vSphere PowerCLI Administration Guide

VMware vSphere PowerCLI 4.1 Update 1

EN-000406-00

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.

VMware Technical Publications Glossary

VMware Technical Publications provides a glossary of terms that might be unfamiliar to you. For definitions of terms as they are used in VMware technical documentation go to http://www.vmware.com/support/pubs.

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

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 4.1 Update 1 ships with more than 200 PowerShell-based cmdlets. The PowerCLI also includes 2 .NET cmdlets for use through PowerShell—the Web Service Access Cmdlets. For more information about the .NET cmdlets, see "Web Service Access Cmdlets" on page 31.

vSphere PowerCLI cmdlets are created to answer the specific needs of the VMware vSphere administration and management. All vSphere PowerCLI cmdlets are found in the `VMware.VimAutomation.ViCore.Cmdlets` 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-PowerCLIEnvironment.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-PowerCLIEnvironment.ps1` script configuration file is not started automatically. To load it, type its name in the console window without specifying the path:

```
Initialize-PowerCLIEnvironment.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 pass
```

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:

```
Start-VM <virtual_machine_name>
```

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

```
Start-VM MyVM
```

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`.

PowerShell offers two risk mitigation parameters in PowerShell: `WhatIf` and `Confirm`. `Whatif` is used when you want to see the effects of a command without running 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.

Specifying Objects

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

Managing Default Servers

vSphere PowerCLI cmdlets run on default vSphere servers, if no target servers can be determined from the provided parameters.

When you connect to a vSphere server 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-Server` to close all active connections to this server, or modify the value of `$DefaultVIServers` manually.

vSphere PowerCLI allows you to work with a single default server instead of using multiple default servers. In this case, the `$DefaultVIServers` variable always contains the last connected server, and its value is updated every time you connect to a new server. Working with a single default server is deprecated and will be removed in a following release.

To switch to a single default server mode

1 Run `Get-PowerCLIConfiguration` to view the actual PowerCLI configuration:

```
Get-PowerCLIConfiguration
```

2 Run `Set-PowerCLIConfiguration` to change the default server mode to `Single`:

```
Set-PowerCLIConfiguration -DefaultVIServerMode Single
```

A lot of PowerCLI cmdlets have a parameter named `Server`. The `Server` parameter allows you to run the cmdlet on servers different from the default ones. This parameter takes both server names and `VIServer` objects.

Running PowerCLI Cmdlets Asynchronously

By default, vSphere 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. In this case, 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 automatically updated 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 initiating other commands, use the `Wait-Task` cmdlet.

Example 2-2. Running Remove-VM With and Without the RunAsync Parameter

```
Remove-VM $vmList
```

The command returns with no output when all virtual machines stored in the `$vmList` variable are removed (simultaneously or not).

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

The command returns immediately and the output consists of one or more `Task` objects.

In vSphere PowerCLI, the `RunAsync` parameter affects only the cmdlets' 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 vSphere 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 2-3. Removing Virtual Machines Consecutively

```
foreach ($vm in $vmList)
    {
        Remove-VM $vm1
    }
```

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

Some PowerCLI features support only Windows XP, Windows Server 2003, and Linux RedHat Enterprise 5. To add support for other guest operating systems, you can use the scripts that are located in the `Script` folder in the PowerCLI installation directory or 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`, and `Remove-VMGuestRoute` must follow the file naming format `<CmdletName>_<OS_Identifier>`, 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 the hard disk resizing functionality of `Set-HardDisk` must follow the file naming format `GuestDiskExpansion_<OS_Identifier>,` where `<OS_Identifier>` is the guest family or the guest ID (as returned by `Get-VMGuest`).

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 prompts for user credentials, as they are not passed as parameters:

```
Connect-VIServer -Server <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-PowerCLIConfiguration -ProxyPolicy NoProxy
```

In vSphere PowerCLI, you can have more than one connections to the same server. To disconnect from a server, you must close all active connections to this server running the `Disconnect-VIServer` cmdlet.

Basic Virtual Machine 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 operating system 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 `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 Retrieve information of the guest OS of the VM virtual machine:

```
Get-VMGuest VM | fc
```

5 Shutdown 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.

Basic Virtual Machine Host Operations

The following examples illustrate 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 add a standalone host to the vCenter Server

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

```
Get-VMHost
```

2 Add the MyHost standalone host:

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

To activate maintenance mode for a host

1 Save the MyHost host object as a variable:

```
$myHost = Get-VMHost -Name MyHost
```

2 Retrieve the cluster to which MyHost belongs and save the cluster object as a variable:

```
$hostCluster = Get-Cluster -VMHost $myHost
```

3 Initialize a task that activates maintenance mode for the MyHost host and save the task object as a variable:

```
$updateHostTask = Set-VMHost -VMHost $myHost -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 Retrieve and apply the recommendations generated by DRS:

```
Get-DrsRecommendation -Cluster $hostCluster | where {$_.Reason -eq "Host is entering
maintenance mode"} | Apply-DrsRecommendation
```

5 Retreve the task output object and save it as a variable:

```
$myUpdatedHost = Wait-Task $updateHostTask
```

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, the datastore provider, 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 system:

```
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 `MainFolder` in it:

```
$mainFolder = Get-Folder -NoRecursion | New-Folder -Name MainFolder
```

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

3 Create a new datacenter called `MyDC` in the `MainFolder` folder:

```
New-Datacenter -Location $mainFolder -Name MyDC
```

4 Create a folder called MyFolder01 under MyDC:

```
Get-Datacenter MyDC | New-Folder -Name MyFolder01
$myFolder01 = Get-Folder -Name MyFolder01
```

NOTE Search in PowerShell is not case-sensitive.

5 Create a new cluster MyCluster01 in the MuFolder01 folder:

```
New-Cluster -Location $MyFolder01 -Name MyCluster01 -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:

```
$myHost01 = Add-VMHost -Name 10.23.112.345 -Location ( Get-Cluster MyCluster01 )
```

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:

```
$myClusterRootRP = Get-ResourcePool -Location ( Get-Cluster MyCluster01 ) -Name Resources
New-ResourcePool -Location $clusterRootRP -Name MyRP01 -CpuExpandableReservation $true
-CpuReservationMhz 500 -CpuSharesLevel high -MemExpandableReservation $true -MemReservationMB
500 -MemSharesLevel high
```

8 Create a virtual machine synchronously:

```
New-VM -Name MyVM1 -VMHost $myHost01 -ResourcePool ( Get-ResourcePool MyRP01 ) -DiskMB 4000
-MemoryMB 256
```

9 Create a virtual machine asynchronously:

```
$vmCreationTask = New-VM -Name MyVM2 -VMHost $myHost01 -ResourcePool (Get-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 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. The example 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 MyVM2 virtual machine:

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

2 Create a template from the MyVM1 virtual machine:

```
New-Template -VM MyVM1 -Name MyVM1Template -Location (Get-Datacenter MyDC )
```

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 MyVM3:

```
Get-Template MyVM1Template | Set-Template -ToVM -Name MyVM3
```

4 Create a template from the MyVM2 virtual machine:

```
New-Template -VM MyVM2 -Name MyVM2Template -Location (Get-Datacenter MyDC )
```

5 Convert this template to a virtual machine named MyVM4:

```
Get-Template MyVM2Template | Set-Template -ToVM -Name MyVM4
```

6 Move the virtual machines into the MyRP01 resource pool:

```
Get-VM MyVM? | Move-VM -Destination ( Get-ResourcePool MyRP01 )
```

Use ? as a wildcard to match just one symbol. The command returns all virtual machines whose names start with MyVM and have one more symbol at the end. In this example, MyVM1, MyVM2, MyVM3, and MyVM4 are retrieved and moved into the MyRP01 resource pool.

7 Start the virtual machines in the MyRP01 resource pool:

```
Get-ResourcePool MyRP01 | 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>MyVM1</Name>
        <HDDCapacity>10000</HDDCapacity>
    </VM>
    <VM>
        <Name>MyVM2</Name>
        <HDDCapacity>10000</HDDCapacity>
    </VM>
</CreateVM>
```

To create a virtual machine by using an XML specification file

1 Read the content of the myVM.xml file:

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

2 Create the virtual machines:

```
$s.CreateVM.VM | foreach { New-VM -VMHost 192.168.10.11 -Name $_.Name -DiskMB $_.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 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-VM -Location ( Get-ResourcePool MyRP01 )
foreach( $vm in $VMs ) { Set-VM -VM $vm -Snapshot ( Get-Snapshot -VM $vm -Name
InitialSnapshot ) }
```

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 10.23.114.123
```

2 Retrieve the resource configuration for the `MyVM1` virtual machine.:

```
Get–VMResourceConfiguration –VM ( Get–VM MyVM1 )
```

3 Display the disk share of the `MyVM1` virtual machine:

```
Get–VMResourceConfiguration –VM ( Get–VM MyVM1 ) | Format–Custom –Property
DiskResourceConfiguration
```

4 Change the memory share of the `MyVM1` virtual machine to low:

```
Get–VM MyVM1 | Get–VMResourceConfiguration | Set–VMResourceConfiguration –MemSharesLevel low
```

5 Update the CPU share of the `MyVM1` virtual machine to high:

```
Get–VM MyVM1 | Get–VMResourceConfiguration | Set–VMResourceConfiguration –CpuSharesLevel high
```

6 Change the disk share of the `MyVM1` virtual machine to 100:

```
$myVM1 = Get–VM MyVM1
$myVM1disk = Get–HardDisk $myVM1
Get–VMResourceConfiguration $myVM1 | Set–VMResourceConfiguration –Disk $myVM1disk
–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 `MyDC`:

```
Get–Datacenter MyDC | Get–VMHost | Format–Custom
```

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

```
Get–Datacenter MyDC | 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 `MyDC` datacenter:

```
Get–Datacenter MyDC | 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 “MyDC” | 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 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 MyESXHost1 | 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 10.23.111.235
```

2 Retrieve the MyVM1 virtual machine and move it to the host named ESXHost2:

```
Get-VM MyVM1 | 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 10.23.111.235
```

2 Retrieve the MyVM1 virtual machine and move it to the datastore named MyDatastore2:

```
Get-VM VM1 | Move-VM -Datastore ( Get-Datastore MyDatastore2 )
```

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 10.32.110.123
```

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 MyHostProfile01 -Description "This is my test profile based on
MyHost01." -ReferenceHost $h
```

4 Get the newly created virtual machine host profile:

```
$hp01 = ( Get-VMHostProfile -Name MyHostProfile01 )[0]
```

5 Change the description of the MyHostProfile01 host profile:

```
Set-VMHostProfile -Profile $hp01 -Description "This is my old test host profile based on
MyHost01."
```

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

```
$myHost02 = Get-VMHost MyHost02
```

7 Associate the MyHost02 virtual machine host with the MyHosProfile01 host profile:

```
Set-VMHost -VMHost $myHost02 -Profile $hp01
```

8 Test if the MyHost02 host is compliant with the MyHostProfile01 profile:

```
Test-VMHostProfileCompliance -VMHost $myHost02
```

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 $myHost02 -Profile $hp01 -Confirm:$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 MyHostProfile01 profile to a file:

```
Export-VMHostProfile -FilePath export.prf -Profile $hp01 -Force
```

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

```
Import-VMHostProfile -FilePath export.prf -Name MyImportedProfile01
```

12 Delete the created profiles:

```
Get-VMHostProfile -Name "MyHostProfile01","MyImportedProfile01" | 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 10.32.110.123
```

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-StatInterval -Name minute -SamplingPeriodSecs 60 -StorageTimeSecs 600
```

3 Create another statistics interval named past hour:

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

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

```
Set-StatInterval -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
```

Configure the NIC Teaming Policy of a Virtual Switch

This example scenario illustrates how to change the load balancing and failover settings of a virtual switch and determine the unused NICs.

To configure the NIC teaming policy of a virtual switch

1 Retrieve 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 make unused in separate variables:

```
$pn5 = $pn.PhysicalNic[2]
$pn6 = $pn.PhysicalNic[3]
$pn7 = $pn.PhysicalNic[0]
```

3 Retrieve the NIC teaming policy of the `VSwitch01` virtual switch:

```
$policy = Get-VirtualSwitch -VMHost ( Get-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 settings of the virtual switch NIC teaming policy:

```
$policy | Set-NicTeamingPolicy -BeaconInterval 3 -LoadBalancingPolicy 3
-NetworkFailoverDetectionPolicy 1 -NotifySwitches $false -FailbackEnabled $false
```

Manage Virtual Appliances

These examples illustrate how to create and manage virtual appliances using the PowerCLI cmdlets.

To create and start a virtual appliance

1 Create a new virtual appliance named `MyVApp` on the specified host:

```
New-VApp -Name MyVApp -CpuLimitMhz 4000 -CpuReservationMhz 1000 -Location ( Get-VMHost
MyHost01 )
```

2 Start the new virtual appliance:

```
Start-VApp MyVApp
```

To change the properties of a virtual appliance

1 Retrieve and stop the MyVApp virtual appliance:

```
Get-VApp MyVApp | Stop-VApp -Confirm:$falseStop-VApp
```

2 Change the name and memory reservation of the MyVApp virtual appliance:

```
Get-VApp MyVApp | Set-VApp -Name OldVApp -MemReservationMB 2000
```

To export a virtual appliance

1 Retrieve the virtual appliance you want to export:

```
$oldVApp = Get-VApp OldVApp
```

2 Export the OldVApp virtual appliance to a local directory and name the exported appliance WebApp:

```
Export-VApp -VApp $oldVApp -Name WebApp -Destination D:\vapps\ -CreateSeparateFolder
```

To import a virtual appliance

1 Import the WebApp virtual appliance from the specified location to the Storage2 datastore:

```
Import-VApp -Source D:\vapps\WebApp\WebApp.ovf -VMHost ( Get-VMHost MyHost01 ) -Datastore (
Get-Datastore -VMHost MyHost01 -Name Storage2 )
```

2 Remove the WebApp appliance and delete it from the datastore:

```
Remove-VApp WebApp -DeletePermanently -Confirm:$false
```

Manage Guest Networks

The following examples illustrate how to retrieve and configure guest network interaces and routes.

To retrieve and configure a network interface

1 Retrieve the guest network interface of the MyVM1 virtual machine:

```
$myVM1 = Get-VM -Name MyVM1
$interface = Get-VMGuestNetworkInterface -VM $myVM1 -HostUser root -HostPassword pass1
-GuestUser user -GuestPassword pass2
```

2 Retrieve the network interface of a guest OS:

```
$guest = Get-VMGuest $myVM1
$interface = Get-VMGuestNetworkInterface -VMGuest $guest -HostUser root -HostPassword pass1
-GuestUser user -GuestPassword pass2 -ToolsWaitSecs 100
```

3 Configure the network interface:

```
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
```

To create a guest route

1 Retrieve the existing routes of the virtual machine stored in the $myVM1 variable:

```
Get-VMGuestRoute -VM $myVM1 -HostUser root -HostPassword pass1 -GuestUser user -GuestPassword
pass2 -ToolsWaitSecs 50
```

2 Retrieve the existing routes of the guest OS stored in the $guest variable:

```
Get-VMGuestRoute -VMGuest $guest -HostUser root -HostPassword pass1 -GuestUser user
-GuestPassword pass2
```

3 Create a new guest route:

```
$route = New-VMGuestRoute -VM $vmWin -HostUser root -HostPassword pass1 -GuestUser user
-GuestPassword pass2 -Destination 192.168.100.10 -Netmask 255.255.255.255 -Gateway
10.23.112.58 -Interface $interface.RouteInterfaceId -ToolsWaitSecs 50
```

4 Remove the guest route:

```
Remove-VMGuestRoute -VMGuestRoute $route -HostUser root -HostPassword pass1 -GuestUser user
-GuestPassword pass2 -ToolsWaitSecs 100 -Confirm:$false
```

NOTE Retrieving and configuring guest network interfaces and routes is supported only on servers that are ESX 3.5 and later.

Work with Host Storages and iSCSI HBA Devices

The following example illustrates enabling iSCSI on a host, adding iSCSI targets, and creating host storages.

To create a new iSCSI host storage

1 Enable software iSCSI on the host:

```
$myHost = Get-VMHost MyESXHost1
Get-VMHostStorage $myHost | Set-VMHostStorage -SoftwareIScsiEnabled $true
```

2 Retrieve the iSCSI HBA on the host:

```
$iscsiHba = Get-VMHostHba -Type iScsi
```

3 Add a new iSCSI target for dynamic discovery (the default port number is 3260):

```
$iscsiHba | New-IScsiHbaTarget -Address 192.168.0.1 -Type Send
```

4 Rescan the HBAs on the host system:

```
Get-VMHostStorage $myHost -RescanAllHba
```

5 Get the lun path (we need the one who's canonical name starts with the device name of the iSCSI HBA):

```
$lunPath = Get-ScsiLun -VMHost $myHost -CanonicalName ($iscsiHba.Device + "*") |
Get-ScsiLunPath
```

6 Create the new storage:

```
New-Datastore -Vmfs -VMHost $myHost -Path $lunpath.LunPath -Name iSCSI
```

Manage PCI and SCSI Passthough Devices

The following example demonstrates working with PCI and SCSI passthrough devices.

To retrieve and add passthrough devices of a host and virtual machine

1 Retrieve the PCI passthrough devices of the `MyESXHost` host:

```
$myHost = Get-VMHost MyESXHost
Get-PassthroughDevice -VMHost $myHost -Type Pci
```

2 Retrieve the SCSI passthrough devices of the `MyVM` virtual machine:

```
$vm = Get-VM MyVM
Get-PassthroughDevice -VM $vm -Type Scsi
```

3 Add a SCSI passthrough device to the `MyVM` virtual machine:

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

4 Remove all passthrough devices of the `MyVM` virtual machine:

```
Get-PassthroughDevice -VM $vm | Remove-PassthroughDevice
```

Create Custom Properties for vSphere Objects

In PowerCLI, you can create custom properties to add more information to vSphere objects. All custom properties are available during the life of the Powershell process or until you removed them using the `Remove-VIProperty` cmdlet. Two types of custom properties are avilable to you:

- **Script properties** are defined by a name and a script that evaluates when the custom property is retrieved for first time.
- **Properties based on extension data properties** refer directly to the property of the corresponding .NET view object.

To create a custom property named ToolsVersion for VirtualMachine objects

1 Create the new custom property based on the `Guest.ToolsVersion` property:

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

2 List the `ToolsVersion` properties of the available virtual machines:

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

To create a custom script property named NameOfHost for VirtualMachine objects

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

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

2 List the `NameOfHost` properties of the available virtual machines:

```
Get-VM | select Name, NameOfHost | Format-Table -AutoSize
```

Apply Customization Specifications to Virtual Machines

In PowerCLI, there are two types of customization specification objects:

- **Persistent** customization specification objects are stored on the vSphere Server. All persistent customization specifications created by vSphere Client or PowerCLI 4.1 or higher are encrypted. Encrypted customization specifications can be applied only by the server which has encrypted them.
- **Non-persistent** customization specification objects exist only inside the current Powershell process. Non-persistent customization specification objects are not encrypted, but cloning them to a vSphere server encrypts them.

To apply a customization object to a cloned virtual machine

1 Retrieve the `Spec` customization specification from the server 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 `VM1` (the name of the virtual machine you want to create):

```
Set-OSCustomizationSpec -OSCustomizationSpec ClientSpec -NamingPrefix VM1
```

3 Create the `VM1` virtual machine 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
```

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

1 Create a non-persistent 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 Retrieve the default NIC mapping objects of the `Spec` specification:

```
Get-OSCustomizationNicMapping -OSCustomizationSpec Spec | Set-OSCustomizationNicMapping
-IpMode UseStaticIP -IpAddress 172.16.1.30 -SubnetMask 255.255.255.0 -DefaultGateway
172.16.1.1 -Dns 172.16.1
```

Each customization specification object has one default NIC mapping object.

3 Modify the default NIC mapping object of the `Spec` customization specification with static IP settings:

```
Get-OSCustomizationNicMapping -OSCustomizationSpec 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
```

To modify multiple NIC mapping objects of a customization specification and apply the specification on an existing virtual machine

1 Retrieve the network adapters for the `VM` virtual machine:

```
Get-NetworkAdapter VM
```

The `VM` virtual machine has two network adapters. The first one is connected to the network with a DHCP server enabled and the second one is connected in a private network.

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 positions, 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 -OSCustomizationSpec Spec -IpMode UseStaticIP -IpAddress
172.16.1.30 -SubnetMask 255.255.255.0 -DefaultGateway 172.16.1.1 -Dns 172.16.1.1
```

4 Retrieve the NIC mapping objects and verify that there are two NIC mapping objects. The default NIC mapping object is DHCP enabled and the newly added one uses a static IP address:

```
Get-OSCustomizationNicMapping -OSCustomizationSpec Spec
```

5 Apply the `Spec` customization specification to the `VM` virtual machine:

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

6 Correlate a network adapter from the `VMNetwork` network with the NIC mapping object that uses DHCP mode:

```
$netAdapter = Get-NetworkAdapter VM | where { $_.NetworkName -eq 'VMNetwork' }
Get-OSCustomizationNicMapping -OSCustomizationSpec Spec | where { $_.IPMode -eq 'UseDHCP' } |
Set-OSCustomizationNicMapping -NetworkAdapterMac $netAdapter.MacAddress
```

Manage Alarms

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

To retrieve the action triggers for a specified alarm

1 Extract all PowerCLI supported alarm actions for the `Host processor status` alarm:

```
Get-AlarmDefinition -Name "Host processor status" | Get-AlarmAction -ActionType
"ExecuteScript", "SendSNMP", "SendEmail"
```

2 Get all the triggers for the first alarm definition found:

```
Get-AlarmAction -AlarmDefinition (Get-AlarmDefinition | select -First 1) |
Get-AlarmActionTrigger
```

To create and modify alarm actions and action triggers

1 For all host alarms, increase the interval after the action repeats by one:

```
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:

```
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:

```
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:

```
Get-AlarmDefinition -Name AlarmDefinitionNew | New-AlarmAction -Snmp
```

5 Create a Script alarm action:

```
Get-AlarmDefinition -Name AlarmDefinitionNew | New-AlarmAction -Script -ScriptPath
'c:\test.ps1'
```

6 Create an action trigger on all actions for the specified alarm:

```
Get-AlarmDefinition -Name AlarmDefinitionNew | Get-AlarmAction | New-AlarmActionTrigger
-StartStatus 'Red' -EndStatus 'Yellow' -Repeat
```

To remove alarm actions and triggers

1 Remove the first action trigger found for an alarm definition:

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

2 Remove all the actions for an alarm definition:

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

Create and Modify Advanced Settings for Cluster and VIServer Object

You can create and modify advanced settings for `Cluster` and `VIServer` objects.

To create and modify advanced settings of a cluster

1 Create a new cluster named Cluster:

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

2 Create two advanced settings for the new cluster:

```
$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:

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

4 Create an advanced setting with a specific name:

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

5 Retrieve the `Service Console` setting and store it in a variable:

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

NOTE You need to mark the wildcard characters [and] in the advanced setting name by using the ` character.

6 Remove the new advanced settings:

```
($setting1, $setting2, $setting3) |  Remove-AdvancedSetting -Confirm:$false
```

To modify the vCenter Server email configuration

1 Retrieve the current e-mail configuration settings of the vCenter Server stored in the `$srv` variable:

```
Get-AdvancedSetting -Entity $srv  -Name mail.*
```

2 Update the SMTP server name and port:

```
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
```

To modify the vCenter Server SNMP configuration

1 Retrieve the current SNMP configuration settings of the vCenter Server stored in the `$srv` variable:

```
Get-AdvancedSetting -Entity $srv  -Name snmp.*
```

2 Modify the SNMP receiver data:

```
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
```

Use ESX CLI with PowerCLI

PowerCLI provides two approaches for working with ESX CLI:

- Through the `Get-ESXCli` cmdlet, which provides direct access to the ESX CLI namespaces, applications, and commands.
- Through .NET methods, which you use to create managed objects that correspond to specific ESX CLI applications. To access the ESX CLI, you can invoke methods on these managed objects. There are two .NET methods you can use for retrieving information of ESX CLI applications:

To work with ESX CLI by using the Get-ESXCli cmdlet

1 Retrieve an ESX CLI instance:

```
$myEsxCli = Get-EsxCli
```

2 View the properties of an ESX CLI instance:

```
$myEsxCli | Get-Member
```

3 Browse inside a specific namespace and view the available applications:

```
$myEsxCli.nmp | Get-Member
```

4 (opt) Retrieve help on a specific application:

```
nmpDeviceHelp = $script:esxCli.nmp.device.help()
```

5 Retrieve the methods of a specific application:

```
$myEsxCli.nmp.device | Get-Member
```

6 (opt) Retrieve help on a specific method:

```
$script:esxCli.nmp.device.help("setpolicy")
```

7 Retrieve a list of methods and store them in a variable:

```
$scsiLunDeviceList = $myEsxCli.nmp.device.list()
```

8 Call the specific method:

```
$myEsxCli.nmp.device.setpolicy($null, $scsiLunDeviceList[0].Device, "VMW_PSP_MRU")
```

To work with ESX CLI using .NET methods

1 Retrieve all managed object instances:

```
$myEsxCli.TypeManager.QueryMoInstances($null)
```

The `QueryMoInstances` method queries all managed object instances that correspond to ESX CLI applications in the high-level binding, both predefined and third-party.

2 Retrieve information for a specific managed object type:

```
$nmpDeviceTypeInfo = $myEsxCli.TypeManager.QueryTypeInfo("vim.EsxCLI.nmp.device")
```

The `QueryTypeInfo` method provides full information about a specific managed object type and its corresponding ESX CLI application.

3 Retrieve the methods available for a specific managed object type:

```
$typeInfo.managedTypeInfo[0].method
```

4 Retrieve an instance of a managed object:

```
$nmpManageObjectInstance
=$myEsxCli.TypeManager.CreateDynamicManagedObject(("ha-cli-handler-nmp-device")
```

The `CreateDynamicManagedObject` method is used to obtain instances of ESX CLI managed objects.

5 Retrieve the methods of a managed object instance by using the `InvokeOperation` method:

```
$nmpManageObjectInstance. InvokeOperation("setpolicy", $cmdParameters)
```

6 Retrieve help about a specific managed object by using the `FetchCLIInfo` method:

```
$localCliInfo =
$myEsxCli.TypeManager.CreateDynamicManagedObject("ha-dynamic-type-manager-local-cli-cliinfo")
$localCliInfo.InvokeOperation("FetchCLIInfo", @{"typeName"=" vim.EsxCLI.nmp.device "})
```

Use ESX Top with PowerCLI

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

To see the used and ready percentage of the virtual CPUs of a virtual machine

1 Identify the group to which the virtual machine belongs:

```
$group = Get-EsxTop -CounterName SchedGroup | where {$_.VMName -eq $vm.Name}
```

2 Collect the IDs of all virtual CPUs of the virtual machine and store them in an array:

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

3 Retrieve the CPU statistics for the virtual machine:

```
$cpuStats = Get-EsxTop -CounterName VCPU | where {$cpuIds -contains $_.VCPUID}
```

4 To calculate the used and ready percentage, use the `UsedTimeInUsec` and `ReadyTimeInUsec` stats:

```
$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 Retrieve the used and ready percentage for each virtual CPU of the virtual machine:

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

Web Service Access Cmdlets

The vSphere PowerCLI 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 operating system 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 MyVM2 virtual machine using a filter by name and populates the view object.

```
$myVM2 = Get–View –ViewType VirtualMachine –Filter @{"Name" = "MyVM2}
$hostView = Get–View –Id $myVM2.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 MyVM2 virtual machine using a filter by name:

```
$myVM2 = Get–View –ViewType VirtualMachine –Filter @{"Name" = "MyVM2}
$hostView = Get–View –Id $myVM2.Runtime.Host
```

2 Print the current power state:

```
$myVM2.Runtime.PowerState
```

3 Change the power state of the virtual machine:

```
If ($myVM2.Runtime.PowerState –ne “PoweredOn”) {
    $vm.PowerOnVM($myVM2.Runtime.Host)
} else {
    $myVM2.PowerOffVM()
}
```

4 Print the value of $myVM2 power state (the power state is still not updated because the virtual machine property values are not updated automatically):

```
$myVM2.Runtime.PowerState
```

5 Update the view:

```
$myVM2.UpdateViewData()
```

6 Show the actual power state of the virtual machine:

```
$myVM2.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 MyHost | Get–View
```

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

1 Retrieve the MyVM2 virtual machine, shut down it, and pass the result to the Get-View cmdlet to retrieve the virtual machine view object:

```
$vmView = Get-VM  MyVM2 | 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.

```
$myVM =  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.

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 all vSphere servers connected within the current vSphere PowerCLI session..

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

Basic Functions of the Inventory Provider

The following procedure illustrates some basic operations with the inventory provider.

To view the content of a default inventory drive

1 Access the vi inventory drive:

```
cd vi:
```

2 List the drive content:

```
dir
```

dir is an alias of the Get-ChildItem cmdlet.

To create a new custom inventory drive

1 Get the root folder of the server:

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

2 Create a PowerShell drive named myVi in the server root folder:

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

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

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

A different way to create a inventory drive is to map an existing inventory path:

```
New-PSDrive -Name myVi -PSProvider VimInventory -Root “vi:\Folder01\Datacenter01”
```

To manage inventory objects through inventory drives

1 Navigate through your server inventory by running the `cd` command with the full path to the host:

```
cd Folder01\DataCenter01\host\Web\Host01
```

2 List 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` retrieves only the virtual machines on the current drive location.

4 Delete a virtual machine named `VM1`:

```
del VM1
```

5 Rename a virtual machine from `VM1New` to `VM1`:

```
ren VM1New VM1
```

6 Start all virtual machines whose names start with `VM`:

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

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.

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

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

Basic Functions of the Datastore Provider

The following procedures illustrate some basic functions of the Datastore Provider.

To browse a default datastore drive

1 Access the `vmstore` drive:

```
cd vmstore:
```

2 List the drive content:

```
dir
```

To create a new custom datastore drive

1 Get a datstore 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 that 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
```

A different way to create a datastore drive is to map an existing datastore path. For example:

```
New-PSDrive -Name ds -PSProvider VimDatastore -Root
vmstore:\Folder01\Datacenter01\Datastore01\Folder01
```

To manage datastores through datastore drives

1 Navigate to a specific folder on the `ds:` drive:

```
cd VirtualMachines\XPVirtualMachine
```

2 List the files of the folder, using the `ls` command:

```
ls
```

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

3 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.

4 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
```

5 Copy a file, using 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, using 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, using the `New-Item` cmdlet or its alias `mkdir`:

```
mkdir -Path ds:\VirtualMachines -Name Folder01 -Type Folder
```

8 Download a file to the local machine using 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, using the `Copy-DatastoreItem` cmdlet:

```
Copy-DatastoreItem C:\Temp\vmware-3.log  ds:\VirtualMachines\XPVirtualMachine\vmware-3new.log
```