PowerCLI Essentials

Have you ever wished you could automatically get a report with all the relevant information about your VMware environments in exactly the format you want? Or that you could automate a crucial task that needs to be performed on a regular basis? Powerful Command-line Interface (PowerCLI) scripts do all these things and much more for VMware environments. PowerCLI is a command-line interface tool used to automate VMware vSphere environments.

This book will show you the intricacies of PowerCLI through real-life examples so that you can discover the art of PowerCLI scripting. At the start, you will be taught how to download and install PowerCLI and will learn about its different versions. Moving on, you will be introduced to the GUI of PowerCLI and will find out how to develop single-line scripts to duplicate running tasks, produce simple reports, and simplify administration. Next, you will learn about the methods available to get information remotely. Towards the end, you will be taught how to set up Orchestrator and build workflows in PowerShell with the update manager and SRM scripts.

Who this book is written for

PowerCLI Essentials is aimed at virtualization professionals and system administrators who want to discover and learn about the automation techniques associated with PowerCLI for complex virtual environments.

What you will learn from this book

- Download and install PowerCLI and its basics as well as the basics of PowerShell
- Enhance your scripting experience
- Build longer scripts and simpler reports
- Relate a task in VMware administration to a PowerCLI script
- discover methods to acquire and change information remotely
- Set up Orchestrator to manage your workflow

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'An Introduction to Essential Administration with PowerCLI'
- A synopsis of the book’s content
- More information on PowerCLI Essentials
About the Author

Chris Halverson is a senior consultant for VMware in the Professional Services Organization in Canada. He specializes in the full Software-Defined Data Center (SDDC) stack, architecting, designing, and deploying customer solutions. He has been active in the VMware community as a VMUG leader for the past 4 years and prides himself as an active participant in the local IT market over the past 17 years. Previous roles have allowed him to work on enterprise architecture bringing process and rigor to the administration aspect of the position and automation that made the job smarter. There is even an aspect where "I replaced myself with a small script" has been heard from him when discussing the former roles.

When Chris is not working on building the virtual community, he shares his time and energy with his tremendous wife, three boys, and one princess. Excited to spend time watching and helping them figure out life for themselves, he encourages them in their sports, through their schooling, and through their own walk of faith.

Over the past few years, Chris has also been able to achieve a dream as a second degree black belt in Tae Kwon Do, crossing off one of those bucket list items.

Chris has also acted as a book reviewer.
Preface

VMware PowerCLI is one of the most utilized command-line interfaces for a VMware vSphere System Administrator. Covering more than 480 different functions of a vSphere system, PowerCLI and PowerShell have become one of the staples of an automation enabler in this space. Taking the viewpoint of an administrator with some experience, Essential VMware Administration with PowerCLI introduces the idea and concept of taking the beginning steps toward developing one-line commands into multi-line scripts that can be used not only by the reader but also by others within their organization.

This book is designed with the mindset of think first, design second, script next, and test last. It covers getting the tool and integrating it with other products in the VMware stack and attempts to build on the knowledge outlined from the chapter before.

What this book covers

Chapter 1, An Introduction to Essential Administration with PowerCLI, sets the stage through discussing how to get PowerCLI, what the difference is between PowerShell and PowerCLI, discusses its version history, and provides a starting point with its installation. This chapter is a means to get you up and running with the right version and the best tools for the job.

Chapter 2, Comparing Point and Click Administration to PowerCLI and Scripting, looks through the eye of an administrator, helping to redefine the typical point and click doldrums and compare them to the exciting and provocative world of scripting.

Chapter 3, Enhancing the Scripting Experience, takes the previous chapter and builds upon it. This chapter will enhance the experience through better practices, help a team build a repository, and make the code reusable.
Chapter 4, Windows Administration within VMware Administration, starts with the preparation of a DevOps practice and the roles around it. This programmatically helps bridge the gap for becoming operationally transformed and influences how an organization can build a private cloud type of environment. We will use this mindset to build and provision a Windows script host, and run PowerCLI and Windows-based PowerShell in the same script.

Chapter 5, Workflows and vRealize Orchestrator, introduces the vRealize Orchestrator product, how workflows are developed, and where to use them. The chapter will provide a walkthrough of the installation of vRealize Orchestrator and where it fits in the environment.

Chapter 6, Running Workflows with Other VMware Products, discusses other VMware products such as NSX, Orchestrator, vRealize Operations Manager, Site Recovery Manager, and VSAN. It takes each technology and product, explains the product, where it fits, and then, finally, how PowerCLI can integrate with them.
An Introduction to Essential Administration with PowerCLI

It's 4 pm on a Friday afternoon; you are packing up for a weekend camping trip with the family when your boss walks up to your desk with that I have an immediate task that I need to hand off before I leave for the weekend look on his face. You say, "Hey sir, I am just heading out camping this weekend and I need to beat the traffic, gotta go!" But you are just not that lucky, he quickly replies that his boss has an important meeting and needs some numbers before a Monday morning meeting, and adds, "I don't have the skills in the infrastructure to get this information and I am desperate to get this tonight. Besides, we will all look like heroes if we do this." "Or at least you will", you mutter under your breath.

Is that a normal scenario in your office? It was my role for a number of clients over the last few years until I started predicting the future and writing PowerCLI scripts. This book is a collection of administration experiences for managing a virtual infrastructure and for incorporating the vast skill of DevOps and scripting.

This chapter deals with the getting started mentality, experiencing taking the System Administrator role to the next level, and making the endless job of system operations a bite-sized effort.

In this chapter, you will learn about:

- PowerShell versions
- PowerCLI versions
- Getting PowerShell and PowerCLI
- Setting up the environment
- PowerShell basics
Why this book, and why now?
System Administration has always been somewhat of a thankless job, and as an admin, making things easier and quicker has always been the end goal. Understanding the underpinnings of PowerShell and PowerCLI will help with mundane daily tasks associated with regular administration. This chapter will help you get up to speed using the command line and allow shortening of time for you to be comfortable and proficient with it.

Understanding PowerShell versions
PowerShell, or PoSH as it is also known, is a shell scripting language that Microsoft created to replace batch and VBScript:

- Version 1 was introduced in 2006, was available to be downloaded for Windows XP, Vista, or Windows Server 2003, and was an optional component of Windows Server 2008
- Version 2 was integrated into Windows 7 and Windows Server 2008 R2 server and was available for download for earlier versions Windows XP SP3, Vista SP1, Server 2003 SP2, and Server 2008
- Version 3 was included in the base version of Windows 8 and Windows Server 2012 and could be installed on Windows 7 SP1 and 2008 R2 SP1
- Version 4 was integrated into Windows 8.1 and Windows Server 2012 R2 and could be installed on Windows 7 SP1, Windows Server 2008 R2 SP1, and Windows Server 2012
- Version 5 was still public beta as of the time of writing this book and is only installable on Windows 8.1, Server 2012, and 2012 R2

The basic structure of PowerShell
PoSH, based on the .NET framework, establishes a structure for programming/scripting into a human-readable format. Previous Windows scripting languages were, for the most part, cryptic and it wasn't always easy to pass a script to another administrator without the other administrator having extensive experience with the language.

Originally, the Command Line Interpreter or CLI (cmd.exe or command.com) allowed the running of specific executables or EXEs through a recipe of other known commands into a batch (.bat) or command script (.cmd). These scripts were used to build a setup for users (such as login scripts) or to launch a customized experience for an application startup or application installation.
The `cscript.exe` command enabled the inclusion of JavaScript or VBScript in the recipe and gave an administrator much more power to do far more complex and useful tools. The script was written with an external program (`notepad.exe`, for example) and then the `cscript.exe` command would have to precede the script.

After Microsoft's previous attempts at power and user friendliness, PoSH follows a basic human readable format of Verb-Noun for commands to be written. Get-Service, for example, uses the verb Get and the noun Service to specify the desire to get what services are running. Switches such as DisplayName can add to the clarity of the command and shorten the output:

```
Get-Service -DisplayName "VMware*"
```

<table>
<thead>
<tr>
<th>Status</th>
<th>Name</th>
<th>DisplayName</th>
</tr>
</thead>
<tbody>
<tr>
<td>Running</td>
<td>VMTools</td>
<td>VMware Tools</td>
</tr>
<tr>
<td>Running</td>
<td>VMUSBArbService</td>
<td>VMware USB Arbitration Service</td>
</tr>
<tr>
<td>Stopped</td>
<td>vmvss</td>
<td>VMware Snapshot Provider</td>
</tr>
<tr>
<td>Running</td>
<td>vmware-view-usbd</td>
<td>VMware View USB</td>
</tr>
<tr>
<td>Running</td>
<td>wsnm</td>
<td>VMware View Client</td>
</tr>
</tbody>
</table>

Although, not all commands contain switches such as get-service, clarification of this topic will be discussed later in this chapter.

**Why is version understanding important?**

As a typical System Administrator of multiple systems, it is likely that PoSH has been used due to newer Microsoft systems' (Exchange, SQL, or basic Windows admin functions) requirements for administration. With every version increment, additional features get added to the product, such as syntax simplification and additional cmdlets (pronounced command-let); this provides more functionality and script simplification possibilities to ensure certain version baselines in the environment allow single source scripts and common programming specifications.

To simplify, using the newest version available allows more functionality and ease of use. To use an analogy of buying a car, getting a newer car generally provides more features, performance, and gadgets. The manufacturer learns from the shortcomings of previous years and makes a better product.
Many companies use a plethora of Windows servers and Windows desktops to support their business's attempt at maintaining a standard platform for operating, deployment, and implementation to reduce the amount of work. In most environments, there are a variety of server versions in place, from Windows Server 2003 to Windows Server 2012 R2, all supporting different applications and being supported by hundreds of vendors. Each iteration of the Server product may, and typically does, have a variety of components that need to be maintained, such as Windows Patches, and in the case of VMware Infrastructure, VMware Tools and Hardware versions (Drivers).

As seen in the Understanding the PowerShell versions section, there are five listed versions available. Of these, version 2 (v2) and version 3 (v3) apply the greatest variance amongst all of the versions. Programming scripts using the v3 syntax will produce errors in v2, whereas v2 syntax will work in v3. Microsoft provides a much simpler experience using the newer versions, and all scripts within this book will be written using v3 unless otherwise stated.

Thankfully, there is a simple way to test what version of the Windows OS is running. To find out the current version of PoSH, type this command:

```powershell
$PSVersionTable.PSVersion.Major
```

It will produce a very brief output, as follows:

3

**v2 or v3 – what's the difference?**

Is there a difference? There are quite a few variations between the two, but a single key differentiator shown here highlights it.

A basic one-line v2 command is as follows:

```powershell
Get-VM | where {$_.Name –eq "Server"}
```

The same command using the v3 syntax is as follows:

```powershell
Get-VM | where Name -eq Server
```

At first glance, the differences are apparent. v2 demands additional brackets, quotes and variable definitions to make the command work, whereas, v3 removes the cryptic demands and simplifies the command string. With v3, it is backwards compatible, so PoSH v2 commands will also work within the v3 command structure.
One of the significant additions to v3 is the inclusion of workflows. Workflows allow the running of multiple scripts in a cascading fashion. This provides a means to develop the ability to write recipes with many parts into a single consistent plan. This can be extremely powerful and useful and is discussed in later chapters.

So the question that begs to be asked is: Why not develop all code within the context of v2? As seen earlier, PoSH v3 simplifies the development of the code base, so why not use this? This is the dilemma of every site or company embarking on a path to automation, whether to choose a path of conformity versus the added administration of functionality. As an Administrator, most environments that are worked on will include a variety of different systems, from 2003 Server, 2008 R2 Server, and 2012 Server, which include different versions of PowerShell. One of the first automation scripts that will be shown will be a sample bit of code to look through the environment and find PowerShell versions and OSs with the installed version of PowerShell.

Because this book deals with workflows and automation in the latter chapters, all code will be written in v3. Version 4 (v4) provides many additional cmdlets but doesn't provide a significant interface change to warrant the inclusion of it. If there is a v4 specific command used, the text will ensure the reader's understanding of such.

### Installing PowerShell v3 on a Windows 7 or Windows 2008 R2 machine

On a standard Windows 7 computer, PoSH v2 comes standard as referred to earlier in this chapter. If running the PoSH command, as shown in the last section, reveals that the system is indeed running v2, there is a simple process that can be gathered from this Microsoft URL:


<table>
<thead>
<tr>
<th>File Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows6.1-KB2596143-x64.msu</td>
<td>15.8 MB</td>
</tr>
<tr>
<td>Windows6.0-KB2596146-x64.msu</td>
<td>14.4 MB</td>
</tr>
<tr>
<td>Windows6.0-KB2596146-x86.msu</td>
<td>10.5 MB</td>
</tr>
<tr>
<td>Windows6.1-KB2596143-x86.msu</td>
<td>11.7 MB</td>
</tr>
<tr>
<td>WMF 3 Release Notes.docx</td>
<td>53 KB</td>
</tr>
</tbody>
</table>

Listing of available files
The one point it doesn't list is the fact that because PoSH v3 uses the .NET framework for the majority of its functions, .NET version 4 at minimum must be installed on the system as well. Both components can be pushed through any patch management framework that may be in use.

Understanding PowerCLI

VMware created an Application Program Interface (API) that provides a way for third-party vendors and developers to have other software products register and control the vSphere environment. PowerCLI is a group of functions, written in PowerShell, that convert the cryptic commands of the API into a human-readable format to simplify the scriptwriter's job. For example, one of the first commands you write is `Connect-VIServer`. This single command is programmed as a group of API calls to initiate a connection to the vCenter server, pass user credentials, and open a network socket to the server. Understanding all of the pieces of how to connect to a vCenter server is now irrelevant and the single command is used.

Each edition of vSphere brought with it a long list of features and options permitting the user to do more and more with the Virtual Environment, and with that, more and more cmdlets needed to be written for PowerCLI to enable better control of the environment. One of the best improvements was in vSphere version 4 and was the creation of Distributed Virtual Switches (DVS). PowerCLI had no means to create, manage, or delete such switches, and on numerous occasions, the API had to be used to perform the function. So, with each additional PowerCLI version, the developers of the toolset brought it closer and closer to feature parity. As of version 6 release 1 (R1), the two products are closer than they have ever been. So, having the latest PowerCLI version is optimal for administering the environment as the newest PowerCLI version is still very capable of controlling older versions of vSphere.

Getting PowerCLI versions

To get the PowerCLI version, either watch as you open the interface or type this command:

```
$(Get-PowerCLIVersion).Major
```

The preceding command line will produce an output as follows:

6

Otherwise, you can type:

```
$(Get-PowerCLIVersion).UserFriendlyVersion
```

--- [6] ---
Now, this command will produce an output as follows:

```plaintext
VMware vSphere PowerCLI 6.0 Release 1 build 2548067
```

As shown, the output can be simple or human readable. Either way works for general examination of the command, but both can be used for different purposes. For example, if there is a need to run a specific command that is only available in version 6, then a query of this command may be ideal to stop the display of an error.

### The PowerCLI change log

These are the highlights of PowerCLI right from version 1.0.1 (when it was named VI Toolkit) up to the latest version 6.0:

- **VI Toolkit 1.0.1 (released in September 2008):** The VI Toolkit was a utility that provided the administrator with simple commands to gather and check the environment for simple information. Even though the majority of the tools still commanded a grasp of the API, the commands provided a huge advantage for conducting repeatable tasks.

- **VI Toolkit 1.5 (released in January 2009):** The release of 1.5 added quite a few new cmdlets, fixed a few bugs in the older version, and notably added `Set Verb` to the cmdlets. The key differentiator was now being able to do more than read and report.

  This version also sparked a number of key players in the community such as Virt-Al, LucD, and ESloof to post numerous blog posts and tons of VMware Community posts on how to do stuff with PowerShell and VMware. This timeframe also sparked the creation of the Virtualization EcoShell Interface and caused the PowerGUI communities to pop up, adding much more credibility to the line.

- **PowerCLI 4.0 (released in May 2009):** Here is the first of the PowerCLI products that aligned with the vSphere release. The highlights were around the Host Profiles and Host security and introduced the `Invoke-vmscript` function.

- **PowerCLI 4.0.1 (released on November 19, 2009):** As this was a minor release, it added a few more cmdlets and fixed some bugs from the previous build. It started the OS customization portion of the command set, and brought about Power state controls for the Hosts.

- **PowerCLI 4.1 (released on July 13, 2010):** This version started the tradition of being released shortly before VMworld to help coincide with the release of the vSphere version. There were very few new cmdlets, but numerous enhancements and bug fixes for previous versions.
• **PowerCLI 4.1 U1 (released on December 1, 2010):** The largest addition to this release was the ability to run ESXCLI (usually reserved for the command line interface on the host, the vSphere CLI command set, or the VMware Management Appliance). This added a huge subset of commands to the Administrator's tool chest with regard to scripts.

Another feature was the ability to control Distributed vSwitches, through the scripts. This allowed the ability to build an environment through a script by initiating the host profiles, as listed earlier, and now build a distributed vSwitch and provision VMs network interface cards to said dvSwitch.

• **PowerCLI 5.0 (released in August 24, 2011):** The 5.0 release introduced feature parity to the Storage I/O control set and was available for download at the same time as the 5.0 vSphere release. Most of this release added functionality to what was already there and fixed a few bugs in the `Invoke-VMScript` cmdlet.

• **PowerCLI 5.0.1 (released on January 9, 2012):** This release added vCloud support to the stack.

• **PowerCLI 5.1 R1 (released on September 10, 2012):** Shortly after VMworld, when vSphere 5.1 was released, v 5.1 was available. This release brought support for Storage Distributed Resource Scheduling (SDRS), Storage vMotion, and added numerous enhancements to the vCloud Director product. This version also brought a new modification to the tool; it deprecated commands for the first time in the product line. This was significant because a script using the command `Set-VMHostAdvancedConfiguration` would have an error because the command had changed.

• **PowerCLI 5.1 R2 (released on February 11, 2013):** Further dvSwitch networking support was added to the cmdlet list, including the ability to export the configuration for product migration. There were a number of vCloud enhancements and additional deprecation of cmdlets.

• **PowerCLI 5.5 R1 (released on September 19, 2013):** Version 5.5 brought a few more dvSwitch cmdlets, introduced tags, allowed management of VSAN, and introduced additional component pieces to the install.

• **PowerCLI 5.5 R2 (released on March 11, 2014):** Release 2 of the 5.5 version added support for PowerShell v4.0, enhanced the tag cmdlets, and at long last, provided 64-bit capability for OS customization. This version also exposed an ability to communicate with a **Site Recovery Manager (SRM).**
• **PowerCLI 5.8 R1 (released on September 9, 2014):** 2014 was the first VMworld that didn't release a new vSphere version. The enhancement that year was in the upgrade to the suite of products. There was an incremental version numbering of PowerCLI, SRM, vCenter Operations Manager, Infrastructure Navigator, and other related products to version 5.8. The list of enhancements for this version of PowerCLI was also one of the smallest, but the tool had its startup time increased and an enhancement to the error reporting within the tool.

• **PowerCLI 6.0 (released on March 12, 2015):** This release coincided with the release of vSphere 6 and provided support for VVOLs and VSAN enhancements and moved some of the cmdlets to be supported as modules instead of snap-ins, as previously performed. The change log discusses re-examining the script code to support this change. It is a fundamental change to the PowerCLI product line, and the scripts in this book will be reflective of this version.

The version listing, at first glance, may seem irrelevant, but as the product has changed and grown, the script methodology and syntax has changed. From personal experience in numerous client sites, there is a mixture of different versions installed at different times by different administrators. This inconsistency, like the PoSH v2 and v3 section previously discussed, can and will cause scripting errors within the automation of the site.

Further reading of the changes can be done at:
https://www.vmware.com/support/developer/PowerCLI/changelog.html

### Getting PowerCLI
To get PowerCLI, there are a few places to start. Google always brings the browser first to the VMware blog site, which is all right as that usually has the correct download link. However, the link is provided later for all the versions of PowerCLI.

It is always better to download direct from VMware. There are other links that can be used to download from, but because it is a programmable interface directly within a production virtual environment, it is ideal to get it from the source.
An Introduction to Essential Administration with PowerCLI

Pre-requisites before getting PowerCLI
Having a VMware ID will be required to be able to download PowerCLI.

Downloading PowerCLI
To download the latest version, perform the following steps:

1. Go to https://www.vmware.com/support/developer/PowerCLI/, and you will see the following window:

   ![Image](vSphere PowerCLI Documentation)

   vSphere PowerCLI Documentation
   Community | Technical Papers | Knowledge Base | SDK Support
   Visit the vSphere 6.0 Documentation Center to learn more about VMware products.

   Release 6.0 Release 1
   Released 12 MAR 2015 | Download | Change Log
   Documentation Resources:
   Release Notes
   User's Guide (pdf)
   Cmdlet Reference | Previous HTML Interface

2. Click on the download link shown in the screenshot and the web page takes the browser to the login page.

3. Enter the VMware ID and the password, and the PowerCLI executable will be downloaded.

Installing PowerCLI
The typical installation of PowerCLI, whether it is 4.0 or 6.0, includes the VIX API (discussed earlier), and newer versions include the Remote Console as well. The remote console provides a remote console to any available VM running in the virtual infrastructure.
The installation shown in the following diagram is PowerCLI Version 6.0 on a Windows Server 2008 R2 platform. Let's follow the installation process:

1. When launching the installation, the shown interface seems a little strange to typical installs:

![VMware vSphere PowerCLI - InstallShield Wizard](image)

2. However, the installer runs separate installations of the components, one for the required **.NET Framework 4.5 Full**, and the PowerShell v3 and remote Console Plug-in 5.1 after. First, the installer downloads the file as follows:

![Downloading: dotnetfx45_full_x86_x64.exe](image)

3. Once the file is downloaded, the installer begins the installation process:
4. Then, the installer runs the installation file:

Once the .NET framework is installed, PowerShell v3 is installed too.
Don't be surprised if the installation fails at this point. When running through this procedure, the following error popped up:

This popup demands that the Windows Management Framework version 3.0 be installed. (Installing WMF 3.0 is shown in the Installing PowerShell v3 section earlier in the chapter.) The installation will exit with an error if this isn't done. Allow the installer to exit, install WMF 3.0, and rerun the installer.

5. Once complete, the VMware Remote Console Plug-in is installed. The remote console displays where it will install the web browser plugins on the system.
6. The Remote Console is installed and the PowerCLI snap-in component is started. A warning about setting the RemoteSigned execution policy is displayed, as shown in the following screenshot (this will be further explained in the next section):

7. Then, finally, the PowerCLI installation begins after clicking on Continue, which brings you the Welcome screen. Click on Next to continue to the installation wizard for PowerCLI:
8. This will bring you to the **VMware Patents** window. Click on **Next** to continue. This will bring you to the **License Agreement** window.

9. Accept the License Agreement, choose the components, set the path of installation, and run the installation process. Finally, you will reach the setup process as shown in the following screenshot:

On older versions, the VIX API is shown in the installer, but is not shown in the newer versions.
The final list of applications from a base Windows 2008 R2 server installation looks like this:

<table>
<thead>
<tr>
<th>Application</th>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft .NET Framework 4.5</td>
<td></td>
</tr>
<tr>
<td>Microsoft Visual C++ 2008 Redistributable - x64</td>
<td>9.0.30729.6161</td>
</tr>
<tr>
<td>Microsoft Visual C++ 2008 Redistributable - x86</td>
<td>9.0.30729.4148</td>
</tr>
<tr>
<td>VMware Remote Console Plug-in 5.1</td>
<td></td>
</tr>
<tr>
<td>VMware Tools</td>
<td></td>
</tr>
<tr>
<td>VMware vSphere PowerCLI</td>
<td></td>
</tr>
</tbody>
</table>

**Getting help**

PoSH v2 installs natively and includes all the help files as installed on a system. Version 3 changes that dichotomy in to forcing the user to download or access the help online. On PoSH v3, the `Update-Help` command should be run to ensure all help files are downloaded onto the system where the scripts will to be written. This helps reduce the file space usage on systems not being used for script development. If help is needed on systems where the `Update-Help` command has not been run, a `Get-Help <command> -online` command can open an associated web page with the appropriate command syntax shown.

Once the help files are downloaded, a simple query such as `Get-Help Get-VM` will produce a limited explanation of the command with the name of the command, synopsis, syntax, description, related links (similar commands), and remarks. The remarks section will remain the same for every help request, as it outlines alternate switches in the getting help section.

```
NAME
  Get-VM

SYNOPSIS
```

The `Get-Help` command has an alias of `Help` that can be also be used for short hand. For example, `Get-Help Get-VM` can be written as `Help Get-VM`.

For most help requests, the scriptwriter either needs an understanding of the syntax or to see examples of the command in use. Using the `-examples` switch displays the example code built into the help files directly. So typing `Get-Help Get-VM -examples` displays the following output:

```
NAME
  Get-VM

SYNOPSIS
```
This cmdlet retrieves the virtual machines on a vCenter Server system.

-------------- Example 1 --------------

C:\PS>Get-VM -Name MyVM*

Retrieves all virtual machines whose names starting with "MyVM".

-------------- Example 2 --------------

C:\PS>$myDatastore = Get-Datastore -Name "MyDatastore"
Get-VM -Datastore $myDatastore

Retrieves all virtual machines that reside on the specified datastore.

-------------- Example 3 --------------

C:\PS>$myDatacenter = Get-Datacenter -Name "MyDatacenter"
Get-VM -Location $myDatacenter

Retrieves all virtual machines in the specified datacenter.

-------------- Example 4 --------------

C:\PS>$myVDSwitch = Get-VDSwitch -Name "MyVDSwitch"
Get-VM -DistributedSwitch $myVDSwitch

Retrieves all virtual machines connected to the specified distributed switch.

Get-Help is an extremely useful portion of the script writing process and for understanding how to get information, but what if the command to be used to get, say, VM information is not known? Running the Get-Help VM command will show every cmdlet available with VM in the name with a brief synopsis of each one.
Another option is to use a search engine to locate an example of the certain task or information needed. For example, searching for "PowerShell get VM guest information" produces a number of results from the Get-VMGuest command to utilizing Get-View, and even discusses some methods to get this information from a Hyper-V server. Where too much information is daunting, understanding the more specific information to be searched for will mean the more relevant information should be returned.

Lastly, never be afraid to ask. VMware communities were built on this premise and some of the best scripters asked questions at one point.

**Setting up the PowerCLI installation**

Once the installation is complete, a couple of minor tweaks are both required and recommended. The first required portion of the environment setup is to set the execution policy. The execution policy definition is taken directly from the PoSH help file:

```powershell
Get-Help About_Execution_Policies
```

Windows PowerShell execution policies let you determine the conditions under which Windows PowerShell loads configuration files and runs scripts.

You can set an execution policy for the local computer, for the current user, or for a particular session. You can also use a Group Policy setting to set execution policy for computers and users.

Execution policies for the local computer and current user are stored in the registry. You do not need to set execution policies in your Windows PowerShell profile. The execution policy for a particular session is stored only in memory and is lost when the session is closed.

The execution policy is not a security system that restricts user actions. For example, users can easily circumvent a policy by typing the script contents at the command line when they cannot run a script. Instead, the execution policy helps users to set basic rules and prevents them from violating them unintentionally.

Of all the Execution Policies, *Restricted* is enabled by default:

*"Restricted" is the default policy*
- Default execution policy.
- Permits individual commands, but will not run scripts.
- Prevents running of all script files, including formatting and configuration files (ps1xml), module script files (.psm1), and Windows PowerShell profiles (.ps1).

After reading the About_Execution_Policies file about execution policies and reading this default policy, it is clear to see that this will not allow the interface to be very useful when running PowerCLI. In fact, if the PowerCLI icon is clicked on without setting the execution policy to a less restrictive setting, the snap-in will not load.

It shows it in red to ensure that the error is read and understood. To bypass this, the Set-ExecutionPolicy command must be run under an administrator credential. Right-click on the icon for PowerCLI and select Run as Administrator. This enables the User Access Control for the PowerCLI/PowerShell window.

One of the following options must be selected:

**AllSigned**

- Scripts can run.
- Requires that all scripts and configuration files be signed be a trusted publisher, including script that you write on the local computer.
- Prompts you before running scripts from publishers that you have not yet classified as trusted or untrusted.
- Risks running signed, but malicious, scripts.

**RemoteSigned**

- Scripts can run.
- Requires a digital signature from a trusted publisher on script and configuration files that are downloaded from the Internet (including e-mail and instant messaging programs).
- Does not require digital signatures on scripts that you have written on the local computer (not downloaded from the Internet).
- Runs Scripts that are downloaded from the Internet and not signed, if the scripts are unblocked, such as by using the Unblock-File cmdlet.
- Risks running unsigned scripts from sources other than the Internet and signed, but malicious, scripts.

UnRestricted

- Unsigned scripts can run. (This risks running malicious scripts.)
- Warns the user before running scripts and configuration files that are downloaded from the Internet.

Bypass

- Nothing is blocked and there are no warnings or prompts.
- This execution policy is designed for configurations in which a Windows PowerShell script is built into a larger application or for configurations in which Windows PowerShell is the foundation for a program that has its own security model.

Undefined

- There is no execution policy set in the current scope.
- If the execution policy in all scopes is Undefined, the effective execution policy is Restricted, which is the default execution policy.

Typically, most environments use the RemoteSigned option as this allows the functionality to run scripts on other computers and allows some additional security from unsigned scripts. If the environment being set up is a highly secure environment, the AllSigned option may be used when proper Public Key Infrastructure (PKI) is set up. This functionality will not be described in this book, and the RemoteSigned option will be assumed for the remainder of the book:

Set-ExecutionPolicy RemoteSigned
Once the `ExecutionPolicy` is set, configuration of the interface window should be next as it requires the additional permissions of the Administrator account. Right-click on the **Heading** window, and select **Properties**.

![Properties Window](image)

The **QuickEdit** mode allows the right-clicking, copying, and pasting functionality of the **Command** window. This feature helps tremendously when doing quick one-liners, needing report information or just wanting a portion of a command for another.
Select the **Layout** tab at the top of the **Options** window.

![Layout tab in PowerCLI properties window](image)

Set **Screen Buffer Size** for **Width** and **Height** whenever possible. The width buffer should match the actual width size. This permits the interface to need limited scrolling when getting output. The height buffer should be large, up to **9999**, but if memory is a concern for the window, reduce the buffer size. This buffer allows for the output of a command to exceed the window size. For example, if the `Get-VM` command produces a listing of 3000 VMs, having a small buffer may not allow the window to be scrolled back to view the entire output. If the buffer size is 250, then 250 lines of scroll back will be permitted.
The last thing to check is the PowerCLI configuration. If the environment being worked on has multiple vCenter servers, having the configuration support multiple systems simultaneously can be advantageous. By default, the configuration is to connect to a single vCenter at a time; however, if the session is connected to one vCenter and a second `Connect-VIServer` command is run, this message pops up on the interface window:

**Working with multiple default servers?**

*Select [Y] if you want to work with more than one default servers. In this case, every time when you connect to a different server using `Connect-VIServer`, the new server connection is stored in an array variable together with the previously connected servers. When you run a cmdlet and the target servers cannot be determined from the specified parameters, the cmdlet runs against all servers stored in the array variable. Select [N] if you want to work with a single default server. In this case, when you run a cmdlet and the target servers cannot be determined from the specified parameters, the cmdlet runs against the last connected server.*

**WARNING: WORKING WITH MULTIPLE DEFAULT SERVERS WILL BE ENABLED BY DEFAULT IN A FUTURE RELEASE.** You can explicitly set your own preference at any time by using the `DefaultServerMode` parameter of `Set-PowerCLIConfiguration`.

[Y] Yes  [N] No  [S] Suspend  [?] Help (default is "Y"): y

The `Get-PowerCLIConfiguration` command will show three scopes of the configuration, Session, User, and AllUsers. The standard setup will show the following output:

<table>
<thead>
<tr>
<th>Scope</th>
<th>ProxyPolicy</th>
<th>DefaultUIUserServerMode</th>
<th>InvalidCertificateAction</th>
<th>DisplayDeprecationWarnings</th>
<th>WebOperationTimeout Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>UseSystemProxy</td>
<td>Multiple</td>
<td>Unset</td>
<td>True</td>
<td>300</td>
</tr>
<tr>
<td>User</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AllUsers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Based on the preceding command connecting to the second vCenter, the configuration will show the following output:

<table>
<thead>
<tr>
<th>Scope</th>
<th>ProxyPolicy</th>
<th>DefaultUIUserServerMode</th>
<th>InvalidCertificateAction</th>
<th>DisplayDeprecationWarnings</th>
<th>WebOperationTimeout Seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>Session</td>
<td>UseSystemProxy</td>
<td>Multiple</td>
<td>Unset</td>
<td>True</td>
<td>300</td>
</tr>
<tr>
<td>User</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AllUsers</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Now, to allow AllUsers by default, the command to enable that would be as follows:

```powershell
Set-PowerCLIConfiguration -DefaultVIServerMode Multiple -confirm:$False
```

Finally, the output would be displayed as follows:

```
<table>
<thead>
<tr>
<th>Scope</th>
<th>ProxyPolicy</th>
<th>DefaultVIServerMode</th>
<th>InvalidCertificateAction</th>
<th>DisplayDeprecationWarnings</th>
<th>WebOperationTimeout</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AllUsers</td>
<td></td>
<td>Multiple</td>
<td>Uncert</td>
<td>True</td>
<td>300</td>
</tr>
</tbody>
</table>
```

Although there are many other environment options that can be performed, these, in the author's opinion, are the most useful.

**Summary**

In this chapter, we have explained what PowerShell is, what the basics of the versions are, and how to get it, install it, and get help for it. We also discussed how to implement VMware PowerCLI and how to setup the interface and provide consistency for the scriptwriter. Although, as the reader, most of this information likely is understood, it will establish a baseline to assist in future chapters with regard to versions and setup.

In the next chapter, we will deal with the adaptation of Point and Click administration to PowerShell commands, look at tools that can be used, and orchestrate a VM build script. We will examine how to do reporting, and change configuration on multiple VMs at the same time.
Where to buy this book

You can buy PowerCLI Essentials from the Packt Publishing website.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.