You can find the most up-to-date technical documentation on the VMware Web site at:
http://www.vmware.com/support/

The VMware Web site also provides the latest product updates.

If you have comments about this documentation, submit your feedback to:
docfeedback@vmware.com
Contents

VMware vSphere PowerCLI User's Guide 5

1 Introduction to VMware vSphere PowerCLI 7
    Microsoft PowerShell Basics 7
    VMware vSphere PowerCLI Concepts 8

2 Installing VMware vSphere PowerCLI 15
    Supported Operating Systems 15
    Supported VMware Environments 16
    Installation Prerequisites for PowerCLI 16
    Perform a Complete Installation of VMware vSphere PowerCLI 16
    Perform a Custom Installation of VMware vSphere PowerCLI 17
    Set the Properties to Support Remote Signing 17
    Uninstall VMware vSphere PowerCLI 18

3 Using PowerCLI Views from .NET 19
    PowerCLI Views 19
    Set Up the Environment to Develop PowerCLI .NET Applications 20
    Updating the Properties of PowerCLI Views 20
    Creating and Using Filters with VimClient.FindEntityView() or VimClient.FindEntityViews() 21
    Saving and Using Server Sessions with PowerCLI Views 22
    Handling Server Errors with PowerCLI Views 22

4 Sample Scripts for Managing vSphere with PowerCLI 23
    Connect to a vSphere Server 26
    Manage Virtual Machines on vSphere 26
    Add a Standalone Host to a vSphere Server 27
    Activate Maintenance Mode for a Host on vSphere 27
    Create vSphere Inventory Objects 28
    Create Virtual Machines on vSphere Using an XML Specification File 28
    Manage Virtual Machine Templates on vSphere 29
    Create and Use Snapshots on vSphere 29
    Update the Resource Configuration Settings of a Virtual Machine on vSphere 30
    Get a List of Hosts on vSphere and View Their Properties 30
    Change the Host Advanced Configuration Settings on vSphere 31
    Move a Virtual Machine to a Different Host Using VMware vMotion 31
    Move a Virtual Machine to a Different Datastore Using VMware Storage vMotion 32
    Create a Host Profile on vSphere 32
    Apply a Host Profile to a Host on vSphere 32
    Manage Statistics and Statistics Intervals on vSphere 33
    Modify the Settings of the NIC Teaming Policy for a Virtual Switch 33
The VMware vSphere PowerCLI User's Guide provides information about installing and using the VMware vSphere PowerCLI cmdlets (pronounced “commandlets”) for managing, monitoring, automating, and handling life-cycle operations for VMware® vSphere and vCloud components.

To help you start with VMware vSphere PowerCLI, this information includes descriptions of specific PowerCLI concepts and features. In addition, this information provides a set of usage examples and sample scripts.

**Intended Audience**

This book is intended for anyone who needs to install and use VMware vSphere PowerCLI. The information in this book is written for administrators and developers who are familiar with virtual machine technology and Windows PowerShell:

- Basic administrators can use cmdlets included in VMware vSphere PowerCLI to manage their vSphere and vCloud infrastructure from the command line.
- Advanced administrators can develop PowerShell scripts that can be reused by other administrators or integrated into other applications.
- Developers can use vSphere PowerCLI views to create .NET applications for managing vSphere objects.
Introduction to VMware vSphere PowerCLI

VMware vSphere PowerCLI contains snapins of cmdlets based on Microsoft PowerShell for automating vSphere and vCloud administration. It provides C# and PowerShell interfaces to VMware vSphere and vCloud APIs.

- **Microsoft PowerShell Basics** on page 7
  VMware vSphere PowerCLI is based on Microsoft PowerShell and uses the PowerShell basic syntax and concepts.

- **VMware vSphere PowerCLI Concepts** on page 8
  VMware vSphere PowerCLI cmdlets are created to automate VMware environments administration and introduce some specifics in addition to the PowerShell concepts.

### Microsoft PowerShell Basics

VMware vSphere PowerCLI is based on Microsoft PowerShell and uses the PowerShell basic syntax and concepts.

Microsoft PowerShell is both a command-line and scripting environment, designed for Windows. It uses the .NET object model and provides administrators with system administration and automation capabilities. To work with PowerShell, you run commands, called cmdlets.

- **PowerShell Command-Line Syntax** on page 7
  PowerShell cmdlets use a consistent verb-noun structure, where the verb specifies the action and the noun specifies the object to operate on.

- **PowerShell Pipelines** on page 8
  A pipeline is a series of commands separated by the pipe operator |.

- **PowerShell Wildcards** on page 8
  PowerShell has a number of pattern-matching operators called wildcards, which work on strings.

- **PowerShell Common Parameters** on page 8
  The Windows PowerShell engine implements a set of reserved parameter names, referred to as common parameters. All PowerShell cmdlets, including the PowerCLI cmdlets, support them.

### PowerShell Command-Line Syntax

PowerShell cmdlets use a consistent verb-noun structure, where the verb specifies the action and the noun specifies the object to operate on.

PowerShell cmdlets follow consistent naming patterns, which makes it easy to figure out how to construct a command if you know the object you want to work with.
All command categories take parameters and arguments. A parameter starts with a hyphen and is used to control the behavior of the command. An argument is a data value consumed by the command.

A simple PowerShell command has the following syntax: `command -parameter1 -parameter2 argument1 - argument2`.

**PowerShell Pipelines**

A pipeline is a series of commands separated by the pipe operator `|`.

Each command in the pipeline receives an object from the previous command, performs some operation on it, and then passes it to the next command in the pipeline. Objects are output from the pipeline as soon as they become available. You can cycle backwards through command history using the up arrow, and can repeat pipelines if you type them on a single line.

**PowerShell Wildcards**

PowerShell has a number of pattern-matching operators called wildcards, which work on strings.

All wildcard expressions can be used with the VMware vSphere PowerCLI cmdlets. For example, you can view a list of all files with a `.txt` extension by running `dir *.txt`. In this case, the asterisk `*` operator matches any combination of characters.

Wildcard patterns allow you to specify character ranges as well. For example, to view all files that start with the letter S or T and have a `.txt` extension, run `dir [st]*.txt`.

You can use the question mark `?` wildcard to match any single character within a sequence of characters. For example, to view all `.txt` files with names that consist of `string` and one more character at the end, run `dir string?.txt`.

**PowerShell Common Parameters**

The Windows PowerShell engine implements a set of reserved parameter names, referred to as common parameters. All PowerShell cmdlets, including the PowerCLI cmdlets, support them.

Some of the PowerShell common parameters are `Verbose`, `Debug`, `ErrorAction`, `ErrorVariable`, `OutVariable`, and `OutBuffer`. For full list of the common parameters and more details on their usage, run `Get-Help about_CommonParameters`.


- **WhatIf** Allows you to see the effects of a command without running it.
- **Confirm** Prompts for your confirmation before running a command that stops a program or service or deletes data.

**VMware vSphere PowerCLI Concepts**

VMware vSphere PowerCLI cmdlets are created to automate VMware environments administration and introduce some specifics in addition to the PowerShell concepts.

- **VMware vSphere PowerCLI Components And Versioning** on page 9
  
  VMware vSphere PowerCLI 5.0.1 consists of two components that users can install and use according to their needs and environments.

- **Loading the Script Configuration File of VMware vSphere PowerCLI** on page 10

  Starting VMware vSphere PowerCLI automatically loads the script configuration file `Initialize-PowerCLIEnvironment.psl`.
Specifying Objects in PowerCLI on page 10

In PowerCLI, all parameters that take inventory objects, datastores, OSCustomizationSpec objects, and VI5Server objects as arguments can be specified by strings and wildcards. This PowerCLI approach is called Object-by-Name (OBN) selection.

Running PowerCLI Cmdlets Asynchronously on page 11

By default, PowerCLI cmdlets return an output only after completion of the requested tasks. If you want a cmdlet to return to the command line immediately, without waiting for the tasks to complete, you can specify the RunAsync parameter.

Using Custom Scripts to Extend the Operating System Support for PowerCLI Cmdlets on page 11

Some PowerCLI features support only Windows 7, Windows Server 2008, Windows XP, Windows Server 2003, and Red Hat Enterprise Linux 5. To add support for other guest operating systems, you can use the scripts that are located in the Script folder of the PowerCLI installation directory or can add your own custom scripts.

Managing Default vSphere Server Connections with PowerCLI on page 11

By default, PowerCLI cmdlets run on the vSphere servers you are connected to, if no target servers can be determined from the provided parameters.

Customization Specification Objects in PowerCLI on page 12

PowerCLI provides two types of objects for customization specification: persistent and nonpersistent.

PowerCLI Views Cmdlets on page 12

The PowerCLI list of cmdlets includes the Get-View and Get-VIObjectByVIView cmdlets, which enable access to PowerCLI views from .NET.

Using ESXCLI with PowerCLI on page 12

PowerCLI provides you the capability to use ESXCLI through its console.

PowerCLI Inventory Provider on page 12

The Inventory Provider is designed to expose an unfiltered inventory view of the inventory items from a server.

PowerCLI Datastore Provider on page 13

The Datastore Provider is designed to provide access to the contents of one or more datastores.

VMware vSphere PowerCLI Components And Versioning

VMware vSphere PowerCLI 5.0.1 consists of two components that users can install and use according to their needs and environments.

vSphere PowerCLI 5.0 is the core component of the PowerCLI package. It contains four snapins with cmdlets for managing vSphere 5.0 features:

- **VMware.VimAutomation.Core**
  - VMware vSphere PowerCLI 5.0 provides cmdlets for automated administration of the vSphere environment.

- **VMware.VimAutomation.License**
  - VMware License PowerCLI 5.0 provides the Get-LicenseDataManager cmdlet for managing VMware License components.

- **VMware.ImageBuilder**
  - VMware ImageBuilder PowerCLI 5.0 provides cmdlets for managing depots, image profiles, and VIBs.

- **VMware.DeployAutomation**
  - VMware Auto Deploy PowerCLI 5.0 provides cmdlets that provide an interface to VMware Auto Deploy for provisioning physical hosts with ESXi software.
vCloud Director PowerCLI 1.5 is an optional component that you can install during the vSphere PowerCLI installation. It provides the following snapin:

VMware.VimAutomation.n.Cloud VMware vCloud Director PowerCLI 1.5 provides cmdlets for automating vCloud Director 1.5 features.

Loading the Script Configuration File of VMware vSphere PowerCLI

Starting VMware vSphere PowerCLI automatically loads the script configuration file Initialize-PowerCLIEnvironment.ps1. This file is located in the Scripts folder in the VMware vSphere PowerCLI installation directory. Loading the file provides access to PowerCLI cmdlets aliases, like Get-VC, Get-ESX, and to other configuration settings. Administrators can edit and extend the script to define cmdlets aliases, configure the environment, or set PowerCLI startup actions.

**NOTE** After editing the Initialize-PowerCLIEnvironment.ps1 script, you might need to sign it.

If you access the PowerCLI snapins from other tools, such as PowerShell Plus or PowerGUI, the Initialize-PowerCLIEnvironment.ps1 script configuration file is not started automatically and you must load it by typing its name in the console window: Initialize-PowerCLIEnvironment.ps1.

Specifying Objects in PowerCLI

In PowerCLI, all parameters that take inventory objects, datastores, OSCustomizationSpec objects, and VISe rver objects as arguments can be specified by strings and wildcards. This PowerCLI approach is called Object-by-Name (OBN) selection.

Instead of assigning an object name to a cmdlet parameter, users can pass the object through a pipeline or a variable. For example, the following three commands are interchangeable:

- Remove-VM -VM "Win XP SP2"
- Get-VM -Name "Win XP SP2" | Remove-VM
- Remove-VM -VM (Get-VM -Name "Win XP SP2")

**NOTE** In VMware vSphere PowerCLI, passing strings as pipeline input is not supported.

If you provide a non-existing object name, an OBN failure occurs. In such cases, PowerCLI generates a non-terminating error and runs the cmdlet ignoring the invalid name.

For more details about OBN, run help about_OBN.

Example: An OBN Failure

This example illustrates the occurrence of an OBN failure.

Set-VM -VM "VM1", "VM2", "VM3" -Server $server1, $server2 -MemoryMB 512

If the VM2 virtual machine does not exist on either of the specified servers, PowerCLI generates a non-terminating error and applies the command only on the VM1 and VM2 virtual machines.
Running PowerCLI Cmdlets Asynchronously

By default, PowerCLI cmdlets return an output only after completion of the requested tasks. If you want a cmdlet to return to the command line immediately, without waiting for the tasks to complete, you can specify the RunAsync parameter.

When you specify the RunAsync parameter, the cmdlet returns Task objects instead of its usual output. The Status property of a returned Task object contains a snapshot of the task’s initial state. This state is not updated automatically and has the values Error, Queued, Running, or Success. You can refresh a task state by retrieving the task object with the Get-Task cmdlet. If you want to observe the progress of a running task and wait for its completion before running other commands, use the Wait-Task cmdlet.

**Note**  In VMware vSphere PowerCLI, the RunAsync parameter affects only a cmdlet’s invocation and does not control whether the initiated tasks run consecutively or in parallel. For example, the Remove-VM cmdlet might remove the specified virtual machines simultaneously or consecutively depending on the PowerCLI internal design. To make sure that tasks initiated by a cmdlet run consecutively, run the cmdlet in a loop, each time applying it to a single object.

**Example: Running Remove-VM with and without the RunAsync Parameter**

```powershell
Remove-VM $vmList
```

The command returns no output when all virtual machines stored in the $vmList variable are removed, irrespective of whether they are removed simultaneously.

```powershell
Remove-VM $vmList -RunAsync
```

The command returns immediately an output that consists of one or more Task objects.

Using Custom Scripts to Extend the Operating System Support for PowerCLI Cmdlets

Some PowerCLI features support only Windows 7, Windows Server 2008, Windows XP, Windows Server 2003, and Red Hat Enterprise Linux 5. To add support for other guest operating systems, you can use the scripts that are located in the Script folder of the PowerCLI installation directory or can add your own custom scripts.

When adding new scripts, use the following file naming guidelines:

- Scripts that extend the operating system support for Get-VMGuestNetworkInterface, Set-VMGuestNetworkInterface, Get-VMGuestRoute, New-VMGuestRoute, Remove-VMGuestRoute, and Set-VMGuestRoute must follow the file-naming convention $CmdletName_OSIdentifier, where OSIdentifier is the guest family or the guest ID as returned by Get-VMGuest, and CmdletName is the cmdlet name written without a hyphen, for example GetVMGuestRoute.

- Scripts that extend the operating system support for resizing the hard disk by using Set-HardDisk must follow the file naming convention GuestDiskExpansion_OSIdentifier, where OSIdentifier is the guest family or the guest ID (as returned by Get-VMGuest).

Managing Default vSphere Server Connections with PowerCLI

By default, PowerCLI cmdlets run on the vSphere servers you are connected to, if no target servers can be determined from the provided parameters.

When you connect to a vSphere server by using Connect-VIServer, the server connection is stored in the $DefaultVIServers array variable. This variable contains all connected servers for the current PowerCLI session. To remove a server from the $DefaultVIServers variable, you can either use Disconnect-VIServer to close all active connections to this server, or modify the value of $DefaultVIServers manually.
Customization Specification Objects in PowerCLI

PowerCLI provides two types of objects for customization specification: persistent and nonpersistent.

**Persistent customization**

Persistent customization specification objects are stored on the vSphere server. All persistent customization specifications created by using vSphere Client or PowerCLI 4.1 or later are encrypted. Encrypted customization specifications can be applied only by the server that has encrypted them.

**Nonpersistent customization**

Nonpersistent customization specification objects exist only inside the current PowerShell process. Nonpersistent customization specification objects are not encrypted, but cloning them to a vSphere server encrypts them.

PowerCLI Views Cmdlets

The PowerCLI list of cmdlets includes the `Get-View` and `Get-VIObjectByVIView` cmdlets, which enable access to PowerCLI views from .NET.

To find more information about PowerCLI views, see “PowerCLI Views,” on page 19.

Using the PowerCLI views cmdlets for low-level VMware vSphere management requires some knowledge of both PowerShell scripting and the VMware vSphere APIs.

Using ESXCLI with PowerCLI

PowerCLI provides you the capability to use ESXCLI through its console.

PowerCLI provides two approaches for working with ESXCLI:

- Through the `Get-ESXCLI` cmdlet, which provides direct access to the ESXCLI namespaces, applications, and commands.

- Through .NET methods, which you use to create managed objects that correspond to specific ESXCLI applications. To access the ESXCLI, you can call methods on these managed objects.

**NOTE**

To call a method of an ESXCLI object, you must provide values for all parameters. If you want to omit a given parameter, specify `$null` for it.

PowerCLI Inventory Provider

The Inventory Provider is designed to expose an unfiltered inventory view of the inventory items from a server. It enables navigation and file-style management of the VMware vSphere inventory. By creating a PowerShell drive based on a managed object (such as a datacenter), you can obtain a view of its contents and the relationships between the items. In addition, you can move, rename, or delete objects by running commands from the PowerCLI console.

When you connect to a server with `Connect-VIServer`, the cmdlet builds two default inventory drives: `vi` and `vis`. The `vi` inventory drive shows the inventory on the last connected server. The `vis` drive contains the inventory of all vSphere servers connected within the current PowerCLI session.

You can use the default inventory drives or create custom drives based on the default ones.
PowerCLI Datastore Provider

The Datastore Provider is designed to provide access to the contents of one or more datastores.

The items in a datastore are files that contain configuration, virtual disk, and the other data associated with a virtual machine.

When you connect to a server with Connect-VIServer, the cmdlet builds two default datastore drives: `vmstores` and `vmstore`. The `vmstore` drive provides a list of the datastores available on the vSphere server that you last connected to. The `vmstores` drive contains all datastores available on all vSphere servers that you connected to within the current PowerCLI session.

You can use the default inventory drives or create custom drives based on the default ones.
Installing VMware vSphere PowerCLI

You can install VMware vSphere PowerCLI components on all supported Windows operating systems. After you have installed the package on your machine, you can run PowerCLI cmdlets to connect to your vSphere or vCloud system by specifying the appropriate connection parameters.

- **Supported Operating Systems** on page 15
  You can install VMware vSphere PowerCLI only on Windows operating systems.

- **Supported VMware Environments** on page 16
  You can use the VMware vSphere PowerCLI 5.0.1 components to manage all supported vSphere and vCloud environments.

- **Installation Prerequisites for PowerCLI** on page 16
  Before installing VMware vSphere PowerCLI, verify that you have installed the required software on the same machine.

- **Perform a Complete Installation of VMware vSphere PowerCLI** on page 16
  By performing a complete PowerCLI installation, you install all PowerCLI components available in the installation package. This installation setup type requires the most disk space.

- **Perform a Custom Installation of VMware vSphere PowerCLI** on page 17
  By performing a custom PowerCLI installation, you can choose the PowerCLI components that you want to install.

- **Set the Properties to Support Remote Signing** on page 17
  For security reasons, Windows PowerShell supports an execution policy feature. It determines whether scripts are allowed to run and whether they must be digitally signed. By default, the execution policy is set to **Restricted**, which is the most secure policy.

- **Uninstall VMware vSphere PowerCLI** on page 18
  You can uninstall VMware vSphere PowerCLI components from your Windows system by using **Add or Remove Programs**.

### Supported Operating Systems

You can install VMware vSphere PowerCLI only on Windows operating systems. VMware vSphere PowerCLI 5.0.1 is supported on the following operating systems:

- Windows 7 Service Pack 1 (32-bit and 64-bit)
- Windows Server 2008 R2 Service Pack 1 (32-bit and 64-bit)
- Windows XP Service Pack 2 (32-bit and 64-bit) and Service Pack 3 (32-bit)
Windows Server 2003 R2 (32-bit and 64-bit)

**Note** The list of supported operating systems might not apply to some PowerCLI cmdlets for managing guest systems.

### Supported VMware Environments

You can use the VMware vSphere PowerCLI 5.0.1 components to manage all supported vSphere and vCloud environments.

vSphere PowerCLI 5.0.1 is compatible with the following vSphere environments:

- VMware ESXi 5.0
- vCenter Server 5.0
- VMware ESX 4.1 Update 2 and vCenter Server 4.1 Update 2
- VMware ESXi 4.1 Update 2
- VMware ESX 4.0 Update 4 and vCenter Server 4.0 Update 4
- VMware ESX 4.0i Update 4
- VMware ESX 3.5 Update 5
- VMware ESXi 3.5 Update 5
- VMware VirtualCenter 2.5 Update 6

vCloud Director PowerCLI 1.5 is compatible with VMware vCloud Director 1.5.

### Installation Prerequisites for PowerCLI

Before installing VMware vSphere PowerCLI, verify that you have installed the required software on the same machine.

To install VMware vSphere PowerCLI 5.0.1, you must first install:

- .NET 2.0, 3.0, or 3.5 with Service Pack 1
- Windows PowerShell 2.0

### Perform a Complete Installation of VMware vSphere PowerCLI

By performing a complete PowerCLI installation, you install all PowerCLI components available in the installation package. This installation setup type requires the most disk space.

**Prerequisites**

Before installing PowerCLI, see “Installation Prerequisites for PowerCLI,” on page 16.

**Procedure**

1. Download the latest version of VMware vSphere PowerCLI from VMware Web site.
2. Navigate to the Web folder that contains the PowerCLI installer file you downloaded and double-click the executable file.
3. On the Welcome page, click Next.
4. Accept the license agreement terms and click Next.
5. On the Destination Folder page, select the location to install VMware vSphere PowerCLI and click Next.
6. On the Setup Type page, select Complete and click Next.
Perform a Custom Installation of VMware vSphere PowerCLI

By performing a custom PowerCLI installation, you can choose the PowerCLI components that you want to install.

Prerequisites

Before installing PowerCLI, see “Installation Prerequisites for PowerCLI,” on page 16.

Procedure

1. Download the latest version of VMware vSphere PowerCLI from VMware Web site.
2. Navigate to the Web folder that contains the PowerCLI installer file you downloaded and double-click the executable file.
3. On the Welcome page, click Next.
4. Accept the license agreement terms and click Next.
5. On the Destination Folder page, select the location to install VMware vSphere PowerCLI and click Next.
6. On the Setup Type page, select Custom and click Next.
7. On the Custom Setup page, select the components that you want to install and click Next.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vSphere PowerCLI</td>
<td>Installs a set of cmdlets for managing vSphere features. This PowerCLI component is mandatory and selected by default.</td>
</tr>
<tr>
<td>vCloud Director PowerCLI</td>
<td>Installs a set of cmdlets for managing vCloud Director features.</td>
</tr>
</tbody>
</table>

8. On the Ready to Install the Program page, click Install to proceed with the installation.
9. Click Finish to complete the installation process.

What to do next

“Set the Properties to Support Remote Signing,” on page 17.

Set the Properties to Support Remote Signing

For security reasons, Windows PowerShell supports an execution policy feature. It determines whether scripts are allowed to run and whether they must be digitally signed. By default, the execution policy is set to Restricted, which is the most secure policy.

If you want to run scripts or load configuration files, you can change the execution policy by using the Set-ExecutionPolicy cmdlet. For more information about the execution policy and script digital signing in Windows PowerShell, run Get-Help About_Signing.

Procedure

1. Select Start > Programs > VMware > VMware vSphere PowerCLI.
   The VMware vSphere PowerCLI console window opens.
2. In the VMware vSphere PowerCLI console window, run Set-ExecutionPolicy RemoteSigned.
Uninstall VMware vSphere PowerCLI

You can uninstall VMware vSphere PowerCLI components from your Windows system by using Add or Remove Programs.

Prerequisites
Close the PowerCLI application before uninstalling the software.

Procedure
1. Select Add or Remove Programs from Control panel.
2. Select VMware vSphere PowerCLI from the list and click Change.
3. On the Program Maintenance page, select Remove and click Next.
4. Click Remove.
Using PowerCLI Views from .NET

You can use .NET to access and use PowerCLI views. Views are .NET objects that provide C# and PowerShell interface to vSphere APIs.

With PowerCLI views, you can develop .NET applications for creating, customizing, or managing vSphere inventory objects.

- **PowerCLI Views** on page 19
  PowerCLI views are .NET objects that correspond to server-side managed objects. Each operation defined on a server managed object has a corresponding view method.

- **Set Up the Environment to Develop PowerCLI .NET Applications** on page 20
  Before creating and running .NET applications for PowerCLI, you must set up your developmental environment.

- **Updating the Properties of PowerCLI Views** on page 20
  The properties of a PowerCLI view contain information about the state of the server-side object at the time the view was created.

- **Creating and Using Filters with VimClient.FindEntityView() or VimClient.FindEntityViews()** on page 21
  You can use filters to reduce large sets of output data by retrieving only the objects that correspond to the specified filter criteria. You can use PowerCLI views to define and use filters to select specific objects based on property values.

- **Saving and Using Server Sessions with PowerCLI Views** on page 22
  With PowerCLI you can save your server session and restore it later. The VimClient class includes several methods for saving and restoring server sessions. This enables you to maintain sessions across applications.

- **Handling Server Errors with PowerCLI Views** on page 22
  Error reporting helps you track and handle server errors. vSphere Web Services API server errors are reported as SOAP exceptions that contain a SoapFault object.

**PowerCLI Views**

PowerCLI views are .NET objects that correspond to server-side managed objects. Each operation defined on a server managed object has a corresponding view method.

A PowerCLI view has the following characteristics:

- It includes properties and methods that correspond to the properties and operations of the server-side managed objects.
It is a static copy of a server-side managed object and is not automatically updated when the object on the server changes.

- It includes additional methods other than the operations offered in the server-side managed object.

Set Up the Environment to Develop PowerCLI .NET Applications

Before creating and running .NET applications for PowerCLI, you must set up your developmental environment.

Procedure

1. In Visual Studio 2005 .NET or later, create a new project or open an existing project.
2. Add a reference to the vSphere API .NET Library (VMware.Vim.dll) from the PowerCLI installation folder.

Now you can use VimClient and other VMware.Vim namespace classes to manage your vSphere inventory.

Updating the Properties of PowerCLI Views

The properties of a PowerCLI view contain information about the state of the server-side object at the time the view was created.

In a production environment, the state of managed objects on the server changes constantly. However, the property values of the objects are not updated automatically. You can synchronize the values of client-side views with the corresponding server-side objects by using the UpdateViewData() method.

Example: Using the UpdateViewData() Method to Refresh a View Object Data

The following code example refreshes the power state information of a virtual machine by using UpdateViewData() method.

```csharp
using VMware.Vim;
using System.Collections.Specialized;
namespace Samples {
    public class Example2_2 {
        public void PowerOffVM() {
            VimClient client = new VimClient();
            ... 

            var vmList = client.FindEntityViews(typeof(VirtualMachine), null, filter, null);
            // Power off the virtual machines.
            foreach (VirtualMachine vm in vmList) {
                // Refresh the state of each view.
                vm.UpdateViewData();
                if (vm.Runtime.PowerState == VirtualMachinePowerState.poweredOn) {
                    vm.PowerOffVM();
                    Console.WriteLine("Stopped virtual machine: {0}", vm.Name);
                } else {
                    Console.WriteLine("Virtual machine {0} power state is: {1}", vm.Name, 
                                    vm.Runtime.PowerState);
                }
            }
        }
    }
}
```
Creating and Using Filters with VimClient.FindEntityView() or VimClient.FindEntityViews()

You can use filters to reduce large sets of output data by retrieving only the objects that correspond to the specified filter criteria. You can use PowerCLI views to define and use filters to select specific objects based on property values.

To apply a filter to the results of VimClient.FindEntityView() or VimClient.FindEntityViews(), you can supply an optional filter parameter. The value of the parameter is a NameValueCollection object containing one or more pairs of filter criteria. Each of the criteria consists of a property path and a match value. The match value can be either a string, or a regular expression object. If the match value is a string, the property value of the target objects must be exactly the same as the string.

Example: Filtering Virtual Machines by Power State

The following commands retrieve all powered-off virtual machines.

```csharp
NameValueCollection filter = new NameValueCollection();
filter.Add("Runtime.PowerState", "PoweredOff")
```

Example: Filtering Objects by Name

The following commands retrieve all virtual machines with names that start with Test.

```csharp
NameValueCollection filter = new NameValueCollection();
filter.Add("name", "^Test")
```

Example: Filter for Creating Views of Windows Virtual Machines Only

The following example uses VimClient.FindEntityViews() in combination with a filter. It retrieves a list of all Windows virtual machines in the virtual environment.

```csharp
NameValueCollection filter = new NameValueCollection();
filter.Add("Config.GuestFullName", "Windows");

IList EntityViewBase vmList =
    client1.FindEntityViews(typeof(VirtualMachine), null, filter, null);

    // Print VM names
    foreach (VirtualMachine vm in vmList) {
        Console.WriteLine(vm.Name);
    }
```

Example: Multiple Criteria Filter

This example uses a filter with multiple criteria. It retrieves all powered-on Windows virtual machines.

```csharp
NameValueCollection filter = new NameValueCollection();
filter.Add("Config.GuestFullName", "Windows");

IList EntityViewBase vmList =
    client1.FindEntityViews(typeof(VirtualMachine), null, filter, null);

    // Print VM names
    foreach (VirtualMachine vm in vmList) {
        Console.WriteLine(vm.Name);
    }
```
Saving and Using Server Sessions with PowerCLI Views

With PowerCLI you can save your server session and restore it later. The VimClient class includes several methods for saving and restoring server sessions. This enables you to maintain sessions across applications.

Instead of storing passwords in applications, you can call the `LoadSession()` method with the name of the session file. The session file does not expose password information, and this enhances security.

**Example: Saving a Session to a File**

This example illustrates how to save a server session to a file by calling `SaveSession()` with the file name.

```csharp
VimClient client1 = new VimClient();
client1.Connect("https://hostname/sdk");
client1.Login("user", "pass");
client1.SaveSession("VimSession.txt");
```

**Example: Loading a Session from a File**

This example illustrates how to load a server session in another application by calling `LoadSession()` with the name of the session file.

```csharp
VimClient client2 = new VimClient();
client2.Connect("https://hostname/sdk");
client2.LoadSession("VimSession.txt");
client2.FindEntityView(typeof(VirtualMachine), null, null, null);
```

Handling Server Errors with PowerCLI Views

Error reporting helps you track and handle server errors. vSphere Web Services API server errors are reported as SOAP exceptions that contain a `SoapFault` object.

Using PowerCLI views provides additional error handling by translating the `SoapFault` object from the `SoapException.Detail` property into a `MethodFault` descendant object and throwing a `VimException` exception.

**Example: Simple Pattern for Error Handling**

The following example illustrates a basic pattern implementation of error handling with PowerCLI views.

```csharp
try {
    // call operations
} catch (VimException ex) {
    if (ex.MethodFault is InvalidLogin) {
        // Handle Invalid Login error
    } else {
        // Handle other server errors
    }
} catch (Exception e) {
    // Handle user code errors
}
```
Sample Scripts for Managing vSphere with PowerCLI

This section provides sample scripts that illustrate managing vSphere with PowerCLI cmdlets.

- **Connect to a vSphere Server** on page 26
  To run PowerCLI cmdlets on vSphere and perform administration or monitoring tasks, first establish a connection to an ESX instance or a vCenter Server.

- **Manage Virtual Machines on vSphere** on page 26
  With VMware vSphere PowerCLI, you can automate various administration tasks on virtual machines, for example retrieving information, shutting down and powering off virtual machines.

- **Add a Standalone Host to a vSphere Server** on page 27
  You can add standalone hosts to a vSphere server by using the `Add-VMHost` cmdlet. After adding the hosts, you will be able to manage them through the vSphere server.

- **Activate Maintenance Mode for a Host on vSphere** on page 27
  To complete some specific administration tasks, you might need to activate maintenance mode for a host. On vSphere, you can activate maintenance mode by using the `Set-VMHost` cmdlet.

- **Create vSphere Inventory Objects** on page 28
  By using PowerCLI cmdlets, you can automate creating different inventory objects on vSphere.

- **Create Virtual Machines on vSphere Using an XML Specification File** on page 28
  You can use a specification provided in an XML file to automate the creation of virtual machines on vSphere.

- **Manage Virtual Machine Templates on vSphere** on page 29
  You can use PowerCLI to create virtual machines templates and convert them to virtual machines on vSphere.

- **Create and Use Snapshots on vSphere** on page 29
  You can use the `Snapshot` parameter of `Get-VM` to take a snapshot of virtual machines and then revert the virtual machines' states back to the snapshot.

- **Update the Resource Configuration Settings of a Virtual Machine on vSphere** on page 30
  You can use the `Set-VMResourceConfiguration` cmdlet to modify the resource configuration properties of a virtual machine, including memory, CPU shares, and other settings.

- **Get a List of Hosts on vSphere and View Their Properties** on page 30
  With PowerCLI, you can get information about all available hosts in a datacenter and view their properties.
- **Change the Host Advanced Configuration Settings on vSphere** on page 31
  You can modify host configuration, including advanced settings related to virtual machine migration, and apply them to another host.

- **Move a Virtual Machine to a Different Host Using VMware vMotion** on page 31
  You can migrate a virtual machine between vSphere hosts by using VMware vMotion.

- **Move a Virtual Machine to a Different Datastore Using VMware Storage vMotion** on page 32
  You can migrate a virtual machine between datastores using the VMware Storage vMotion feature of vSphere.

- **Create a Host Profile on vSphere** on page 32
  The VMware Host Profiles feature enables you to create standard configurations for ESX and ESXi hosts. With PowerCLI, you can automate creation and modifying of host profiles.

- **Apply a Host Profile to a Host on vSphere** on page 32
  To simplify operational management of large-scale environments, you can apply standard configurations called host profiles to hosts on vSphere. If you want to set up a host to use the same host profile as a reference host, you can attach the host to a profile.

- **Manage Statistics and Statistics Intervals on vSphere** on page 33
  You can use the PowerCLI cmdlets to automate tasks for viewing and managing statistics for vSphere inventory objects.

- **Modify the Settings of the NIC Teaming Policy for a Virtual Switch** on page 33
  You can specify the NIC teaming policy on a vSwitch. The NIC teaming policy determines the load balancing and failover settings of a virtual switch and allows you to specify unused NICs.

- **Create a Virtual Appliance on vSphere** on page 34
  With PowerCLI, you can create and manage virtual appliances.

- **Modify the Properties of a Virtual Appliance** on page 34
  With PowerCLI, you can start and stop virtual appliances, and modify their properties.

- **Export or Import Virtual Appliances** on page 34
  You can import and export virtual appliances to OVA and OVF files.

- **Configure a Network Interface** on page 35
  You can modify the IP and routing configuration settings of a guest network interface.

- **Add and Configure a Guest Route** on page 35
  You can add new guest routes for virtual machines and modify the routes properties.

- **Create an iSCSI Host Storage** on page 36
  For a host, you can enable iSCSI, add iSCSI targets, and create new host storages.

- **Add Passthrough Devices to a Host and Virtual Machine** on page 36
  You can get information about existing passthrough devices and add new SCSI and PCI devices to virtual machines and hosts.

- **Create a Custom Property Based on an Extension Data Property** on page 37
  You can create custom properties to add more information to vSphere objects. Custom properties based on extension data properties correspond directly to the property of the corresponding .NET view object.

- **Create a Script-Based Custom Property for a vSphere Object** on page 37
  You can create a custom property by writing a script and providing a name for the property. The script evaluates when the custom property is called for first time.
Apply a Customization Object to a Cloned Virtual Machine on page 37
You can apply a custom configuration to a cloned virtual machine by using a customization object.

Modify the Default NIC Mapping Object of a Customization Specification on page 38
You can modify the default NIC mapping object of a customization specification and apply the specification on a newly created virtual machine.

Modify Multiple NIC Mapping Objects of a Customization Specification on page 38
You can modify multiple NIC mapping objects of a customization specification and apply the specification to an existing virtual machine.

Create a vSphere Role and Assign Permissions to a User on page 39
With PowerCLI, you can automate management of vSphere permissions, roles, and privileges.

View the Action Triggers for an Alarm on vSphere on page 40
You can see which action triggers are configured for an alarm.

Create and Modify Alarm Actions and Alarm Triggers on vSphere on page 40
With PowerCLI, you can create and modify vSphere alarm actions and alarm triggers.

Remove Alarm Actions and Triggers on page 41
In some cases, you might want to remove obsolete alarm actions and triggers.

Create and Modify Advanced Settings for a Cluster on page 41
You can customize the behavior of a cluster on vSphere by creating and modifying custom advanced settings for it.

Modify the vCenter Server Email Configuration on page 42
You can modify the email configuration settings of a vCenter Server.

Modify the vCenter Server SNMP Configuration on page 42
To use SNMP, you must first configure the SNMP settings of the vCenter Server.

Use Esxtop to Get Information on the Virtual CPUs of a Virtual Machine on page 42
You can use the Get-ESXTop cmdlet to retrieve real-time data for troubleshooting performance problems.

Filter vSphere Objects with Get-View on page 43
You can use the Get-View cmdlet to filter vSphere objects before performing various actions on them.

Populate a View Object with Get-View on page 43
You can use the Get-View cmdlet to update a view object by using the information from a previously called managed object.

Update the State of a Server-Side Object on page 44
You can use the Get-View cmdlet to update server-side objects.

Reboot a Host with Get-View on page 45
You can reboot a host by using its corresponding view object.

Modify the CPU Levels of a Virtual Machine with Get-View and Get-VIObjectByVIView on page 45
You can modify the CPU levels of a virtual machine using a combination of the Get-View and Get-VIObjectByVIView cmdlets.

Browse the Default Inventory Drive on page 46
You can browse the default inventory drive and view its contents.

Create a New Custom Inventory Drive on page 46
In addition to the default drive, you can create new custom inventory drives by using the New-PSDrive cmdlet.
Manage Inventory Objects Through Inventory Drives on page 46
You can use the PowerCLI Inventory Provider to browse, modify, and remove inventory objects from inventory drives.

Browse the Default Datastore Drives on page 47
You can use the PowerCLI Datastore Provider to browse the default datastore drives: vmstore and vmstores.

Create a New Custom Datastore Drive on page 47
You can use the PowerCLI datastore provider to create custom datastore drives.

Manage Datastores Through Datastore Drives on page 48
You can use the PowerCLI Datastore Provider to browse datastores from datastore drives.

Connect to a vSphere Server
To run PowerCLI cmdlets on vSphere and perform administration or monitoring tasks, first establish a connection to an ESX instance or a vCenter Server.

Prerequisites
If you use a proxy server for the connection, verify that it is configured properly, so that the connection is kept alive long enough for long PowerCLI tasks to complete running.

**NOTE** You can remove a proxy by running `Set-PowerCLIConfiguration -ProxyPolicy NoProxy`.

Procedure
- Run `Connect-VIServer` with the server name and valid credentials.
  ```
  Connect-VIServer -Server esx3.example.com -Protocol http -User admin -Password pass
  ```
- You can have more than one connections to the same server. For more information, see “Managing Default vSphere Server Connections with PowerCLI” on page 11.

Manage Virtual Machines on vSphere
With VMware vSphere PowerCLI, you can automate various administration tasks on virtual machines, for example retrieving information, shutting down and powering off virtual machines.

Procedure
1. View all virtual machines on the target system.
   ```
   Get-VM
   ```
2. Save the name and the power state properties of the virtual machines in the ResourcePool resource pool into a file named `myVMProperties.txt`.
   ```
   $respool = Get-ResourcePool ResourcePool
   Get-VM -Location $respool | Select-Object Name, PowerState > myVMProperties.txt
   ```
3. Start the VM virtual machine.
   ```
   Get-VM VM | Start-VM
   ```
4. Get information of the guest OS of the VM virtual machine.
   ```
   Get-VMGuest VM | fc
   ```
5. Shut down the OS of the VM virtual machine.
   ```
   Shutdown-VMGuest VM
   ```
6. Power off the VM virtual machine.
   Stop-VM VM

7. Move the virtual machine VM from the Host01 host to the Host02 host.
   Get-VM -Name VM -Location Host01 | Move-VM -Destination Host02

   **Note** If the virtual machine you want to move across hosts is powered on, it must be located on a shared storage registered as a datastore on both the original and the new host.

---

### Add a Standalone Host to a vSphere Server

You can add standalone hosts to a vSphere server by using the `Add-VMHost` cmdlet. After adding the hosts, you will be able to manage them through the vSphere server.

#### Prerequisites

Connect to a vSphere server.

#### Procedure

1. View all hosts on the vSphere server that you have established a connection with.
   Get-VMHost

2. Add the Host standalone host.
   Add-VMHost -Name Host -Location (Get-Datacenter DC) -User root -Password pass

---

### Activate Maintenance Mode for a Host on vSphere

To complete some specific administration tasks, you might need to activate maintenance mode for a host. On vSphere, you can activate maintenance mode by using the `Set-VMHost` cmdlet.

#### Prerequisites

Connect to a vSphere server.

#### Procedure

1. Save the Host host object as a variable.
   $host = Get-VMHost -Name Host

2. Get the cluster to which Host belongs and save the cluster object as a variable.
   $hostCluster = Get-Cluster -VMHost $host

3. Start a task that activates maintenance mode for the Host host and save the task object as a variable.
   $updateHostTask = Set-VMHost -VMHost $host -State "Maintenance" -RunAsync

   **Note** If the host is not automated or is partially automated and has powered-on virtual machines running on it, you must specify the RunAsync parameter and wait until all powered-on virtual machines are relocated or powered off before applying DRS recommendations.

4. Get and apply the recommendations generated by DRS.
   Get-DrsRecommendation -Cluster $hostCluster | where {$_.Reason -eq "Host is entering maintenance mode"} | Apply-DrsRecommendation

5. Get the task output object and save it as a variable.
   $myUpdatedHost = Wait-Task $updateHostTask
Create vSphere Inventory Objects

By using PowerCLI cmdlets, you can automate creating different inventory objects on vSphere.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. Get the inventory root folder and create a new folder called Folder in it.
   
   ```
   $folder = Get-Folder -NoRecursion | New-Folder -Name Folder
   ```

2. Create a new datacenter called DC in the Folder folder.
   
   ```
   New-Datacenter -Location $folder -Name DC
   ```

3. Create a folder called Folder1 under DC.
   
   ```
   Get-Datacenter DC | New-Folder -Name Folder1
   $folder1 = Get-Folder -Name Folder1
   ```

4. Create a new cluster Cluster1 in the Folder1 folder.
   
   ```
   New-Cluster -Location $folder1 -Name Cluster1 -DrsEnabled -DrsAutomationLevel FullyAutomated
   ```

5. Add a host in the cluster by using the `Add-VMHost` command, and provide credentials when prompted.
   
   ```
   $host1 = Add-VMHost -Name 10.23.112.345 -Location ( Get-Cluster Cluster1 )
   ```

6. Create a resource pool in the cluster's root resource pool.
   
   ```
   $myClusterRootRP = Get-ResourcePool -Location ( Get-Cluster Cluster1 ) -Name Resources
   New-ResourcePool -Location $myClusterRootRP -Name MyRP01 -CpuExpandableReservation $true -CpuReservationMhz 500 -CpuSharesLevel high -MemExpandableReservation $true -MemReservationMB 500 -MemSharesLevel high
   ```

7. Create a virtual machine asynchronously.
   
   ```
   $vmCreationTask = New-VM -Name VM2 -VMHost $host1 -ResourcePool MyRP01 -DiskMB 4000 -MemoryMB 256 -RunAsync
   ```

The `RunAsync` parameter specifies that the command runs asynchronously. This means that in contrast to a synchronous operation, you do not have to wait for the process to complete before supplying the next command at the command line.

Create Virtual Machines on vSphere Using an XML Specification File

You can use a specification provided in an XML file to automate the creation of virtual machines on vSphere.

**Prerequisites**

Connect to a vSphere server.

The `myVM.xml` file must be present with the following content:

```xml
<CreateVM>
   <VM>
      <Name>MyVM1</Name>
      <HDDCapacity>10000</HDDCapacity>
   </VM>
</CreateVM>
```
<Name>MyVM2</Name>
<HDDCapacity>10000</HDDCapacity>
</VM>
</CreateVM>

Procedure
1. Read the content of the myVM.xml file.
   ```powershell
   [xml]$s = Get-Content myVM.xml
   ```
2. Create the virtual machines.
   ```powershell
   $s.CreateVM.VM | foreach { New-VM -VMHost 192.168.10.11 -Name $_.Name -DiskMB $_.HDDCapacity}
   ```

Manage Virtual Machine Templates on vSphere

You can use PowerCLI to create virtual machines templates and convert them to virtual machines on vSphere.

**Note** A virtual machine template is a reusable image created from a virtual machine. The template, as a derivative of the source virtual machine, includes virtual hardware components, an installed guest operating system, and software applications.

Prerequisites
Connect to a vSphere server.

Procedure
1. Create a template from the VM1 virtual machine.
   ```powershell
   New-Template -VM VM1 -Name VM1Template -Location (Get-Datacenter DC )
   ```
2. Convert the VM1Template template for use by a virtual machine named VM3.
   ```powershell
   Get-Template VM1Template | Set-Template -ToVM -Name VM3
   ```
3. Create a template from the VM2 virtual machine.
   ```powershell
   New-Template -VM VM2 -Name VM2Template -Location (Get-Datacenter DC )
   ```
4. Convert the VM2Template template to a virtual machine named VM4.
   ```powershell
   Get-Template VM2Template | Set-Template -ToVM -Name VM4
   ```
5. Convert the VM4 virtual machine to a template.
   ```powershell
   Set-VM -VM VM4 -ToTemplate -Name "VM4Template"
   ```
6. Create a template called VM3Template by cloning VM2Template.
   ```powershell
   Get-Template VM2Template | New-Template -Name VM3Template –VMHost $targetVMHost
   ```

Create and Use Snapshots on vSphere

You can use the Snapshot parameter of Get-VM to take a snapshot of virtual machines and then revert the virtual machines' states back to the snapshot.

**Note** A snapshot captures the memory, disk, and settings state of a virtual machine at a particular moment. When you revert to a snapshot, you return all these items to the state they were in at the time you took that snapshot.

Prerequisites
Connect to a vSphere server.
Procedure

1. Take a snapshot of all virtual machines in the MyRP01 resource pool.

   Get-ResourcePool MyRP01 | Get-VM | New-Snapshot -Name InitialSnapshot

   The Location parameter takes arguments of the VIContainer type, on which Cluster, Datacenter, Folder, ResourcePool, and VMHost object types are based. Therefore, the Location parameter can use arguments of all these types.

2. Revert all virtual machines in the MyRP01 resource pool to the InitialSnapshot snapshot.

   $VMs = Get-ResourcePool MyRP01 | Get-VM
   foreach( $vm in $VMs ) { Set-VM -VM $vm -Snapshot –Snapshot InitialSnapshot }

**Update the Resource Configuration Settings of a Virtual Machine on vSphere**

You can use the Set-VMResourceConfiguration cmdlet to modify the resource configuration properties of a virtual machine, including memory, CPU shares, and other settings.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. View the resource configuration for the VM1 virtual machine.

   Get-VMResourceConfiguration -VM VM1

2. View the disk share of the VM1 virtual machine.

   Get-VMResourceConfiguration -VM VM1 | Format-Custom –Property DiskResourceConfiguration

3. Change the memory share of the VM1 virtual machine to low.

   Get-VM VM1 | Get-VMResourceConfiguration | Set-VMResourceConfiguration –MemSharesLevel low

4. Change the CPU shares of the VM1 virtual machine to high.

   Get-VM VM1 | Get-VMResourceConfiguration | Set-VMResourceConfiguration –CpuSharesLevel high

5. Change the disk share of the VM1 virtual machine to 100.

   $vm1 = Get-VM VM1
   $vm1disk = Get-HardDisk $vm1
   Get-VMResourceConfiguration $vm1 | Set-VMResourceConfiguration –Disk $vm1disk –
   DiskSharesLevel custom –NumDiskShares 100

**Get a List of Hosts on vSphere and View Their Properties**

With PowerCLI, you can get information about all available hosts in a datacenter and view their properties.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. Get a list of all hosts that are part of a datacenter.

   Get-Datacenter DC | Get-VMHost | Format-Custom
2 View the properties of the first host in the datacenter.
   Get-Datacenter DC | Get-VMHost | Select-Object -First 1 | Format-Custom

3 View the Name and the OverallStatus properties of the hosts in the DC datacenter.
   Get-Datacenter DC | Get-VMHost | Get-View | Format-Table -Property Name, OverallStatus -AutoSize

4 View all hosts and their properties, and save the results to a file.
   Get-Datacenter DC | Get-VMHost | Format-Custom | Out-File -FilePath hosts.txt

5 View a list of the hosts that are in maintenance mode and can be configured for vMotion operations.
   Get-VMHost -State maintenance | Get-View | Where-Object -FilterScript { $_.capability -ne $null -and $_.capability.vmotionSupported }
Move a Virtual Machine to a Different Datastore Using VMware Storage vMotion

You can migrate a virtual machine between datastores using the VMware Storage vMotion feature of vSphere.

**Note** You can use VMware Storage vMotion to move a powered-on virtual machine from one datastore to another.

**Prerequisites**

Connect to a vSphere server.

The host on which the virtual machine is running must have access both to the datastore where the virtual machine is located and to the destination datastore.

**Procedure**

- Get the VM1 virtual machine and move it to a datastore named DS2:
  ```powershell
  Get-VM VM1 | Move-VM -Datastore DS2
  ```

Create a Host Profile on vSphere

The VMware Host Profiles feature enables you to create standard configurations for ESX and ESXi hosts. With PowerCLI, you can automate creation and modifying of host profiles.

**Prerequisites**

Connect to a host that runs vCenter Server 4.0 or later.

**Procedure**

1. Get the host named Host1 and store it in the `$host` variable.
   ```powershell
   $host = Get-VMHost Host1
   ```
2. Create a profile based on the Host1 host.
   ```powershell
   New-VMHostProfile -Name MyHostProfile01 -Description "This is my test profile based on Host1." -ReferenceHost $host
   ```
3. Get the newly created host profile.
   ```powershell
   $hp1 = Get-VMHostProfile -Name MyHostProfile01
   ```
4. Change the description of the HostProfile1 host profile.
   ```powershell
   Set-VMHostProfile -Profile $hp1 -Description "This is my old test host profile based on Host1."
   ```

Apply a Host Profile to a Host on vSphere

To simplify operational management of large-scale environments, you can apply standard configurations called host profiles to hosts on vSphere. If you want to set up a host to use the same host profile as a reference host, you can attach the host to a profile.

**Prerequisites**

Connect to a host that runs vCenter Server 4.0 or later.
Procedure
1 Get the Host2 host.
   $host2 = Get-VMHost Host2
2 Attach the Host2 host to the HostProfile1 host profile.
   Set-VMHost -VMHost $host2 -Profile HostProfile1
3 Verify that the Host2 host is compliant with the HostProfile1 profile.
   Test-VMHostProfileCompliance -VMHost $host2
   The output of this command contains the host’s incompliant settings, if any.
4 Apply the profile to the Host2 host.
   $neededVariables = Apply-VMHostProfile -Entity $host2 -Profile $hp1 -Confirm:$false
   The $neededVariables variable contains the names of all required variables and their default or current values, as returned by the server. Otherwise, the $neededVariables variable contains the name of the host on which the profile has been applied.

Manage Statistics and Statistics Intervals on vSphere

You can use the PowerCLI cmdlets to automate tasks for viewing and managing statistics for vSphere inventory objects.

You can modify the properties of a statistics interval and view statistics for a specified cluster.

Prerequisites
Connect to a vSphere server.

Procedure
1 Increase the amount of time for which statistics of the previous day are stored.
   Set-StatInterval -Interval "past day" -StorageTimeSecs 700000
2 View the available memory metric types for the Cluster1 cluster.
   $cluster = Get-Cluster Cluster1
   $statTypes = Get-StatType -Entity $cluster -Interval "past day" -Name mem.*
3 View the cluster statistics collected for the day.
   Get-Stat -Entity $cluster -Start ([System.DateTime]::Now.AddDays(-1)) -Finish ([System.DateTime]::Now) -Stat $statTypes

Modify the Settings of the NIC Teaming Policy for a Virtual Switch

You can specify the NIC teaming policy on a vSwitch. The NIC teaming policy determines the load balancing and failover settings of a virtual switch and allows you to specify unused NICs.

Prerequisites
Connect to a vSphere server.

Procedure
1 Get a list of the physical NIC objects on the host network and store them in a variable.
   $pn = Get-VMHost 10.23.123.128 | Get-VMHostNetwork | select -Property physicalnic
2 Store the physical NIC objects you want to mark as unused in separate variables.
   $pn5 = $pn.PhysicalNic[2]
   $pn6 = $pn.PhysicalNic[3]
   $pn7 = $pn.PhysicalNic[0]

3 View the NIC teaming policy of the VSwitch01 virtual switch.
   $policy = Get-VirtualSwitch -VMHost 10.23.123.128 -Name VSwitch01 | Get-NicTeamingPolicy

4 Change the policy of the switch to indicate that the $pn5, $pn6, and $pn7 network adapters are unused.
   $policy | Set-NicTeamingPolicy -MakeNicUnused $pn5, $pn6, $pn7

5 Modify the load balancing and failover settings of the virtual switch NIC teaming policy.
   $policy | Set-NicTeamingPolicy -BeaconInterval 3 -LoadBalancingPolicy 3 -NetworkFailoverDetectionPolicy 1 -NotifySwitches $false -FailbackEnabled $false

Create a Virtual Appliance on vSphere

With PowerCLI, you can create and manage virtual appliances.

**Prerequisites**
Connect to a vSphere server.

**Procedure**
1 Create a new virtual appliance named VApp on a host.
   New-VApp -Name VApp -CpuLimitMhz 4000 -CpuReservationMhz 1000 -Location ( Get-VMHost Host1 )
2 Start the new virtual appliance.
   Start-VApp VApp

Modify the Properties of a Virtual Appliance

With PowerCLI, you can start and stop virtual appliances, and modify their properties.

**Prerequisites**
Connect to a vSphere server.

**Procedure**
1 Get the virtual appliance named VApp and stop it.
   Get-VApp VApp | Stop-VApp -Confirm:$false
2 Change the name and memory reservation for the VApp virtual appliance.
   Get-VApp VApp | Set-VApp -Name OldVApp -MemReservationMB 2000

Export or Import Virtual Appliances

You can import and export virtual appliances to OVA and OVF files.

**Prerequisites**
Connect to a vSphere server.
Chapter 4 Sample Scripts for Managing vSphere with PowerCLI

Configure a Network Interface

You can modify the IP and routing configuration settings of a guest network interface.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. Get the network interface of a guest operating system.
   
   ```powershell
   $vm1 = Get-VM -Name VM1
   $guest = Get-VMGuest $vm1
   $interface = Get-VMGuestNetworkInterface -VMGuest $guest -HostUser root -HostPassword pass1 -GuestUser user -GuestPassword pass2 -ToolsWaitSecs 100
   ```

2. Configure the network interface.
   
   ```powershell
   Set-VMGuestNetworkInterface -VMGuestNetworkInterface $interface -HostUser root -HostPassword pass1 -GuestUser user -GuestPassword pass2 -IPPolicy static -IP 10.23.112.69 -Gateway 10.23.115.253 -DnsPolicy static -Dns (10.23.108.1, 10.23.108.2) -WinsPolicy dhcp
   ```

Add and Configure a Guest Route

You can add new guest routes for virtual machines and modify the routes properties.

**Prerequisites**

Connect to a host that runs ESX 3.5 or later.

**Procedure**

1. View the existing routes of the virtual machine stored in the `$myVM1` variable.
   
   ```powershell
   Get-VMGuestRoute -VM $vm1 -HostUser root -HostPassword pass1 -GuestUser user -GuestPassword pass2 -ToolsWaitSecs 50
   ```

2. View the existing routes of the guest OS stored in the `$guest` variable.
   
   ```powershell
   Get-VMGuestRoute -VMGuest $guest -HostUser root -HostPassword pass1 -GuestUser user -GuestPassword pass2
   ```

3. Add a new guest route to the virtual machine.
   
   ```powershell
   ```
4 Configure the guest route.

```
$route = Set-VMGuestRoute -VMGuestRoute $route -HostUser root -HostPassword pass1 -GuestUser
user -GuestPassword pass2 -Netmask 255.255.255.254 -Gateway 10.23.112.57
```

Create an iSCSI Host Storage

For a host, you can enable iSCSI, add iSCSI targets, and create new host storages.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. Enable software iSCSI on a host.
   ```
   $host = Get-VMHost ESXHost1
   Get-VMHostStorage $myHost | Set-VMHostStorage -SoftwareIScsiEnabled $true
   ```

2. Get the iSCSI HBA that is on the host.
   ```
   $iscsiHba = Get-VMHostHba -Type iScsi
   ```

3. Add a new iSCSI target for dynamic discovery.
   ```
   $iscsiHba | New-IScsiHbaTarget -Address 192.168.0.1 -Type Send
   ```

4. Rescan the HBAs on the host.
   ```
   Get-VMHostStorage $host -RescanAllHba
   ```

5. Get the path to the SCSI LUN.
   ```
   $lunPath = Get-ScsiLun -VMHost $host -CanonicalName ($iscsiHba.Device + ")") | Get-ScsiLunPath
   ```
   You can specify the LUN path by using its canonical name beginning with the device name of the iSCSI HBA.

6. Create a new host storage.
   ```
   New-Datastore -Vmfs -VMHost $host -Path $lunpath.LunPath -Name iSCSI
   ```

Add Passthrough Devices to a Host and Virtual Machine

You can get information about existing passthrough devices and add new SCSI and PCI devices to virtual machines and hosts.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. Get a list of the PCI passthrough devices of the VMHost host
   ```
   $host = Get-VMHost ESXHost
   Get-PassthroughDevice -VMHost $host -Type Pci
   ```

2. Get a list of the SCSI passthrough devices of the VM virtual machine
   ```
   $vm = Get-VM VM
   Get-PassthroughDevice -VM $vm -Type Scsi
   ```
3 Add a SCSI passthrough device to the VM virtual machine

```powershell
$scsiDeviceList = Get-PassthroughDevice -VMHost ESXHost -Type Scsi
Add-PassthroughDevice -VM $vm -PassthroughDevice $scsiDeviceList[0]
```

### Create a Custom Property Based on an Extension Data Property

You can create custom properties to add more information to vSphere objects. Custom properties based on extension data properties correspond directly to the property of the corresponding .NET view object.

**Prerequisites**

Connect to a vSphere server.

**Procedure**


   ```powershell
   New-VIProperty -ObjectType VirtualMachine -Name ToolsVersion -ValueFromExtensionProperty 'Guest.ToolsVersion'
   ```

2. View the `ToolsVersion` properties of the available virtual machines.

   ```powershell
   Get-VM | Select Name, ToolsVersion
   ```

   You have created a custom property named `ToolsVersion` for `VirtualMachine` objects.

### Create a Script-Based Custom Property for a vSphere Object

You can create a custom property by writing a script and providing a name for the property. The script evaluates when the custom property is called for first time.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. Create a new custom property named `NameOfHost` that stores the name of the host on which a virtual machine resides.

   ```powershell
   New-VIProperty -Name NameOfHost -ObjectType VirtualMachine -Value { return $args[0].VMHost.Name }
   ```

2. View the `NameOfHost` properties of the available virtual machines.

   ```powershell
   Get-VM | Select Name, NameOfHost | Format-Table -AutoSize
   ```

   You created a custom script property named `NameOfHost` for `VirtualMachine` objects.

### Apply a Customization Object to a Cloned Virtual Machine

You can apply a custom configuration to a cloned virtual machine by using a customization object.

**Note** This feature runs only on 32-bit PowerCLI process.

**Prerequisites**

Connect to a vSphere server.
Procedure

1. Get the Spec customization specification and clone it for temporary use.

   Get-OSCustomizationSpec Spec | New-OSCustomizationSpec -Type NonPersistent -Name ClientSpec

2. Change the NamingPrefix property of the customization object to the name of the virtual machine you want to create.

   Set-OSCustomizationSpec -Spec ClientSpec -NamingPrefix VM1

3. Create a virtual machine named VM1 by cloning the existing VM virtual machine and applying the customization specification.

   Get-VM VM | New-VM -VMHost Host -Datastore Storage1 -OSCustomizationSpec ClientSpec -Name VM1

Modify the Default NIC Mapping Object of a Customization Specification

You can modify the default NIC mapping object of a customization specification and apply the specification on a newly created virtual machine.

Procedure

1. Create a nonpersistent customization specification for Windows operating systems.

   New-OSCustomizationSpec -Type NonPersistent -Name Spec -OSType Windows -Workgroup Workgroup -OrgName Company -Fullname User -ProductKey "valid_key" -ChangeSid -TimeZone "Central European" -NamingScheme VM

2. View the default NIC mapping objects of the Spec specification.

   Get-OSCustomizationNicMapping -Spec Spec | Set-OSCustomizationNicMapping -IpMode UseStaticIP -IpAddress 172.16.1.30 -SubnetMask 255.255.255.0 -DefaultGateway 172.16.1.1 -Dns 172.16.1.1

   Each customization specification object has one default NIC mapping object.

3. Modify the default NIC mapping object of the Spec customization specification to use static IP.

   Get-OSCustomizationNicMapping -Spec Spec | Set-OSCustomizationNicMapping -IpMode UseStaticIP -IpAddress 172.16.1.30 -SubnetMask 255.255.255.0 -DefaultGateway 172.16.1.1 -Dns 172.16.1.1

4. Create a new virtual machine named VM1 from a template, and apply the static IP settings.

   New-VM -Name VM1 -VMHost Host -Datastore Storage1 -OSCustomizationSpec Spec -Template Template

Modify Multiple NIC Mapping Objects of a Customization Specification

You can modify multiple NIC mapping objects of a customization specification and apply the specification to an existing virtual machine.

Procedure

1. Get the network adapters of a virtual machine named VM.

   Get-NetworkAdapter VM

   When you apply a customization specification, each network adapter of the customized virtual machine must have a corresponding NIC mapping object. You can correlate network adapters and NIC mapping objects either by their position numbers, or by MAC address.

2. Create a customization specification named Spec.

   New-OSCustomizationSpec -Type NonPersistent -Name Spec -OSType Windows -Workgroup Workgroup -OrgName Company -Fullname User -ProductKey "valid_key" -ChangeSid -TimeZone "Central European" -NamingScheme VM
3 Add a new NIC mapping object that uses a static IP address.

   New-OSCustomizationNicMapping -Spec Spec -IpMode UseStaticIP -IpAddress 172.16.1.30 -SubnetMask 255.255.255.0 -DefaultGateway 172.16.1.1 -Dns 172.16.1.1

4 View the NIC mapping objects and verify that two NIC mapping objects are available.

   Get-OSCustomizationNicMapping -Spec Spec

   The default NIC mapping object is DHCP enabled, and the newly added one uses a static IP address.

5 Apply the Spec customization specification to the VM virtual machine.

   Get-VM VM | Set-VM -OSCustomizationSpec -Spec Spec

6 Associate a network adapter from the VMNetwork network with the NIC mapping object that uses DHCP mode.

   $netAdapter = Get-NetworkAdapter VM | where { $_.NetworkName -eq 'VMNetwork' }

Create a vSphere Role and Assign Permissions to a User

With PowerCLI, you can automate management of vSphere permissions, roles, and privileges.

**NOTE**  vSphere permissions determine your level of access to vCenter Server, ESX, and ESXi hosts. Privileges define individual rights to perform actions and access object properties. Roles are predefined sets of privileges.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1 Get the privileges of the Readonly role.

   $readOnlyPrivileges = Get-VIPrivilege -Role Readonly

2 Create a new role with custom privileges.

   $role1 = New-VIRole -Privilege $readOnlyPrivileges -Name Role1

3 Add the PowerOn privileges to the new role.

   $powerOnPrivileges = Get-VIPrivilege -Name "PowerOn"
   $role1 = Set-VIRole -Role $role1 -AddPrivilege $powerOnPrivileges

4 Create a permission and apply it to a vSphere root object.

   $rootFolder = Get-Folder -NoRecursion
   $permission1 = New-VIPermission -Entity $rootFolder -Principal "user" -Role readonly -Propagate

   The Principal parameter accepts both local and domain users and groups if the vSphere server is joined in AD.

5 Update the new permission with the custom role.

   $permission1 = Set-VIPermission -Permission $permission1 -Role $role1

You created a new role and assigned permissions to a user.
View the Action Triggers for an Alarm on vSphere

You can see which action triggers are configured for an alarm.

Prerequisites

Connect to a vSphere server.

Procedure

1. Get all PowerCLI supported alarm actions for the Host Processor Status alarm.
   ```powershell
   Get-AlarmDefinition -Name "Host Processor Status" | Get-AlarmAction -ActionType "ExecuteScript", "SendSNMP", "SendEmail"
   ```

2. Get all the triggers for the first alarm definition found.
   ```powershell
   Get-AlarmAction -AlarmDefinition (Get-AlarmDefinition | select -First 1) | Get-AlarmActionTrigger
   ```

Create and Modify Alarm Actions and Alarm Triggers on vSphere

With PowerCLI, you can create and modify vSphere alarm actions and alarm triggers.

Prerequisites

Connect to a vSphere server.

Procedure

1. For all host alarms, modify the interval after the action repeats.
   ```powershell
   Get-AlarmDefinition -Entity (Get-VMHost) | foreach { $_ | Set-AlarmDefinition -ActionRepeatMinutes ($_.ActionRepeatMinutes + 1)}
   ```

2. Modify the name and the description of a specified alarm definition, and enable the alarm.
   ```powershell
   Get-AlarmDefinition -Name AlarmDefinition | Set-AlarmDefinition -Name AlarmDefinitionNew -Description 'Alarm Definition Description' -Enabled:$true
   ```

3. Create an alarm action email for the renamed alarm definition.
   ```powershell
   Get-AlarmDefinition -Name AlarmDefinitionNew | New-AlarmAction -Email -To 'test@vmware.com' -CC @('test1@vmware.com', 'test2@vmware.com') -Body 'Email text' -Subject 'Email subject'
   ```

4. Create an snmp alarm action.
   ```powershell
   Get-AlarmDefinition -Name AlarmDefinitionNew | New-AlarmAction -Snmp
   ```

5. Create a script alarm action.
   ```powershell
   Get-AlarmDefinition -Name AlarmDefinitionNew | New-AlarmAction -Script -ScriptPath 'c:\test.ps1'
   ```

6. Create an action trigger on all actions for the specified alarm.
   ```powershell
   Get-AlarmDefinition -Name AlarmDefinitionNew | Get-AlarmAction | New-AlarmActionTrigger -StartStatus 'Red' -EndStatus 'Yellow' -Repeat
   ```
Remove Alarm Actions and Triggers

In some cases, you might want to remove obsolete alarm actions and triggers.

Prerequisites

Connect to a vSphere server.

Procedure

1. Remove the first one from the action triggers found for an alarm definition.

   ```powershell
   Get-AlarmDefinition -Name AlarmDefinition | Get-AlarmAction | Get-AlarmActionTrigger | select -First 1 | Remove-AlarmActionTrigger -Confirm:$false
   ```

2. Remove all the actions for an alarm definition.

   ```powershell
   Get-AlarmDefinition -Name AlarmDefinition | Get-AlarmAction | Remove-AlarmAction -Confirm:$false
   ```

Create and Modify Advanced Settings for a Cluster

You can customize the behavior of a cluster on vSphere by creating and modifying custom advanced settings for it.

Prerequisites

Connect to a vSphere server.

Procedure

1. Create a new cluster named Cluster.

   ```powershell
   $cluster = New-Cluster -Name Cluster -Location (Get-Datacenter Datacenter)
   ```

2. Create two advanced settings for the new cluster.

   ```powershell
   $setting1 = New-AdvancedSetting -Type "ClusterHA" -Entity $cluster -Name 'das.defaultfailoverhost' -Value '192.168.10.1'
   $setting2 = New-AdvancedSetting -Type "ClusterHA" -Entity $cluster -Name 'das.isolationaddress' -Value '192.168.10.2'
   ```

3. Modify the value of the advanced setting stored in the $setting2 variable.

   ```powershell
   Get-AdvancedSetting -Entity $cluster -Name 'das.isolationaddress' | Set-AdvancedSetting -Value '192.168.10.3' -Confirm:$false
   ```

4. Create another advanced setting.

   ```powershell
   New-AdvancedSetting -Entity $cluster -Name 'das.allowNetwork[Service Console]' -Value $true -Type 'ClusterHA'
   ```

5. Get the Service Console setting and store it in a variable.

   ```powershell
   $setting3 = Get-AdvancedSetting -Entity $entity -Name 'das.allowNetwork[Service Console]'
   ```

   The ` character is used to to escape the wildcard characters [ and ] in the advanced setting name.
Modify the vCenter Server Email Configuration

You can modify the email configuration settings of a vCenter Server.

Prerequisites

Connect to a vSphere server.

Procedure

1. View the current email configuration settings of the vCenter Server from the $srv variable.
   
   ```powershell
   Get-AdvancedSetting -Entity $srv  -Name mail.*
   ```

2. Update the SMTP server name and port.
   
   ```powershell
   Get-AdvancedSetting -Entity $srv  -Name mail.smtp.server | Set-AdvancedSetting -Value smtp.vmware.com
   Get-AdvancedSetting -Entity $srv  -Name mail.smtp.port | Set-AdvancedSetting -Value 25
   ```

Modify the vCenter Server SNMP Configuration

To use SNMP, you must first configure the SNMP settings of the vCenter Server.

Prerequisites

Connect to a vSphere server.

Procedure

1. View the current SNMP configuration settings of the vCenter Server from the $srv variable.
   
   ```powershell
   Get-AdvancedSetting -Entity $srv  -Name snmp.*
   ```

2. Modify the SNMP receiver data.
   
   ```powershell
   Get-AdvancedSetting -Entity $srv  -Name snmp.receiver.2.community | Set-AdvancedSetting -Value public
   Get-AdvancedSetting -Entity $srv  -Name snmp.receiver.2.enabled | Set-AdvancedSetting -Value $true
   Get-AdvancedSetting -Entity $srv  -Name snmp.receiver.2.name | Set-AdvancedSetting -Value 192.168.1.10
   ```

   Now you can use SNMP with vCenter Server.

Use Esxtop to Get Information on the Virtual CPUs of a Virtual Machine

You can use the Get-ESXTop cmdlet to retrieve real-time data for troubleshooting performance problems.

Prerequisites

Connect to a server that runs ESX 4.0, vCenter Server 5.0 or later.

Procedure

1. Get the group to which the virtual machine belongs and save it as a variable.
   
   ```powershell
   $group = Get-EsxTop -CounterName SchedGroup | where {$_.VMName -eq $vm.Name}
   ```
2 Get the IDs of all virtual CPUs of the virtual machine and store them in an array.

```powershell
$gr = Get-EsxTop -TopologyInfo -Topology SchedGroup | %{$_.Entries} | where {$_.["GroupId"] -eq $group.GroupID}
$cpuIds = @()
$gr.CpuClient | %{$cpuIds += $_.CPUClientID}
```

3 Get the CPU statistics for the virtual machine.

```powershell
$cpuStats = Get-EsxTop -CounterName VCPU | where {$_.["VCPUID"] -contains $_.VCPUID}
```

4 Calculate the used and ready for use percentage by using the `UsedTimeInUsec` and `ReadyTimeInUsec` stats.

```powershell
$result = @() $cpuStats | %{
    $row = "" | select VCPUID, Used, Ready; 
    $row.VCPUID = $_.VCPUID; 
    $row.Used = [math]::Round(([double]$_.["UsedTimeInUsec"]/[double]$_.["UpTimeInUsec"]*100, 2); 
    $row.Ready = [math]::Round(([double]$_.["ReadyTimeInUsec"]/[double]$_.["UpTimeInUsec"]*100, 2); ` 
    $result += $row
}
```

5 View the used and ready for use percentage for each virtual CPU of the virtual machine.

```powershell
$result | Format-Table -AutoSize
```

**Filter vSphere Objects with Get-View**

You can use the `Get-View` cmdlet to filter vSphere objects before performing various actions on them.

The filter parameter is a `HashTable` object containing one or more pairs of filter criteria. Each of the criteria consists of a property path and a value that represents a regular expression pattern used to match the property.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1 Create a filter by the power state and the guest operating system name of the virtual machines.

   ```powershell
   $filter = @("Runtime.PowerState" ="poweredOn"; "Config.GuestFullName" = "Windows XP")
   ```

2 Get a list of the virtual machines by using the created filter and call the `ShutdownGuest` method for each virtual machine in the list.

   ```powershell
   Get-View -ViewType "VirtualMachine" -Filter $filter | foreach{$_.["ShutdownGuest"]()}
   ```

The filter gets a list of the powered-on virtual machines whose guest OS names contain the string `Windows XP`. The `Get-View` cmdlet then initiates shutdown for each guest operating system in the list.

**Populate a View Object with Get-View**

You can use the `Get-View` cmdlet to update a view object by using the information from a previously called managed object.

**Prerequisites**

Connect to a vSphere server.
**Procedure**

1. Get the VM2 virtual machine by name.
   `$vm2 = Get-View -ViewType VirtualMachine -Filter @{"Name" = "VM2"}`
   `$hostView = Get-View -Id $vm2.Runtime.Host`

2. View the current power state.
   `$vm2.Runtime.PowerState`

3. Power off the virtual machine.
   ```post
   If ($vm2.Runtime.PowerState -ne "PoweredOn") {
     $vm.PowerOnVM($vm2.Runtime.Host)
   } else {
     $vm2.PowerOffVM()
   }
   ```

4. View the value of `$vm2` power state.
   `$vm2.Runtime.PowerState`

   The power state is still not updated because the virtual machine property values are not updated automatically.

5. Update the view object.
   `$vm2.UpdateViewData()`

6. Obtain the actual power state of the virtual machine.
   `$vm2.Runtime.PowerState`

---

**Update the State of a Server-Side Object**

You can use the `Get-View` cmdlet to update server-side objects.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. Get the VM2 virtual machine by name.
   `$vm2 = Get-View -ViewType VirtualMachine -Filter @{"Name" = "VM2"}`
   `$hostView = Get-View -Id $vm2.Runtime.Host`

2. View the current power state.
   `$vm2.Runtime.PowerState`

3. Power off the virtual machine.
   ```post
   If ($vm2.Runtime.PowerState -ne "PoweredOn") {
     $vm.PowerOnVM($vm2.Runtime.Host)
   } else {
     $vm2.PowerOffVM()
   }
   ```

4. View the value of the `$vm2` power state.
   `$vm2.Runtime.PowerState`

   The power state is not updated yet because the virtual machine property values are not updated automatically.
5 Update the view object.
   $vm2.UpdateViewData()

6 Obtain the actual power state of the virtual machine.
   $vm2.Runtime.PowerState

Reboot a Host with Get-View

You can reboot a host by using its corresponding view object.

Prerequisites
Connect to a vSphere server.

Procedure
1 Use the Get-VMHost cmdlet to get a host by its name, and pass the result to the Get-View cmdlet to get the corresponding view object.
   $hostView = Get-VMHost -Name Host | Get-View
2 Call the reboot method of the host view object to reboot the host.
   $hostView.RebootHost()

Modify the CPU Levels of a Virtual Machine with Get–View and Get–VIObjectByVIView

You can modify the CPU levels of a virtual machine using a combination of the Get–View and Get–VIObjectByVIView cmdlets.

Prerequisites
Connect to a vSphere server.

Procedure
1 Get the VM2 virtual machine, shut it down, and pass it to the Get-View cmdlet to view the virtual machine view object.
   $vmView = Get-VM VM2 | Stop-VM | Get-View
2 Create a VirtualMachineConfigSpec object to modify the virtual machine CPU levels and call the ReconfigVM method of the virtual machine view managed object.
   $spec = New-Object VMware.Vim.VirtualMachineConfigSpec;
   $spec.CpuAllocation.Shares.Level = "normal";
   $spec.CpuAllocation.Limit = -1;
   $vmView .ReconfigVM_Task($spec)
3 Get the virtual machine object by using the Get-VIObjectByVIView cmdlet and start the virtual machine.
   $vm = Get-VIObjectByVIView $vmView | Start-VM
Browse the Default Inventory Drive

You can browse the default inventory drive and view its contents.

**Note** For more information about the Inventory Provider and the default inventory drive, see “PowerCLI Inventory Provider,” on page 12.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. Navigate to the vi inventory drive.
   
   ```
   cd vi:
   ```

2. View the drive content.
   
   ```
   dir
   ```

   `dir` is an alias of the `Get-ChildItem` cmdlet.

Create a New Custom Inventory Drive

In addition to the default drive, you can create new custom inventory drives by using the `New-PSDrive` cmdlet.

**Note** An alternative to creating an inventory drive is to map an existing inventory path. For example, run:

```powershell
New-PSDrive -Name myVi -PSProvider VimInventory -Root "vi:\Folder01\Datacenter01"
```

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. Get the root folder of the server.
   
   ```powershell
   $root = Get-Folder -NoRecursion
   ```

2. Create a PowerShell drive named myVi in the server root folder.
   
   ```powershell
   New-PSDrive -Location $root -Name myVi -PSProvider VimInventory -Root '\'
   ```

   **Note** You can use the `New-InventoryDrive` cmdlet, which is an alias of `New-PSDrive`. This cmdlet creates a new inventory drive using the `Name` and `Datastore` parameters. For example:

   ```powershell
   Get-Folder -NoRecursion | New-VIInventoryDrive -Name myVi
   ```

Manage Inventory Objects Through Inventory Drives

You can use the PowerCLI Inventory Provider to browse, modify, and remove inventory objects from inventory drives.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. Navigate to a host in your server inventory by running the `cd` command with the full path to the host.
   
   ```powershell
   cd Folder01\DataCenter01\host\Web\Host01
   ```
2 View the content of the host using the `ls` command.
   
   `ls`

   `ls` is the UNIX style alias of the `Get-ChildItem` cmdlet.
   
   This command returns the virtual machines and the root resource pool of the host.

3 View only the virtual machines on the host.
   
   `Get-VM`

   When called within the inventory drive, `Get-VM` gets a list only of the virtual machines on the current drive location.

4 Delete a virtual machine named VM1.
   
   `del VM1`

5 Rename a virtual machine, for example, from VM1New to VM1.
   
   `ren VM1New VM1`

6 Start all virtual machines with names that start with VM.
   
   `dir VM* | Start-VM`

### Browse the Default Datastore Drives

You can use the PowerCLI Datastore Provider to browse the default datastore drives: `vmstore` and `vmstores`.

**Note** For more information about default datastore drives, see “PowerCLI Datastore Provider,” on page 13.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1 Navigate to the `vmstore` drive.
   
   `cd vmstore:`

2 View the drive content.
   
   `dir`

### Create a New Custom Datastore Drive

You can use the PowerCLI datastore provider to create custom datastore drives.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1 Get a datastore by its name and assign it to the `$datastore` variable.
   
   `$datastore = Get-Datastore Storage1`
2 Create a new PowerShell drive ds: in $datastore.

    New-PSDrive -Location $datastore -Name ds -PSProvider VimDatastore -Root '\'

**NOTE** You can use the New-PSDrive cmdlet, which is an alias of New-DatastoreDrive. It creates a new datastore drive using the Name and Datastore parameters. For example: Get-Datastore Storage1 |New-DatastoreDrive -Name ds.

## Manage Datastores Through Datastore Drives

You can use the PowerCLI Datastore Provider to browse datastores from datastore drives.

**Prerequisites**

Connect to a vSphere server.

**Procedure**

1. Navigate to a folder on the ds: drive.
   ```
   cd VirtualMachines\XPVirtualMachine
   ```

2. View the files of the folder by running the ls command.
   ```
   ls
   ```
   *ls* is the UNIX style alias of the Get-ChildItem cmdlet.

3. Rename a file by running the Rename-Item cmdlet or its alias ren.
   ```
   ren vmware-3.log vmware-3old.log
   ```
   *All file operations apply only on files in the current folder.*

4. Delete a file by running the Remove-Item cmdlet or its alias del.
   ```
   del ds:\VirtualMachines\XPVirtualMachine\vmware-2.log
   ```

5. Copy a file by running the Copy-Item cmdlet or its alias copy.
   ```
   copy ds:\VirtualMachines\XPVirtualMachine\vmware-3old.log ds:\VirtualMachines\vmware-3.log
   ```

6. Copy a file to another datastore by running the Copy-Item cmdlet or its alias copy.
   ```
   copy ds:\Datacenter01\Datastore01\XPVirtualMachine\vmware-1.log ds:\Datacenter01\Datastore02\XPVirtualMachine02\vmware.log
   ```

7. Create a new folder by running the New-Item cmdlet or its alias mkdir.
   ```
   mkdir -Path ds:\VirtualMachines -Name Folder01 -Type Folder
   ```

8. Download a file from the datastore drive to the local machine by running the Copy-DatastoreItem cmdlet.
   ```
   Copy-DatastoreItem ds:\VirtualMachines\XPVirtualMachine\vmware-3.log C:\Temp\vmware-3.log
   ```

9. Upload a file from the local machine by running the Copy-DatastoreItem cmdlet.
   ```
   Copy-DatastoreItem C:\Temp\vmware-3.log ds:\VirtualMachines\XPVirtualMachine\vmware-3new.log
   ```
Symbols

.NET, environment 20

A
advanced settings
cluster 41
host 31
vCenter Server email configuration 42
vCenter Server SNMP configuration 42
alarms
actions 40
alarm actions remove remove 41
alarm triggers remove 41
alarms actions 40
alarms triggers 40
triggers 40
API access cmdlets
CPU levels modify 45
filter objects 43
asynchronously running cmdlets 11

C
cluster, advanced settings 41
common parameters 8
connect, vSphere server 26
create 28
custom properties
create 37
custom properties based on extension data 37
script custom properties 37
custom scripts, extend the OS support 11
customization specification
apply 37
default NIC mapping 38
modify 38
multiple NIC mappings 38
nonpersistent 12
persistent 12

D
datastore drives
create 47
manage datastores 48
datastore drives browse 47
datastore provider
browse datastore drives 47
create datastore drives 47
manage datastores 48

E
ESXCLI 12
esxtop 42

G
Get-View
filter objects 43
populate objects 43
reboot host 45
server-side objects update 44
guest network interface, configure 35
guest route
add 35
configure 35

H
host
adding to a server 27
maintenance mode 27
host profiles
apply 32
attach 32
create 32
modify 32
test 32
host storage, iSCSI 36
hosts
advanced settings 31
properties 30

I
install
allow running scripts 17
complete installation 16
custom PowerCLI installation 17
prerequisites 16
set remote signing 17
supported operating systems 15
supported VMware environments 16
installation prerequisites 16
introduction
PowerCLI specifics 7
powershell 7
inventory drive browse 46
inventory drives
  create 46
  manage inventory 46
inventory objects, create 28
inventory provider
  browse inventory drive 46
  create inventory drives 46
  default inventory drive 46
  manage inventory 46
iSCSI HBA 36
iSCSI target 36

M
  maintenance mode, activate 27
  manage datastores 48
  manage inventory 46

N
  network, guest network interface 35
  NIC teaming policy 33

O
  OBN, OBN failure 10
  OS support extend 11
  OVA 34
  OVF 34

P
  passthrough devices
    add 36
    PCI 36
    SCSI 36
    view 36
  PCI 36
  permissions 39
  PowerCLI configuration file 10
  PowerCLI snapins 9
  PowerCLI specifics
    configuration file 10
    custom scripts 11
    customization specification 12
    datastore provider 13
    default vsphere server connections 11
    inventory provider 12
    OBN 10
    OBN failure 10
    running powercli cmdlets asynchronously 11
  specifying objects 10
  starting PowerCLI 10
  using ESXCLI 12
  views cmdlets 12
  powershell
    cmdlet syntax 7
  common parameters 8
    pipeline 8
    wildcards 8
  PowerShell 7

R
  remote signing 17
  roles
    create 39
    privileges 39

S
  SCSI 36
  server connection, default vsphere server connections 11
  server-side objects 44
  snapshots
    create 29
    use 29
  statistics, statistics intervals 33
  Storage vMotion 32
  supported operating systems 15
  supported VMware environments 16

T
  templates, manage 29

U
  uninstall 18

V
  vCenter Server email configuration 42
  vCenter Server SNMP configuration 42
  view objects, populate 43
  views
    characteristics 19
    error handling 22
    filters 21
    server sessions 22
    update 20
  views cmdlets 12
  virtual appliance
    create 34
    start 34
  virtual appliances
    export 34
    import 34
    properties 34
    stop 34
  virtual machines
    create 28
    migrate between datastores 32
    migrate between hosts 31
    move 26
power off 26
resource configuration 30
start 26
Storage vMotion 32
vMotion 31
xml specification 28
virtual switch
  NIC teaming policy 33
  settings 33
vMotion 31
vSphere, examples 23
vSphere server
  connect 26
  default connections 11

W
wildcards 8

X
xml specification 28