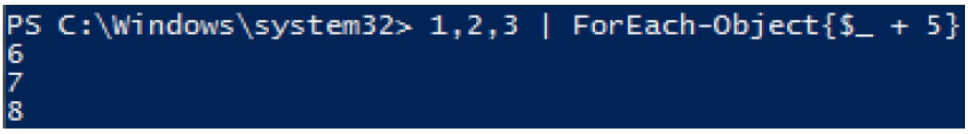
The first command stores the current state of all running processes into the variable `$a`, which identifies what processes are currently running. The next command introduces a new running process, which does not affect the objects stored in `$a`. Next, you take another "snapshot" of running processes, which contains the newly-introduced process, and store this into `$b`. What output do you expect `Compare-Object` to show?

```
PS C:\Windows\system32> $a = Get-Process
PS C:\Windows\system32> Start-Process notepad
PS C:\Windows\system32> $b = Get-Process
PS C:\Windows\system32> Compare-Object $a $b

InputObject                                          SideIndicator
-----------                                          -------------
System.Diagnostics.Process (notepad)                 =>
```

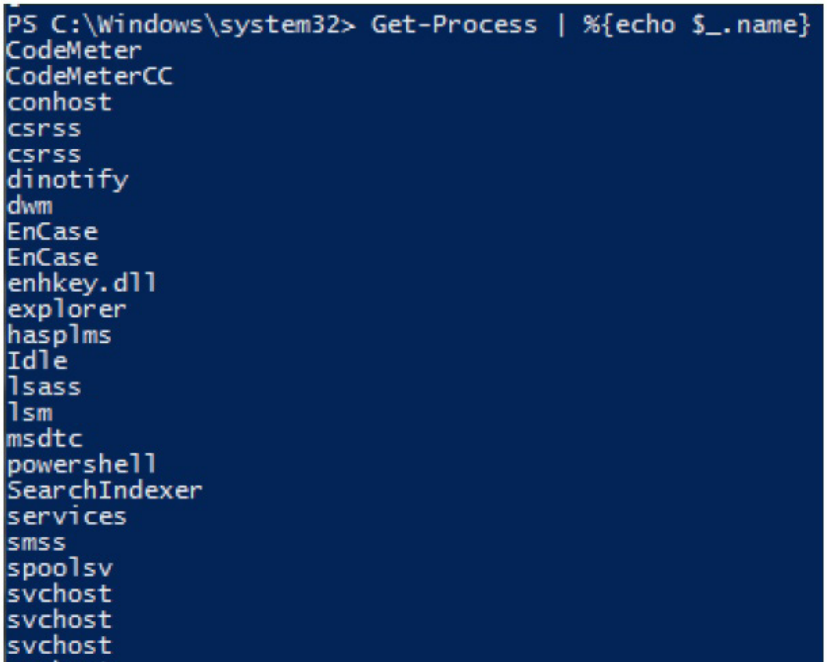
ForEach-Object

This cmdlet allows you to execute a loop that applies a series of commands to each object in a collection. The alias for this command is "%" and can be seen in action in the example below. Note: `ForEach-Object` uses the same special variable, `$_`, that we saw in `Where-Object`.

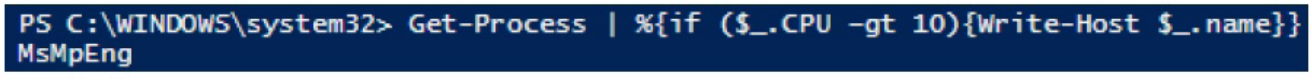
```
1,2,3 | ForEach-Object{$_ + 5}
```




```
Get-Process | %{echo $_.name}
```




```
Get-Process | %{if ($_.CPU –gt 10){Write-Host $_.name}}
```



Exercise 3-7, Syntactical Exercise: ForEach-Object

Get-ChildItem

This cmdlet is similar to the `DIR` command from CMD, but this cmdlet returns a loopable PowerShell object that has properties and methods of its own. Some of the available parameters for Get-ChildItem are:

- `File` – Returns only files, not directories
- `Recurse` – Returns contents of current directory and all subdirectories
- `Force` – Includes system files and hidden files in results
- `Attributes [ATTRIBUTE]` – Only lists files with a specified attribute. If there are multiple attributes then each specified attribute should be separated by a comma. For example:
  `Get-ChildItem -Attributes Hidden, !ReadOnly+Encrypted, !Archive`
- Comma-separate multiple attributes

Get-ChildItem has these properties available in looping cmdlets via the `$_` variable and a dot (`$_.`):

- `Attributes` – Contains attributes of the file (Archive, Hidden, etc.)
- `Extension` – Contains the file extension for the file
- `FullName` – Contains the absolute path of the file

```
PS C:\Users\Administrator> Get-ChildItem


    Directory: C:\Users\Administrator


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-r---        9/13/2016   7:26 AM                Contacts
d-r---        9/13/2016   7:26 AM                Desktop
d-r---       10/27/2016   3:55 PM                Documents
d-r---        9/13/2016   7:26 AM                Downloads
d-r---        9/13/2016   7:26 AM                Favorites
d-r---        9/13/2016   7:26 AM                Links
d-r---        9/13/2016   7:26 AM                Music
d-r---        9/13/2016  10:54 AM                OneDrive
d-r---        9/13/2016   7:26 AM                Pictures
d-r---        9/13/2016   7:26 AM                Saved Games
d-r---        9/13/2016   7:26 AM                Searches
d-r---        9/13/2016   7:26 AM                Videos

PS C:\Users\Administrator> Get-ChildItem -File -Recurse | %{$_.FullName}
C:\Users\Administrator\Favorites\Bing.url
C:\Users\Administrator\Links\Desktop.lnk
C:\Users\Administrator\Links\Downloads.lnk
```

Exercise 3-8, Syntactical Exercise: Get-ChildItem and File Properties

Get-Content

`Get-Content` (employs the aliases `cat` and `gc`) will retrieve the contents of a file, one line at a time. By default, each line will be treated as a different object when used in a pipeline. This means that each line will be accessed separately when using `ForEach-Object` or `Where-Object.`

Some noteworthy parameters are as follows:

`-Delimiter [STRING]` – Specifies delimiter to access each object

`-ReadCount [NUMBER]` – Specifies number of lines per object to pass through a pipe

`-Tail [NUMBER]` – Retrieves specified number of lines beginning at the bottom of the file

`-TotalCount [NUMBER]` – Retrieves specified number of lines beginning at the top of the file

The following property is useful when considering the Get-Content output as a whole:

- `Count` – Returns the number of objects in collection
  - Depending on the file, might output 4 since the file has 4 lines
  - E.g.: `(Get-Content desktop.ini).Count`

```
PS C:\Users\Administrator\Desktop> Get-Content .\desktop.ini

[.ShellClassInfo]
LocalizedResourceName=@%SystemRoot%\system32\shell32.dll,-21769
IconResource=%SystemRoot%\system32\imageres.dll,-183

PS C:\Users\Administrator\Desktop> (Get-Content .\desktop.ini).Count
4

PS C:\Users\Administrator\Desktop> $x = 0; Get-Content .\desktop.ini | %{$x = $x + 1; echo "$x: $_"}
1:
2: [.ShellClassInfo]
3: LocalizedResourceName=@%SystemRoot%\system32\shell32.dll,-21769
4: IconResource=%SystemRoot%\system32\imageres.dll,-183
```

The code above shows the content of desktop.ini. Next, the count of the Get-Content of desktop.ini is retrieved. Finally, each line is shown with its line number to help visualize the count.

Exercise 3-9, Syntactical Exercise: Get-Content

Get-FileHash

This cmdlet creates a hash of a file by applying a hashing algorithm. A noteworthy parameter for `Get-FileHash` is:

- -Algorithm – Specified algorithm to use for file hash
  - MD5
  - SHA1
  - SHA256 – Default algorithm
  - SHA384
  - SHA512

Noteworthy properties of the Get-FileHash output object (typically accessed with $_.):

- -Hash – Contains hash of file
- -Path – Contains path of hashed file

For an example of `Get-FileHash` in PowerShell, try the following:

```
Get-FileHash C:\Users\Public\hashme.txt
```

Exercise 3-10, Practical Exercise: Get-ChildItem and Get-FileHash

Exercise 3-11, Practical Exercise (Advanced): Scripting, Get-FileHash, and Get-Content

Exercise 3-12, Practical Exercise (Advanced): Scripting, Get-FileHash, Get-Content, and Compare-Object

Get-NetUDPEndpoint

Get-`NetUDPEndpoint` will retrieve active UDP endpoint (stream) statistics. Because UDP is a connectionless protocol, we will not use the word connections here. Some noteworthy parameters of `Get-NetUDPEndpoint` are as follows:

- `-LocalPort [PORT]` – Filters results by local port number(s)
  - Comma-separated for multiple ports
- `-LocalAddress [IP]` – Filters results by the address used locally
  - Comma-separated values, again
  - Useful when looking for local ports serving UDP endpoints

- Note: `0.0.0.0` in IPv4, :: in IPv6, states that the host is acting as a server and is listening on all network addresses
- Note: UDP endpoints CAN be opened on specific IP addresses as well

Noteworthy properties of output object (typically accessed with `$_`.):
`-OwningProcess` – Contains the process ID of the process that opened the endpoint

To visualize the commands output, try it yourself: `Get-NetUDPEndpoint`

```
PS C:\WINDOWS\system32> Get-NetUDPEndpoint

LocalAddress                                LocalPort
------------                                ---------
::                                          500
0.0.0.0                                     63198
127.0.0.1                                   62357
127.0.0.1                                   61070
192.168.242.1                               61069
192.168.146.1                               61068
0.0.0.0                                     61067
192.168.56.1                                61066
0.0.0.0                                     54793
0.0.0.0                                     22350
0.0.0.0                                     5355
192.168.242.1                               5353
192.168.146.1                               5353
192.168.56.1                                5353
```

Exercise 3-13, Syntactical Exercise: Get-NetUDPEndpoint

Get-NetTCPConnection

This cmdlet retrieves active TCP connection statistics. Noteworthy parameters: for `Get-NetTCPConnection:`

`-LocalPort [PORT]` – Filters results by local port number(s)
- Comma-separated for multiple ports

`-RemotePort [PORT]` – Filters results by remote port number(s)
- Comma-separated for multiple ports

`-State [STATE]` – Filters results by connection state
- State examples: `Listen, Established, Closed`
- The state `Listen` can identify when the system is hosting a connection and waiting for a message

Noteworthy properties of output object (typically accessed with `$_.`):

`-OwningProcess` – Contains the process ID of the process that opened the endpoint

`-State` – Contains the connection state of the connection

To visualize the command's output, try it yourself: `GetNetTCPConnection`

```
PS C:\WINDOWS\system32> Get-NetTCPConnection

LocalAddress                        LocalPort RemoteAddress                       RemotePort State       AppliedSetting OwningProcess
------------                        --------- -------------                       ---------- -----       -------------- -------------
::                                  50680     ::                                  0          Bound                      5444
::1                                 50680     ::1                                 22350      Established Internet       5444
::                                  49670     ::                                  0          Listen                     912
::                                  49668     ::                                  0          Listen                     932
::                                  49667     ::                                  0          Listen                     1844
::                                  49666     ::                                  0          Listen                     1152
::                                  49665     ::                                  0          Listen                     472
::                                  49664     ::                                  0          Listen                     796
::                                  47001     ::                                  0          Listen                     4
::1                                 22350     ::1                                 50680      Established Internet       2792
::                                  22350     ::                                  0          Listen                     2792
::                                  1947      ::                                  0          Listen                     2772
::                                  445       ::                                  0          Listen                     4
::                                  135       ::                                  0          Listen                     420
```

Exercise 3-14, Syntactical Exercise: Get-NetTCPConnection

FOR

This cmdlet will execute a loop for a specified number of times.

Syntax:

```
for (<Assign Variable>; <Condition>;
     <Increment Variable>){<Statement>;
```

This will run `Statement` until `Condition` is false.

For example:

```
For ($i = 1; $i -le 10; $i = $i + 1) { Write-Host $i }
```

```
PS C:\Windows\system32> For ($i = 1; $i -le 10; $i = $i + 1) { Write-Host $i }
1
2
3
4
5
6
7
8
9
10
```

While

This cmdlet will execute a loop until a specific user-supplied condition is met. If a while loop is not carefully configured, the loop may cycle continuously. Make sure you avoid this.

Syntax:

```
While (<Condition>){<Statement>;}
```

Runs `Statement` until `Condition` is false.

Example:

```
$i = 0; While ($i -ne 10) { $i = $i + 1; Write-Host $i }
```

```
1  $i = 0;
2  While ($i -ne 10){
3      $i = $i + 1;
4      Write-Host $i
5  }
```

```
PS C:\Windows\system32> $i = 0; While ($i -ne 10) { $i = $i + 1; Write-Host $i }
1
2
3
4
5
6
7
8
9
10
```

Exercise 3-15, Practical Exercise: Scripting, While, and Compare-Object

Function

A function is a group of commands that work together to perform a specific task or set of tasks. `Functions` allow us to create a new function with a new name which can be called as if it were a cmdlet.

Syntax:

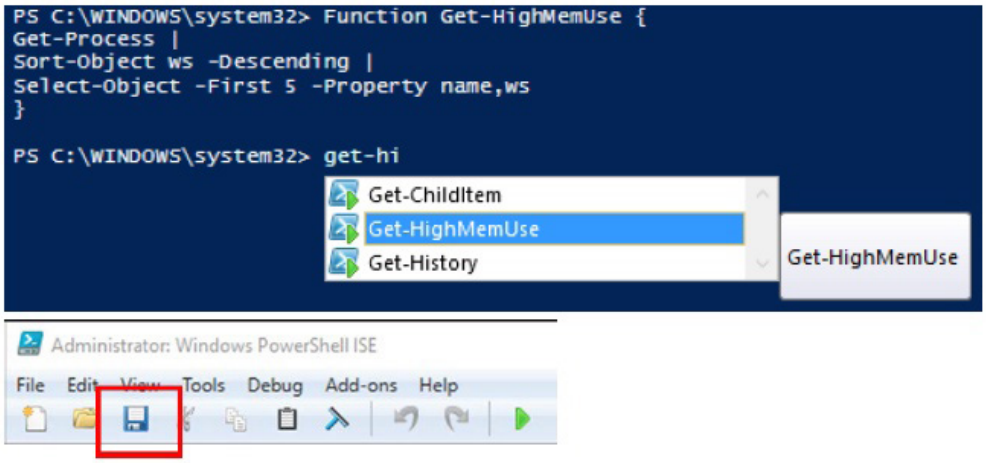
```
function <NAME> { <COMMANDS> }
```

The function will be available while the script or session is active. It will be able to be aliased and auto-completed. Here is a function that outputs running processes using the most RAM.

```
function Get-HighMemUse { Get-Process | Sort-Object ws
-Descending | Select-Object –First 5 –Property name,ws }
```

```
Function Get-HighMemUse {
     Get-Process |
     Sort-Object ws -Descending |
     Select-Object -First 5 -Property name,ws
}
```

Test the function by typing `Get-Hi<TAB>`, where `<TAB>` represents pressing the Tab button on your keyboard. The function will be available for the remainder of the session. Save the script as `get_highmemuse.ps1` in an appropriate location. Remember to use dot sourcing to retain variables and functions defined in scripts during a PowerShell session.



Export-Module

This cmdlet can export functions, variables, and more from a module and may be used only inside of a module. Exporting allows the module member to be used via the PowerShell console and consists of two important parameters:

- `Function <FUNCTIONNAME>` – Exports a function
- `Variable <VARIABLENAME>` – Exports a variable

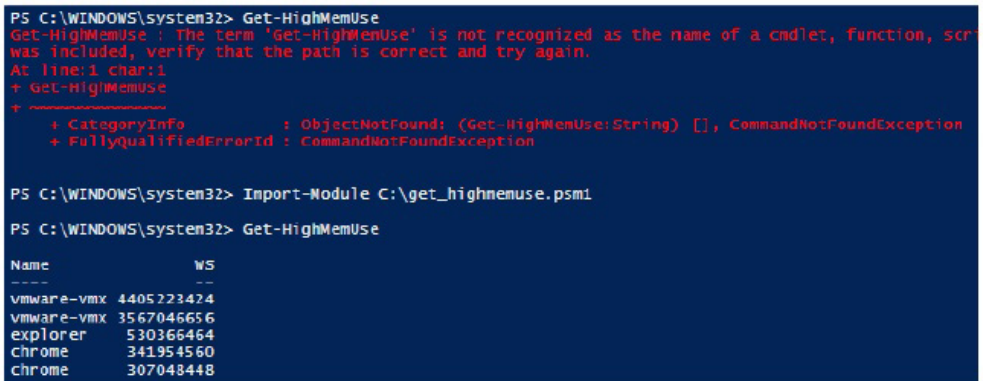
For an example in PowerShell, rename get_highmemuse.ps1 from the last Function example to get_highmemuse.psm1. This seemingly innocuous change actually turns the script into a module, which allows its contents to be exported. Let's do that now by adding the following line to the end of the new PowerShell module: `Export-ModuleMember –Function "Get-*"`. This line will export any function in the module whose name begins with Get-, which is a convenience for the scripter.

Import-Module

This critical cmdlet imports functions, variables, and more from a module. It allows exported module members to be imported. Imported module members can be used throughout the duration of the PowerShell session.

Try the following in a new PowerShell session, using the path to your saved `get_highmemuse.psm1 module:`

```
Import-Module C:\get_highmemuse.psm1
```



Exercise 3-16, Practical Exercise (Advanced): Scripting, Export-ModuleMember, Import-Module, and Function

Regular Expressions

Regular expressions (regex) are used to search for data, such as a string of text, when the structure of the data (string) or a part of the data (string) is known. First, we'll consider the following: `"This is a test" –match "test"`. In this instance, the -match operator is a regex `test`.

`This is a test` is then checked for the characters "`test`", which is considered a substring and is what the regex is searching for.

```
PS C:\WINDOWS\system32> "This is a test" -match "test"
True
```

While the asterisk "*" can be a multi-character wildcard when searching PowerShell cmdlets and "?" can be a single character wildcard, neither of them can be used in regex. The `-match` operator only recognizes regex, not PowerShell wildcards.

```
PS C:\WINDOWS\system32> Get-Command *service*

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Function        Get-NetFirewallServiceFilter                       2.0.0.0    NetSecurity
Function        Set-NetFirewallServiceFilter                       2.0.0.0    NetSecurity
Cmdlet          Get-Service                                        3.1.0.0    Microsoft.PowerShell.Management
Cmdlet          New-Service                                        3.1.0.0    Microsoft.PowerShell.Management
Cmdlet          New-WebServiceProxy                                3.1.0.0    Microsoft.PowerShell.Management
Cmdlet          Restart-Service                                    3.1.0.0    Microsoft.PowerShell.Management
Cmdlet          Resume-Service                                     3.1.0.0    Microsoft.PowerShell.Management
Cmdlet          Set-Service                                        3.1.0.0    Microsoft.PowerShell.Management
Cmdlet          Start-Service                                      3.1.0.0    Microsoft.PowerShell.Management
Cmdlet          Stop-Service                                       3.1.0.0    Microsoft.PowerShell.Management
Cmdlet          Suspend-Service                                    3.1.0.0    Microsoft.PowerShell.Management
Application     AgentService.exe                                   10.0.14... C:\WINDOWS\system32\AgentService.exe
Application     SensorDataService.exe                              10.0.14... C:\WINDOWS\system32\SensorDataService.exe
Application     services.exe                                       10.0.14... C:\WINDOWS\system32\services.exe
Application     services.msc                                       0.0.0.0    C:\WINDOWS\system32\services.msc
Application     TieringEngineService.exe                           10.0.14... C:\WINDOWS\system32\TieringEngineService.exe


PS C:\WINDOWS\system32> Get-Command ???-Service

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Cmdlet          Get-Service                                        3.1.0.0    Microsoft.PowerShell.Management
Cmdlet          New-Service                                        3.1.0.0    Microsoft.PowerShell.Management
Cmdlet          Set-Service                                        3.1.0.0    Microsoft.PowerShell.Management
```

We'll only briefly cover regex in this course, and our discussion will be limited to the operators below.

`-match` – Case-Insensitive ("test" matches "Test") regex matching
- E.g.: `"This is a test" –match "test"`
- This comparison actually searches the entire string `This is a test` for the string `test`

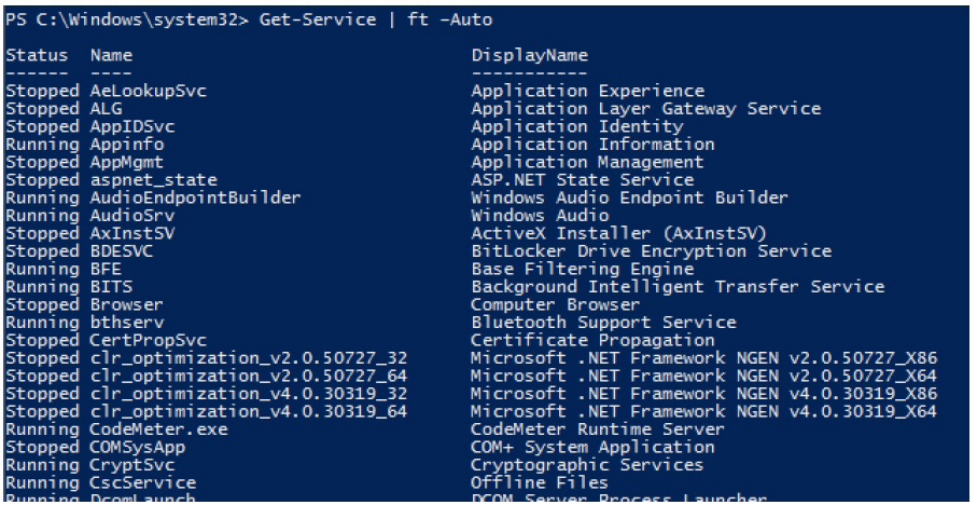
`-cmatch` – Case-Sensitive ("test" does not match "Test") regex matching
- * - represents repeating 0 or more of the preceding character
- A* matches any number of consecutive A's
- . - single character wildcard (represents 1 character)
- .* matches any number of any characters
- \ - escape character (e.g., can be used to search for "." or "*")
- example: `"4 * 3.4" –match "4 \* 3\.4"`

```
PS C:\Windows\system32> "4 * 3.4" -match "4 \* 3\.4"
True
```

Exercise 3-17, Practical Exercise (Advanced): Regular Expressions

Output Formatting

PowerShell can output in more ways than just the columns we've seen thus far. `Format-Table` is the typical format we've seen, and it has an alias of `ft`. This cmdlet can be used with the `-Auto` parameter to have PowerShell adjust column width as it deems fit. As an example, try the following: `Get-Service | ft –Auto.`

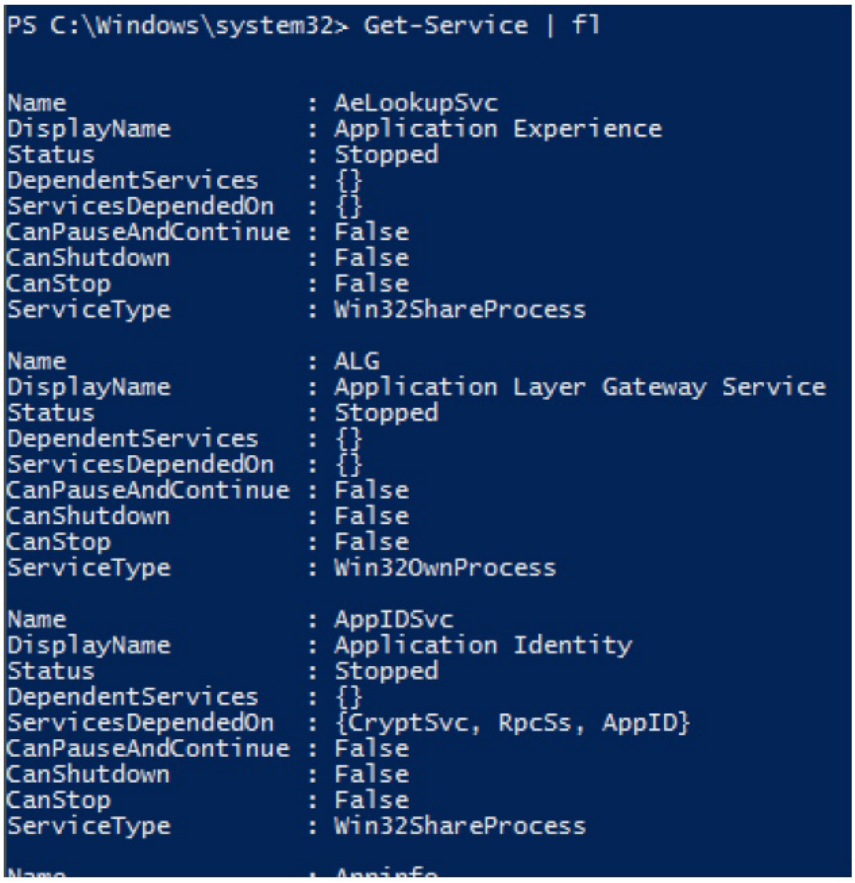


```
PS C:\Windows\system32> Get-Service | ft -Auto

Status  Name                               DisplayName
------  ----                               -----------
Stopped AeLookupSvc                        Application Experience
Stopped ALG                                Application Layer Gateway Service
Stopped AppIDSvc                           Application Identity
Running Appinfo                            Application Information
Stopped AppMgmt                            Application Management
Stopped aspnet_state                       ASP.NET State Service
Running AudioEndpointBuilder               Windows Audio Endpoint Builder
Running AudioSrv                           Windows Audio
Stopped AxInstSV                           ActiveX Installer (AxInstSV)
Stopped BDESVC                             BitLocker Drive Encryption Service
Running BFE                                Base Filtering Engine
Running BITS                               Background Intelligent Transfer Service
Stopped Browser                            Computer Browser
Running bthserv                            Bluetooth Support Service
Stopped CertPropSvc                        Certificate Propagation
Stopped clr_optimization_v2.0.50727_32     Microsoft .NET Framework NGEN v2.0.50727_X86
Stopped clr_optimization_v2.0.50727_64     Microsoft .NET Framework NGEN v2.0.50727_X64
Stopped clr_optimization_v4.0.30319_32     Microsoft .NET Framework NGEN v4.0.30319_X86
Stopped clr_optimization_v4.0.30319_64     Microsoft .NET Framework NGEN v4.0.30319_X64
Running CodeMeter.exe                      CodeMeter Runtime Server
Stopped COMSysApp                          COM+ System Application
Running CryptSvc                           Cryptographic Services
Running CscService                         Offline Files
Running DcomLaunch                         DCOM Server Process Launcher
```

Notice how this gets rid of the ellipses, which would typically cut off longer `Names` and `DisplayNames`.

Format-List

`Format-List` is another formatting option that takes common properties (which are typically output as column names) and lists them vertically for each instance in a collection. The alias for `Format-List` is fl, and can be tested in the following example: `Get-Service | fl`



Notice that `Format-List` outputs more properties per instance that `Format-Table`, but still does not list all available properties.

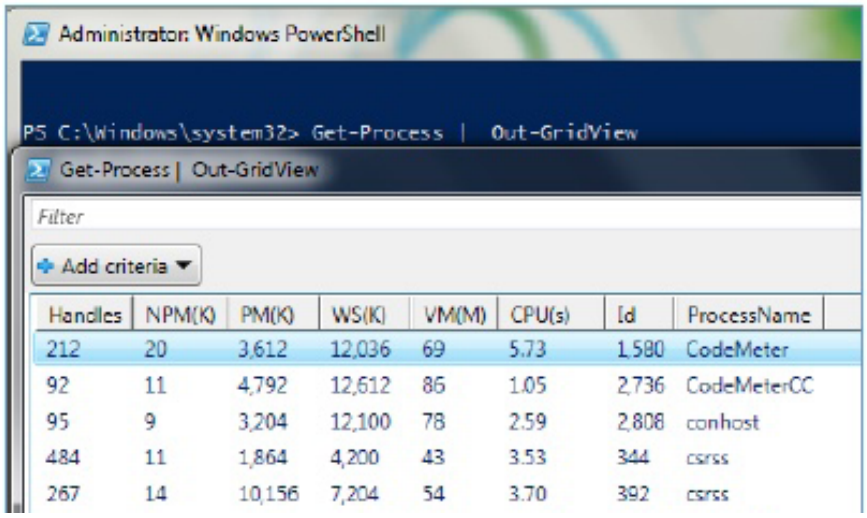
Format-Custom

`Format-Custom`, alias `fc`, will output according to a specified format. This format can be specified with the -View option, but we won't go into custom formatting, as that is a class of its own (see the MSDN page titled "How to Create a Formatting File [.format.ps1xml]" for more information). However, we will use the default Format-Custom view to get a glimpse at the class and instance layout that PowerShell uses to see services by running the following command: `Get-Service | fc.`

Out-GridView

The final format we'll look at is a graphical format generated by PowerShell: `Out-GridView`. This cmdlet will pop up a separate window containing the output in filterable and searchable fields. Try the following as an example:

```
Get-Process | Out-GridView
```



Exercise 3-18 (Advanced): Searching the Microsoft Developer Network (MSDN) to Accomplish Real-World Tasks

MODULE 4

Windows Objects

This module will examine Windows Objects and discuss advanced PowerShell commands and scripts that are used remotely on target machines.

OBJECTIVES *After completing this module, students will be able to:*
Describe the Common Information Model (CIM) use, structure, and components
Describe the relationship between Windows Management Instrumentation (WMI) and CIM
Compose PowerShell commands to access WMI

Lesson 1

CIM Use

This lesson provides a basic history of the Confirmation Information Model (CIM) and describes how it is implemented via PowerShell.

Remote Management

Some common Get- cmdlets feature a -ComputerName and a -Credential option. However, these Get- cmdlets could not retrieve collections from a remote system. To accomplish those tasks, you must use Windows Management Instrumentation (WMI). WMI is a Microsoft (MS) implementation of the Common Information Model (CIM) standard, which is written for Windows.

CIM

CIM was developed by the Distributed Management Task Force (DMTF), which is a joint effort of Microsoft, Intel, VMware, and other major actors in the technology industry to set industry standards. The Task Force created CIM to be a conceptual model that standardizes the management of computing devices. Common elements that need to be managed (e.g., services) are represented by objects (e.g., Win32_Service). Objects are constantly and clearly presented to the managing applications. These include object classes, properties, methods, etc, which are described by the Managed Object Format (MOF), a standard language that defines CIM-managed elements. A distinction between CIM and WMI must be made because CIM is a joint standard for non-Windows devices and software (devices/software made by non-MS DMTF members).

However, CIM is an information model only and it can be implemented in any application or database. Also, CIM can be seen as an extension of the other management standards, such as Simple Network Management Protocol (SNMP) and Desktop Management Interface (DMI). The term CIM does not include CIM implementations, access methods, communication protocols, or other CIM-related specifications. CIM employs Object-Oriented Modeling, which means elements to be managed are represented by objects; this approach is similar to but different than Object-Oriented Programming (OOP).

Lesson 2

CIM Structure and Components

This lesson will discuss the various components of CIM and how to use them in PowerShell. It will also prepare the student for WMI implementation.

CIM

CIM is made up of multiple structures and components. Examine the list below:

Component	Can be considered
Classes	Types of things
Subclasses	Subtypes of things
Instances	Things
Properties	Attributes
Methods	Operations

Classes

Classes refers to "Types of Things." These act as a blueprint, or template of an object. An instance of an object, made from the blueprint, is a practical form of the class. The best way to describe this is to consider a real world object, such as a car. The idea of a car could be considered a class. Cars have properties such as color, model, and transmission types. Cars are also able to perform specific tasks, such as drive, reverse, park, or honk its horn.

Specifically, a 2016 Honda Civic could be considered an instance ("Things") of a car class. A 2016 Honda Civic might have the color "blue", model "Civic", transmission type "manual." The Civic would be able to drive, reverse, park and honk its horn. In other words, the instance of a car would inherit methods and properties of the car class, but only as they apply to the instance.

Properties

Properties are "attributes." They hold specific information describing class instances. Examples of properties for the class Win32_Process are "Name" and "ProcessId."

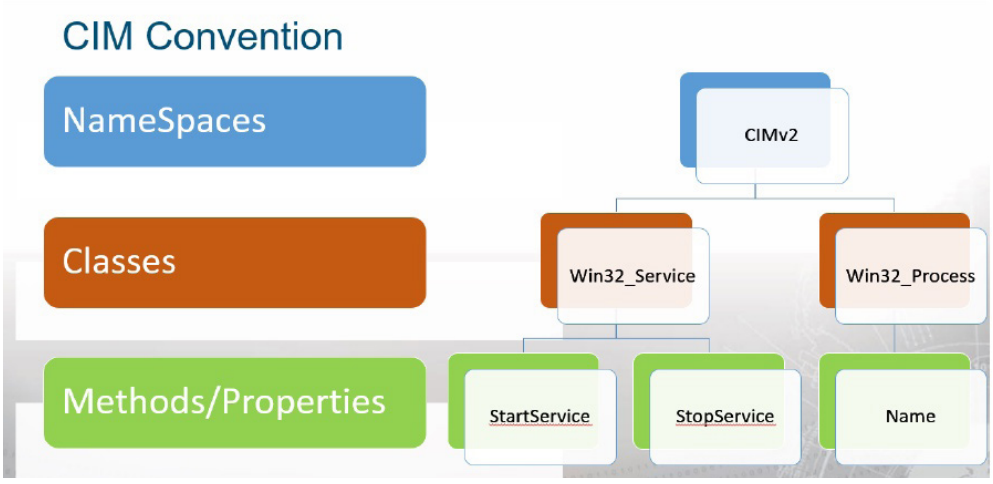
Methods

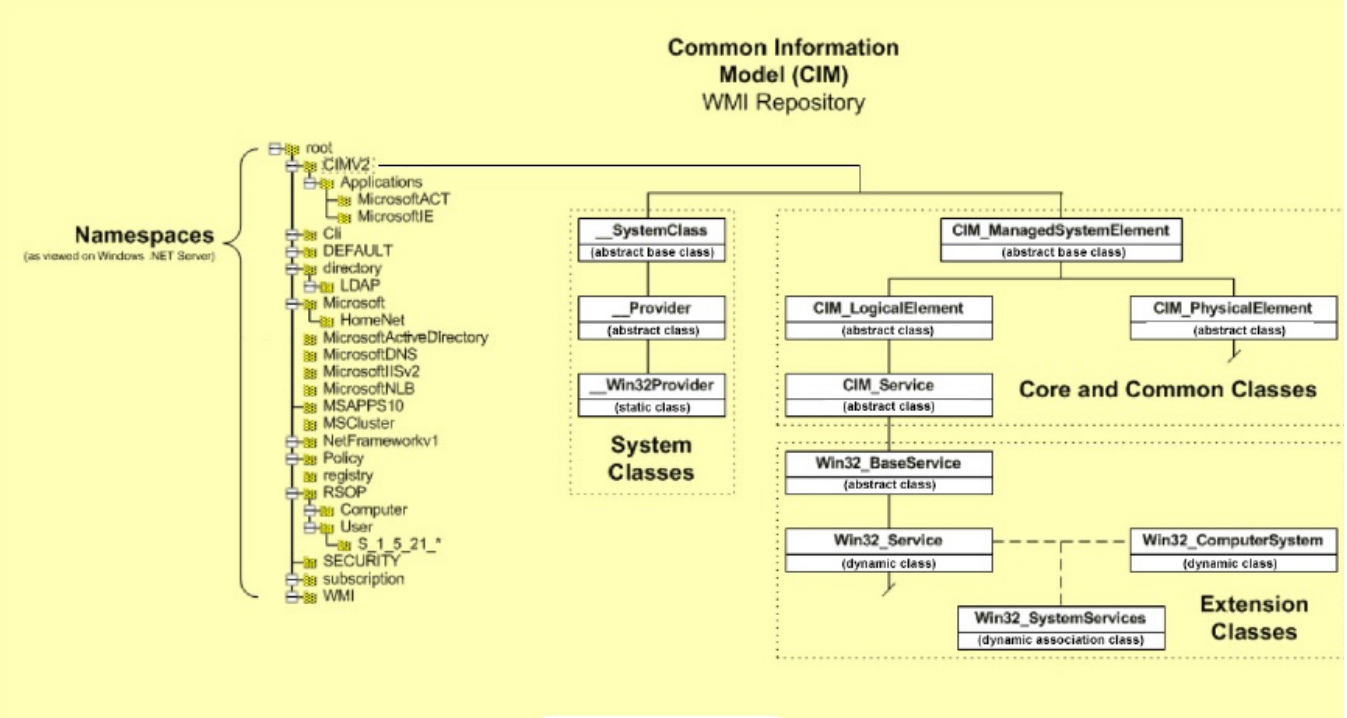
Methods are "operations" that can modify/access class instance properties. Examples of methods for the class Win32_Process are "Terminate" and "SetPriority."

CIM Convention

Conventions are a vital part of the coding process. A schema is a structural model for classes. "SchemaName_ClassName" is the class naming convention that you will often see in WMI (e.g., Win32_Process). Schemas are used for administrative tasks, such as building and formatting classes. Although we will not deal with schemas in this course, it is helpful to know why some classes will begin with "Win32" and others "MSFT" and others yet "CIM".

Namespaces are also used by CIM to structure code. A namespace separates a group of classes. Why would anyone put some classes in an entirely separate namespace? For security, to separate access permissions for certain classes, or perhaps for naming purposes, as some class names might be generic enough to refer to several classes. Functionality could be yet another reason to separate classes, as one might desire to find an obscure class to manage a certain element, or perhaps one would like to see all classes available to manage a certain element. The default namespace is "root/CIMv2" in WMI.





WMI Location

Some readers may be wondering, "Where is WMI located?" WMI is accessed by live queries in this course. Certain class definitions, namespace definitions, etc., can be stored in MOF files on the file system, but queries will pull instances, live instantiations of classes. The heart of WMI is located in C:\Windows\System32\wbem. Although, again, accessing WMI is via live queries, not necessarily reading from files. Visit https://technet.microsoft.com/en-us/library/cc180440.aspx for more information, if interested.

Lesson 3

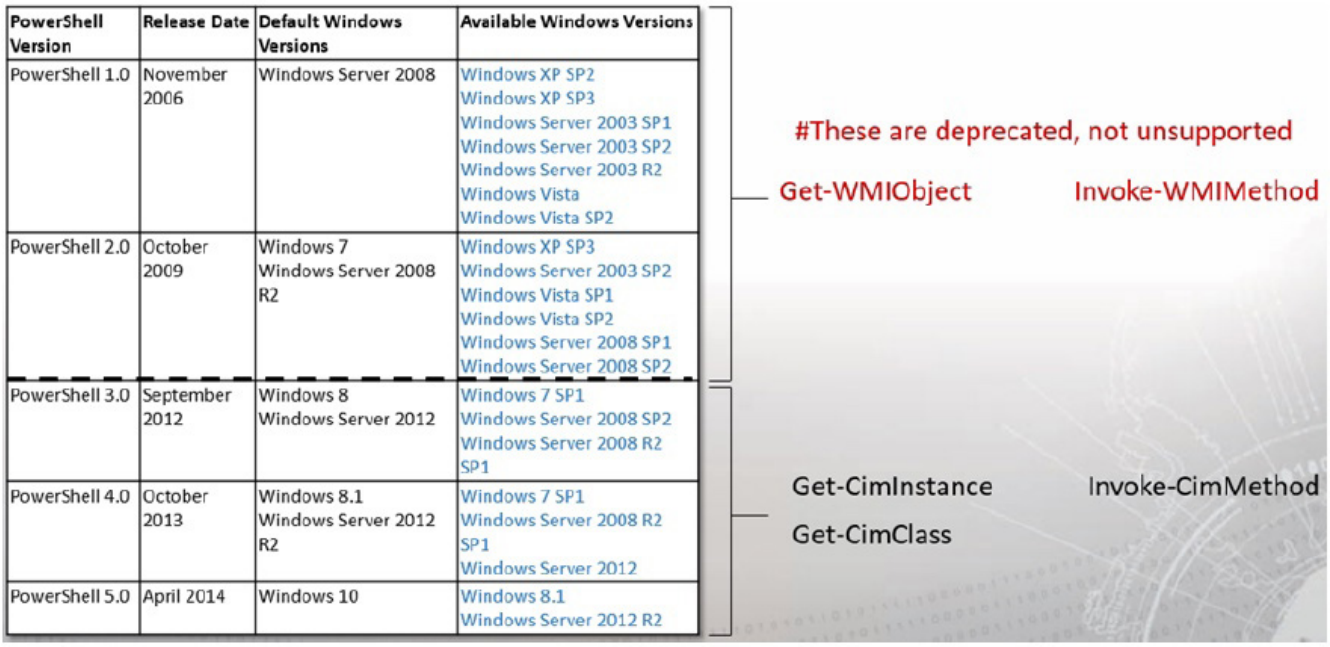
Accessing WMI via PowerShell

This lesson will show students how to use WMI via PowerShell. Check the version of PowerShell you are using, as WMI functionality changes depending on the PowerShell version that is being used.

WMI and PowerShell

There are numerous versions of PowerShell available for use. The diagram below lists the PowerShell versions and the PowerShell commands that access WMI. The functionality of the cmdlets in red have been replaced and updated by the cmdlets in black.

PowerShell Version	Release Date	Default Windows Versions	Available Windows Versions
PowerShell 1.0	November 2006	Windows Server 2008	Windows XP SP2 Windows XP SP3 Windows Server 2003 SP1 Windows Server 2003 SP2 Windows Server 2003 R2 Windows Vista Windows Vista SP2
PowerShell 2.0	October 2009	Windows 7 Windows Server 2008 R2	Windows XP SP3 Windows Server 2003 SP2 Windows Vista SP1 Windows Vista SP2 Windows Server 2008 SP1 Windows Server 2008 SP2
PowerShell 3.0	September 2012	Windows 8 Windows Server 2012	Windows 7 SP1 Windows Server 2008 SP2 Windows Server 2008 R2 SP1
PowerShell 4.0	October 2013	Windows 8.1 Windows Server 2012 R2	Windows 7 SP1 Windows Server 2008 R2 SP1 Windows Server 2012
PowerShell 5.0	April 2014	Windows 10	Windows 8.1 Windows Server 2012 R2



PowerShell WMI

There are a few differences between PowerShell versions pre-3.0 and post-3.0 (which includes version 3.0); these will affect our discussion of cmdlets that interact with WMI. In the next section, cmdlets will be introduced that will work on PowerShell versions 3.0 through 5.1. This will be expanded upon in greater detail throughout the module, but be aware of these differences as we progress.

Get-WmiObject (Available in PowerShell 1.0 and 2.0)

Function: Retrieves instances of a specified WMI class
Alias: `gwmi`
Syntax: `gwmi <CLASSNAME>`
Try: `gwmi Win32_OperatingSystem -ComputerName X.X.X.X -Credential $cred`
`X.X.X.X` is the IP (or hostname) of the computer

```
PS C:\WINDOWS\system32> gwmi Win32_OperatingSystem -ComputerName              -Credential $cred


SystemDirectory : C:\Windows\system32
Organization    :
BuildNumber     : 7601
RegisteredUser  : Windows User
SerialNumber    : 00371-705-1482436-06904
Version         : 6.1.7601
```

The `gwmi` cmdlet can also be used to enumerate all classes in a specified namespace. Options:

- `-Namespace <NAMESPACENAME>` - Specifies the namespace that `gwmi` should search for the specified class (if no namespace is specified, `Root\Cimv2` is implied)
- `-Recurse` - Searches specified namespace and all the children of the namespace for a specified class
- `-List` – Lists specified WMI classes within specified namespace

Try: `gwmi –Recurse –List -ComputerName X.X.X.X -Credential $cred`

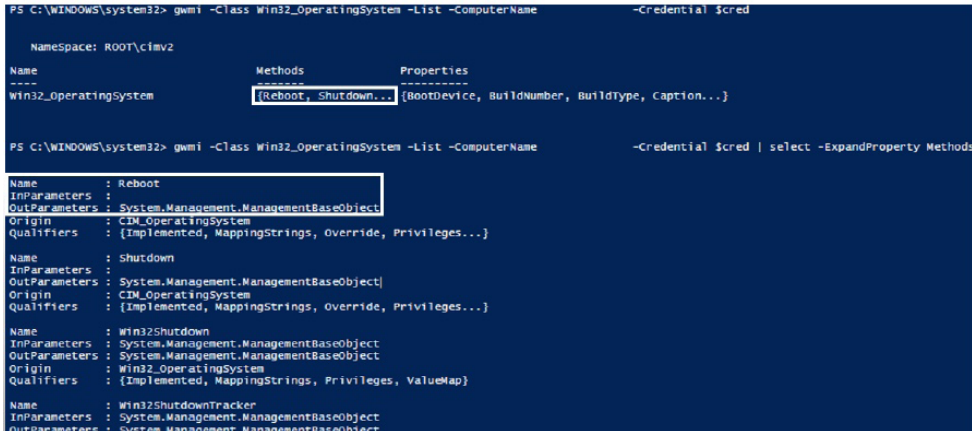
```
PS C:\WINDOWS\system32> gwmi -Recurse -List -ComputerName    . . .    -Credential $cred


   NameSpace: ROOT\CIMV2

Name                                Methods              Properties
----                                -------              ----------
__SystemClass                       {}                   {}
__thisNAMESPACE                     {}                   {SECURITY_DESCRIPTOR}
__NAMESPACE                         {}                   {Name}
__Provider                          {}                   {Name}
__Win32Provider                     {}                   {ClientLoadableCLSID, CLSID, Concurrency, DefaultMachineName...}
__ProviderRegistration              {}                   {provider}
__EventProviderRegistration         {}                   {EventQueryList, provider}
__ObjectProviderRegistration        {}                   {InteractionType, provider, QuerySupportLevels, SupportsBatching...}
__ClassProviderRegistration         {}                   {CacheRefreshInterval, InteractionType, PerUserSchema, provider...}
__InstanceProviderRegistration      {}                   {InteractionType, provider, QuerySupportLevels, SupportsBatching...}
__MethodProviderRegistration        {}                   {provider}
__PropertyProviderRegistration      {}                   {provider, SupportsGet, SupportsPut}
__EventConsumerProviderRegistration {}                   {ConsumerClassNames, provider}
__IndicationRelated                 {}                   {}
__EventFilter                       {}                   {CreatorSID, EventAccess, EventNamespace, Name...}
__EventConsumer                     {}                   {CreatorSID, MachineName, MaximumQueueSize}
__FilterToConsumerBinding           {}                   {Consumer, CreatorSID, DeliverSynchronously, DeliveryQoS...}
__AggregateEvent                    {}                   {NumberOfEvents, Representative}
__TimerNextFiring                   {}                   {NextEvent64BitTime, TimerId}
__Event                             {}                   {SECURITY_DESCRIPTOR, TIME_CREATED}
__ExtrinsicEvent                    {}                   {SECURITY_DESCRIPTOR, TIME_CREATED}
Win32_DeviceChangeEvent             {}                   {EventType, SECURITY_DESCRIPTOR, TIME_CREATED}
Win32_SystemConfigurationChangeE... {}                   {EventType, SECURITY_DESCRIPTOR, TIME_CREATED}
Win32_VolumeChangeEvent             {}                   {DriveName, EventType, SECURITY_DESCRIPTOR, TIME_CREATED}
MSFT_WMI_GenericNonCOMEvent         {}                   {ProcessId, PropertyNames, PropertyValues, ProviderName...}
MSFT_NCProvEvent                    {}                   {Namespace, ProviderName, Result, SECURITY_DESCRIPTOR...}
```

NOTE: `Get-WMIObject` has been superseded by `Get-CimInstance` and is not available in PowerShell version 6.0

Before we continue with WMI, let's revisit some applications of the `gwmi` cmdlet and take a fuller advantage of its capabilities.

Try: `gwmi Win32_OperatingSystem -List -ComputerName X.X.X.X -Credential $cred`

Try: `gwmi Win32_OperatingSystem -List -ComputerName X.X.X.X -Credential $cred | select -ExpandProperty Methods`



Exercise 4-1, Syntactical Exercise: Get-WmiObject and Select-Object

FILTER

The `gwmi` cmdlet is also able to filter on its own based on properties of instances.

Filters: `-Filter <FILTER>`
Allows retrieval of specific WMI instances that match a filter condition

Example 1:
`gwmi Win32_Process -Filter "Handle=9280"`

- Filters by a numeric value
- Filters by "`Handle`"
- In order for this example to work on your system, you'll need to first run `gwmi Win32_Process` to find an example handle to filter by
- Perform `gwmi Win32_Process` first to find a random handle to try

Example 2:
`gwmi Win32_Service -Filter "Name='WinRM'"`

- Filters by a string value using single quotes (apostrophes)
- Filters by "`Name`"

```
PS C:\WINDOWS\system32> gwmi Win32_Process -Filter "Handle=9280"


__GENUS                 : 2
__CLASS                 : Win32_Process
__SUPERCLASS            : CIM_Process
__DYNASTY               : CIM_ManagedSystemElement
__RELPATH               : Win32_Process.Handle="9280"
__PROPERTY_COUNT        : 45
__DERIVATION            : {CIM_Process, CIM_LogicalElement, CIM_ManagedSystemElement}
__SERVER                :
__NAMESPACE             : root\cimv2
__PATH                  : \\              \root\cimv2:Win32_Process.Handle="9280"
Caption                 : chrome.exe
PS C:\WINDOWS\system32> gwmi Win32_service -Filter "Name='WinRM'"


ExitCode  : 1077
Name      : WinRM
ProcessId : 0
StartMode : Manual
State     : Stopped
Status    : OK
```

Exercise 4-2, Syntactical Exercise: Get-WmiObject -Filter

Invoke-WmiMethod (Available in PowerShell 1.0 and 2.0)

`Invoke-WmiMethod` – Calls a method associated with a WMI class or instance

Alias: `iwmi`
Syntax: `iwmi –Name <METHODNAME>`
Try: `gwmi Win32_Service -Filter "Name='W32Time'" | iwmi -Name StopService`

- Afterward, start it back up. It's the Windows Time service.
- Note that an instance was required to be retrieved before invoking the method.

```
PS C:\WINDOWS\system32> gwmi Win32_Service -Filter "Name='W32Time'" | iwmi -Name StopService


__GENUS          : 2
__CLASS          : __PARAMETERS
__SUPERCLASS     :
__DYNASTY        : __PARAMETERS
__RELPATH        :
__PROPERTY_COUNT : 1
__DERIVATION     : {}
__SERVER         :
__NAMESPACE      :
__PATH           :
ReturnValue      : 5
PSComputerName   :
```

Exercise 4-3, Syntactical Exercise: Invoke-WmiMethod

PowerShell WMI (Post-PS 3.0)

In this section we will cover WMI commands that only work in PowerShell 3.0 or later.

Get-CimInstance

Function: Retrieves instances of a specified CIM class. As a note, while the functionality of `Get-CimInstance` is similar to `gwmi`, it does NOT retain the ability to search classes with `-List` and wildcards. However, another cmdlet does, and we'll discuss this cmdlet soon enough.
Alias: `gcim`
Try: `gcim Win32_Process -KeyOnly`

This example retrieves the key property (the one that matters to WMI)

Try: `gcim -Namespace <TAB>  #Hit "Tab" after space to see available namespaces`

- Only works on localhost (not on remote systems)

```
PS C:\WINDOWS\system32> gcim Win32_Process -KeyOnly


ProcessName               :
Handles                   :
VM                        :
WS                        :
Path                      :
Caption                   :
Description               :
InstallDate               :
Name                      :
Status                    :
CreationClassName         :
CreationDate              :
CSCreationClassName       :
CSName                    :
ExecutionState            :
Handle                    : 0
KernelModeTime            :
OSCreationClassName       :
OSName                    :
Priority                  :
TerminationDate           :
UserModeTime              :
WorkingSetSize            :
CommandLine               :
ExecutablePath            :
HandleCount               :
MaximumWorkingSetSize     :
MinimumWorkingSetSize     :
OtherOperationCount       :
OtherTransferCount        :
PageFaults                :
PageFileUsage             :
```

Note that only the `Handle` property was populated. This is because the key property for `Win32_Process` is the process `Handle`, which is the same as its `ProcessId.` Also note that the key property of a class is the best property to filter by when trying to retrieve a specific instance of that class, since that key property will be unique for each instance of a class.

Exercise 4-4, Syntactical Exercise: Get-CimInstance -Filter

Invoke-CimMethod

Function: Invokes instance or static method of a CIM class

Alias: `icim`
Try: `gcim Win32_Process -Filter "name='notepad.exe'" | icim -Name terminate`

Determine what the above command will do first. You will need to open an instance of Notepad before executing it.

```
PS C:\WINDOWS\system32> gcim Win32_Process -Filter "name='notepad.exe'" | icim -Name terminate

ReturnValue PSComputerName
----------- --------------
          0
```

Exercise 4-5, Syntactical Exercise: Invoke-CimMethod

Get-CimClass

Function: Retrieves CIM classes of a specified namespace. This cmdlet replaces `gwmi` (which is deprecated).

Options:

- `-ClassName <CLASSNAME>` - Specifies a class to retrieve
- `-Namespace <NAMESPACENAME>` - Specifies a namespace from which to retrieve records
- `-MethodName <METHODNAME>` - Specifies a method to retrieve
- `-PropertyName <PROPERTYNAME>` - Specifies a property to retrieve

Try:

```
gcls          #Enumerates all classes in root\CIMv2
```

```
PS C:\WINDOWS\system32> gcls


   NameSpace: ROOT/CIMV2

CimClassName                        CimClassMethods      CimClassProperties
------------                        ---------------      ------------------
__SystemClass                       {}                   {}
__thisNAMESPACE                     {}                   {SECURITY_DESCRIPTOR}
__Provider                          {}                   {Name}
__Win32Provider                     {}                   {Name, ClientLoadableCLSID, CLSID, Concurrency...}
__ProviderRegistration              {}                   {provider}
__EventProviderRegistration         {}                   {provider, EventQueryList}
__ObjectProviderRegistration        {}                   {provider, InteractionType, QuerySupportLevels, SupportsBatching...}
__ClassProviderRegistration         {}                   {provider, InteractionType, QuerySupportLevels, SupportsBatching...}
__InstanceProviderRegistration      {}                   {provider, InteractionType, QuerySupportLevels, SupportsBatching...}
__MethodProviderRegistration        {}                   {provider}
__PropertyProviderRegistration      {}                   {provider, SupportsGet, SupportsPut}
__EventConsumerProviderRegistration {}                   {provider, ConsumerClassNames}
__NAMESPACE                         {}                   {Name}
__IndicationRelated                 {}                   {}
__EventFilter                       {}                   {CreatorSID, EventAccess, EventNamespace, Name...}
__EventConsumer                     {}                   {CreatorSID, MachineName, MaximumQueueSize}
EventViewerConsumer                 {}                   {CreatorSID, MachineName, MaximumQueueSize, Description...}
__FilterToConsumerBinding           {}                   {Consumer, CreatorSID, DeliverSynchronously, DeliveryQoS...}
__AggregateEvent                    {}                   {NumberOfEvents, Representative}
__TimerNextFiring                   {}                   {NextEvent64BitTime, TimerId}
__Event                             {}                   {SECURITY_DESCRIPTOR, TIME_CREATED}
__ExtrinsicEvent                    {}                   {SECURITY_DESCRIPTOR, TIME_CREATED}
Win32_DeviceChangeEvent             {}                   {SECURITY_DESCRIPTOR, TIME_CREATED, EventType}
Win32_SystemConfigurationChangeE... {}                   {SECURITY_DESCRIPTOR, TIME_CREATED, EventType}
Win32_VolumeChangeEvent             {}                   {SECURITY_DESCRIPTOR, TIME_CREATED, EventType, DriveName}
MSFT_WMI_GenericNonCOMEvent         {}                   {SECURITY_DESCRIPTOR, TIME_CREATED, ProcessId, PropertyNames...}
```

Try:

`gcls *process*` #Retrieves all classes with "process" in the name

```
PS C:\WINDOWS\System32> gcls *process*


   NameSpace: ROOT/CIMV2

CimClassName                        CimClassMethods      CimClassProperties
------------                        ---------------      ------------------
Win32_ProcessTrace                  {}                   {SECURITY_DESCRIPTOR, TIME_CREATED, ParentProcessID, ProcessID...}
Win32_ProcessStartTrace             {}                   {SECURITY_DESCRIPTOR, TIME_CREATED, ParentProcessID, ProcessID...}
Win32_ProcessStopTrace              {}                   {SECURITY_DESCRIPTOR, TIME_CREATED, ParentProcessID, ProcessID...}
CIM_Process                         {}                   {Caption, Description, InstallDate, Name...}
Win32_Process                       {Create, Terminat... {Caption, Description, InstallDate, Name...}
CIM_Processor                       {SetPowerState, R... {Caption, Description, InstallDate, Name...}
Win32_Processor                     {SetPowerState, R... {Caption, Description, InstallDate, Name...}
CIM_AssociatedProcessorMemory       {}                   {Antecedent, Dependent, BusSpeed}
Win32_AssociatedProcessorMemory     {}                   {Antecedent, Dependent, BusSpeed}
CIM_ProcessExecutable               {}                   {Antecedent, Dependent, BaseAddress, GlobalProcessCount...}
Win32_SessionProcess                {}                   {Antecedent, Dependent}
Win32_ComputerSystemProcessor       {}                   {GroupComponent, PartComponent}
Win32_SystemProcesses               {}                   {GroupComponent, PartComponent}
CIM_ProcessThread                   {}                   {GroupComponent, PartComponent}
CIM_OSProcess                       {}                   {GroupComponent, PartComponent}
Win32_NamedJobObjectProcess         {}                   {Collection, Member}
Win32_ProcessStartup                {}                   {CreateFlags, EnvironmentVariables, ErrorMode, FillAttribute...}
Win32_PerfFormattedData_Counters... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfRawData_Counters_PerPr... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfFormattedData_Counters... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfRawData_Counters_PerPr... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfFormattedData_Counters... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfRawData_Counters_Proce... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfFormattedData_Lsa_Secu... {}                   {Caption, Description, Name, Frequency_Object...}
```

Consider what this command would retrieve:

`gcls Win32* -MethodName Crea*`

```
PS C:\WINDOWS\system32> gcls Win32* -MethodName Crea*


   NameSpace: ROOT/cimv2

CimClassName                        CimClassMethods      CimClassProperties
------------                        ---------------      ------------------
Win32_Process                       {Create, Terminat... {Caption, Description, InstallDate, Name...}
Win32_BaseService                   {StartService, St... {Caption, Description, InstallDate, Name...}
Win32_Service                       {StartService, St... {Caption, Description, InstallDate, Name...}
Win32_TerminalService               {StartService, St... {Caption, Description, InstallDate, Name...}
Win32_SystemDriver                  {StartService, St... {Caption, Description, InstallDate, Name...}
Win32_ScheduledJob                  {Create, Delete}     {Caption, Description, InstallDate, Name...}
Win32_DfsNode                       {Create}             {Caption, Description, InstallDate, Name...}
Win32_Share                         {Create, SetShare... {Caption, Description, InstallDate, Name...}
Win32_ClusterShare                  {Create, SetShare... {Caption, Description, InstallDate, Name...}
Win32_ShadowCopy                    {Create, Revert}     {Caption, Description, InstallDate, Name...}
Win32_ShadowStorage                 {Create}             {AllocatedSpace, DiffVolume, MaxSpace, UsedSpace...}
```

Why CIM?

It is important to keep in mind that WMI is just a CIM server, a CIM implementation (CIM-plementation). Conflicts within the operating system management standards can break automation, and having a script to perform the same management across multiple operating systems makes everything easier. For responders, CIM is important to know and understand because they can use CIM to manage non-Windows devices with PowerShell.

The Difference CIM-plified

The `gwmi` and `iwmi` cmdlets, along with other deprecated PowerShell WMI cmdlets, use Distributed Component Object Model (DCOM), a Microsoft proprietary communication technology, when accessing remote computers.

In contrast, the `gcim, gcls, icim,` and other PowerShell cmdlets that implement CIM employ the Windows Remote Management (WinRM) protocol.

WinRM is Windows' implementation of the Web Services-Management (WS-Man) protocol; it still accesses WMI and CIM, but has changes in the remote access protocol. WS-Man is based on a simpler, open standard developed by the DMTF and is a firewall-friendly way to access CIM servers. With WinRM, the return objects of CIM cmdlets have more standard PowerShell properties and methods. However, a few properties and methods that WMI cmdlets are programmed to inherit from the actual WMI classes are lost. So there's a bit of give and take, but the overall direction is facing forward.

The Reason for Retention

If we were to automate mundane administrative tasks, forensics tasks, etc., we would want the automation to work on all relevant devices, regardless of their OS or version. However, the device's operating system will impact your script. Therefore, if you want your script to work with non-Windows devices, even though you lose support for some older versions of Windows, CIM cmdlets are the way to go. On the other hand, if supporting older Windows versions, such as Windows 7, is important, WMI cmdlets should be used in your tasks.

WMI Study: Win32_Process

List available instance methods/properties:

```
gwmi –Class Win32_Process | gm
```

```
PS C:\WINDOWS\system32> gwmi -Class Win32_Process | gm


   TypeName: System.Management.ManagementObject#root\cimv2\Win32_Process

Name                      MemberType     Definition
----                      ----------     ----------
Handles                   AliasProperty  Handles = Handlecount
ProcessName               AliasProperty  ProcessName = Name
PSComputerName            AliasProperty  PSComputerName = __SERVER
VM                        AliasProperty  VM = VirtualSize
WS                        AliasProperty  WS = WorkingSetSize
AttachDebugger            Method         System.Management.ManagementBaseObject
GetAvailableVirtualSize   Method         System.Management.ManagementBaseObject
GetOwner                  Method         System.Management.ManagementBaseObject
GetOwnerSid               Method         System.Management.ManagementBaseObject
SetPriority               Method         System.Management.ManagementBaseObject
Terminate                 Method         System.Management.ManagementBaseObject
Caption                   Property       string Caption {get;set;}
CommandLine               Property       string CommandLine {get;set;}
CreationClassName         Property       string CreationClassName {get;set;}
CreationDate              Property       string CreationDate {get;set;}
CSCreationClassName       Property       string CSCreationClassName {get;set;}
CSName                    Property       string CSName {get;set;}
Description               Property       string Description {get;set;}
ExecutablePath            Property       string ExecutablePath {get;set;}
ExecutionState            Property       uint16 ExecutionState {get;set;}
Handle                    Property       string Handle {get;set;}
```

List available class methods/properties:

```
gwmi –Class Win32_Process –List | gm
```

```
PS C:\WINDOWS\system32> gwmi -Class Win32_Process -List | gm


   TypeName: System.Management.ManagementClass#ROOT\cimv2\Win32_Process

Name                MemberType    Definition
----                ----------    ----------
Name                AliasProperty Name = __Class
PSComputerName      AliasProperty PSComputerName = __SERVER
Create              Method        System.Management.ManagementBaseObject
__CLASS             Property      string __CLASS {get;set;}
__DERIVATION        Property      string[] __DERIVATION {get;set;}
__DYNASTY           Property      string __DYNASTY {get;set;}
__GENUS             Property      int __GENUS {get;set;}
__NAMESPACE         Property      string __NAMESPACE {get;set;}
__PATH              Property      string __PATH {get;set;}
__PROPERTY_COUNT    Property      int __PROPERTY_COUNT {get;set;}
__RELPATH           Property      string __RELPATH {get;set;}
__SERVER            Property      string __SERVER {get;set;}
__SUPERCLASS        Property      string __SUPERCLASS {get;set;}
ConvertFromDateTime ScriptMethod  System.Object ConvertFromDateTime();
ConvertToDateTime   ScriptMethod  System.Object ConvertToDateTime();
```

Multiple methods to start or terminate a process:

```
iwmi -Class Win32_Process -Name Create -ArgumentList
notepad.exe

(gwmi win32_process -Filter "name='notepad.exe'").
terminate()

gwmi -Class win32_process | Where-Object {$_.name -eq
"notepad.exe"} | %{$_.terminate()}

gwmi Win32_Process -Filter "name='notepad.exe'" | iwmi
-Name terminate

gwmi -Class win32_process -Filter "name='notepad.exe'" |

%{$_.terminate()}

gwmi -Class win32_process | %{if ($_.name -eq

"notepad.exe"){$_.terminate()}}
```

Exercise 4-6, Syntactical Exercise: Get-WmiObject or Get-CimInstance

Exercise 4-7, Syntactical Exercise: Get-WmiObject or Get-CimInstance

Exercise 4-8, Practical Exercise: USB Detection Using WMI

Exercise 4-9, Practical Exercise: Starting Processes With WMI

MODULE 5

WMIC

This module will continue to examine Windows objects and discuss an additional tool for accessing Windows objects, both locally and remotely, on machines that do not have PowerShell installed.

OBJECTIVES *After completing this module, students will be able to:*
Show the WMIC command structure
Employ WMIC commands to access WMI
Carry out WMIC commands for remote administration
Implement WMIC commands to query and call methods on CIM servers through WMI

Lesson 1

WMIC Command Structures

This lesson will discuss WMIC, its functions, and how its commands are structured. There are important differences between WMIC and PowerShell commands, so pay attention to them.

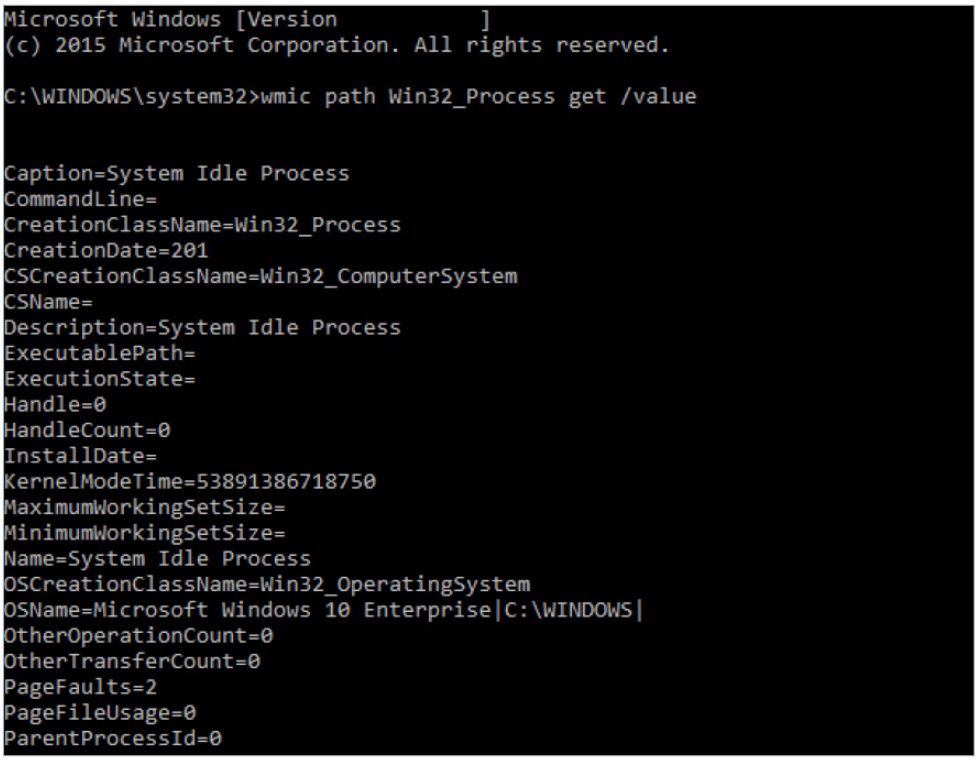
WMIC Introduction

WMIC is the WMI command line utility that can access WMI in pre-PowerShell machines but it can still access WMI to this day. We learn about WMIC because it can be used to access WMI on operating systems made earlier than Windows XP SP2. Furthermore, our understanding of WMI can be enhanced by learning WMIC, and this understanding can be applied to PowerShell scripts via the WMI/CIM cmdlets.

```
WMIC Syntax
wmic [class|path] <CLASS/PATH NAME> <VERB> <OPTIONAL
MODIFIERS>
```

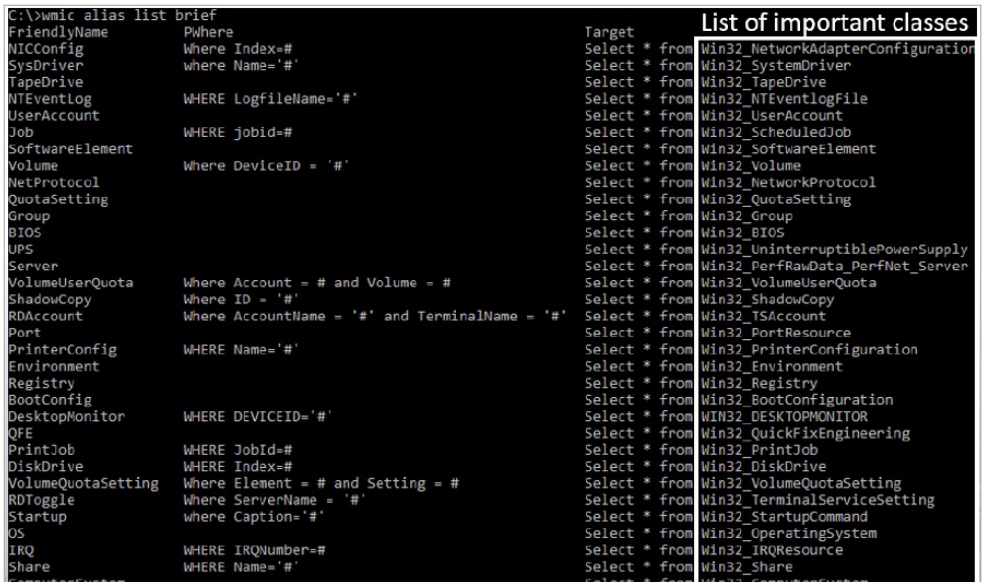
The `class` keyword is chosen when you access the WMI class, or "template." When you are accessing an instance of a WMI class, the `path` keyword will be chosen. Try to remember these definitions for later because their meanings will become clearer with use.

Try: `wmic path Win32_Process get /value`



The `wmic` command has aliases, just like PowerShell does. To access the list of WMIC aliases, type `wmic alias list brief.` Though it is true that `path` or `class` needs to come after `wmic,` an alias has the capability to bypass the `[class|path]` portion of the syntax. In this case, even `alias` is an alias.

The keyword `list` outputs pre-defined sets of data, and it can only be used with aliases. Adding `brief` after `list` outputs only a core set of properties. Omitting `brief` outputs all associated properties.




```
C:\>wmic alias list brief
FriendlyName          PWhere                                                Target
NICConfig             Where Index=#                                         Select * from Win32_NetworkAdapterConfiguration
SysDriver             where Name='#'                                        Select * from Win32_SystemDriver
TapeDrive                                                                   Select * from Win32_TapeDrive
NTEventLog            WHERE LogfileName='#'                                 Select * from Win32_NTEventlogFile
UserAccount                                                                 Select * from Win32_UserAccount
Job                   WHERE jobid=#                                         Select * from Win32_ScheduledJob
SoftwareElement                                                             Select * from Win32_SoftwareElement
Volume                Where DeviceID = '#'                                  Select * from Win32_Volume
NetProtocol                                                                 Select * from Win32_NetworkProtocol
QuotaSetting                                                                Select * from Win32_QuotaSetting
Group                                                                       Select * from Win32_Group
BIOS                                                                        Select * from Win32_BIOS
UPS                                                                         Select * from Win32_UninterruptiblePowerSupply
Server                                                                      Select * from Win32_PerfRawData_PerfNet_Server
VolumeUserQuota       Where Account = # and Volume = #                      Select * from Win32_VolumeUserQuota
ShadowCopy            Where ID = '#'                                        Select * from Win32_ShadowCopy
RDAccount             Where AccountName = '#' and TerminalName = '#'        Select * from Win32_TSAccount
Port                                                                        Select * from Win32_PortResource
PrinterConfig         WHERE Name='#'                                        Select * from Win32_PrinterConfiguration
Environment                                                                 Select * from Win32_Environment
Registry                                                                    Select * from Win32_Registry
BootConfig                                                                  Select * from Win32_BootConfiguration
DesktopMonitor        WHERE DEVICEID='#'                                    Select * from WIN32_DESKTOPMONITOR
QFE                                                                         Select * from Win32_QuickFixEngineering
PrintJob              WHERE JobId=#                                         Select * from Win32_PrintJob
DiskDrive             WHERE Index=#                                         Select * from Win32_DiskDrive
VolumeQuotaSetting    Where Element = # and Setting = #                     Select * from Win32_VolumeQuotaSetting
RDToggle              Where ServerName = '#'                                Select * from Win32_TerminalServiceSetting
Startup               where Caption='#'                                     Select * from Win32_StartupCommand
OS                                                                          Select * from Win32_OperatingSystem
IRQ                   WHERE IRQNumber=#                                     Select * from Win32_IRQResource
Share                 WHERE Name='#'                                        Select * from Win32_Share
```

Note that when certain classes were given aliases in WMIC, these classes are used regularly or are otherwise important enough to warrant aliases. Therefore, one way to find which WMI class is suitable for a specific purpose is to browse the WMIC aliases.

Lesson 2

WMIC Remote Execution

This lesson will guide the user on how to enable and execute remote execution.

Remote WMIC on Remote Host

To access WMI on the remote host with WMIC, you'll need to go through the remote host's Windows firewall. To do this, obtain administrator status on the remote host and run the following command remotely:

```
netsh firewall set service remoteadmin enable
```

netsh Firewall Output

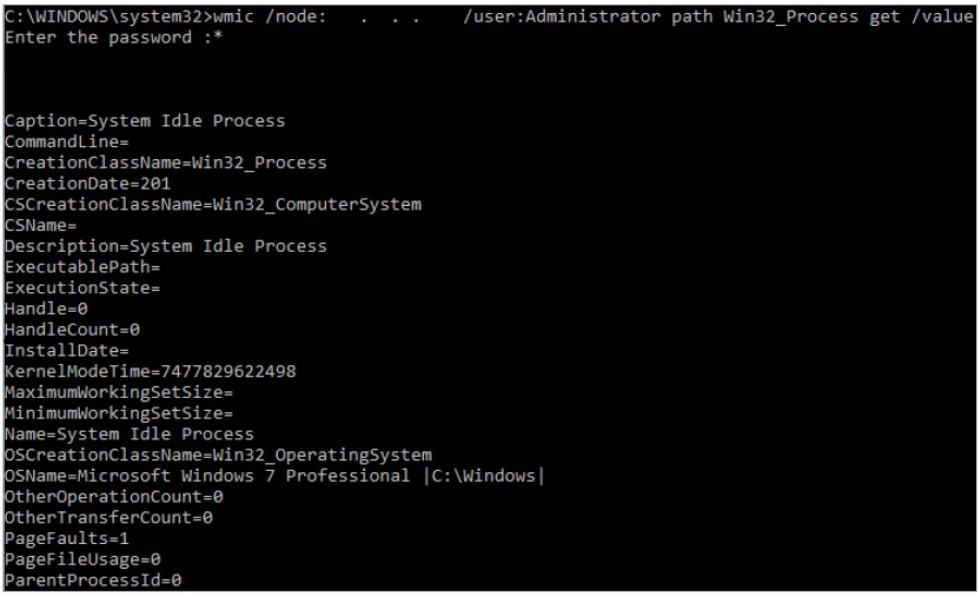
Even though the above command is deprecated, we teach it because it is compatible with older versions of Windows and remains supported by modern versions of Windows.

Remote WMIC on Localhost

Syntax: wmic /node:`<COMPUTERNAME> /user:<USERNAME> <DESIRED WMIC SYNTAX>`

Try: `wmic /node:X.X.X.X /user:Administrator path Win32_Process get /value`

- Where `X.X.X.X` represents an IP or hostname



Lesson 3

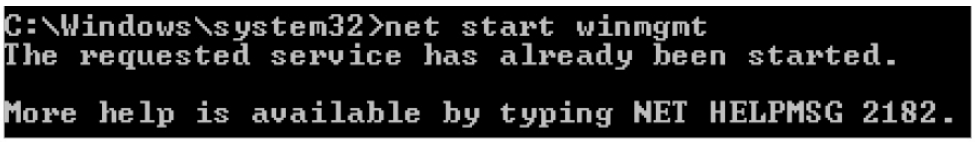
WMIC Command Structures and Syntax

This lesson will examine the structure and syntax of WMIC commands via WMIC.

WMIC Initial Command

If WMIC is stopped, turn it on via CMD by running the following command as administrator:

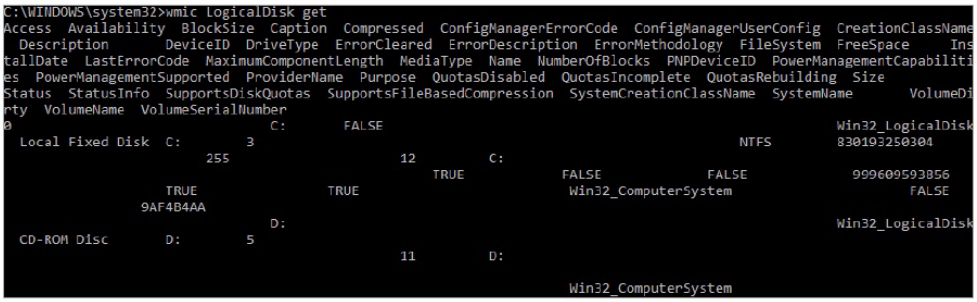
```
net start winmgmt
```



WMIC Output

`get`
Function: This command can retrieve a specific set of properties from instances of WMI classes.
Try: `wmic LogicalDisk get`

- LogicalDisk is an alias for Win32_LogicalDisk



Since this text output is very difficult to read, you can redirect the output to a text file.

Try: `wmic LogicalDisk get > LogicalDisk.txt`

Then try: `LogicalDisk.txt`

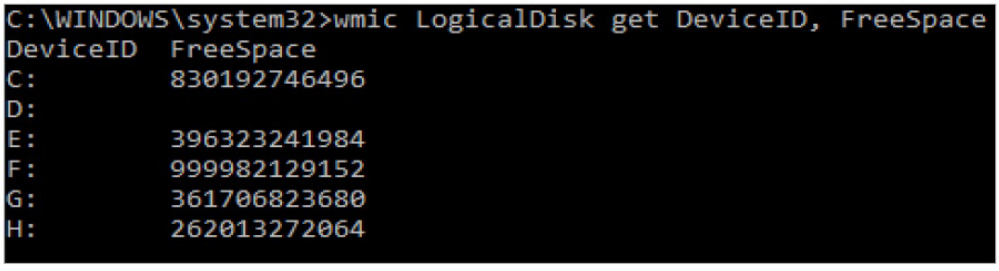
Because the text file will contain a large amount of information, and you will have to view the output from the command line, filtering the text file for relevant information is our best option. The syntax for filtering is as follows:

```
wmic <PATH ARGUMENT(S)> get <COLUMN NAME(S)>
```

The column (property) names must be comma-separated and will tell WMIC to retrieve only those columns from a specified path (instances of a WMI class).

Try: `wmic LogicalDisk get DeviceID,FreeSpace`

While `DeviceID` and `FreeSpace` are used here, feel free to try with any of the columns identified in the text file.



Exercise 5-1, Syntactical Exercise: WMIC, Aliases

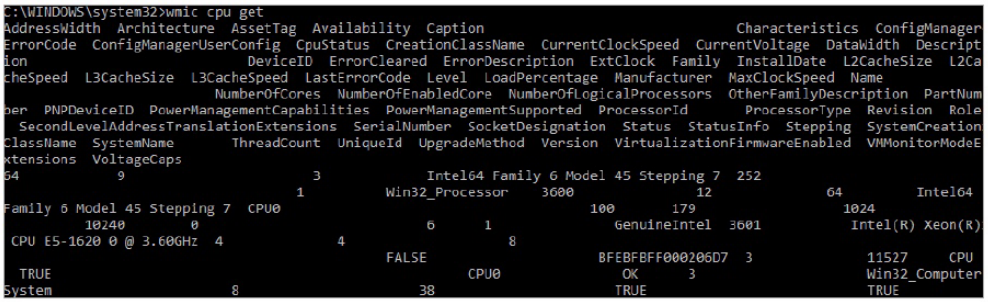
Exercise 5-2, Syntactical Exercise: WMIC, Aliases, Remoting

/value

Function: Can retrieve specific set of properties

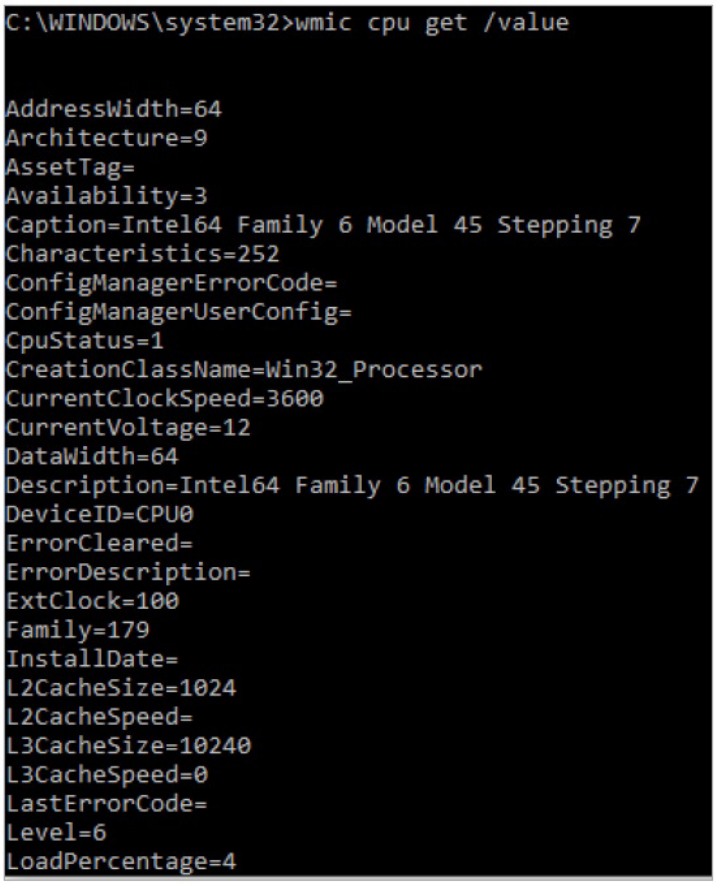
Options: `/value` – Returns output in a list format

- Follows `get` in syntax
- *For reference, try:* `wmic cpu get`
  - `cpu` is an alias for `Win32_Processor`



As you can see, the filtered output is still too messy to read. Sorting is important.

Then, try: `wmic cpu get /value`



Here, the output looks better and it is in a format easily read by the user.

/format

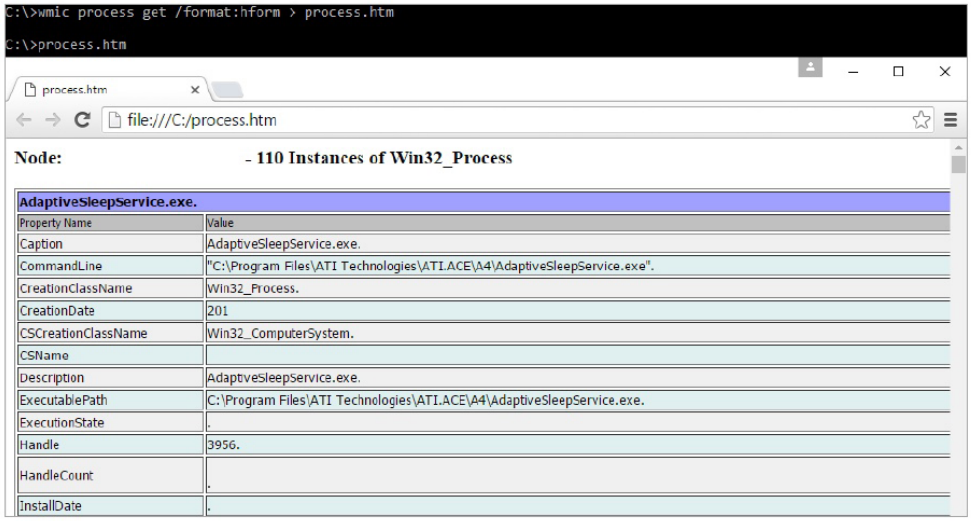
Function: Formats output in accordance with specification (i.e., `<FORMAT>`)
Format: `/format:<FORMAT>` – optional modifier

- Output typically redirected to a file (e.g., "`> outfile.txt`")

Specifications: `hform` – Outputs as HTML, with class properties in left column and instance properties in the right

Try: `wmic process get /format:hform > process.htm`
Store in a directory OUTSIDE of `C:\Windows`

Then try: `process.htm`

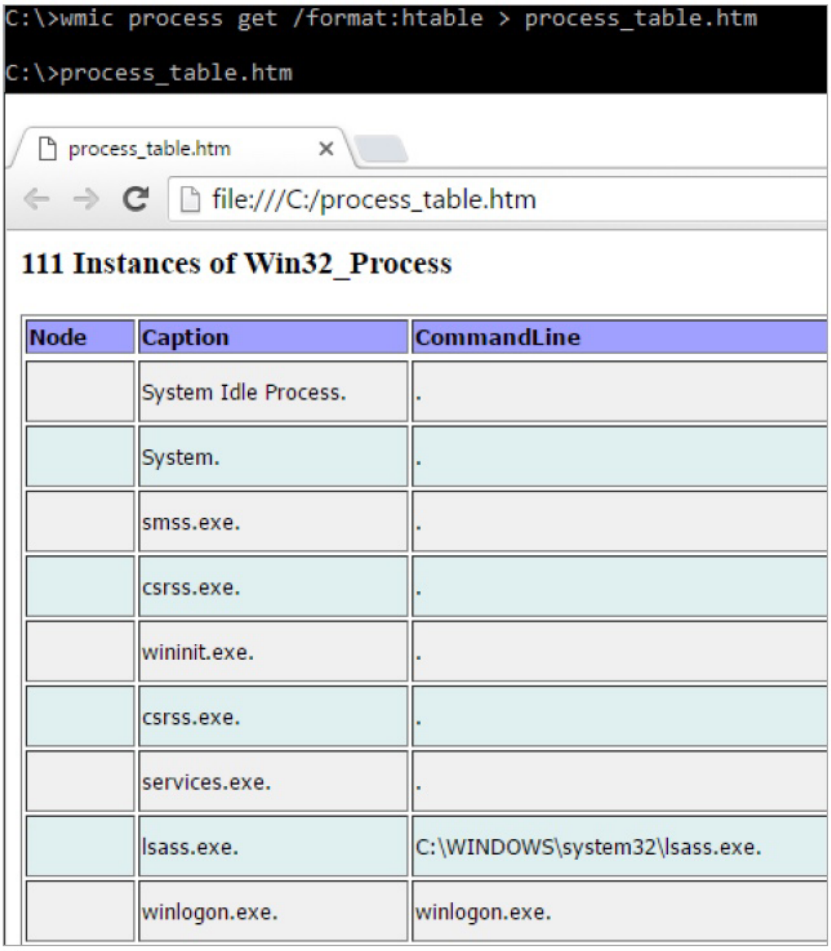


Options:

- `htable` – Outputs as HTM table, with class properties as column names, each row is a new instance

Try: `wmic process get /format:htable > process_table.htm`

Then try: `process_table.htm`



Other options:

`xml` – Outputs as XML

- Some analytic/forensic applications only allow importing from XML files, which is the typical reason to output in this format.

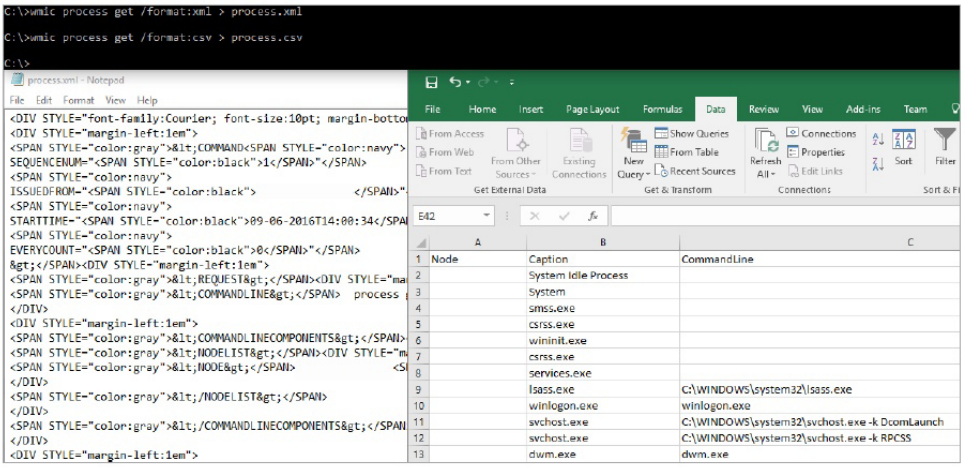
Try: `wmic process get /format:xml > process.xml`

- Now you may view the XML file

`csv` – Outputs as CSV, able to be manipulated in Excel

Try: `wmic process get /format:csv > process.csv`

- Now you may view the CSV file. Import into Excel with the Excel command Data > Get External Data > From Text



Both the XML and CSV output are shown above.

Exercise 5-3, Practical Exercise: WMIC, Remoting, Format

where

Function: Allows `wmic` output to be selected and displayed according to specifications.

Try: `wmic path Win32_Service where "name='WinMgmt'"`

- There is a direct relationship between the where clause and a specific option in the WMI/CIM cmdlets of PowerShell.

Then try : `wmic path Win32_Service where "name='WinMgmt'" get Name,Caption,State`

```
C:\>wmic path Win32_Service where "name='WinMgmt'" get Name,Caption,State
Caption                                Name     State
Windows Management Instrumentation  Winmgmt  Running
```

This output is cleaner.

When the property is an integer (number), use standard operators:

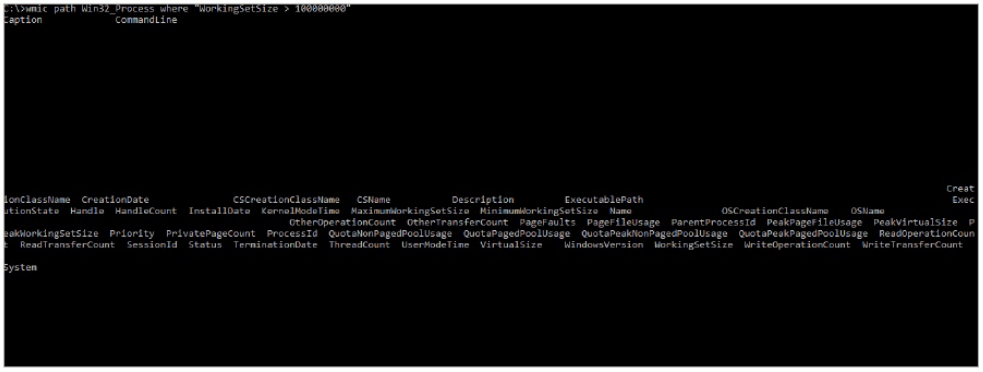
- = - Equal to
- < - Less than
- > - Greater than
- <= - Less than or equal to
- >= - Greater than or equal to
- != or <> - Not equal to

When the property is a string (e.g., “test”), some operators work (=, !=, <>).

Technically, the other operators wouldn’t produce a halting error. However, they’re typically avoided in practice since their effect in filtering may be unexpected.

First, we’ll try to filter by a numeric value. Try: `wmic path Win32_Process where "WorkingSetSize > 100000000”`

Before running the command, consider what it might retrieve.



Consider how we could modify the command to clean up the output shown above.

Next, we’ll filter by a string. Note the placement of the single quotes when filtering by strings. Try: `wmic path Win32_Process where "name='notepad.exe'" get /value`

Again, try to anticipate what the command will retrieve prior to running.

```
C:\>wmic path Win32_Process where "name='notepad.exe'" get /value


Caption=notepad.exe
CommandLine="C:\WINDOWS\system32\NOTEPAD.EXE" C:\
CreationClassName=Win32_Process
CreationDate=201
CSCreationClassName=Win32_ComputerSystem
CSName=
Description=notepad.exe
ExecutablePath=C:\WINDOWS\system32\NOTEPAD.EXE
ExecutionState=
Handle=10420
HandleCount=408
InstallDate=
KernelModeTime=16718750
MaximumWorkingSetSize=1380
MinimumWorkingSetSize=200
Name=notepad.exe
OSCreationClassName=Win32_OperatingSystem
OSName=Microsoft Windows 10 Enterprise|C:\WINDOWS|
OtherOperationCount=3878
OtherTransferCount=92306
PageFaults=21353
PageFileUsage=16280
ParentProcessId=8012
PeakPageFileUsage=22224
PeakVirtualSize=2199356448768
PeakWorkingSetSize=48408
Priority=8
PrivatePageCount=16670720
ProcessId=10420
QuotaNonPagedPoolUsage=26
QuotaPagedPoolUsage=348
QuotaPeakNonPagedPoolUsage=34
QuotaPeakPagedPoolUsage=619
ReadOperationCount=682
```

Exercise 5-4, Syntactical Exercise: WMIC, Where

AND

AND – Requires all statements to be true

Condition 1	Condition 2	Result
True	True	True
True	False	False
False	True	False
False	False	False

(Grab a red AND blue book)

Condition 1	Condition 2	Result
The book is red	The book is blue	Grab Book
The book is red	The book is not blue	Don't Grab Book
The book is not red	The book is blue	Don't Grab Book
The book is not red	The book is not blue	Don't Grab Book

Try: `wmic process where "WorkingSetSize > 100000000 AND name<>'chrome.exe'" get /value`

- What records will this retreive?

```
C:\>wmic process where "WorkingSetSize > 100000000 and name<>'chrome.exe'" get /value


Caption=System
CommandLine=
CreationClassName=Win32_Process
CreationDate=201
CSCreationClassName=Win32_ComputerSystem
CSName=
Description=System
ExecutablePath=
ExecutionState=
Handle=4
HandleCount=1444
InstallDate=
KernelModeTime=45836718750
MaximumWorkingSetSize=
MinimumWorkingSetSize=
Name=System
OSCreationClassName=Win32_OperatingSystem
OSName=Microsoft Windows 10 Enterprise|C:\WINDOWS|
OtherOperationCount=552964
OtherTransferCount=126672279
PageFaults=180316
PageFileUsage=608
ParentProcessId=0
```

```
WorkingSetSize=124334080
```

Consider what will happen when you run this command in the field. If you are retrieving a large amount of irrelevant information, you'll need to filter that data in order to find your information. You can get rid of irrelevant data by filtering for the properties you are searching for.

Imagine you were aware that a piece of malware that uses large amount of physical RAM, and you wanted to see if the artifact was present on the system. Even after you filter for processes that use large amounts of RAM, the search generated too many instances of chrome.exe, perhaps, obfuscating the rest of the results by sheer number. In such a scenario, you would use an AND operator, and perhaps even more than one AND, if the results were still not relevant enough.

OR

Function: Boolean operator that is true if one or both conditions are true.

Condition 1	Condition 2	Result
True	True	True
True	False	True
False	True	True
False	False	False

Condition 1	Condition 2	Result
The book is red	The book is blue	Grab Book
The book is red	The book is not blue	Grab Book
The book is not red	The book is blue	Grab Book
The book is not red	The book is not blue	Don't Grab Book

Try: `wmic process where "name='powershell_ise.exe' OR name='chrome.exe'" get /value`

```
C:\>wmic process where "name='powershell_ise.exe' or name='chrome.exe'" get /value


Caption=chrome.exe
CommandLine="C:\Program Files (x86)\Google\Chrome\Application\chrome.exe"
CreationClassName=Win32_Process
CreationDate=201
CSCreationClassName=Win32_ComputerSystem
CSName=
Description=chrome.exe
ExecutablePath=C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
ExecutionState=
Handle=5836
HandleCount=3603
InstallDate=
KernelModeTime=12024843750
MaximumWorkingSetSize=1380
MinimumWorkingSetSize=200
Name=chrome.exe
OSCreationClassName=Win32_OperatingSystem
OSName=Microsoft Windows 10 Enterprise|C:\WINDOWS|
OtherOperationCount=6663998
OtherTransferCount=220610408
PageFaults=5644138
PageFileUsage=156212
ParentProcessId=4496
PeakPageFileUsage=164772
PeakVirtualSize=1276821504
PeakWorkingSetSize=208320
Priority=8
PrivatePageCount=159961088
ProcessId=5836
QuotaNonPagedPoolUsage=90
QuotaPagedPoolUsage=2082
QuotaPeakNonPagedPoolUsage=436
```

Using the **OR** operator allows you to broaden your result set. In this way, it's something like the opposite of using the **AND** operator, which narrows your result set.

NOT

Function: Boolean operation that is true if a condition is false.

Condition	Result
True	False
True	False
False	True
False	True

(Grab a NOT red book)

Condition	Result
The book is red	Don't Grab Book
The book is red	Don't Grab Book
The book is not red	Grab Book
The book is not red	Grab Book

Try: `wmic process where "name='chrome.exe' and NOT WorkingSetSize > 100000000" get /value`

```
C:\>wmic process where "name='powershell_ise.exe' or name='chrome.exe'" get /value


Caption=chrome.exe
CommandLine="C:\Program Files (x86)\Google\Chrome\Application\chrome.exe"
CreationClassName=Win32_Process
CreationDate=201
CSCreationClassName=Win32_ComputerSystem
CSName=
Description=chrome.exe
ExecutablePath=C:\Program Files (x86)\Google\Chrome\Application\chrome.exe
ExecutionState=
Handle=5836
HandleCount=3603
InstallDate=
KernelModeTime=12024843750
MaximumWorkingSetSize=1380
MinimumWorkingSetSize=200
Name=chrome.exe
OSCreationClassName=Win32_OperatingSystem
OSName=Microsoft Windows 10 Enterprise|C:\WINDOWS|
OtherOperationCount=6663998
OtherTransferCount=220610408
PageFaults=5644138
PageFileUsage=156212
ParentProcessId=4496
PeakPageFileUsage=164772
```

The `NOT` operator is often not necessary because changing other operators can generate the same results. In the example above, the greater than ">" sign could have been flipped to a less than or equal to "<=" and the `NOT` operator removed, and this would have yielded the same results. However, using a `NOT` operator can be easier than rewriting your filter or retrieving a data set in a manner you're unfamiliar with.

like

Function: This operator is used used to filter with wildcards. wmic's approach to wildcards is the same as PowerShell.

- % - similar to *, represents 0+ characters
- _ - similar to ., represents 1 character

Try: `wmic path Win32_UserAccount where "Caption like '%ue_t'" get /value`

```
C:\>wmic path Win32_UserAccount where "Caption like '%ue_t'" get /value


AccountType=512
Caption=                \Guest
Description=Built-in account for guest access to the computer/domain
Disabled=TRUE
Domain=
FullName=
InstallDate=
LocalAccount=TRUE
Lockout=FALSE
Name=Guest
PasswordChangeable=FALSE
PasswordExpires=FALSE
PasswordRequired=FALSE
SID=S-
SIDType=1
Status=Degraded
```

Exercise 5-5, Syntactical Exercise: WMIC, Where

Exercise 5-6, Syntactical Exercise: PowerShell Analogs

call <METHODNAME>

Function: Invokes a WMI method `<METHODNAME>`.

Try: `wmic class Win32_Service where "name='wecsvc'" call startservice`

- "Windows Event Collector" Service

```
C:\>wmic class Win32_Service where "name='wecsvc'" call startservice
Invalid Verb.
```

This is an example of a typical minute error. Make sure you know whether you're calling a class method or an instance (`path`) method. Here, we'll change the command syntax to replace `class` with `path: wmic path Win32_Service where "name='wecsvc'" call startservice`

```
C:\>wmic path Win32_Service where "name='wecsvc'" call startservice
Executing (\\DESKTOP-UP9S5L3\root\cimv2:Win32_Service.Name="Wecsvc")->startservice()
Method execution successful.
Out Parameters:
instance of __PARAMETERS
{
        ReturnValue = 0;
};
```

The return value above, zero, usually means that the specified method executed successfully. If you get an unexpected `ReturnValue` that is not equal to 0, search for the method name with "return value" and the name of the class to which the method definition belongs.

So now that we know how to call methods, how do we find which methods to call? Instead of using the MSDN, a fast and easy way to enumerate available methods for a class is already built into WMIC. To find a list of all available methods for a class/path, use `/?`

Try: `wmic path Win32_Process call /?`

```
C:\>wmic path Win32_Process call /?

Method execution operations.
USAGE:

CALL <method name> [<actual paramlist>]
NOTE: <actual paramlist> ::= <actual param> | <actual param>,  <actual paramlist>

The following verb(s)/method(s) are available:

Call                    [ In/Out ]Params&type                   Status
====                    =====================                   ======
AttachDebugger          [OUT]ReturnValue(uint32)                Implemented

Create                  [IN ]CommandLine(string)                Implemented

                        [IN ]CurrentDirectory(string)

                        [IN ]ProcessStartupInformation(object:Win32_ProcessStartup)

                        [OUT]ProcessId(uint32)

                        [OUT]ReturnValue(uint32)
```

Examine the `IN` and `OUT` parameters for methods, which are included with the method names. `IN` parameters are entered via the command line following the method name, while `OUT` parameters are usually included in the output of the command. The datatypes of the `IN` and `OUT` parameters follow their names in parentheses. You can consider the type `uint32` as an integer, but this is just good information to know because learning WMI datatypes is not an objective of this course.

WMI Study: Win32_Process

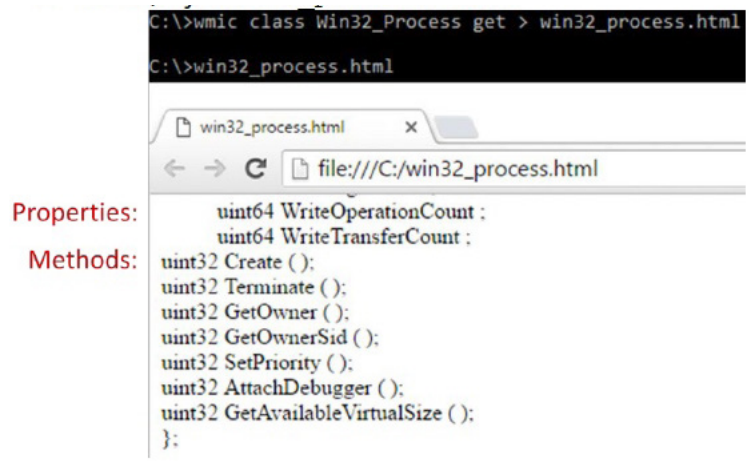
During this study, we'll look at tasks that can be done with WMI and examine how the tasks can be performed in several ways.

Listing Available Class Methods/Properties

With WMIC:

Try: `wmic class Win32_Process get > win32_process.html`

Now open: `win32_process.html`



Now with PowerShell:

```
gwmi –Class Win32_Process | gm
```

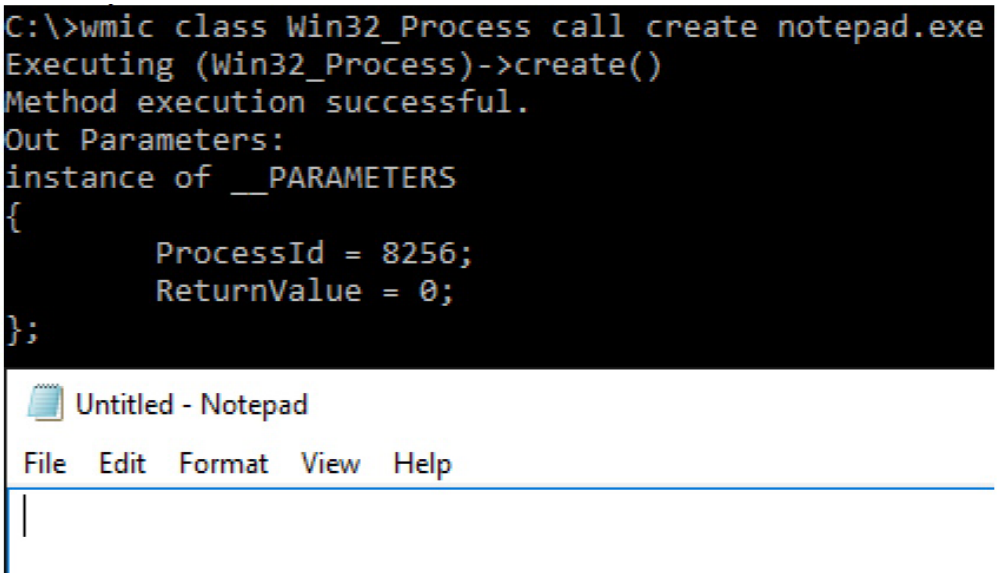
```
PS C:\WINDOWS\system32> gwmi -Class Win32_Process | gm

   TypeName: System.Management.ManagementObject#root\cimv2\Win32_Process

Name                      MemberType     Definition
----                      ----------     ----------
Handles                   AliasProperty  Handles = Handlecount
ProcessName               AliasProperty  ProcessName = Name
PSComputerName            AliasProperty  PSComputerName = __SERVER
VM                        AliasProperty  VM = VirtualSize
WS                        AliasProperty  WS = WorkingSetSize
AttachDebugger            Method         System.Management.ManagementBaseObject AttachDebugger()
GetAvailableVirtualSize   Method         System.Management.ManagementBaseObject GetAvailableVirtualSize()
GetOwner                  Method         System.Management.ManagementBaseObject GetOwner()
GetOwnerSid               Method         System.Management.ManagementBaseObject GetOwnerSid()
SetPriority               Method         System.Management.ManagementBaseObject SetPriority(System.Int32 Priority)
Terminate                 Method         System.Management.ManagementBaseObject Terminate(System.UInt32 Reason)
Caption                   Property       string Caption {get;set;}
CommandLine               Property       string CommandLine {get;set;}
CreationClassName         Property       string CreationClassName {get;set;}
CreationDate              Property       string CreationDate {get;set;}
CSCreationClassName       Property       string CSCreationClassName {get;set;}
CSName                    Property       string CSName {get;set;}
Description               Property       string Description {get;set;}
ExecutablePath            Property       string ExecutablePath {get;set;}
ExecutionState            Property       uint16 ExecutionState {get;set;}
Handle                    Property       string Handle {get;set;}
HandleCount               Property       uint32 HandleCount {get;set;}
InstallDate               Property       string InstallDate {get;set;}
KernelModeTime            Property       uint64 KernelModeTime {get;set;}
MaximumWorkingSetSize     Property       uint32 MaximumWorkingSetSize {get;set;}
```

Notice the methods that are listing the available class methods and properties have limitations. The differences are important enough to prefer one to the other, but each method should be understood in case one is unavailable.

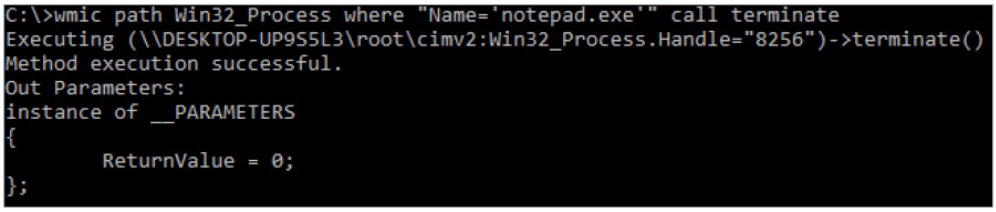
Starting/Terminating a Process

With WMIC:

Try: `wmic class Win32_Process call create notepad.exe`



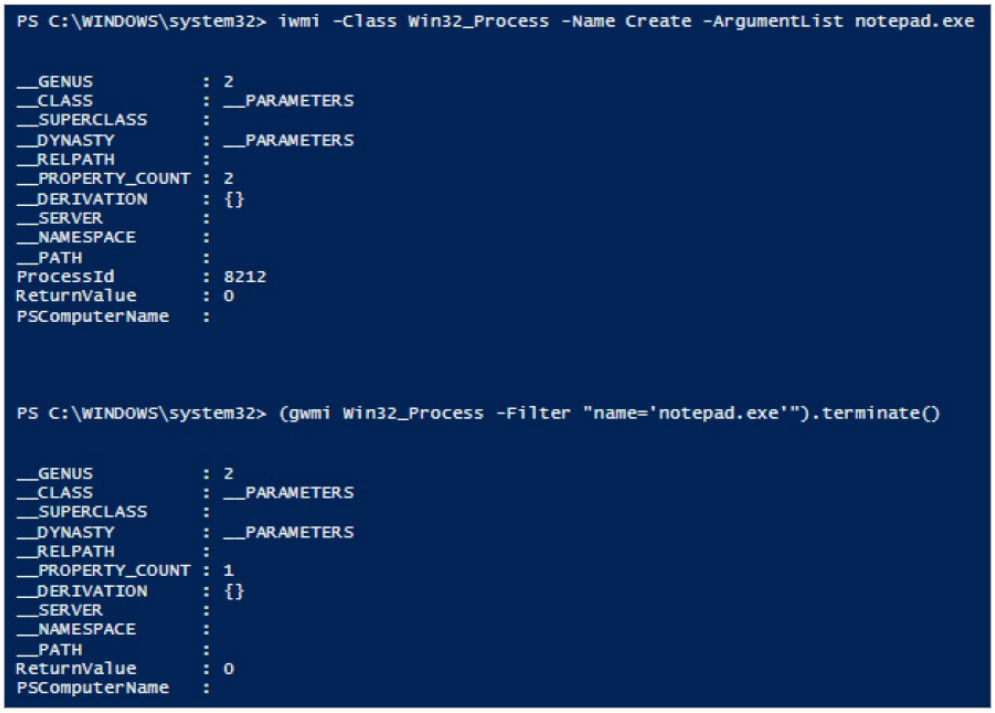
Try: `wmic path Win32_Process where "Name='notepad.exe'" call terminate`



The two commands call different types of methods. The method `create`, which called on a class without requiring an instance of the object (), simply called the method (and specified a single parameter, `notepad.exe`). On the other hand, the method `terminate` required an instance of the class, since one cannot terminate what is not instantiated. In the case of `terminate`, the list of class instances was filtered to `notepad.exe`, which were properly terminated.

Finally, note that both `class` and `path` were properly used.

Now with PowerShell:
Try: `iwmi -Class Win32_Process -Name Create -ArgumentList notepad.exe`



Try: `(gwmi Win32_Process -Filter "name='notepad.exe'").terminate()`

Because these examples are still accessing WMI, albeit from PowerShell, they must also consider which methods will first require an instance. In the case of create, an instance is not necessary, since the creation of an instance is its purpose. In the case of terminate, an instance is required, and is therefore retrieved first. PowerShell has its own way of calling methods and retrieving lists, however, which makes the commands arguably simpler.

Exercise 5-7, Syntactical Exercise: WMIC, Call, Classes vs. Instances

Exercise 5-8, Syntactical Exercise: Call, Classes vs. Instances, PS Analogs

Windows PowerShell Cookbook

All Updates/Installed Software

WMIC

```
wmic product get name,version
```

All installed programs (not applications like Solitaire or Notepad)

```
wmic qfe get
```

All Windows updates

PS

```
Get-Package
```

Force Silent Shutdown WMI

WMI

```
wmic /node:X.X.X.X /user:[USERNAME] path win32_operatingsystem
where "name like 'micro%'" call win32shutdown 5
```

- Replace `[USERNAME] with a desired username`
- Replace X.X.X.X with a desired IP

PS

```
(gwmi win32_operatingsystem -Filter "name like 'micro%'"
-ComputerName X.X.X.X -Credential$cred).win32shutdown(5)
```

Software That Runs on Windows Startup

WMIC

```
wmic startup list brief
```

Preferred

PS

```
Get-ItemProperty HKLM:\SOFTWARE\Microsoft\Windows\
CurrentVersion\run
```

This is not the sole registry location, just common

Discover/Change Services on Startup

WMIC

```
wmic service where "StartMode='Auto'" get Name,State
```

```
wmicservice where "namelike 'Fax' OR name like
'dhcp'" call ChangeStartMode Disabled
```

Replace Fax or dhcp with a desired service

Retrieve All Users

WMIC

```
wmic useraccount list brief
```

PS

```
gwmi Win32_UserAccount
```

Retrieve Log Files/Entries

WMIC

```
wmic path Win32_NTEventLogFile where "logfilename='security'" get
FileSize,MaxFileSize,NumberOfRecords
```

```
wmic path Win32_NTLogEvent where
"logfile='application' and type!='information'" get
CategoryString,EventCode,EventType,Message,TimeGenerated /value
```

Retrieve Log Files/Entries

PS

```
gwmi Win32_NTEventLogFile -Filter "logfilename='security'" |
select FileSize,MaxFileSize,NumberOfRecords
```

```
gwmi Win32_NTLogEvent -Filter "logfile='application'
and type!='information'" -Property
CategoryString,EventCode,EventType,Message,TimeGenerated
```

Retrieve Log Files/Entries

PS

```
Get-EventLog -List
```

```
Get-WinEvent -ListLog *
```

Retrieves all log files

```
Get-WinEvent -LogName Application
```

PS: List/Modify Firewall Rules

```
Get-NetFirewallRule -all
```

```
New-NetFirewallRule
```

- Options:
  - Required: `-DisplayName <DISPLAYNAME>`
  - `-Action [ALLOW|BLOCK|NOTCONFIGURED]`
  - `-RemoteAddress <IPIFNEEDED>`
  - `-Direction [INBOUND|OUTBOUND]`
  - `-LocalPort <LOCALPORT>`
  - `-RemotePort <REMOTEPORT>`

PS: Traverse the Registry

```
cd HKLM:\
```

```
cd HKCU:\
```

PS: File Hash

```
Get-FileHash -Algorithm SHA1 file.txt
```

- Replace `SHA1` with a desired algorithm
- Replace `file.txt` with a desired filename
- Algorithms: `SHA1, SHA256, SHA384, SHA512, MACTripleDES, MD5, RIPEMD160`

PS: Retrieve Network Connections

```
Get-NetTCPConnection
```

```
Get-NetUDPEndpoint | select
LocalAddress,LocalPort,OwningProcess
```

- Find the OwningProcess (Process that opened that connection)
  - `gwmi Win32_Process -Filter "ProcessID=4"`
    - Replace `4` with whatever PID is in `OwningProcess`
  - `gps -id 4`

PS: Export/Convert Results

```
Get-Process | Export-Csv procs.csv
```

Replace Get-Process with a desired cmdlet

```
Get-Process | ConvertTo-Html > procs.html
```

PS: Ping Sweep/Port Scan

```
1..255 %{echo "X.X.X.$_";ping -n 1 -w 100
X.X.X.$_ | Select-String ttl}
```

where X.X.X. represents the first three octets of the IP network

```
1..1024 | %{echo ((new-object Net.Sockets.TcpClient).
Connect("X.X.X.X",$_)) "Port $_ is open"} 2>$null
```

where X.X.X.X represents the scanned IP

PS: Download/wget File

From PS

```
(New-Object System.Net.WebClient).DownloadFile('http://X.com/
nc.exe', 'nc.exe')
```

- Replace `'http://X.com/nc.exe'` with a website of your choosing
- Replace `'nc.exe'` with a filename of your choosing

From CMD

```
Powershell -c "(New-Object System.Net.WebClient).
DownloadFile('http://X.com/nc.exe', 'nc.exe')"
```

Windows PowerShell Cheat Sheet

Help Commands

PS

- `Get-Help` - `man`
- `Get-Alias` - `gal`
- `Get-Command` - `gcm`
- `Get-Member` - `gm`

WMI

- PS:

```
gwmi -Recurse -List
```

```
gwmi <CLASS> | gm
```

```
gcls
```

```
gcls <CLASS> | select -ExpandProperty cimclassmethods
```

- WMIC:

```
wmic alias list brief
```

```
wmic /?WMIC:
```

```
wmic <CLASS|PATH|ALIAS> call /?
```

PS Syntax

- "Get" Verb –Typically displays or retrieves data
- "Set" Verb – Typically inputs or changes data
- "New" Verb – Typically creates a new item or object
- "Remove" Verb – Typically deletes data
- "Add" Verb – Typically appends data

Initial Commands

WMIC

```
net start/stop winmgmt
```

Turn WMIC on/off via CMD

PS

```
Set-ExecutionPolicy RemoteSigned
```

```
Update-Help
```

```
$WhatIfPreference = $true
```

For testing only!

Initial Remoting Commands

WMIC

```
netsh firewall set service remoteadmin enable
```

PS

On remote computer:

- **PS**

```
Enable-PSRemoting
```

- **CMD**

```
powershell -c Enable-PSRemoting
```

On local computer:

- **PS**

```
Set-Item WSMan:\localhost\Client\TrustedHosts -Value [X.X.X.X]
```

```
$cred = Get-Credential
```

PS Remote Session

```
Enter-PSSession X.X.X.X -Credential [USERNAME|CREDENTIAL
VARIABLE]
```

- Replace `X.X.X.X with a desired IP`
- E.g: `Enter-PSSession 192.168.0.0 -Credential $cred`

To exit session:

```
exit
```

Remote WMI

PS

```
gwmi Win32_OperatingSystem -ComputerName X.X.X.X - Credential
$cred
```

- Replace Win32_OperatingSystem with a desired class
- Replace X.X.X.X with a desired IP

WMIC

```
wmic /node:X.X.X.X /user:<USERNAME> <DESIRED WMIC SYNTAX>
```

PS Scripting

Basic Scripting

- `Read-Host` – Receive command line input
- `Write-Host` – Outputs text to the command line
- `if, elseif, else` – If statements
- `For` – Run a loop a specific number of times
- `While` – Run a loop until a specific condition is met

Special Characters

- | – Pipe
- () – Grouping expression
- # – One-line comment
- ` – PowerShell escape character (Note: Backtick, NOT apostrophe)
- . – Dot sourcing
- ; – Denotes end of command

Cmdlets

- `Select-Object -Select` – Retrieve objects (columns) from a collection
  - E.g.: `Get-Process | select name,ws`
  - To expand a contracted list: `<…> | select -ExpandProperty <PROPERTYNAME>`
- `Compare-Object -Compare` – Compare all objects in two collections
  - E.g.: `compare $a $b`
- `Where-Object – ?` – Check each row of data for a specified condition
  - E.g.: `Get-Process | ?{$_.name -match "cmd"}`
- `ForEach-Object - %` – Run commands against each item in a collection
  - E.g.: `Get-Process | %{Write-Host $_.name,$_.ws}`
- `Select-String – sls` – Search for string within file or output
  - E.g.: `sls -Path C:\get_highmemuse.psm1 -Pattern export`
  - `Select-String` will not retrieve an object. It makes a new object with fewer properties.

Functions/Modules

- `function <NAME> { <COMMANDS> }`
  - E.g.: `function G-Proc { Get-Process }`
- `Export-ModuleMember` – Export functions, variables, etc., from a script
  - E.g.: `Export-ModuleMember -Function "G-Proc"`
- `Import-Module` – Import functions, variables, and more from a script
  - E.g.: `Import-Module C:\g-proc.psm1`

PS Regular Expressions

- `-match` – Case-insensitive ("test" matches "Test")
- `-cmatch` – Case-sensitive ("test" does not match "Test")
- * – Represents 0 or more characters
- . – Represents 1 character
- \ – Escape character (e.g. to search for "." or "*" themselves)

PS Formatting

- `Format-Table – ft` – Normal formatting, but can expand contracted items
  - E.g.: `Get-Service | ft –auto`
- `Format-List – fl` – List properties and property names for each instance
  - E.g.: `Get-Service | fl`
- `Format-Custom – fc` – Format as classes and instances
  - E.g.: `Get-Service | fc`
- `Out-GridView – ogv` – Format as a filterable table in a new window
  - E.g.: `Get-Service | Out-GridView`

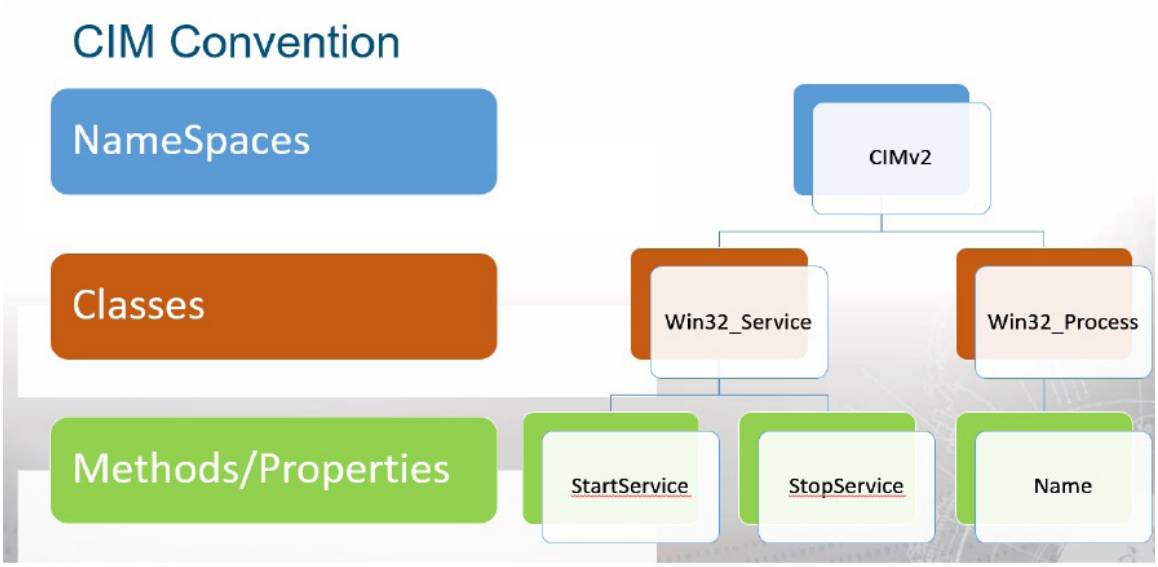
PowerShell Condition Operators

Operator	Definition
-lt	Less than
-le	Less than or equal to
-gt	Greater than
-ge	Greater than or equal to
-eq	Equal to. If the left-hand side of the operator is an array and the right-hand side is a scalar, the equivalent values of the left-hand side will be returned
-ne	Not equal to. If the left-hand side of the operator is an array and the right-hand side is a scalar, the not equivalent values of the left-hand side will be returned
-contains	Determine elements in a group; this always returns Boolean `$True` or `$False`
-notcontains	Determine excluded elements in a group, this always returns Boolean `$True` or `$False.`
-like	Like – uses wildcards for pattern matching
-notlike	Not like – uses wildcards for pattern matching
-match	Match – uses regular expressions for pattern matching
-notmatch	Not match – uses regular expressions for pattern matching
-band	Bitwise AND
-bor	Bitwise OR
-is	Is of type
-isnot	Is not of type
-clt	Less than (case-sensitive)
-cle	Less than or equal to (case-sensitive)
-cgt	Greater than (case-sensitive)
-cge	Greater than or equal to (case-sensitive)
-ceq	Equal to (case-sensitive)
-cne	Not equal to (case-sensitive)
-clike	Like (case-sensitive)
-cnotlike	Not like (case-sensitive)
-ccontains	Left-hand side contains right-hand side in a case-sensitive manner

Operator	Definition
-cnotcontains	Determine excluded elements in a group in a case-sensitive manner
-cmatch	Match (case-sensitive)
-cnotmatch	Not match (case-sensitive)
+	Add
-	Subtract
*	Multiply
/	Divide
%	Modulo
-not	Logical not
!	Logical not
-band	Binary and
-bor	Binary or
-bnot	Binary not
-replace	Replace (e.g. `"abcde" –replace "b","B"`) (case-insensitive)
-ireplace	Case-insensitive replace (e.g. `"abcde" -ireplace "B","3"`)
-creplace	Case-sensitive replace (e.g. `"abcde" -creplace "b","3"`)
-and	AND (e.g. `($a -ge 5 -AND $a -le 15)`)
-or	OR (e.g. `($a -eq "A" -OR $a -eq "B")`)
-is	IS type (e.g. `$a -is [int]`)
-isnot	IS not type (e.g. `$a -isnot [int]`)
-as	Convert to type (e.g. `1 -as [string] treats 1 as a string`)
..	Range operator (e.g. `foreach ($i in 1..10) {$i }`)
&	Call operator (e.g. `$a = "Get-ChildItem" &$a executes Get-ChildItem`)
. (dot followed by a space)	Call operator (e.g. `$a = "Get-ChildItem" . $a executes Get-ChildItem in the current scope`)
-F	Format operator (e.g. `foreach ($p in Get-Process) { "{0,-15} has {1,6} handles" -F $p.processname,$p.Handlecount }`)

PowerShell Forensics

Command	Alias	Description
Get-ChildItem	GCI	(or DIR or LS) Similar to "dir" command, it gets the items and child items from one or more directories. It can also identify the MAC time stamps
Get-ItemProperty	GP	Primarily used to get the property values of registry entries
Get-WmiObject	GWMI	Lists details of a WMI class
Get-CimInstance	GCIM	Accesses WMI/CIM via WSMan/WinRM by default
Get-Process	GPS	Lists the processes that are running on the machine
Get-Service	GSV	Lists the services that are running on the machine
Get-WinEvent	N/A	Lists the events from event logs and event tracing files
Get-HotFix	N/A	Lists the hotfixes applied on the machine
Get-Content	GC	Lists the contents of a file
Write-Host	N/A	Enables writing messages to the console; Useful if the command or the script need to be run interactively
Get-FileHash	N/A	Accepts input for the path to the file to hash, and it returns an object with the path to the file and the hash value



WMI Access

PS

- `Get-WmiObject – gwmi`
  - E.g.: `gwmi win32_process -Filter "name='cmd.exe' and handle=8012"`
- `Invoke-WmiMethod – iwmi`
  - E.g.: `iwmi -Class Win32_Process -Name Create -ArgumentList notepad.exe`
- `Get-CimInstance – gcim`
  - E.g.: `gcim Win32_Process`
- `Invoke-CimMethod – icim`
  - E.g.: `icim -ClassName Win32_Process -MethodName Create -Arguments @{commandline="notepad.exe"}`
- `Get-CimClass – gcls`
- `wmic class Win32_Process`
  - Returns class definition in HTML format
  - Replace `Win32_Process` with your desired class
- `wmic path Win32_Process`
  - Returns current instances of specified class

WMIC Verbs/Clauses:

- `get` – Retrieves specific sets of properties
  - E.g.: `wmic path Win32_Process get name,handle`
- `where` – Retrieves specific instances
  - E.g.: `wmic path Win32_Process where "name='cmd.exe'" get name,handle`
- `like` – Operator to specify instances using regular expression-type filtering
  - % - similar to *, represents 0+ characters
  - _ - similar to ., represents 1 character
  - E.g.: `wmic path Win32_Process where "name like 'cm%'" get name,handle`
  - E.g.: `wmic path Win32_Process where "name like 'vm%' and (priority>=8 or priority=6)" get name,handle,priority`
- `call <METHODNAME> <PARAMETERS>` – Invokes a WMI method
  - E.g.: `wmic class Win32_Process call create notepad.exe`
    - Invokes a static, or class, method
  - E.g.: `wmic path Win32_Process where "name='notepad.exe'" call terminate`
    - Invokes an instance, or path, method

WMIC Format Switches

- `/value` – Returns output in a list format; used after get
  - E.g.: `wmic cpu get /value`
- `/format:<FORMAT>` – Outputs in a specified format; used after get
  - `hform` – Outputs as HTM, with class properties in left column, instance properties in the right
    - E.g.: `wmic cpu get /format:hform`
  - `htable` – Outputs as HTM table, with class properties as column names, each row is a new instance
  - `xml`– Outputs as XML, useable by some applications which may receive the data
  - `csv`– Outputs as CSV, able to be manipulated in Excel

3rd Party PS Modules

PowerForensics:

- Digital forensics framework for NTFS
- https://github.com/Invoke-IR/PowerForensics

PowerSploit:

- Aids pen testers for all phases of assessments
- https://github.com/PowerShellMafia/PowerSploit

CimSweep

- CIM/WMI tool suite for incident response
- https://github.com/PowerShellMafia/CimSweep

Nishang:

- Framework with scripts and payloads for pen testing
- https://github.com/samratashok/nishang

Kansa:

- Framework for incidence response
- https://github.com/davehull/Kansa

Acronyms and Terms

A, B	
Algorithm	A step-by-step procedure for solving a problem or accomplishing some end, especially by a computer
Array	Variable that holds a list of values
C	
CIM	Common Information Model
CMD	Windows Command Prompt
Cmdlet	PowerShell commands
D, E	
DCOM	Distributed Component Object Mod
DMI	Desktop Management Interface
DMTF	Distributed Management Task Force
F, G, H	
File system	Data structure for data management, storage, and retrieval; aka "filesystem"
Hotfix	A software patch
I, J, K, L	
ISE	Integrated Scripting Environment
M, N	
Modulo	Operation that returns the remainder of division
MOF	Managed Object Format
O	
OS	Operating System
P, Q, R	
Procedure	A series of actions that are done in a certain way or order
Process	In pseudocode, a process is a series of actions that produces something or that leads to a particular result
PS	PowerShell

S, T, U, V	
Scalar	Variable that holds a single value
SNMP	Simple Network Management Protocol
SP	Service Pack
SSH	Secure Shell
W, X, Y, Z	
WinRM	Windows Remote Management
WMI	Windows Management Instrumentation
WMIC	WMI Command-line
WS-Man	Web Services-Management

EXERCISES

Exercise 1-1, Syntactical Exercise: CD and Variables

End State
Navigate to the C:\Windows\System32\drivers\etc directory using one system variable.
Rubric
• Navigate to C:\Windows\System32\drivers\etc
• Use one variable

Exercise 1-2, Practical Exercise: CMD Navigation and Data Gathering for Extraction

Background
You are a penetration tester looking to extract a directory listing from a remote box ("PR Win10" VM). You have previously retrieved administrator credentials for the system. The directory listing will be of the Documents, Downloads, and Desktop folders of the user Carl. Temporary storage for the directory listings will need to be created, and you decide to name the directory "TMP" and place it in the filesystem root. Cover your tracks as needed. At a later time, when more information has been collected, you intend to retrieve all files located in this TMP directory.
End State
Make a new directory located at the root of the filesystem called "TMP".
Make directory listings for the Documents, Downloads, and Desktop folders of the user Carl.
Save the text file(s) of these directory listings to the TMP directory.
If relevant, delete any files you might have accidentally created during your work.
Rubric
• Create a text file or text files from the directory listings of the Documents, Downloads, and Desktop folders of the user Carl
• Create a new directory named "TMP" at the root of the file system
• Store the text file(s) in TMP
• Remove any additional files outside of TMP created during the exercise

Exercise 1-3, Practical Exercise: Batch File Scripting

End State
Write a batch file that copies 2 system files to a new directory.
The batch file should navigate into C:\Windows\System32\drivers\etc
The files "hosts" and "networks" should be copied to C:\
The files should then be moved to a new directory: C:\hosts_networks
A directory listing for C:\hosts_networks should be sent to C:\hosts_networks\dir.txt
The file should close displaying "Process Complete."

Example closing display:
 	
`C:\hosts_networks>[command]`
 	
`Process Complete`

Rubric
• Navigate to C:\Windows\System32\drivers\etc
• Copy the two files to C:\
• Create a new directory
• Move the two files into the new directory
• Send the new directory's listing to dir.txt
• Display "Process Complete" once batch file is done

Exercise 2-1, Syntactical Exercise: Pseudocoding

End State
Provide directions on how to turn left in a car at a two-way stop sign.
Note: Assume brakes (not emergency brakes) are already applied.
Rubric
• Account for different transmission types
• Ensure traffic is clear before turning

Exercise 2-2, Syntactical Exercise: Pseudocoding

End State
Retrieve five numbers from the user.
Output the numbers.
Output the sum of the numbers.
Rubric
• Pseudocode to retrieve 5 numbers from a user
• Pseudocode to output the numbers and the sum of the numbers

Exercise 2-3, Syntactical Exercise: Pseudocoding

End State
Given: You have a file with a list of colors, with "done" as the last entry.
Read the file.
Count how many unique colors are referenced in the file.
Count the number of times that the color "red" occurs.
Output how many unique colors are referenced in the file.
Output the number of times that the color "red" occurs.
Rubric
• Pseudocode to read the file
• Pseudocode to count how many unique colors are referenced
• Pseudocode to count how many times "red" is referenced
• Pseudocode to output both counts

Exercise 2-4, Practical Exercise: Pseudocoding

Background
You are an incident responder searching a potentially infected computer for malware used in a recent security breach. You need to check to see if the malware is running on the system. The malware runs as a service named "O? 'ŽrtñåÈ²$Ó". Therefore, you must retrieve a list of services running on the computer, and see if any of those match this name. You decide to pseudocode the script first.
End State
Given: You have the name of a service created by malware: O? 'ŽrtñåÈ²$Ó
Get a list of all services running on the system.
Compare the name of each service to the name given: O? 'ŽrtñåÈ²$Ó
If the service name matches the malicious service name (the name given), output that the malware is likely present on the system.
If no service name matches the malicious service name (the name given), output that the malware might not be present or active on the system.
Rubric
• Pseudocode to list all running services
• Pseudocode to compare each name in the list to the name of the malicious service
• Pseudocode to output that the malware had been discovered, if a service name matches
• Pseudocode to output that the malware might not be present or active if no service name matches

Exercise 2-5, Syntactical Exercise: Pseudocoding

Background
You are an incident responder searching a potentially infected computer for malware used in a recent security breach. You have a list of MD5 hashes for known malware executables. You intend to create a PowerShell script to retrieve a list of files from a user's Downloads folder and retrieve the hashes of those files for comparison against the list of known malware hashes. However, you must first pseudocode this script to understand how to write it.
End State
Given: You have a list of hash values corresponding to known malware executables.
Retrieve a list of all files in the Downloads folder for a user (Does not require specific language).
Calculate the MD5 hash of each file in the Downloads folder.
For each hash value calculated, compare this retrieved hash to all hashes in the file of malware hashes.
Output whether or not the search yielded a positive result (found malware).
If malware was discovered, output the name of each file in the Downloads folder that matched.
Rubric
• Pseudocode to list all files in the Downloads folder
• Pseudocode to calculate a hash for each file in the Downloads folder
• Pseudocode to compare each hash to all hashes in the file of malware hashes
• Pseudocode to output that there was a match or that no matches were discovered
• Pseudocode to output the name of each file that matches, if discovered

Exercise 3-1, Syntactical Exercise: Get-Command and Wildcards

End State
Using Get-Command (gcm), find an appropriate command to retrieve all running processes.
Retrieve a list of all running processes.
Rubric
• Retrieve a list of all running processes

Exercise 3-2, Syntactical Exercise: Get-Command, Wildcards, Service Manipulation, and WhatIf

End State
Find commands that manipulate services.
Using the "-WhatIf" option, stop the service DHCP, to test out the command syntax (straightforward).
Start the service W32Time.
Stop the service W32Time.
Rubric
• Pretend to stop the DHCP service for command verification
• After verifying the command, starts and stops the W32Time service

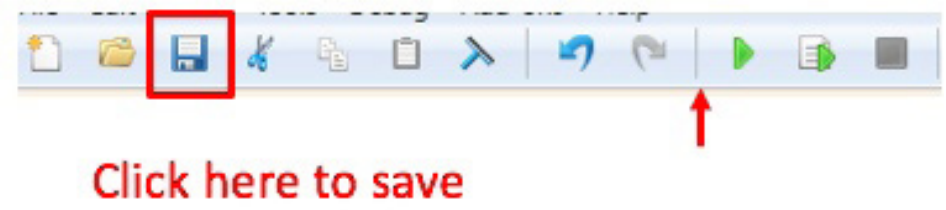
Exercise 3-3, Syntactical Exercise: Variables, Casting, and User Input

End State
Accept an integer from the user and store the value into an integer variable called number.
Add 10 to the variable.
Compare.
Accept a number from the user and store the value into a string variable called newnumber.
Add 10 to the variable in the same way as before.
Rubric
• Read a number from the user
• Add 10 to the number
• Compare: Notice the difference when the variable is cast as a string. Discuss with your classmates or instructors if you don't understand why there's a difference.

Exercise 3-4, Syntactical Exercise: Sorting

End State
Retrieve a list of all running processes.
Sort the processes by the total CPU processing time in descending order.
Rubric
• Retrieve all running processes
• Sorts processes by CPU time in descending order

Exercise 3-5, Syntactical Exercise: If/Else, Scripting

End State
Write a script that:
• Prompts the user for a number
• Stores that number into an appropriate variable type
• Outputs whether that number was greater than, or equal to/less than 62
Pseudocode first!
Save the script anywhere (see graphic below). 
Rubric
• Pseudocode script
• Prompt the user for and accepts a number
• Store number into appropriate variable type
• Output number's relation to 62
• The script should be able to be run from a saved script file (.ps1)

Exercise 3-6, Practical Exercise: Scripting, Where-Object, and Byte Conversions Background

You are an incident responder looking for malware on a potentially compromised host. The malware disguises itself as a legitimate process by using the name "svchost". However, the malware has one tell-tale sign of its presence, which is that it needs to be allocated more than 100 MB of physical RAM. Check to make sure that the host isn't compromised by this malware.
End State
Retrieve all running processes on the host with the name "svchost".
Filter this result to only those allocated over 100 MB of physical memory.
Hint: WorkingSet (alias: WS) is a property of System.Diagnostics.Process (the object returned from Get-Process), which represents physical memory use in bytes.
Verify
If the malware does not appear to be present on the system, filter the "svchost" processes to only those allocated more than 1 MB of RAM.
If results are returned during verification, you indeed ran the correct command.
If you still have no results, you may have an issue with your script.
Rubric
• Retrieve all "svchost" processes allocated over 100 MB of RAM
• Verify: Retrieve all "svchost" processes allocated over 1 MB of RAM

Exercise 3-7, Syntactical Exercise: ForEach-Object

End State
Retrieve a list of all running processes.
Using ForEach-Object, attempt to stop all running processes.
Use the -WhatIf option for stopping the processes, so that nothing is actually stopped.
Rubric
• Attempt to stop all running processes using ForEach-Object

Exercise 3-8, Syntactical Exercise: Get-ChildItem and File Properties

End State
Retrieve a list of hidden files in Carl's Downloads directory.
Include any hidden files in this list.
Retrieve a list of system files in Carl's Desktop directory.
Include any system files in this list.
Rubric
• Retrieve only hidden files in Carl's Downloads
• Retrieve only system files in Carl's Desktop

Exercise 3-9, Syntactical Exercise: Get-Content

End State
Given: You have a list of hash values corresponding to known malware executables. The file is located in C:\Users\Administrator\Documents\bad_hashes.txt
Retrieve the total number of lines contained in the list of hash values.
Retrieve the last 2 hashes from the list of hash values.
Rubric
• Retrieve the total number of hashes contained in bad_hashes.txt
• Retrieve only the last two hashes contained in the file

Exercise 3-10, Practical Exercise: Get-ChildItem and Get-FileHash

Background
You are an incident responder looking for indications of a classified file on an unclassified system ("PR Win10" VM). You have been instructed to calculate hashes for all files located in Carl's Documents folder, and to save these hashes to a file for later use. These hashes will be used to determine whether or not classified data resides in that folder.
End State
Calculate a SHA1 hash for each file located in the user Carl's Documents directory.
System files and hidden files must be included in the hashes.
Output each hash into a single file.
The output file must contain only hashes.
Rubric
• Calculate hashes of all files in Carl's Documents directory
• Create a single text file containing only hashes of the files in Documents

Exercise 3-11, Practical Exercise (Advanced): Scripting, Get-FileHash, and Get-Content

Background
You are an incident responder looking for indications of malware on a potentially infected system. You have been instructed to calculate hashes for all files located in Mary's Documents, Downloads, and Desktop folders. You intend to check each hash against the hashes contained within the C:\Users\Administrator\Documents\bad_hashes.txt file. First, you intend to pseudocode the script. Then, you intend to implement the pseudocode in PowerShell using a combination of Get-Content, ForEach-Object, and Get-FileHash.
End State
Given: You have a list of hash values corresponding to known malware executables. The file is located in C:\Users\Administrator\Documents\bad_hashes.txt
Calculate a SHA256 hash for each file in Mary's Documents, Downloads, and Desktop folders.
Check each hash against every hash in the bad_hashes.txt file.
Keep only the PS objects that match a hash in the bad_hashes.txt file.
Output the name and hash for any file hashes that match one of the given hashes.
Report: Write a quick paragraph providing your findings and a suggested course of action.
Rubric
• Calculate hashes of all files in Mary's Documents, Downloads, and Desktop folders
• Retain only PS objects that match a hash listed in bad_hashes.txt
• Use ForEach-Object for matching
• Output only the name and hash for retained objects
• Write a quick paragraph providing your findings and a suggested course of action

Exercise 3-12, Practical Exercise (Advanced): Scripting, Get-FileHash, Get-Content, and Compare-Object

Background
You are an incident responder looking for indications of a classified file on an unclassified system ("PR Win10" VM). You have been instructed to calculate hashes for all files located in Carl's Documents, Downloads, and Desktop folders. You intend to check each hash against the hashes contained within the C:\Users\Administrator\Documents\classified_doc_hashes.txt file. This file contains hashes of several classified documents that had been previously downloaded by Carl. First, you intend to pseudocode the script. Then, you intend to implement the pseudocode in PowerShell using a combination of Get-Content, Compare-Object, and Get-FileHash.
End State
Calculate a SHA256 hash for each file in Carl's Documents, Downloads, and Desktop folders.
Check each hash against every hash in the classified_doc_hashes.txt file.
Keep only the PS objects that match a hash in the classified_doc_hashes.txt file.
Output the name and hash for any file hashes that match one of the classified document hashes.
Report: Write a quick paragraph providing your findings and a suggested course of action.
Rubric
• Calculate hashes of all files in Carl's Documents, Downloads, and Desktop folders
• Retain only PS objects which match a hash listed in classified_doc_hashes.txt
• Use Compare-Object
• Output only the name and hash for retained objects
• Write a quick paragraph providing your findings and a suggested course of action

Exercise 3-13, Syntactical Exercise: Get-NetUDPEndpoint

Task
Retrieve a list of all active UDP endpoints on a system.
Filter that list to only those endpoint statistics with the host address as "0.0.0.0" or "::".
Note: This will retrieve a list of UDP endpoints opened by the system and listening on all available IP addresses, although other hosted endpoints may exist on specific IP addresses.
For each statistic in the resulting list, find the process which goes with the opened endpoint.
Rubric
• Retrieve active UDP endpoints with the host address as "0.0.0.0" or "::"
• Retrieve the process that opened each endpoint

Exercise 3-14, Syntactical Exercise: Get-NetTCPConnection

Background
You are an incident responder attempting to triage a system that may have been infected with a malware that allegedly opens a backdoor on the system in the form of a TCP port. You are aware of some ports commonly used by malware, to include ports 80, 443, 1090, 3389, 4444, 12345, 27374, 31337, and 65000. However, you would like to retrieve all TCP connections in the Listen state just to be certain. Furthermore, you would like to find the name of the process that opened each connection to assist in your triage.
Task
Retrieve a list of active TCP connections in the Listen state on the system.
Output only the port number and process name associated with each connection.
Rubric
• Retrieve a list of active TCP connections in the Listen state
• Output the connection's local port number and the name of the process that opened it

Exercise 3-15, Practical Exercise: Scripting, While, and Compare-Object

Background
You are a penetration tester trying to see any time that a user creates or terminates a process on an infected system ("PR Win10" VM). You would like to receive a notification via the command line (simply output to the screen) when a process is started or closed. However, you also would like the screen to remain uncluttered by the output (the output must not be excessive).
End State
Retrieve a list of running processes on the system.
Using an infinite While loop: • Retrieve all running processes on the system • Compare the current running processes with the previously retrieved process list
If there is a difference between the two, output that difference and update your process list.
Rubric
• Continuously retrieve running processes
• Output recently opened or closed processes
• Does not continuously output if the user is not using the system

Exercise 3-16, Practical Exercise (Advanced): Scripting, Export-ModuleMember, Import-Module, and Function

Background
You are an incident responder looking for indications of malware on a potentially infected system ("PR Win10" VM). You would like to see whether the system is running any services that are not on the Security Technical Implementation Guide (STIG) list. A list of legitimate services is available in the C:\Users\Administrator\Documents\stig_list.txt file. Furthermore, you would like to automate this task, because it is menial and needs to be run frequently. Therefore, you decide to make a module containing one function, which will retrieve all running services and check for any that are not on the stig_list.txt file.
End State
Create a function to perform the following: • Retrieve a list of running services • Check to see if each service is on the STIG list • If any service discovered was not on the STIG list, keep that service • Print out all running services not on the STIG list (stig_list.txt)
Store this function in a .ps1 file.
The .ps1 file must be able to be imported.
The function must be able to be called, once imported.
Report: Write a quick paragraph providing your findings and a suggested course of action.
Rubric
• Retrieve a list of running services
• Output only services that are not on the STIG list
• Make a function for the task
• Make a module containing this function
• Import the function from the module via PowerShell
• Write a quick paragraph providing your findings and a suggested course of action

Exercise 3-17, Practical Exercise (Advanced): Regular Expressions

Background
You are an incident responder looking for indications of a classified file on an unclassified system ("PR Win10" VM). You have no hash for the file you're searching for, but you do know that the data is encrypted, and how that encrypted text looks. The encrypted data will have 40 characters, and these characters will either be 0 through 9 or lowercase "a" through lowercase "f." In addition, the data will start with "171b6", and end with "5ee78". You intend to look through all of the files in Carl's Downloads and Documents folders.
End State
Search each file in those directories for a string beginning with "171b6", and ending with "5ee78".
For every file that matches, output the name and path of that file.
Report: Write a quick paragraph providing your findings and a suggested course of action.
Rubric
• Retrieve all files in Carl's Downloads and Documents folders
• Check each file for the specified pattern
• Output the name and path for each matching file
• Write a quick paragraph providing your findings and a suggested course of action

Exercise 3-18, Practical Exercise (Advanced): MSDN, Get-ItemProperty, PSDrives and PSProviders

Background
You are an incident responder looking for indications of persistence of malware in the registry of a system. You are aware that one of the most common places for persistence in the registry is in one of the "Run Keys," which are located as follows:
HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\CurrentVersion\Run
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run
You intend to check to see if these Run Keys contain indicators of malware persistence.
Task
Given: Different combinations of the following search terms will assist you in this task: MSDN, Get-ItemProperty, PSDrives, PSProviders

Exercise 4-1, Syntactical Exercise: Get-WmiObject and Select-Object

End State
List all of the classes in the Root\Microsoft\Windows\SMB namespace with "network" in their names, and expand the properties column to see all of the properties in each class.
Rubric
• Use the Get-WmiObject command
• List all of the classes with "network" in their names
• Limit the list to just the Root\Microsoft\Windows\SMB namespace
• Expand the properties column

Exercise 4-2, Syntactical Exercise: Get-WmiObject -Filter

End State
List all of the processes with a WorkingSetSize that is greater than 100MB and print only their Name, Handle, and WorkingSetSize.
Rubric
• Use the Get-WmiObject command with the -Filter flag
• List all processes with a working set greater than 100MB
• Print just the Name, Handle, and WorkingSetSize of the processes

Exercise 4-3, Syntactical Exercise: Invoke-WmiMethod

End State
Start a notepad process normally and use Invoke-WmiMethod to stop the process.
Rubric
• Use the Invoke-WmiMethod command
• Stop the notepad process that was started

Exercise 4-4, Syntactical Exercise: Get-CimInstance -Filter

End State
List all of the processes with a WorkingSetSize that is greater than 100MB and print only the Name, Handle, and WorkingSetSize.
Rubric
• Use the Get-CimInstance command with the -Filter flag
• List all processes with a working set greater than 100MB
• Print just the Name, Handle, and WorkingSetSize of the processes

Exercise 4-5, Syntactical Exercise: Invoke-CimMethod

End State
Start a notepad process normally and use Invoke-CimMethod to stop the process.
Rubric
• Use the Invoke-CimInstance command
• Stop the notepad process that was started

Exercise 4-6, Syntactical Exercise: Get-WmiObject or Get-CimInstance

End State
List all currently running services.
Rubric
• Use the Get-WmiObject or Get-CimInstance command
• Retrieve a list of services
• Filter list to include only services that are currently running

Exercise 4-7, Syntactical Exercise: Get-WmiObject or Get-CimInstance

End State
List all users on the machine. Determine the key property of the class that holds the user information.
Hint: Search for "user" classes.
Rubric
• Use the Get-WmiObject or Get-CimInstance command
• Retrieve a list of all users on the system
• Determine the key property of the class that contains the user information

Exercise 4-8, Practical Exercise (Advanced): USB Detection Using WMI

Background
You are a security engineer looking to augment your local defenses against infection via external media. You have decided to ban the use of USB drives in any of the computers on your network, hoping to safeguard against threats from rogue USB devices.
End State
Write a script to ring the system alarm continuously while any USB drives are plugged into the system.
Start by looking up what class contains information about disk drives on the system.
Hint: The system alarm can be sounded by printing "`a" to the console.
Rubric
• Use the Get-WmiObject or Get-CimInstance command
• Continuously search for a USB drive
• Ring the system alarm while a drive is inserted, stops when it is removed

Exercise 4-9, Practical Exercise: Starting Processes With WMI

Background
You are a system administrator who needs to be able to start multiple instances of certain programs in order to service the needs of your organization. On a normal day, you may be asked to start anywhere from 0 to 20 instances of any given application to support the users on the network. You have decided to automate this task to save you the tedious task of clicking on an icon over and over again every day.
End State
Write a script that prompts the user for the name of a process and a number of times to open this process.
Check to see that the supplied number is between 0 and 20 inclusive.
If the supplied number is between 0 and 20, start the process that number of times. Otherwise, print an error and exit.
Rubric
• Use the Get-WmiObject or Get-CimInstance command
• Prompt the user for a process name
• Prompt the user for a number, handling it as such
• Bound checks the number to ensure that it is between 0 and 20 inclusive
• Print an error if the number is out of bounds
• Start the supplied process the given number of times

Exercise 5-1, Syntactical Exercise: WMIC, Aliases

End State
Retrieve a list of local storage devices on the system.
Output must contain only the "core set" (or main) properties of the class representative of local storage devices.
Rubric
• Retrieve a list of local storage devices
• Output must contain only the "core set" of properties of the local storage devices' class

Exercise 5-2, Syntactical Exercise: WMIC, Aliases, Remoting

End State
Retrieve a list of local storage devices on a remote system ("PR Win10" VM).
Output must contain only the "core set" (or main) properties of the class representative of local storage devices.
Rubric
• Retrieve a list of local storage devices
• Output must contain only the "core set" of properties of the local storage devices' class

Exercise 5-3, Practical Exercise: WMIC, Remoting, Format

Background
You are a penetration tester looking for a way to create a custom exploit for a system ("PR Win10" VM). Therefore, you need to gather technical information on its hardware, specifically the processor, which you then will forward to your supporting exploit developers. They expect this technical information in a CSV format so they can easily access the data. You decide to use WMIC to retrieve this information. You know the class you need will be the Win32_Processor class, and the information (properties) you need will consist of the following: DeviceID, Name, Caption, AddressWidth, L2CacheSize, L3CacheSize, NumberOfEnabledCore, and NumberOfLogicalProcessors.
End State
Given: You know the class you would like to query: Win32_Processor
Given: You know the properties you need: DeviceID, Name, Caption, AddressWidth, L2CacheSize, L3CacheSize, NumberOfEnabledCore, and NumberOfLogicalProcessors
Retrieve required processor information from a remote host.
Use the given class and the given properties.
Output this information in CSV format and save it to a file.
Rubric
• Retrieve processor information from a remote host
• Retrieve only given properties from the given class
• Output information to a CSV file

Exercise 5-4, Syntactical Exercise: WMIC, Where

End State
Retrieve a list of local storage devices on a remote system ("PR Win10" VM).
Have the list contain only the instance that has a DeviceID of "C:"
Output this in a readable format.
Rubric
• Retrieve a list of local storage devices on a remote system
• Filter list to just the instance with the DeviceID of "C:"
• Output is in a readable format

Exercise 5-5, Syntactical Exercise: WMIC, Where

End State
Retrieve a set of specific running services from the system.
The running services' names should start with the letter "W" and should have a StartMode of Auto.
Output only the "Name" and "State" properties for those services.
Rubric
• Retrieve a list of running services with names beginning with "W"
• Output only the "Name" and "State" properties of those services

Exercise 5-6, Syntactical Exercise: PowerShell Analogs

End State
Redo Module 5 Exercise 4, but using PowerShell.
Retrieve a set of specific running services from the system.
The running services' names should start with the letter "W" and should have a StartMode of Auto.
Output only the "Name" and "State" properties for those services.
Rubric
• Use PowerShell
• Retrieve a list of running services with names beginning with "W"
• Output only the "Name" and "State" properties of those services

Exercise 5-7, Syntactical Exercise: WMIC, Call, Classes vs. Instances

End State
Using WMIC:
• Retrieve all running services whose names begin with the characters “sp”
• Stop those services
Rubric
• Use only WMIC
• Retrieve a list of running services with names beginning with “sp”
• Stop all services retrieved

Exercise 5-8, Syntactical Exercise: Call, Classes vs. Instances, PS Analogs

End State
Using only PS cmdlets that interact with WMI:
• Retrieve all running services whose names begin with the characters “sp”
• Stop those services.
Rubric
• Use only PS cmdlets that interact with WMI
• Retrieve a list of running services with names beginning with “sp”
• Stop all services retrieved