

# vSphere PowerCLI User's Guide

VMware vSphere PowerCLI 5.0

EN-000462-00

**vmware**<sup>®</sup>

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](mailto:docfeedback@vmware.com)

Copyright © 2010, 2011 VMware, Inc. All rights reserved. This product is protected by U.S. and international copyright and intellectual property laws. VMware products are covered by one or more patents listed at <http://www.vmware.com/go/patents>.

VMware is a registered trademark or trademark of VMware, Inc. in the United States and/or other jurisdictions. All other marks and names mentioned herein may be trademarks of their respective companies.

**VMware, Inc.**  
3401 Hillview Ave.  
Palo Alto, CA 94304  
[www.vmware.com](http://www.vmware.com)

# Contents

About This Book	3
<b>1 Introduction to vSphere PowerCLI</b>	<b>5</b>
Microsoft PowerShell Basics	5
PowerShell Command-Line Syntax	5
PowerShell Pipelines	5
PowerShell Wildcards	5
PowerShell Common Parameters	6
vSphere PowerCLI Specific Concepts	6
Specifying Objects in PowerCLI	6
Running PowerCLI Cmdlets Asynchronously	7
Using Custom Scripts to Extend the Operating System Support for PowerCLI Cmdlets	7
Launching vSphere PowerCLI	8
Adding External PowerCLI Snapins	8
Getting Help About a Cmdlet	8
Connecting to a Server	9
Managing Default Server Connections with PowerCLI	9
<b>2 vSphere PowerCLI System Requirements</b>	<b>11</b>
PowerCLI Installation Prerequisites	11
Supported Operating Systems	11
Supported VMware Environments	11
<b>3 Installing vSphere PowerCLI</b>	<b>13</b>
Downloading the vSphere PowerCLI Binary	13
Installing vSphere PowerCLI on Windows	13
Setting the Properties to Support Remote Signing	14
Uninstalling vSphere PowerCLI	14
<b>4 vSphere PowerCLI Usage Examples</b>	<b>15</b>
Basic Cmdlets Usage	15
Display Help for a Cmdlet	15
Connect to a Server	15
Basic Virtual Machine Operations	16
Basic Host Operations	16
Advanced Cmdlets Usage	17
Create vSphere Objects	17
Use Virtual Machine Templates	18
Create Virtual Machines Using an XML Specification File	19
Create Snapshots	19
Update the Resource Configuration Settings of a Virtual Machine	20
List Hosts and Display Their Properties	20
Change the Host Advanced Configuration Settings	20
Migrate a Virtual Machine	21
Manage Host Profiles	21
Manage Statistics and Statistics Intervals	22
Configure the NIC Teaming Policy of a Virtual Switch	22

Manage Virtual Appliances	23
Manage Guest Networks	23
Work with Host Storages and iSCSI HBA Devices	24
Manage PCI and SCSI Passthrough Devices	25
Create Custom Properties for vSphere Objects	25
Apply Customization Specifications to Virtual Machines	25
Manage vSphere Permissions, Roles, and Privileges	27
Manage Alarms	27
Create and Modify Advanced Settings for Cluster and VServer Object	28
Use ESXCLI with PowerCLI	29
Use esxtop with PowerCLI	30
API Access Cmdlets	31
Filter vSphere Objects	31
Populate a View Object	32
Update the State of a Server-Side Object	32
Mixed Usage of vSphere PowerCLI and API Access Cmdlets	32
The Inventory Provider	33
Basic Functions of the Inventory Provider	33
The Datastore Provider	34
Basic functions of the Datastore Provider	34

# About This Book

---

The *vSphere PowerCLI User's Guide* provides information about installing and using the VMware vSphere PowerCLI cmdlets (pronounced “commandlets”) for managing, monitoring, automating, and handling life-cycle operations for VMware vSphere components – virtual machines, datacenters, storage, networks, and so on.

## Intended Audience

This book is intended for anyone who needs to install and 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](mailto: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](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>.

# Introduction to vSphere PowerCLI

---

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.

- [“Microsoft PowerShell Basics”](#) on page 5
- [“vSphere PowerCLI Specific Concepts”](#) on page 6

## Microsoft PowerShell Basics

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

### PowerShell Command-Line Syntax

PowerShell cmdlets use a consistent verb-noun structure, where the verb specifies the action and the noun specifies the object to operate on. PowerShell cmdlets follow consistent naming patterns, which makes it easy to figure out how to construct a command if you know the object you want to work with.

All command categories take parameters and arguments. A parameter starts with a hyphen and is used to control the behavior of the command. An argument is a data value consumed by the command.

A simple PowerShell command has the following syntax:

```
command -parameter1 -parameter2 argument1 -argument2
```

### PowerShell Pipelines

A pipeline is a series of commands separated by the pipe operator |. Each command in the pipeline receives an object from the previous command, performs some operation on it, and then passes it 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.

### PowerShell 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.

## PowerShell 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 Concepts

vSphere PowerCLI cmdlets are created to answer the needs of the VMware vSphere administration and introduce some specific concepts.

### Specifying Objects in PowerCLI

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 1-1. 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")
```

## 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 1-2. 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 1-3. 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 7, Windows Server 2008, 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`, `Remove-VMGuestRoute`, and `Set-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`).

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

Administrators can edit and extend the script to define cmdlets aliases, configure the environment, or set vSphere PowerCLI start up actions.

---

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

---

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

```
Add-PSSnapin <snapin_name>
```

In this case, the `Initialize-PowerCLIEnvironment.ps1` script configuration file is not started automatically and you must load it by typing its name in the console window:

```
Initialize-PowerCLIEnvironment.ps1
```

## Adding External PowerCLI Snapins

The vSphere PowerCLI package includes three additional snapins for managing specific vSphere features:

- `VMware.VimAutomation.License` - The Licensing snapin contains the `Get-LicenseDataManager` cmdlet for managing VMware License components..
- `VMware.ImageBuilder` - The Image Builder snapin contains cmdlets for managing depots, image profiles, and VIBs.
- `VMware.DeployAutomation` - The Auto Deploy snapin contains cmdlets that provide an interface to VMware Auto Deploy for provisioning physical hosts with ESXi software. Listing the Cmdlets in a PowerCLI Snapin

To view the installed PowerShell snapins, use the `Get-PSSnapin` command. You can filter the vSphere PowerCLI snapins by using the `-Name` parameter and a wildcard:

```
Get-PSSnapin -Name *vmware*You can list all PowerCLI cmdlets by using the Get-VICommand cmdlet.
```

Another way for retrieving a list of all cmdlets in a snapin is to use the `Get-Command` cmdlet with the `-PSSnapin` parameter. For example, run:

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

The cmdlets in the specified PowerCLI snapin 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
```

## Getting Help About a Cmdlet

You can retrieve information about a specific cmdlet by using the `Get-Help` cmdlet or its alias - `help`. For example, to get help about the `Add-VMHost` cmdlet, type:

```
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
```

## Managing Default Server Connections with PowerCLI

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.



# vSphere PowerCLI System Requirements

---

# 2

Before installation, you must verify that vSphere PowerCLI is compatible with your system. This chapter discusses the following topics:

- [“PowerCLI Installation Prerequisites”](#) on page 11
- [“Windows PowerShell 1.0/2.0”](#) on page 11
- [“Supported VMware Environments”](#) on page 11

## PowerCLI Installation Prerequisites

To install vSphere PowerCLI, you must have installed the following software:

- .NET 2.0 SP1
- Windows PowerShell 1.0/2.0

## Supported Operating Systems

VMware vSphere PowerCLI 5.0 is supported on the 32-bit and 64-bit versions of the following Windows operating systems:

- Windows 7
- Windows Server 2008
- Windows Vista
- Windows XP Service Pack 2
- Windows 2003 Server Service Pack 2

---

**NOTE** The list of supported operating systems might not apply to some of the PowerCLI cmdlets that operate within guest operating systems.

---

## Supported VMware Environments

vSphere PowerCLI 5.0 is compatible with the following VMware environments:

- VMware ESXi 5.0
- vCenter Server 5.0
- VMware ESX 4.1/vCenter Server 4.1
- VMware ESXi 4.1
- VMware ESX 4.0 Update 2/vCenter Server 4.0 Update 2
- VMware ESX 4.0 Update 1/vCenter Server 4.0 Update 1

- VMware ESX 4.0i Update 1
- VMware ESX 3.5 Update 5
- VMware ESXi 3.5 Update 5
- VMware VirtualCenter 2.5 Update 6
- VMware ESX 3.0.3 Update 1

# Installing vSphere PowerCLI

---

You can install vSphere PowerCLI on all supported Windows operating systems. After you have installed the package on your machine, you can invoke vSphere PowerCLI cmdlets to connect to your ESX or vCenter Server system by specifying the appropriate connection parameters.

- [“Downloading the vSphere PowerCLI Binary”](#) on page 13
- [“Installing vSphere PowerCLI on Windows”](#) on page 13
- [“Uninstalling vSphere PowerCLI”](#) on page 14

## Downloading the vSphere PowerCLI Binary

You can download the latest vSphere PowerCLI version from the vSphere PowerCLI home page at <http://www.vmware.com/go/powercli>.

## Installing vSphere PowerCLI on Windows

Before installing vSphere PowerCLI, read [“vSphere PowerCLI System Requirements”](#) on page 11.

### To install vSphere PowerCLI using the Windows installer

- 1 Launch the installer by double-clicking the executable file you downloaded. See [“Downloading the vSphere PowerCLI Binary”](#) on page 13.
- 2 Click **Next** in the Welcome page to continue with the installation.
- 3 Read and accept the license agreement terms.
- 4 Click **Next**.

The Destination Folder page appears.

- 5 If you do not want to install the vSphere PowerCLI in the default location, click **Change** and select a different directory.

The default location is:

```
C:\Program Files\VMware\Infrastructure\vSphere PowerCLI
```

- 6 Click **Next**.
- 7 In the Ready to Install the Program page choose whether to create a desktop shortcut.
- 8 Click **Install** to proceed with the installation.
- 9 Click **Finish** to complete the installation process.

If the PowerShell Execution Policy on your machine is set incorrectly, a warning message appears before finalizing the vSphere PowerCLI installation. Ignore it and continue with the installation. To find more information about the PowerShell Execution Policy, see [“Setting the Properties to Support Remote Signing”](#) on page 14.

## Setting the Properties to Support Remote Signing

For security reasons, Windows PowerShell supports an execution policy feature. It determines whether scripts are allowed to run and whether they must be digitally signed. By default, the execution policy is set to **Restricted**, which is the most secure policy. If you want to run scripts or load configuration files, you can change the execution policy by using the `Set-ExecutionPolicy` cmdlet.

### To set the PowerShell execution policy

- 1 Choose Start > Programs > VMware > VMware vSphere PowerCLI > VMware vSphere PowerCLI. The vSphere PowerCLI console window opens.
- 2 In the vSphere PowerCLI console window, type:

```
Set-ExecutionPolicy RemoteSigned
```

To get more information about the execution policy and script digital signing in Windows PowerShell, use the following command:

```
Get-Help About_Signing
```

## Uninstalling vSphere PowerCLI

To uninstall the vSphere PowerCLI from your Windows system, use the **Add or Remove Programs** control panel.

# vSphere PowerCLI Usage Examples

---

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

The chapter discusses these topics:

- [“Basic Cmdlets Usage”](#) on page 15
- [“Advanced Cmdlets Usage”](#) on page 17
- [“API Access Cmdlets”](#) on page 31
- [“The Inventory Provider”](#) on page 33
- [“The Datastore Provider”](#) on page 34

## Basic Cmdlets Usage

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

### Display Help for a Cmdlet

You can get help for a specific cmdlet by supplying the `Get-Help` command in the vSphere PowerCLI console.

#### To get information on a cmdlet

- 1 Run the `Get-Help` cmdlet with the specific cmdlet name:

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

- 2 (opt) Retrieve more detailed information by using the `-full` parameter:

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

### Connect to a Server

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.

#### To connect to a local server

- 1 Run the `Connect-VIServer` cmdlet with the server name:

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

The cmdlet prompts for user credentials, as they are not passed as parameters.

---

**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
```

---

## 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 Host Operations

The following examples illustrate some basic operations with 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 Host standalone host:  

```
Add-VMHost -Name Host -Location (Get-Datacenter DC) -User root -Password pass
```

### To activate maintenance mode for a host

- 1 Save the Host host object as a variable:  

```
$host = Get-VMHost -Name Host
```
- 2 Retrieve the cluster to which Host belongs and save the cluster object as a variable:  

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

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

```
$updateHostTask = Set-VMHost -VMHost $host -State "Maintenance" -RunAsync
```

---

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

---

- 4 Retrieve and apply the recommendations generated by DRS:

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

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

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

## Advanced Cmdlets Usage

This section contains examples that 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 by 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 `Folder` in it:

```
$folder = Get-Folder -NoRecursion | New-Folder -Name Folder
```

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

- 3 Create a new datacenter called `DC` in the `Folder` folder:

```
New-Datacenter -Location $folder -Name DC
```

- 4 Create a folder called `Folder1` under `DC`:

```
Get-Datacenter DC | New-Folder -Name Folder1  
$folder1 = Get-Folder -Name Folder1
```

---

**NOTE** Search in PowerShell is not case-sensitive.

---

- 5 Create a new cluster `Cluster1` in the `Folder1` folder:

```
New-Cluster -Location $folder1 -Name Cluster1 -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:

```
$host1 = Add-VMHost -Name 10.23.112.345 -Location ( Get-Cluster Cluster1 )
```

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 Cluster1 ) -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 VM1 -VMHost $host1 -ResourcePool MyRP01 -DiskMB 4000 -MemoryMB 256
```

- 9 Create a virtual machine asynchronously:

```
$vmCreationTask = New-VM -Name VM2 -VMHost $host1 -ResourcePool MyRP01 -DiskMB 4000 -MemoryMB
256 -RunAsync
```

The `-RunAsync` parameter specifies that the command runs asynchronously. This means that in contrast to a synchronous operation, you do not have to wait for the process to complete before supplying the next command 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 VM2 virtual machine:

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

- 2 Create a template from the VM1 virtual machine:

```
New-Template -VM VM1 -Name VM1Template -Location (Get-Datacenter DC )
```

---

**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 VM3:

```
Get-Template VM1Template | Set-Template -ToVM -Name VM3
```

- 4 Create a template from the VM2 virtual machine:

```
New-Template -VM VM2 -Name VM2Template -Location (Get-Datacenter DC )
```

- 5 Convert this template to a virtual machine named VM4:

```
Get-Template VM2Template | Set-Template -ToVM -Name VM4
```

- 6 Convert the VM4 virtual machine to a template:

```
Set-VM -VM VM4 -ToTemplate -Name "VM4Template"
```

- 7 Create another template by cloning VM2Template:

```
Get-Template VM2Template | New-Template -Name VM3Template -VMHost $targetVMHost
```

- 8 Move the virtual machines into the MyRP01 resource pool:

```
Get-VM VM? | 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 VM and have one more symbol at the end. In this example, VM1, VM2, VM3, and VM4 are retrieved and moved into the MyRP01 resource pool.

- 9 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>
```

- 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-ResourcePool MyRP01 | Get-VM
foreach( $vm in $VMs ) { Set-VM -VM $vm -Snapshot -Snapshot 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 VM1 virtual machine.:  

```
Get-VMResourceConfiguration -VM VM1
```
- 3 Display the disk share of the VM1 virtual machine:  

```
Get-VMResourceConfiguration -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 MyVM1 virtual machine to 100:  

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

## List Hosts and Display Their Properties

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

### To list the available hosts and display their properties

- 1 List all hosts that are part of the datacenter named DC:  

```
Get-Datacenter DC | Get-VMHost | Format-Custom
```
- 2 Display the properties of the first host in the datacenter:  

```
Get-Datacenter DC | Get-VMHost | Select-Object -First 1 | Format-Custom
```
- 3 Display the Name and the OverallStatus properties of the hosts in the DC datacenter:  

```
Get-Datacenter DC | Get-VMHost | Get-View | Format-Table -Property Name, OverallStatus
-AutoSize
```
- 4 Display all hosts and their properties, and save the results to a file:  

```
Get-Datacenter DC | Get-VMHost | Format-Custom | Out-File -FilePath hosts.txt
```
- 5 List the hosts that are in maintenance mode and can be configured for VMotion operations:  

```
Get-VMHost -State maintenance | Get-View | Where-Object -FilterScript { $_.capability -ne
>null -and $_.capability.vmotionSupported }
```

## Change the Host Advanced Configuration Settings

This procedure shows how to configure hosts settings related to virtual machines migration.

### To change the host advanced configuration settings

- 1 Change the migration time-out for the ESXHost1 host.:  

```
Get-VMHost ESXHost1 | Set-VmHostAdvancedConfiguration -Name Migrate.NetTimeout -Value
( [system.int32] 10 )
```

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

```
Get-VMHost ESXHost1 | Set-VMHostAdvancedConfiguration -Name Migrate.MemChksum -Value
( [system.int32] 1 )
```
- 3 Get the ESXHost1 host migration settings:
 

```
$migrationSettings = Get-VMHost ESXHost1 | Get-VMHostAdvancedConfiguration -Name Migrate.*
```
- 4 Apply the migration settings to ESXHost2:
 

```
Set-VMHostAdvancedConfiguration -VMHost ESXHost2 -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 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 Retrieve the VM1 virtual machine and move it to the datastore named DS2:
 

```
Get-VM VM1 | Move-VM -Datastore DS2
```

---

**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.

---

## Manage Host Profiles

This scenario illustrates how to get use of 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 Server 4.0 or later. Earlier releases do not support host profiles.
- 2 Get the host named Host1 and store it in the \$host variable:
 

```
$host = Get-VMHost Host1
```
- 3 Create a profile based on the Host1 host:
 

```
New-VMHostProfile -Name MyHostProfile01 -Description "This is my test profile based on
Host1." -ReferenceHost $host
```
- 4 Get the newly created host profile:
 

```
$hp1 = Get-VMHostProfile -Name MyHostProfile01
```
- 5 Change the description of the HostProfile1 host profile:
 

```
Set-VMHostProfile -Profile $hp1 -Description "This is my old test host profile based on
Host1."
```
- 6 Get the Host2 host, on which to apply the testProfile host profile:
 

```
$host2 = Get-VMHost Host2
```

- 7 Associate the MyHost02 host with the HosProfile1 host profile:  

```
Set-VMHost -VMHost $host2 -Profile $hp1
```
- 8 Test if the Host2 host is compliant with the HostProfile1 profile:  

```
Test-VMHostProfileCompliance -VMHost $host2
```

The output of this command contains the host's in-compliant settings, if any.
- 9 Apply the profile to the Host2 host:  

```
$neededVariables = Apply-VMHostProfile -Entity $host2 -Profile $hp1 -Confirm:$false
```

The `$neededVariables` variable contains the names of all required variables and their default or current values, as returned by the server. Otherwise, the `$neededVariables` variable contains the name of the host on which the profile has been applied.
- 10 Export the HostProfile1 profile to a file:  

```
Export-VMHostProfile -FilePath export.prf -Profile $hp1 -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 HostProfile1, 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 manage statistics and statistics intervals

- 1 Extend the storage time of the past day statistics interval:  

```
Set-StatInterval -Interval "past day" -StorageTimeSecs 700000
```
- 2 List the available memory metric types for the Cluster cluster:  

```
$cluster = Get-Cluster Cluster1
$statTypes = Get-StatType -Entity $cluster -Interval "past day" -Name mem.*
```
- 3 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 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 VApp on the specified host:  
`New-VApp -Name VApp -CpuLimitMhz 4000 -CpuReservationMhz 1000 -Location ( Get-VMHost Host1 )`
- 2 Start the new virtual appliance:  
`Start-VApp VApp`

### To change the properties of a virtual appliance

- 1 Retrieve and stop the VApp virtual appliance:  
`Get-VApp VApp | Stop-VApp -Confirm:$false`
- 2 Change the name and memory reservation of the VApp virtual appliance:  
`Get-VApp VApp | 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 Host1 ) -Datastore ( Get-Datastore -VMHost MyHost01 -Name Storage2 )`
- 2 Remove the WebApp appliance and delete it permanently:  
`Remove-VApp WebApp -DeletePermanently -Confirm:$false`

## Manage Guest Networks

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

### To retrieve and configure a network interface

- 1 Retrieve the guest network interface of the VM1 virtual machine:  
`$vm1 = Get-VM -Name VM1  
 $interface = Get-VMGuestNetworkInterface -VM $vm1 -HostUser root -HostPassword pass1  
 -GuestUser user -GuestPassword pass2`
- 2 Retrieve the network interface of a guest OS:  
`$guest = Get-VMGuest $vm1  
 $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 and configure a guest route**

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

```
Get-VMGuestRoute -VM $vm1 -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 Configure the guest route:

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

## 5 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:

```
$host = Get-VMHost ESXHost1
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 $host -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 $host -CanonicalName ($iscsiHba.Device + "*" ) |
Get-ScsiLunPath
```

## 6 Create the new storage:

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

## Manage PCI and SCSI Passthrough 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 ESXHost host:

```
$host = Get-VMHost ESXHost
Get-PassthroughDevice -VMHost $host -Type Pci
```

- 2 Retrieve the SCSI passthrough devices of the VM virtual machine:

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

- 3 Add a SCSI passthrough device to the VM virtual machine:

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

- 4 Remove all passthrough devices of the VM 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 available 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 -Spec 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 -Spec Spec | Set-OSCustomizationNicMapping -IpMode UseStaticIP -IpAddress 172.16.1.30 -SubnetMask 255.255.255.0 -DefaultGateway 172.16.1.1 -Dns 172.16.1
```

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 -Spec Spec | Set-OSCustomizationNicMapping -IpMode UseStaticIP -IpAddress 172.16.1.30 -SubnetMask 255.255.255.0 -DefaultGateway 172.16.1.1 -Dns 172.16.1.1
```
- 4 Create a new virtual machine named VM1 from a template and apply the static IP settings:  

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

### 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 -Spec Spec -IpMode UseStaticIP -IpAddress 172.16.1.30 -SubnetMask 255.255.255.0 -DefaultGateway 172.16.1.1 -Dns 172.16.1.1
```

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

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

## Manage vSphere Permissions, Roles, and Privileges

vSphere permissions determine your level of access to vCenter Server, ESX, and ESXi hosts. Privileges define individual rights to perform specific actions and access object properties. Roles are predefined sets of privileges. With PowerCLI, you can automate managing vSphere permissions, roles, and privileges.

### To create a role and assign permissions to a user

- 1 Retrieve the privileges of the read-only role:
 

```
$readOnlyPrivileges = Get-VIPrivilege -Role readonly
```
- 2 Create a new role with custom privileges:
 

```
$role1 = New-VIRole -Privilege $readOnlyPrivileges -Name Role1
```
- 3 Add "Power On" privileges to the new role:
 

```
$powerOnPrivileges = Get-VIPrivilege -Name "Power On"
$role1 = Set-VIRole -Role $role1 -AddPrivilege $powerOnPrivileges
```
- 4 Create a permission, apply it to a vSphere root object, and propagate them across the hierarchy:
 

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

---

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

---

- 5 Update the role of the new permission with the custom role:
 

```
$permission1 = Set-VIPermission -Permission $permission1 -Role $role1
```
- 6 Remove the new permission:
 

```
Remove-VIPermission $permission
```
- 7 Remove the custom role:
 

```
Remove-VIRole Role1
```

## Manage Alarms

With vSphere PowerCLI, you can manage alarms definitions, 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 VIMServer Object

You can create and modify advanced settings for Cluster and VIMServer 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 ESXCLI with PowerCLI

PowerCLI provides two approaches for working with ESXCLI:

- Through the `Get-ESXcli` cmdlet, which provides direct access to the ESXCLI namespaces, applications, and commands.
- Through `.NET` methods, which you use to create managed objects that correspond to specific ESXCLI applications. To access the ESXCLI, you can invoke methods on these managed objects.

---

**NOTE** To invoke a method of an ESXCLI object, you must provide values to all parameters. If you want to omit a given parameter, pass `$null`.

---

There are two `.NET` methods you can use for retrieving information of ESXCLI applications:

### To work with ESXCLI by using the `Get-ESXcli` cmdlet

- 1 Retrieve an ESXCLI instance:

```
$myEsxcli = Get-Esxcli
```

- 2 View the properties of an ESXCLI instance:

```
$myEsxcli
```

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

```
$myEsxcli.nmp
```

- 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
```
- 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 ESXCLI using .NET methods

- 1 Retrieve all managed object instances:  

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

The `QueryMoInstances` method queries all managed object instances that correspond to ESXCLI applications in the high-level binding, both predefined and third-party.
- 2 Retrieve information for a specific managed object type:  

```
$nmpDeviceInfo = $myEsxCli.TypeManager.QueryTypeInfo("vim.ExxCLI.nmp.device")
```

The `QueryTypeInfo` method provides full information about a specific managed object type and its corresponding ESXCLI application.
- 3 Retrieve the methods available for a specific managed object type:  

```
$typeInfo.managedTypeInfo[0].method
```
- 4 Retrieve an instance of a managed object:  

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

The `CreateDynamicManagedObject` method is used to obtain instances of ESXCLI managed objects.
- 5 Retrieve the methods of a managed object instance by using the `InvokeOperation` method:  

```
$nmpManagedObjectInstance.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.ExxCLI.nmp.device "})
```

## Use esxtop 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
```

## API Access Cmdlets

The vSphere PowerCLI list of cmdlets includes two API 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*.

Using the API 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 VM2 virtual machine using a filter by name and populates the view object.

```
$vm2 = Get-View -ViewType VirtualMachine -Filter @{"Name" = "VM2"}
$hostView = Get-View -Id $vm2.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 VM2 virtual machine using a filter by name:

```
$vm2 = Get-View -ViewType VirtualMachine -Filter @{"Name" = "VM2"}
$hostView = Get-View -Id $vm2.Runtime.Host
```

- 2 Print the current power state:

```
$vm2.Runtime.PowerState
```

- 3 Change the power state of the virtual machine:

```
If ($vm2.Runtime.PowerState -ne "PoweredOn") {
    $vm.PowerOnVM($vm2.Runtime.Host)
} else {
    $vm2.PowerOffVM()
}
```

- 4 Print the value of `$vm2` power state (the power state is still not updated because the virtual machine property values are not updated automatically):

```
$vm2.Runtime.PowerState
```

- 5 Update the view:

```
$vm2.UpdateViewData()
```

- 6 Show the actual power state of the virtual machine:

```
$vm2.Runtime.PowerState
```

## Mixed Usage of vSphere PowerCLI and API Access Cmdlets

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

### To reboot a host

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

```
$hostView = Get-VMHost -Name Host | Get-View
```

- 2 Call the `RebootHost` 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 VM2 virtual machine, shut down it, and pass the result to the `Get-View` cmdlet to retrieve the virtual machine view object:

```
$vmView = Get-VM VM2 | Stop-VM | Get-View
```

- 2 Create a `VirtualMachineConfigSpec` object to modify the virtual machine CPU levels and call the `ReconfigVM` method of the virtual machine view managed object.

```
$spec = New-Object VMware.Vim.VirtualMachineConfigSpec;
$spec.CPUAllocation = 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.

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 an 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 default datastore drives

- 1 Access the `vmstore` drive:

```
cd vmstore:
```

- 2 Access the `vmstores` drives:

```
cd vmstores:
```

- 3 List the drive content:

```
dir
```

**To create a new custom datastore drive**

- 1 Get a datastore by its name and assign it to the `$datastore` variable:

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

- 2 Create a new PowerShell drive `ds:` in `$datastore`:

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

---

**NOTE** You can use the `New-PSDrive` cmdlet 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
```

or

```
New-PSDrive -Name ds -PSProvider VimDatastore -Root  
vmstores:\192.168.10.112@443\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
```

