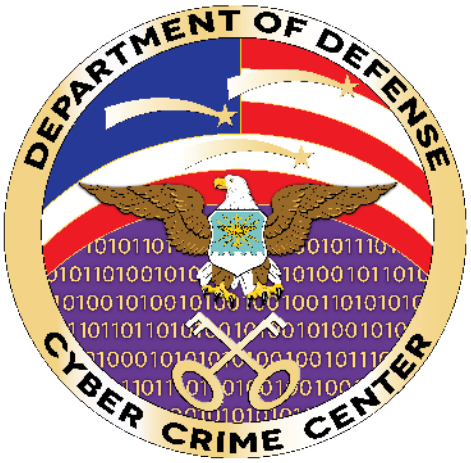
POWERSHELL FOR RESPONDERS (PR)

Resource Document



DCITA

A division of the DoD Cyber Crime Center

911 Elkridge Landing Road
Airport Square 11 Building, Suite 200
Linthicum, MD 21090
Tel: 410-981-1165
Fax: 410-850-8936
www.dc3.mil

[v.1801]

CONTENTS

PowerShell Cmdlets and Syntax

Lesson 1

PowerShell Syntax

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

Syntax and Verbs

PowerShell syntax is written in commands, called cmdlets (pronounced "command-lets"). These are two-word commands that use built-in scripts to grab data of a specified group and display the data in a logical format. Cmdlets use a verb-noun syntax. While a little cumbersome, they allow for extrapolation of commands. Some examples are as follows:

- 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.

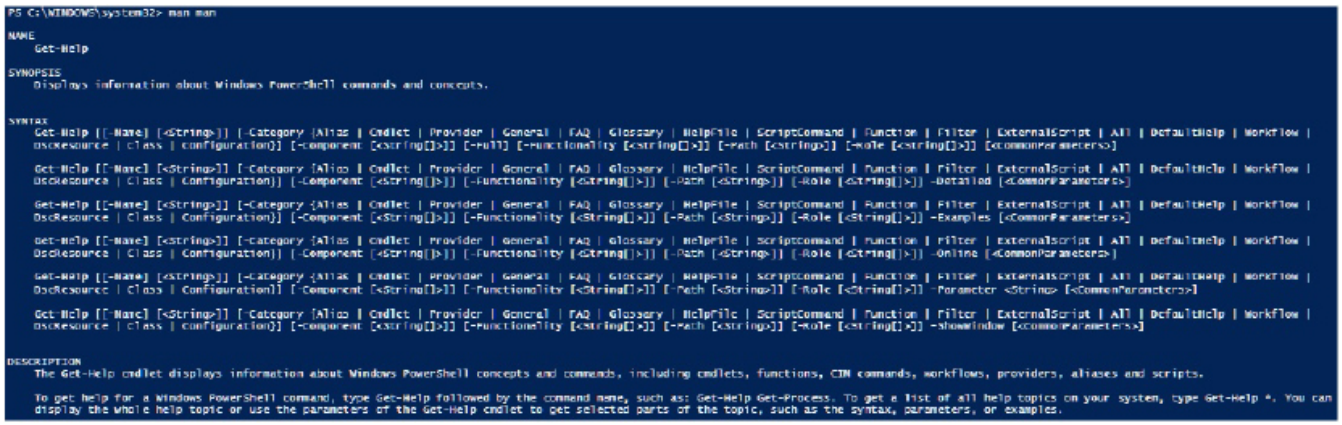
For additional syntax help, there are three PowerShell cmdlets that provide the user with information about other cmdlets. These are as follows:

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

Get-Help

Get-Help is a cmdlet that displays descriptions and parameter and return information for PowerShell cmdlets. Get-Help can list optional parameters, required parameters, what type of input to use in parameters, and other technical information regarding cmdlet use. Get-Help is also known as "man" in PowerShell, just like the cmdlet to retrieve cmdlet 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 with no cmdlet following the command.)

Alias

PowerShell also has what is called aliases. An alias is a shortened or abbreviated cmdlet that will perform the same function as the longer cmdlet. These exist to make scripting more efficient for the user. You might have already used an alias if you followed along on your computer, because "man" is just an alias for the "Get-Help" command. To retrieve a list of all aliases within PowerShell, use the cmdlet `Get-Alias`, or `gal` for short.

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

CommandType     Name                                               Version    Source
-----------     ----                                               -------    ------
Alias           % -> ForEach-Object
Alias           ? -> Where-Object
Alias           ac -> Add-Content
Alias           asnp -> Add-PSSnapin
Alias           cat -> Get-Content
Alias           cd -> Set-Location
Alias           CFS -> ConvertFrom-String                          3.1.0.0    Microsoft.PowerShell.Utility
Alias           chdir -> Set-Location
Alias           clc -> Clear-Content
Alias           clear -> Clear-Host
Alias           clhy -> Clear-History
Alias           cli -> Clear-Item
Alias           clp -> Clear-ItemProperty
Alias           cls -> Clear-Host
Alias           clv -> Clear-Variable
Alias           cnsn -> Connect-PSSession
Alias           compare -> Compare-Object
Alias           copy -> Copy-Item
Alias           cp -> Copy-Item
Alias           cpi -> Copy-Item
Alias           cpp -> Copy-ItemProperty
Alias           curl -> Invoke-WebRequest
Alias           cvpa -> Convert-Path
```

Get-Command

The final cmdlet we'll look at for syntactical help in PowerShell is Get-Command, or gcm for short. Get-Command will 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 "$" symbol. For example, "$test" could be a variable called "test" also.

Lesson 2

PowerShell Setup

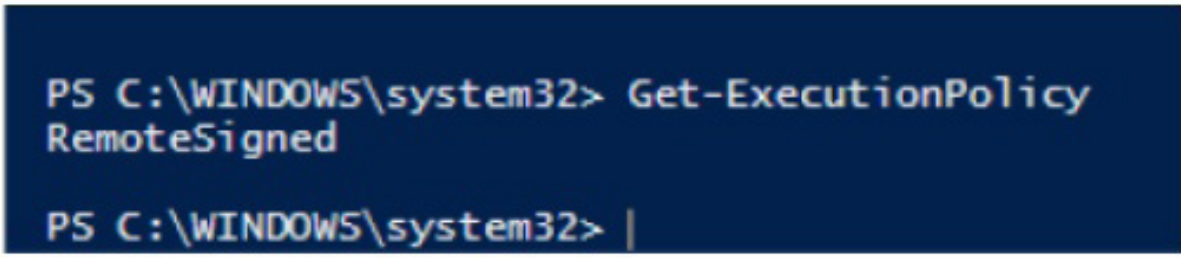
This lesson will show students how to set up PowerShell for designated tasks in the form of execution policies. These will establish how PowerShell is meant to interact with the various systems with which it will come in contact.

PowerShell Setup

PowerShell Initial Cmdlets – Execution Policies

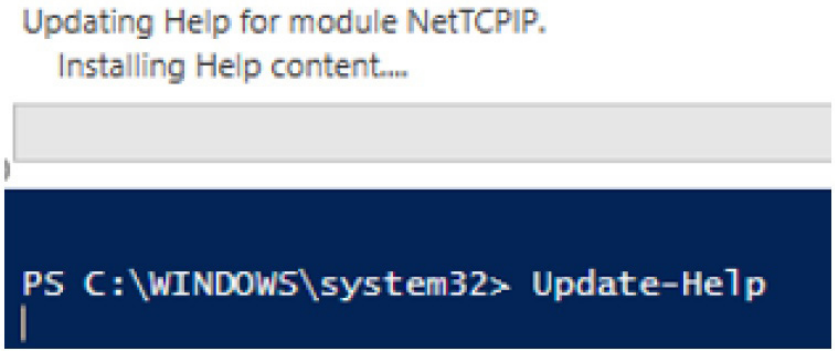
PowerShell has five types of execution policies, which are detailed as follows:

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	Any script can be run.
Bypass	Slightly worse than unrestricted, does not check Execution Policy before running scripts. To set the execution policy for PowerShell, you must run the following as an administrator: `Set-ExecutionPolicy RemoteSigned`



Update Help

It is important when running PowerShell to ensure that you update the help information for the Get-Help cmdlet. To update help, simply run Update-Help as an administrator.



Lesson 3

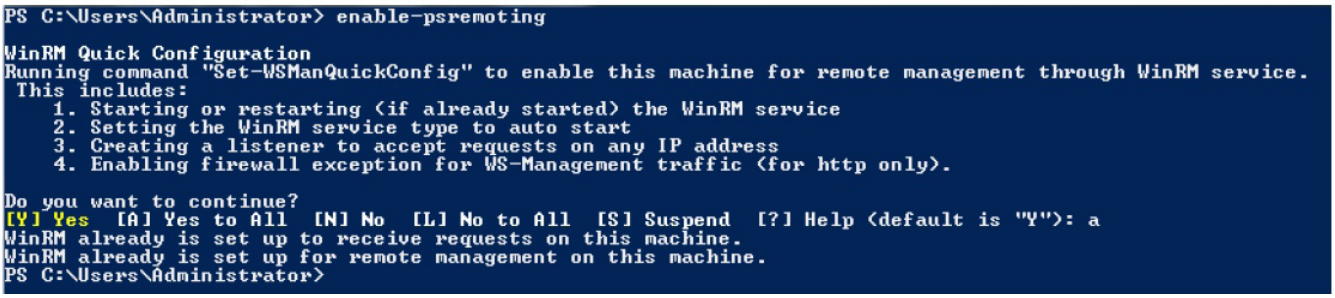
PowerShell Remote Execution

This lesson will show students how to begin to establish remote execution of cmdlets using PowerShell.

Remote PowerShell Administration

An important part of using PowerShell as a responder is to be able to remote into other machines to run commands. To do so, the user must establish a remote connection. To perform simple remoting, cmdlet `Enable-PSRemoting` must be run on the local machine in order to use PowerShell as an administrator on the remote system. The cmdlet configures the remote computer to receive remote PS cmdlets. To execute this, type the following command on the localhost:

```
Enable-PSRemoting
```



The next vital cmdlet, `New-PSSession`, creates a persistent "remote" session on localhost (the same box). If this cmdlet is run successfully on the remote host, remoting is verified. Enter the following command on the remote host in Powershell:

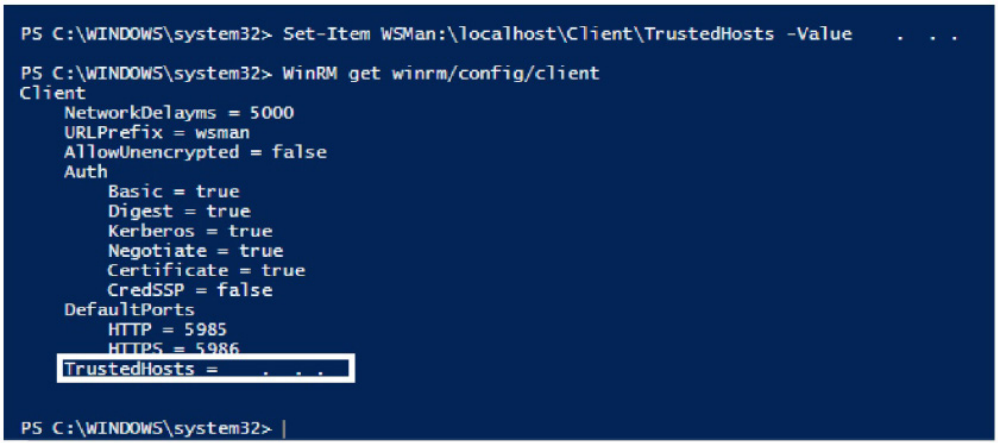
```
New-PSSession
```

The next step is to prepare the local system to remote into the remote host. The first step in this process is to ensure that PowerShell is being run as an administrator. The following cmdlet will be used on the local host to have the computer trust specified hosts to receive sensitive credentials. Type the following command on the local host:

```
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). To verify that the remote connection was established, use the following cmdlet, also on the local computer:

```
WinRM get winrm/config/client
```



To get credentials to send to the remote machine, use the following command on the local computer:

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

From there, you may input the username and password of the remote user. To enter the session, the command `Enter-PSSession X.X.X.X -Credential $cred` should be used.
Likewise, the command `Exit-PSSession X.X.X.X -Credential $cred` should be used to exit the session.

```
PS C:\WINDOWS\system32> $cred = Get-Credential
cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:

PS C:\WINDOWS\system32> Enter-PSSession             -Credential $cred

[           ]: PS C:\Users\Administrator\Documents> Exit-PSSession

PS C:\WINDOWS\system32>
```

Lesson 4

PowerShell Commands and Cmdlets

As stated in the previous module, PowerShell has an extended reach in comparison to CMD. To exemplify this, we can look at 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  DcomLauncher       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 some special characters used in PowerShell that can help you search for or filter certain data. We'll look at two special characters that can serve this purpose now. The first is the special character "*" (asterisk, also known as 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. What's more is that the asterisk represents 0 or more of any character. For example, "*" could represent "`aaa`", "`something entirely too long`" or "" (nothing). As an example, in PowerShell, typing the command `Get-Service Win*` will retrieve all services starting with Win and ending with anything (including any services simply named `Win`).

A useful tool for used for searching and filtering is the special character "?" (question mark). The question mark can be used in PowerShell as another type of wildcard – one that represents only a single character. As another example, type the following in PowerShell:

```
Get-Service d??svc
```

This command will retrieve all services starting with the "d" character, followed by any two characters (denoted by "??"), and ending with the "svc" characters. As shown below, whether you use a single character wildcard or a multi-character wildcard, your results may contain data for which you did not intend to search or filter, which is an important tip to note when using wildcards.

```
PS C:\Users\Administrator> gsv d??svc

Status   Name               DisplayName
------   ----               -----------
Stopped  DcpSvc             DataCollectionPublishingService
Stopped  DsmSvc             Device Setup Manager


PS C:\Users\Administrator> gsv d*svc

Status   Name               DisplayName
------   ----               -----------
Stopped  DcpSvc             DataCollectionPublishingService
Stopped  defragsvc          Optimize drives
Stopped  DmEnrollmentSvc    Device Management Enrollment Service
Running  DoSvc              Delivery Optimization
Stopped  dot3svc            Wired AutoConfig
Stopped  DsmSvc             Device Setup Manager
Running  DsSvc              Data Sharing Service
```

As the user, you can also search for and through lists of cmdlets. For example, `Get-Command *service*` outputs the following:

```
[           ]: PS C:\Users\Administrator\Documents> Get-Command *service*

CommandType     Name                                               Definition
-----------     ----                                               ----------
Application     api-ms-win-service-core-l1-1-0.dll                 C:\Windows\system32\api-ms-win-service-core-l
Application     api-ms-win-service-management-l1-1-0.dll           C:\Windows\system32\api-ms-win-service-manage
Application     api-ms-win-service-management-l2-1-0.dll           C:\Windows\system32\api-ms-win-service-manage
Application     api-ms-win-service-winsvc-l1-1-0.dll               C:\Windows\system32\api-ms-win-service-winsvc
Application     AuxiliaryDisplayServices.dll                       C:\Windows\system32\AuxiliaryDisplayServices.
Application     auxiliarydisplayservices.mof                       C:\Windows\System32\Wbem\auxiliarydisplayserv
Cmdlet          Get-Service                                        Get-Service [[-Name] <String[]>] [-ComputerNa
Application     IconCodecService.dll                               C:\Windows\system32\IconCodecService.dll
Cmdlet          New-Service                                        New-Service [-Name] <String> [-BinaryPathName
Cmdlet          New-WebServiceProxy                                New-WebServiceProxy [-Uri] <Uri> [[-Class] <S
Application     OpcServices.dll                                    C:\Windows\system32\OpcServices.dll
Cmdlet          Restart-Service                                    Restart-Service [-Name] <String[]> [-Force] [
Cmdlet          Resume-Service                                     Resume-Service [-Name] <String[]> [-PassThru]
Application     ServiceModel.mof                                   C:\Windows\System32\Wbem\ServiceModel.mof
Application     ServiceModel.mof.uninstall                         C:\Windows\System32\Wbem\ServiceModel.mof.uni
Application     ServiceModel35.mof                                 C:\Windows\System32\Wbem\ServiceModel35.mof
Application     ServiceModel35.mof.uninstall                       C:\Windows\System32\Wbem\ServiceModel35.mof.u
Application     services.exe                                       C:\Windows\system32\services.exe
Application     services.mof                                       C:\Windows\System32\Wbem\services.mof
Application     services.msc                                       C:\Windows\system32\services.msc
Cmdlet          Set-Service                                        Set-Service [-Name] <String> [-ComputerName <
Cmdlet          Start-Service                                      Start-Service [-Name] <String[]> [-PassThru]
Cmdlet          Stop-Service                                       Stop-Service [-Name] <String[]> [-Force] [-Pa
Cmdlet          Suspend-Service                                    Suspend-Service [-Name] <String[]> [-PassThru
Application     themeservice.dll                                   C:\Windows\system32\themeservice.dll
Application     WcsPlugInService.dll                               C:\Windows\system32\WcsPlugInService.dll
Application     webservices.dll                                    C:\Windows\system32\webservices.dll
Application     WPDShServiceObj.dll                                C:\Windows\system32\WPDShServiceObj.dll
Application     WPDShServiceObj.mof                                C:\Windows\System32\Wbem\WPDShServiceObj.mof
Application     XpsRasterService.dll                               C:\Windows\system32\XpsRasterService.dll
Application     xpsservices.dll                                    C:\Windows\system32\xpsservices.dll
```

Another use of special characters in PowerShell is to denote variables. As a note, some special characters will interfere with running both CMD and PowerShell commands properly, so it is important to ensure that you are using the special character correctly. In PowerShell, the character "`$`" (dollar sign) denotes a variable. For example, to create a variable named "`x`" and assign it a value of 10, the command would look like the following: `$x = 10`. You could simply run the command `echo $x` to output the value of x. In addition, the combination "`$_`" (dollar sign, underscore) denotes a special variable in PowerShell that we will discuss later in this 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 that are used to modify the current working environment. With the `WhatIf` flag, the user can verify the intended action of the cmdlet without modifying the environment (like a dry run). Once the intended result is verified, the `WhatIf` option can be removed allowing the cmdlet to execute normally.

Also, `WhatIf` is always implied if the `WhatIf` environmental variable is set to the following:

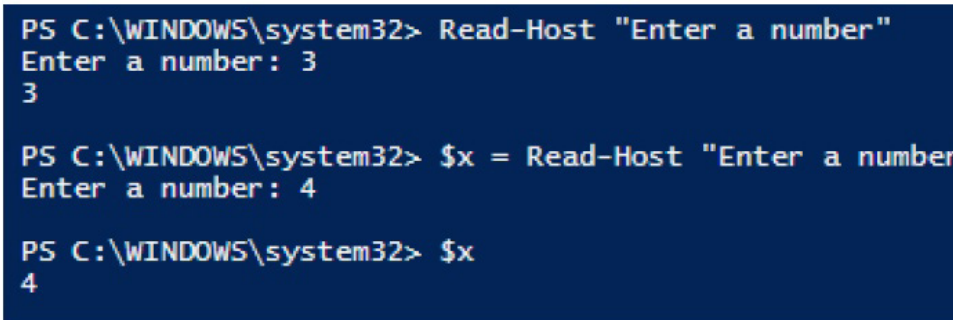
```
true$WhatIfPreference = $true
```

- Run the command `dir variable:\` to be certain that WhatIf is true. (lists all variables)
- Type: `Stop-Computer` to be even more certain (This will shut down the local host unless in `WhatIf` mode.)

```
PSSessionOption                 System.Management.Automation.Remoting.PSSessionOption
PSUICulture                     en-US
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> |
```

Read-Host

Read-Host is able to receive command line input. For example, Read-Host "Enter a number" will ask the user to enter a number. When a number (or 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. As an important note, however, 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.



That note leads us to consider datatypes, which are concepts that help PowerShell determine how to handle 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.

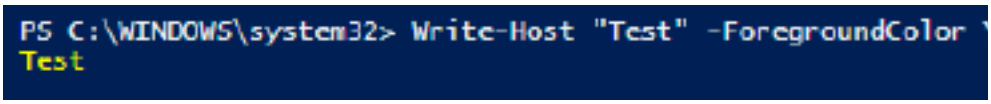
Datatypes	
`int`	Short for integer, meaning a whole number, for example, 5, 6, 7, but not 5.5, 3.14, 7¾.
`string`	Uses text as text, and nothing more, for example, "Test", "Hello, World!", "Fifty-six" are text. "56" is also read as text.
`float`	Short for floating point number, meaning a decimal point number, such as 5.5, 3.14, 7.75. Also compatible with 5, 6, 7.

Write-Host

Write-Host outputs text to the command line. It is similar to the ECHO command, but is done in the style of PowerShell. For example, try the following:

```
Write-Host “Test” -ForegroundColor Yellow
```

This is useful for warnings, errors, or highlighting items of importance with PowerShell.

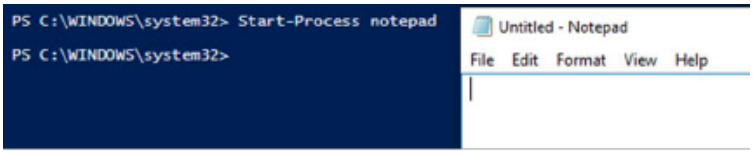


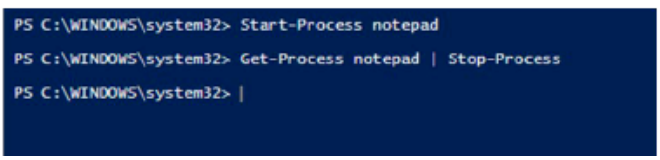
Special Characters in PowerShell

| = Piping

(pipe)

The pipe character enables the output of one command to be used as the input of another. This concept is what makes command-line scripting powerful in its own right. The chain of commands made using pipes can be referred to as a “pipeline.” The act of chaining commands via pipes can be referred to as “piping,” and chained commands can be referred to as “piped.” Let’s prepare for an example in PowerShell by running the following: `Start-Process notepad`. This command opens the Notepad application. The command `Get-Process notepad | Stop-Process` will close the Notepad application (along with any other instances of Notepad that may be open, so be careful). This is done because you are asking PowerShell to retrieve the process of Notepad, then to pipe that process into the `Stop-Process` cmdlet, which stops the retrieved process.





. = Access Object Members

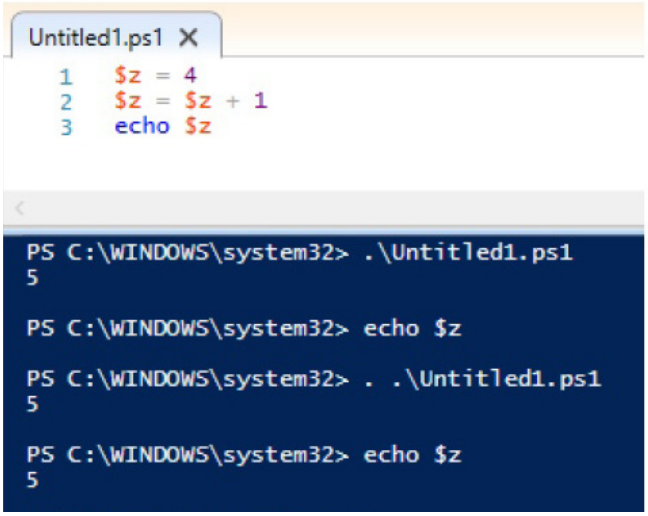
(dot or period)

The dot, or period, can be used to access object members, such as properties or methods. The syntax for this use of the dot is as follows: $VARIABLE.PROPERTY. To set up an example of this in PowerShell, run the following: `$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 in PowerShell:

```
$Service.DisplayName
```

Another use of the dot character in PowerShell is called dot sourcing. Dot sourcing adds functions, aliases, and variables of a script to the current PowerShell script or session. For example, when using the following command using dot sourcing, we can see the outcome below:

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



As you can see, this allows a user to call any variable from the script after it finishes.

( ) = Grouping Expression

(parentheses)

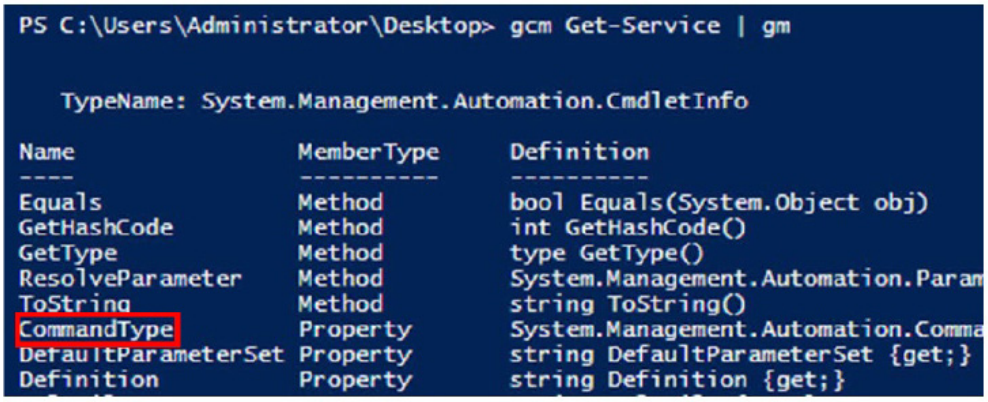
The parentheses are special characters in PowerShell that can be used for different tasks in different contexts. The first context we'll discuss is when parentheses are used to surround a command. In this context, the parentheses as a whole act as a variable containing the output of the

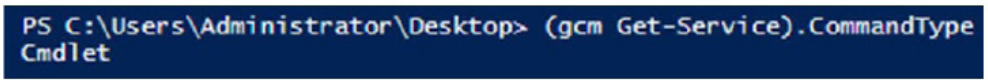
enclosed command. In this way, members (to include properties and methods) of the enclosed command can be accessed just as a variable that holds the command's output object. To start our example, we'll first browse the members of the Get-Command output using the following command:

```
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 feel free to try one of your choosing. Finally, we'll run the following command:

```
(gcm Get-Service).CommandType
```





The second context in which to use parentheses is arithmetic. Similar to a mathematical string, the grouping expression gives precedence to cmdlets within the parentheses. The following will demonstrate the grouping process within PowerShell.

```
$x = 4
($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.

= Comment Output

(hashtag or pound)

"#" specifies that everything following it, until the end of the line, should not be interpreted as a command. This allows for comments to be placed after commands and is most useful in scripts, especially when they become rather lengthy. 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
```

` = Escape Character

(backtick, not apostrophe)

The escape character is a powerful character that allows the programmer to use the next character outside of its normal PowerShell context. Try typing: `echo `$x` to see how the escape character works within PowerShell.

On the same lines of using a character differently, the backtick character will have special implementations when used in conjunction with specific characters. See below for just two of these special uses.

`` `n (newline character) `` Type: `echo "This is a`ntest"`

`` `t (tab) `` Type: `echo "This is`t`ta test"`

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

PS C:\WINDOWS\system32> echo "This is a`ntest"
This is a
test

PS C:\WINDOWS\system32> echo "This is`t`ta test"
This is     a test

PS C:\WINDOWS\system32>
```

As you can see above, normally the command echo $x would yield the value of the "x" variable. However, when using the escape character, it will return $x because the escape character ignores the "$" symbol. In the next cases, the backtick is used with specific characters that have special uses relating to formatting.

; = Denotes end of command

(semicolon)

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

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

```
PS C:\WINDOWS\system32> echo (5 + 4); echo Test;
9
Test

PS C:\WINDOWS\system32>
```

Additional Special Characters

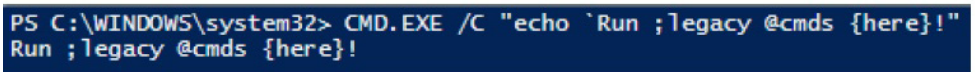
The following characters have special meaning in PowerShell. Thus, it is important to either avoid using them or use the backtick (`) to use them without causing unexpected results.

- @ (at symbol)
- :: (double colon)
- , (comma)
- { } (curly brackets)
- [ ] (brackets)

For any CMD commands that use PS special characters in their arguments, type:

```
CMD.EXE /C “echo `Run ;legacy @cmds {here}!"
```

This method passes the command to CMD without parsing the special characters.



Cmdlets

Get-Member

Get-Member is a cmdlet that enumerates an object or cmdlet’s properties and methods. The Get-Member cmdlet allows the user to know what they can or cannot access, and which properties and methods are available. It also 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 above, 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 discrete pieces of information about the object to which they belong.

Get-Package

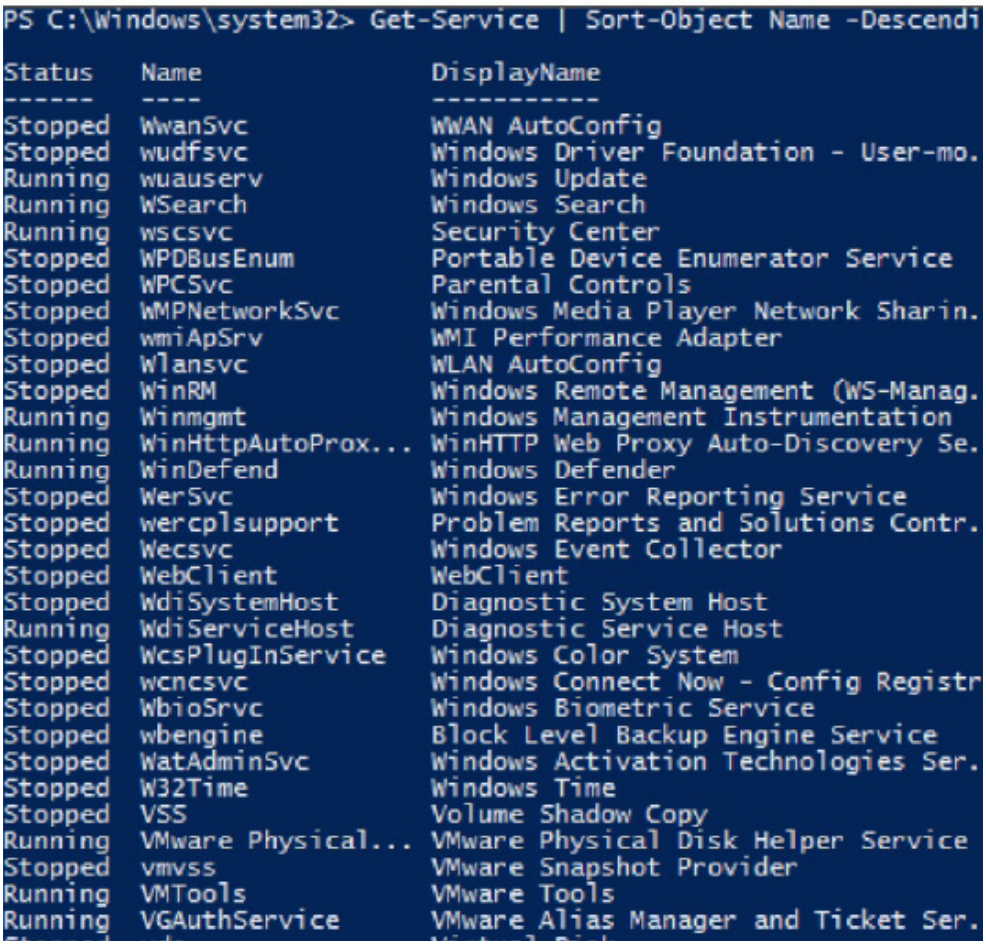
`Get-Package` is a useful cmdlet that retrieves a list of installed software and Windows updates on the system.

```
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
```



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
```

As an example of the customization options available with `Select-Object`, the above script will output three processes with the longest CPU time in seconds.

PowerShell Scripting Command If Statements: If, ElseIf, Else

Similar to other programming languages, `if` statements can be used to put specific restrictions or contingencies on how or why 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	Determine elements in a group
-notcontains	Determine excluded elements in a group

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

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

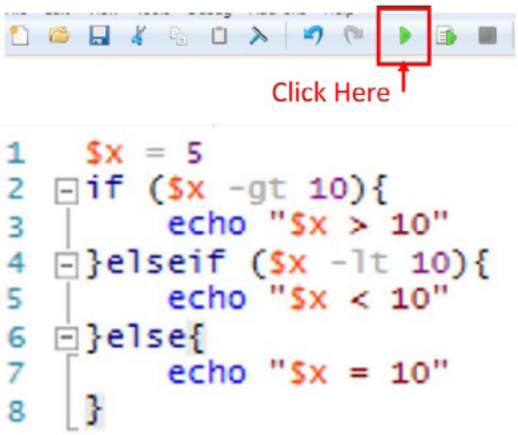
The conditional expression contained in the if statement is what is evaluated. If the condition is `True`, the script contained within the braces (code block) that follow will be executed, and the `else` statement and its block of code will be skipped. If the condition is `False`, PowerShell will skip 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 next portion of the `if` statement we'll learn about 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, there is the possibility that no `if` or `elseif` script block will run at all.

A better way to write `if` statements is in the script pane, located at the top of PowerShell ISE. This allows you to write your script line by line, which makes it more easily readable. In addition, because `if` statements are often used in scripts rather than in one-line PowerShell commands, it makes sense to practice writing `if` statements within the script pane. 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.}
```




```
PS C:\Windows\System32> $x = 5
if ($x -gt 10){
    echo "$x > 10"
}elseif ($x -lt 10){
    echo "$x < 10"
}else{
    echo "$x = 10"
}
5 < 10
```

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
```

Where-Object

The `Where-Object` cmdlet will check each row of data for a specified condition, only returning that row if the condition is met. Think of `Where-Object` as a filter.

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

`$_` is a variable that holds special meaning in PowerShell. It holds a single object taken from the pipeline. In the example above, `$_` is shown to hold 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. See the next screenshot or try it for yourself to visualize this description.

```
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

PowerShell allows you to describe data sizes without writing out the whole numeric value of the size. For example, 1KB refers to 1 kilobyte (for the real conversion, lookup kibibyte). However, it can also refer to 1024 bytes. In PowerShell, different properties can be measured in different units. As an example, the `VirtualMemorySize` property of a process is measured in megabytes, while the `WorkingSet` size (physical memory size) is measured in kilobytes. Because data sizes are measured in different units in different contexts, PowerShell allows you to simply use `1KB` or `5MB` or `20GB` to refer to 1 kilobyte, 5 megabytes, or 20 gigabytes, respectively. From there, PowerShell will check your unit and compare it with the unit of the interesting property and compare the actual values. By default, PowerShell operates in bytes, so when a numeric value is used to describe data size without specifying units, bytes is assumed to be the unit of data size. In summation, the syntax of byte conversions is as follows: `[NUMBER][BYTE ID]`. Byte identifiers are as follows:

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

Type the following in PowerShell to view this conversion in action:

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

Compare-Object

This cmdlet will compare one collection of objects against another collection of objects for differences, 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)
```

As with other commands, it is up to the user to specify what collection is to be referenced. By default, the comparison will only output the differences between the two collections, while indicating which collection contains the different objects. For this reason, we'll look at two options that allow comparisons to show matching objects instead.

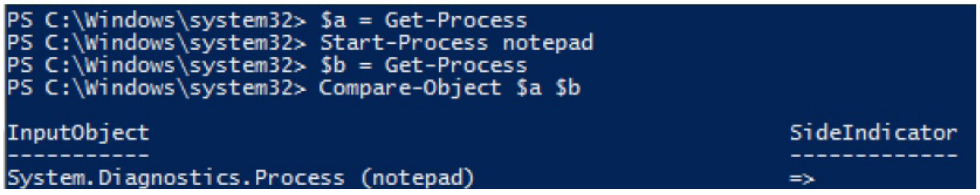
`-IncludeEqual` – Instead of only outputting differences, the output includes the objects that are equal

`-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

As an example, type the following commands in PowerShell, one at a time (hit return after each line):

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

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



In the above example, the SideIndicator shows the result of the Compare-Object command. Basically, it is stating that the InputObject, in this case, System.Diagnostics.Process (notepad) is present in the $b file but not the $a file. The symbols used in the SideIndicator are as follows:

=> (equals sign, right arrow)	Item was added The InputObject is present in the difference (second) file, but not in the reference (first) file.

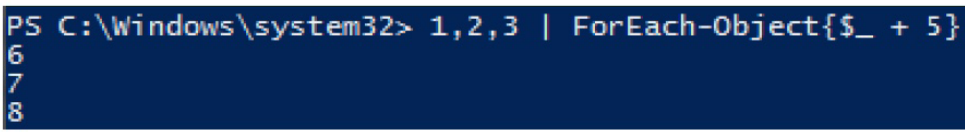
<= (left arrow, equals sign)	Item was removed Present in reference (first) file, but not in the difference (second) file.
== (double equals sign)	Present in both files Try adding the -IncludeEqual parameter, and notice the == underneath SideIndicator in the output.

ForEach-Object

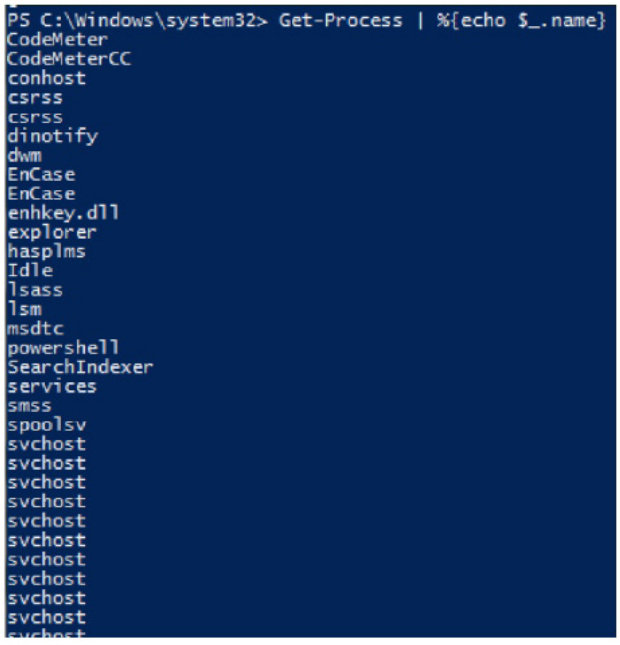
This looping cmdlet allows you to run commands against (for) each object in a collection. The alias for this command is % (percent sign) and can be seen in action in the example below.

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}}
```

```
PS C:\WINDOWS\system32> Get-Process | %{if ($_.CPU -gt 10){Write-Host $_.name}}
MsMpEng
```

Get-ChildItem

This cmdlet is similar to `DIR` from CMD, but this cmdlet returns a PowerShell object, which means it can be used in looping and have properties and methods of its own. There are a few options of importance with `Get-ChildItem`, which are identified below.

`-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]` – Lists files with a specified attribute; comma-separate multiple attributes
Some noteworthy properties, accessed in looping cmdlets via the $_ variable along with a dot $_. (dollar sign, underscore, dot) are as follows:

- `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
```

Get-Content

Get-Content, 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 options 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, using this property might output 4 since the file has 4 lines. For example:

```
(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 above code first 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.

Get-FileHash

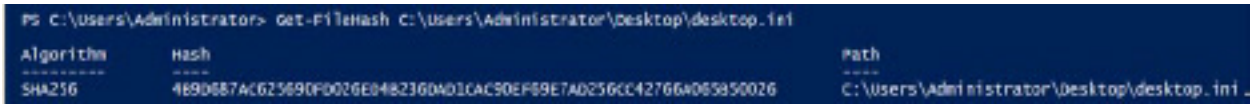
This cmdlet generates a hash of a specified file using a specified hashing algorithm. A useful option of `Get-FileHash` is the following:

- -Algorithm – Specified algorithm to use for file hash with the following parameters:
  - MD5
  - SHA1
  - SHA256 – Default algorithm

Properties of output object (typically accessed with $_.) are:

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

For an example of `Get-FileHash` in PowerShell, type the following:
`Get-FileHash C:\Users\Administrator\Desktop\desktop.ini`



`Get-NetUDPEndpoint` will retrieve active UDP endpoint (stream) statistics. As a note, because UDP is a connectionless protocol, we will not refer to these as connections. Some options of `this` cmdlet are as follows:

`-LocalPort [PORT]` – Filters results by local port number or a comma-separated list for multiple ports

`-LocalAddress [IP]` – Filters results by the address used locally or comma-separated list of values

- 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

A property of output object (typically accessed with `$_`.) is:
`OwningProcess` – Contains the process ID of the process that opened the endpoint

To visualize the command's output, type: `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
```

Get-NetTCPConnection

This cmdlet will retrieve active TCP connection statistics. Some options of Get-NetTCPConnection are as follows:

`-LocalPort [PORT]` – Filters results by local port number or a comma-separated list for multiple ports

`-RemotePort [PORT]` – Filters results by remote port number or a comma-separated list for multiple ports

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

Properties of output object (typically accessed with `$_.`) are:

- `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, type: `Get-NetTCPConnection`

```
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
```

For

This cmdlet will run a loop for a specific number of times. The syntax is important in this cmdlet as the user needs to specify specific value and characters to run the for cmdlet successfully.

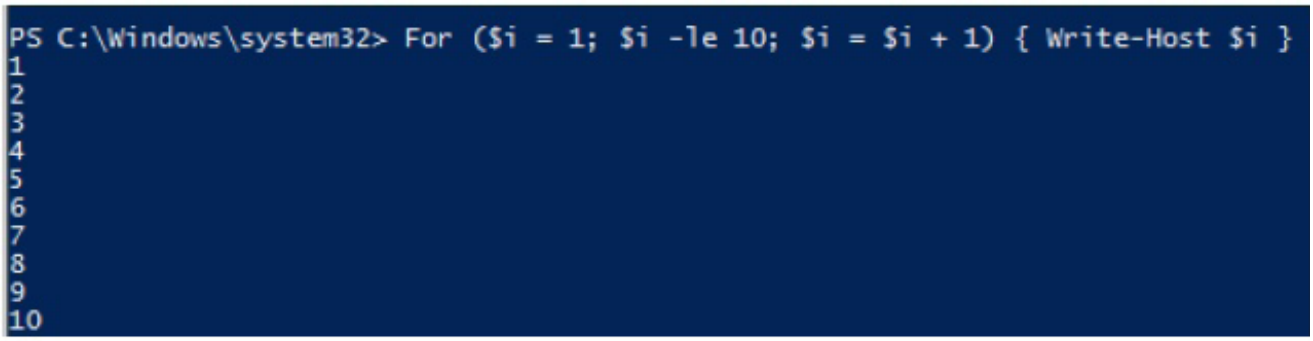
Syntax:

```
For (<ASSIGN VARIABLE>; <CONDITION>;
     <INCREMENT VARIABLE>){ <SCRIPT>;
}
```

This will run `SCRIPT` until `CONDITION` is false.

For example:

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



While

This cmdlet will run a loop until a specific condition is met. The conditions are supplied by the user and, if not carefully done, the script may loop continuously. It is critical to ensure that you have the right conditions in place to avoid this.

Syntax:

```
While (<CONDITION>){
     <SCRIPT>;
}
```

Runs `SCRIPT` 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
```

Function

A function is a group of commands that work together to perform a specific task or set of tasks. The `Function` cmdlet allows us give a name to this grouping of commands, allowing its re-use simply by calling the assigned name as if it were another cmdlet.

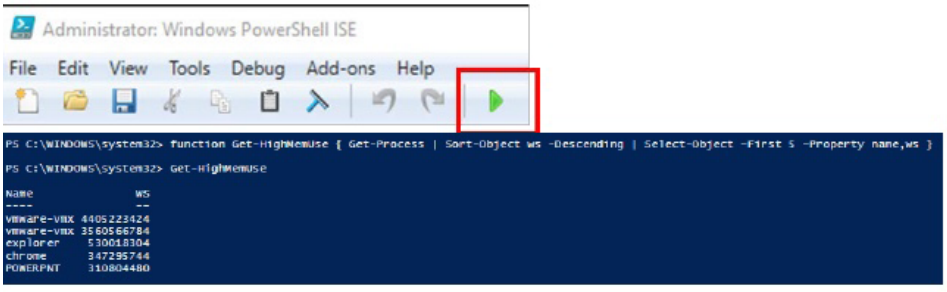
Syntax:

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

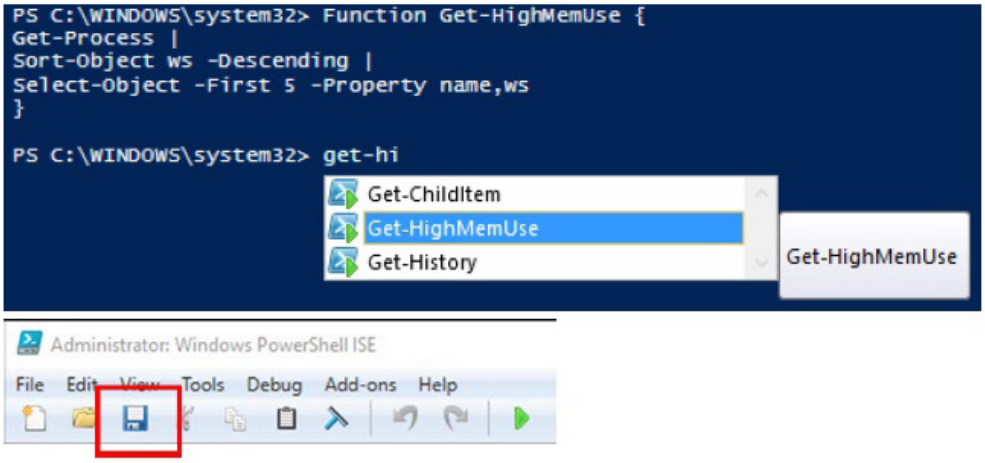
The function will be available for the duration of the script or session. It will be able to be aliased and auto-completed, just like other cmdlets. For example, consider needing to write a function to output running processes using the most RAM:

```
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
}
```



You would be able to test the function by typing `Get-Hi<TAB>` where `<TAB>` is the physical key press of 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 a memorable hard disk location. Recall that, to retain variables and functions defined in scripts during a PowerShell session, Dot Sourcing is required.



Export-Module

This will allow the user to export functions, variables, and more from a module and will build into the import function. It 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 options:

`-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 will turn 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-*"
```

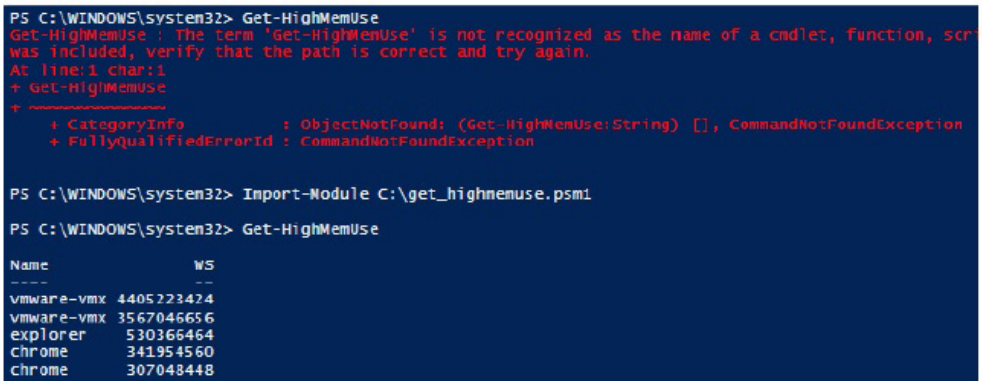
This line will export any function in the module whose name begins with Get-, which is a convenience for the scripter.

Import-Module

This critical cmdlet allows the user to import 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.

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

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



Regular Expressions

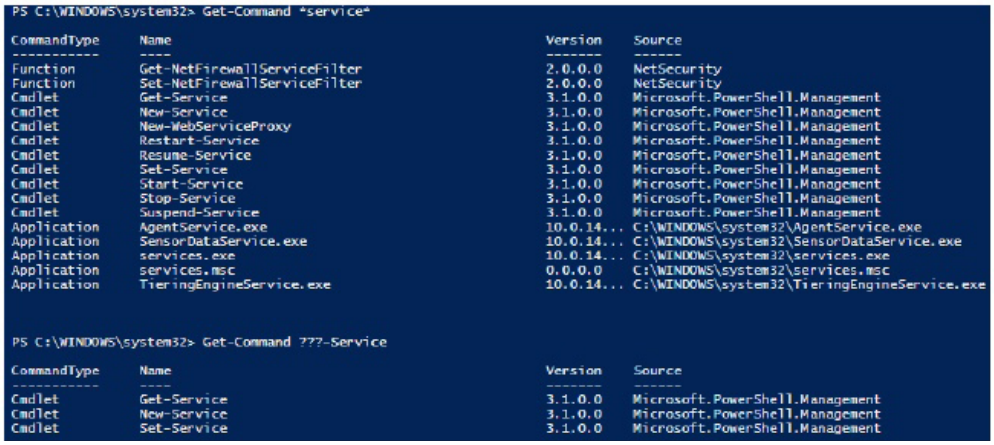
Regular Expression (regex) is a method of searching for data, such as a string of text, when the structure of the data (string) or a part of the data (string) is known. For starters, we'll consider the following: `"This is a test" -match "test"`. In this instance, the `-match` operator is taking in the regex `test`.

`This is a test` is then checked to see if it contains the following string of characters: `test`. The string test within `This is a test` is considered a substring, and is what the regex is searching for.

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

Recall that the "*" (asterisk) can be used as a multi-character wildcard when used with PowerShell cmdlets. For example, `Get-Command *service*` will search for all cmdlets containing "service" in the name. While the asterisk is very useful for searching, it is not being used here as part of regex. The same is true of the single character wildcard "`?`" (question mark) in PowerShell cmdlets. For example, `Get-Command ???-service` would retrieve all cmdlets with three characters preceding `-service`, but this would still not be using regex, only PowerShell wildcards.

The `-match` operator does not recognize PowerShell wildcards; instead it uses regular expressions, which we will cover next.



We'll only briefly cover regex in this course, and our discussion will be limited to the operators below. For more practice with regex, feel free to take Log Analysis (LA).

`-match` – Case-insensitive ("test" matches "Test") regex matching For example, `"This is a test" -match "test"` will search the string "`This is a test`" for the string "`test`"

`-cmatch` – Case-sensitive ("test" does not match "Test") regex matching

Special Characters:

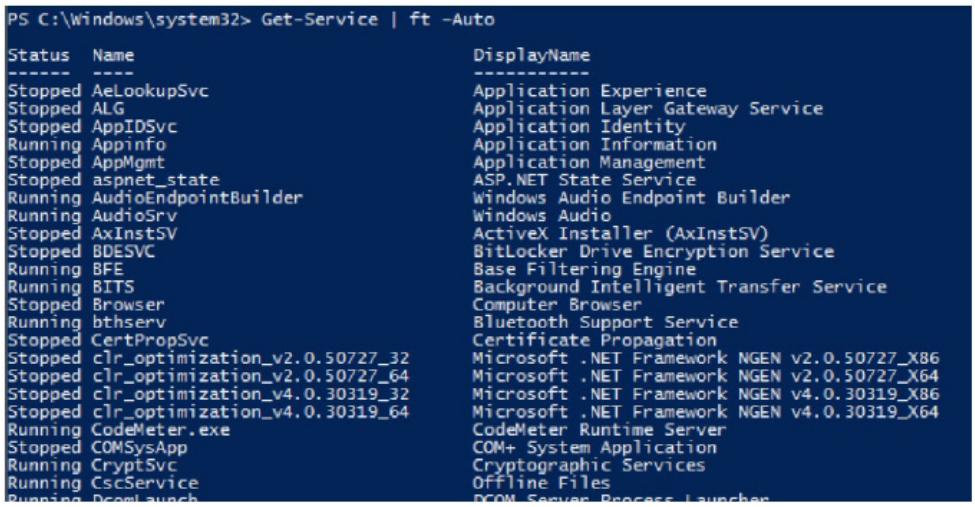
- * (asterisk) – Represents repeating 0 or more of the preceding character; for example, A* matches any number of consecutive A's
- . (dot) – Single character wildcard (represents 1 character); for example, the combination ".*" matches any number of any characters

\ (backslash) – Escape character can be used to search for "." or "*" as in `"4 * 3.4" –match "4 \* 3\.4"`

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

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` option 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

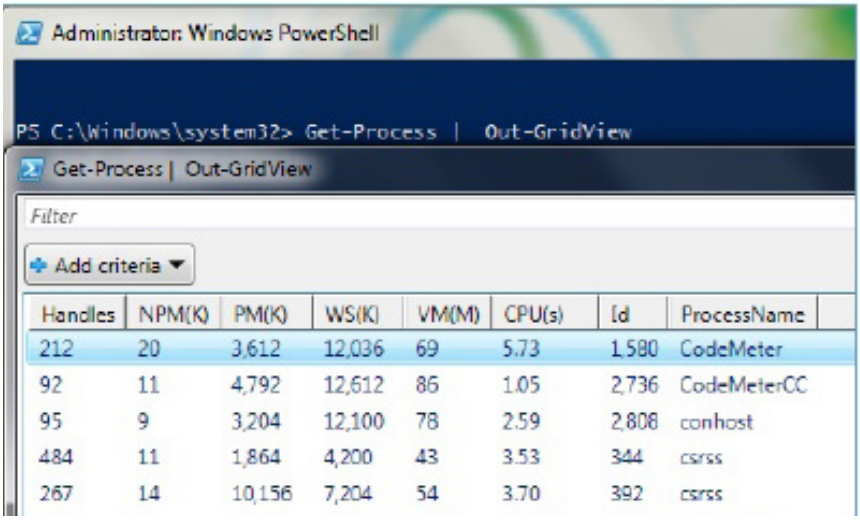
`Format-Custom`, alias fc, will output according to a specified format. This format can be specified with the -View option, but we won't go into custom formatting, as that is a class of its own. However, we will use the default Format-Custom view to get a glimpse at the class and instance layout that PowerShell uses to see services by running the following command:
`Get-Service | fc`

For more information on custom formatting, see the Microsoft Developer Network page titled "How to Create a Formatting File [.format.ps1xml]" at https://msdn.microsoft.com/en-us/library/dd878339(v=vs.85).aspx

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

class ServiceController
{
  Status = Stopped
  Name = AeLookupSvc
  DisplayName = Application Experience
}

class ServiceController
{
  Status = Stopped
  Name = ALG
  DisplayName = Application Layer Gateway Service
}

class ServiceController
{
  Status = Stopped
  Name = AppIDSvc
  DisplayName = Application Identity
}

class ServiceController
{
  Status = Running
  Name = Appinfo
  DisplayName = Application Information
}

class ServiceController
```

Out-GridView

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



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 a common registry location.

Discover/Change Services on Startup

WMIC

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

```
wmic service 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
```

- Important options:
  - `-DisplayName <DISPLAYNAME>` This option is required.
  - `-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 PID from `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

Command	Alias
`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

Operator	Definition
"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

WMIC

```
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` or `Select` – Retrieve objects (columns) from a collection
  - E.g.: `Get-Process | select name,ws`
  - To expand a contracted list: `<…> | select -ExpandProperty <PROPERTYNAME>`
- `Compare-Object` or `Compare` – Compare all objects in two collections
  - E.g.: `compare $a $b`
- `Where-Object` or `?` – Check each row of data for a specified condition
  - E.g.: `Get-Process | ?{$_.name -match "cmd"}`
- `ForEach-Object` or `%` – Run commands against each item in a collection
  - E.g.: `Get-Process | %{Write-Host $_.name,$_.ws}`
- `Select-String` or `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` or `ft` – Normal formatting, but can expand contracted items
  - E.g.: `Get-Service | ft –auto`
- `Format-List` or `fl` – List properties and property names for each instance
  - E.g.: `Get-Service | fl`
- `Format-Custom` or `fc` – Format as classes and instances
  - E.g.: `Get-Service | fc`
- `Out-GridView` or `ogv` – Format as a filterable table in a new window
  - E.g.: `Get-Service | Out-GridView`

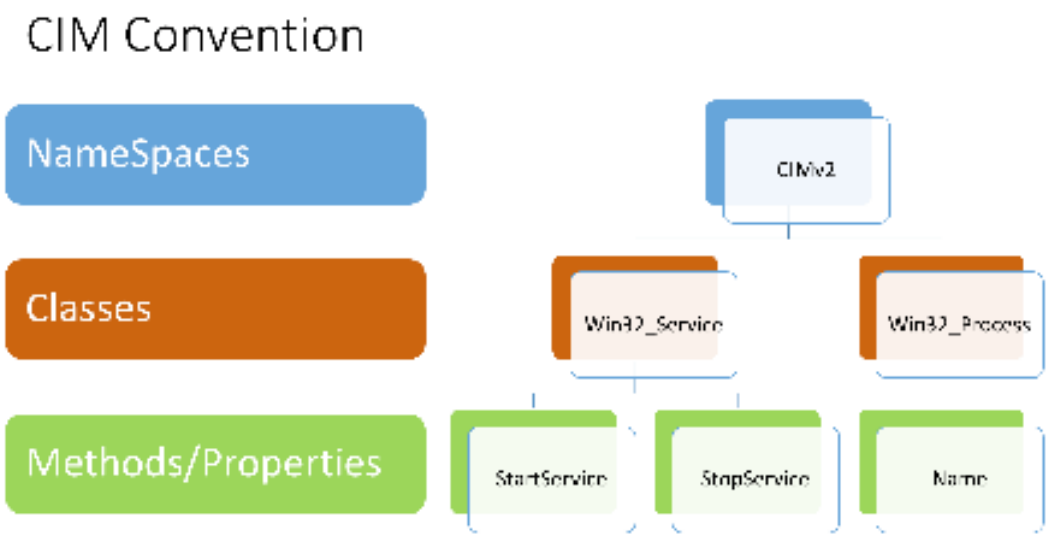
PowerShell Condition Operators

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 tracingfiles
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

- `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 of Filesystem	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/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