Introduction to Windows PowerShell Fundamentals

Windows Server 2012

Hands-on lab

Windows PowerShell is a command-line shell and scripting language that helps IT professionals achieve greater control and productivity. Using a new, admin-focused scripting language, more than 230 standard command-line tools, and consistent syntax and utilities, Windows PowerShell allows IT professionals to more easily control system administration and accelerate automation.

Produced by HynesITe, Inc.
Version 4.0
1/3/2013

Introduction

Estimated time to complete this lab

30 minutes

Objectives

In this lab, you will learn the fundamentals of using Windows PowerShell commands (known as cmdlets), including the following techniques:

- Exploring Windows PowerShell Help

  Windows PowerShell has a powerful documentation mechanism. Administrators can query the help subsystem with a unified command set. Developers are provided with a set of common tools to lower the time invested in documentation.

- Constructing a pipeline

  Windows PowerShell is different from other shells because it does not use strings as parameters; instead it uses .NET objects which can be navigated, processed, reflected, and formatted.

- Using formatting commands

  Windows PowerShell does not limit the kind of formatting that can be applied to a simple object nor does it place any restrictions on the destination of the output. Developers can extend the wide range of available choices through the development of cmdlets.

- Using filtering and sorting commands

  As a complement to formatting, filtering and sorting commands are very useful for cmdlet output manipulation. Windows PowerShell provides typical filtering and sorting cmdlets for most tasks, and developers can also extend them by creating new cmdlets.

- Using -WhatIf and -Confirm

  These are common switches you can apply to cmdlets.

- Working with variables

  Like any Windows-based scripting language, Windows PowerShell has variables too, but they are much more powerful than the variables in older scripting languages. Windows PowerShell variables are actually mapped to underlying classes in the Microsoft® .NET Framework. And in the Framework, variables are objects, meaning they can store data and also manipulate it in many ways.

- Working with providers

  The provider represents a set of stored data (e.g. the Microsoft Windows Registry, the Windows file system, Active Directory) that can be accessed and navigated through Windows PowerShell commands.

Prerequisites

Before working on this lab, you must have:

- An understanding of concepts such as virtual machines, virtual hard disks, and virtual networks.
- The ability to work in a command-line environment.

Overview of the lab

Windows PowerShell is a command-line shell and scripting language that helps IT professionals achieve greater control and productivity. Using a new admin-focused scripting language, more than 230 standard command-line tools, and consistent syntax and utilities, Windows PowerShell allows IT professionals to more easily control system administration and accelerate automation.

Intended audience

This lab is intended for network administrators who wish to learn the Windows PowerShell interface and language.

Virtual machine technology

This lab is completed using virtual machines that run on Windows Server 2012 Hyper-V technology. To log on to the virtual machines, press CTRL+ALT+END and enter the following credentials:

- Username: **Administrator**
- Password: **Passw0rd!**

Computers in this lab

This lab uses computers as described in the following table. Before you begin the lab, you must ensure that the virtual machines are started and then log on to the computers.

Virtual Machine	Role
DC	Domain Controller, the client for this lab

All user accounts in this lab use the password **Passw0rd!**

Note regarding pre-release software

Portions of this lab may include software that is not yet released, and as such may still contain active or known issues. While every effort has been made to ensure this lab functions as written, unknown or unanticipated results may be encountered as a result of using pre-release software.

Note regarding user account control

Some steps in this lab may be subject to user account control. User account control is a technology which provides additional security to computers by requesting that users confirm actions that require administrative rights. Tasks that generate a user account control confirmation are denoted using a shield icon. If you encounter a shield icon, confirm your action by selecting the appropriate button in the dialog box that is presented.

Exercise 1: Exploring Windows PowerShell

In this exercise, you will explore several Windows PowerShell commands and features including help, object formatting, and safety features.

Note: Every command has three different levels of help available:

1. The **default** view shows the command description and syntax.
2. The **detailed** view adds usage examples and complete documentation.
3. The **full** view adds command’s technical details including parameter and return value data types.

Reviewing the help available in Windows PowerShell

In this step, you will learn how to view the different levels of help content available for a cmdlet.

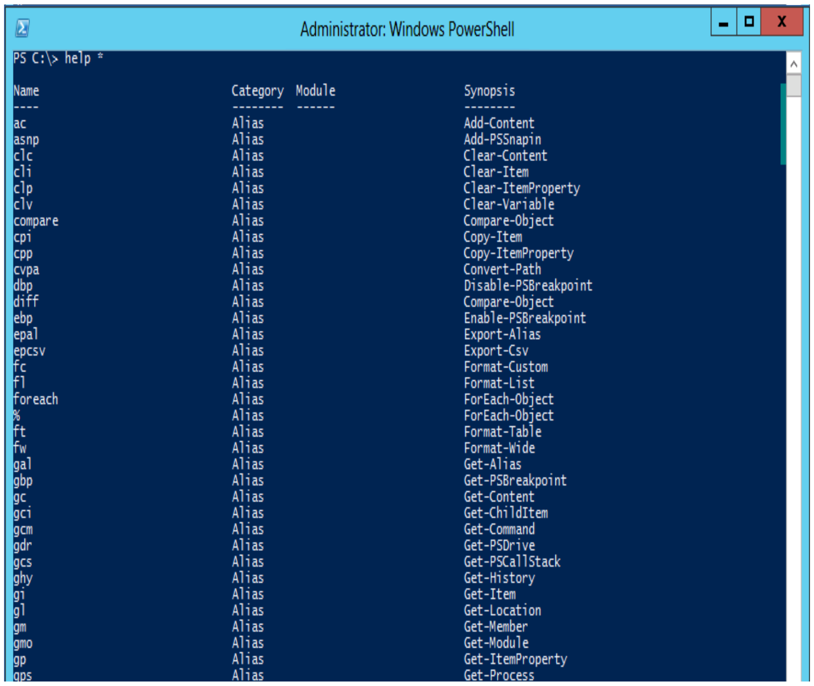
1. To open a new Windows PowerShell command window, on the taskbar, click **Windows PowerShell**.

   Note: When you first open Windows PowerShell it will automatically start loading all available modules. As you begin to type it will try to figure out which module you need. This can cause a delay which can last a couple minutes.

2. At the Windows PowerShell command prompt, type the following command, and then press ENTER to see a list of available help topics.

   ```
   help *
   ```

3. The command will fill an entire screen and then pause. Press ENTER to show the next output line, or press SPACE BAR to advance to the next page.



```
PS C:\> help *

Name                             Category  Module                    Synopsis
----                             --------  ------                    --------
ac                               Alias                               Add-Content
asnp                             Alias                               Add-PSSnapin
clc                              Alias                               Clear-Content
cli                              Alias                               Clear-Item
clp                              Alias                               Clear-ItemProperty
clv                              Alias                               Clear-Variable
compare                          Alias                               Compare-Object
cpi                              Alias                               Copy-Item
cpp                              Alias                               Copy-ItemProperty
cvpa                             Alias                               Convert-Path
dbp                              Alias                               Disable-PSBreakpoint
diff                             Alias                               Compare-Object
ebp                              Alias                               Enable-PSBreakpoint
epal                             Alias                               Export-Alias
epcsv                            Alias                               Export-Csv
fc                               Alias                               Format-Custom
fl                               Alias                               Format-List
foreach                          Alias                               ForEach-Object
%                                Alias                               ForEach-Object
ft                               Alias                               Format-Table
fw                               Alias                               Format-Wide
gal                              Alias                               Get-Alias
gbp                              Alias                               Get-PSBreakpoint
gc                               Alias                               Get-Content
gci                              Alias                               Get-ChildItem
gcm                              Alias                               Get-Command
gdr                              Alias                               Get-PSDrive
gcs                              Alias                               Get-PSCallStack
ghy                              Alias                               Get-History
gi                               Alias                               Get-Item
gl                               Alias                               Get-Location
gm                               Alias                               Get-Member
gmo                              Alias                               Get-Module
gp                               Alias                               Get-ItemProperty
gps                              Alias                               Get-Process
```

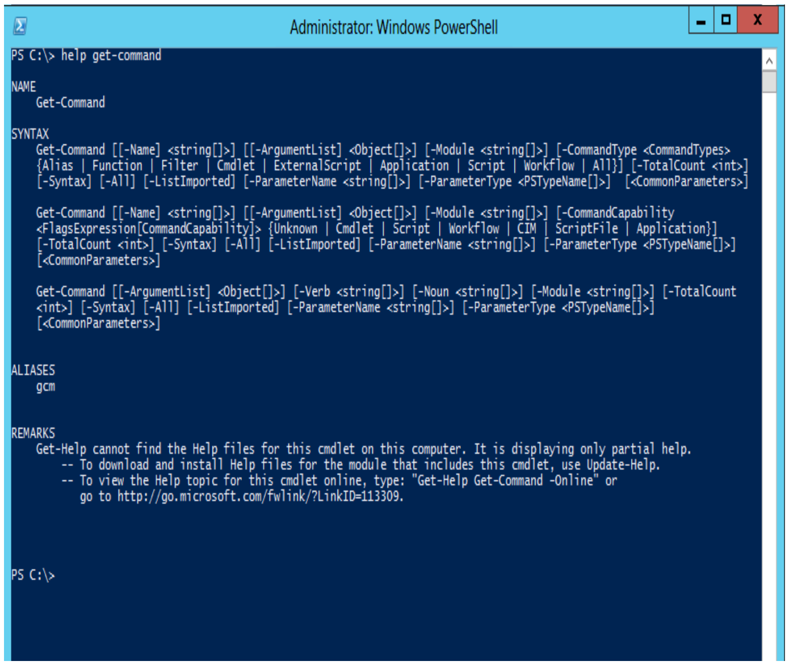
Figure 1: Executing the Help command

NOTE: In the output you can see a second column with a value of **Alias**, **Cmdlet**, **Provider**, or **HelpFile**. An alias is an alternative name for a command, usually an abbreviation or a name used by other shells for similar functionality. Cmdlets are .NET classes that are exposed as APIs, commands, and GUIs. Providers are extensions to Windows PowerShell that provide features such as policy verification or metadata augmentation. Help files contain information for different topics, including command descriptions and topics on how to extend Windows PowerShell.

4. Press SPACE BAR until the command prompt returns. Alternatively, you can type the letter **Q** to cancel the output.
5. To view help information about the **Get-Command** cmdlet, at the Windows PowerShell command prompt, type the following command, and then press ENTER.

```
help Get-Command
```

The help contains the syntax for the command as well as a brief description.



Administrator: Windows PowerShell

```
PS C:\> help get-command

NAME
    Get-Command

SYNTAX
    Get-Command [[-Name] <string[]>] [[-ArgumentList] <Object[]>] [-Module <string[]>] [-CommandType <CommandTypes>
    {Alias | Function | Filter | Cmdlet | ExternalScript | Application | Script | Workflow | All}] [-TotalCount <int>]
    [-Syntax] [-All] [-ListImported] [-ParameterName <string[]>] [-ParameterType <PSTypeName[]>]  [<CommonParameters>]

    Get-Command [[-Name] <string[]>] [[-ArgumentList] <Object[]>] [-Module <string[]>] [-CommandCapability
    <FlagsExpression[CommandCapability]> {Unknown | Cmdlet | Script | Workflow | CIM | ScriptFile | Application}]
    [-TotalCount <int>] [-Syntax] [-All] [-ListImported] [-ParameterName <string[]>] [-ParameterType <PSTypeName[]>]
    [<CommonParameters>]

    Get-Command [[-ArgumentList] <Object[]>] [-Verb <string[]>] [-Noun <string[]>] [-Module <string[]>] [-TotalCount
    <int>] [-Syntax] [-All] [-ListImported] [-ParameterName <string[]>] [-ParameterType <PSTypeName[]>]
    [<CommonParameters>]


ALIASES
    gcm


REMARKS
    Get-Help cannot find the Help files for this cmdlet on this computer. It is displaying only partial help.
        -- To download and install Help files for the module that includes this cmdlet, use Update-Help.
        -- To view the Help topic for this cmdlet online, type: "Get-Help Get-Command -Online" or
           go to http://go.microsoft.com/fwlink/?LinkID=113309.



PS C:\>
```

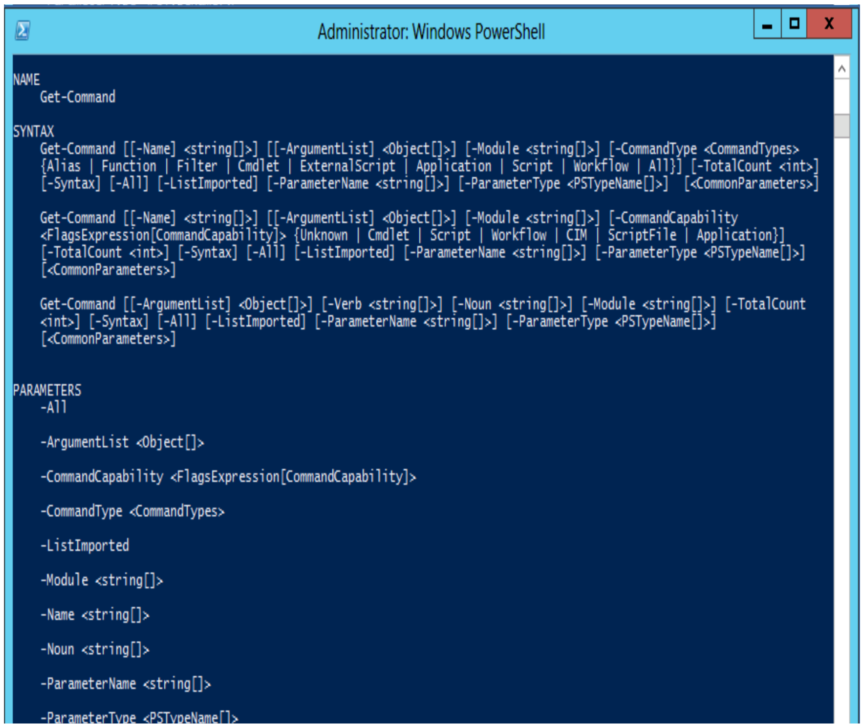
Ge

Figure 2: Obtaining help for the Get-Command cmdlet

6. To see detailed help for the **Get-Command** cmdlet, at the Windows PowerShell command prompt, type the following command, and then press ENTER.

```
↪ help Get-Command -Detailed
```

The output includes details about the parameters for the cmdlet, as well as some examples.

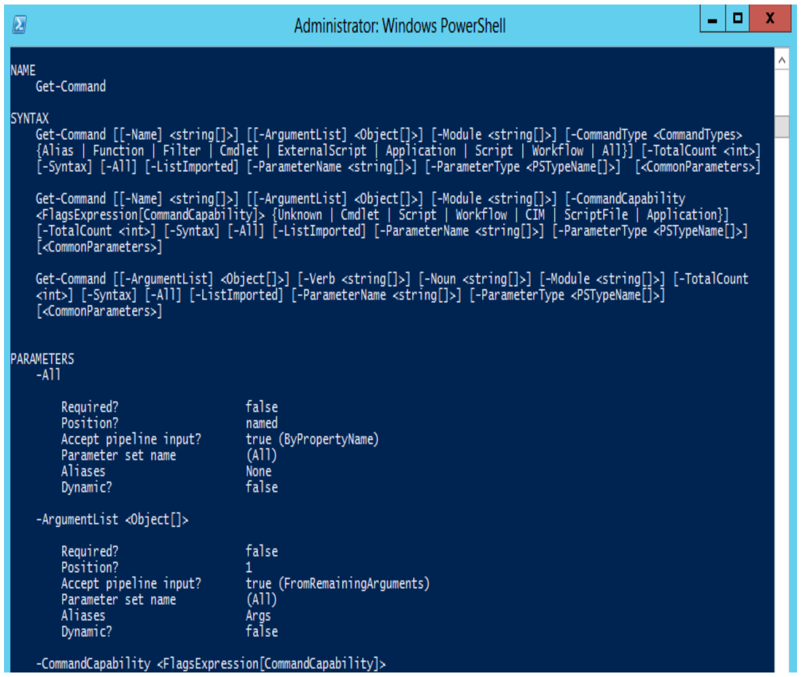


```
NAME
    Get-Command

SYNTAX
    Get-Command [[-Name] <string[]>] [[-ArgumentList] <Object[]>] [-Module <string[]>] [-CommandType <CommandTypes>
    {Alias | Function | Filter | Cmdlet | ExternalScript | Application | Script | Workflow | All}] [-TotalCount <int>]
    [-Syntax] [-All] [-ListImported] [-ParameterName <string[]>] [-ParameterType <PSTypeName[]>]  [<CommonParameters>]

    Get-Command [[-Name] <string[]>] [[-ArgumentList] <Object[]>] [-Module <string[]>] [-CommandCapability
    <FlagsExpression[CommandCapability]> {Unknown | Cmdlet | Script | Workflow | CIM | ScriptFile | Application}]
    [-TotalCount <int>] [-Syntax] [-All] [-ListImported] [-ParameterName <string[]>] [-ParameterType <PSTypeName[]>]
    [<CommonParameters>]

    Get-Command [[-ArgumentList] <Object[]>] [-Verb <string[]>] [-Noun <string[]>] [-Module <string[]>] [-TotalCount
    <int>] [-Syntax] [-All] [-ListImported] [-ParameterName <string[]>] [-ParameterType <PSTypeName[]>]
    [<CommonParameters>]


PARAMETERS
    -All

    -ArgumentList <Object[]>

    -CommandCapability <FlagsExpression[CommandCapability]>

    -CommandType <CommandTypes>

    -ListImported

    -Module <string[]>

    -Name <string[]>

    -Noun <string[]>

    -ParameterName <string[]>

    -ParameterType <PSTypeName[]>
```

Figure 3: Obtaining help for the Get-Command cmdlet in detailed mode

7. To view the entire help content for the **Get-Command** cmdlet, at the Windows PowerShell command prompt, type the following command, and then press ENTER.

```
↪ help Get-Command –Full
```



```
NAME
    Get-Command

SYNTAX
    Get-Command [[-Name] <string[]>] [[-ArgumentList] <Object[]>] [-Module <string[]>] [-CommandType <CommandTypes>
    {Alias | Function | Filter | Cmdlet | ExternalScript | Application | Script | Workflow | All}] [-TotalCount <int>]
    [-Syntax] [-All] [-ListImported] [-ParameterName <string[]>] [-ParameterType <PSTypeName[]>]  [<CommonParameters>]

    Get-Command [[-Name] <string[]>] [[-ArgumentList] <Object[]>] [-Module <string[]>] [-CommandCapability
    <FlagsExpression[CommandCapability]> {Unknown | Cmdlet | Script | Workflow | CIM | ScriptFile | Application}]
    [-TotalCount <int>] [-Syntax] [-All] [-ListImported] [-ParameterName <string[]>] [-ParameterType <PSTypeName[]>]
    [<CommonParameters>]

    Get-Command [[-ArgumentList] <Object[]>] [-Verb <string[]>] [-Noun <string[]>] [-Module <string[]>] [-TotalCount
    <int>] [-Syntax] [-All] [-ListImported] [-ParameterName <string[]>] [-ParameterType <PSTypeName[]>]
    [<CommonParameters>]


PARAMETERS
    -All

        Required?                    false
        Position?                    named
        Accept pipeline input?       true (ByPropertyName)
        Parameter set name           (All)
        Aliases                      None
        Dynamic?                     false

    -ArgumentList <Object[]>

        Required?                    false
        Position?                    1
        Accept pipeline input?       true (FromRemainingArguments)
        Parameter set name           (All)
        Aliases                      Args
        Dynamic?                     false

    -CommandCapability <FlagsExpression[CommandCapability]>
```

Figure 4: View the entire help content for the Get-Command cmdlet

- NOTE: The full help for the cmdlet includes parameter data types and notes. This is a technical view of the command's help.
- NOTE: Help regarding conceptual topics in Windows PowerShell begins with the prefix **about_**. To display help about a Windows PowerShell concept, type **Get-Help** followed by the concept name. To view a list of conceptual topics type **Get-Help about**.

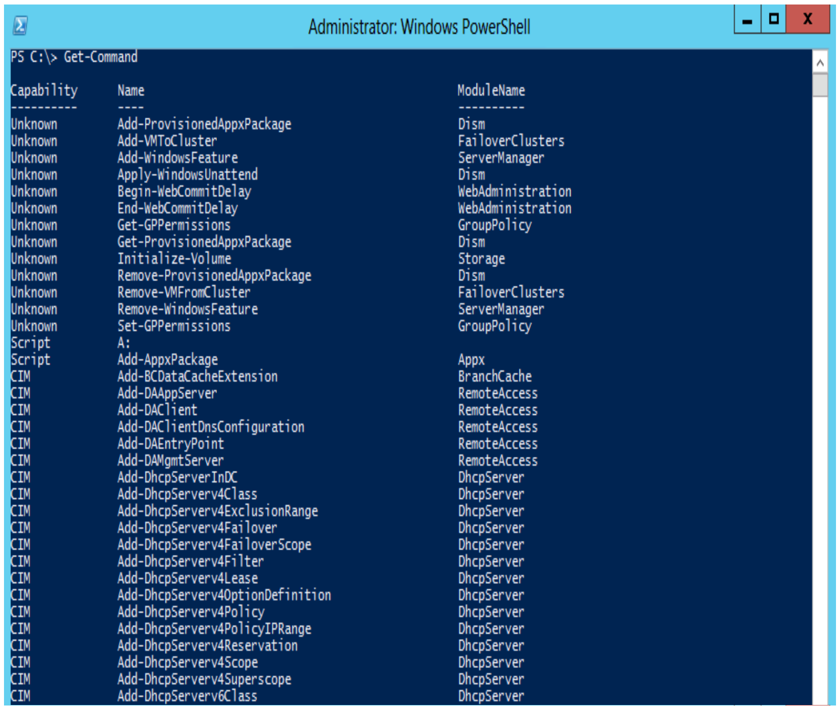
List the commands available in Windows PowerShell

In this step, you will list all available commands in Windows PowerShell.

1. To view the list of available commands, at the Windows PowerShell command prompt, type the following command, and then press ENTER.

```
↪ Get-Command
```

2. Review the list of commands available. Take note of the naming convention for commands with a CommandType of **Cmdlet**.



Administrator: Windows PowerShell

```
PS C:\> Get-Command

Capability    Name                                    ModuleName
----------    ----                                    ----------
Unknown       Add-ProvisionedAppxPackage              Dism
Unknown       Add-VMToCluster                         FailoverClusters
Unknown       Add-WindowsFeature                      ServerManager
Unknown       Apply-WindowsUnattend                   Dism
Unknown       Begin-WebCommitDelay                    WebAdministration
Unknown       End-WebCommitDelay                      WebAdministration
Unknown       Get-GPPermissions                       GroupPolicy
Unknown       Get-ProvisionedAppxPackage              Dism
Unknown       Initialize-Volume                       Storage
Unknown       Remove-ProvisionedAppxPackage           Dism
Unknown       Remove-VMFromCluster                    FailoverClusters
Unknown       Remove-WindowsFeature                   ServerManager
Unknown       Set-GPPermissions                       GroupPolicy
Script        A:
Script        Add-AppxPackage                         Appx
CIM           Add-BCDataCacheExtension                BranchCache
CIM           Add-DAAppServer                         RemoteAccess
CIM           Add-DAClient                            RemoteAccess
CIM           Add-DAClientDnsConfiguration            RemoteAccess
CIM           Add-DAEntryPoint                        RemoteAccess
CIM           Add-DAMgmtServer                        RemoteAccess
CIM           Add-DhcpServerInDC                      DhcpServer
CIM           Add-DhcpServerv4Class                   DhcpServer
CIM           Add-DhcpServerv4ExclusionRange          DhcpServer
CIM           Add-DhcpServerv4Failover                DhcpServer
CIM           Add-DhcpServerv4FailoverScope           DhcpServer
CIM           Add-DhcpServerv4Filter                  DhcpServer
CIM           Add-DhcpServerv4Lease                   DhcpServer
CIM           Add-DhcpServerv4OptionDefinition        DhcpServer
CIM           Add-DhcpServerv4Policy                  DhcpServer
CIM           Add-DhcpServerv4PolicyIPRange           DhcpServer
CIM           Add-DhcpServerv4Reservation             DhcpServer
CIM           Add-DhcpServerv4Scope                   DhcpServer
CIM           Add-DhcpServerv4Superscope              DhcpServer
CIM           Add-DhcpServerv6Class                   DhcpServer
```

Figure 5: List of Windows PowerShell commands

- NOTE: Windows PowerShell uses verb-noun naming conventions to make cmdlets discoverable and obvious in what they do.

3. At the command prompt, type the following command, but do not press ENTER.

```
↪ Get-C
```

4. Press TAB. Windows PowerShell expands the command to **Get-CAAuthorityInformationAccess**.
5. Press TAB again. The command is changed to **Get-CACrlDistributionPoint**.
6. You can continue pressing TAB to cycle through all the available commands that begin with **Get-C**. Similarly, you can cycle backwards by pressing SHIFT-TAB.

   NOTE: You can easily display a list of available Windows PowerShell commands. In addition, you can enter a portion of a command name and use tab-completion to resolve the partial command to a full Windows PowerShell command.

7. Press ENTER to execute the expanded Windows PowerShell command.

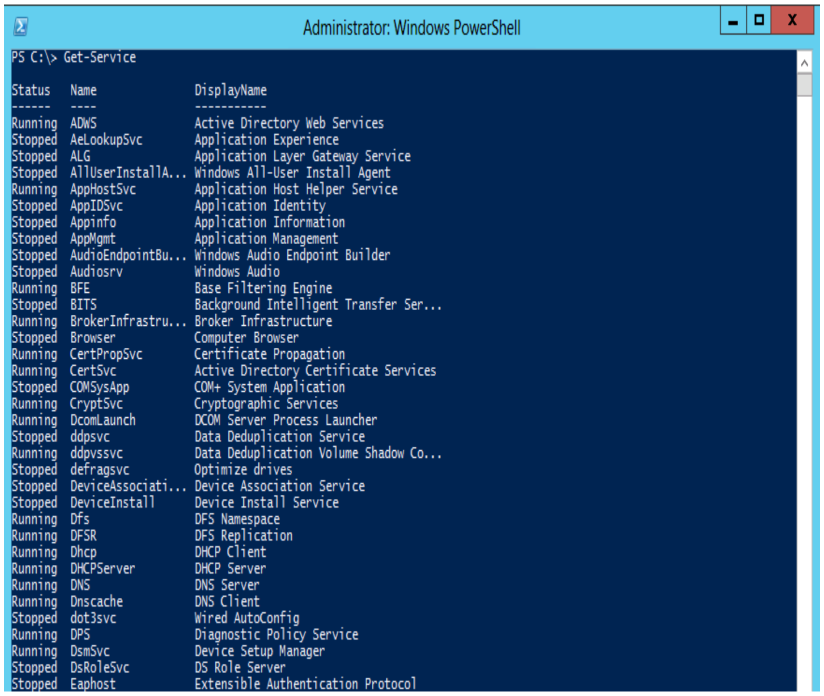
Format and filter output using Windows PowerShell

In this step, you will learn how to use command parameters and cmdlets to filter data. Also, you will learn how to format the displayed output.

1. To view a list of services installed on the computer, at the Windows PowerShell command prompt, type the following command, and then press ENTER.

```
↪ Get-Service
```

A list of services is displayed.



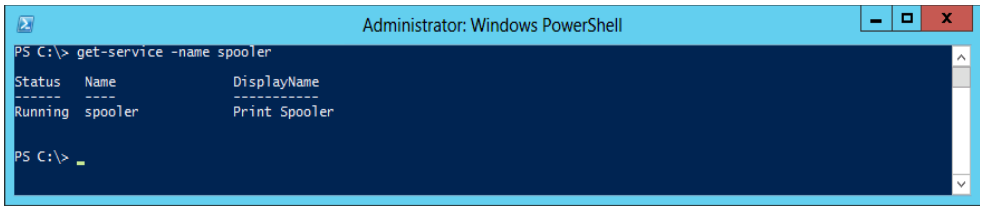
```
PS C:\> Get-Service

Status   Name               DisplayName
------   ----               -----------
Running  ADWS               Active Directory Web Services
Stopped  AeLookupSvc        Application Experience
Stopped  ALG                Application Layer Gateway Service
Stopped  AllUserInstallA... Windows All-User Install Agent
Running  AppHostSvc         Application Host Helper Service
Stopped  AppIDSvc           Application Identity
Stopped  Appinfo            Application Information
Stopped  AppMgmt            Application Management
Stopped  AudioEndpointBu... Windows Audio Endpoint Builder
Stopped  Audiosrv           Windows Audio
Running  BFE                Base Filtering Engine
Stopped  BITS               Background Intelligent Transfer Ser...
Running  BrokerInfrastru... Broker Infrastructure
Stopped  Browser            Computer Browser
Running  CertPropSvc        Certificate Propagation
Running  CertSvc            Active Directory Certificate Services
Stopped  COMSysApp          COM+ System Application
Running  CryptSvc           Cryptographic Services
Running  DcomLaunch         DCOM Server Process Launcher
Stopped  ddpsvc             Data Deduplication Service
Running  ddpvssvc           Data Deduplication Volume Shadow Co...
Stopped  defragsvc          Optimize drives
Stopped  DeviceAssociati... Device Association Service
Stopped  DeviceInstall      Device Install Service
Running  Dfs                DFS Namespace
Running  DFSR               DFS Replication
Running  Dhcp               DHCP Client
Running  DHCPServer         DHCP Server
Running  DNS                DNS Server
Running  Dnscache           DNS Client
Stopped  dot3svc            Wired AutoConfig
Running  DPS                Diagnostic Policy Service
Running  DsmSvc             Device Setup Manager
Stopped  DsRoleSvc          DS Role Server
Stopped  Eaphost            Extensible Authentication Protocol
```

Figure 6: The Get-Service command

- NOTE: Windows PowerShell provides a complete set of verbs to query and manipulate services, including **Get, New**, **Restart, Resume, Set, Start, Stop**, and **Suspend**. To view a list of service related commands type **Get-Command -Noun Service**.

2. Enter the following to view the status of the Spooler service:

```
↪ Get-Service –Name Spooler
```



```
PS C:\> get-service -name spooler

Status   Name               DisplayName
------   ----               -----------
Running  spooler            Print Spooler

PS C:\> _
```

Figure 7: Status of the Spooler service

- NOTE: The **Get-Service** command resolves the first input as the name parameter when you do not specify a parameter name for the input.

3. Enter the following command to obtain the same result as the previous step.

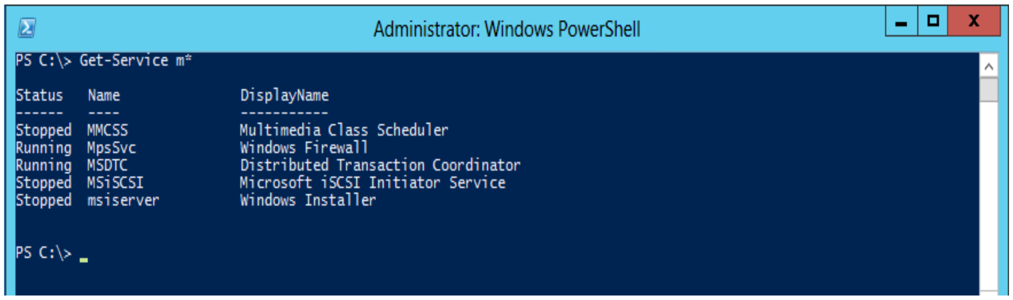
```
↪ Get-Service Spooler
```

- NOTE: You can use positioned or named parameters when you invoke cmdlets. Windows PowerShell provides built-in code that relieves cmdlet developers from having to parse program parameters.

4. Type the following command, and then press ENTER to view a list of all services that begin with M.

```
↪ Get-Service M*
```

- NOTE: The status of all the services beginning with M is shown.



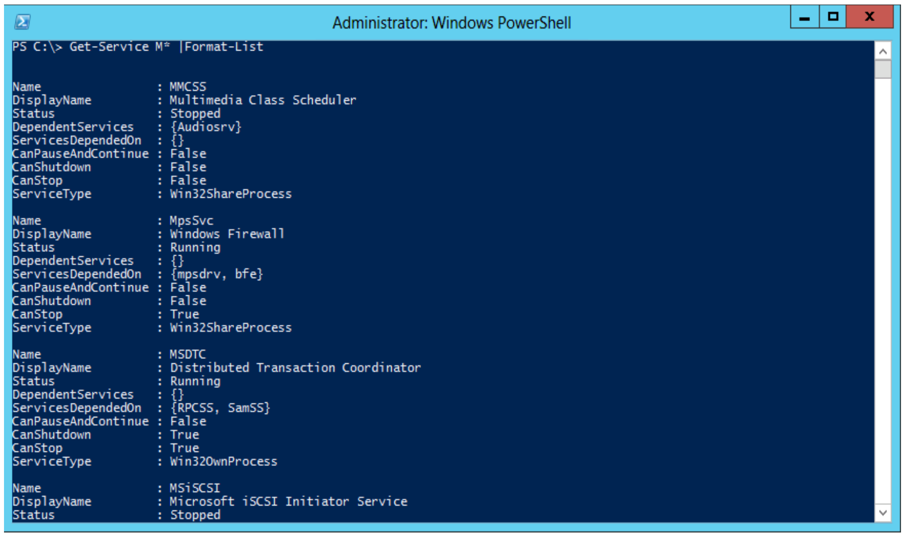
```
PS C:\> Get-Service m*

Status   Name               DisplayName
------   ----               -----------
Stopped  MMCSS              Multimedia Class Scheduler
Running  MpsSvc             Windows Firewall
Running  MSDTC              Distributed Transaction Coordinator
Stopped  MSiSCSI            Microsoft iSCSI Initiator Service
Stopped  msiserver          Windows Installer

PS C:\> _
```

Figure 8: Status of the services beginning with M

- NOTE: Many cmdlets allow the use of wildcards; you can use the wildcard to filter the results to a subset of all results.

5. Enter the following to see the same list of services. This time, the output is shown in list format.

```
↪ Get-Service M* | Format-List
```



```
PS C:\> Get-Service M* |Format-List


Name                : MMCSS
DisplayName         : Multimedia Class Scheduler
Status              : Stopped
DependentServices   : {Audiosrv}
ServicesDependedOn  : {}
CanPauseAndContinue : False
CanShutdown         : False
CanStop             : False
ServiceType         : Win32ShareProcess

Name                : MpsSvc
DisplayName         : Windows Firewall
Status              : Running
DependentServices   : {}
ServicesDependedOn  : {mpsdrv, bfe}
CanPauseAndContinue : False
CanShutdown         : False
CanStop             : True
ServiceType         : Win32ShareProcess

Name                : MSDTC
DisplayName         : Distributed Transaction Coordinator
Status              : Running
DependentServices   : {}
ServicesDependedOn  : {RPCSS, SamSS}
CanPauseAndContinue : False
CanShutdown         : True
CanStop             : True
ServiceType         : Win32OwnProcess

Name                : MSiSCSI
DisplayName         : Microsoft iSCSI Initiator Service
Status              : Stopped
```

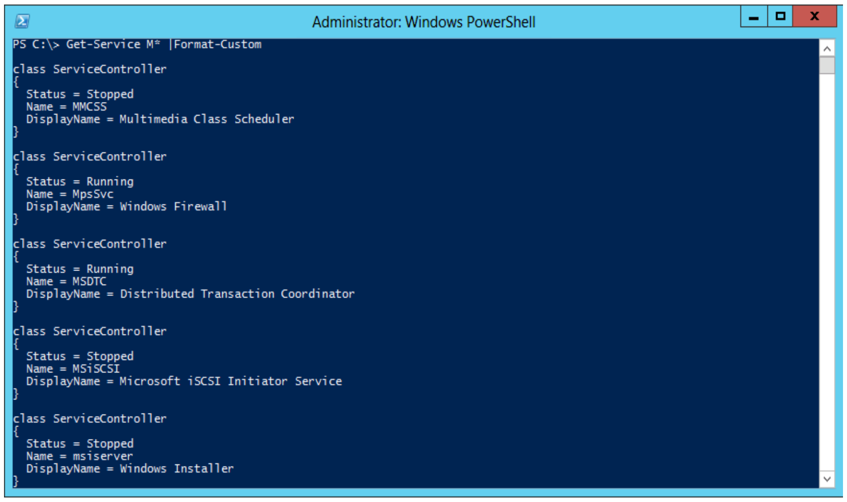
Figure 9: Using the format-list command

NOTE: In this example, there are two commands separated by a pipe (|) character. This means that the output of the first command is used as the second command's input.

6. Enter the following to see the same output, this time shown in a custom format.

```
↳ Get-Service M* | Format-Custom
```

NOTE: The status of all services beginning with M will be shown in the custom format.



```
PS C:\> Get-Service M* |Format-Custom

class ServiceController
{
  Status = Stopped
  Name = MMCSS
  DisplayName = Multimedia Class Scheduler
}

class ServiceController
{
  Status = Running
  Name = MpsSvc
  DisplayName = Windows Firewall
}

class ServiceController
{
  Status = Running
  Name = MSDTC
  DisplayName = Distributed Transaction Coordinator
}

class ServiceController
{
  Status = Stopped
  Name = MSiSCSI
  DisplayName = Microsoft iSCSI Initiator Service
}

class ServiceController
{
  Status = Stopped
  Name = msiserver
  DisplayName = Windows Installer
}
```

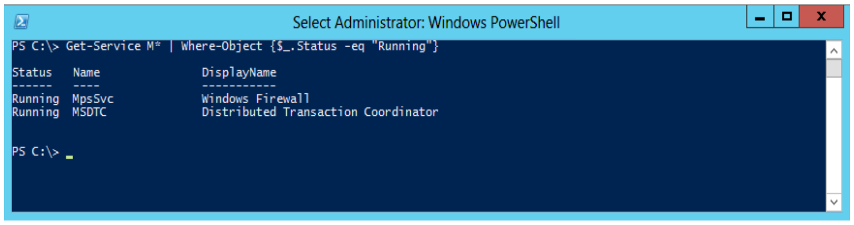
Figure 10: Using the Format-Custom command

NOTE: Windows PowerShell includes several defined format commands, each one with many configuration options. This provides a great deal of flexibility in output formatting. Developers can also create additional format commands.

7. Type the following command, and then press ENTER to view all the running services that begin with M.

```
↪ Get-Service M* | Where-Object {$_.Status -eq "Running"}
```

NOTE: Only services in a running state are shown this time.


Select Administrator: Windows PowerShell

```
PS C:\> Get-Service M* | Where-Object {$_.Status -eq "Running"}

Status   Name               DisplayName
------   ----               -----------
Running  MpsSvc             Windows Firewall
Running  MSDTC              Distributed Transaction Coordinator


PS C:\> _
```

Figure 11: Using the Where-Object command to filter the list to show only running services

NOTE: In this command:

– **Where-Object** is a command for performing a filter on the input.

– **{}{ }**are delimiters for Windows PowerShell code blocks.

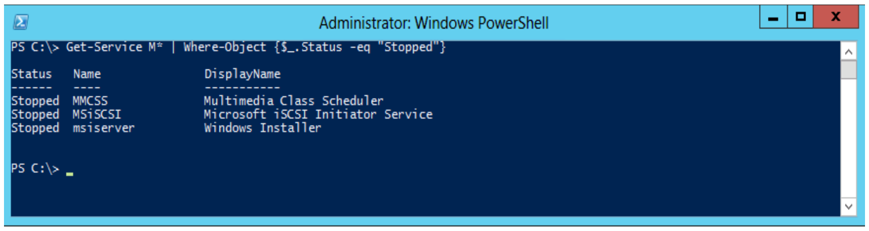
– **$_** refers to the input object (all services that begin with M).

– **-eq** specifies that the left-hand argument, in this case the Status property ($_.Status), will be compared for equality against the right-hand argument, the value “Running”.

8. To view all the stopped services that begin with M, type the following command, and then press ENTER.

```
↪ Get-Service M* | Where-Object {}{$_.Statuseq "Stopped"}
```

NOTE: Only those services that are stopped are shown.


Administrator: Windows PowerShell

```
PS C:\> Get-Service M* | Where-Object {$_.Status -eq "Stopped"}

Status   Name               DisplayName
------   ----               -----------
Stopped  MMCSS              Multimedia Class Scheduler
Stopped  MSiSCSI            Microsoft iSCSI Initiator Service
Stopped  msiserver          Windows Installer

PS C:\> _
```

Figure 12: Using the Where-Object command to filter the list to show only stopped services

NOTE: This is an example usage of the alias mechanism. Here, **Where** is an alias for the **Where-Object** command.

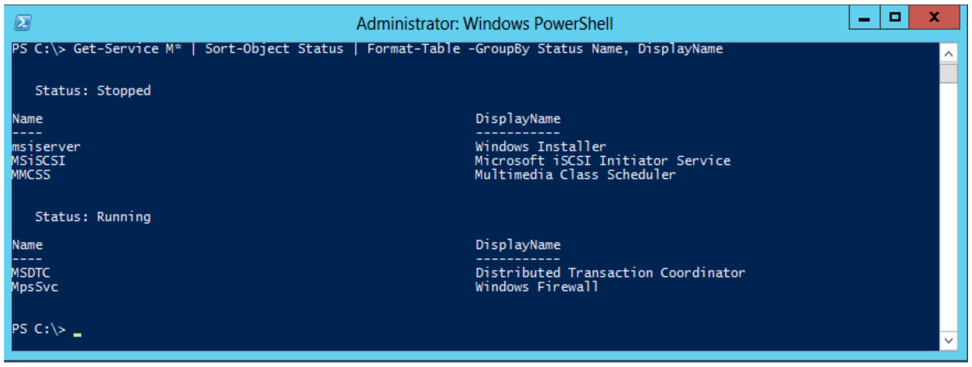
9. To view a list of all services ordered by their status, type the following command, and then press ENTER.

```
↪ Get-Service | Sort-Object Status
```

NOTE: The **Sort-Object** command can order objects returned by a previous cmdlet using one or more of their properties. You can also use the alias **Sort** to refer to the **Sort-Object** command.

10. Type the following command, and then press ENTER to view the Name and DisplayName of all services beginning with M, with the output ordered and grouped by Status.

```
Get-Service M* | Sort-Object Status | Format-Table -GroupBy Status
Name, DisplayName
```



Administrator: Windows PowerShell

```
PS C:\> Get-Service M* | Sort-Object Status | Format-Table -GroupBy Status Name, DisplayName

   Status: Stopped

Name                                                        DisplayName
----                                                        -----------
msiserver                                                   Windows Installer
MSiSCSI                                                     Microsoft iSCSI Initiator Service
MMCSS                                                       Multimedia Class Scheduler

   Status: Running

Name                                                        DisplayName
----                                                        -----------
MSDTC                                                       Distributed Transaction Coordinator
MpsSvc                                                      Windows Firewall

PS C:\> _
```

N

Figure 13: Using the Sort-Object and Format-Table command to group the output data

View object metadata

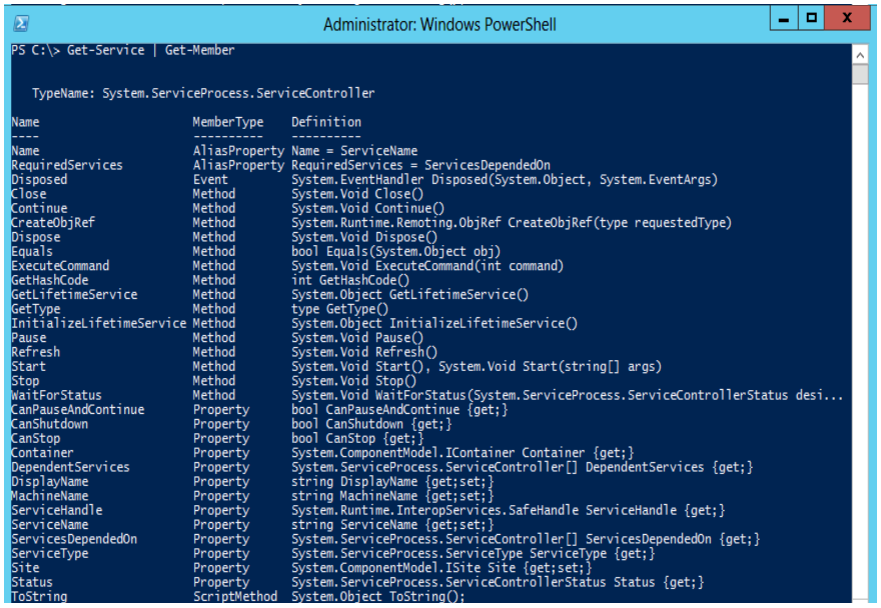
Everything in Windows PowerShell is a .NET object (and available for reflection), including cmdlets, services, and processes. Windows PowerShell extends the .NET type reflector to allow simpler access for administrative purposes.

In this step, you will learn how to view the type members of those objects.

1. To view the type metadata for the object output by the Get-Service command, type the following command, and then press ENTER.

```
Get-Service | Get-Member
```

NOTE: The type metadata for the object output by Get-Service is displayed. Note that the data type, in this case the System.ServiceProcess.ServiceController class, is also displayed.

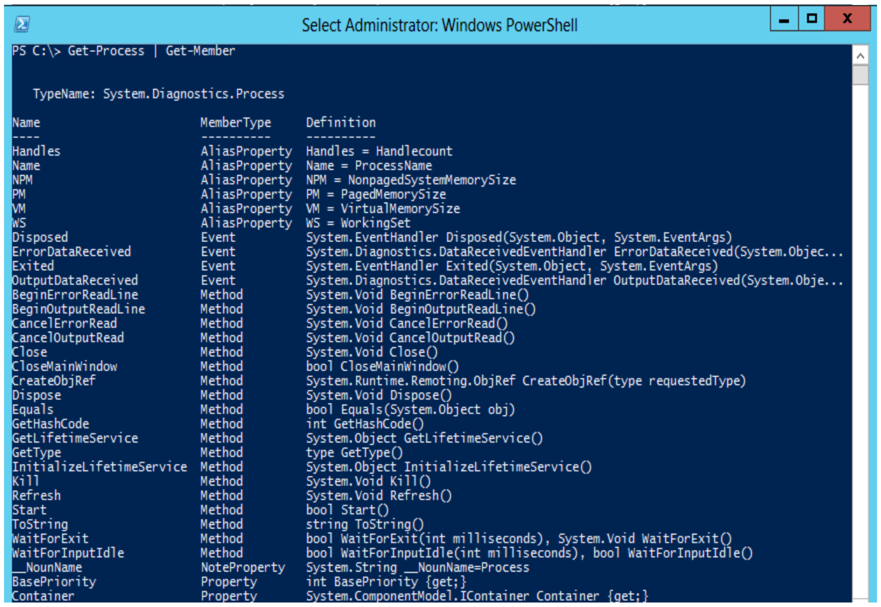

Administrator: Windows PowerShell

```
PS C:\> Get-Service | Get-Member


   TypeName: System.ServiceProcess.ServiceController

Name                      MemberType    Definition
----                      ----------    ----------
Name                      AliasProperty Name = ServiceName
RequiredServices          AliasProperty RequiredServices = ServicesDependedOn
Disposed                  Event         System.EventHandler Disposed(System.Object, System.EventArgs)
Close                     Method        System.Void Close()
Continue                  Method        System.Void Continue()
CreateObjRef              Method        System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose                   Method        System.Void Dispose()
Equals                    Method        bool Equals(System.Object obj)
ExecuteCommand            Method        System.Void ExecuteCommand(int command)
GetHashCode               Method        int GetHashCode()
GetLifetimeService        Method        System.Object GetLifetimeService()
GetType                   Method        type GetType()
InitializeLifetimeService Method        System.Object InitializeLifetimeService()
Pause                     Method        System.Void Pause()
Refresh                   Method        System.Void Refresh()
Start                     Method        System.Void Start(), System.Void Start(string[] args)
Stop                      Method        System.Void Stop()
WaitForStatus             Method        System.Void WaitForStatus(System.ServiceProcess.ServiceControllerStatus desi...
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;}
Status                    Property      System.ServiceProcess.ServiceControllerStatus Status {get;}
ToString                  ScriptMethod  System.Object ToString();
```

Figure 14: Using the Get-Member command to view object metadata

2. To view the type metadata for the object output by the Get-Process command, type the following command, and then press ENTER.

```
↪ Get-Process | Get-Member
```


Select Administrator: Windows PowerShell

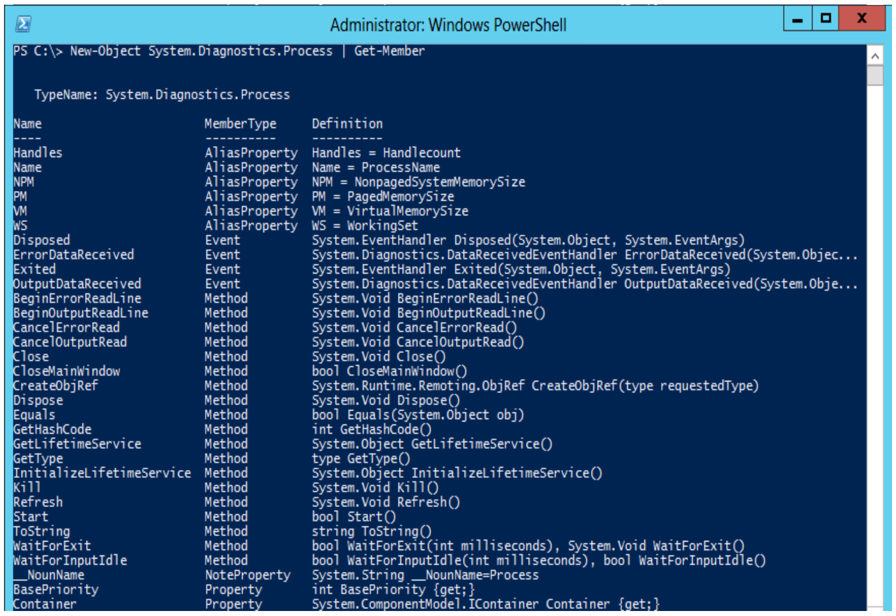
```
PS C:\> Get-Process | Get-Member


   TypeName: System.Diagnostics.Process

Name                       MemberType     Definition
----                       ----------     ----------
Handles                    AliasProperty  Handles = Handlecount
Name                       AliasProperty  Name = ProcessName
NPM                        AliasProperty  NPM = NonpagedSystemMemorySize
PM                         AliasProperty  PM = PagedMemorySize
VM                         AliasProperty  VM = VirtualMemorySize
WS                         AliasProperty  WS = WorkingSet
Disposed                   Event          System.EventHandler Disposed(System.Object, System.EventArgs)
ErrorDataReceived          Event          System.Diagnostics.DataReceivedEventHandler ErrorDataReceived(System.Objec...
Exited                     Event          System.EventHandler Exited(System.Object, System.EventArgs)
OutputDataReceived         Event          System.Diagnostics.DataReceivedEventHandler OutputDataReceived(System.Obje...
BeginErrorReadLine         Method         System.Void BeginErrorReadLine()
BeginOutputReadLine        Method         System.Void BeginOutputReadLine()
CancelErrorRead            Method         System.Void CancelErrorRead()
CancelOutputRead           Method         System.Void CancelOutputRead()
Close                      Method         System.Void Close()
CloseMainWindow            Method         bool CloseMainWindow()
CreateObjRef               Method         System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose                    Method         System.Void Dispose()
Equals                     Method         bool Equals(System.Object obj)
GetHashCode                Method         int GetHashCode()
GetLifetimeService         Method         System.Object GetLifetimeService()
GetType                    Method         type GetType()
InitializeLifetimeService  Method         System.Object InitializeLifetimeService()
Kill                       Method         System.Void Kill()
Refresh                    Method         System.Void Refresh()
Start                      Method         bool Start()
ToString                   Method         string ToString()
WaitForExit                Method         bool WaitForExit(int milliseconds), System.Void WaitForExit()
WaitForInputIdle           Method         bool WaitForInputIdle(int milliseconds), bool WaitForInputIdle()
__NounName                 NoteProperty   System.String __NounName=Process
BasePriority               Property       int BasePriority {get;}
Container                  Property       System.ComponentModel.IContainer Container {get;}
```

Figure 15: Members of the System.Diagnostics.Process type

3. To create a new object of type System.Diagnostics.Process and view the type metadata for the class, type the following command, and then press ENTER.

```
↪ New-Object System.Diagnostics.Process | Get-Member
```

NOTE: The type metadata for the System.Diagnostics.Process class is shown.



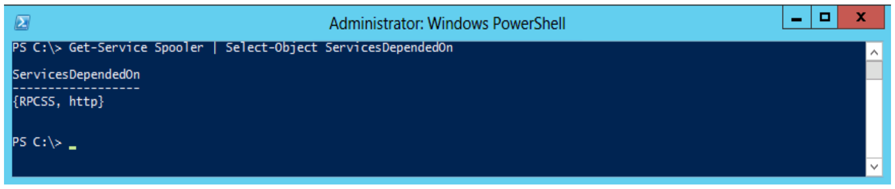
```
PS C:\> New-Object System.Diagnostics.Process | Get-Member


   TypeName: System.Diagnostics.Process

Name                      MemberType     Definition
----                      ----------     ----------
Handles                   AliasProperty  Handles = Handlecount
Name                      AliasProperty  Name = ProcessName
NPM                       AliasProperty  NPM = NonpagedSystemMemorySize
PM                        AliasProperty  PM = PagedMemorySize
VM                        AliasProperty  VM = VirtualMemorySize
WS                        AliasProperty  WS = WorkingSet
Disposed                  Event          System.EventHandler Disposed(System.Object, System.EventArgs)
ErrorDataReceived         Event          System.Diagnostics.DataReceivedEventHandler ErrorDataReceived(System.Objec...
Exited                    Event          System.EventHandler Exited(System.Object, System.EventArgs)
OutputDataReceived        Event          System.Diagnostics.DataReceivedEventHandler OutputDataReceived(System.Obje...
BeginErrorReadLine        Method         System.Void BeginErrorReadLine()
BeginOutputReadLine       Method         System.Void BeginOutputReadLine()
CancelErrorRead           Method         System.Void CancelErrorRead()
CancelOutputRead          Method         System.Void CancelOutputRead()
Close                     Method         System.Void Close()
CloseMainWindow           Method         bool CloseMainWindow()
CreateObjRef              Method         System.Runtime.Remoting.ObjRef CreateObjRef(type requestedType)
Dispose                   Method         System.Void Dispose()
Equals                    Method         bool Equals(System.Object obj)
GetHashCode               Method         int GetHashCode()
GetLifetimeService        Method         System.Object GetLifetimeService()
GetType                   Method         type GetType()
InitializeLifetimeService Method         System.Object InitializeLifetimeService()
Kill                      Method         System.Void Kill()
Refresh                   Method         System.Void Refresh()
Start                     Method         bool Start()
ToString                  Method         string ToString()
WaitForExit               Method         bool WaitForExit(int milliseconds), System.Void WaitForExit()
WaitForInputIdle          Method         bool WaitForInputIdle(int milliseconds), bool WaitForInputIdle()
__NounName                NoteProperty   System.String __NounName=Process
BasePriority              Property       int BasePriority {get;}
Container                 Property       System.ComponentModel.IContainer Container {get;}
```

Figure 16: Type members of a new object

4. Type the following command, and then press ENTER to view a list of services that the Spooler service depends on.

```
↪ Get-Service Spooler | Select-Object ServicesDependedOn
```

NOTE: A list of services that the Spooler service requires to start is displayed.



```
PS C:\> Get-Service Spooler | Select-Object ServicesDependedOn

ServicesDependedOn
------------------
{RPCSS, http}

PS C:\> _
```

Figure 17: Inspecting the ServicesDependedOn property of the Spooler service

Using whatif and confirm

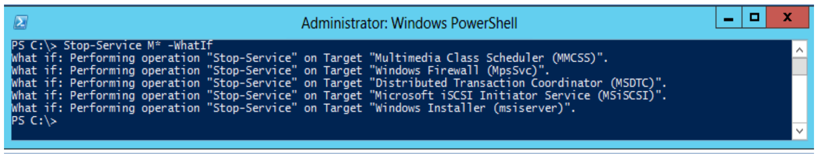
Windows PowerShell allows administrators to safely test and use commands. In this step, you will use the whatif and confirm commands.

- **[Command] -WhatIf** flag shows you the results without actually performing the action.
- **[Command] -Confirm** flag asks you to confirm the operation before it executes.

1. Type the following command, and then press ENTER to see a list of services that would be stopped if you ran the Stop-Service command.

```
↪ Stop-Service M* -WhatIf
```

NOTE: A list of services that would be stopped is displayed.



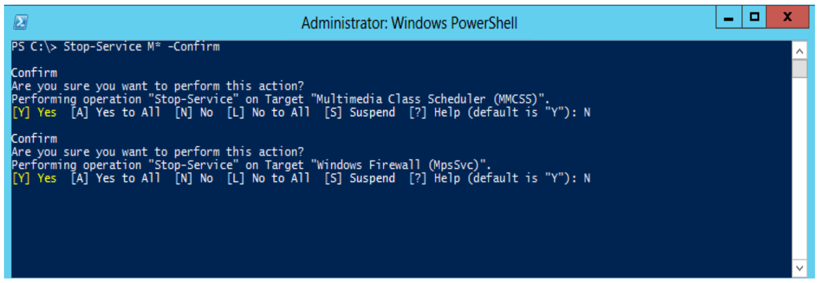
```
PS C:\> Stop-Service M* -WhatIf
What if: Performing operation "Stop-Service" on Target "Multimedia Class Scheduler (MMCSS)".
What if: Performing operation "Stop-Service" on Target "Windows Firewall (MpsSvc)".
What if: Performing operation "Stop-Service" on Target "Distributed Transaction Coordinator (MSDTC)".
What if: Performing operation "Stop-Service" on Target "Microsoft iSCSI Initiator Service (MSiSCSI)".
What if: Performing operation "Stop-Service" on Target "Windows Installer (msiserver)".
PS C:\>
```

Figure 18: The Stop Service command with the WhatIf flag specified

2. Type the following command, and then press ENTER to confirm whether or not to stop each service.

```
↪ Stop-Service M* -Confirm
```

NOTE: For each service beginning with M, you will be asked if it should be stopped. For this lab, reply No in each case.



```
PS C:\> Stop-Service M* -Confirm

Confirm
Are you sure you want to perform this action?
Performing operation "Stop-Service" on Target "Multimedia Class Scheduler (MMCSS)".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"): N

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

Figure 19: The Stop-Service command with the confirm flag specified

Creating and manipulating variables

In this step, you will be working with variables in Windows PowerShell. You will learn about their declaration, usage, and behavior.

1. To create a variable to hold a string value, type the following command, and then press ENTER.

```
↪ $var = "Hi there"
```

NOTE: You have just created a variable named **var**, and you assigned the string value **Hi there** to it.

NOTE: A variable in Windows PowerShell must begin with the dollar sign (**$**). If special characters are needed in a variable name, curly braces can be used to surround the variable name (**{}**).

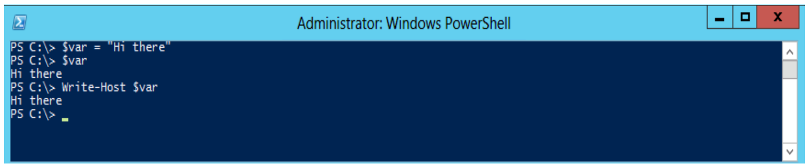
2. To output the value stored in this variable, type the following command, and then press ENTER.

```
↪ $var
```

- NOTE: A better way to output variable values is to use a cmdlet named Write-Host before the variable name, clearly showing it will output the values to the host.

```
↪ Write-host $var
```

The results should be similar to the following screenshot.



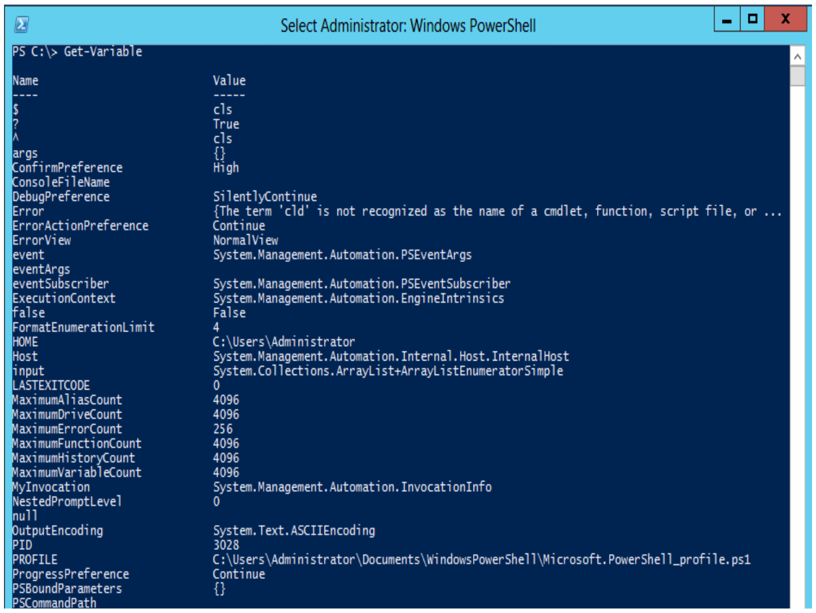
```
PS C:\> $var = "Hi there"
PS C:\> $var
Hi there
PS C:\> Write-Host $var
Hi there
PS C:\> _
```

Figure 20: Printing a variable value

- NOTE: By default, a variable will have a null value. Null values in Windows PowerShell are represented as **$null**.

3. Variables in Windows PowerShell can be listed and accessed under a special location. To display the list of currently declared variables, type the following command, and then press ENTER.

```
↪ Get-Variable
```



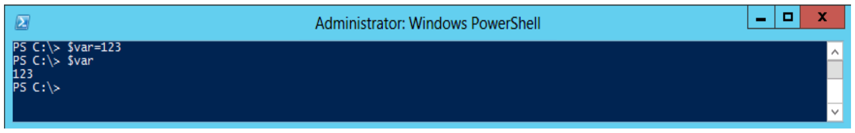
```
PS C:\> Get-Variable

Name                           Value
----                           -----
$                              cls
?                              True
^                              cls
args                           {}
ConfirmPreference              High
ConsoleFileName
DebugPreference                SilentlyContinue
Error                          {The term 'cld' is not recognized as the name of a cmdlet, function, script file, or ...
ErrorActionPreference          Continue
ErrorView                      NormalView
event                          System.Management.Automation.PSEventArgs
eventArgs
eventSubscriber                System.Management.Automation.PSEventSubscriber
ExecutionContext               System.Management.Automation.EngineIntrinsics
false                          False
FormatEnumerationLimit         4
HOME                           C:\Users\Administrator
Host                           System.Management.Automation.Internal.Host.InternalHost
input                          System.Collections.ArrayList+ArrayListEnumeratorSimple
LASTEXITCODE                   0
MaximumAliasCount              4096
MaximumDriveCount              4096
MaximumErrorCount              256
MaximumFunctionCount           4096
MaximumHistoryCount            4096
MaximumVariableCount           4096
MyInvocation                   System.Management.Automation.InvocationInfo
NestedPromptLevel              0
null
OutputEncoding                 System.Text.ASCIIEncoding
PID                            3028
PROFILE                        C:\Users\Administrator\Documents\WindowsPowerShell\Microsoft.PowerShell_profile.ps1
ProgressPreference             Continue
PSBoundParameters              {}
PSCommandPath
```

Figure 21: List of currently declared PowerShell variables

4. By default, variables in Windows PowerShell are non-typed, which means they can hold an arbitrary value. To change the variable value and type to an integer, type the following command, pressing ENTER after each line.

```
↪ $var = 123
↪ $var
```

Now the variable is holding the integer value **123**.



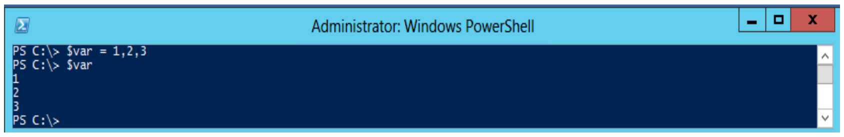
```
PS C:\> $var=123
PS C:\> $var
123
PS C:\>
```

Figure 22: Assigning an integer value

5. Variables can also contain lists (similar to arrays). To change the value to an array of integers, type the following commands, pressing ENTER after each line.

```
↪ $var = 1,2,3
↪ $var
```

NOTE: This time, the variable is holding an array of integers.



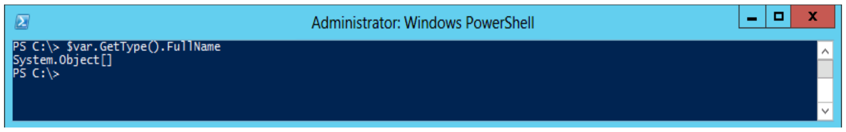
```
PS C:\> $var = 1,2,3
PS C:\> $var
1
2
3
PS C:\>
```

Figure 23: Creating an array of integers

6. To see the new type of the variable, type the following command, and then press ENTER.

```
↪ $var.GetType().FullName
```

NOTE: The variable is now an array object of type System.Object[].



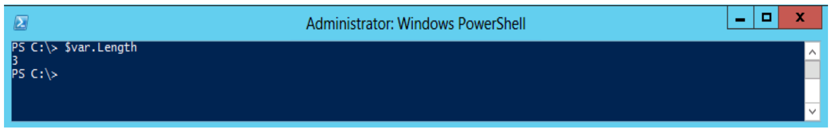
```
PS C:\> $var.GetType().FullName
System.Object[]
PS C:\>
```

Figure 24: Obtaining the type of the variable

7. Arrays can be also manipulated through their .NET methods. For example, they can be queried on their length. To use the Length property to retrieve the number of elements of the array, type the following command, and then press ENTER.

```
↪ $var.Length
```

NOTE: The size of the array is displayed.



```
PS C:\> $var.Length
3
PS C:\>
```

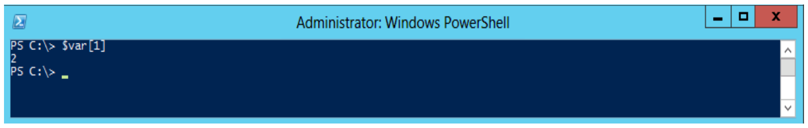
Figure 25: Displaying the array length

8. You can also access individual elements within an array by using square brackets ([]). To retrieve the second element of the array, type the following command, and then press ENTER.

```
↪ $var[1]
```

📌 NOTE: Arrays in Windows PowerShell are zero-based, which means that the first element will always be at position (index value) 0.

The value of the second element of the array is displayed.



Administrator: Windows PowerShell

```
PS C:\> $var[1]
2
PS C:\> _
```

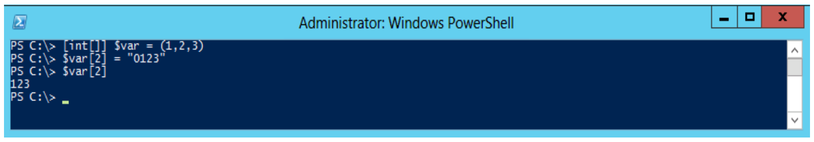
Figure 26: Retrieving an element of the array

9. You can also type a variable by prefixing its declaration with the desired data type name. To re-declare the variable as an array of integers, type the following command, and then press ENTER.

```
↪  [int[]] $var = (1,2,3)
```

10. To assign the string value 0123 to the third element of the array, and then display it, type the following commands, pressing ENTER after each one.

```
↪ $var[2] = "0123"
↪ $var[2]
```



Administrator: Windows PowerShell

```
PS C:\> [int[]] $var = (1,2,3)
PS C:\> $var[2] = "0123"
PS C:\> $var[2]
123
PS C:\> _
```

Figure 27: Implicit conversion from string to integer

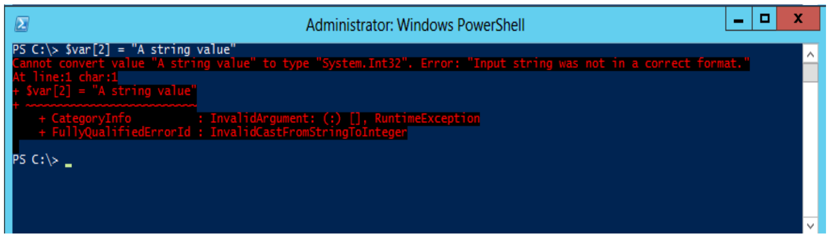
📌 NOTE: Examining the output of the variable, the string 0123 was converted into the number 123. Windows PowerShell implicitly converted the stored value to match the destination variable type.

📌 NOTE: Implicit type conversions are done only for typed variables. Unlike strongly typed languages where a variable can only be assigned an object of the correct type, Windows PowerShell allows the assignment of any object, as long as it is convertible to the target type, by the extensive use of implicit conversions.

11. When an implicit conversion is not available, it displays an error. To attempt to set a string value that cannot be converted, type the following command, and then press ENTER.

```
↪ $var[2] = "A string value"
```

📌 NOTE: The following error is displayed.



```
PS C:\> $var[2] = "A string value"
Cannot convert value "A string value" to type "System.Int32". Error: "Input string was not in a correct format."
At line:1 char:1
+ $var[2] = "A string value"
+ ~~~~~~~~~~~~~~~~~~~~~~~~~~
    + CategoryInfo          : InvalidArgument: (:) [], RuntimeException
    + FullyQualifiedErrorId : InvalidCastFromStringToInteger

PS C:\> _
```

Figure 28: Type conversion error

Working with strings

In this step, you will use different operators to deal with strings values in Windows PowerShell.

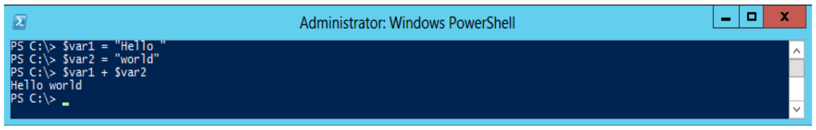
1. To create and initialize two string variables named var1 and var2, type the following commands, pressing ENTER after each one.

```
↪ $var1 = "Hello "
↪ $var2 = "world"
```

2. To use the plus (**+**) operator to concatenate the two string variables, type the following command, and then press ENTER.

```
↪ $var1 + $var2
```

NOTE: The result of this operation is a new Hello World string, as shown in the following screen shot.



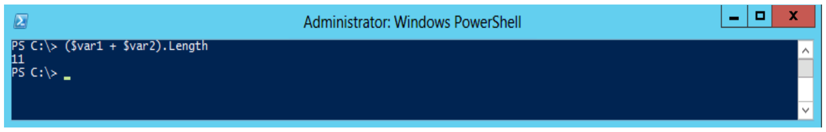
```
PS C:\> $var1 = "Hello "
PS C:\> $var2 = "world"
PS C:\> $var1 + $var2
Hello world
PS C:\> _
```

Figure 29: Concatenating two string variables

NOTE: Windows PowerShell defines the behavior of the **+** operator for numbers, strings, arrays and hash tables. Adding two numbers produces a numeric result following the numeric widening rules. Adding two strings performs a string concatenation, resulting in a new string, and adding two arrays joins the two arrays (array concatenation).

3. You can use the .NET **String** properties to inspect the string objects. To use the Length property to obtain the size in characters of the previous string concatenation, type the following command, and then press ENTER.

```
↪  ($var1 + $var2).Length
```



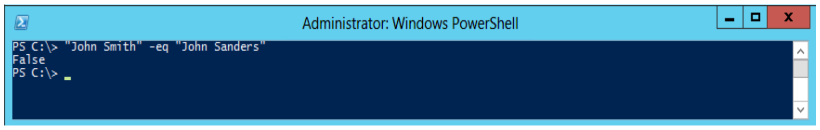
```
PS C:\> ($var1 + $var2).Length
11
PS C:\> _
```

Figure 30: Using the Length property

4. Windows PowerShell also provides other kinds of binary operators, like comparison operators. To verify if two strings are equal, type the following command, and then press ENTER,

```
↪ "John Smith" -eq "John Sanders"
```

NOTE: The result of the comparison is shown in the following screen shot.


Administrator: Windows PowerShell

```
PS C:\> "John Smith" -eq "John Sanders"
False
PS C:\> _
```

Figure 31: Comparing two strings

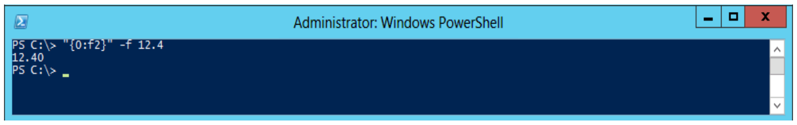
NOTE: There are other comparison operators, like **–ne** (not equals) **–gt** (greater than), **–lt** (less than), **-ge** (greater than or equal) and **–le** (less than or equals).

NOTE: For each of these operators there is also a base or unqualified operator form, like **–eq** and its two variants **–ceq** and **–ieq.** The “c” variant is case-sensitive and the “I” variant is case-insensitive.

5. Formatting is a common task that can also be done in Windows PowerShell. To use a custom format to display 12.4 as 12.40, type the following command, and then press ENTER.

```
↪ "{0:f2}" -f 12.4
```

NOTE: The output should look like the following screenshot.


Administrator: Windows PowerShell

```
PS C:\> "{0:f2}" -f 12.4
12.40
PS C:\> _
```

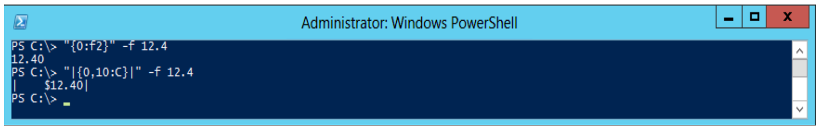
Figure 32: Formatting decimal values

6. To display the same number as currency, and then to pad it to 10 characters aligned to the right, type the following command, and then press ENTER.

NOTE: Use the vertical bars to see the added padding:

```
↪ "|{0,10:C}|" -f 12.4
```

The currency symbol configured in the current culture of the local machine is used.

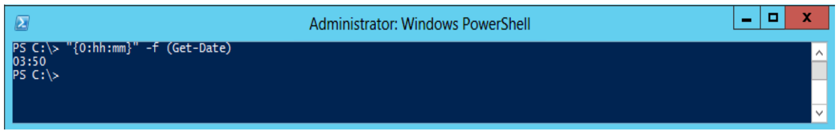

Administrator: Windows PowerShell

```
PS C:\> "{0:f2}" -f 12.4
12.40
PS C:\> "|{0,10:C}|" -f 12.4
|    $12.40|
PS C:\> _
```

Figure 33: Formatting currency values

7. Date and time formatting can also be done. To display only hours and minutes from the current date, type the following command, and then press ENTER.

```
↪ "{0:hh:mm}" -f (Get-Date)
```



```
PS C:\> "{0:hh:mm}" -f (Get-Date)
03:50
PS C:\>
```

Figure 34: Formatting date and time values

Creating a script file

Script files are used to store Windows PowerShell commands in a file, providing an easy way to run a list of commands. You only need to tell Windows PowerShell to run the script file.

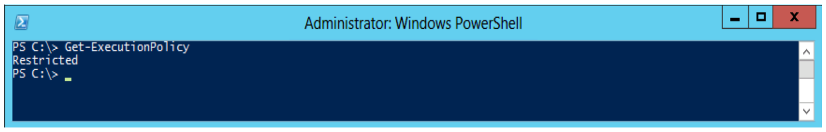
In this step, you will learn how to create and run script files. To understand the reasons behind the security features of Windows PowerShell, you will be introduced to a Windows PowerShell security feature called execution policies. The execution policy enables you to determine which Windows PowerShell scripts (if any) will be allowed to run on your computer. Windows PowerShell has four different execution policies:

- **Restricted** – No scripts are allowed to run. Windows PowerShell can only be used in interactive mode.
- **AllSigned** – Only scripts signed by a trusted publisher can be run.
- **RemoteSigned** – Downloaded scripts must be signed by a trusted publisher before they can be run.
- **Unrestricted** – No restrictions; all Windows PowerShell scripts can be run.

NOTE: When you first install Windows PowerShell, it will have the default value for the execution policy as Restricted.

1. To display the current execution policy, type the following command, and then press ENTER.

```
↪ Get-ExecutionPolicy
```



```
PS C:\> Get-ExecutionPolicy
Restricted
PS C:\> _
```

Figure 35: The current execution policy is displayed

2. Before running a script file, you will have to change the execution policy. To change the execution policy to RemoteSigned and verify the change, type the following commands, pressing ENTER after each one.

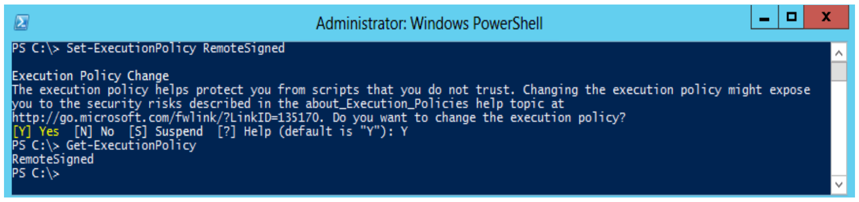
```
↪ Set-ExecutionPolicy RemoteSigned
↪ Get-ExecutionPolicy
```

- NOTE: Changing the execution policy requires administrative privileges. If you are running Windows Vista or Windows 7, on the Start menu, right-click the Windows PowerShell icon, and then click Run as Administrator.
- NOTE: For other platforms, either log in with an administrative account or open a Windows PowerShell console by using the runas command and supplying appropriate credentials. For example:

  runas /user:username PowerShell

  where *username* is an account with administrative privileges.

  Remember that it is best practice to use administrative privileges only for operations that require it.



```
PS C:\> Set-ExecutionPolicy RemoteSigned

Execution Policy Change
The execution policy helps protect you from scripts that you do not trust. Changing the execution policy might expose
you to the security risks described in the about_Execution_Policies help topic at
http://go.microsoft.com/fwlink/?LinkID=135170. Do you want to change the execution policy?
[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): Y
PS C:\> Get-ExecutionPolicy
RemoteSigned
PS C:\>
```

Figure 36: Changing the execution policy

- NOTE: More information on Windows PowerShell security can be found at: http://technet.microsoft.com/en-us/magazine/2007.09.powershell.aspx.

3. To create the script file, open Server Manager, click **Tools** and then click **Windows PowerShell ISE**.

   - NOTE: To create script files, you don't need a special editor. In this example you are going to use the Windows PowerShell Integrated Scripting Environment (ISE). The Windows PowerShell ISE is a host application that enables you to run commands, write, test, and debug scripts in a friendly, syntax-colored, Unicode-compliant environment.

4. On the View menu, click **Show Script Pane**.

   - NOTE: The ISE has two windows, or panes, which solves the dilemma of editing text in a separate application.

     1. The **Script Pane** at the top allows you to compose, edit, debug, and run functions, and scripts. (Note: The Script Pane is not displayed by default – on the View menu, click Script Pane to display it)
     2. The **Console Pane** at the bottom is used for running interactive commands, just as you would in the Windows PowerShell text-based console.

5. In the script pane, type the following commands, pressing ENTER after each line.

```
↪ # test.ps1
↪ # Show Hello and time.
↪
↪ "" # Blank Line
↪ "Hello " + $env:UserName + "!"
↪ "Time is " + "{0:HH}:{0:mm}" -f (Get-Date)
↪ "" # Blank Line
```

6. To save the file as a Windows PowerShell script file, on the File menu, click **Save As**.
7. In the file name, type **test.ps1**, in the location, type **C:\users\Administrator\Desktop**, and then click **Save**.

   📌 NOTE: When you create your script file, the filename must have a .ps1 extension.

8. Click in the command pane, type the following command, and then press ENTER.

```
↪ cd C:\users\Administrator\Desktop
```

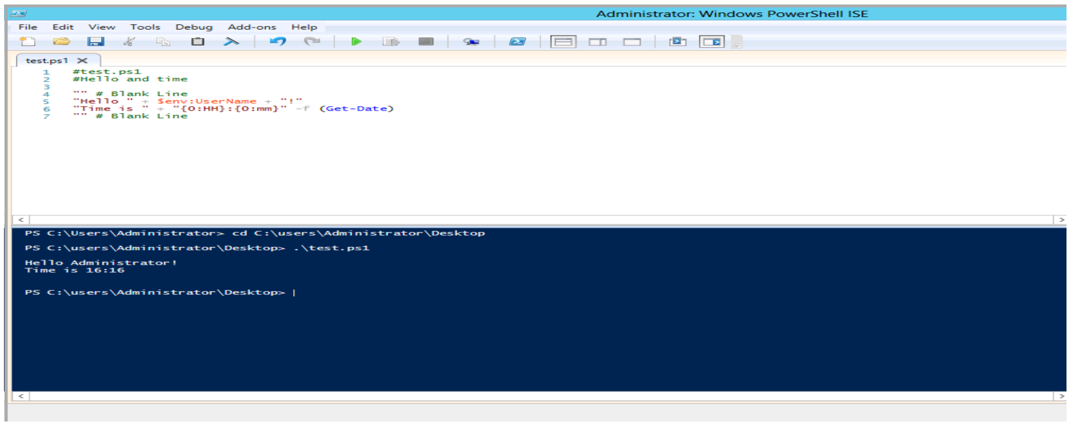
9. To execute the script file you created in the previous step, type the following command, and then press ENTER.

```
↪ .\test.ps1
```

   📌 NOTE: You can follow the same procedure to execute scripts in the Windows PowerShell text-based console. From the ISE, you can also execute a script using the Run command on the File menu.

   📌 NOTE: Preceding the script name with directory information, in this case the current directory (**.**), instructs Windows PowerShell to run a script.

   📌 NOTE: There must be no space between **.** and the script name. Adding the .ps1 extension is optional. You must specify the path to the script file, even if the script is in the current directory.


Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
test.ps1

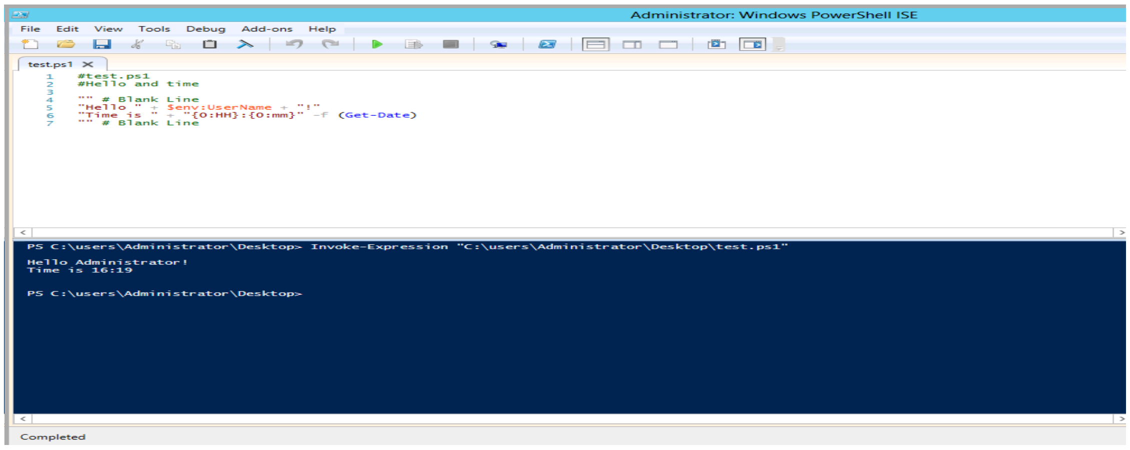
```
#test.ps1
#Hello and time

"" # Blank Line
"Hello " + $env:UserName + "!"
"Time is " + "{0:HH}:{0:mm}" -f (Get-Date)
"" # Blank Line
```

```
PS C:\Users\Administrator> cd C:\users\Administrator\Desktop

PS C:\users\Administrator\Desktop> .\test.ps1

Hello Administrator!
Time is 16:16

PS C:\users\Administrator\Desktop> |
```

Figure 37: Running the script file from the current directory

10. To use the Invoke-Expression command as an alternative way of running the same script, type the following command, and then press ENTER.

```
↪ Invoke-Expression "C:\Users\Administrator\Desktop\test.ps1"
```


Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
test.ps1
Completed

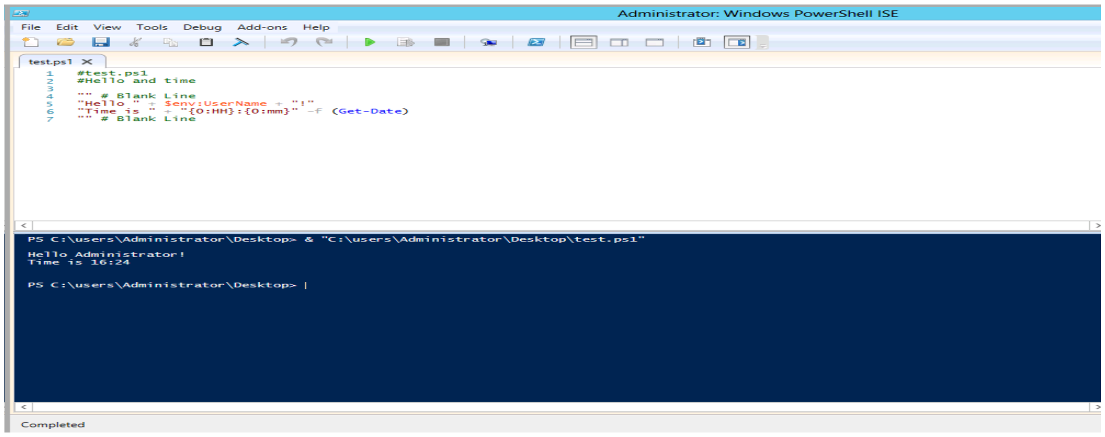
```
#test.ps1
#Hello and time

"" # Blank Line
"Hello " + $env:UserName + "!"
"Time is " + "{0:HH}:{0:mm}" -f (Get-Date)
"" # Blank Line
```

```
PS C:\users\Administrator\Desktop> Invoke-Expression "C:\users\Administrator\Desktop\test.ps1"

Hello Administrator!
Time is 16:19

PS C:\users\Administrator\Desktop>
```

Figure 38: Running the script file using Invoke-Expression

11. To use the invoke operator (&) to execute the command in the string that follows, type the following command, and then press ENTER.

```
↪ & "C:\Users\Administrator\Desktop\test.ps1"
```


Administrator: Windows PowerShell ISE
File Edit View Tools Debug Add-ons Help
test.ps1
Completed

```
#test.ps1
#Hello and time

"" # Blank Line
"Hello " + $env:UserName + "!"
"Time is " + "{0:HH}:{0:mm}" -f (Get-Date)
"" # Blank Line
```

```
PS C:\users\Administrator\Desktop> & "C:\users\Administrator\Desktop\test.ps1"

Hello Administrator!
Time is 16:24

PS C:\users\Administrator\Desktop> |
```

Figure 39: Running the script file using ampersand (&)

12. To close the test.ps1 script, on the test.ps1 tab, click the **X**.

   📌 NOTE: As an alternative, on the File menu, click Close.

   📌 NOTE: Many commands in the ISE have keyboard shortcuts. You can see the keyboard shortcuts to the right of their respective commands in the menus at the top of the ISE. The keyboard shortcut to close the active file in the script pane is Ctrl+F4.

Creating functions

In Windows PowerShell, you can declare functions. Functions are reusable pieces of code that can be called as many times as you want after you declare them.

In this step, you will declare a function, learn about using different types of parameters, and specify default values for these parameters.

1. In the Windows PowerShell ISE, press CTRL+N.
2. In the script pane, type the following code.

```
↪ function Get-Soup (
↪     [switch] $please,
↪     [string] $soup = "chicken noodle"
↪ )
↪ {}{
↪     if ($please) {
↪         "Here's your $soup soup"
↪     }
↪     else {
```

```
↪          "No soup for you!"
↪      }
↪  }
```

📌 This command declares a **Get-Soup** function which will receive two parameters, **$please** and **$soup**

3. To run the script, press F5, and then click the **Script** arrow to hide the script pane.

   📌 NOTE: It is not necessary to save a script before executing it in the ISE. This makes it easy to use the script pane to quickly test commands and code snippets.

4. To call the script with no parameters, in the command pane, type the following command, and then press ENTER.

   ```
   ↪ Get-Soup
   ```

5. To call the Get Soup function with the $please parameter, type the following command, and then press ENTER.

   📌 NOTE: To specify a parameter declared as a switch, specify ***–parameter name*** after the function name.

   ```
   ↪ Get-Soup -please
   ```

   📌 NOTE: Since you didn't specify a value for the **$soup** parameter, the default value (chicken noodle) is used.

6. To call Get-Soup specifying a value for $soup, type the following command, and then press ENTER.

   ```
   ↪ Get-Soup -please tomato
   ```

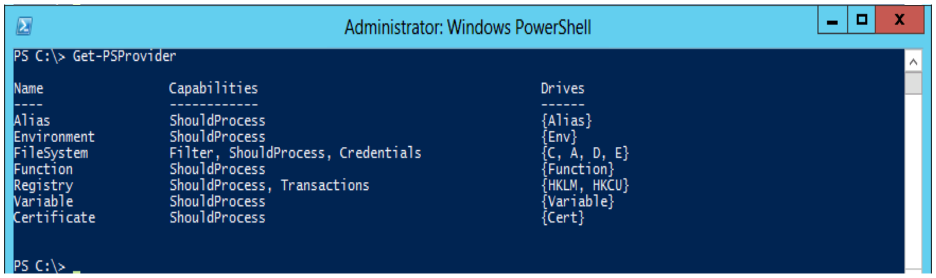
Working with providers

Windows PowerShell providers enable you to access data that would not otherwise be easily accessible at the command line. The data that a provider exposes appears in a drive, much like a hard drive, and is presented in a consistent format that resembles the file system. You can use any of the built-in cmdlets that the provider supports to manage the data in the provider drive, in addition to custom cmdlets that are designed especially for the data. By default, Windows PowerShell includes several providers that allow you to access common data stores in Windows, such as the file system, registry, and certificate store.

In this step, you will list the available providers and the drives which make use of these providers. You will also create a new drive using the registry provider.

1. On the taskbar, click **Windows PowerShell**.
2. To display a list of the available providers, type the following command, and then press ENTER.

   ```
   ↪ Get-PSProvider
   ```

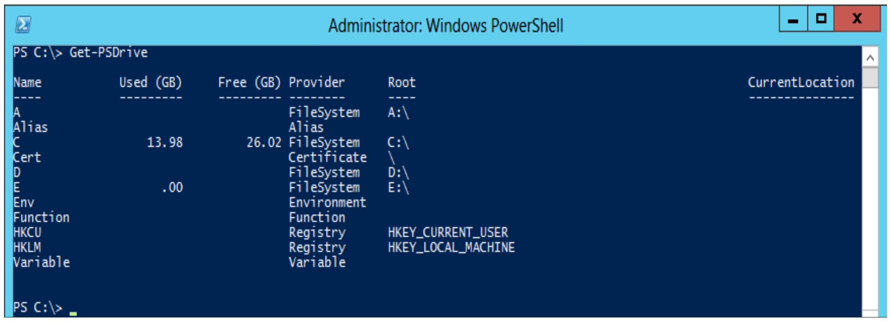


```
PS C:\> Get-PSProvider

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

PS C:\> _
```

Figure 40: List of Windows PowerShell providers

3. To display a list of the available drives, type the following command, and then press ENTER.

```
↪ Get-PSDrive
```

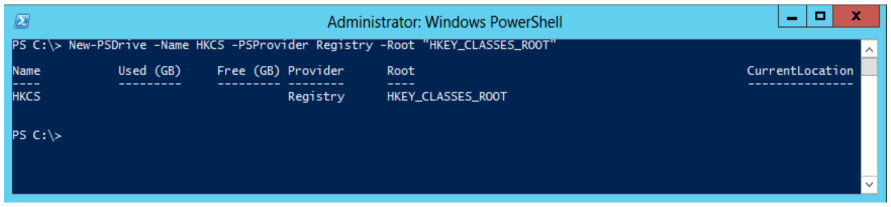


```
PS C:\> Get-PSDrive

Name           Used (GB)     Free (GB) Provider      Root                                               CurrentLocation
----           ---------     --------- --------      ----                                               ---------------
A                                      FileSystem    A:\
Alias                                  Alias
C                  13.98         26.02 FileSystem    C:\
Cert                                   Certificate   \
D                                      FileSystem    D:\
E                    .00               FileSystem    E:\
Env                                    Environment
Function                               Function
HKCU                                   Registry      HKEY_CURRENT_USER
HKLM                                   Registry      HKEY_LOCAL_MACHINE
Variable                               Variable

PS C:\> _
```

Figure 41: List of Windows PowerShell drives

4. To create a new drive for the HKEY_CLASSES_ROOT hive in the registry using the Registry provider, type the following command, and then press ENTER.

```
↪ New-PSDrive -Name HKCS -PSProvider Registry -Root
  "HKEY_CLASSES_ROOT"
```



```
PS C:\> New-PSDrive -Name HKCS -PSProvider Registry -Root "HKEY_CLASSES_ROOT"

Name           Used (GB)     Free (GB) Provider      Root                                               CurrentLocation
----           ---------     --------- --------      ----                                               ---------------
HKCS                                   Registry      HKEY_CLASSES_ROOT

PS C:\>
```

Figure 42: Adding a new Windows PowerShell drive

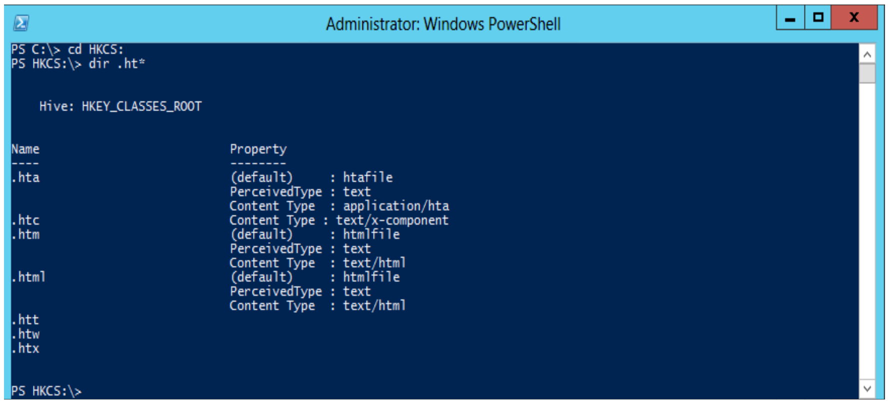
NOTE: As with the file system, the registry can also be modified using Windows PowerShell drives. Be aware that when modifying the registry, changes may cause the system to fail.

5. To browse to the newly created drive, called HKCS, as if you were working with a drive in the file system, type the following commands, pressing ENTER after each one.

```
↪ cd HKCS:
↪ dir .ht*
```

- NOTE: This will set the current location to the newly created HKCS drive, and then display the list of registry entries that match the filter expression.

- IMPORTANT: The trailing colon (:) after the drive name indicates a drive change which is different from a folder change. The colon is necessary; otherwise, Windows PowerShell will display an error.

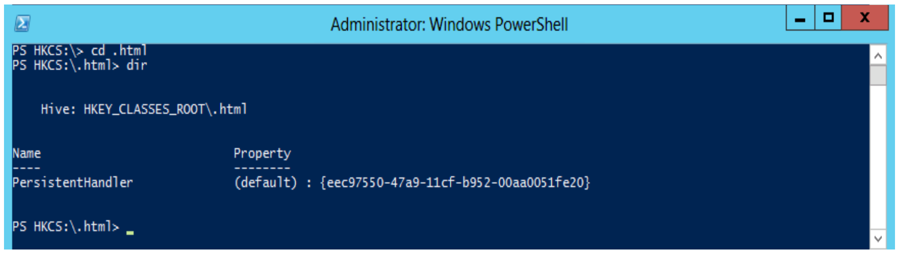


```
PS C:\> cd HKCS:
PS HKCS:\> dir .ht*


    Hive: HKEY_CLASSES_ROOT


Name                           Property
----                           --------
.hta                           (default)     : htafile
                               PerceivedType : text
                               Content Type  : application/hta
.htc                           Content Type : text/x-component
.htm                           (default)     : htmlfile
                               PerceivedType : text
                               Content Type  : text/html
.html                          (default)     : htmlfile
                               PerceivedType : text
                               Content Type  : text/html

.htt
.htw
.htx


PS HKCS:\>
```

Figure 43: Browsing the HKCS drive

6. To change the current folder to another folder inside the HKCS drive, and then list its contents, type the following commands, pressing ENTER after each line.

```
↪ cd .html
↪ dir
```



```
PS HKCS:\> cd .html
PS HKCS:\.html> dir


    Hive: HKEY_CLASSES_ROOT\.html


Name                           Property
----                           --------
PersistentHandler              (default) : {eec97550-47a9-11cf-b952-00aa0051fe20}


PS HKCS:\.html> _
```

Figure 44: Listing a key of the registry as if it were a folder in the file system

- NOTE: Many of the same commands used to manipulate the file system, such as **cd** and **dir**, also work with other providers.

Summary

In this lab, you have learned how to use the basic Windows PowerShell cmdlets, declare and manipulate variables, perform operations with strings, create and execute scripts, define functions, and work with Windows PowerShell drives and providers.