vSphere PowerCLI Administration Guide

VMware vSphere PowerCLI 4.0

EN-000153-00

vmware®

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

VMware, Inc.
3401 Hillview Ave.
Palo Alto, CA 94304
www.vmware.com

Contents

About This Book

The *vSphere PowerCLI Administration Guide* provides information about using the VMware vSphere PowerCLI cmdlets (pronounced "commandlets") set that ships with vSphere PowerCLI for managing, monitoring, automating, and handling life-cycle operations for VMware vSphere components—virtual machines, datacenters, storage, networks, and so on.

Intended Audience

This book is intended for anyone who needs to use vSphere PowerCLI. The information in this book is written for administrators who are familiar with virtual machine technology and Windows PowerShell. There are two categories of users for vSphere PowerCLI:

- **Basic** administrators can use PowerShell commands included in vSphere PowerCLI to manage their VMware infrastructure from the command line.
- **Advanced** administrators can develop PowerShell scripts that may be reused by other administrators or integrated into other applications.

NOTE All vSphere PowerCLI users are expected to be familiar with the details of VMware vSphere administration and the Windows operating system. Solution developers are expected to be familiar with the .NET infrastructure and the VIM object model as well.

Document Feedback

VMware welcomes your suggestions for improving our documentation. If you have comments, send your feedback to docfeedback@vmware.com.

Technical Support and Education Resources

The following sections describe the technical support resources available to you. To access the current version of this book and other books, go to http://www.vmware.com/support/pubs.

Online and Telephone Support

To use online support to submit technical support requests, view your product and contract information, and register your products, go to http://www.vmware.com/support.

Customers with appropriate support contracts should use telephone support for the fastest response on priority 1 issues. Go to http://www.vmware.com/support/phone_support.html.

Support Offerings

To find out how VMware support offerings can help meet your business needs, go to http://www.vmware.com/support/services.

VMware Professional Services

VMware Education Services courses offer extensive hands-on labs, case study examples, and course materials designed to be used as on-the-job reference tools. Courses are available onsite, in the classroom, and live online. For onsite pilot programs and implementation best practices, VMware Consulting Services provides offerings to help you assess, plan, build, and manage your virtual environment. To access information about education classes, certification programs, and consulting services, go to http://www.vmware.com/services.

Getting Started with vSphere PowerCLI 1

vSphere PowerCLI provides easy-to-use C# and PowerShell interface to VMware vSphere APIs. It ships with a number of cmdlets that you can use to perform various administration tasks on VMware vSphere components. This chapter explains how to get started using the vSphere PowerCLI cmdlets.

This chapter covers the following topics:

- "Introduction to the vSphere PowerCLI Cmdlets" on page 7
- "Launching vSphere PowerCLI" on page 8

Introduction to the vSphere PowerCLI Cmdlets

Microsoft PowerShell is both a command-line and scripting environment, designed for Windows. It leverages the .NET object model and provides administrators with management and automation capabilities. Working with PowerShell, like with any other console environment, is done by typing commands. In PowerShell commands are called cmdlets, which term we will use throughout this guide.

vSphere PowerCLI 4.0 ships with 165 cmdlets:

- 163 PowerShell-based cmdlets.
- 2 .NET cmdlets for use through PowerShell—the so called Web Service Access Cmdlets. See "Web Service Access Cmdlets" on page 21.

vSphere PowerCLI cmdlets are created to answer the specific needs of the VMware vSphere administration and management. The vSphere PowerCLI cmdlets are found in the `VMware.VimAutomation.Core` snapin.

Command-Line Syntax

vSphere PowerCLI command-line syntax is the same as generic PowerShell 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 looks like the following:

```
command –parameter1 –parameter2 argument1 –argument2
```

Launching vSphere PowerCLI

To launch vSphere PowerCLI from the **Start** menu, click **Programs** > **VMware** > **VMware vSphere PowerCLI**> **VMware vSphere PowerCLI**.

The script configuration file `Initialize-VIToolkitEnvironment.ps1` is loaded automatically. This file is located in the `Scripts` folder in the vSphere PowerCLI installation directory. Administrators can edit and extend the script to define cmdlets aliases, configure the environment, or set vSphere PowerCLI start up actions.

NOTE Instead of launching the vSphere PowerCLI console, administrators can also access the vSphere PowerCLI snapin directly from other tools, like PowerShell Plus or PowerGUI, by running:

```
Add-PSSnapin VMware.VimAutomation.Core
```

In this case, the `Initialize-VIToolkitEnvironment.ps1` script configuration file is not started automatically. To load it, type its name in the console window without specifying the path:

```
Initialize-VIToolkitEnvironment.ps1
```

Loading the file provides access to vSphere PowerCLI cmdlets aliases, like `Get-VC`, `Get-ESX`, and to other configuration settings.

List All vSphere PowerCLI Cmdlets

If you are new to vSphere PowerCLI, one thing you want to know is what cmdlets are available to you. To get a list of all vSphere PowerCLI cmdlets, use the `Get-Command` cmdlet with the `-PSSnapin` parameter in the following way:

```
Get-Command -PSSnapin VMware.VimAutomation.Core
```

The vSphere PowerCLI cmdlets are listed in the console window as one long, scrolling topic. You can view them a single page at a time by piping the results of the `Get-Command` cmdlet to the `more` option in the following way:

```
Get-Command -PSSnapin VMware.VimAutomation.Core | more
```

Displaying Help for Any Cmdlet

You can get help for a specific cmdlet by supplying the `Get-Help` command in the vSphere PowerCLI Console. For example, for information on the `Add-VMHost` cmdlet, run the `Get-Help` command as follows:

```
Get-Help Add-VMHost
```

For more detailed information, add the `-full` parameter:

```
Get-Help Add-VMHost -full
```

Alternatively, you can use the `help` alias with any cmdlet:

```
help Add-VMHost
```

To view detailed help information page by page, pipe the `help` cmdlet to the `more` cmdlet:

```
help Add-VMHost -full | more
```

Connecting to a Server

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

In the vSphere PowerCLI console window, type the following cmdlet:

```
Connect-VIServer -Server <Server_Address>
```

where `<Server_Address>` is the IP address or DNS name of the vCenter Server or ESX host. When prompted, enter your user name and password to authenticate with the server.

Another way is to put all information on the command line at once, using the `Protocol`, `User`, and `Password` parameters. For example:

```
Connect-VIServer -Server 192.168.10.10 -Protocol http -User admin -Password sck9p84
```

After a connection is established, you are ready to run the vSphere PowerCLI cmdlets.

For example, you can start a specific virtual machine using the following cmdlet:

```
Get-VM -name <virtual_machine_name> | Start-VM
```

For example, to run a virtual machine named `MyVM`, run:

```
Get-VM -name MyVM | Start-VM
```

Basic Cmdlet Usage 2

This chapter explores the basics of the vSphere PowerCLI cmdlets usage.

The chapter discusses the following topics:

NOTE This chapter does not discuss PowerShell basics. You are expected to have knowledge of PowerShell and its command-line and scripting conventions.

PowerShell Cmdlet Usage

In this section, some of the cmdlets syntax and usage basic concepts are described.

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 along to the next command in the pipeline. Objects are output from the pipeline as soon as they become available. You can type a pipeline on a single line, or spread it across multiple lines. You can cycle backwards through command history using the up arrow, so it is easier to repeat pipelines if you type them on a single line.

Wildcards

PowerShell has a number of pattern matching operators called wildcards, which work on strings. For example, to display all files with a `.txt` extension, run:

```
dir *.txt
```

In this example, the asterisk * operator matches any combination of characters.

Wildcard patterns allow you to specify character ranges as well. For example, to display 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 display all `.txt` files whose names consist of 'string' and one more character in the end, run:

```
dir string?.txt
```

All wildcard expressions can be used with the vSphere PowerCLI cmdlets.

Common Parameters

The Windows PowerShell engine implements a set of reserved parameter names, referred to as common parameters. All PowerShell cmdlets, including the vSphere PowerCLI cmdlets, support them. Common parameters are: `Verbose`, `Debug`, `ErrorAction`, `ErrorVariable`, `OutVariable`, and `OutBuffer`. Respectively, the following aliases are reserved for these parameters: `vb`, `db`, `ea`, `ev`, `ov`, and `ob`.

In addition, there are two risk mitigation parameters in PowerShell: `WhatIf` and `Confirm`. `Whatif` is used when you want to see the effects of a command without executing it. `Confirm` is used when a cmdlet performs an operation that stops a program or service or deletes data.

For more details on the usage of common parameters, use the following command:

```
Get-Help about_CommonParameters
```

vSphere PowerCLI Specific Cmdlet Usage

This section explores some specific concepts of the vSphere PowerCLI cmdlets.

The Server Parameter

By default, vSphere PowerCLI cmdlets are executed using the server connection stored in the `$DefaultVIServer` variable. This variable contains the latest connection established to an ESX host or vCenter Server. To apply vSphere PowerCLI cmdlets on other servers, administrators can use the `Server` parameter., which is common for most of the vSphere PowerCLI cmdlets.

Specifying Objects

In vSphere PowerCLI, all parameters that take as arguments inventory objects (`Cluster`, `Datacenter`, `Folder`, `ResourcePool`, `Template`, `VirtualMachine`, `VMHost`), datastores, or `OSCustomizationSpec` objects can be specified by strings and wildcards. This is called Object-by-Name selection (OBN). If a provided object name is not recognized, an OBN failure occurs. In such cases, a non-terminating error is generated and the cmdlet is executed ignoring the invalid name.

Example 2-1. 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, a non-terminating error is thrown and the command is applied only on the `VM1` and `VM2` virtual machines.

For more details on OBN, use the following command:

```
help about_OBN
```

Instead of assigning an object name to a cmdlet parameter, users can pass the object through a pipeline or a variable.

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

For example, the following three lines 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")
```

Examples of Basic Usage of the vSphere PowerCLI Cmdlets

This section provides some examples of basic vSphere PowerCLI cmdlets usage.

Connecting to a Server

The following cmdlet establishes a connection to a local server and asks for credentials (user name and password), as they are not passed as parameters:

```
Connect-VIServer -Server <VI_server_address>
```

For example:

```
Connect-VIServer -Server esx3.example.com
```

NOTE If a proxy server is used for the connection, the administrator should verify that it is configured properly, so that the connection is kept alive long enough not to break the long running vSphere PowerCLI tasks. To remove a proxy, run the following command:

```
Set-VIToolkitConfiguration -ProxyPolicy NoProxy
```

Basic Virtual Machines Operations

The following scenario shows how to retrieve information of available virtual machines and their operation system. It also demonstrates how to shut down a virtual machine guest OS and to power off the virtual machine using vSphere PowerCLI cmdlets.

To manage virtual machines

1 After establishing a connection to a server, list all virtual machines on the target system:

```
Get-VM
```

2 Save the name and the power state properties of the virtual machines in the `myPool` resource pool into a file named `myVMProperties.txt`:

```
$pool = Get-ResourcePool myPool

Get-VM -Location $pool | Select-Object Name, PowerState > myVMProperties.txt
```

3 Start the `MyVM` virtual machine:

```
Get-VM "myVM" | Start-VM
```

4 Retrieve information of the guest OS of the `myVM` virtual machine:

```
Get-VMGuest myVM | fc
```

5 Shutdown the OS of the `myVM` virtual machine:

```
Shutdown-VMGuest "myVM"
```

6 Power off the `myVM` virtual machine:

```
Stop-VM "myVM"
```

7 Move the virtual machine `myVM` from the `ABC` host to the `XYZ` host:

```
Get-VM -Name “myVM” -Location (Get-VMHost ‘ABC’) | Move-VM -Destination (Get-VMHost ‘XYZ’)
```

NOTE If the virtual machine to be moved 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.

Basic Virtual Machine Hosts Operations

The following example scenario illustrates some basic operations with virtual machine hosts, like adding a host to a vCenter Server, putting a host into maintenance mode, shutting down, and removing a host from the vCenter Server.

To manage virtual machine hosts

1 List all hosts on the target VMware vSphere server you have established a connection with:

```
Get-VMHost
```

2 Add a standalone virtual machine host to the vCenter Server:

```
Add-VMHost -Name myHost -Location (Get-Datacenter Main) -User root -Password pass
```

3 Activate maintenance mode for the `myHost` virtual machine host:

```
Get-VMHost myHost | Set-VMHost -State maintenance
```

4 Remove the `myHost` virtual machine host from maintenance mode:

```
Get-VMHost myHost | Set-VMhost -State connected
```

5 Shut down the `myHost` virtual machine host

```
$host = Get-VMHost myHost

$hv = Get-View $h.ID

$hv.ShutdownHost_Task($true)
```

6 Remove the `myHost` virtual machine host without confirmation:

```
Remove-VMHost $host -Confirm:$false
```

Advanced Cmdlet Usage 3

This chapter provides examples of advanced usage of the vSphere PowerCLI cmdlets.

The chapter discusses these topics:

Examples of Advanced Cmdlet Usage

This section contains examples of using the vSphere PowerCLI cmdlets, the Web Service Access cmdlets, and the inventory provider functionality for retrieving and managing VMware vSphere objects.

Using the vSphere PowerCLI Cmdlets

The following examples illustrate how to use advanced functionality provided by the vSphere PowerCLI cmdlets:

Create vSphere Objects

The following scenario illustrates common methods for creating folders, datacenters, clusters, resource pools and virtual machines using vSphere PowerCLI cmdlets.

To create inventory objects

1 Establish a connection to a vCenter server, use the `Connect-VIServer` command:

```
Connect-VIServer -Server <Server_Address>
```

For example:

```
Connect-VIServer -Server 192.168.10.10
```

When prompted, provide the administrator's user name and password to authenticate access on the server.

2 Get the inventory root folder and create a new folder called `USA` in it:

```
$usaFolder = Get-Folder -NoRecursion | New-Folder -Name USA
```

Note that the information about the location of the new folder is specified through the pipeline.

3 Create a new datacenter called `OrlandoDC` in the `USA` folder:

```
New-Datacenter -Location $usaFolder -Name OrlandoDC
```

4 Create a folder called `Engineering` under `OrlandoDC`:

```
Get-Datacenter OrlandoDC | New-Folder -Name Engineering

$engineeringFolder = Get-Folder -Name Engineering
```

NOTE Search in PowerShell is not case-sensitive.

5 Create a new cluster `ResearchAndDevelopmentCluster` in the `Engineering` folder:

```
New-Cluster -Location $engineeringFolder -Name ResearchAndDevelopmentCluster -DrsEnabled
-DrsAutomationLevel FullyAutomated
```

NOTE Drs (distributed resource scheduler) is a facility that allows automatic allocation of cluster resources.

6 Add a host in the cluster using the `Add-VMHost` command, which prompts you for credentials:

```
$hostInCluster1 = Add-VMHost -Name <host IP address> -Location ( Get-Cluster
ResearchAndDevelopmentCluster )
```

The parentheses interpolate the object returned by the `Get-Cluster` command into `Location` parameter.

7 Create a resource pool in the cluster's root resource pool:

```
$clusterRootRP = Get-ResourcePool -Location ( Get-Cluster ResearchAndDevelopmentCluster )
-Name Resources

New-ResourcePool -Location $clusterRootRP -Name DevelopmentResources
-CpuExpandableReservation $true -CpuReservationMhz 500 -CpuSharesLevel high
-MemExpandableReservation $true -MemReservationMB 500 -MemSharesLevel high
```

8 Create a virtual machine synchronously:

```
New-VM -Name DevelopmentVM1 -VMHost $hostInCluster1 -ResourcePool ( Get-ResourcePool
DevelopmentResources ) -DiskMB 4000 -MemoryMB 256
```

9 Create a virtual machine asynchronously:

```
$vmCreationTask = New-VM -Name DevelopmentVM2 -VMHost $hostInCluster1 -ResourcePool
(Get-ResourcePool DevelopmentResources) -DiskMB 4000 -MemoryMB 256 -RunAsync
```

The `-RunAsync` parameter specifies that the command will be executed 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 in the command line.

Use Virtual Machine Templates

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.

This procedure illustrates how to create virtual machines templates and convert them to virtual machines. It uses the VMware vSphere objects created in the previous example.

To create and use virtual machine templates

1 Add an additional 2GB hard disk to the `DevelopmentVM2` virtual machine:

```
$vmCreationTask | Wait-Task | New-HardDisk -CapacityKB ( 2 * 1024 * 1024 )
```

2 Create a template from the `DevelopmentVM1` virtual machine:

```
New-Template -VM “DevelopmentVM1” -Name WebTemplate -Location (Get-Datacenter orlandoDC )
```

NOTE Note that on VirtualCenter 2.0 and VirtualCenter 2.5, the virtual machine must be powered off before creating a template based on it. On VirtualCenter 2.5 Update 2, the virtual machine can be powered off or powered on, but not suspended.

3 Convert this template for a use by a virtual machine named `DevelopmentVM3`:

```
Get-Template WebTemplate | Set-Template -ToVM -Name DevelopmentVM3
```

4 Create a template from the `DevelopmentVM2` virtual machine:

```
New-Template -VM “DevelopmentVM2” -Name DatabaseTemplate -Location (Get-Datacenter orlandoDC )
```

5 Convert this template to a virtual machine named `DevelopmentVM4`:

```
Get-Template DatabaseTemplate | Set-Template -ToVM -Name DevelopmentVM4
```

6 Move the virtual machines into the DevelopmentResources resource pool:

```
Get-VM DevelopmentVM? | Move-VM -Destination ( Get-ResourcePool DevelopmentResources )
```

Use ? as a wildcard to match just one symbol. The command returns all virtual machines whose names start with DevelopmentVM and have one more symbol at the end. In this example, DevelopmentVM1, DevelopmentVM2, DevelopmentVM3, and DevelopmentVM4 are retrieved and moved into the DevelopmentResources resource pool.

7 Start the virtual machines in the DevelopmentResources resource pool:

```
Get-ResourcePool DevelopmentResources | Get-VM | Start-VM
```

Create Virtual Machines Using an XML Specification File

This example illustrates how to create virtual machines in accordance with the specification provided in an XML file.

Consider a myVM.xml file, with the following content:

```
<CreateVM>
    <VM>
        <Name>DevelopmentVM1</Name>
        <HDDCapacity>10000</HDDCapacity>
    </VM>
    <VM>
        <Name>DevelopmentVM2</Name>
        <HDDCapacity>10000</HDDCapacity>
    </VM>
</CreateVM>
```

1 To read the content of the myVM.xml file, run:

```
[xml]$s = Get-Content myVM.xml
```

2 To create the virtual machines, run:

```
$s.CreateVM.VM | where { New-VM -VMHost 192.168.10.11 -Name $_.Name -Disk MB $_.HDDCapacity}
```

Create Snapshots

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.

The following procedure illustrates taking a snapshot of virtual machines and then reverting the virtual machines to it.

To create and use snapshots

1 Take a snapshot of all virtual machines in the DevelopmentResources resource pool:

```
Get-ResourcePool DevelopmentResources | Get-VM | New-Snapshot -Name InitialDevConfig
```

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 DevelopmentResources resource pool to the InitialDevConfig snapshot:

```
$VMs = Get-VM -Location ( Get-ResourcePool DevelopmentResources )
foreach( $vm in $VMs ) { Set-VM -VM $vm -Snapshot ( Get-Snapshot -VM $vm -Name
InitialDevConfig ) }
```

Remove vSphere Objects

This procedure illustrates how to remove some of the VMware vSphere objects created in the previous examples.

To remove inventory objects

1 Retrieve the virtual machines using the '?' wildcard symbol in the `Name` parameter and remove them along with their disks:

```
Get-VM DevelopmentVM? | Remove-VM -DeleteFromDisk
```

2 Remove the `USA` folder:

```
Remove-Folder -Folder ( Get-Folder usa )
```

By executing the `Remove-Folder` command, all child objects for the folder—datacenters, folders, clusters, hosts, resource pools, and so on—are also removed.

Update the Resource Configuration Settings of a Virtual Machine

The following procedure illustrates how to retrieve and modify the resource configuration properties of a virtual machine.

To update the resource configuration of a virtual machine

1 Establish a connection to an ESXi host by the `Connect-VIServer` command:

```
Connect-VIServer -Server <Server_Address>
```

2 Retrieve the resource configuration for the `VM1` virtual machine.:

```
Get-VMResourceConfiguration -VM ( Get-VM VM1 )
```

3 Display the disk share of the `VM1` virtual machine:

```
Get-VMResourceConfiguration -VM ( Get-VM VM1 ) | Format-Custom -Property 
DiskResourceConfiguration
```

4 Change the memory share of the `VM1` virtual machine to low:

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

5 Update the CPU share of the `VM1` virtual machine to high:

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

6 Change the disk share of the `VM1` virtual machine to 100:

```
$vm = Get-VM VM1

$disk = Get-HardDisk $vm

Get-VMResourceConfiguration $vm | Set-VMResourceConfiguration -Disk $disk -DiskSharesLevel 
custom -NumDiskShares 100
```

List Various Virtual Machine Hosts and Displaying Their Properties

This scenario illustrates how to list all available virtual machine hosts in a datacenter and display their properties.

To list the available hosts and display their properties

1 List all virtual machine hosts that are part of the datacenter named `CaliforniaDC`:

```
Get-Datacenter CaliforniaDC | Get-VMHost | Format-Custom
```

2 Display the properties of the first virtual machine host in the datacenter:

```
Get-Datacenter CaliforniaDC | Get-VMHost | Select-Object -First 1 | Get-View | Format-Custom
```

3 Displays the `Name` and the `OverallStatus` properties of the virtual machine hosts in the `CaliforniaDC` datacenter:

```
Get-Datacenter CaliforniaDC | Get-VMHost | Get-View | Format-Table -Property 
Name,OverallStatus -AutoSize
```

4 Display all virtual machine hosts and their properties, and save the results to a file:

```
Get-Datacenter “CaliforniaDC” | Get-VMHost | Get-View | Format-Custom | Out-File -FilePath
hosts.txt
```

5 List the virtual machine 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 }
```

Change the Host Advanced Configuration Settings

This procedure shows how to migrate virtual machines from one virtual machine host to another.

To change the host advanced configuration settings

1 Change the migration time-out for the `MyESXHost1` virtual machine host:.

```
Get-VMHost MyESXHost1 | Set-VmHostAdvancedConfiguration -Name Migrate.NetTimeout -Value (
[system.int32] 10 )
```

2 Enable making checksum of the virtual machines memory during the migration:

```
Get-VMHost MyESXHost | Set-VmHostAdvancedConfiguration -Name Migrate.MemChksum -Value (
[system.int32] 1 )
```

3 Get the `MyESXHost1` virtual machine host migration settings:

```
$migrationSettings = Get-VMHost MyESXHost1 | Get-VmHostAdvancedConfiguration -Name Migrate.*
```

4 Apply the migration settings to `MyESXHost2`:

```
Set-VmHostAdvancedConfiguration -VMHost ( Get-VMHost MyESXHost2 ) -Hashtable
$migrationSettings
```

Migrate a Virtual Machine

The following procedures illustrate how to migrate a virtual machine between hosts and datastores using the VMotion and Storage VMotion features.

To move a virtual machine using VMotion

1 Establish a connection to a server using the `Connect-VIServer` command:

```
Connect-VIServer -Server <Server_Address>
```

2 Retrieve the `VM1` virtual machine and move it to the host named `EsxHost2`:

```
Get-VM VM1 | Move-VM -Destination ( Get-VMHost EsxHost2 )
```

NOTE VMotion allows to move a virtual machine that is powered on from one host to another. The virtual machine must be stored on a datastore shared by the current and destination hosts, and the VMotion interfaces on the two hosts must be properly configured.

To move a virtual machine using Storage VMotion

1 Establish a connection to a server using the `Connect-VIServer` command:

```
Connect-VIServer -Server <Server_Address>
```

2 Retrieve the `VM1` virtual machine and move it to the datastore named `Datastore2`:

```
Get-VM VM1 | Move-VM -Datastore ( Get-Datastore Datastore2 )
```

NOTE Storage VMotion allows to move a virtual machine that is powered on from one datastore to another. 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.

Use Virtual Machine Host Profiles

This scenario illustrates how to get use of the virtual machine host profiles.

To create and apply host profiles

1 Establish a connection to a server using the `Connect-VIServer` command:

```
Connect-VIServer -Server <Server_Address>
```

The server must be vCenter 4.0 or later. Earlier releases do not support host profiles.

2 Get the virtual machine host named `MyHost01` and store it in the `$h` variable:

```
$h = Get-VMHost MyHost01
```

3 Create a profile based on the `MyHost01` virtual machine host:

```
New-VMHostProfile -Name testProfile -Description "This is my test host profile."
-ReferenceHost $h
```

4 Get the newly created virtual machine host profile:

```
$p = ( Get-VMHostProfile -Name testProfile )[0]
```

5 Change the description of the `testProfile` host profile:

```
Set-VMHostProfile -Profile $p -Description "This is my test virtual machine host profile."
```

6 Get the `MyHost02` virtual machine host, on which to apply the `testProfile` virtual machine host profile:

```
$applyHost = Get-VMHost MyHost02
```

7 Associate the `MyHost02` virtual machine host with the `testProfile` host profile:

```
Set-VMHost -VMHost $applyHost -Profile $p
```

8 Test if the `MyHost02` host is compliant with the `testProfile` profile:

```
Test-VMHostProfileCompliance -VMHost $applyHost
```

The output of this command contains the host's incompliant settings, if any.

9 Apply the profile to the `MyHost02` host:

```
$neededVariables = Apply-VMHostProfile -Entity $applyHost -Profile $p -Confirm:$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.

10 Export the `testProfile` profile to a file:

```
Export-VMHostProfile -FilePath export.prf -Profile $p -Force
```

11 Import a new profile from the `export.prf` file:

```
Import-VMHostProfile -FilePath export.prf -Name importedProfile
```

12 Delete the created profiles:

```
Get-VMHostProfile -Name "testProfile","importedProfile" | Remove-VMHostProfile
-Confirm:$false
```

Manage Statistics and Statistics Intervals

This example scenario shows how to use the vSphere PowerCLI cmdlets to retrieve and manage inventory objects' statistics.

To create and manage statistics and statistics intervals

1 Establish a connection to a server by using the `Connect-VIServer` command:

```
Connect-VIServer -Server <Server_Address>
```

The server must have VirtualCenter 2.0 or higher installed. Earlier releases do not support creating statistics intervals.

2 Create a new statistics interval named `minute`:

```
New-StatsInterval -Name minute -SamplingPeriodSecs 60 -StorageTimeSecs 600
```

3 Create another statistics interval named `past hour`:

```
New-StatsInterval -Name "past hour" -SamplingPeriodSecs (60 * 60) -StorageTimeSecs 50000
```

NOTE The sampling period of a new statistics interval must be a multiple of the previous interval sampling period.

4 Create a third statistics interval named `past day`:

```
New-StatsInterval -Name "past day" -SamplingPeriodSecs ( 60 * 60 * 12 ) -StorageTimeSecs
500000
```

5 Extend the storage time of the `past day` statistics interval:

```
Set-StatsInterval -Interval "past day" -StorageTimeSecs 700000
```

6 List the available memory metric types for the `MyCluster` cluster:

```
$cluster = Get-Cluster MyCluster1

$statTypes = Get-StatType -Entity $cluster -Interval "past day" -Name mem.*
```

7 List the cluster statistics collected for the day:

```
Get-Stat -Entity $cluster -Start ([System.DateTime]::Now.AddDays(-1)) -Finish
([System.DateTime]::Now) -Stat $statTypes
```

Web Service Access Cmdlets

The vSphere PowerCLI 4.0 list of cmdlets includes two Web Service Access cmdlets:

- `Get-View`
- `Get-VIObjectByVIView`

They enable access to the programming model of the vSphere SDK for .NET from PowerShell and can be used to initiate vSphere .NET objects. Each object:

- Is a static copy of a server-side managed object and is not automatically updated when the object on the server changes.
- Includes properties and methods that correspond to the properties and operations of the server-side managed object. For more information about server-side object methods and properties, check the *VMware vSphere API Reference Guide* (http://www.vmware.com/support/pubs/sdk_pubs.html).

Using the Web Service Access cmdlets for low-level VMware vSphere management requires some knowledge of both PowerShell scripting and the VMware vSphere API.

Filter vSphere Objects

This procedure illustrates the use of the Get-View cmdlet in combination with a filter. The filter parameter is a HashTable 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.

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

To create and apply a filter

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

```
$filter = @{"Runtime.PowerState" ="poweredOn"; "Config.GuestFullName" = "Windows XP"}
```

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

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

Populate a View Object

This procedure illustrates how to populate a view object from an already retrieved managed object using the Get-View cmdlet.

To populate a view object

1 Get the Development2 virtual machine using a filter by name and populates the view object.

```
$vm = Get-View -ViewType VirtualMachine -Filter @{"Name" = "DevelopmentVM2}
$hostView = Get-View -Id $vm.Runtime.Host
```

2 Retrieve runtime information:

```
$hostView.Summary.Runtime
```

Update the State of a Server-Side Object

This procedure illustrates how to update the state of server-side objects.

To update the state of a server-side object

1 Get the Development2 virtual machine using a filter by name:

```
$vm = Get-View -ViewType VirtualMachine -Filter @{"Name" = "DevelopmentVM2}
$hostView = Get-View -Id $vm.Runtime.Host
```

2 Print the current power state:

```
$vm.Runtime.PowerState
```

3 Change the power state of the virtual machine:

```
If ($vm.Runtime.PowerState -ne “PoweredOn”) {
    $vm.PowerOnVM($vm.Runtime.Host)
} else {
    $vm.PowerOffVM()
}
```

4 Print the value of $vm power state (the power state is still not updated because the virtual machine property values are not updated automatically):

```
$vm.Runtime.PowerState
```

5 Update the view:

```
$vm.UpdateViewData()
```

6 Show the actual power state of the virtual machine:

```
$vm.Runtime.PowerState
```

Mixed Usage of vSphere PowerCLI and Web Service Access Cmdlets

To get more advantages of the usability and functionality of the vSphere PowerCLI cmdlets and the Web Service Access cmdlets you can use them together.

To reboot a virtual machine host

1 Use the `Get-VMHost` cmdlet to get a virtual machine host by its name, and pass the result to the `Get-View` cmdlet to retrieve the host view:

```
$hostView = Get-VMHost -Name "host.domain.com" | Get-View
```

"`host.domain.name`" is the IP address or the DNS name of the ESX host as shown in the inventory.

2 Call the reboot method of the host view object to reboot the host:

```
$hostView.RebootHost()
```

To modify the CPU levels of a virtual machine

This example shows how to modify the CPU levels of a virtual machine using combination of the `Get-View` and `Get-VIObjectByVIView` cmdlets.

1 Retrieve the `Development2` virtual machine, shut down it, and pass the result to the `Get-View` cmdlet to retrieve the virtual machine view object:

```
$vmView = Get-VM  DevelopmentVM2 | 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 = New-Object VMware.Vim.ResourceAllocationInfo;
$spec.CpuAllocation.Shares = New-Object VMware.Vim.SharesInfo;
$spec.CpuAllocation.Shares.Level = "normal";
$spec.CpuAllocation.Limit = -1;
$vmView .ReconfigVM_Task($spec)
```

3 Get a virtual machine object by using the `Get-VIObjectByVIView` cmdlet and start the virtual machine.

```
$vm =  Get-VIObjectByVIView $vmView  | Start-VM
```

The Inventory Provider

The Inventory Provider (`VimInventory`) is designed to expose a raw inventory view of the inventory items from a server. It enables interactive 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 obtain a view of its contents and the relationships between the items. In addition, you are able to manipulate objects (move, rename or delete them) by running commands from the vSphere PowerCLI console.

Basic Functions of the Inventory Provider

The following procedure illustrates some basic functions of the inventory provider.

To create and use a inventory drive

1 Establish a connection to the server using the `Connect-VIServer` command:

```
Connect-VIServer -Server <Server_Address>
```

For example:

```
Connect-VIServer -Server 192.168.10.10
```

When prompted, provide the administrator user name and password to authenticate access on the server.

2 Get the root folder of the server:

```
$root = Get-Folder -NoRecursion
```

3 Create a PowerShell drive named `VI`, based on the server root folder. You can use the built-in `New-psDrive` cmdlet.

```
New-PSDrive -Location $root -Name vi -PSProvider VimInventory -Root '\'
```

NOTE In this release, a single backslash is the required value for the `-Root` parameter.

A different way to create a inventory drive is to use the `New-VIInventoryDrive` command, which is an alias of `New-PSDrive.` It creates a new inventory drive using the `Name` and `Location` parameters.
For example:

```
Get-Folder -NoRecursion | New-VIInventoryDrive -Name vi
```

4 Access the new drive by running the following command:

```
cd vi:
```

To list the drive content, run

```
dir
```

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

5 Navigate through your server inventory using the `cd` command with the full path to the host. For a fictional VMware vSphere inventory, it might look like the following:

```
cd Folder01\DataCenter01\host\Web\LiveHost01
```

6 List the content of the host using the `ls` command:

```
ls
```

`ls` is the UNIX style alias of the `Get-ChildItem` cmdlet.

This command will return the virtual machines and the root resource pool of the host. To view only the virtual machines of the host, run the following cmdlet:

```
Get-VM
```

When called within the inventory drive, `Get-VM` retrieves only the virtual machines on the current drive location.

7 Delete a virtual machine named LiveVm01:

```
del LiveVm01
```

8 Rename a virtual machine from LiveVm01New to LiveVm01:

```
ren LiveVm01New LiveVm01
```

9 Start all virtual machines whose names start with LiveVm:

```
dir LiveVm* | Start-VM
```

Note that the provider file-style operations work on the same inventory objects as the vSphere PowerCLI cmdlets.

The Datastore Provider

The Datastore Provider (`VimDatastore`) 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.All file operations are case-sensitive.

NOTE On VirtualCenter 2.0 and ESX 3.0, Datastore Provider supports only browsing and deleting items. On VirtualCenter 2.5 and ESX 3.5, all file operations are available, including moving, copying, and renaming items.

Basic functions of the Datastore Provider

The following procedure illustrates some basic functions of the Datastore Provider.

To create and use a datastore drive

1 Establish a connection to the server using the Connect-VIServer command:

```
Connect-VIServer -Server <Server_Address>
```

For example:

```
Connect-VIServer -Server 192.168.10.10
```

When prompted, provide the administrator user name and password to authenticate access on the server.

2 Get a datstore by its name and assign it to the $datastore variable.

```
$datastore = Get-Datastore Storage1
```

3 Create a PowerShell drive ds:, based on $datastore. You can use the built-in New-PSDrive cmdlet.

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

A different way to create a datastore drive is to use the New-DatastoreDrive command, which is an alias of New-PSDrive. It creates a new datastore drive using the Name and Datastore parameters. For example:

```
Get-Datastore Storage1 | New-DatastoreDrive -Name ds
```

4 Access the new drive by running the following command:

```
cd ds:
```

To list the drive content, run:

```
dir
```

dir is an alias of the Get-ChildItem cmdlet.

5 Navigate to a specific folder on the ds: drive, using the cd command:

```
cd <Folder_Name>
```

For a fictional VMware vSphere inventory, the command might look like the following:

```
cd VirtualMachines\XPVirtualMachine
```

6 List the files of the folder using the ls command:

```
ls
```

ls is the UNIX style alias of the Get-ChildItem cmdlet.

7 Rename a file using the Rename-Item cmdlet or its alias ren. For example, to change the name of the vmware-3.log file to vmware-3old.log, run the following command:

```
ren vmware-3.log vmware-3old.log
```

All file operations apply only on files in the current folder.

8 Delete a file using the Remove-Item cmdlet or its alias del. For example, to remove the vmware-3old.log file from the XPVirtualMachine folder, use the following command:

del ds:\VirtualMachines\XPVirtualMachine\vmware-2.log

9 Copy a file using the Copy-Item cmdlet or its alias copy. For example:

```
copy ds:\VirtualMachines\XPVirtualMachine\vmware-3old.log ds:\VirtualMachines\vmware-3.log
```