Getting Started with PowerShell

Windows PowerShell is a task-based command-line shell and scripting language designed specifically for system administration. It helps IT professionals to control and automate the administration of Windows OS and applications that run on Windows.

Getting Started with PowerShell is designed to help you get up and running with PowerShell, taking you from the basics of installation to writing scripts and web server automation. This book, as an introduction to the central topics of PowerShell, covers finding and understanding PowerShell commands and packaging code for reusability, right through to a practical example of automating IIS.

Along the way, you will learn to perform data manipulation and solve common problems using basic file input/output functions. By the end of this book, you will be familiar with PowerShell and be able to utilize the lessons learned from the book to automate your servers.

Who this book is written for

This book is intended for Windows administrators or DevOps users who need to use PowerShell to automate tasks. Whether you know nothing about PowerShell or know just enough to get by, this guide will give you what you need to take your scripting to the next level.

What you will learn from this book

- Learn to verify your installed version of PowerShell, upgrade it, and start a PowerShell session using the ISE
- Discover PowerShell commands and cmdlets and understand PowerShell formatting
- Use the PowerShell help system to understand what particular cmdlets do
- Utilize the pipeline to perform typical data manipulation
- Package your code in scripts, functions, and modules
- Solve common problems using basic file input/output functions
- Find system information with WMI and CIM
- Automate IIS functionality and manage it using the WebAdministration module

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'First Steps'
- A synopsis of the book’s content
- More information on *Getting Started with PowerShell*
About the Author

Michael Shepard has been working with computers since the early 1980s, starting with an Apple II at school and a Commodore 64 at home. He first worked in IT in 1989 and has been a full-time professional in this field since 1997. He has been working at Jack Henry and Associates since 2000. Mike's focus has changed over the years from being a database application developer to a DBA (an application administrator), and he is now a solutions architect. In his years as a DBA, he found PowerShell to be a critical component in creating the automation required to keep up with a growing set of servers and applications. He is active in the PowerShell community on StackOverflow and in the projects on CodePlex. He has been blogging about PowerShell since 2009 at https://powershellstation.com and is the author of *PowerShell Troubleshooting Guide* by Packt Publishing.
Since its introduction in 2007, Windows PowerShell has become the *de facto* scripting solution for Windows system administrators and other IT professionals. With its extensive functionality and powerful language, PowerShell can be used to automate almost any area of the Windows ecosystem. If you haven't learned PowerShell yet, there's no time like the present. This book will guide you through the most important topics to begin your learning journey. If you have used PowerShell before but wish to have a refresher, this book should serve as a guide to the topics that you need to understand in order to be successful in using PowerShell for your scripting and automation needs.

**What this book covers**

*Chapter 1, First Steps*, covers the installation of Powershell and the validation of the installed version, as well as how to issue the first few commands.

*Chapter 2, Building Blocks*, introduces the "Big 3" cmdlets that reveal the workings of the PowerShell cmdlets.

*Chapter 3, Objects and PowerShell*, explains how the output in PowerShell is always an object, and how we can deal with these objects.

*Chapter 4, Life on the Assembly Line*, presents the PowerShell pipeline and the *-object cmdlets that provide a tremendous ability for data manipulation.

*Chapter 5, Formatting Output*, explains the PowerShell formatting system, including how the output is formatted by default, as well as how to control formatting.

*Chapter 6, Scripts*, shows how sequences of commands can be packaged into a reusable unit.
Chapter 7, Functions, explains how to use functions as a second way of building reusable blocks of code.

Chapter 8, Modules, demonstrates PowerShell's method of organizing code libraries.

Chapter 9, File I/O, shows several ways to work with files in PowerShell for input and output.

Chapter 10, WMI and CIM, explains how to access WMI repositories with the WMI and CIM cmdlets.

Chapter 11, Web Server Administration, demonstrates a few techniques to use PowerShell to administer the IIS web server.

Appendix, Next Steps, suggests some "next steps" that you will find useful as skills to add to your PowerShell repertoire.
First Steps

In the world of Microsoft system administration, it is becoming impossible to avoid PowerShell. If you've never looked at PowerShell before or want a refresher to make sure you understand what is going on, this chapter would be a good place to start. We will cover the following topics:

- Determining the installed PowerShell version
- Installing/Upgrading PowerShell
- Starting a PowerShell session
- Simple PowerShell commands
- PowerShell Aliases

Determining the installed PowerShell version

There are many ways to determine which version of PowerShell, if any, is installed on a computer. We will examine how this information is stored in the registry and how to have PowerShell tell you the version.

It's helpful while learning new things to actually follow along on the computer as much as possible. A big part of the learning process is developing "muscle memory", where you get used to typing the commands you see throughout the book. As you do this, it will also help you pay closer attention to the details in the code you see, since the code won't work correctly if you don't get the syntax right.
Using the registry to find the installed version

If you're running at least Windows 7 or Server 2008, you already have PowerShell installed on your machine. Windows XP and Server 2003 can install PowerShell, but it isn't pre-installed as part of the Operating System (OS). To see if PowerShell is installed at all, no matter what OS is running, inspect the following registry entry:

```
HKLM\Software\Microsoft\PowerShell\1\Install
```

If this exists and contains the value of 1, PowerShell is installed. To see, using the registry, what version of PowerShell is installed, there are a couple of places to look at.

First, the following value will exist if PowerShell 3.0 or higher is installed:

```
HKLM\SOFTWARE\Microsoft\PowerShell\3\PowerShellEngine\PowerShellVersion
```

If this value exists, it will contain the PowerShell version that is installed. For instance, my Windows 7 computer is running PowerShell 4.0, so `regedit.exe` shows the value of 4.0 in that entry:

![Registry Editor](image.png)

If this registry entry does not exist, you have either PowerShell 1.0 or 2.0 installed. To determine which, examine the following registry entry (note the 3 is changed to 1):

```
HKLM\SOFTWARE\Microsoft\PowerShell\1\PowerShellEngine\PowerShellVersion
```

This entry will either contain 1.0 or 2.0, to match the installed version of PowerShell.

It is important to note that all versions of PowerShell after 2.0 include the 2.0 engine, so this registry entry will exist for any version of PowerShell. For example, even if you have PowerShell 5.0 installed, you will also have the PowerShell 2.0 engine installed and will see both 5.0 and 2.0 in the registry.
Using PowerShell to find the installed version

No matter what version of PowerShell you have installed, it will be installed in the same place. This installation directory is called **PSHOME** and is located at:

```
%WINDIR%\System32\WindowsPowerShell\v1.0\n```

In this folder, there will be an executable file called **PowerShell.exe**. There should be a shortcut to this in your start menu, or in the start screen, depending on your operating system. In either of these, search for PowerShell and you should find it. Running this program opens the PowerShell console, which is present in all the versions.

At first glance, this looks like Command Prompt but the text (and perhaps the color) should give a clue that something is different:

```
PS C:\Windows\System32\WindowsPowerShell\v1.0>
```

---

Download the example code

You can download the example code files from your account at [http://www.packtpub.com](http://www.packtpub.com) for all the Packt Publishing books you have purchased. If you purchased this book elsewhere, you can visit [http://www.packtpub.com/support](http://www.packtpub.com/support) and register to have the files e-mailed directly to you.
In this console, type `$PSVersionTable` in the command-line and press Enter. If the output is an error message, PowerShell 1.0 is installed, because the `$PSVersionTable` variable was introduced in PowerShell 2.0. If a table of version information is seen, the PSVersion entry in the table is the installed version. The following is the output on a typical computer, showing that PowerShell 4.0 is installed:

![Image of PowerShell output](image.png)

**Warning**

You might find some references on the Internet telling you to use the `get-host` cmdlet to easily find the PowerShell version. Unfortunately, this only tells you the version of the PowerShell host, that is, the program you're using to run PowerShell.

**Installing/upgrading PowerShell**

If you don't have PowerShell installed or want a more recent version of PowerShell, you'll need to find the Windows Management Framework (WMF) download that matches the PowerShell version you want. WMF includes PowerShell as well as other related tools such as Windows Remoting (WinRM), Windows Management Instrumentation (WMI), and Desired State Configuration (DSC). The contents of the distribution change from version to version, so make sure to read the release notes included in the download. Here are links to the installers:

<table>
<thead>
<tr>
<th>PowerShell Version</th>
<th>URL</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td><a href="http://support.microsoft.com/kb/926139">http://support.microsoft.com/kb/926139</a></td>
</tr>
<tr>
<td>2.0</td>
<td><a href="http://support2.microsoft.com/kb/968929/en-us">http://support2.microsoft.com/kb/968929/en-us</a></td>
</tr>
<tr>
<td>PowerShell Version</td>
<td>URL</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------------</td>
</tr>
</tbody>
</table>

Note that PowerShell 5.0 has not been officially released, so the table lists the February 2015 preview, the latest at the time of writing.

The PowerShell 1.0 installer was released as an executable (.exe), but since then the releases have all been as standalone Windows update installers (.msu). All of these are painless to execute. You can simply download the file and run it from the explorer or from the Run… option in the start menu. PowerShell installs don't typically require a reboot but it's best to plan on doing one, just in case.

It's important to note that you can only have one version of PowerShell installed, and you can't install a lower version than the version that was shipped with your OS. Also, there are noted compatibility issues between various versions of PowerShell and Microsoft products such as Exchange, System Center, and Small Business Server, so make sure to read the system requirements section on the download page. Most of the conflicts can be resolved with a service pack of the software, but you should be sure of this before upgrading PowerShell on a server.

**Starting a PowerShell session**

We already started a PowerShell session earlier in the section on using PowerShell to find the installed version. So, what more is there to see? It turns out that there is more than one program used to run PowerShell, possibly more than one version of each of these programs, and finally, more than one way to start each of them. It might sound confusing but it will all make sense shortly.

**PowerShell hosts**

A PowerShell host is a program that provides access to the PowerShell engine in order to run PowerShell commands and scripts. The `PowerShell.exe` that we saw in the `PSHOME` directory earlier in this chapter is known as the **console host**. It is cosmetically similar to Command Prompt (`cmd.exe`) and only provides a command-line interface. Starting with Version 2.0 of PowerShell, a second host was provided.
The Integrated Scripting Environment (ISE) is a graphical environment providing multiple editors in a tabbed interface along with menus and the ability to use plugins. While not as fully featured as an Integrated Development Environment (IDE), the ISE is a tremendous productivity tool used to build PowerShell scripts and is a great improvement over using an editor, such as notepad for development.

The ISE executable is stored in `PSHOME`, and is named `powershell_ise.exe`. In Version 2.0 of the ISE, there were three sections, a tabbed editor, a console for input, and a section for output. Starting with Version 3.0, the input and output sections were combined into a single console that is more similar to the interface of the console host. The Version 4.0 ISE is shown as follows:
I will be using the **Light Console, Light Editor** theme for the ISE in most of the screenshots for this book, because the dark console does not work well on the printed page. To switch to this theme, open the **Options** item in the **Tools** Menu and select **Manage Themes...** in the options window:
Press the Manage Themes... button, select the Light Console, Light Editor option from the list and press OK. Press OK again to exit the options screen and your ISE should look something similar to the following:

![Windows PowerShell ISE](image)

Note that you can customize the appearance of the text in the editor and the console pane in other ways as well. Other than switching to the light console display, I will try to keep the settings to default.
64-bit and 32-bit PowerShell

In addition to the console host and the ISE, if you have a 64-bit operating system, you will also have 64-bit and 32-bit PowerShell installations that will include separate copies of both the hosts.

As mentioned before, the main installation directory, or $PSHOME, is found at $WINDIR\System32\WindowsPowerShell\v1.0. The version of PowerShell in $PSHOME matches that of the operating system. In other words, on a 64-bit OS, the PowerShell in $PSHOME is 64-bit. On a 32-bit system, $PSHOME has a 32-bit PowerShell install. On a 64-bit system, a second 32-bit system is found in $WINDIR\SysWOW64\WindowsPowerShell\v1.0.

Isn't that backward?

It seems backward that the 64-bit install is in the System32 folder and the 32-bit install is in SysWOW64. The System32 folder is always the primary system directory on a Windows computer, and this name has remained for backward compatibility reasons. SysWOW64 is short for Windows on Windows 64-bit. It contains the 32-bit binaries required for 32-bit programs to run in a 64-bit system, since 32-bit programs can't use the 64-bit binaries in System32.

Looking in the Program Files\Accessories\Windows PowerShell menu in the start menu of a 64-bit Windows 7 install, we see the following:

Here, the 32-bit hosts are labeled as (x86) and the 64-bit versions are undesignated. When you run the 32-bit hosts on a 64-bit system, you will also see the (x86) designation in the title bar:
PowerShell as an administrator
When you run a PowerShell host, the session is not elevated. This means that even though you might be an administrator of the machine, the PowerShell session is not running with administrator privileges. This is a safety feature to help prevent users from inadvertently running a script that damages the system.

In order to run a PowerShell session as an administrator, you have a couple of options. First, you can right-click on the shortcut for the host and select Run as administrator from the context menu. When you do this, unless you have disabled the UAC alerts, you will see a User Account Control (UAC) prompt verifying whether you want to allow the application to run as an administrator.

Selecting Yes allows the program to run as an administrator, and the title bar reflects that this is the case:
The second way to run one of the hosts as an administrator is to right-click on the shortcut and choose **Properties**. On the shortcut tab of the properties window, press the **Advanced** button. In the **Advanced Properties** window that pops up, check the **Run as administrator** checkbox and press **OK**, and **OK** again to exit out of the properties window:

Using this technique will cause the shortcut to always launch as an administrator, although the UAC prompt will still appear.

---

If you choose to disable UAC, PowerShell hosts always run as administrators. Note that disabling UAC alerts is not recommended.
Simple PowerShell commands

Now that we know all the ways that can get a PowerShell session started, what can we do in a PowerShell session? I like to introduce people to PowerShell by pointing out that most of the command-line tools that they already know work fine in PowerShell. For instance, try using `DIR`, `CD`, `IPCONFIG`, and `PING`. Commands that are part of Command Prompt (think DOS commands) might work slightly different in PowerShell if you look closely, but typical command-line applications work exactly the same as they have always worked in Command Prompt:

PowerShell commands, called **cmdlets**, are named with a verb-noun convention. Approved verbs come from a list maintained by Microsoft and can be displayed using the `get-verb` cmdlet:
By controlling the list of verbs, Microsoft has made it easier to learn PowerShell. The list is not very long and it doesn't contain verbs that have the same meaning (such as Stop, End, Terminate, and Quit), so once you learn a cmdlet using a specific verb, you can easily guess the meaning of the cmdlet names that include the verb.

Some other easy to understand cmdlets are:

- **Clear-Host** (clears the screen)
- **Get-Date** (outputs the date)
- **Start-Service** (starts a service)
- **Stop-Process** (stops a process)
- **Get-Help** (shows help about something)

Note that these use several different verbs. From this list, you can probably guess what cmdlet you would use to stop a service. Since you know there's a **Start-Service** cmdlet, and you know from the **Stop-Process** cmdlet that Stop is a valid verb, it is logical that **Stop-Service** is what you would use. The consistency of PowerShell cmdlet naming is a tremendous benefit to learners of PowerShell, and it is a policy that is important as you write the PowerShell code.

---

**What is a cmdlet?**

The term cmdlet was coined by Jeffery Snover, the inventor of PowerShell to refer to the PowerShell commands. The PowerShell commands aren't particularly different from other commands, but by giving a unique name to them, he ensured that PowerShell users would be able to use search engines to easily find PowerShell code simply by including the term cmdlet.
PowerShell aliases

If you tried to use `DIR` and `CD` in the last section, you may have noticed that they didn't work exactly as the DOS commands that they resemble. In case you didn't see this, enter `DIR /S` on a PowerShell prompt and see what happens. You will either get an error complaining about a path not existing, or get a listing of a directory called `S`. Either way, it's not the listing of files including subdirectories. Similarly, you might have noticed that `CD` in PowerShell allows you to switch between drives without using the `/D` option and even lets you change to a UNC path. The point is that, these are not DOS commands. What you're seeing is a PowerShell alias.

Aliases in PowerShell are alternate names that PowerShell uses for both PowerShell commands and programs. For instance, in PowerShell, `DIR` is actually an alias for the `Get-ChildItem` cmdlet. The `CD` alias points to the `Set-Location` cmdlet. Aliases exist for many of the cmdlets that perform operations similar to the commands in DOS, Linux, or Unix shells. Aliases serve two main purposes in PowerShell, as follows:

- They allow more concise code on Command Prompt
- They ease users' transition from other shells to PowerShell

To see a list of all the aliases defined in PowerShell, you can use the `Get-Alias` cmdlet. To find what an alias references, type `Get-Alias <alias>`. For example, see the following screenshot:

To find out what aliases exist for a cmdlet, type `Get-Alias -Definition <cmdlet>` as follows:
Here we can see that the `Get-ChildItem` cmdlet has three aliases. The first and last assist in transitioning from DOS and Linux shells, and the middle one is an abbreviation using the first letters in the name of the cmdlet.

**Summary**

This chapter focused on figuring out what version of PowerShell was installed and the many ways to start a PowerShell session. A quick introduction to PowerShell cmdlets showed that a lot of the command-line knowledge we have from DOS can be used in PowerShell and that aliases make this transition easier.

In the next chapter, we will look at the `Get-Command`, `Get-Help`, and `Get-Member` cmdlets, and learn how they unlock the entire PowerShell ecosystem for us.

**For further reading**

The Monad Manifesto, which outlines the original vision of the PowerShell project:


- `get-help about_aliases`
- `get-help about_PowerShell.exe`
- `get-help about_PowerShell_ISE.exe`
Where to buy this book

You can buy Getting Started with PowerShell 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.