

Windows® PowerShell 3.0

Ed Wilson



online book + practice files

Step by Step



Windows PowerShell 3.0 Step by Step

Your hands-on, step-by-step guide to automating Windows® administration with Windows PowerShell 3.0

Teach yourself the fundamentals of Windows PowerShell 3.0 command line interface and scripting language—one step at a time. Ideal for those with fundamental programming skills, this tutorial provides practical, learn-by-doing exercises to help you automate maintenance and administrative tasks.

Discover how to:

- Manage local and remote systems using built-in cmdlets
- Write scripts to handle recurring operations
- Concurrently accomplish multiple tasks
- Connect to a remote system and run commands
- Reuse code and simplify script creation
- Manage users, groups, and computers with Active Directory®
- Track down and fix script errors with the Windows PowerShell debugger
- Execute scripts to administer and troubleshoot Microsoft Exchange Server 2010

Your *Step by Step* digital content includes:

- Downloadable practice files
See <http://go.microsoft.com/fwlink/?Linkid=275531>
- Fully searchable ebook. See the instruction page at the back of the book

microsoft.com/mspress

U.S.A. \$54.99
Canada \$57.99
[Recommended]

Programming/Windows PowerShell

About the Author

Ed Wilson is a senior consultant at Microsoft and a well-known scripting expert who delivers popular workshops. He's written several books on Windows scripting, including *Windows PowerShell Scripting Guide* and *Windows PowerShell 2.0 Best Practices*.

DEVELOPER ROADMAP

Start Here!

- Beginner-level instruction
- Easy-to-follow explanations and examples
- Exercises to build your first projects



Step by Step

- For experienced developers learning a new topic
- Focus on fundamental techniques and tools
- Hands-on tutorial with practice files plus eBook



Developer Reference

- Professional developers; intermediate to advanced
- Expertly covers essential topics and techniques
- Features extensive, adaptable code examples



Focused Topics

- For programmers who develop complex or advanced solutions
- Specialized topics; narrow focus; deep coverage
- Features extensive, adaptable code examples



ISBN: 978-0-7356-6339-8



9 0000



9 780735 663398

Microsoft®

Windows PowerShell™ 3.0 Step by Step

Ed Wilson

Copyright © 2013 by Ed Wilson

All rights reserved. No part of the contents of this book may be reproduced or transmitted in any form or by any means without the written permission of the publisher.

ISBN: 978-0-735-66339-8

2 3 4 5 6 7 8 9 10 LSI 8 7 6 5 4 3

Printed and bound in the United States of America.

Microsoft Press books are available through booksellers and distributors worldwide. If you need support related to this book, email Microsoft Press Book Support at mspinput@microsoft.com. Please tell us what you think of this book at <http://www.microsoft.com/learning/booksurvey>.

Microsoft and the trademarks listed at <http://www.microsoft.com/about/legal/en/us/IntellectualProperty/Trademarks/EN-US.aspx> are trademarks of the Microsoft group of companies. All other marks are property of their respective owners.

The example companies, organizations, products, domain names, email addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, email address, logo, person, place, or event is intended or should be inferred.

This book expresses the author's views and opinions. The information contained in this book is provided without any express, statutory, or implied warranties. Neither the authors, Microsoft Corporation, nor its resellers, or distributors will be held liable for any damages caused or alleged to be caused either directly or indirectly by this book.

Acquisitions and Developmental Editor: Michael Bolinger

Production Editor: Kristen Borg

Editorial Production: Zyg Group, LLC

Technical Reviewer: Thomas Lee

Copyeditor: Zyg Group, LLC

Indexer: Zyg Group, LLC

Cover Design: Twist Creative • Seattle

Cover Composition: Zyg Group, LLC

Illustrators: Rebecca Demarest and Robert Romano

*To Teresa, who makes each day seem fresh with opportunity
and new with excitement.*

Contents at a Glance

	<i>Foreword</i>	<i>xix</i>
	<i>Introduction</i>	<i>xxi</i>
CHAPTER 1	Overview of Windows PowerShell 3.0	1
CHAPTER 2	Using Windows PowerShell Cmdlets	23
CHAPTER 3	Understanding and Using PowerShell Providers	65
CHAPTER 4	Using PowerShell Remoting and Jobs	107
CHAPTER 5	Using PowerShell Scripts	131
CHAPTER 6	Working with Functions	171
CHAPTER 7	Creating Advanced Functions and Modules	209
CHAPTER 8	Using the Windows PowerShell ISE	251
CHAPTER 9	Working with Windows PowerShell Profiles	267
CHAPTER 10	Using WMI	283
CHAPTER 11	Querying WMI	307
CHAPTER 12	Remoting WMI	337
CHAPTER 13	Calling WMI Methods on WMI Classes	355
CHAPTER 14	Using the CIM Cmdlets	367
CHAPTER 15	Working with Active Directory	383
CHAPTER 16	Working with the AD DS Module	419
CHAPTER 17	Deploying Active Directory with Windows Server 2012	447
CHAPTER 18	Debugging Scripts	461
CHAPTER 19	Handling Errors	501
CHAPTER 20	Managing Exchange Server	539
APPENDIX A	Windows PowerShell Core Cmdlets	571
APPENDIX B	Windows PowerShell Module Coverage	579
APPENDIX C	Windows PowerShell Cmdlet Naming	583
APPENDIX D	Windows PowerShell FAQ	587
APPENDIX E	Useful WMI Classes	597
APPENDIX F	Basic Troubleshooting Tips	621
APPENDIX G	General PowerShell Scripting Guidelines	625
	<i>Index</i>	<i>633</i>

Contents

Foreword xix
Introduction xxi

Chapter 1 Overview of Windows PowerShell 3.0 1

Understanding Windows PowerShell 1
 Using cmdlets 3
 Installing Windows PowerShell 3
 Deploying Windows PowerShell to down-level
 operating systems 4
Using command-line utilities 5
Security issues with Windows PowerShell 6
 Controlling execution of PowerShell cmdlets 7
 Confirming actions 8
 Suspending confirmation of cmdlets 9
Working with Windows PowerShell 10
 Accessing Windows PowerShell 10
 Configuring the Windows PowerShell console 11
Supplying options for cmdlets 12
Working with the help options 13
Exploring commands: step-by-step exercises 19
Chapter 1 quick reference 22

What do you think of this book? We want to hear from you!
Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

microsoft.com/learning/booksurvey

Chapter 2	Using Windows PowerShell Cmdlets	23
	Understanding the basics of cmdlets	23
	Using the <i>Get-ChildItem</i> cmdlet	24
	Obtaining a directory listing	24
	Formatting a directory listing using the <i>Format-List</i> cmdlet	26
	Using the <i>Format-Wide</i> cmdlet	27
	Formatting a directory listing using <i>Format-Table</i>	29
	Formatting output with <i>Out-GridView</i>	31
	Leveraging the power of <i>Get-Command</i>	36
	Searching for cmdlets using wildcard characters	36
	Using the <i>Get-Member</i> cmdlet	44
	Using the <i>Get-Member</i> cmdlet to examine properties and methods	44
	Using the <i>New-Object</i> cmdlet	50
	Creating and Using the <i>wshShell</i> Object	50
	Using the <i>Show-Command</i> cmdlet	52
	Windows PowerShell cmdlet naming helps you learn	54
	Windows PowerShell verb grouping	54
	Windows PowerShell verb distribution	55
	Creating a Windows PowerShell profile	57
	Finding all aliases for a particular object	59
	Working with cmdlets: step-by-step exercises	59
	Chapter 2 quick reference	63
Chapter 3	Understanding and Using PowerShell Providers	65
	Understanding PowerShell providers	65
	Understanding the alias provider	66
	Understanding the certificate provider	68
	Understanding the environment provider	76
	Understanding the filesystem provider	80
	Understanding the function provider	85

Using the registry provider to manage the Windows registry	87
The two registry drives	87
Understanding the variable provider	97
Exploring PowerShell providers: step-by-step exercises	101
Chapter 3 quick reference	106
Chapter 4 Using PowerShell Remoting and Jobs	107
Understanding Windows PowerShell remoting	107
Classic remoting	107
WinRM	112
Using Windows PowerShell jobs	119
Using Windows PowerShell remoting: step-by-step exercises	127
Chapter 4 quick reference	130
Chapter 5 Using PowerShell Scripts	131
Why write Windows PowerShell scripts?	131
Scripting fundamentals	133
Running Windows PowerShell scripts	133
Enabling Windows PowerShell scripting support	134
Transitioning from command line to script	136
Running Windows PowerShell scripts	138
Understanding variables and constants	141
Use of constants	146
Using the <i>While</i> statement	147
Constructing the <i>While</i> statement in PowerShell	148
A practical example of using the <i>While</i> statement	150
Using special features of Windows PowerShell	150
Using the <i>Do...While</i> statement	151
Using the range operator	152
Operating over an array	152
Casting to ASCII values	152

Using the <i>Do...Until</i> statement	153
Comparing the PowerShell <i>Do...Until</i> statement with VBScript	154
Using the Windows PowerShell <i>Do</i> statement	154
The <i>For</i> statement	156
Using the <i>For</i> statement	156
Using the <i>Foreach</i> statement	158
Exiting the <i>Foreach</i> statement early	159
The <i>If</i> statement	161
Using assignment and comparison operators	163
Evaluating multiple conditions	164
The <i>Switch</i> statement	164
Using the <i>Switch</i> statement	165
Controlling matching behavior	167
Creating multiple folders: step-by-step exercises	168
Chapter 5 quick reference	170

Chapter 6 Working with Functions 171

Understanding functions	171
Using functions to provide ease of code reuse	178
Including functions in the Windows PowerShell environment	180
Using dot-sourcing	180
Using dot-sourced functions	182
Adding help for functions	184
Using a <i>here-string</i> object for help	184
Using two input parameters	186
Using a type constraint in a function	190
Using more than two input parameters	192
Use of functions to encapsulate business logic	194
Use of functions to provide ease of modification	196
Understanding filters	201
Creating a function: step-by-step exercises	205
Chapter 6 quick reference	208

Chapter 7 Creating Advanced Functions and Modules 209

- The *[cmdletbinding]* attribute 209
 - Easy verbose messages 210
 - Automatic parameter checks 211
 - Adding support for the *-whatif* parameter 214
 - Adding support for the *-confirm* parameter 215
 - Specifying the default parameter set 216
- The *parameter* attribute. 217
 - The *mandatory* parameter property. 217
 - The *position* parameter property 218
 - The *ParameterSetName* parameter property 219
 - The *ValueFromPipeline* property 220
 - The *HelpMessage* property 221
- Understanding modules 222
- Locating and loading modules. 222
 - Listing available modules 223
 - Loading modules 225
- Installing modules. 227
 - Creating a per-user Modules folder 227
 - Working with the *\$modulePath* variable 230
 - Creating a module drive 232
 - Checking for module dependencies. 234
 - Using a module from a share. 237
- Creating a module 238
- Creating an advanced function: step-by-step exercises 245
- Chapter 7 quick reference 249

Chapter 8 Using the Windows PowerShell ISE 251

- Running the Windows PowerShell ISE. 251
 - Navigating the Windows PowerShell ISE. 252
 - Working with the script pane. 254
 - Tab expansion and IntelliSense 256

Working with Windows PowerShell ISE snippets	257
Using Windows PowerShell ISE snippets to create code.	257
Creating new Windows PowerShell ISE snippets	259
Removing user-defined Windows PowerShell ISE snippets	261
Using the Commands add-on: step-by-step exercises.	262
Chapter 8 quick reference	265
Chapter 9 Working with Windows PowerShell Profiles	267
Six Different PowerShell profiles	267
Understanding the six different Windows PowerShell profiles	268
Examining the <i>\$profile</i> variable	268
Determining whether a specific profile exists.	270
Creating a new profile.	270
Design considerations for profiles	271
Using one or more profiles.	273
Using the All Users, All Hosts profile	275
Using your own file	276
Grouping similar functionality into a module	277
Where to store the profile module	278
Creating a profile: step-by-step exercises.	278
Chapter 9 quick reference	282
Chapter 10 Using WMI	283
Understanding the WMI model	284
Working with objects and namespaces	284
Listing WMI providers	289
Working with WMI classes.	289
Querying WMI.	293
Obtaining service information: step-by-step exercises	298
Chapter 10 quick reference.	305

Chapter 11 Querying WMI	307
Alternate ways to connect to WMI	307
Selective data from all instances	316
Selecting multiple properties.	316
Choosing specific instances	319
Utilizing an operator	321
Where is the <i>where</i> ?	325
Shortening the syntax.	325
Working with software: step-by-step exercises.	327
Chapter 11 quick reference	335
Chapter 12 Remoting WMI	337
Using WMI against remote systems	337
Supplying alternate credentials for the remote connection.	338
Using Windows PowerShell remoting to run WMI.	341
Using CIM classes to query WMI classes	343
Working with remote results.	344
Reducing data via Windows PowerShell parameters.	347
Running WMI jobs	350
Using Windows PowerShell remoting and WMI:	
Step-by-step exercises	352
Chapter 12 quick reference	354
Chapter 13 Calling WMI Methods on WMI Classes	355
Using WMI cmdlets to execute instance methods	355
Using the <i>terminate</i> method directly	357
Using the <i>Invoke-WmiMethod</i> cmdlet	358
Using the <i>[wmi]</i> type accelerator	360
Using WMI to work with static methods.	361
Executing instance methods: step-by-step exercises	364
Chapter 13 quick reference	366

Chapter 14 Using the CIM Cmdlets	367
Using the CIM cmdlets to explore WMI classes	367
Using the <i>-classname</i> parameter	367
Finding WMI class methods	368
Filtering classes by qualifier	369
Retrieving WMI instances	371
Reducing returned properties and instances	372
Cleaning up output from the command	373
Working with associations	373
Retrieving WMI instances: step-by-step exercises	379
Chapter 14 quick reference	382
Chapter 15 Working with Active Directory	383
Creating objects in Active Directory	383
Creating an OU	383
ADSI providers	385
LDAP names	387
Creating users	393
What is user account control?	396
Working with users	397
Creating multiple organizational units: step-by-step exercises	412
Chapter 15 quick reference	418
Chapter 16 Working with the AD DS Module	419
Understanding the Active Directory module	419
Installing the Active Directory module	419
Getting started with the Active Directory module	421
Using the Active Directory module	421
Finding the FSMO role holders	422
Discovering Active Directory	428
Renaming Active Directory sites	431
Managing users	432
Creating a user	435
Finding and unlocking Active Directory user accounts	436

Finding disabled users	438
Finding unused user accounts	440
Updating Active Directory objects: step-by-step exercises	443
Chapter 16 quick reference	445
Chapter 17 Deploying Active Directory with Windows Server 2012	447
Using the Active Directory module to deploy a new forest	447
Adding a new domain controller to an existing domain	453
Adding a read-only domain controller	455
Domain controller prerequisites: step-by-step exercises	457
Chapter 17 quick reference	460
Chapter 18 Debugging Scripts	461
Understanding debugging in Windows PowerShell	461
Understanding three different types of errors	461
Using the <i>Set-PSDebug</i> cmdlet	467
Tracing the script	467
Stepping through the script	471
Enabling strict mode	479
Using <i>Set-PSDebug -Strict</i>	479
Using the <i>Set-StrictMode</i> cmdlet	481
Debugging the script	483
Setting breakpoints	483
Setting a breakpoint on a line number	483
Setting a breakpoint on a variable	485
Setting a breakpoint on a command	489
Responding to breakpoints	490
Listing breakpoints	492
Enabling and disabling breakpoints	494
Deleting breakpoints	494
Debugging a function: step-by-step exercises	494
Chapter 18 quick reference	499

Chapter 19 Handling Errors	501
Handling missing parameters	501
Creating a default value for a parameter.	502
Making the parameter mandatory	503
Limiting choices.	504
Using <i>PromptForChoice</i> to limit selections	504
Using <i>Test-Connection</i> to identify computer connectivity	506
Using the <i>-contains</i> operator to examine contents of an array	507
Using the <i>-contains</i> operator to test for properties.	509
Handling missing rights	512
Attempt and fail	512
Checking for rights and exiting gracefully.	513
Handling missing WMI providers.	513
Incorrect data types	523
Out-of-bounds errors.	526
Using a boundary-checking function.	526
Placing limits on the parameter.	528
Using <i>Try...Catch...Finally</i>	529
Catching multiple errors	532
Using <i>PromptForChoice</i> to limit selections: Step-by-step exercises.	534
Chapter 19 quick reference	537
Chapter 20 Managing Exchange Server	539
Exploring the Exchange 2010 cmdlets	539
Working with remote Exchange servers	540
Configuring recipient settings	544
Creating the user and the mailbox	544
Reporting user settings.	548
Managing storage settings	550
Examining the mailbox database	550
Managing the mailbox database.	551

Managing Exchange logging	553
Managing auditing	557
Parsing the audit XML file	562
Creating user accounts: step-by-step exercises	565
Chapter 20 quick reference	570
Appendix A Windows PowerShell Core Cmdlets	571
Appendix B Windows PowerShell Module Coverage	579
Appendix C Windows PowerShell Cmdlet Naming	583
Appendix D Windows PowerShell FAQ	587
Appendix E Useful WMI Classes	597
Appendix F Basic Troubleshooting Tips	621
Appendix G General PowerShell Scripting Guidelines	625
<i>Index</i>	633
<i>About the Author</i>	667

What do you think of this book? We want to hear from you!

Microsoft is interested in hearing your feedback so we can continually improve our books and learning resources for you. To participate in a brief online survey, please visit:

microsoft.com/learning/booksurvey

Foreword

I've always known that automation was a critical IT Pro skill. Automation dramatically increases both productivity and quality of IT operations; it is a transformational skill that improves both the companies and the careers of the individuals that master it. Improving IT Pro automation was my top priority when I joined Microsoft in 1999 as the Architect for management products and technologies. That led to inventing Windows PowerShell and the long hard road to making it a centerpiece of the Microsoft management story. Along the way, the industry made some dramatic shifts. These shifts make it even more critical for IT Pros to become experts of automation.

During the development of PowerShell V1, the team developed a very strong partnership with Exchange. We thought Exchange would drive industry adoption of PowerShell. You can imagine our surprise (and delight) when we discovered that the most active PowerShell V1 community was VMWare customers. I reached out to the VMWare team to find out why it was so successful with their customers. They explained to me that their customers were IT Pros that were barely keeping up with the servers they had. When they adopted virtualization, they suddenly had 5-10 times the number of servers so it was either "automate or drown." Their hair was on fire and PowerShell was a bucket of water.

The move to the cloud is another shift that increases the importance of automation. The entire DevOps movement is all about making change safe through changes in culture and automation. When you run cloud scale applications, you can't afford to have it all depend upon a smart guy with a cup of coffee and a mouse—you need to automate operations with scripts and workflows. When you read the failure reports of the biggest cloud outages, you see that the root cause is often manual configuration. When you have automation and an error occurs, you review the scripts and modify them to it doesn't happen again. With automation, Nietzsche was right: that which does not kill you strengthens you. It is no surprise that Azure has supported PowerShell for some time, but I was delighted to see that Amazon just released 587 cmdlets to manage AWS.

Learning automation with PowerShell is a critical IT Pro skill and there are few people better qualified to help you do that than Ed Wilson. Ed Wilson is the husband of The Scripting Wife and the man behind the wildly popular blog The Scripting Guy. It is no exaggeration to say that Ed and his wife Teresa are two of the most active people in the PowerShell community. Ed is known for his practical "how to" approach to PowerShell. Having worked with so many customers and people learning PowerShell, Ed knows what questions you are going to have even before you have them and has taken the time to lay it all out for you in his new book: *Windows PowerShell 3.0 Step by Step*.

—Jeffrey Snover, Distinguished Engineer and Lead Architect, Microsoft Windows

Introduction

Windows PowerShell 3.0 is an essential management and automation tool that brings the simplicity of the command line to next generation operating systems. Included in Windows 8 and Windows Server 2012, and portable to Windows 7 and Windows Server 2008 R2, Windows PowerShell 3.0 offers unprecedented power and flexibility to everyone from power users to enterprise network administrators and architects.

Who should read this book

This book exists to help IT Pros come up to speed quickly on the exciting Windows PowerShell 3.0 technology. *Windows PowerShell 3.0 Step by Step* is specifically aimed at several audiences, including:

- **Windows networking consultants** Anyone desiring to standardize and to automate the installation and configuration of dot-net networking components.
- **Windows network administrators** Anyone desiring to automate the day-to-day management of Windows dot-net networks.
- **Microsoft Certified Solutions Experts (MCSEs) and Microsoft Certified Trainers (MCTs)** Windows PowerShell is a key component of many Microsoft courses and certification exams.
- **General technical staff** Anyone desiring to collect information, configure settings on Windows machines.
- **Power users** Anyone wishing to obtain maximum power and configurability of their Windows machines either at home or in an unmanaged desktop workplace environment.

Assumptions

This book expects that you are familiar with the Windows operating system, and therefore basic networking terms are not explained in detail. The book does not expect you to have any background in programming, development, or scripting. All elements related to these topics, as they arise, are fully explained.

Who should not read this book

Not every book is aimed at every possible audience. This is not a Windows PowerShell 3.0 reference book, and therefore extremely deep, esoteric topics are not covered. While some advanced topics are covered, in general the discussion starts with beginner topics and proceeds through an intermediate depth. If you have never seen a computer, nor have any idea what a keyboard or a mouse are, then this book definitely is not for you.

Organization of this book

This book is divided into three sections, each of which focuses on a different aspect or technology within the Windows PowerShell world. The first section provides a quick overview of Windows PowerShell and its fundamental role in Windows Management. It then delves into the details of Windows PowerShell remoting. The second section covers the basics of Windows PowerShell scripting. The last portion of the book covers different management technology and discusses specific applications such as Active Directory and Exchange.

Finding your best starting point in this book

The different sections of *Windows PowerShell 3.0 Step by Step* cover a wide range of technologies associated with the data library. Depending on your needs and your existing understanding of Microsoft data tools, you may wish to focus on specific areas of the book. Use the following table to determine how best to proceed through the book.

If you are	Follow these steps
New to Windows PowerShell	Focus on Chapters 1–3 and 5–9, or read through the entire book in order.
An IT pro who knows the basics of Windows PowerShell and only needs to learn how to manage network resources	Briefly skim Chapters 1–3 if you need a refresher on the core concepts. Read up on the new technologies in Chapters 4 and 10–14.
Interested in Active Directory and Exchange	Read Chapters 15–17 and 20.
Interested in Windows PowerShell Scripting	Read Chapters 5–8, 18, and 19.

Most of the book's chapters include hands-on samples that let you try out the concepts just learned.

Conventions and features in this book

This book presents information using conventions designed to make the information readable and easy to follow.

- Each chapter concludes with two exercises.
- Each exercise consists of a series of tasks, presented as numbered steps (1, 2, and so on) listing each action you must take to complete the exercise.
- Boxed elements with labels such as “Note” provide additional information or alternative methods for completing a step successfully.
- Text that you type (apart from code blocks) appears in bold.
- A plus sign (+) between two key names means that you must press those keys at the same time. For example, “Press Alt+Tab” means that you hold down the Alt key while you press the Tab key.
- A vertical bar between two or more menu items (e.g. File | Close), means that you should select the first menu or menu item, then the next, and so on.

System requirements

You will need the following hardware and software to complete the practice exercises in this book:

- One of the following: Windows 7, Windows Server 2008 with Service Pack 2, Windows Server 2008 R2, Windows 8 or Windows Server 2012.
- Computer that has a 1.6GHz or faster processor (2GHz recommended)
- 1 GB (32 Bit) or 2 GB (64 Bit) RAM (Add 512 MB if running in a virtual machine or SQL Server Express Editions, more for advanced SQL Server editions)
- 3.5 GB of available hard disk space
- 5400 RPM hard disk drive
- DirectX 9 capable video card running at 1024 × 768 or higher-resolution display

- DVD-ROM drive (if installing Visual Studio from DVD)
- Internet connection to download software or chapter examples

Depending on your Windows configuration, you might require Local Administrator rights to install or configure Visual Studio 2010 and SQL Server 2008 products.

Code samples

Most of the chapters in this book include exercises that let you interactively try out new material learned in the main text. All sample projects, in both their pre-exercise and post-exercise formats, can be downloaded from the following page:

http://aka.ms/PowerShellSBS_book

Follow the instructions to download the scripts.zip file.



Note In addition to the code samples, your system should have Windows PowerShell 3.0 installed.

Installing the code samples

Follow these steps to install the code samples on your computer so that you can use them with the exercises in this book.

1. After you download the scripts.zip file, make sure you unblock it by right-clicking on the scripts.zip file, and then clicking on the Unblock button on the property sheet.
2. Unzip the scripts.zip file that you downloaded from the book's website (name a specific directory along with directions to create it, if necessary).

Acknowledgments

I'd like to thank the following people: my agent Claudette Moore, because without her this book would never have come to pass. My editors Devon Musgrave and Michael Bolinger for turning the book into something resembling English, and my technical

reviewer Thomas Lee whose attention to detail definitely ensured a much better book. Lastly I want to acknowledge my wife Teresa (aka the Scripting Wife) who read every page and made numerous suggestions that will be of great benefit to beginning scripters.

Errata and book support

We've made every effort to ensure the accuracy of this book and its companion content. Any errors that have been reported since this book was published are listed on our Microsoft Press site:

<http://www.microsoftpressstore.com/title/9780735663398>

If you find an error that is not already listed, you can report it to us through the same page.

If you need additional support, email Microsoft Press Book Support at *mspinput@microsoft.com*.

Please note that product support for Microsoft software is not offered through the addresses above.

We want to hear from you

At Microsoft Press, your satisfaction is our top priority, and your feedback our most valuable asset. Please tell us what you think of this book at:

<http://www.microsoft.com/learning/booksurvey>

The survey is short, and we read every one of your comments and ideas. Thanks in advance for your input!

Stay in touch

Let's keep the conversation going! We're on Twitter: *<http://twitter.com/MicrosoftPress>*

Understanding and Using PowerShell Providers

After completing this chapter, you will be able to:

- Understand the role of providers in Windows PowerShell.
- Use the *Get-PSProvider* cmdlet.
- Use the *Get-PSDrive* cmdlet.
- Use the *New-PSDrive* cmdlet.
- Use the *Get-Item* cmdlet.
- Use the *Set-Location* cmdlet.
- Use the file system model to access data from each of the built-in providers.

Microsoft Windows PowerShell provides a consistent way to access information external to the shell environment. To do this, it uses *providers*. These providers are actually .NET programs that hide all the ugly details to provide an easy way to access information. The beautiful thing about the way the provider model works is that all the different sources of information are accessed in exactly the same manner using a common set of cmdlets—*Get-ChildItem*, for example—to work with different types of data. This chapter demonstrates how to leverage the PowerShell providers.



Note All scripts and files mentioned in this chapter are available via the Microsoft TechNet Script Center (http://aka.ms/powershellsbs_book).

Understanding PowerShell providers

By identifying the providers installed with Windows PowerShell, you can begin to understand the capabilities intrinsic to a default installation. Providers expose information contained in different data stores by using a drive-and-file-system analogy. An example of this is obtaining a listing of registry keys—to do this, you would connect to the registry “drive” and use the *Get-ChildItem* cmdlet, which is exactly the same method you would use to obtain a listing of files on the hard

drive. The only difference is the specific name associated with each drive. Developers familiar with Windows .NET programming can create new providers, but writing a provider can be complex. (See [http://msdn.microsoft.com/en-us/library/windows/desktop/ee126192\(v=vs.85\).aspx](http://msdn.microsoft.com/en-us/library/windows/desktop/ee126192(v=vs.85).aspx) for more information.) When a new provider is created, it might ship in a snap-in. A *snap-in* is a *dynamic-link library (DLL)* file that must be installed into Windows PowerShell. After a snap-in has been installed, it cannot be uninstalled unless the developer provides removal logic—however, the snap-in can be removed from the current Windows PowerShell console. The preferred way to ship a provider is via a Windows PowerShell module. Modules are installable via an Xcopy deployment, and therefore do not necessarily require admin rights.

To obtain a listing of all the providers, use the *Get-PSProvider* cmdlet. This command produces the following list on a default installation of Windows PowerShell (Windows 8 does not include the *WSMan* provider):

Name	Capabilities	Drives
----	-----	-----
Alias	ShouldProcess	{Alias}
Environment	ShouldProcess	{Env}
FileSystem	Filter, ShouldProcess, Crede...	{C, A, D}
Function	ShouldProcess	{Function}
Registry	ShouldProcess, Transactions	{HKLM, HKCU}
Variable	ShouldProcess	{Variable}
Certificate	ShouldProcess	{Cert}
WSMan	Credentials	{WSMan}

Understanding the alias provider

In Chapter 1, “Overview of Windows PowerShell 3.0,” I presented the various help utilities available that show how to use cmdlets. The alias provider provides easy-to-use access to all aliases defined in Windows PowerShell. To work with the aliases on your machine, use the *Set-Location* cmdlet and specify the `Alias:\` drive. You can then use the same cmdlets you would use to work with the file system.



Tip With the alias provider, you can use a *Where-Object* cmdlet and filter to search for an alias by name or description.

Working with the alias provider

1. Open the Windows PowerShell console.
2. Obtain a listing of all the providers by using the *Get-PSProvider* cmdlet.

- The PowerShell drive (PS drive) associated with the alias provider is called Alias. This is shown in the listing produced by the *Get-PSProvider* cmdlet. Use the *Set-Location* cmdlet to change to the Alias drive. Use the *sl* alias to reduce typing. This command is shown here:

```
sl alias:\
```

- Use the *Get-ChildItem* cmdlet to produce a listing of all the aliases that are defined on the system. To reduce typing, use the alias *gci* in place of *Get-ChildItem*. This is shown here:

```
gci
```

- Use a *Where-Object* cmdlet filter to reduce the amount of information that is returned by using the *Get-ChildItem* cmdlet. Produce a listing of all the aliases that begin with the letter *s*. This is shown here:

```
gci | Where name -like "s*"
```

- To identify other properties that could be used in the filter, pipeline the results of the *Get-ChildItem* cmdlet into the *Get-Member* cmdlet. This is shown here (keep in mind that different providers expose different objects that will have different properties):

```
Get-ChildItem | Get-Member
```

- Press the up arrow key twice, and edit the previous filter to include only definitions that contain the word *set*. The modified filter is shown here:

```
gci | Where definition -like "set*"
```

- The results of this command are shown here:

CommandType	Name	ModuleName
-----	----	-----
Alias	cd -> Set-Location	
Alias	chdir -> Set-Location	
Alias	sal -> Set-Alias	
Alias	sbp -> Set-PSBreakpoint	
Alias	sc -> Set-Content	
Alias	set -> Set-Variable	
Alias	si -> Set-Item	
Alias	sl -> Set-Location	
Alias	sp -> Set-ItemProperty	
Alias	sv -> Set-Variable	
Alias	swmi -> Set-WmiInstance	

- Press the up arrow key three times, and edit the previous filter to include only names of aliases that are like the letter *w*. This revised command is shown here:

```
gci | Where name -like "*w*"
```

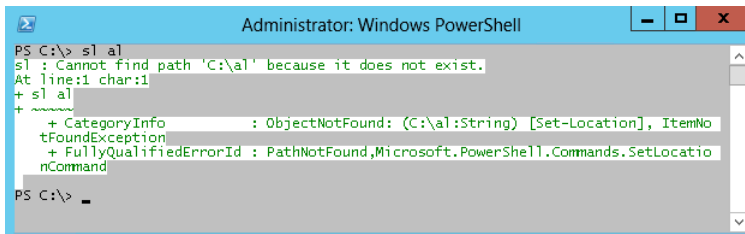
The results from this command will be similar to those shown here:

CommandType	Name	ModuleName
Alias	fw -> Format-Wide	
Alias	gwmi -> Get-WmiObject	
Alias	iwmi -> Invoke-WmiMethod	
Alias	iwr -> Invoke-WebRequest	
Alias	pwd -> Get-Location	
Alias	rwmi -> Remove-WmiObject	
Alias	swmi -> Set-WmiInstance	
Alias	where -> Where-Object	
Alias	wjb -> Wait-Job	
Alias	write -> Write-Output	

10. In the preceding list, note that *where* is an alias for the *Where-Object* cmdlet. Press the up arrow key one time to retrieve the previous command. Edit it to use the *where* alias instead of spelling out the entire *Where-Object* cmdlet name. This revised command is shown here:

```
gci | where name -like "*W*"
```

Caution When using the *Set-Location* cmdlet to switch to a different PS drive, you must follow the name of the PS drive with a colon. A trailing forward slash or backward slash is optional. An error will be generated if the colon is left out, as shown in Figure 3-1. I prefer to use the backward slash (\) because it is consistent with normal Windows file system operations.



```
Administrator: Windows PowerShell
PS C:\> sl al
sl : Cannot find path 'C:\al' because it does not exist.
At line:1 char:1
+ sl al
+ ~~~~~
+ CategoryInfo          : ObjectNotFound: (C:\al:String) [Set-Location], ItemNo
tFoundException
+ FullyQualifiedErrorId : PathNotFound,Microsoft.PowerShell.Commands.SetLocati
onCommand
PS C:\> _
```

FIGURE 3-1 Using *Set-Location* without a colon results in an error.

Understanding the certificate provider

The preceding section explored working with the alias provider. Because the file system model applies to the certificate provider in much the same way as it does the alias provider, many of the same cmdlets can be used. To find information about the certificate provider, use the *Get-Help* cmdlet and search for *about_Providers*. If you are unsure what articles in help may be related to certificates, you can use the wildcard asterisk (*) parameter. This command is shown here:

```
Get-Help *cer*
```

In addition to allowing you to use the certificate provider, Windows PowerShell gives you the ability to sign scripts; Windows PowerShell can work with signed and unsigned scripts as well. The certificate provider gives you the ability search for, copy, move, and delete certificates. Using the certificate provider, you can open the Certificates Microsoft Management Console (MMC). The commands used in the following procedure use the certificate provider to obtain a listing of the certificates installed on the local computer.

Obtaining a listing of certificates

1. Open the Windows PowerShell console.
2. Set your location to the cert PS drive. To do this, use the *Set-Location* cmdlet, as shown here:

```
Set-Location cert:\
```

3. Use the *Get-ChildItem* cmdlet to produce a list of the certificates, as shown here:

```
Get-ChildItem
```

The list produced is shown here:

```
Location      : CurrentUser
StoreNames    : {? , UserDS , AuthRoot , CA...}
```

```
Location      : LocalMachine
StoreNames    : {? , AuthRoot , CA , AddressBook...}
```

4. Use the *-recurse* argument to cause the *Get-ChildItem* cmdlet to produce a list of all the certificate stores and the certificates in those stores. To do this, press the up arrow key one time and add the *-recurse* argument to the previous command. This is shown here:

```
Get-ChildItem -recurse
```

5. Use the *-path* argument for *Get-ChildItem* to produce a listing of certificates in another store, without using the *Set-Location* cmdlet to change your current location. Use the *gci* alias, as shown here:

```
GCI -path currentUser
```

Your listing of certificate stores will look similar to the one shown here:

```
Name : ?
```

```
Name : UserDS
```

```
Name : AuthRoot
```

```
Name : CA
```

```
Name : AddressBook
```

```

Name : ?
Name : Trust
Name : Disallowed
Name : _NMSTR
Name : ?????k
Name : My
Name : Root
Name : TrustedPeople
Name : ACRS
Name : TrustedPublisher
Name : REQUEST

```

6. Change your working location to the `currentuser\authroot` certificate store. To do this, use the `sl` alias followed by the path to the certificate store (`sl` is an alias for the `Set-Location` cmdlet). This command is shown here:

```
sl currentuser\authroot
```

7. Use the `Get-ChildItem` cmdlet to produce a listing of certificates in the `currentuser\authroot` certificate store that contain the name `C&W` in the subject field. Use the `gci` alias to reduce the amount of typing. Pipeline the resulting object to a `Where-Object` cmdlet, but use the `where` alias instead of typing `Where-Object`. The code to do this is shown here:

```
gci | where subject -like "*c&w*"
```

On my machine, there are four certificates listed. These are shown here:

Thumbprint	Subject
-----	-----
F88015D3F98479E1DA553D24FD42BA3F43886AEF	O=C&W HKT SecureNet CA SGC Root, C=hk
9BACF3B664EAC5A17BED08437C72E4ACDA12F7E7	O=C&W HKT SecureNet CA Class A, C=hk
4BA7B9DDD68788E12FF852E1A024204BF286A8F6	O=C&W HKT SecureNet CA Root, C=hk
47AFB915CDA26D82467B97FA42914468726138DD	O=C&W HKT SecureNet CA Class B, C=hk

- Use the up arrow key, and edit the previous command so that it will return only certificates that contain the phrase *SGC Root* in the subject property. The revised command is shown here:

```
gci | where subject -like "*SGC Root*"
```

- The resulting output on my machine contains an additional certificate. This is shown here:

Thumbprint	Subject
-----	-----
F88015D3F98479E1DA553D24FD42BA3F43886AEF	O=C&W HKT SecureNet CA SGC Root, C=hk
687EC17E0602E3CD3F7DFBD7E28D57A0199A3F44	O=SecureNet CA SGC Root, C=au

- Use the up arrow key and edit the previous command. This time, change the *Where-Object* cmdlet so that it filters on the thumbprint attribute that is equal to *F88015D3F98479E1DA553D24FD42BA3F43886AEF*. You do not have to type that, however; to copy the thumbprint, you can highlight it and press Enter in Windows PowerShell, as shown in Figure 3-2. The revised command is shown here:

```
gci | where thumbprint -eq "F88015D3F98479E1DA553D24FD42BA3F43886AEF"
```

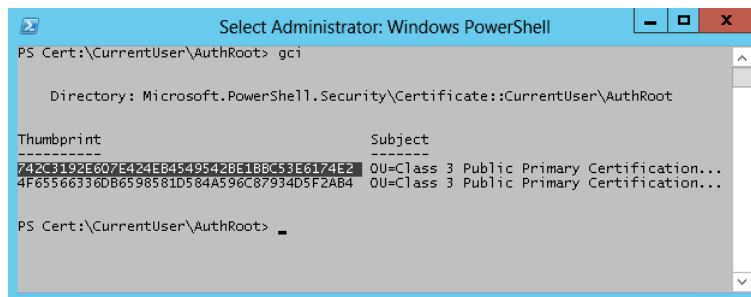


FIGURE 3-2 Highlight items to copy using the mouse.



Troubleshooting If copying from inside the Windows PowerShell console window does not work, then you may need to enable QuickEdit mode. To do this, right-click the PowerShell icon in the upper-left corner of the Windows PowerShell window. Choose Properties, click the Options tab, and then select QuickEdit Mode. This is shown in Figure 3-3.

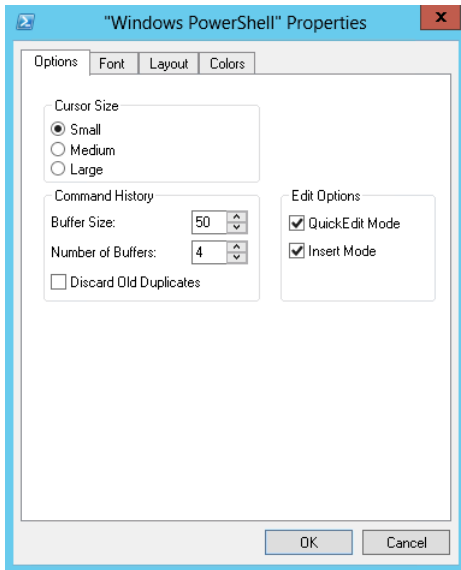


FIGURE 3-3 Enable QuickEdit mode to enable clipboard support.

11. To see all the properties of the certificate, pipeline the certificate object to a *Format-List* cmdlet and choose all the properties. The revised command is shown here:

```
gci | where thumbprint -eq " E0AB059420725493056062023670F7CD2EFC6666" |  
Format-List *
```

The output contains all the properties of the certificate object and is shown here:

```
PSPath : Microsoft.PowerShell.Security\Certificate::currentuser\  
        authroot\E0AB059420725493056062023670F7CD2EFC6666  
PSParentPath : Microsoft.PowerShell.Security\Certificate::currentuser\  
        authroot  
PSChildName : E0AB059420725493056062023670F7CD2EFC6666  
PSDrive : Cert  
PSProvider : Microsoft.PowerShell.Security\Certificate  
PSIsContainer : False  
EnhancedKeyUsageList : {Server Authentication (1.3.6.1.5.5.7.3.1), Code Signing  
        (1.3.6.1.5.5.7.3.3), Time Stamping (1.3.6.1.5.5.7.3.8)}  
DnsNameList : {Thawte Premium Server CA}
```

```

SendAsTrustedIssuer      : False
EnrollmentPolicyEndPoint : Microsoft.CertificateServices.Commands.EnrollmentEndPoint
                           Property
EnrollmentServerEndPoint : Microsoft.CertificateServices.Commands.EnrollmentEndPoint
                           Property
PolicyId                 :
Archived                 : False
Extensions               : {System.Security.Cryptography.Oid}
FriendlyName             : Thawte Premium Server CA (SHA1)
IssuerName               : System.Security.Cryptography.X509Certificates.X500
                           DistinguishedName
NotAfter                 : 1/1/2021 6:59:59 PM
NotBefore                : 7/31/1996 8:00:00 PM
HasPrivateKey            : False
PrivateKey               :
PublicKey                : System.Security.Cryptography.X509Certificates.PublicKey
RawData                  : {48, 130, 3, 54...}
SerialNumber             : 36122296C5E338A520A1D25F4CD70954
SubjectName              : System.Security.Cryptography.X509Certificates.X500
                           DistinguishedName
SignatureAlgorithm       : System.Security.Cryptography.Oid
Thumbprint               : E0AB059420725493056062023670F7CD2EFC6666
Version                  : 3
Handle                   : 647835770000
Issuer                   : E=premium-server@thawte.com, CN=Thawte Premium Server
                           CA, OU=Certification Services Division, O=Thawte
                           Consulting cc, L=Cape Town, S=Western Cape, C=ZA
Subject                  : E=premium-server@thawte.com, CN=Thawte Premium Server
                           CA, OU=Certification Services Division, O=Thawte
                           Consulting cc, L=Cape Town, S=Western Cape, C=ZA

```

12. Open the Certificates MMC file. This MMC file is called Certmgr.msc; you can launch it by simply typing the name inside Windows PowerShell, as shown here:

```
Certmgr.msc
```

13. But it is more fun to use the *Invoke-Item* cmdlet to launch the Certificates MMC. To do this, supply the PS drive name of cert:\ to the *Invoke-Item* cmdlet. This is shown here:

```
Invoke-Item cert:\
```

14. Compare the information obtained from Windows PowerShell with the information displayed in the Certificates MMC. It should be the same. The certificate is shown in Figure 3-4.

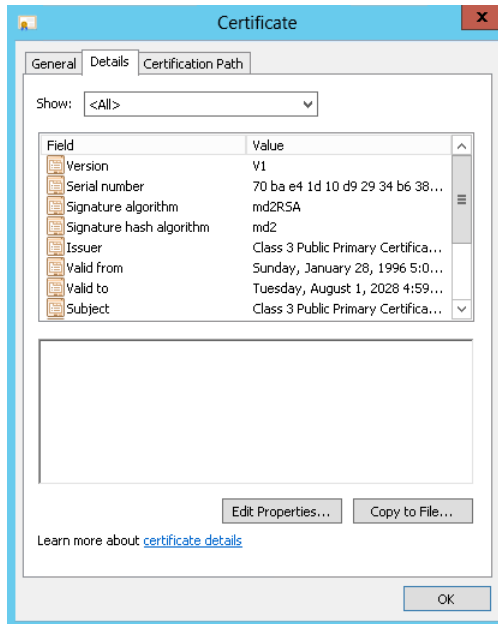


FIGURE 3-4 Certmgr.msc can be used to examine certificate properties.

This concludes this procedure.

Searching for specific certificates

To search for specific certificates, you may want to examine the *subject* property. For example, the following code examines the *subject* property of every certificate in the currentuser store beginning at the root level. It does a recursive search, and returns only the certificates that contain the word *test* in some form in the *subject* property. This command and associated output appear here:

```
PS C:\Users\administrator.IAMMRED> dir Cert:\CurrentUser -Recurse | ? subject -match 'test'
```

```
Directory: Microsoft.PowerShell.Security\Certificate::CurrentUser\Root
```

Thumbprint	Subject
-----	-----
8A334AA8052DD244A647306A76B8178FA215F344	CN=Microsoft Testing Root Certificate A...
2BD63D28D7BCD0E251195AEB519243C13142EBC3	CN=Microsoft Test Root Authority, OU=Mi...

To delete these *test* certificates simply requires pipelining the results of the previous command to the *Remove-Item* cmdlet.



Note When performing any operation that may alter system state, it is a good idea to use the *-whatif* parameter to prototype the command prior to actually executing it.

The following command uses the *-whatif* parameter from *Remove-Item* to prototype the command to remove all of the certificates from the currentuser store that contain the word *test* in the *subject* property. Once completed, retrieve the command via the up arrow key and remove the *-whatif* switched parameter from the command prior to actual execution. This technique appears here:

```
PS C:\Users\administrator.IAMMRED> dir Cert:\CurrentUser -Recurse | ? subject -match
'test' | Remove-Item -WhatIf
What if: Performing operation "Remove certificate" on Target "Item: CurrentUser\Root\
8A334AA8052DD244A647306A76B8178FA215F344 ".
What if: Performing operation "Remove certificate" on Target "Item: CurrentUser\Root\
2BD63D28D7BCD0E251195AEB519243C13142EBC3 ".
PS C:\Users\administrator.IAMMRED> dir Cert:\CurrentUser -Recurse | ? subject -match
'test' | Remove-Item
```

Finding expiring certificates

A common task in companies using certificates is to identify certificates that either have expired or are about to expire. Using the certificate provider, it is simple to identify expired certificates. To do this, use the *notafter* property from the certificate objects returned from the certificate drives. One approach is to look for certificates that expire prior to a specific date. This technique appears here:

```
PS Cert:\> dir .\CurrentUser -Recurse | where notafter -lt "5/1/2012"
```

A more flexible approach is to use the current date—therefore, each time the command runs, it retrieves expired certificates. This technique appears here:

```
PS Cert:\> dir .\CurrentUser -Recurse | where notafter -lt (Get-Date)
```

One problem with simply using the *Get-ChildItem* cmdlet on the currentuser store is that it returns certificate stores as well as certificates. To obtain only certificates, you must filter out the *psiscontainer* property. Because you will also need to filter based upon date, you can no longer use the simple *Where-Object* syntax. The *\$_* character represents the current certificate as it comes across the pipeline. Because you're comparing two properties, you must repeat the *\$_* character for each property. The following command retrieves the expiration dates, thumbprints, and subjects of all expired certificates. It also creates a table displaying the information. (The command is a single logical command, but it is broken at the pipe character to permit better display in the book.)

```
PS Cert:\> dir .\CurrentUser -Recurse |
where { !$_.psiscontainer -AND $_.notafter -lt (Get-Date)} |
ft notafter, thumbprint, subject -AutoSize -Wrap
```



Caution All versions of Microsoft Windows ship with expired certificates to permit verification of old executables that were signed with those certificates. Do not arbitrarily delete an expired certificate—if you do, you could cause serious damage to your system.

If you want to identify certificates that will expire in the next 30 days, you use the same technique involving a compound *Where-Object* command. The command appearing here identifies certificates expiring in the next 30 days:

```
PS Cert:\> dir .\CurrentUser -Recurse |  
where { $_.NotAfter -gt (Get-Date) -AND $_.NotAfter -le (Get-Date).Add(30) }
```

Understanding the environment provider

The environment provider in Windows PowerShell is used to provide access to the system environment variables. If you open a CMD (command) shell and type **set**, you will obtain a listing of all the environment variables defined on the system. (You can run the old-fashioned command prompt inside Windows PowerShell.)



Note It is easy to forget you are running the CMD prompt when you are inside of the Windows PowerShell console. Typing **exit** returns you to Windows PowerShell. The best way to determine whether you are running the command shell or Windows PowerShell is to examine the prompt. The default Windows PowerShell prompt is PS C:\>, assuming that you are working on drive C.

If you use the *echo* command in the CMD interpreter to print out the value of *%windir%*, you will obtain the results shown in Figure 3-5.

```
Administrator: Windows PowerShell  
PS C:\> cmd  
Microsoft Windows [Version 6.2.8380]  
(c) 2012 Microsoft Corporation. All rights reserved.  
  
C:\>echo %windir%  
C:\Windows  
  
C:\>set  
ALLUSERSPROFILE=C:\ProgramData  
APPDATA=C:\Users\administrator\AppData\Roaming  
CLIENTNAME=EDLT  
CommonProgramFiles=C:\Program Files\Common Files  
COMPUTERNAME=W8CLIENT8  
ComSpec=C:\Windows\system32\cmd.exe  
FP_NO_HOST_CHECK=NO  
HOMEDRIVE=C:  
HOMEPATH=\Users\administrator  
LOCALAPPDATA=C:\Users\administrator\AppData\Local  
LOGONSERVER=\\DC1  
NUMBER_OF_PROCESSORS=1  
OS=Windows_NT  
Path=C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\  
PATHEXT=.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC;.CPL  
PROCESSOR_ARCHITECTURE=x86  
PROCESSOR_IDENTIFIER=x86 Family 6 Model 23 Stepping 10, GenuineIntel  
PROCESSOR_LEVEL=6  
PROCESSOR_REVISION=170a  
ProgramData=C:\ProgramData  
ProgramFiles=C:\Program Files  
PROMPT=$PS  
PSModulePath=C:\Users\administrator\Documents\WindowsPowerShell\Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules\  
PUBLIC=C:\Users\Public  
SESSIONNAME=RDP-Tcp#0  
SystemDrive=C:  
SystemRoot=C:\Windows  
TEMP=C:\Users\ADMINI~1\AppData\Local\Temp  
TMP=C:\Users\ADMINI~1\AppData\Local\Temp  
USERDOMAIN=IAMMRED.NET
```

FIGURE 3-5 Use *set* at a CMD prompt to see environment variables.

Various applications and other utilities use environment variables as a shortcut to provide easy access to specific files, folders, and configuration data. By using the environment provider in Windows PowerShell, you can obtain a listing of the environment variables. You can also add, change, clear, and delete these variables.

Obtaining a listing of environment variables

1. Open the Windows PowerShell console.
2. Obtain a listing of the PS drives by using the *Get-PSDrive* cmdlet. This is shown here:

```
Get-PSDrive
```

3. Note that the Environment PS drive is called *Env*. Use the *Env* name with the *Set-Location* cmdlet and change to the Environment PS drive. This is shown here:

```
Set-Location Env:\
```

4. Use the *Get-Item* cmdlet to obtain a listing of all the environment variables on the system. This is shown here:

```
Get-Item *
```

5. Use the *Sort-Object* cmdlet to produce an alphabetical listing of all the environment variables by name. Use the up arrow key to retrieve the previous command, and pipeline the returned object into the *Sort-Object* cmdlet. Use the *-property* argument, and supply *name* as the value. This command is shown here:

```
Get-Item * | Sort-Object -property name
```

6. Use the *Get-Item* cmdlet to retrieve the value associated with the environment variable *windir*. This is shown here:

```
Get-Item windir
```

7. Use the up arrow key and retrieve the previous command. Pipeline the object returned to the *Format-List* cmdlet and use the wildcard character to print out all the properties of the object. The modified command is shown here:

```
Get-Item windir | Format-List *
```

8. The properties and their associated values are shown here:

```
PSPath      : Microsoft.PowerShell.Core\Environment::windir
PSDrive    : Env
PSProvider  : Microsoft.PowerShell.Core\Environment
PSIsContainer : False
Name       : windir
Key        : windir
Value      : C:\WINDOWS
```

This concludes this procedure. Do not close Windows PowerShell. Leave it open for the next procedure.

Creating a temporary new environment variable

1. You should still be in the Environment PS drive from the previous procedure. If not, use the *Set-Location env:* command).
2. Use the *Get-Item* cmdlet to produce a listing of all the environment variables. Pipeline the returned object to the *Sort-Object* cmdlet using the property *name*. To reduce typing, use the *gi* alias and the *sort* alias. This is shown here:

```
gi * | sort -property name
```

3. Use the *New-Item* cmdlet to create a new environment variable. The *-path* argument will be dot (.) because you are already on the *env:* PS drive. The *-Name* argument will be *admin*, and the *-value* argument will be your given name. The completed command is shown here:

```
New-Item -Path . -Name admin -Value mred
```

4. Use the *Get-Item* cmdlet to ensure the *admin* environment variable was properly created. This command is shown here:

```
Get-Item admin
```

The results of the previous command are shown here:

Name	Value
admin	mred

5. Use the up arrow key to retrieve the previous command. Pipeline the results to the *Format-List* cmdlet and choose All Properties. This command is shown here:

```
Get-Item admin | Format-List *
```

The results of the previous command include the PS path, PS drive, and additional information about the newly created environment variable. These results are shown here:

```
PSPath      : Microsoft.PowerShell.Core\Environment::admin
PSDrive     : Env
PSProvider  : Microsoft.PowerShell.Core\Environment
PSIsContainer : False
Name        : admin
Key         : admin
Value       : mred
```

The new environment variable exists until you close the Windows PowerShell console.

This concludes this procedure. Leave PowerShell open for the next procedure.

Renaming an environment variable

1. Use the *Get-ChildItem* cmdlet to obtain a listing of all the environment variables. Pipeline the returned object to the *Sort-Object* cmdlet and sort the list on the *name* property. Use the *gci* and *sort* aliases to reduce typing. The code to do this is shown here:

```
gci | sort -property name
```

2. The *admin* environment variable should be near the top of the list of system variables. If it is not, then create it by using the *New-Item* cmdlet. The *-path* argument has a value of dot (.); the *-name* argument has the value of *admin*, and the *-value* argument should be the user's given name. If this environment variable was created in the previous exercise, then PowerShell will report that it already exists. The command appearing here allows you to re-create the *admin* environment variable:

```
New-Item -Path . -Name admin -Value mred
```

3. Use the *Rename-Item* cmdlet to rename the *admin* environment variable to *super*. The *-path* argument combines the PS drive name with the environment variable name. The *-NewName* argument is the desired new name without the PS drive specification. This command is shown here:

```
Rename-Item -Path env:admin -NewName super
```

4. To verify that the old environment variable *admin* has been renamed *super*, press the up arrow key two or three times to retrieve the *gci | sort -property name* command. This command is shown here:

```
gci | sort -property name
```

This concludes this procedure. Do not close Windows PowerShell. Leave it open for the next procedure.

Removing an environment variable

1. Use the *Get-ChildItem* cmdlet to obtain a listing of all the environment variables. Pipeline the returned object to the *Sort-Object* cmdlet and sort the list on the *name* property. Use the *gci* and *sort* aliases to reduce typing. The code to do this is shown here:

```
gci | sort -property name
```

2. The *super* environment variable should be in the list of system variables. If it is not, then create it by using the *New-Item* cmdlet. The *-path* argument has a value of dot (*.*), the *-name* argument has a value of *super*, and the *-value* argument should be the user's given name. If this environment variable was created in the previous exercise, then PowerShell will report that it already exists. If you have deleted the *admin* environment variable, the command appearing here creates it:

```
New-Item -Path . -Name super -Value mred
```

3. Use the *Remove-Item* cmdlet to remove the *super* environment variable. The name of the item to be removed is typed following the name of the cmdlet. If you are still in the *env:\ PS* drive, you will not need to supply a *-path* argument. The command is shown here:

```
Remove-Item env:super
```

4. Use the *Get-ChildItem* cmdlet to verify that the environment variable *super* has been removed. To do this, press the up arrow key two or three times to retrieve the *gci | sort -property name* command. This command is shown here:

```
gci | sort -property name
```

This concludes this procedure.

Understanding the filesystem provider

The filesystem provider is the easiest Windows PowerShell provider to understand—it provides access to the file system. When Windows PowerShell is launched, it automatically opens on the user documents folder. Using the Windows PowerShell filesystem provider, you can create both directories and files. You can retrieve properties of files and directories, and you can delete them as well. In addition, you can open files and append or overwrite data to the files. This can be done with inline code, or by using the pipelining feature of Windows PowerShell. The commands used in the procedure are in the *IdentifyingPropertiesOfDirectories.txt*, *CreatingFoldersAndFiles.txt*, and *ReadingAndWritingForFiles.txt* files and are available from the Technet Script Repository, at http://aka.ms/powershellSBS_book.

Working with directory listings

1. Open the Windows PowerShell console.
2. Use the *Get-ChildItem* cmdlet to obtain a directory listing of drive C. Use the *gci* alias to reduce typing. This is shown here:

```
GCI C:\
```

3. Use the up arrow key to retrieve the *gci C:* command. Pipeline the object created into a *Where-Object* cmdlet and look for containers. This will reduce the output to only directories. The modified command is shown here:

```
GCI C:\ | where psiscontainer
```

4. Use the up arrow key to retrieve the *gci C:\ | where psiscontainer* command, and use the exclamation point (!) (meaning *not*) to retrieve only items in the PS drive that are not directories. The modified command is shown here. (The simplified *Where-Object* syntax does not support using the *not* operator directly on the input property.)

```
gci | ? {!(($psitem.psiscontainer)}
```

This concludes this procedure. Do not close Windows PowerShell. Leave it open for the next procedure.

Identifying properties of directories

1. Use the *Get-ChildItem* cmdlet and supply a value of *C:* for the *-path* argument. Pipeline the resulting object into the *Get-Member* cmdlet. Use the *gci* and *gm* aliases to reduce typing. This command is shown here:

```
gci -path C:\ | gm
```

2. The resulting output contains methods, properties, and more. Filter the output by pipelining it into a *Where-Object* cmdlet and specifying the *membertype* attribute as equal to *property*. To do this, use the up arrow key to retrieve the previous *gci -path C:\ | gm* command. Pipeline the resulting object into the *Where-Object* cmdlet and filter on the *membertype* attribute. The resulting command is shown here:

```
gci -path C:\ | gm | Where {$_.membertype -eq "property"}
```

3. On Windows 8, you need to use the *-force* parameter to see hidden files. Here is the command:

```
gci -path C:\ -force | gm | Where {$_.membertype -eq "property"}
```

- The preceding `gci -path C:\ | gm | where {$_.membertype -eq "property"}` command returns information on both the *System.IO.DirectoryInfo* and *System.IO.FileInfo* objects (on Windows 8, you need to use the `-force` switch to see hidden files). To reduce the output to only the properties associated with the *System.IO.FileInfo* object, you need to use a compound *Where-Object* cmdlet. Use the up arrow key to retrieve the `gci -path C:\ | gm | where {$_.membertype -eq "property"}` command. Add the *And* conjunction and retrieve objects that have a type name that is like `*file*`. The modified command is shown here:

```
gci -path C:\ | gm |
where {$_.membertype -eq "property" -AND $_.typename -like "*file*"}
```

- On Windows 8, you need to use the `-force` parameter. Here is the command to do that:

```
gci -path C:\ -force | gm |
where {$_.membertype -eq "property" -AND $_.typename -like "*file*"}
```

- The resulting output contains only the properties for a *System.IO.FileInfo* object. These properties are shown here:

```
TypeName: System.IO.FileInfo
```

Name	MemberType	Definition
Attributes	Property	System.IO.FileAttributes Attributes {get;set;}
CreationTime	Property	System.DateTime CreationTime {get;set;}
CreationTimeUtc	Property	System.DateTime CreationTimeUtc {get;set;}
Directory	Property	System.IO.DirectoryInfo Directory {get;}
DirectoryName	Property	System.String DirectoryName {get;}
Exists	Property	System.Boolean Exists {get;}
Extension	Property	System.String Extension {get;}
FullName	Property	System.String FullName {get;}
IsReadOnly	Property	System.Boolean IsReadOnly {get;set;}
LastAccessTime	Property	System.DateTime LastAccessTime {get;set;}
LastAccessTimeUtc	Property	System.DateTime LastAccessTimeUtc {get;set;}
LastWriteTime	Property	System.DateTime LastWriteTime {get;set;}
LastWriteTimeUtc	Property	System.DateTime LastWriteTimeUtc {get;set;}
Length	Property	System.Int64 Length {get;}
Name	Property	System.String Name {get;}

This concludes this procedure. Do not close Windows PowerShell. Leave it open for the next procedure.

Creating folders and files

- Use the *Get-Item* cmdlet to obtain a listing of files and folders. Pipeline the resulting object into the *Where-Object* cmdlet and use the *PsisContainer* property to look for folders. Use the *name* property to find names that contain the word *my* in them. Use the *gi* alias and the *where* alias to reduce typing. The command is shown here:

```
Set-Location c:\Mytest
GI * | Where {$_.PsisContainer -AND $_.name -Like "*my*"}
```

2. If you were following along in the previous chapters, you will have a folder called Mytest off the root of drive C. Use the *Remove-Item* cmdlet to remove the Mytest folder. Specify the *-recurse* argument to also delete files contained in the C:\Mytest folder. If your location is still set to Env, then change it to C or search for C:\Mytest. The command is shown here:

```
RI mytest -recurse
```

3. Press the up arrow key twice and retrieve the *gi * | where {\$_.PsisContainer -AND \$_.name -Like "*my*"}* command to confirm the folder was actually deleted. This command is shown here:

```
gi * | where {$_.PsisContainer -AND $_.name -Like "*my*"} 
```

4. Use the *New-Item* cmdlet to create a folder named Mytest. Use the *-path* argument to specify the path of C:\. Use the *-name* argument to specify the name of Mytest, and use the *-type* argument to tell Windows PowerShell the new item will be a directory. This command is shown here:

```
New-Item -Path C:\ -name mytest -type directory
```

The resulting output, shown here, confirms the operation:

```
Directory: Microsoft.PowerShell.Core\FileSystem::C:\

Mode                LastWriteTime         Length Name
----                -
d-----           5/4/2012   2:43 AM                mytest
```

5. Use the *New-Item* cmdlet to create an empty text file. To do this, use the up arrow key and retrieve the previous *New-Item -path C:\ -name Mytest -type directory* command. Edit the *-path* argument so that it is pointing to the C:\Mytest directory. Edit the *-name* argument to specify a text file named Myfile, and specify the *-type* argument as *file*. The resulting command is shown here:

```
New-Item -path C:\mytest -name myfile.txt -type file
```

The resulting message, shown here, confirms the creation of the file:

```
Directory: Microsoft.PowerShell.Core\FileSystem::C:\mytest

Mode                LastWriteTime         Length Name
----                -
-a---           5/4/2012   3:12 AM             0 myfile.txt
```

This concludes this procedure. Do not close Windows PowerShell. Leave it open for the next procedure.

Reading and writing for files

1. Delete Myfile.txt (created in the previous procedure). To do this, use the *Remove-Item* cmdlet and specify the *-path* argument as C:\Mytest\Myfile.txt. This command is shown here:

```
RI -Path C:\mytest\myfile.txt
```

2. Use the up arrow key twice to retrieve the *New-Item -path C:\Mytest -name Myfile.txt -type* command. Add the *-value* argument to the end of the command line and supply a value of *My file*. This command is shown here:

```
New-Item -Path C:\mytest -Name myfile.txt -Type file -Value "My file"
```

3. Use the *Get-Content* cmdlet to read the contents of myfile.txt. This command is shown here:

```
Get-Content C:\mytest\myfile.txt
```

4. Use the *Add-Content* cmdlet to add additional information to the myfile.txt file. This command is shown here:

```
Add-Content C:\mytest\myfile.txt -Value "ADDITIONAL INFORMATION"
```

5. Press the up arrow key twice and retrieve the *Get-Content C:\mytest\myfile.txt* command, which is shown here:

```
Get-Content C:\mytest\myfile.txt
```

6. The output from the *Get-Content C:\mytest\myfile.txt* command is shown here:

```
My fileADDITIONAL INFORMATION
```

7. Press the up arrow key twice, and retrieve the *Add-Content C:\mytest\myfile.txt -value "ADDITIONAL INFORMATION"* command to add additional information to the file. This command is shown here:

```
Add-Content C:\mytest\myfile.txt -Value "ADDITIONAL INFORMATION"
```

8. Use the up arrow key to retrieve the *Get-Content C:\mytest\myfile.txt* command, which is shown here:

```
Get-Content C:\mytest\myfile.txt
```

9. The output produced is shown here. Notice that the second time the command runs, the "ADDITIONAL INFORMATION" string is added to a new line in the original file.

```
My fileADDITIONAL INFORMATION  
ADDITIONAL INFORMATION
```

10. Use the *Set-Content* cmdlet to overwrite the contents of the Myfile.txt file. Specify the *-value* argument as *Setting information*. This command is shown here:

```
Set-Content C:\mytest\myfile.txt -value "Setting information"
```

11. Use the up arrow key to retrieve the *Get-Content C:\Mytest\Myfile.txt* command, which is shown here:

```
Get-Content C:\mytest\myfile.txt
```

The output from the *Get-Content* command is shown here:

```
Setting information
```

This concludes this procedure.

Understanding the function provider

The function provider provides access to the functions defined in Windows PowerShell. By using the function provider, you can obtain a listing of all the functions on your system. You can also add, modify, and delete functions. The function provider uses a file system–based model, and the cmdlets described earlier apply to working with functions. The commands used in the following procedure are in the ListingAllFunctionsOnTheSystem.txt file.

Listing all functions on the system

1. Open the Windows PowerShell console.
2. Use the *Set-Location* cmdlet to change the working location to the Function PS drive. This command is shown here:

```
Set-Location function:\
```
3. Use the *Get-Childitem* cmdlet to enumerate all the functions. Do this by using the *gci* alias, as shown here:

```
gci
```
4. The resulting list contains many functions that use *Set-Location* to change the current location to different drive letters. A partial view of this output is shown here:

CommandType	Name	ModuleName
-----	----	-----
Function	A:	
Function	B:	
Function	C:	
Function	cd..	
Function	cd\	
Function	Clear-Host	
<truncated...>		

```

Function      Get-Verb
Function      H:
Function      help
Function      I:
Function      ImportSystemModules
<truncated...>
Function      mkdir
Function      more
Function      N:
Function      O:
Function      oss
Function      P:
Function      Pause
Function      prompt
<truncated ...>
Function      TabExpansion2
<truncated ...>

```

- To return only the functions that are used for drives, use the *Get-ChildItem* cmdlet and pipe the object returned into a *Where-Object* cmdlet. Use the default *\$_* variable to filter on the *definition* attribute. Use the *-like* argument to search for definitions that contain the word *set*. The resulting command is shown here:

```
gci | Where definition -like "set*"
```

- If you are more interested in functions that are not related to drive mappings, then you can use the *-notlike* argument instead of *-like*. The easiest way to make this change is to use the up arrow key and retrieve the *gci | where {\$_.definition -like "set*"}* command, and then change the filter from *-like* to *-notlike*. The resulting command is shown here:

```
gci | Where definition -notlike "set*"
```

The resulting listing of functions is shown here:

CommandType	Name	ModuleName
-----	----	-----
Function	Clear-Host	
Function	Get-Verb	
Function	help	
Function	ImportSystemModules	
Function	mkdir	
Function	more	
Function	oss	
Function	Pause	
Function	prompt	
Function	TabExpansion2	

7. Use the *Get-Content* cmdlet to retrieve the text of the *pause* function. This is shown here (*gc* is an alias for the *Get-Content* cmdlet):

```
gc pause
```

The content of the *pause* function is shown here:

```
Read-Host 'Press Enter to continue...' | Out-Null
```

This concludes this procedure.

Using the registry provider to manage the Windows registry

In Windows PowerShell 1.0, the registry provider made it easy to work with the registry on the local system. Unfortunately, without remoting, you were limited to working with the local computer or using some other remoting mechanism (perhaps a log-on script) to make changes on remote systems. Beginning with Windows PowerShell 2.0, the inclusion of remoting makes it possible to make remote registry changes as easily as changing the local registry.

The registry provider permits access to the registry in the same manner that the filesystem provider permits access to a local disk drive. The same cmdlets used to access the file system—*New-Item*, *Get-ChildItem*, *Set-Item*, *Remove-Item*, and so on—also work with the registry.

The two registry drives

By default, the registry provider creates two registry drives. To find all of the drives exposed by the registry provider, use the *Get-PSDrive* cmdlet. These drives appear here:

```
PS C:\> Get-PSDrive -PSProvider registry | select name, root
```

Name	Root
-----	-----
HKCU	HKEY_CURRENT_USER
HKLM	HKEY_LOCAL_MACHINE

You can create additional registry drives by using the *New-PSDrive* cmdlet. For example, it is common to create a registry drive for the HKEY_CLASSES_ROOT registry hive. The code to do this appears here:

```
PS C:\> New-PSDrive -PSProvider registry -Root HKEY_CLASSES_ROOT -Name HKCR
```

WARNING: column "CurrentLocation" does not fit into the display and was removed.

Name	Used (GB)	Free (GB)	Provider	Root
-----	-----	-----	-----	-----
HKCR			Registry	HKEY_CLASSES_ROOT

Once created, the new HKCR drive is accessible in the same way as any other drive. For example, to change the working location to the HKCR drive, use either the *Set-Location* cmdlet or one of its aliases (such as *cd*). This technique appears here:

```
PS C:\> Set-Location HKCR:
```

To determine the current location, use the *Get-Location* cmdlet. This technique appears here:

```
PS HKCR:\> Get-Location
```

```
Path
----
HKCR:\
```

Once you've set the new working location, explore it by using the *Get-ChildItem* cmdlet (or one of the aliases for that cmdlet, such as *dir*). This technique appears in Figure 3-6.

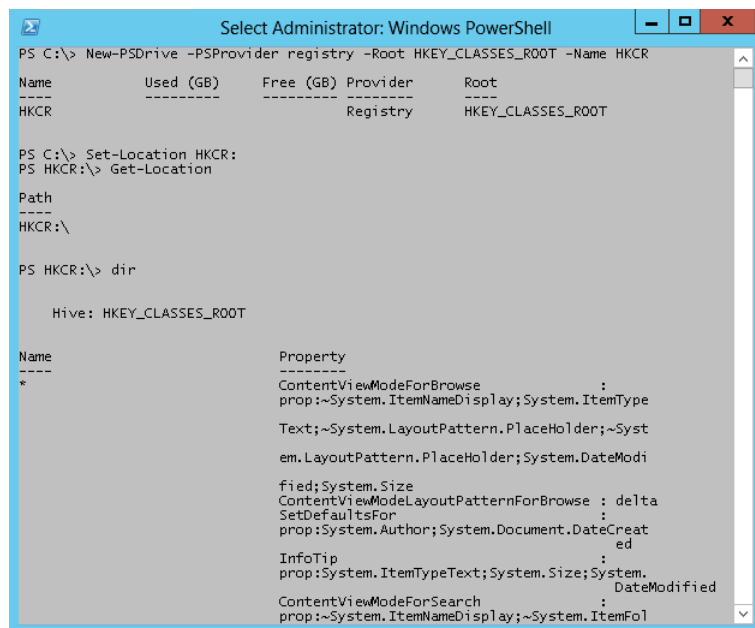


FIGURE 3-6 Creating a new registry drive for the HKEY_CLASSES_ROOT registry hive enables easy access to class registration information.

Retrieving registry values

To view the values stored in a registry key, use either the *Get-Item* or the *Get-ItemProperty* cmdlet. Using the *Get-Item* cmdlet reveals there is one property (named *default*). This appears here:

```
PS HKCR:\> Get-Item .\.\ps1 | fl *
```

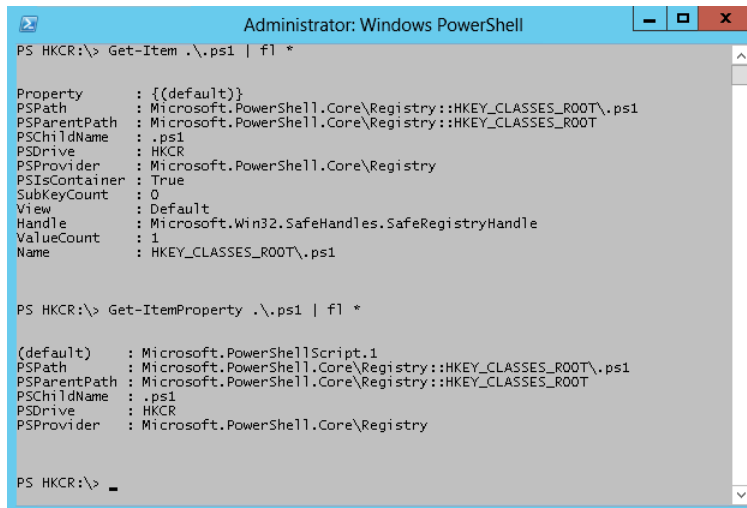
```
PSPath          : Microsoft.PowerShell.Core\Registry::HKEY_CLASSES_ROOT\.\ps1
PSParentPath    : Microsoft.PowerShell.Core\Registry::HKEY_CLASSES_ROOT
PSChildName     : .ps1
PSDrive         : HKCR
PSProvider      : Microsoft.PowerShell.Core\Registry
PSIsContainer   : True
Property        : {(default)}
SubKeyCount     : 1
ValueCount      : 1
Name           : HKEY_CLASSES_ROOT\.\ps1
```

To access the value of the *default* property, you must use the *Get-ItemProperty* cmdlet, as shown here:

```
PS HKCR:\> Get-ItemProperty .\.\ps1 | fl *
```

```
PSPath          : Microsoft.PowerShell.Core\Registry::HKEY_CLASSES_ROOT\.\ps1
PSParentPath    : Microsoft.PowerShell.Core\Registry::HKEY_CLASSES_ROOT
PSChildName     : .ps1
PSDrive         : HKCR
PSProvider      : Microsoft.PowerShell.Core\Registry
(default)       : Microsoft.PowerShellScript.1
```

The technique for accessing registry keys and the values associated with them appears in Figure 3-7.



```
Administrator: Windows PowerShell
PS HKCR:\> Get-Item .\.\ps1 | fl *
Property        : {(default)}
PSPath          : Microsoft.PowerShell.Core\Registry::HKEY_CLASSES_ROOT\.\ps1
PSParentPath    : Microsoft.PowerShell.Core\Registry::HKEY_CLASSES_ROOT
PSChildName     : .ps1
PSDrive         : HKCR
PSProvider      : Microsoft.PowerShell.Core\Registry
PSIsContainer   : True
SubKeyCount     : 0
View           : Default
Handle         : Microsoft.Win32.SafeHandles.SafeRegistryHandle
ValueCount      : 1
Name           : HKEY_CLASSES_ROOT\.\ps1

PS HKCR:\> Get-ItemProperty .\.\ps1 | fl *
(default)       : Microsoft.PowerShellScript.1
PSPath          : Microsoft.PowerShell.Core\Registry::HKEY_CLASSES_ROOT\.\ps1
PSParentPath    : Microsoft.PowerShell.Core\Registry::HKEY_CLASSES_ROOT
PSChildName     : .ps1
PSDrive         : HKCR
PSProvider      : Microsoft.PowerShell.Core\Registry
```

FIGURE 3-7 Use the *Get-ItemProperty* cmdlet to access registry property values.

Returning only the value of the *default* property requires a bit of manipulation. The *default* property requires using literal quotation marks to force the evaluation of the parentheses in the name. This appears here:

```
PS HKCR:\> (Get-ItemProperty .\ps1 -Name '(default)').'(default)'
Microsoft.PowerShellScript.1
```

The registry provider provides a consistent and easy way to work with the registry from within Windows PowerShell. Using the registry provider, you can search the registry, create new registry keys, delete existing registry keys, and modify values and access control lists (ACLs) from within Windows PowerShell.

The commands used in the following procedure are in the UnderstandingTheRegistryProvider.txt file. Two PS drives are created by default. To identify the PS drives that are supplied by the registry provider, you can use the *Get-PSDrive* cmdlet, pipeline the resulting objects into the *Where-Object* cmdlet, and filter on the *provider* property while supplying a value that is like the word *registry*. This command is shown here:

```
PS C:\> Get-PSDrive | ? provider -match registry
```

Name	Used (GB)	Free (GB)	Provider	Root
HKCR			Registry	HKEY_CLASSES_ROOT
HKCU			Registry	HKEY_CURRENT_USER
HKLM			Registry	HKEY_LOCAL_MACHINE

Obtaining a listing of registry keys

1. Open the Windows PowerShell console.
2. Use the *Get-ChildItem* cmdlet and supply *HKLM:\PSDrive* as the value for the *-path* argument. Specify the software key to retrieve a listing of software applications on the local machine. The resulting command is shown here:

```
GCI -path HKLM:\software
```

A partial listing of similar output is shown here. The corresponding keys, as displayed in Regedit.exe, are shown in Figure 3-8.

```
Hive: HKEY_LOCAL_MACHINE\SOFTWARE
```

Name	Property
ATI Technologies	
Classes	
Clients	
Intel	
Microsoft	
ODBC	
Policies	

RegisteredApplications

```
Paint : SOFTWARE\Microsoft\Windows\CurrentVersion\Applets\Paint\Capabilities
Windows Search : Software\Microsoft\Windows Search\Capabilities
Windows Disc Image Burner : Software\Microsoft\IsoBurn\Capabilities
Windows File Explorer : SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\Capabilities
Windows Photo Viewer : Software\Microsoft\Windows Photo Viewer\Capabilities
Wordpad : Software\Microsoft\Windows\CurrentVersion\Applets\Wordpad\Capabilities
Windows Media Player : Software\Clients\Media\Windows Media Player\Capabilities
Internet Explorer : SOFTWARE\Microsoft\Internet Explorer\Capabilities
Windows Address Book : Software\Clients\Contacts\Address Book\Capabilities
```

This concludes this procedure. Do not close Windows PowerShell. Leave it open for the next procedure.

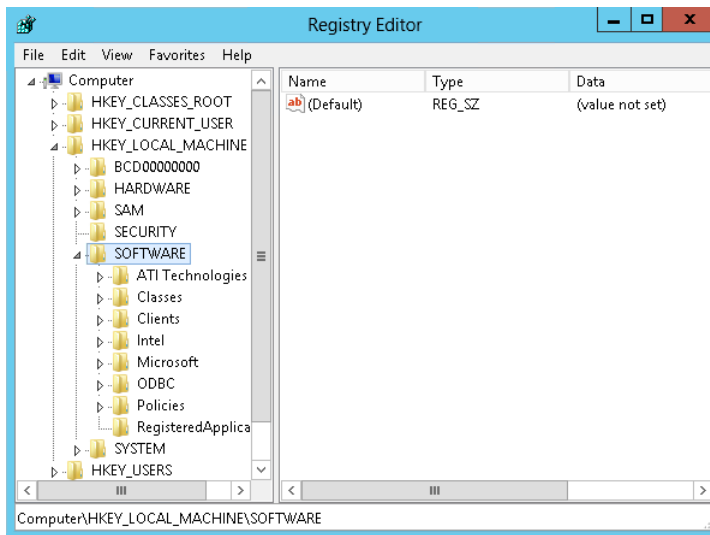


FIGURE 3-8 A Regedit.exe view of HKEY_LOCAL_MACHINE\SOFTWARE.

Searching for software

1. Use the *Get-ChildItem* cmdlet and supply a value for the *-path* argument. Use the HKLM:\PS drive and supply a path of *SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall*. To make the command easier to read, use a single quote (') to encase the string. You can use tab completion to assist with the typing. The completed command is shown here:

```
gci -path 'HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall'
```

The resulting listing of software is shown in the output here, in abbreviated fashion:

```
Hive: HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall
```

Name	Property
-----	-----
AddressBook	
CNXT_AUDIO_HDA	DisplayName : Conexant 20672 SmartAudio HD DisplayVersion : 8.32.23.2 VersionMajor : 8 VersionMinor : 0 Publisher : Conexant DisplayIcon : C:\Program Files\CONEXANT\CNXT_AUDIO_HDA\UIU64a.exe UninstallString : C:\Program Files\CONEXANT\CNXT_AUDIO_HDA\UIU64a.exe -U -G -Ichdrt.inf
Connection Manager	SystemComponent : 1
DirectDrawEx	
DXM_Runtime	
Fontcore	
IE40	
IE4Data	
IE5BAKEX	
IEData	
MobileOptionPack	
MPlayer2	
Office15.PROPLUS	Publisher : Microsoft Corporation CacheLocation : C:\MSOCache\All Users DisplayIcon : C:\Program Files\Common

2. To retrieve information on a single software package, you will need to add a *Where-Object* cmdlet. You can do this by using the up arrow key to retrieve the previous *gci -path 'HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall'* command and pipelining the resulting object into the *Where-Object* cmdlet. Supply a value for the *name* property, as shown in the code listed here. Alternatively, supply a name from the previous output.

```
PS C:\> gci -path 'HKLM:SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall' | where  
name -match 'office'
```

This concludes this procedure.

Creating new registry keys

Creating a new registry key by using Windows PowerShell is the same as creating a new file or a new folder—all three processes use the *New-Item* cmdlet. In addition to using the *New-Item* cmdlet, you might use the *Test-Path* cmdlet to determine if the registry key already exists. You may also wish to change your working location to one of the registry drives. If you do this, you might use the *Push-Location* cmdlet, *Set-Location* and the *Pop-Location* cmdlets. This is, of course, the long way of doing things. These steps appear next.



Note The registry contains information vital to the operation and configuration of your computer. Serious problems could arise if you edit the registry incorrectly. Therefore, it is important to back up your system prior to attempting to make any changes. For information about backing up your registry, see Microsoft TechNet article KB322756. For general information about working with the registry, see Microsoft TechNet article KB310516.

1. Store the current working location by using the *Push-Location* cmdlet.
2. Change the current working location to the appropriate registry drive by using the *Set-Location* cmdlet.
3. Use the *Test-Path* cmdlet to determine if the registry key already exists.
4. Use the *New-Item* cmdlet to create the new registry key.
5. Use the *Pop-Location* cmdlet to return to the starting working location.

The following example creates a new registry key named HSG off the HKEY_CURRENT_USERS software registry hive. It illustrates each of the five steps detailed previously.

Push-Location

Set-Location HKCU:

Test-Path .\Software\test

New-Item -Path .\Software -Name test

Pop-Location

The commands and the associated output from the commands appear in Figure 3-9.

```
Administrator: Windows PowerShell
PS C:\> Push-Location
PS C:\> Set-Location HKCU:
PS HKCU:\> Test-Path .\Software\test
False
PS HKCU:\> New-Item -Path .\Software -Name test

Hive: HKEY_CURRENT_USER\Software

Name          Property
----          -
test

PS HKCU:\> Pop-Location
PS C:\>
```

FIGURE 3-9 Creating a new registry key by using the *New-Item* cmdlet.

The short way to create a new registry key

It is not always necessary to change the working location to a registry drive when creating a new registry key. In fact, it is not even necessary to use the *Test-Path* cmdlet to determine if the registry key exists. If the registry key already exists, an error is generated. If you want to overwrite the registry key, use the *-force* parameter. This technique works for all the Windows PowerShell providers, not just for the registry provider.



Note How to deal with an already existing registry key is one of those *design decisions* that confront IT professionals who venture far into the world of scripting. Software developers are very familiar with these types of decisions and usually deal with them in the analyzing-requirements portion of the development life cycle. IT professionals who open the Windows PowerShell ISE first and think about the design requirements second can become easily stymied, and possibly write in problems. For more information about this, see my book *Windows PowerShell 2.0 Best Practices* (Microsoft Press, 2010).

The following example creates a new registry key named *test* in the HKCU:\SOFTWARE location. Because the command includes the full path, it does not need to execute from the HKCU drive. Because the command uses the *-force* switched parameter, the command overwrites the HKCU:\SOFTWARE\TEST registry key if it already exists.

```
New-Item -Path HKCU:\Software -Name test -Force
```



Note To see the *New-Item* cmdlet in action when using the *-force switched* parameter, use the *-verbose* switched parameter. The command appears here:

```
New-Item -Path HKCU:\Software -Name test -Force -Verbose
```

1. Include the full path to the registry key to create.
2. Use the *-force* parameter to overwrite any existing registry key of the same name.

In Figure 3-10, the first attempt to create a test registry key fails because the key already exists. The second command uses the *-force* parameter, causing the command to overwrite the existing registry key, and therefore it succeeds without creating an error.

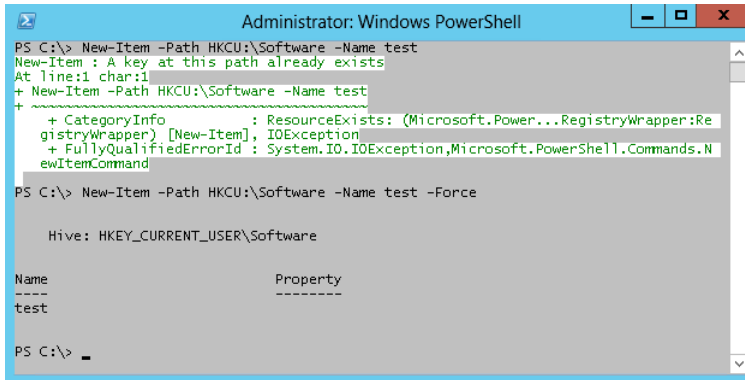


FIGURE 3-10 Use the *-force* parameter when creating a new registry key to overwrite the key if it already exists.

Setting the default value for the key

The previous examples do not set the default value for the newly created registry key. If the registry key already exists (as it does in this specific case), you can use the *Set-Item* cmdlet to assign a default value to the registry key. The steps to accomplish this appear here:

1. Use the *Set-Item* cmdlet and supply the complete path to the existing registry key.
2. Supply the default value in the *value* parameter of the *Set-Item* cmdlet.

The following command assigns the value *test key* to the default property value of the HSG registry key contained in the HKCU:\SOFTWARE location:

```
Set-Item -Path HKCU:\Software\test -Value "test key"
```

Using *New-Item* to create and assign a value

It is not necessary to use the *New-Item* cmdlet to create a registry key and then to use the *Set-Item* cmdlet to assign a default value. You can combine these steps into a single command. The following command creates a new registry key with the name of HSG1 and assigns a default value of *default value* to the registry key:

```
New-Item -Path HKCU:\Software\hsg1 -Value "default value"
```

Modifying the value of a registry property value

Modifying the value of a registry property value requires using the *Set-PropertyItem* cmdlet.

1. Use the *Push-Location* cmdlet to save the current working location.
2. Use the *Set-Location* cmdlet to change to the appropriate registry drive.
3. Use the *Set-ItemProperty* cmdlet to assign a new value to the registry property.
4. Use the *Pop-Location* cmdlet to return to the original working location.

When you know that a registry property value exists, the solution is simple: you use the *Set-ItemProperty* cmdlet and assign a new value. The code that follows saves the current working location, changes the new working location to the registry key, uses the *Set-ItemProperty* cmdlet to assign new values, and then uses the *Pop-Location* cmdlet to return to the original working location.



Note The code that follows relies upon positional parameters for the *Set-ItemProperty* cmdlet. The first parameter is *-path*. Because the *Set-Location* cmdlet set the working location to the registry key, a period identifies the path as the current directory. The second parameter is the name of the registry property to change—in this example, it is *newproperty*. The last parameter is *-value*, and that defines the value to assign to the registry property. In this example, it is *mynewvalue*. The command with complete parameter names would thus be *Set-ItemProperty -Path . -name newproperty -value mynewvalue*. The quotation marks in the following code are not required, but do not harm anything either.

```
PS C:\> Push-Location
PS C:\> Set-Location HKCU:\Software\test
PS HKCU:\Software\test> Set-ItemProperty . newproperty "mynewvalue"
PS HKCU:\Software\test> Pop-Location
PS C:\>
```

Of course, all the pushing, popping, and setting of locations is not really required. It is entirely possible to change the registry property value from any location within the Windows PowerShell provider subsystem.

The short way to change a registry property value

To change a registry property value simply, use the *Set-ItemProperty* cmdlet to assign a new value. Ensure you specify the complete path to the registry key. Here is an example of using the *Set-ItemProperty* cmdlet to change a registry property value without first navigating to the registry drive.

```
PS C:\> Set-ItemProperty -Path HKCU:\Software\test -Name newproperty -Value anewvalue
```

Dealing with a missing registry property

If you need to set a registry property value, you can set the value of that property easily by using the *Set-ItemProperty* cmdlet. But what if the registry property does not exist? How do you set the property value then? You can still use the *Set-ItemProperty* cmdlet to set a registry property value, even if the registry property does not exist, as follows:

```
Set-ItemProperty -Path HKCU:\Software\test -Name missingproperty -Value avalue
```

To determine if a registry key exists, you can simply use the *Test-Path* cmdlet. It returns *true* if the key exists and *false* if it does not exist. This technique appears here:

```
PS C:\> Test-Path HKCU:\Software\test
True
PS C:\> Test-Path HKCU:\Software\test\newproperty
False
```

Unfortunately, this technique does not work for a registry key property. It always returns *false*—even if the registry property exists. This appears here:

```
PS C:\> Test-Path HKCU:\Software\test\newproperty
False
PS C:\> Test-Path HKCU:\Software\test\bogus
False
```

Therefore, if you do not want to overwrite a registry key property if it already exists, you need a way to determine if the registry key property exists—and using the *Test-Path* cmdlet does not work. The following procedure shows how to handle this.

Testing for a registry key property prior to writing a new value

1. Use the *if* statement and the *Get-ItemProperty* cmdlet to retrieve the value of the registry key property. Specify the *erroraction* (*ea* is an alias) of *silentlycontinue* (0 is the enumeration value associated with *silentlycontinue*).
2. In the script block for the *if* statement, display a message that the registry property exists, or simply exit.
3. In the *else* statement, call *Set-ItemProperty* to create and set the value of the registry key property.

This technique appears here:

```
if((Get-ItemProperty -Path HKCU:\Software\test -Name bogus -ea 0).bogus)
{'Property already exists'}
ELSE { Set-ItemProperty -Path HKCU:\Software\test -Name bogus -Value 'initial value' }
```

Understanding the variable provider

The variable provider provides access to the variables that are defined within Windows PowerShell. These variables include both user-defined variables, such as *\$mred*, and system-defined variables, such as *\$host*. You can obtain a listing of the cmdlets designed to work specifically with variables by using the *Get-Help* cmdlet and specifying the asterisk (*) variable. The commands used in the procedure are in the *UnderstandingTheVariableProvider.txt* and *WorkingWithVariables.txt* files. To return only cmdlets, you use the *Where-Object* cmdlet and filter on the category that is equal to *cmdlet*. This command is shown here:

```
Get-Help *variable | Where-Object category -eq "cmdlet"
```

The resulting list contains five cmdlets, but is a little jumbled and difficult to read. So let's modify the preceding command and specify the properties to return. To do this, use the up arrow key and pipeline the returned object into the *Format-List* cmdlet. Add the three properties you are interested in: *name*, *category*, and *synopsis*. The revised command is shown here:

```
Get-Help *variable | Where-Object {$_.category -eq "cmdlet"} |  
Format-List name, category, synopsis
```



Note You will not get this output from Windows PowerShell 3.0 if you have not run the *Update-Help* cmdlet.

The resulting output is much easier to read and understand; it is shown here:

```
Name      : Get-Variable  
Category  : Cmdlet  
Synopsis  : Gets the variables in the current console.
```

```
Name      : New-Variable  
Category  : Cmdlet  
Synopsis  : Creates a new variable.
```

```
Name      : Set-Variable  
Category  : Cmdlet  
Synopsis  : Sets the value of a variable. Creates the variable if one with the requested  
name does not exist.
```

```
Name      : Remove-Variable  
Category  : Cmdlet  
Synopsis  : Deletes a variable and its value.
```

```
Name      : Clear-Variable  
Category  : Cmdlet  
Synopsis  : Deletes the value of a variable.
```

Working with variables

1. Open the Windows PowerShell console.
2. Use the *Set-Location* cmdlet to set the working location to the Variable PS drive. Use the *sl* alias to reduce typing needs. This command is shown here:

```
SL variable:\
```

3. Produce a complete listing of all the variables currently defined in Windows PowerShell. To do this, use the *Get-ChildItem* cmdlet. You can use the alias *gci* to produce this list. The command is shown here:

```
Get-ChildItem
```

- The resulting list is jumbled. Press the up arrow key to retrieve the *Get-ChildItem* command, and pipeline the resulting object into the *Sort-Object* cmdlet. Sort on the *name* property. This command is shown here:

```
Get-ChildItem | Sort Name
```

The output from the previous command is shown here:

Name	Value
----	-----
\$	variable:
?	True
^	s1
args	{}
ConfirmPreference	High
ConsoleFileName	
DebugPreference	SilentlyContinue
Error	{Failed to update Help for the module(s) 'Schedule...
ErrorActionPreference	Continue
ErrorView	NormalView
ExecutionContext	System.Management.Automation.EngineIntrinsics
false	False
FormatEnumerationLimit	4
HOME	C:\Users\administrator
Host	System.Management.Automation.Internal.Host.Interna...
input	System.Collections.ArrayList+ArrayListEnumeratorSi...
MaximumAliasCount	4096
MaximumDriveCount	4096
MaximumErrorCount	256
MaximumFunctionCount	4096
MaximumHistoryCount	4096
MaximumVariableCount	4096
MyInvocation	System.Management.Automation.InvocationInfo
NestedPromptLevel	0
null	
OutputEncoding	System.Text.AsciiEncoding
PID	3308
PROFILE	C:\Users\administrator\Documents\WindowsPowerShell...
ProgressPreference	Continue
PSBoundParameters	{}
PSCommandPath	
PSCulture	en-US
PSDefaultParameterValues	{}
PSEmailServer	
PSHOME	C:\Windows\System32\WindowsPowerShell\v1.0
PSScriptRoot	
PSSessionApplicationName	wsman
PSSessionConfigurationName	http://schemas.microsoft.com/powershell/Microsoft...
PSSessionOption	System.Management.Automation.Remoting.PSSessionOption
PSUICulture	en-US
PSVersionTable	{PSVersion, WSManStackVersion, SerializationVersio...
PWD	Variable:\

```

ShellId           Microsoft.PowerShell
StackTrace        at System.Management.Automation.CommandDiscover...
true              True
VerbosePreference SilentlyContinue
WarningPreference Continue
WhatIfPreference  False

```

5. Use the *Get-Variable* cmdlet to retrieve a specific variable. Use the *ShellId* variable. You can use tab completion to speed up typing. The command is shown here:

```
Get-Variable ShellId
```

6. Press the up arrow key to retrieve the previous *Get-Variable ShellId* command. Pipeline the object returned into a *Format-List* cmdlet and return all properties. This is shown here:

```
Get-Variable ShellId | Format-List *
```

The resulting output includes the description of the variable, value, and other information shown here:

```

PSPath           : Microsoft.PowerShell.Core\Variable::shellid
PSDrive          : Variable
PSProvider       : Microsoft.PowerShell.Core\Variable
PSIsContainer    : False
Name             : ShellId
Description      : The ShellID identifies the current shell. This is used by
                  #Requires.
Value            : Microsoft.PowerShell
Visibility       : Public
Module           :
ModuleName       :
Options          : Constant, AllScope
Attributes       : {}

```

7. Create a new variable called *administrator*. To do this, use the *New-Variable* cmdlet. This command is shown here:

```
New-Variable administrator
```

8. Use the *Get-Variable* cmdlet to retrieve the new *administrator* variable. This command is shown here:

```
Get-Variable administrator
```

The resulting output is shown here. Notice that there is no value for the variable.

```

Name              Value
----              -
administrator

```

9. Assign a value to the new administrator variable. To do this, use the *Set-Variable* cmdlet. Specify the *administrator* variable name, and supply your given name as the value for the variable. This command is shown here:

```
Set-Variable administrator -value mred
```

10. Press the up arrow key one time to retrieve the previous *Get-Variable administrator* command. This command is shown here:

```
Get-Variable administrator
```

The output displays both the variable name and the value associated with the variable. This is shown here:

Name	Value
-----	-----
administrator	mred

11. Use the *Remove-Variable* cmdlet to remove the administrator variable you previously created. This command is shown here:

```
Remove-Variable administrator
```

You could also use the *Del* alias, as follows:

```
Del variable:administrator
```

12. Press the up arrow key one time to retrieve the previous *Get-Variable administrator* command. This command is shown here:

```
Get-Variable administrator
```

The variable is deleted. The resulting output is shown here:

```
Get-Variable : Cannot find a variable with name 'administrator'.  
At line:1 char:13  
+ Get-Variable <<<< administrator
```

This concludes this procedure.

Exploring PowerShell providers: step-by-step exercises

In this exercise, you'll explore the use of the certificate provider in Windows PowerShell. You will navigate the certificate provider by using the same types of commands used with the file system. You will then explore the environment provider by using the same methodology.

Exploring the certificate provider

1. Open the Windows PowerShell console.
2. Obtain a listing of all the properties available for use with the *Get-ChildItem* cmdlet by piping the results into the *Get-Member* cmdlet. To filter out only the properties, pipeline the results into a *Where-Object* cmdlet and specify the *membertype* to be equal to *property*. This command is shown here:

```
Get-ChildItem |Get-Member | Where-Object {$_.membertype -eq "property"}
```

3. Set your location to the Certificate drive. To identify the Certificate drive, use the *Get-PSDrive* cmdlet. Use the *Where-Object* cmdlet and filter on names that begin with the letter c. This is shown here:

```
Get-PSDrive |where name -like "c*"
```

The results of this command are shown here:

Name	Used (GB)	Free (GB)	Provider	Root
----	-----	-----	-----	----
C	110.38	38.33	FileSystem	C:\
Cert			Certificate	\

4. Use the *Set-Location* cmdlet to change to the Certificate drive:

```
SI cert:\
```

5. Use the *Get-ChildItem* cmdlet to produce a listing of all the certificates on the machine:

```
GCI
```

The output from the previous command is shown here:

```
Location : CurrentUser
StoreNames : {?, UserDS, AuthRoot, CA...}

Location : LocalMachine
StoreNames : {?, AuthRoot, CA, AddressBook...}
```

6. The listing seems somewhat incomplete. To determine whether there are additional certificates installed on the machine, use the *Get-ChildItem* cmdlet again, but this time specify the *-recurse* argument. Modify the previous command by using the up arrow key. The command is shown here:

```
GCI -recurse
```

7. The output from the previous command seems to take a long time to run and produces hundreds of lines of output. To make the listing more readable, pipe the output to a text file, and then open the file in Notepad. The command to do this is shown here:

```
GCI -recurse >C:\a.txt;notepad.exe a.txt
```

This concludes this step-by-step exercise.

In the following exercise, you'll work with the Windows PowerShell environment provider.

Examining the environment provider

1. Open the Windows PowerShell console.
2. Use the *New-PSDrive* cmdlet to create a drive mapping to the alias provider. The name of the new PS drive will be *al*. The *-PSProvider* parameter is *alias*, and the root will be dot (.). This command is shown here:

```
New-PSDrive -name al -PSProvider alias -Root .
```

3. Change your working location to the new PS drive you called *al*. To do this, use the *sl* alias for the *Set-Location* cmdlet. This is shown here:

```
SL al:\
```

4. Use the *gci* alias for the *Get-ChildItem* cmdlet, and pipeline the resulting object into the *Sort-Object* cmdlet by using the *sort* alias. Supply *name* as the property to sort on. This command is shown here:

```
GCI | Sort -property name
```

5. Press the up arrow key to retrieve the previous *gci | sort -property name* command, and modify it to use a *Where-Object* cmdlet to return aliases only when the name begins with a letter after *t* in the alphabet. Use the *where* alias to avoid typing the entire name of the cmdlet. The resulting command is shown here:

```
GCI | sort -property name | Where Name -gt "t"
```

6. Change your location back to drive C. To do this, use the *sl* alias and supply the *C:* argument. This is shown here:

```
SL C:\
```

7. Remove the PS drive mapping for *al*. To do this, use the *Remove-PSDrive* cmdlet and supply the name of the PS drive to remove. Note that this command does not take a trailing colon (:) or colon with backslash (:). The command is shown here:

```
Remove-PSDrive al
```


8. Use the *Get-PSDrive* cmdlet to ensure the al drive has been removed. This is shown here:

```
Get-PSDrive
```

9. Use the *Get-Item* cmdlet to obtain a listing of all the environment variables. Use the *-path* argument and supply *env:* as the value. This is shown here:

```
Get-Item -path env:\
```

10. Press the up arrow key to retrieve the previous command and pipeline the resulting object into the *Get-Member* cmdlet. This is shown here:

```
Get-Item -path env:\ | Get-Member
```

The results from the previous command are shown here:

```
TypeName: System.Collections.Generic.Dictionary'2+ValueCollection[[System.
String, mscorlib, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e0
89],[System.Collections.DictionaryEntry, mscorlib, Version=2.0.0.0, Culture=
neutral, PublicKeyToken=b77a5c561934e089]]
```

Name	MemberType	Definition
CopyTo	Method	System.Void CopyTo(DictionaryEntry[] array, Int32...
Equals	Method	System.Boolean Equals(Object obj)
GetEnumerator	Method	System.Collections.Generic.Dictionary'2+ValueColl...
GetHashCode	Method	System.Int32 GetHashCode()
GetType	Method	System.Type GetType()
get_Count	Method	System.Int32 get_Count()
ToString	Method	System.String ToString()
PSDrive	NoteProperty	System.Management.Automation.PSDriveInfo PSDrive=Env
PSIsContainer	NoteProperty	System.Boolean PSIsContainer=True
PSPath	NoteProperty	System.String PSPath=Microsoft.PowerShell.Core\En...
PSProvider	NoteProperty	System.Management.Automation.ProviderInfo PSProvi...
Count	Property	System.Int32 Count {get;}

11. Press the up arrow key twice to return to the *Get-Item -path env:* command. Use the Home key to move your insertion point to the beginning of the line. Add a variable called *\$objEnv* and use it to hold the object returned by the *Get-Item -path env:* command. The completed command is shown here:

```
$objEnv=Get-Item -path env:\
```

12. From the listing of members of the environment object, find the count property. Use this property to print out the total number of environment variables. As you type *\$o*, try to use tab completion to avoid typing. Also try to use tab completion as you type the *c* in *count*. The completed command is shown here:

```
$objEnv.Count
```

- 13.** Examine the methods of the object returned by *Get-Item -path env:*. Notice there is a *Get_Count* method. Let's use that method. The code is shown here:

```
$objEnv.Get_count
```

When this code is executed, however, the results define the method rather than execute the *Get_Count* method. These results are shown here:

```
MemberType           : Method
OverloadDefinitions  : {System.Int32 get_Count()}
TypeNameOfValue      : System.Management.Automation.PSMethod
Value                : System.Int32 get_Count()
Name                 : get_Count
IsInstance           : True
```

- 14.** To retrieve the actual number of environment variables, you need to use empty parentheses at the end of the method. This is shown here:

```
$objEnv.Get_count()
```

- 15.** If you want to know exactly what type of object is contained in the *\$objEnv* variable, you can use the *GetType* method, as shown here:

```
$objEnv.GetType()
```

This command returns the results shown here:

```
IsPublic IsSerial Name                                     BaseType
-----
False    True     ValueCollection                                         System.Object
```

This concludes this exercise.

Chapter 3 quick reference

To	Do this
Produce a listing of all variables defined in a Windows PowerShell session	Use the <i>Set-Location</i> cmdlet to change location to the Variable PS drive, and then use the <i>Get-ChildItem</i> cmdlet.
Obtain a listing of all the aliases	Use the <i>Set-Location</i> cmdlet to change location to the Alias PS drive, and then use the <i>Get-ChildItem</i> cmdlet to produce a listing of aliases. Pipeline the resulting object into the <i>Where-Object</i> cmdlet and filter on the <i>name</i> property for the appropriate value.
Delete a directory that is empty	Use the <i>Remove-Item</i> cmdlet and supply the name of the directory.
Delete a directory that contains other items	Use the <i>Remove-Item</i> cmdlet and supply the name of the directory and specify the <i>-recurse</i> argument.
Create a new text file	Use the <i>New-Item</i> cmdlet and specify the <i>-path</i> argument for the directory location. Supply the <i>-name</i> argument and specify the <i>-type</i> argument as <i>file</i> . Example: <i>New-Item -path C:\Mytest -name Myfile.txt -type file</i> .
Obtain a listing of registry keys from a registry hive	Use the <i>Get-ChildItem</i> cmdlet and specify the appropriate PS drive name for the <i>-path</i> argument. Complete the path with the appropriate registry path. Example: <i>gci -path HKLM:\software</i>
Obtain a listing of all functions on the system	Use the <i>Get-ChildItem</i> cmdlet and supply the PS drive name of <i>function:\</i> to the <i>-path</i> argument. Example: <i>gci -path function:\</i>

Index

Symbols

- \$\$ variable, 142
- \$acl variable, 362
- \$args variable, 139, 142, 211, 213
- \$aryElement variable, 413
- \$aryLog variable, 554, 556
- \$aryServer variable, 569
- \$aryText array, 413
- \$aryText variable, 413, 416
- \$aryUsers variable, 566, 567
- \$ary variable, 151, 154, 158
- \$bios variable, 354
- \$caps array, 153
- \$caption variable, 505
- \$_ character, 75
- \$choiceRTN variable, 505
- \$class variable, 525
- \$clsID variable, 520
- \$cn variable, 344, 464
- \$colDrives variable, 62
- \$colPrinters variable, 62
- \$computerName variable, 62, 502, 503
- \$confirmPreference variable, 216
- \$constASCII variable, 324
- \$credential variable, 341, 444, 464
- \$cred variable, 118, 127
- \$dc1 variable, 116
- \$DebugPreference variable, 465
- \$(dollar sign) character, 141
- \$driveData variable, 187, 189
- \$drives hash table, 527
- \$dteDiff variable, 329
- \$dteEnd variable, 329
- \$dteMaxAge variable, 568
- \$dteStart variable, 329
- \$env:psmodulepath variable, 222
- \$ErrorActionPreference variable, 391, 392, 524, 525, 623
- \$error.clear() method, 391
- \$error variable, 142, 191, 389, 390, 392, 624
- \$ExecutionContext variable, 142
- \$false variable, 142
- \$foreach variable, 142
- \$FormatEnumerationLimit value, 381
- \$formatEnumeration variable, 225
- \$help parameter, 184
- \$HOME variable, 142
- \$host variable, 97, 142
- \$input variable, 142, 202, 594
- \$intGroupType variable, 394, 395
- \$intSize variable, 568, 570
- \$intUsers variable, 415
- \$i++ operator, 415
- \$i++ syntax, 149
- \$item variable, 264
- \$i variable, 143, 148, 152, 328, 390, 415, 547, 566
- \$LastExitCode variable, 142
- \$logon variable, 374
- \$Match variable, 142
- \$MaximumHistoryCount variable, 594
- \$message variable, 505
- \$modulepath variable, 233
- \$month parameter, 206
- \$MyInvocation variable, 142
- \$namespace variable, 524, 525
- \$newAry variable, 567
- \$noun variable, 507
- \$null variable, 142
- \$num variable, 477, 478, 485, 486, 487, 490
- \$obj1 variable, 529, 530
- \$objADSI variable, 384, 413, 415
- \$objDisk variable, 313
- \$objEnv variable, 104, 105
- \$objGroup variable, 395

\$objOU variable

- \$objOU variable, 384
- \$objUser variable, 395, 415
- \$objWMI Services variable, 322, 328
- \$objWMI variable, 631
- \$OFS variable, 142
- \$oldVerbosePreference variable, 516, 521
- \$oupath variable, 435
- \$password variable, 546, 566, 568
- \$path parameter, 206, 207
- \$process variable, 138, 264, 345, 364
- \$profile variable, 268–270, 279
- \$providername variable, 518, 521
- \$provider variable, 518
- \$PSCmdlet variable, 219
- \$PSHome variable, 142, 267, 272
- \$PSModulePath variable, 232
- \$psSession variable, 353
- \$PSVersionTable variable, 225
- \$query variable, 326
- \$rtn variable, 124
- \$scriptRoot variable, 469, 470
- \$servers array, 509, 510
- \$session variable, 345, 352
- \$share variable, 365
- \$ShellID variable, 142
- \$StackTrace variable, 142
- \$strClass variable, 384, 395, 412, 413, 414, 415
- \$strComputer variable, 320, 322, 327
- \$strDatabase variable, 546, 566
- \$strDomain variable, 410, 546, 566
- \$strFile variable, 323
- \$strFname variable, 547, 567
- \$strLevel variable, 555
- \$strLname variable, 547
- \$strLogIdent variable, 555, 556
- \$strLogPath variable, 569
- \$strLog variable, 555
- \$strManager variable, 410
- \$strName variable, 142, 143, 408, 412, 415
- \$strOUName variable, 384, 413, 414
- \$strOU variable, 410, 546, 566, 567
- \$strPath variable, 142
- \$strUserName variable, 142
- \$strUserPath variable, 142
- \$strUser variable, 410, 415
- \$this variable, 142
- \$true variable, 142
- \$userDomain variable, 62
- \$userName variable, 62
- \$users variable, 443
- \$^ variable, 142
- \$_ variable, 86, 137, 142, 183, 332
- \$? variable, 142
- \$VerbosePreference variable, 210, 516, 519, 521
- \$verbose variable, 516
- \$v variable, 381
- \$wmiClass variable, 320
- \$wmiFilter variable, 320
- \$wmiNS variable, 322, 327
- \$wmiQuery variable, 322, 328
- \$wshnetwork.EnumPrinterConnections()
command, 62
- \$wshnetwork variable, 61
- \$xml variable, 563, 565
- \$year parameter, 206
- \$zip parameter, 190
- [0] syntax, 230
- & (ampersand) character, 12
- * (asterisk) wildcard operator, 7, 17, 21, 68, 293, 309, 442
- ' (backtick) character, 137, 480, 628
- \ (backward slash), 68
- ! CALL prefix, 470
- ^ character, 291
- __CLASS property, 188
- : (colon), 68
- computername parameter, 108, 118, 124, 246
- { } (curly brackets), missing, 177–178
- __DERIVATION property, 188
- __DYNASTY property, 188
- = (equal) character, 162, 320
- = (equal sign), 162, 320
- ! (exclamation mark), 470
- Force parameter, 459
- __GENUS property, 188
- ' (grave accent) character, 143, 319, 321
- > (greater-than) symbol, 320
- < (less-than) symbol, 320
- __Namespace class, 287
- __NAMESPACE property, 188
- `n escape sequence, 328
- __PATH property, 188
- | (pipe) character, 24, 324, 556
- + (plus symbol), 137, 143
- property argument, 77
- __PROPERTY_COUNT property, 188
- __provider class, 289
- ? (question mark), 291
- >> (redirect-and-append arrow), 6

- > (redirection arrow), 6, 318
- __RELSPATH property, 188
- #requires statement, 234
- __SERVER property, 188
- ! SET keyword, 470
- . (shortcut dot), 320
- ' (single quote) character, 92
- __SUPERCLASS property, 188
- %windir% variable, 51

A

- abstract qualifier, 371
- abstract WMI classes, 370
- access control lists (ACLs), 90, 362
- Access Denied error, 287, 463, 464
- Access property, 187
- account lockout policy, checking, 430
- accounts, user
 - creating, 395–396
 - deleting, 411–412
- AccountsWithNoRequiredPassword.ps1 script, 132
- ACLs (access control lists), 90, 362
- action parameter, 488
- Active Directory
 - cmdlets for
 - creating users using, 435–436
 - discovering information about forest and domain, 428–431
 - finding information about domain controller using, 424–428
 - committing changes to, 389
 - finding unused user accounts using, 440–442
 - installing RSAT for, 420
 - locked-out users, unlocking, 436–437
 - managing users using, 432–434
 - objects in
 - ADSI providers and, 385–387
 - binding and, 388
 - connecting to, 388
 - error handling, adding, 392
 - errors, 389–392
 - LDAP naming convention and, 387–388
 - organizational units, creating, 383–384, 413–414
 - overview, 383
 - objects, updating using Active Directory module, 443–444

- querying, 590
- renaming sites, 431–432
- users
 - address information, exposing, 400–401
 - computer account, 395–396
 - creating, 435–436
 - deleting, 411–412
 - disabled, finding, 438–439
 - finding and unlocking user accounts, 436–437
 - general user information, 398–399
 - groups, 394–395
 - managing, 432–434
 - multiple users, creating, 408–409
 - multivalued users, creating, 414–417
 - organizational settings, modifying, 409–411
 - overview, 393–394
 - passwords, changing, 444–445
 - profile settings, modifying, 403–405
 - properties, modifying, 397–398
 - telephone settings, modifying, 405–407
 - unused user accounts, finding, 440–442
 - user account control, 396–397
- Active Directory Domain Services. *See* AD DS
- Active Directory Management Gateway Service (ADMGS), 419
- Active Directory Migration Tool (ADMT), 385
- Active Directory module
 - automatic loading of, 421
 - connecting to server containing, 421–422
 - default module locations, 421
 - finding FSMO role holders, 422–427
 - importing via Windows PowerShell profile, 436
 - installing, 419–420
 - overview, 419
 - updating Active Directory objects using, 443–444
 - verifying, 421
- Active Directory Service Interfaces (ADSI), 383, 385–387
- ActiveX Data Object (ADO), 153
- Add cmdlet, 583
- Add-Computer cmdlet, 571
- Add-Content cmdlet, 84, 571
- Add Criteria button, 33
- Add-Member cmdlet, 571
- AD_Doc.txt file, 431, 462
- AddOne filter, 202
- AddOne function, 490

Add-Printer cmdlet

- Add-Printer cmdlet, 571
- Add-PrinterDriver cmdlet, 571
- Add-PrinterPort cmdlet, 571
- Add-RegistryValue function, 467, 468–469, 470
- address information, 400–401
- Address tab, Active Directory Users and Computers, 401
- AD DS (Active Directory Domain Services)
 - AD DS Tool, 385
 - deploying
 - domain controller, adding to domain, 453–455
 - domain controller, adding to new forest, 458–459
 - domain controller prerequisites, installing, 457–458
 - features, adding, 448
 - forests, creating, 452–453
 - infrastructure prerequisites, 447
 - IP address assignment, 448
 - read-only domain controller, adding, 455–457
 - renaming computer, 448
 - restarting computer, 449
 - role-based prerequisites, 448
 - script execution policy, setting, 447
 - verification steps, 449–450
 - tools installation, 448
- ADSDeployment module, 452, 454, 456, 459
- AddTwo function, 490
- Add-Type cmdlet, 571
- Add-WindowsFeature cmdlet, 386, 420, 448, 455, 458
- AD LDS Tool, 385
- ADMGS (Active Directory Management Gateway Service), 419
- admin environment variable, 78, 79
- Administrator Audit Logging feature, 557
- administrator variable, 100
- ADMT (Active Directory Migration Tool), 385
- ADO (ActiveX Data Object), 153
- ADSI (Active Directory Service Interfaces), 383, 385–387
- ADsPath, 384
- ADS_UF_ACCOUNTDISABLE flag, 397
- ADS_UF_DONT_EXPIRE_PASSWORD flag, 397
- ADS_UF_DONT_REQUIRE_PREAUTH flag, 397
- ADS_UF_ENCRYPTED_TEXT_PASSWORD_ALLOWED flag, 397
- ADS_UF_HOMEDIR_REQUIRED flag, 397
- ADS_UF_INTERDOMAIN_TRUST_ACCOUNT flag, 397
- ADS_UF_LOCKOUT flag, 397
- ADS_UF_MNS_LOGON_ACCOUNT flag, 397
- ADS_UF_NORMAL_ACCOUNT flag, 397
- ADS_UF_NOT_DELEGATED flag, 397
- ADS_UF_PASSWD_CANT_CHANGE flag, 397
- ADS_UF_PASSWD_NOTREQD flag, 396, 397
- ADS_UF_PASSWORD_EXPIRED flag, 397
- ADS_UF_SCRIPT flag, 397
- ADS_UF_SERVER_TRUST_ACCOUNT flag, 397
- ADS_UF_SMARTCARD_REQUIRED flag, 397
- ADS_UF_TEMP_DUPLICATE_ACCOUNT flag, 397
- ADS_UF_TRUSTED_FOR_DELEGATION flag, 397
- ADS_UF_TRUSTED_TO_AUTHENTICATE_FOR_DELEGATION flag, 397
- ADS_UF_USE_DES_KEY_ONLY flag, 397
- ADS_UF_WORKSTATION_TRUST_ACCOUNT flag, 396, 397
- alias argument, 567
- aliases, 489, 626–627
 - creating for cmdlets, 19
 - finding all for object, 59
 - finding for cmdlets, 150–151
 - provider for, 66–68
 - setting, 246
- AllowMaximum property, 315
- AllowPasswordReplicationAccountName parameter, 456
- AllSigned execution policy, 134
- All Users, All Hosts profile, 275–276
- AllUsersCurrentHost profile, 269
- alphabetical sorting, 77
- ampersand (&) character, 12
- a parameter, 212
- AppLocker module, 580
- Appx module, 580
- ArgumentList block, 263
- arguments, for cmdlets, 12
- [array] alias, 146, 190
- Array function, 151
- array objects, 54
- arrays
 - using -contains operator to examine contents of, 507–509
 - creating, 589
 - indexing, 377
- ASCII values, casting to, 152–153
- asjob parameter, 350, 353
- asplaintext argument, 545, 566
- assignment operators, 163

association classes, WMI, 370, 373–378
 asterisk (*) wildcard operator, 7, 17, 21, 68, 293, 309, 442
 ast-write-time property, 30
 Attributes property, 82
 audit logging (Exchange Server 2010), 557–561
 -autosize argument, 313, 327, 331
 -AutoSize parameter, 27
 Availability property, 187

B

Backspace key, 38
 backtick (`) character, 137, 480, 628
 backup domain controllers (BDCs), 385
 backward slash (\), 68
 basename property, 230
 BDCs (backup domain controllers), 385
 Begin block, 199, 205
 BestPractices module, 580
 binary SD format, 362
 binding, 388
 BIOS information, 115, 308–311, 371
 bios pattern, 291
 BitsTransfer module, 236, 580
 BlockSize property, 187
 bogus module, 234
 [bool] alias, 146, 190
 boundary-checking function, 526–527
 BranchCache module, 579
 breakpoints

- deleting, 494
- enabling and disabling, 494
- ID number, 494
- listing, 492–493
- purpose of, 483
- responding to, 490–492
- script location and, 485
- setting
 - on commands, 489–490
 - on line number, 483–484
 - on variables, 485–489
 - overview, 483
- vs. stepping functionality, 483
- storage location, 492

- Break statement, 160, 167
- business logic
- encapsulating with functions, 194–196
- program logic vs., 194

BusinessLogicDemo.ps1 script, 194
 Bypass execution policy, 134
 bypass option, 134, 136, 238
 [byte] alias, 146, 190

C

canonical aliases, 626–627
 Caption property, 187, 315
 Case Else expression, 165
 casting, 152–153
 Catch block, 529. *See also* Try...Catch...Finally blocks
 CategoryInfo property, 389
 C attribute, 388
 -ccontains operator, 507
 cd alias, 67
 cd .. command, 7
 Certificate drive, 102
 certificates

- deleting, 74
- finding expired, 75
- listing, 69–73
- provider for, 68
- searching, 74–75
- viewing properties of, 72–73

- Certificates Microsoft Management Console (MMC), 69
 Certmgr.msc file, 73–74
 [char] alias, 146, 190
 char data type, 153
 chdir alias, 67
 Check-AllowedValue function, 526
 Checkpoint cmdlet, 584
 Checkpoint-Computer cmdlet, 571
 Chkdsk method, 187
 ChoiceDescription class, 505
 choices, limiting. *See* limiting choices
 cimclassname property, 380, 381
 cimclassqualifiers property, 380
 CIM cmdlets
- filtering classes by qualifier, 369–371
- finding WMI class methods, 368–369
- module for, 580
- overview, 367
- retrieving associated WMI classes, 381–382
- using -classname parameter, 367–368
- video classes, 380–381
- CIM (Common Information Model), 108, 112, 343–344, 579. *See also* CIM cmdlets

CIM_LogicalDevice class

- CIM_LogicalDevice class, 362
- CIM_UnitaryComputerSystem class, 290
- CIMWin32WMI provider, 516
- class argument, 321
- Class box, 253
- classes
 - in WMI, 289–293
 - querying WMI, 293–296
 - retrieving data from specific instances of, 319–320
 - retrieving every property from every instance of, 314
 - retrieving specific properties from, 316
- classname parameter, 348, 367–368, 368, 372
- class parameter, 264, 523
- __CLASS property, 517
- Clear cmdlet, 583
- Clear-Content cmdlet, 571
- Clear-EventLog cmdlet, 571
- Clear-Host cmdlet, 60, 478
- Clear-Item cmdlet, 571
- Clear-ItemProperty cmdlet, 571
- clear method, 392
- Clear-Variable cmdlet, 571
- ClientLoadableCLSID property, 517
- cls command, 21
- CLSID property, 517, 519
- CMD (command) shell, 1, 76
- [cmdletbinding] attribute
 - adding -confirm support, 215–216
 - adding -whatif support to function, 214–215
 - enabling for functions, 210
 - for functions, checking parameters automatically, 211–214
 - overview, 209, 209–210
 - specifying default parameter set, 216–217
 - verbose switch for, 210–211
- [CmdletBinding()] attribute, 464, 465
- CmdletInfo object, 540
- cmdlets. *See also* CIM cmdlets
 - Active Directory
 - creating users using, 435–436
 - finding information about domain controller using, 424–428
 - finding locked out users using, 436
 - finding unused user accounts using, 440–442
 - managing users using, 432–434
 - defined, 3
 - descriptions of all, 571–578
 - displaying graphical command picker of, 52
 - execution of
 - confirming, 8
 - controlling, 7
 - finding aliases for, 150–151
 - for working with event logs, 587
 - most important, 587
 - names of, 626–627
 - naming, 3, 54–56, 583–586
 - verb distribution, 55–56
 - verb grouping for, 54–55
 - number of on installation, 587
 - options for, 12
 - overview, 3, 23–24
 - searching for using wildcards, 36–39, 43
 - suspending execution of, 9
 - using Get-Command cmdlet for, 36–39, 43
 - verbs for, 174
 - with Exchange Server 2010, 539–540
- cmdlets parameter, 559
- cn alias, 124, 247
- CN attribute, 388
- cn parameter, 465
- code formatting. *See* formatting code
- code, reusing, 178–179
- colon (:), using after PS drive name, 68
- column heading buttons, 32
- columns argument, 28
- command (CMD) shell, 1
- commandlets. *See* cmdlets
- command-line input, 501
- command-line parameter, 502–503
- command-line utilities
 - exercises using, 20–21
 - ipconfig command, 5
 - multiple, running, 6
 - overview, 4, 5
- command parameter, 489
- commands
 - most powerful, 588
 - setting breakpoints on, 489–490
 - whether completed successfully, 592
- Commands add-on
 - overview, 252–256
 - turning off, 256
 - using with script pane, 255
- command window, prompt for, 76
- comments, 179, 627–628
- Common Information Model. *See* CIM
- comobject parameter, 50, 61, 62
- Compare cmdlet, 584

- Compare-Object cmdlet, 571
 - comparison operators, 162–163
 - compatibility aliases, 626
 - Complete cmdlet, 584
 - Complete-Transaction cmdlet, 571
 - Compressed property, 187
 - computer account, 395–396
 - computer connectivity, identifying, 506
 - computername parameter, 182, 293, 344
 - Concurrency property, 517
 - ConfigManagerErrorCode property, 187
 - ConfigManagerUserConfig property, 187
 - ConfigurationNamingContext property, 431
 - ConfigureTransportLogging.ps1 script, 557
 - Confirm:\$false command, 434
 - confirm argument, 8–10
 - Confirm cmdlet, 585
 - confirmimpact property, 216
 - ConfirmingExecutionOfCmdlets.txt file, 8
 - confirm parameter, 12, 438, 629
 - confirm switch, 215–216, 437
 - Connect cmdlet, 584
 - connectivity. *See* computer connectivity
 - Connect-WSMan cmdlet, 571
 - console, launch options for, 11
 - ConsoleProfile variable, 280
 - console window
 - copying in, 72
 - quotation marks in, 133
 - constants, 587, 631
 - compared with variables, 146
 - creating, 170
 - creating in scripts, 146
 - using, 146–147
 - contains operator, 504, 594
 - using to examine contents of array, 507–509
 - using to test for properties, 509–511
 - Continue command, 491
 - Continue statement, 191
 - Control Properties dialog box, 285
 - ConversionFunctions.ps1 script, 179
 - ConversionModuleV6 module, 237
 - Convert cmdlet, 585
 - ConvertFrom cmdlet, 584
 - ConvertFrom-Csv cmdlet, 571
 - ConvertFrom-DateTime method, 188
 - ConvertFrom-Json cmdlet, 571
 - ConvertFrom-StringData cmdlet, 571
 - Convert-Path cmdlet, 571
 - ConvertTo cmdlet, 584
 - ConvertTo-Csv cmdlet, 572
 - ConvertTo-DateTime method, 188
 - ConvertTo-Html cmdlet, 572
 - ConvertTo-Json cmdlet, 572
 - ConvertToMeters.ps1 script, 178
 - ConvertTo-SecureString cmdlet, 435, 545, 566
 - ConvertTo-Xml cmdlet, 572
 - Copy button, Commands add-on, 255
 - Copy cmdlet, 584
 - copying from PowerShell window, 72
 - Copy-Item cmdlet, 230, 279, 572
 - Copy-ItemProperty cmdlet, 572
 - Copy-Module function, 229, 231
 - Copy-Modules.ps1 script, 229, 231, 237, 241, 244
 - counting backward, 595
 - count parameter, 506
 - count property, 104, 125, 212, 389
 - CountryCode attribute, 401
 - country codes, 401–402
 - CPU (central processing unit), listing processes using
 - CPU time criteria, 34
 - CreateDnsDelegation parameter, 459
 - CreateShortCutToPowerShell.vbs script, 141
 - CreatingFoldersAndFiles.txt file, 80
 - CreationClassName property, 187
 - CreationTime property, 82
 - CreationTimeUtc property, 82
 - credentials
 - credential parameter, 109, 110, 591
 - for remote connection, 339–342
 - CRSS process, 216
 - Ctrl+J shortcut, 257
 - Ctrl+N shortcut, 254, 258
 - Ctrl+V shortcut, 255, 258
 - curly brackets ({}), missing, 177–178
 - Current Host profile, 268
 - current property, 202
 - CurrentUserCurrentHost property, 269, 270
 - Current User profile, 268
 - CurrentUser scope, 134
- ## D
- DatabasePath parameter, 459
 - data types, incorrect, 523–525
 - date, obtaining current, 75
 - DateTime object, 205
 - [DBG] prefix, 495
 - DC attribute, 388

DDL (dynamic-link library) file

- DDL (dynamic-link library) file, 66
- Debug cmdlet, 585
- debugging. *See also* errors
 - cmdlets for, list of, 483
 - functions, 495–496
 - scripts, using breakpoints
 - deleting breakpoints, 494
 - enabling and disabling breakpoints, 494
 - exercise, 496–498
 - listing breakpoints, 492–493
 - responding to breakpoints, 490–492
 - setting on commands, 489–490
 - setting on line number, 483–484
 - setting on variables, 485–489
 - using Set-PSDebug cmdlet
 - overview, 467
 - script-level tracing, 467–471
 - stepping through script, 471–479
 - strict mode, enabling
 - overview, 479
 - using Set-PSDebug -Strict, 479–480
 - using Set-StrictMode cmdlet, 481–482
- debug parameter, 12, 465
- Debug-Process cmdlet, 572
- [decimal] alias, 146, 190
- DefaultDisplayPropertySet configuration, 294
- DEFAULT IMPERSONATION LEVEL key, 307
- DefaultMachineName property, 517
- DefaultParameterSetName property, 216, 217
- default property, 89, 90
- default value, setting for registry keys, 95
- definition attribute, 86
- definition parameter, 150
- Delete method, 412
- DeleteUser.ps1 script, 412
- deleting
 - breakpoints, 494
 - users, 411–412
- DemoAddOneFilter.ps1 script, 203
- DemoAddOneR2Function.ps1 script, 203
- DemoBreakFor.ps1 script, 160
- DemoDoUntil.vbs script, 154
- DemoDoWhile.ps1 script, 151
- DemoDoWhile.vbs script, 151
- DemoExitFor.ps1 script, 160
- DemoExitFor.vbs script, 160
- DemoForEachNext.vbs script, 158
- DemoForEach.ps1 scrip, 158
- DemoForLoop.ps1 script, 156, 157
- DemoForLoop.vbs script, 156
- DemoForWithoutInitOrRepeat.ps1 script, 156, 157
- demofIfElseIfElse.ps1 script, 164
- DemofIfElse.vbs script, 163
- Demof.ps1 script, 161
- Demof.vbs script, 162
- DemoQuitFor.vbs script, 161
- DemoSelectCase.vbs script, 164, 166
- DemoSwitchArrayBreak.ps1 script, 167
- DemoSwitchArray.ps1 scrip, 167
- DemoSwitchMultiMatch.ps1 script, 166
- DemoTrapSystemException.ps1 script, 191
- DemoWhileLessThan.ps1 script, 148, 149
- dependencies, checking for modules, 234–236
- deploying
 - AD DS (Active Directory Domain Services)
 - domain controller, adding to domain, 453–455
 - domain controller, adding to new forest, 458–459
 - domain controller prerequisites, installing, 457–458
 - features, adding, 448
 - forest, creating, 452–453
 - infrastructure prerequisites, 447
 - IP address assignment, 448
 - read-only domain controller, adding, 455–457
 - renaming computer, 448
 - restarting computer, 449
 - role-based prerequisites, 448
 - script execution policy, setting, 447
 - verification steps, 449–450
 - PowerShell to enterprise systems, 4
- deprecated qualifier, 370
- __DERIVATION property, 517
- Descending parameter, 35
- description parameter, 187, 260, 315, 627
- design considerations, analyzing before
 - development, 94
- detailed argument, 21
- DeviceID property, 187
- dir alias, 88
- DirectAccessClientComponents module, 580
- directories
 - creating, 82–83
 - listing contents of, 81
 - listing contents with Get-ChildItem cmdlet, 24–26
 - formatting with Format-List cmdlet, 26
 - formatting with Format-Table cmdlet, 29
 - formatting with Format-Wide cmdlet, 27–29
 - properties for, 81–82

DirectoryInfo object, 44
 DirectoryListWithArguments.ps1 script, 131–132
 DirectoryName property, 82
 Directory property, 82
 Directory Restore Password prompt, 456
 Disable cmdlet, 583
 Disable-ComputerRestore cmdlet, 572
 Disable-PSBreakpoint cmdlet, 483, 494, 572
 Disable-WSManCredSSP cmdlet, 572
 Disconnect cmdlet, 584
 Disconnect-WSMan cmdlet, 572
 -Discover switch, 424
 Diskinfo.txt file, 318
 disktype property, 146
 Dism module, 580
 Dismount cmdlet, 585
 DisplayCapitalLetters.ps1 script, 153
 displaying commands, using Show-Command cmdlet, 52
 DisplayName property, 302–303, 432
 divide-by-zero error, 492
 DivideNum function, 490, 491–492, 492
 DnsClient module, 580
 DNS Manager tool, 453
 DNS server, adding to IP configuration, 453
 DNSServerSearchOrder property, 196
 Documents and Settings\%username% folder, 141
 Do keyword, 154
 dollar sign (\$), 141, 189
 domain controller

- adding to domain, 453–455
- adding to new forest, 458–459
- checking, 430
- prerequisites, installing, 457–458

 -DomainMode parameter, 459
 -DomainName parameter, 459
 DomainNamingMaster role, 425
 -DomainNetbiosName parameter, 459
 domain password policy, checking, 429
 Do statement, 152, 154
 dot-sourced functions, using, 182–184
 DotSourceScripts.ps1 script, 198
 dot-sourcing scripts, 178, 179–181, 180–181
 dotted notation, 39, 357
 [double] alias, 146, 190
 Do...Until statement, 155
 DoWhileAlwaysRuns.ps1 script, 155
 Do...While statement

- always runs once, 155
- casting and, 152–153

- in VBScript compared with in PowerShell, 151
- range operator, 152

 drives

- creating for modules, 232–233
- creating for registry, 87
- for registry, 87–88
- using WMI with, 312–314

 DriveType property, 187, 312, 314
 dynamic-link library (DLL) file, 66
 dynamic qualifier, 370, 371
 dynamic WMI classes, 370
 __DYNASTY property, 517

E

ea alias, 97, 136
 -ea parameter, 27
 echo command, 76
 -edbFilePath parameter, 551
 Else clause, 97, 163, 169, 236
 Else If clause, 163
 empty parentheses, 105
 Enable cmdlet, 583
 Enable-ComputerRestore cmdlet, 572
 Enabled property, 517
 Enable-Mailbox cmdlet, 544, 559
 Enable-PSBreakpoint cmdlet, 483, 494, 572
 Enable-PSRemoting function, 112
 Enable-WSManCredSSP cmdlet, 572
 -enddate parameter, 559
 EndlessDoUntil.ps1 script, 155
 End parameter, 201
 Enter cmdlet, 585
 Enter in Windows PowerShell option, 71
 enterprise systems, deploying PowerShell to, 4
 Enter-PSSession cmdlet, 115, 116, 127, 428, 444
 EnumNetworkDrives method, 61
 EnumPrinterConnections method, 61
 Environment PS drive, 77
 environment variables

- creating temporary, 78
- deleting, 80
- listing, 77–78
- provider for, 76
- renaming, 79
- viewing using WMI, 330–335

 -eq operator, 162
 -equals argument, 300, 304
 equal sign (=), 162, 320

error[0] variable

- error[0] variable, 389
- erroraction parameter, 136
- ErrorAction parameter, 12
- ErrorCleared property, 187
- ErrorDescription property, 187
- error handling
 - incorrect data types, 523–525
 - limiting choices
 - using -contains operator to examine contents of array, 507–509
 - using -contains operator to test for properties, 509–511
 - overview, 504
 - using PromptForChoice, 504–505, 534–535
 - using Test-Connection to identify computer connectivity, 506
- missing parameters
 - assigning value in param statement, 502–503
 - detecting missing value and assigning in script, 502
 - making parameter mandatory, 503
 - overview, 501
- missing rights
 - attempt and fail, 512
 - checking for rights and exiting gracefully, 513
 - overview, 512
- missing WMI providers, 513–523
- out-of-bounds errors
 - overview, 526
 - placing limits on parameter, 528
 - using boundary-checking function, 526–527
- using Try...Catch...Finally
 - Catch block, 529
 - catching multiple errors, 532–533
 - exercise, 536–537
 - Finally block, 529–530
- error messages
 - importance of, 136
 - using Trap keyword to avoid confusing messages, 191–192
- ErrorMethodology property, 187
- ErrorRecord class, 191
- ErrorRecord object, 532
- errors. *See also* debugging
 - command for ignoring, 589
 - logic, 466
 - run-time, 462–465
 - simple typing errors, 479–480
 - syntax, 461–462
 - terminating vs. nonterminating, 512
 - ErrorVariable parameter, 12
- escape character (\), 149, 157
- examples argument, 18, 21
- Exception property, 532
- Exchange Server 2010, 562–565
 - audit logging, 557–561
 - cmdlets with, 539–540
 - logging settings, 553–557
 - overview, 553
 - transport-logging levels, 554–557
 - mailboxes, creating
 - multiple mailboxes, 546–547
 - using Enable-Mailbox cmdlet, 543–544
 - when creating user, 544–546
 - message tracking, 568–570
 - parsing audit XML file, 562–565
 - remote servers, 540–543
 - reporting user settings, 548–550
 - storage settings
 - mailbox database, 550–552
 - overview, 550–551
 - user accounts, creating
 - exercise, 565–568
 - when creating mailbox, 544–546
- exclamation mark (!), 470
- execution policies for scripts
 - overview, 134
 - required for using profiles, 268
 - required for using snippets, 259
 - retrieving current, 135–136
 - setting, 135–136
- execution policy, restricted, 513
- execution, unwanted, preventing using While statement, 155–156
- Exists property, 82
- Exit cmdlet, 585
- exit command, 115, 128
- Exit For statement, 159
- Exit statement, 160–161
- ExpandEnvironmentStrings method, 51
- expanding strings, 148, 157
- expired certificates
 - finding, 75
 - needed for old executables, 75
- explorer filter, 34
- Export-Alias cmdlet, 572
- Export-CliXML cmdlet, 345, 563, 572
- Export cmdlet, 583
- Export-Console cmdlet, 11
- Export-Csv cmdlet, 572

exportedcommands property, 225
 Export-FormatData cmdlet, 572
 Export-ModuleMember cmdlet, 241, 248
 Export-PSSession cmdlet, 572
 Extension property, 82, 193

F

FacsimileTelephoneNumber attribute, 406
 FeatureLog.txt file, 450
 FileInfo object, 44
 -filePath argument, 323
 files
 creating, 82–83
 overwriting contents of, 85
 reading from, 84–85
 writing to, 84–85
 FileSystemObject, 150
 FileSystem property, 187
 filesystem provider, 80
 FilterHasMessage.ps1 script, 204
 Filter keyword, 196, 204
 -filter parameter, 199, 312, 326–327, 347, 372, 425, 440, 518, 589
 quotation marks used with, 318
 using to reduce number of returned WMI class instances, 378
 filters
 advantages of, 204–205
 overview, 201–203
 performance and, 203–204
 readability of, 204–205
 FilterToday.ps1 script, 205
 Finally block, of Try...Catch...Finally, 529–530
 Find and Replace feature, 622
 FindLargeDocs.ps1 script, 196
 firewall exceptions, 114
 -firstname argument, 568
 fl alias, 295
 folders
 creating, 82–83
 for user modules, 227–230
 multiple
 creating using scripts, 168–169
 deleting using scripts, 169–170
 -force parameter, 81, 82, 94, 112, 134, 269, 279, 434, 440, 545, 552
 foreach alias, 143
 Foreach alias, 489

ForEach cmdlet, 413, 585
 ForEach-Object cmdlet, 137, 159, 183, 287, 292, 381, 382, 489, 550
 foreach snippet, 264
 Foreach statement
 exiting early, 159–160
 overview, 158
 using from inside PowerShell console, 159
 ForEach statement, 443
 -foregroundcolor argument, 328
 ForEndlessLoop.ps1 script, 157
 -ForestMode parameter, 459
 forests
 adding domain controller to, 458–459
 creating, 452–453
 For keyword, 156
 Format cmdlet, 309, 584
 Format-Custom cmdlet, 572
 Format-IPOutput function, 200
 Format-List cmdlet, 26, 72, 77, 98, 143, 269, 309, 316, 321, 386, 485, 525, 549, 550, 572
 Format-NonIPOutput function, 200
 *.format.ps1xml files, 371
 Format-Table cmdlet, 29, 139, 255, 313, 318, 373, 380, 493, 564, 572
 formatting code, 628–629
 constants, 631
 functions, 629–630
 template files, 630
 formatting returned data, 189
 Format-Wide cmdlet, 572
 alias for, 68
 formatting output with, 27–29
 using, 27–29
 For...Next loop, 152
 For statement
 flexibility of, 156–157
 in VBScript compared with in PowerShell, 156
 making into infinite loop, 157–158
 FreeSpace property, 187, 189
 FSMO (Flexible Single Master Operation), 422–427
 fsutil utility, 2, 20
 ft alias, 295
 -full argument, 19, 21
 FullName property, 82, 231
 FullyQualifiedErrorId property, 389
 Function drive, 181
 FunctionGetIPDemo.ps1 script, 198
 FunctionInfo object, 540
 Function keyword, 172, 174, 177, 186, 193, 205, 279

function libraries, creating

- function libraries, creating, 178–179
- function notation, 481
- function provider, 85
- functions
 - adding help for
 - overview, 184
 - using here-string object for, 184–186
 - advantages of using, 197–198
 - as filters, 201–204
 - [cmdletbinding] attribute for, 209–210
 - adding -confirm support, 215–216
 - adding -whatif support, 214–215
 - checking parameters automatically, 211–214
 - specifying default parameter set, 216–217
 - verbose switch, 210–211
 - comments at end of, 179
 - creating, 172
 - debugging, 495–496
 - delimiting script block on, 177
 - dot-sourced, 182–184
 - enabling [cmdletbinding] attribute for, 210
 - encapsulating business logic with, 194–196
 - flexibility of, 198–199
 - formatting, 629–630
 - including in PowerShell using dot-sourcing, 180–181
 - including in scripts, 625
 - in VBScript, 171
 - listing all, 85–87
 - naming, 174–175, 628
 - parameters for
 - overview, 176
 - using more than two, 192–193
 - using two input parameters, 186–187
 - passing values to, 175
 - performance of, 203–204
 - readability of, 198
 - reusability of, 198
 - separating data and presentation activities into different functions, 199–202
 - signature of, 195
 - type constraints in, 190–191
 - using for code reuse, 178–179
 - using from imported module, 242–244
 - using Get-Help cmdlet with, 243–245
- Functions.psm1 module, 239
- fw alias, 68

G

- gal alias, 45–46
- gc alias, 150
- gci alias, 79, 85, 333
- gcm alias, 37, 238
- __GENUS property, 517
- ge operator, 162
- Get-Acl cmdlet, 362
- Get-ADDefaultDomainPasswordPolicy cmdlet, 429
- Get-ADDomain cmdlet, 429
- Get-ADDomainController cmdlet, 424, 430
- Get-ADForest cmdlet, 428
- Get-ADObject cmdlet, 425, 431
- Get-ADOrganizationalUnit cmdlet, 435
- Get-ADRootDSE cmdlet, 431
- Get-ADUser cmdlet, 435, 443, 444
- Get-Alias cmdlet, 21, 24, 150, 332, 572
- Get-AllowedComputerAndProperty.ps1 script, 511
- Get-AllowedComputer function, 508, 509, 510
- Get-ChildItem cmdlet, 20, 75, 131, 196, 231, 237, 331, 572
 - alias for, 67
 - exercises using, 59–60
 - listing certificates using, 69
 - listing directory contents with, 24–26
 - listing registry keys using, 65
- Get-Choice function, 505
- Get-CimAssociatedInstance cmdlet, 374, 377, 378, 381, 382
- Get-CimClass cmdlet, 367–368, 380, 381
- Get-CimInstance cmdlet, 183, 246, 343, 353, 371, 373, 381
- Get cmdlet, 583
- Get-Command cmdlet, 21, 36–39, 43, 56, 172, 238, 242, 421, 423, 579
- Get-Command -module <modulename> command, 225
- Get-ComputerInfo function, 241, 242
- Get-ComputerRestorePoint cmdlet, 572
- Get-Content cmdlet, 150, 177, 185, 413, 415, 462–463, 508, 563, 572, 627
- Get-ControlPanellItem cmdlet, 572
- Get_Count method., 105
- Get-Credential cmdlet, 127, 339, 444, 456, 541
- Get-Culture cmdlet, 572
- Get-Date cmdlet, 20, 329, 572
- Get-DirectoryListing function, 192, 193
- Get-DirectoryListingToday.ps1 script, 193
- Get-Discount function, 194

- Get-DiskInformation function, 527
- Get-DiskSpace.ps1 script, 189
- Get-Doc function, 196
- Get-Event cmdlet, 572
- Get-EventLog cmdlet, 573, 588
- Get-EventLogLevel cmdlet, 553, 555
- Get-EventSubscriber cmdlet, 573
- Get-ExchangeServer cmdlet, 542
- Get-ExCommand cmdlet, 539, 540, 543
- Get-ExecutionPolicy cmdlet, 135, 259, 278
- Get-FilesByDate function, 194, 205
- Get-FilesByDate.ps1 script, 207
- Get-FilesByDateV2.ps1 file, 207
- GetFolderPath method, 272
- Get-FormatData cmdlet, 573
- Get-FreeDiskSpace function, 186
- Get-FreeDiskSpace.ps1 script, 186
- GetHardDiskDetails.ps1 script, 146
- Get-Help cmdlet, 58, 68, 243, 245, 540
 - creating alias for, 19
 - examples using, 21
 - overview, 15–20
- Get-History cmdlet, 332
- Get-Host cmdlet, 573
- Get-HotFix cmdlet, 573
- GetInfoByZip method, 190
- GetIPDemoSingleFunction.ps1 script, 197
- Get-IPObjectDefaultEnabledFormatNonIPOutput.ps1 script, 200
- Get-IPObjectDefaultEnabled.ps1 script, 199
- Get-IPObject function, 199, 200
- Get-IseSnippet cmdlet, 261
- Get-Item cmdlet, 573
- Get-ItemProperty cmdlet, 89, 143, 308, 573
- Get-Job cmdlet, 121, 351
- Get-Location cmdlet, 68, 573
- Get-Mailbox cmdlet, 548
- Get-MailboxDatabase cmdlet, 550, 551
- Get-MailboxServer cmdlet, 550
- Get-MailboxStatistics cmdlet, 558
- Get-Member cmdlet, 67, 122, 268, 269, 374, 378, 381, 529, 573
 - exercises using, 59–60
 - retrieving information about objects using, 44–48
- Get-Member object, 376
- Get-Module cmdlet, 223, 241
- Get-MyBios function, 245, 247, 248
- Get-MyBios.ps1 file, 248
- Get-MyModule function, 234, 236, 419
- Get-MyModule.ps1 script, 236
- Get-Net6to4Configuration job, 124
- Get-NetAdapter cmdlet, 126, 448, 457
- Get-NetConnectionProfile function, 225
- Get-OperatingSystemVersion function, 174, 228
- Get-OperatingSystemVersion.ps1 script, 174
- Get-OptimalSize function, 244
- Get-PowerShellRequirements.ps1 script, 3–4
- Get-PrintConfiguration cmdlet, 573
- Get-Printer cmdlet, 573
- Get-PrinterDriver cmdlet, 573
- Get-PrinterPort cmdlet, 573
- Get-PrinterProperty cmdlet, 573
- Get-PrintJob cmdlet, 573
- Get-Process cmdlet, 7, 129, 174, 263, 317, 573, 592
- Get-Process note* command, 8–9
- Get-PSBreakPoint cmdlet, 483, 485, 492, 493, 494, 497, 498, 573
- Get-PSCallStack cmdlet, 483, 491, 573
- Get-PSDrive cmdlet, 18, 77, 87, 520, 573
- Get-PSProvider cmdlet, 66, 67, 573
- Get-PSSession cmdlet, 116
- Get-Random cmdlet, 573
- Get-Service cmdlet, 174, 573
- Get-TextStatistics function, 174, 176
- Get-TextStats function, 180, 183
- Get-TraceSource cmdlet, 573
- Get-Transaction cmdlet, 573
- Get-TypeData cmdlet, 573
- GetType method, 523
- Get-UILanguage cmdlet, 573
- Get-Unique cmdlet, 573
- Get-ValidWmiClass function, 523, 524, 525
- Get-Variable administrator command, 101
- Get-Variable cmdlet, 573
- Get-Variable ShellId command, 100
- Get-Verb cmdlet, 3, 54, 205, 542
- Get-WindowsFeature cmdlet, 385, 386, 420, 448
- GetWmiClassesFunction.ps1 script, 184
- Get-WmiInformation function, 525
- Get-WmiNameSpace function, 286–288
- Get-WmiObject cmdlet, 68, 115, 124, 139, 174, 189, 196, 199, 253, 255, 264, 286, 291, 308, 311, 312, 314, 316, 317, 318, 322, 326, 338, 350, 355, 358, 364, 373, 428, 502, 509, 511, 514, 525, 573, 621
- Get-WmiProvider function, 289, 516, 521
- Get-WSManCredSSP cmdlet, 573
- Get-WSManInstance cmdlet, 573
- gh alias, 281

G+H keystroke combination

- G+H keystroke combination, 19
- ghy alias, 332, 334
- gi alias, 78, 82
- globally unique identifier (GUID), 425
- gm alias, 122, 292, 361
- gmb alias, 248
- GPO (Group Policy Object), 4
- gps alias, 31, 122, 129
- grave accent character (`), 137, 143, 319, 321
- greater-than (>) symbol, 320
- Group cmdlet, 585
- Group-Object cmdlet, 172, 573
- group policy, 337–338, 513
- Group Policy Object (GPO), 4
- groups, 394–395
- groupScope parameter, 433
- gsv alias, 32, 130
- gt argument, 59, 61, 162
- GUID (globally unique identifier), 425
- gwmi alias, 68, 291, 296, 301, 311, 330, 355
- gwmi win32_logicaldisk command, 312

H

- hard-coded numbers, avoiding, 631
- [hashtable] alias, 146, 190
- HasMessage filter, 204
- hasmoredata property, 129
- Height parameter, 52
- Help command, 13–20, 491
- Help function, 18, 249
- HelpMessage parameter property, 217, 221
- here-string object, 184–186
- Hit Variable breakpoint, 486
- HKEY_CLASSES_ROOT registry hive, 87, 281, 519
- HomeDirectory attribute, 404
- HomeDrive attribute, 405
- HomePhone attribute, 405
- HostingModel property, 517
- hostname command, 6
- HSG key, 93
- Hungarian Notation, 631
- Hyperv server, 425

I

- icontains operator, 507
- IdentifyingPropertiesOfDirectories.txt file, 80
- IdentifyServiceAccounts.ps1 script, 323

- identity parameter, 425, 434, 438, 439, 443, 548
- id parameter, 494
- IDs for jobs, 120
- If statement, 97, 157, 515
 - assignment operators, 163
 - compared with VBScript's If...Then...End statement, 161
 - comparison operators, 162–163
- ihy alias, 334
- ImpersonationLevel property, 517
- Import-Alias cmdlet, 574
- Import-Clixml cmdlet, 574
- Import cmdlet, 583
- Import-Csv cmdlet, 574
- importing modules, 241–242
- Import-LocalizedData cmdlet, 574
- Import-Module cmdlet, 225, 226, 237, 241, 248, 421, 422, 443
- Import-PSSession cmdlet, 541, 574
- in32_PerfFormattedData_TermService_TerminalServicesSession class, 618
- incorrect data types, 523–525
- info attribute, 407
- InitializationReentrancy property, 517
- InitializationTimeoutInterval property, 517
- InitializeAsAdminFirst property, 517
- Initialize cmdlet, 585
- initializing variables, 623
- inline code vs. functions, 197–198
- InLineGetIPDemo.ps1 script, 196, 197
- inputobject argument, 48, 300, 377, 381
- Insert button, 253, 255
- Install-ADDSDomainController cmdlet, 454, 456
- Install-ADDSEForest cmdlet, 459
- InstallDate property, 187, 315
- installDNS parameter, 454, 459
- installed software, finding, 327–330
- installing
 - Active Directory module, 419–420
 - PowerShell 3.0, 3
 - RSAT for Active Directory, 420
- InstallNewForest.ps1 script, 452
- instance methods, executing
 - Invoke-WmiMethod cmdlet, 358–360
 - overview, 355–357
 - using terminate method directly, 357–358
 - [wmi] type accelerator, 360–361
- [int] alias, 146, 190
- integers, 145

IntelliSense, 256, 462
 International module, 580
 Internet Protocol (IP) addresses, 112, 196
 adding DNS servers, 453
 assigning, 448
 InvocationInfo property, 390
 Invoke-AsWorkflow cmdlet, 574
 Invoke cmdlet, 583
 Invoke-Command cmdlet, 308, 341, 342, 350
 running command on multiple computers
 using, 118–120
 running single command using, 117–118
 Invoke-Expression cmdlet, 574
 Invoke-History cmdlet, 281
 Invoke-Item cmdlet, 73, 574
 Invoke-RestMethod cmdlet, 574
 Invoke-WebRequest cmdlet, 68, 574
 Invoke-WmiMethod cmdlet, 68, 262, 357, 358–360,
 359, 574
 Invoke-WSManAction cmdlet, 574
 IPAddress property, 196
 ipconfig command, 5, 6
 IP (Internet Protocol) addresses, 112, 196
 adding DNS servers, 453
 assigning, 448
 IPPhone attribute, 406
 IPSubNet property, 196
 iSCSI module, 580
 IscsiTarget module, 580
 ise alias, 271
 ISE module, 581
 ISEProfile variable, 280
 IsGlobalCatalog property, 425
 IsNullOrEmpty method, 443
 IsReadOnly property, 82
 IsToday filter, 205
 i variable, 151
 IwbemObjectSet object, 328
 iwmi alias, 68
 iwr alias, 68

J

jobs
 checking status of, 124–127
 IDs for, 120
 naming, 121–122
 naming return object, 123–124
 overview, 119

receiving, 120–121, 123–125
 removing, 121
 running, 120
 using cmdlets with, 122–124
 Join cmdlet, 584
 Join-Path cmdlet, 230, 287, 574
 join static method, String class, 593

K

Kds module, 580
 -keep parameter, 121, 126, 130, 351
 -key parameter, 468
 keys, registry
 creating and setting value at once, 95
 creating using full path, 94
 creating with New-Item cmdlet, 93
 listing, 65, 90–91
 overwriting, 94
 setting default value, 95

L

language parser, 461
 LastAccessTime property, 82
 LastAccessTimeUtc property, 82
 LastErrorCode property, 187
 LastWriteTime property, 60, 82, 206
 LastWriteTimeUtc property, 82
 -latest parameter, 176
 l attribute, 401
 launch options for console, 11
 -LDAPFilter parameter, 435
 LDAP (Lightweight Directory Access Protocol), 284,
 385, 387–388, 425
 length property, 30, 150
 Length property, 82
 -le operator, 162
 less-than (<) symbol, 320
 Lightweight Directory Access Protocol (LDAP), 284,
 385, 387–388, 425
 -like operator, 86, 162
 Limit cmdlet, 585
 Limit-EventLog cmdlet, 574
 limiting choices
 using -contains operator to examine contents of
 array, 507–509
 using -contains operator to test for
 properties, 509–511

overview

- overview, 504
 - using PromptForChoice, 504–505, 534–535
 - using Test-Connection to identify computer connectivity, 506
- line number, setting breakpoints, 483–484
- list argument, 290
- ListAvailable parameter, 223, 226, 235, 241, 278, 421
- List command, 491
- listing
 - certificates, 69–73
 - directory contents, 81
 - directory contents with Get-ChildItem cmdlet
 - formatting with Format-List cmdlet, 26
 - formatting with Format-Table cmdlet, 29
 - formatting with Format-Wide cmdlet, 27–29
 - overview, 24–26
 - environment variables, 77–78
 - filtered process list, 34
 - functions, 85–87
 - modules, 223–225
 - providers, 66
 - registry keys, 65, 90–91
 - WMI classes, 290–291
- ListProcessesSortResults.ps1 script, 132
- literal strings, 149
- loading modules, 225–227
- LocalMachine scope, 134
- Local User Management module, 445
- locations for modules, 222
- LockedOut parameter, 436
- locked-out users, 436–437
- logging service accounts, 323–324
- logging settings (Exchange Server 2010)
 - overview, 553
 - transport-logging levels
 - configuring, 554–557
 - reporting, 554–555
- logic errors, 466
- logon.vbs script, 404
- LogPath parameter, 459
- [long] alias, 146, 190
- looping
 - Do...While statement, 152–154
 - Foreach statement, 159–160
 - While statement, 150
- lt operator, 162

M

- Mailbox2 database, 551
- mailboxes (Exchange Server 2010)
 - creating
 - using Enable-Mailbox cmdlet, 544
 - when creating user, 544
 - database for
 - examining, 550–551
 - managing, 551–552
- ManagementClass object, 291
- mandatory parameter property, 217–218, 503
- manifest for modules, 241
- match operator, 59, 162, 291
- MaximumAllowed property, 315
- MaximumComponentLength property, 187
- MD alias, 365
- MeasureAddOneFilter.ps1 script, 201
- MeasureAddOneR2Function.ps1 script, 204
- Measure cmdlet, 584
- Measure-Command cmdlet, 574
- Measure-Object cmdlet, 313, 574
- MediaType property, 187
- Members parameter, 434
- MemberType method, 48
- membertype parameter, 46, 47, 81, 122
- message tracking (Exchange Server 2010), 568–570
- MessageTrackingLogEnabled argument, 569
- MessageTrackingLogMaxAge argument, 569
- MessageTrackingLogMaxDirectorySize argument, 570
- MessageTrackingLogPath argument, 570
- method notation, 481
- methods
 - of WMI classes, 368–369
 - retrieving for objects using Get-Member cmdlet, 44–48
- Microsoft Exchange Server 2010. *See* Exchange Server 2010
- Microsoft Management Console (MMC), 69, 386
- Microsoft.PowerShell.Diagnostics module, 580
- Microsoft.PowerShell.Host module, 581
- Microsoft.PowerShell.Management module, 223, 579
- Microsoft.PowerShell.Security module, 580
- Microsoft.PowerShell.Utility module, 223, 579
- Microsoft Systems Center Configuration Manager package, 4
- Microsoft TechNet article KB310516, 93
- Microsoft TechNet article KB322756, 93

- Microsoft TechNet Script Center, 65, 153
- Microsoft.WSMan.Management module, 580
- missing parameters, handling
 - assigning value in param statement, 502–503
 - detecting missing value and assigning in script, 502
 - making parameter mandatory, 503
 - overview, 501
- missing rights, handling
 - attempt and fail, 512
 - checking for rights and exiting gracefully, 513
 - overview, 512
- missing WMI providers, handling, 513–523
- misspelled words, 462, 621
- mkdir function, 365
- MMAgent module, 580
- MMC (Microsoft Management Console), 69, 386
- Mobile attribute, 406
- mode parameter, 486, 487
- ModifySecondPage.ps1 script, 405
- ModifyUserProperties.ps1 script, 398
- module coverage, 579–582
- Module parameter, 242, 421
- \$modulePath variable, 230–231
- modules
 - checking for dependencies, 234–236
 - creating
 - manifest for, 241
 - overview, 244
 - using Get-Help cmdlet with, 243–245
 - using Windows PowerShell ISE, 238–239
 - creating drive for, 232–233
 - deploying providers in, 66
 - directory for, 229
 - features of, 227
 - user folders for, 227–230
 - using functions from imported, 242–244
 - getting list of, 592
 - grouping profile functionality into, 277–278
 - importing, 241–242, 244
 - installing, 244
 - listing all available, 223–225
 - listing loaded, 223
 - loading, 225–227
 - locations for, 222
 - \$modulePath variable, 230–231
 - overview, 222
 - using with profiles, 274
 - script execution policy required to install, 232
 - using from shared location, 237–239

- Mount cmdlet, 585
- Mount-Database function, 552
- Move-ADObject cmdlet, 435
- Move cmdlet, 584
- Move-Item cmdlet, 574
- Move-ItemProperty cmdlet, 574
- moveNext method, 202
- mred alias, 60
- MsDtc module, 579
- MSIPROV WMI provider, 516
- multiple commands, running, 6
- multiple folders
 - creating using scripts, 168–169
 - deleting using scripts, 169–170
- multiple users, creating, 408–409
- multivalued users, creating, 414–417
- MyDocuments variable, 280
- myfile.txt file, 84
- Mytestfile.txt file, 20
- Mytest folder, 83

N

- named parameters, 628
- Name input box, 252
- name parameter, 78, 143, 218, 317, 433, 551
- Name property, 30, 82, 92, 187, 289, 291, 315, 517
- namespace parameter, 285, 289, 293, 328
- __NAMESPACE property, 517
- namespaces
 - explained, 284
 - exploring, 367
 - in WMI, 284–288
- __namespace WMI class, 517
- Name variable, 331
- naming
 - cmdlets, 3, 54–56, 583–586
 - verb distribution, 55–56
 - verb grouping for, 54–55
 - functions, 174–175, 628
 - jobs, 121–122
 - return object for job, 123–124
 - variables, 631
- NDS provider, 385
- ne operator, 162
- NetAdapter module, 579
- NetBIOS name, 458
- NetConnection module, 225, 581
- NetLbfo module, 580

NetQos module

- NetQos module, 580
- NetSecurity module, 579
- NetSwitchTeam module, 580
- NetTCPIP module, 580
- NetworkConnectivityStatus module, 580
- network shares, modules from, 237–239
- NetworkTransition module, 579
- New-ADComputer cmdlet, 432
- New-ADGroup cmdlet, 433
- New-AdminAuditLogSearch cmdlet, 560, 562
- New-ADOrganizationalUnit cmdlet, 432
- New-Alias cmdlet, 19, 248, 574
- New-CimSession cmdlet, 343
- New cmdlet, 583
 - newest parameter, 126
- New-Event cmdlet, 574
- New-EventLog cmdlet, 574
- New-ExchangeSession function, 542
- New-IseSnippet cmdlet, 259, 260, 630
- New-Item cmdlet, 78, 93, 169, 230, 270, 278, 574
- New-ItemProperty cmdlet, 574
- New-Line function, 180, 183
- NewMailboxAndUser.ps1 script, 545
- New-Mailbox cmdlet, 539, 545
- New-MailBoxDatabase cmdlet, 551, 552
 - NewName parameter, 79
- New-NetIPAddress cmdlet, 453, 458
- New-Object cmdlet, 44, 529, 530, 536, 574
 - exercises, 61
 - using, 50–51
- New-PSDrive cmdlet, 87, 103, 232, 520, 574
- New-PSSession cmdlet, 116, 353, 541
- New-Service cmdlet, 574
- New-TimeSpan cmdlet, 329, 574
- New-Variable cmdlet, 100, 168, 324, 574
- New-WebServiceProxy cmdlet, 574
- New-WsManInstance cmdlet, 575
- New-WsManSessionOption cmdlet, 575
- Next keyword, 156
- NFS module, 579
 - noexit parameter, 138, 140
- nonterminating errors, 512
- nopprofile parameter, 223
- notafter property, 75
- Notepad.exe file, 7
- notlike operator, 86, 162
- notmatch operator, 162
- not operator, 81, 228, 235
- noun parameter, 42
- Novell Directory Services servers, 385

- Novell NetWare 3.x servers, 385
- NumberOfBlocks property, 188
- numbers
 - hard-coded, avoiding, 631
 - random, generating, 591
- NWCOMPAT provider, 385
- NwTraders.msft domain, 384, 385, 413

O

- O attribute, 388
- Object Editor, for Win32_Product WMI class, 518
- objects
 - finding aliases for, 59
 - New-Object cmdlet, 50–51
 - retrieving information about using Get-Member cmdlet, 44–48
- objFile variable, 147
- objFSO variable, 147
- objWMIServices variable, 320
- off parameter, 479
- ogv alias, 32
- On Error Resume Next command, 136
- OneStepFurtherWindowsEnvironment.txt file, 335
- opening PowerShell, 10, 11
- OpenTextFile method, 147
- OperationTimeoutInterval property, 517
- operators for WMI queries, 321–322
- optional modules, 419
 - option parameter, 146, 168
- options for cmdlets, 12
- organizational settings, modifying, 409–411
- organizational units (OUs), 4, 383–384, 413, 432
- Organization tab, Active Directory Users and Computers, 409, 411
- OSinfo.txt file, 319
- OtherFacsimileTelephoneNumber attribute, 407
- OtherHomePhone attribute, 407
- OtherIPPhone attribute, 407
- OtherMobile attribute, 407
- OtherPager attribute, 407
- OtherTelephone attribute, 399
- OU attribute, 388
- OUs (organizational units), 4, 383–384, 413, 432
 - OutBuffer parameter, 12
- Out cmdlet, 583
- Out-File cmdlet, 324, 575, 592
- Out-GridView cmdlet, 31–34, 309, 565, 575
- Out-Null cmdlet, 230, 233, 520

- out-of-bounds errors, handling
 - overview, 526
 - placing limits on parameter, 528
 - using boundary-checking function, 526–527
- Out-Printer cmdlet, 575
- output
 - formatting with Format-Table cmdlet, 29
 - formatting with Format-Wide cmdlet, 27–29
 - formatting with Out-GridView cmdlet, 31–34
 - transcript tool and, 115–116
- Out-String cmdlet, 575
- OutVariable parameter, 12

P

- Pager attribute, 406
- parameter attribute
 - HelpMessage property, 221
 - mandatory property, 217–218
 - overview, 217
 - ParameterSetName property, 219
 - position property, 218–219
 - ValueFromPipeline property, 220–221
- parameters
 - missing, handling
 - assigning value in param statement, 502–503
 - detecting missing value and assigning in script, 502
 - making parameter mandatory, 503
 - overview, 501
 - named vs. unnamed, 628
 - placing limits on, 528
 - reducing data via, 347–350
- ParameterSetName parameter property, 217, 219, 246
- Parameters For... parameter box, 254
- parameters, function
 - avoiding use of many, 194
 - checking automatically, 211–214
 - using more than two, 192–193
 - using multiple, 186–187
 - positional, 96
 - specifying, 176
 - specifying default parameter set, 216–217
 - switched parameters, 193
 - unhandled, 213–214
- param keyword, 465, 502–503
- Param statement, 192, 209
- Pascal case, 385
- passthru parameter, 137
- passwords
 - changing, 444
 - domain password policy, checking, 429
- Paste button, Command add-on, 255
- Paste command, 255
- path parameter, 69, 78, 80, 96, 143, 150, 176, 192, 415, 432, 433
- Path property, 315, 359, 517
- paths
 - for module location, 229
 - for profiles, 267
- pause function, 87
- PDCs (primary domain controllers), 385
- performance, of functions, 203–204
- PerLocaleInitialization property, 517
- permission issues, 462, 463
- PerUserInitialization property, 517
- PING commands, 114
- PinToStartAndTaskBar.ps1 script, 11
- pipe character (|), 24, 75, 324, 556, 622
- pipeline, avoiding breaking, 621
- PKI module, 580
- plus symbol (+), 137, 143
- PNPDeviceID property, 188
- Pop cmdlet, 585
- Pop-Location cmdlet, 93, 96, 575
- Popup method, 62
- poshlog directory, 448
- positional parameters, 96, 175
- position message, 136
- position parameter property, 218–219
- postalCode attribute, 401
- postOfficeBox attribute, 401
- PowerManagementCapabilities property, 188
- PowerManagementSupported property, 188
- PowerShell
 - adding to task bar in Windows 7, 10–11
 - deploying to enterprise systems, 4
 - opening, 10, 11
 - profiles for, 57
- PowerShell.exe file, 141
- primary domain controllers (PDCs), 385
- PrintManagement module, 580
- Process block, 200, 203, 205
- processes
 - filtered list of, 34, 35
 - retrieving list of running processes, 317–318
- process ID, 8
- Process scope, 134

profileBackup.ps1 file

- profileBackup.ps1 file, 279
- ProfilePath attribute, 404
- profiles
 - All Users, All Hosts profile, 275–276
 - using central script for, 276–277
 - creating, 57, 270–271
 - deciding how to use, 271–272
 - determining existence of, 270
 - grouping functionality into module, 277–278
 - using modules with, 274
 - using multiple, 273–275
 - overview, 267–268
 - paths for, 267
 - \$profile variable, 268–270
 - script execution policy required for, 268
 - usage patterns for, 272
- program logic, 194
- ProhibitSendQuota property, 549
- PromptForChoice method, 504–505, 534–535
- prompt, PowerShell, 76
- properties
 - using -contains operator to test for, 509–511
 - for certificates, 72–73
 - for directories, 81–82
 - retrieving every property from every instance of class, 314
 - retrieving for objects using Get-Member cmdlet, 44–48
 - retrieving specific properties from, 316
- __PROPERTY_COUNT property, 518
- property parameter, 26, 256, 296, 313, 325, 326, 347, 372, 373, 441
- ProtectedFromAccidentalDeletion parameter, 433
- __provider class, 517
- ProviderName property, 188
- provider property, 90
- providers
 - alias, 66–68
 - certificate, 68
 - defined, 65
 - environment provider, 76
 - filesystem provider, 80
 - function provider, 85
 - in WMI, 289
 - listing, 66
 - overview, 65–66
 - registry, 90
 - variable, 97–98
- __provider WMI system class, 517
- .ps1 extension, 133

- PSComputerName property, 118, 183, 342
- Psconsole file, 11
- psconsolefile argument, 12
- .psd1 extension, 228
- PSDiagnostics module, 580
- PSDrives
 - for registry, 87–88, 520
 - switching, 68
- PsisContainer property, 75, 82
- .psm1 extension, 228, 237, 239
- PSModulePath variable, 229, 421
- PSProvider parameter, 103
- PSScheduledJob module, 580
- PSStatus property, 188, 295
- PSWorkflow module, 581
- Pure property, 517
- Purpose property, 188
- Push cmdlet, 585
- Push-Location cmdlet, 93, 575
- Put method, 393, 395
- pwd alias, 68

Q

- QualifierName parameter, 367, 369
- querying
 - Active Directory, 590
 - WMI
 - eliminating WMI query argument, 320–321
 - finding installed software, 327–330
 - identifying service accounts, 322–323
 - logging service accounts, 323–324
 - obtaining BIOS information, 308–311
 - using operators, 321–322
 - overview, 293
 - retrieving data from specific instances of class, 319–320
 - retrieving default WMI settings, 308
 - retrieving every property from every instance of class, 314
 - retrieving information about all shares on local machine, 315
 - retrieving list of running processes, 317–318
 - retrieving specific properties from class, 316
 - shortening syntax, 325–326
 - specific class, 293–296
 - specifying maximum number of connections to server, 316–317
 - substituting Where clause with variable, 325

- viewing Windows environment
 - variables, 330–335
- Win32_Desktop class, 296–298
- working with disk drives, 312–314
- query parameter, 314, 348
- QuickEdit mode, 72
- quiet parameter, 506
- QuotasDisabled property, 188
- QuotasIncomplete property, 188
- QuotasRebuilding property, 188
- quotation marks, 189
 - in console, 133
 - used with -filter argument, 318

R

- random numbers, 591
- range operator, 152
- rate parameter, 195
- RDN (relative distinguished name), 384, 387
- readability
 - of filters, 204–205
 - of functions, 198
- Read cmdlet, 585
- Read-Host cmdlet, 174, 546, 575, 594
- ReadingAndWritingForFiles.txt file, 80
- Read mode, 485
- read-only variables, 587
- ReadUserInfoFromReg.ps1 script
 - cmdlets used, 143
 - code, 143–144
 - variables used, 142
- ReadWrite mode, 485
- rebooting server, 454, 456
- rebootoncompletion parameter, 459
- Receive cmdlet, 584
- Receive-Job cmdlet, 120, 123, 129, 350, 353, 354
- recipient settings, configuring (Exchange Server 2010)
 - mailbox, creating
 - multiple mailboxes, 546–547
 - using Enable-Mailbox cmdlet, 544
 - when creating user, 544–546
 - reporting user settings, 548–550
- recurse parameter, 27, 29, 61, 69, 83, 102, 196, 231
- recycled variables, 631
- redirect-and-append arrow (>>), 6
- redirection arrow (>), 6, 318

- red squiggly lines, 462
- Regedit.exe file, 90
- Register cmdlet, 583
- Register-EngineEvent cmdlet, 575
- Register-ObjectEvent cmdlet, 575
- Register-WmiEvent cmdlet, 575
- registry
 - backing up, 93
 - determining existence of property, 96
 - drives for, 87–88
 - keys for
 - creating and setting value at once, 95
 - creating using full path, 94
 - creating with New-Item cmdlet, 93
 - overwriting, 94
 - setting default value, 95
 - listing keys in, 65, 90–91
 - modifying property value, 95
 - modifying property value using full path, 96
 - provider overview, 90
 - remote access to, 87
 - retrieving default property value from, 90
 - retrieving values from, 89–90
 - searching for software in, 92
 - taking care when modifying, 93
 - testing for property before writing, 97
- regular expressions, 591
- relative distinguished name (RDN), 384, 387
- __RelPath property, 358, 359, 360, 518
- RemoteDesktop module, 579
- Remote Management firewall exception, 114
- remote procedure call (RPC), 338
- Remote Server Administration Tools (RSAT), 419
- remote servers, 540–543
- RemoteSigned execution policy, 134
- remoting
 - accessing local registry, 87
 - cmdlets for, 107–112
 - configuring, 112–114
 - creating session, 115–118
 - credential parameter support, 110
 - firewall exceptions, 114
 - impersonating current user, 115
 - running command as different user, 110–111
 - running single command
 - on multiple computers, 118–120
 - on single computer, 117–118
 - saving sessions, 116–117
 - testing configuration, 113–114

Windows PowerShell

- Windows PowerShell
 - discovering information about forest and domain, 428–431
 - obtaining FSMO information using, 428
- WMI
 - disadvantages of, 341
 - remote results, 344–348
 - supplying alternate credentials for remote connection, 338–341
 - using CIM classes to query WMI classes, 343–344
 - using group policy to configure WMI, 337–338
- Remove-ADGroupMember cmdlet, 434
- Remove cmdlet, 583
- Remove-Computer cmdlet, 575
- Remove-Event cmdlet, 575
- Remove-EventLog cmdlet, 575
- Remove-IsSnippet cmdlet, 261
- Remove-Item cmdlet, 74, 80, 83, 169, 279, 575
- Remove-ItemProperty cmdlet, 575
- Remove-Job cmdlet, 121
- Remove-MailboxDatabase cmdlet, 552
- Remove-Printer cmdlet, 575
- Remove-PrinterDriver cmdlet, 575
- Remove-PrinterPort cmdlet, 575
- Remove-PrintJob cmdlet, 575
- Remove-PSBreakPoint cmdlet, 483, 494, 497, 498, 575
- Remove-PSDrive cmdlet, 103, 521, 575
- Remove-PSSession cmdlet, 116
- Remove-TypeData cmdlet, 575
- RemoveUserFromGroup.ps1 script, 434
- Remove-Variable cmdlet, 101, 575
- Remove-WmiObject cmdlet, 68, 365, 575
- Remove-WSMInstance cmdlet, 575
- Rename-ADObject cmdlet, 432
- Rename cmdlet, 584
- Rename-Computer cmdlet, 448, 455, 458, 575
- Rename-Item cmdlet, 79, 575
- Rename-ItemProperty cmdlet, 575
- Rename-Printer cmdlet, 575
- renaming environment variables, 79
- Repair cmdlet, 585
- Repeat command, 491
- Replace method, System.String .NET Framework class, 595
- replicationsourcedc parameter, 454
- reporting user settings (Exchange Server 2010), 548–550
- ReportTransportLogging.ps1 script, 555
- requires statement, 246
- Reset cmdlet, 585
- Reset-ComputerMachinePassword cmdlet, 576
- Reset method, 187, 362
- Resolve cmdlet, 584
- Resolve-Path cmdlet, 576
- Resolve-ZipCode function, 190
- Resolve-ZipCode.ps1 script, 190
- “Resource not available” run-time error, 462
- resources, unavailable, 462
- Restart cmdlet, 584
- Restart-Computer cmdlet, 449, 454, 456, 458, 576
- restart parameter, 448
- Restart-PrintJob cmdlet, 576
- Restart-Service cmdlet, 576
- Restore cmdlet, 585
- Restore-Computer cmdlet, 576
- Restricted execution policy, 134, 136, 513
- resultclassname parameter, 377
- Resume cmdlet, 584
- Resume-PrintJob cmdlet, 576
- Resume-Service cmdlet, 576
- RetrieveAndSortServiceState.ps1 script, 139
- ReturnValue, 304
- returnvalue property, 363
- reusability of functions, 198
- rich types, 627
- rights, missing. *See* missing rights, handling
- root/cimv2 WMI namespace, 369, 370
- route print command, 6
- RPC (remote procedure call), 338
- rsat-ad-tools feature, 421
- RSAT (Remote Server Administration Tools), 419, 420
- Run as different user command, 110–111
- Run As Different User dialog box, 111
- Run button, 252
- Run dialog box, 138
- Run ISE As Administrator option, 251
- run method, 51
- RunningMultipleCommands.txt file, 6
- Run Script button, 255
- run-time errors, 462–465
- rwmi alias, 68

S

- sal alias, 67
- sAMAccountName attribute, 393, 394
- Saver cmdlet, 584
- sbp alias, 67
- sc alias, 67
- scheduled tasks, 132
- ScheduledTasks module, 580
- SchemaMaster role, 425
- ScreenSaverExecutable property, 297
- ScreenSaverSecure property, 297
- ScreenSaverTimeout property, 297
- Screen* wildcard pattern, 297
- script block, 148
- scriptblock parameter, 128
- script execution policies
 - overview, 57, 134
 - required for using profiles, 268
 - required for using snippets, 259
 - required to install modules, 232
 - retrieving current, 135–136
 - setting, 135–136
- script-level tracing
 - enabling, 467
 - trace level 1, 468–469
 - trace level 2, 470–471
- script pane
 - in Windows PowerShell ISE, 254–255
 - opening new, 254
 - running commands in, 255
 - using Commands add-on with, 255
- script parameter, 485, 486, 489
- ScriptPath attribute, 404
- scripts. *See also* constants; error handling; variables
 - advantages of using, 131–133
 - using arrays to run commands multiple times, 138
 - creating multiple folders using, 168–169
 - debugging using breakpoints
 - deleting breakpoints, 494
 - enabling and disabling breakpoints, 494
 - exercise, 496–498
 - listing breakpoints, 492–493
 - responding to breakpoints, 490–492
 - setting on commands, 489–490
 - setting on line number, 483–484
 - setting on variables, 485–489
 - deleting multiple folders using, 169–170
 - dot-sourcing, 178, 179–180, 180–181
 - enabling support for, 134–135
 - execution policies for
 - overview, 134, 513
 - retrieving current, 135–136
 - setting, 135–136
 - functions in, 197–198, 625
 - using to hold profile information, 276–277
 - need for modification of, 196
 - overview, 133
 - using -passthru parameter, 137–138
 - readability of, 627–628
 - running, 133
 - as scheduled tasks, 132
 - inside PowerShell, 140
 - outside PowerShell, 140–141
 - overview, 138–140
 - sharing, 132
 - writing, 136–138
- SDDL (Security Descriptor Definition Language), 362
- SDDLToBinarySD method, 363
- SDDLToWin32SD method, 363
- Search-ADAccount cmdlet, 436, 437, 438
- Search-AdminAuditLog cmdlet, 558
- SearchBase parameter, 440
- searching
 - certificates, 74–75
 - for cmdlets using wildcards, 36–39, 43
- secret commands, 132
- SecureBoot module, 580
- security
 - confirming execution of cmdlets, 8
 - controlling cmdlet execution, 7
 - overview, 6–7
 - suspending execution of cmdlets, 9
- Security Descriptor Definition Language (SDDL), 362
- SecurityDescriptor property, 517
- select alias, 293, 296, 340
- Select Case statement (VBScript), 164–165
- Select cmdlet, 584
- Select Columns dialog box, 35
- Select-Object cmdlet, 225, 286, 293, 296, 309, 313, 340, 381, 564, 576
- Select statement, 316
- Select-String cmdlet, 294, 576
- Select-Xml cmdlet, 576
- Send cmdlet, 584
- Send-MailMessage cmdlet, 576
- SendTo folder shortcut, 141
- serveraddresses parameter, 453
- ServerCore module, 581

ServerManager module

- ServerManager module, 448, 580
- ServerManagerTasks module, 580
- server parameter, 551
- __SERVER property, 518
- servers, maximum number of connections to, 316–317
- service accounts
 - identifying, 322–323
 - logging, 323–324
- ServiceAccounts.txt file, 324
- ServiceDependencies.ps1 script, 631
- Service Pack (SP) 1, 3
- sessions
 - creating remote, 115–118
 - saving remote, 116–117
- Set-ADAccountPassword cmdlet, 435, 444
- Set-AdminAuditLog cmdlet, 558
- Set-AdminAuditLogConfig cmdlet, 558
- Set-ADObject cmdlet, 432
- Set-ADUser cmdlet, 443
- set alias, 67
- Set-Alias cmdlet, 67, 576
- Set cmdlet, 583
- Set-Content cmdlet, 67, 576
- Set-Date cmdlet, 576
- Set-DNSClientServerAddress cmdlet, 453
- Set-EventLogLevel cmdlet, 554
- Set-ExecutionPolicy cmdlet, 134, 232, 259, 513
- Set-Info() method, 389, 393, 396, 414, 416
- Set-Item cmdlet, 67, 95, 576
- Set-ItemProperty cmdlet, 67, 96, 576
- Set-Location cmdlet, 93, 331, 576
 - alias for, 67
 - switching PS drive using, 68
 - working with aliases using, 66
- Set-MailboxServer cmdlet, 569
- SetPowerState method, 187, 362
- Set-PrintConfiguration cmdlet, 576
- Set-Printer cmdlet, 576
- Set-PrinterProperty cmdlet, 576
- Set-Profile function, 279, 280
- Set-PropertyItem cmdlet, 95
- Set-PSBreakPoint cmdlet, 67, 483, 496, 576
- Set-PSDebug cmdlet, 624
 - overview, 467
 - script-level tracing using
 - enabling, 467
 - trace level 1, 468–469
 - trace level 2, 470–471
 - step parameter, 472–478
 - stepping through script, 471–479
 - strict mode, enabling, 479–480
- Set-Service cmdlet, 576
- Set-StrictMode cmdlet, 481–482
- Set-StrictMode -Version 2 command, 481
- Set-TraceSource cmdlet, 576
- Set-Variable cmdlet, 67, 101, 146, 576
- Set-WmiInstance cmdlet, 67, 68, 576
- Set-WSManInstance cmdlet, 576
- Set-WSManQuickConfig cmdlet, 576
- shared folders, 237–239
- ShareNoQuery.ps1 script, 321
- shares, retrieving information about, 315
- ShellId variable, 100
- shortcut dot (.), 320
- shortcuts, adding to SendTo folder, 141
- Show cmdlet, 584
- Show-Command cmdlet, 52, 576
- Show Commands Add-On option, 256
- Show-ControlPanelItem cmdlet, 576
- Show-EventLog cmdlet, 576
- Show MOF button, 361
- si alias, 67
- signature of functions, 195
- SilentlyContinue parameter, 392
- simple typing errors, 479–480
- [single] alias, 146, 190
- single quote (') character, 92, 320
- Single-Threaded Apartment model (STA), 273
- SIN method, 363
- Size property, 188
- sl alias, 67, 70, 115, 331
- SmallBios.ps1 script, 309
- SmbShare module, 580
- SmbWitness module, 581
- snap-ins
 - defined, 66, 222, 234
 - uninstalling, 66
- snippets
 - creating code with, 257–259
 - creating user-defined, 259–260
 - defined, 257
 - removing user-defined, 261–262
 - script execution policy required for, 259
- software, installed
 - finding using WMI, 327–330
 - searching for in registry, 92
- Software Update Services (SUS), 4
- sort alias, 78, 299
- Sort cmdlet, 584

- sorting
 - alphabetical listings, 77
 - list of processes, 35
- Sort-Object cmdlet, 139, 298, 302, 322, 576
- space, in path of script, 588
- sp alias, 67
- special variables, 142
- spelling, 621
- Split cmdlet, 567, 584
- split method, 229, 232
- Split-Path cmdlet, 576
- SP (Service Pack) 1, 3
- squiggly lines, 462
- Start cmdlet, 583
 - startdate parameter, 560
- Start-Job cmdlet, 120, 123, 125
- Start-Process cmdlet, 577
- Start-Service cmdlet, 300, 577
- StartService method, 305
- Start-Sleep cmdlet, 577
- Start Snippets option, 257
- Start-Transaction cmdlet, 577
- Start-Transcript cmdlet, 58, 115, 273, 591
- STA (Single-Threaded Apartment model), 273
- state property, 302
- static methods, 361–363, 365–366
- st attribute, 401
- StatusInfo property, 188
- status of jobs, checking, 124–127
- Status property, 188, 298, 301, 315
- Step-Into command, 491
- Step-Out command, 491
- Step-Over command, 491
- step parameter, 472–478
- Stop cmdlet, 491, 584
- Stop-Computer cmdlet, 577
- Stop-Job cmdlet, 125
- StopNotepadSilentlyContinuePassThru.ps1 script, 138
- Stop-Process cmdlet, 8–10, 137, 214, 263, 577
- Stop-Service cmdlet, 214, 300, 577
- Storage module, 579
- storage settings (Exchange Server 2010)
 - mailbox database
 - examining, 550–551
 - managing, 551–552
 - overview, 550–551
- streetAddress attribute, 401
- Street attribute, 388
- strict mode, enabling
 - overview, 479
 - using Set-PSDebug -Strict, 479–480
 - using Set-StrictMode cmdlet, 481–482
- Strict parameter, 480
- [string] alias, 146, 190
- String Attribute Editor, ADSI Edit, 388
- String class, 232
- strings
 - expanding, 148, 157
 - literal, 149
- subject property, 74
- subroutines in VBScript, 171
- __SUPERCLASS property, 518
- supervariable, 79
- SupportsDiskQuotas property, 188
- SupportsExplicitShutdown property, 517
- SupportsExtendedStatus property, 517
- SupportsFileBasedCompression property, 188
- SupportsQuotas property, 517
- SupportsSendStatus property, 517
- SupportsShouldProcess attribute, 214, 215
- SupportsShutdown property, 517
- SupportsThrottling property, 517
- suspend argument, 7
- Suspend cmdlet, 584
- suspending execution of cmdlets, 9
- Suspend-PrintJob cmdlet, 577
- Suspend-Service cmdlet, 577
- SUS (Software Update Services), 4
- sv alias, 67
- Switch cmdlet, 584
- Switch_DebugRemoteWMI_Session.ps1 script, 465
- switched parameters, 193
- Switch statement
 - compared with VBScript's Select Case statement, 164–165
 - Defining default condition, 165–166
 - evaluating arrays, 166–167
 - handling multiple parameters using, 219
 - matching behavior, controlling, 167
 - matching with, 166
- swmi alias, 67
- syntax argument, 43
- syntax errors, 461–462
- SystemCreationClassName property, 188
- System.Diagnostics.Process .NET Framework object, 122
- System.DirectoryServices.DirectoryEntry object, 384
- System.Environment .NET Framework class, 272

System.Exception Catch block

- System.Exception Catch block, 534
- System.Exception error, 529, 531
- System.IO.DirectoryInfo object, 82
- System.IO.FileInfo class, 82, 230
- System.Management.Automation.LineBreak .NET Framework class, 483, 485
- System.Management.Automation.
 - PSArgumentException object, 532
- System.Management.ManagementClass class, 523
- System.Math class, 363
- SystemName property, 188
- SystemSecurity class, 290
- System.String class, 229
- System.SystemException class, 191
- System.Xml.XmlDocument type, 563
- SysVolpath parameter, 459

T

- `t command, 588
- tab completion, 24, 51, 104, 140
- tab expansion, 256, 358, 367, 381, 462–463
- TargetObject property, 390
- taskbar, adding shortcuts to, 10–11
- Tasks menu, 251
- TechNet Script Center Script Repository, 445
- TechNet Script Repository, 80
- TechNet wiki, 257
- Tee cmdlet, 584
- Tee-Object cmdlet, 577
- telephone settings, modifying, 405–407
- Telephones tab, Active Directory Users and Computers, 405
- template files, 630
- terminate method, 355, 357–358, 360
- terminating errors, 512
- testB object, 391
- Test cmdlet, 583
- Test-ComputerPath.ps1 script, 506
- Test-ComputerSecureChannel cmdlet, 577
- Test-Connection cmdlet, 464, 504, 506, 577
- Test-Mandatory function, 218
- Test-ModulePath function, 228, 231
- Test-ParameterSet function, 219
- Test-Path cmdlet, 93, 94, 97, 228, 270, 278, 467, 469, 519, 520, 577, 623
- Test-PipedValueByPropertyName function, 220
- Test-ValueFromRemainingArguments function, 220
- Test-WSMan cmdlet, 113, 577
- TextFunctions.ps1 script, 180, 183
- Text parameter, 260
- TextStreamObject, 150
- Then keyword, 161
- thumbprint attribute, 71
- Title parameter, 260
- Today parameter, 193
- totalSeconds property, 329
- Trace cmdlet, 584
- Trace-Command cmdlet, 577
- trace parameter, 470
- tracing, script-level. *See* script-level tracing
- Transcript command, 58
- transcript tool, 115–116
- transport-logging levels (Exchange Server 2010)
 - configuring, 554–557
 - reporting, 554–555
- Trap statement, 191, 513
- triple-arrow prompt, 9
- troubleshooting, 621–624
- TroubleshootingPack module, 581
- TrustedPlatformModule module, 580
- Try...Catch...Finally, error handling using
 - Catch block, 529
 - catching multiple errors, 532–533
 - exercise, 536–537
 - Finally block, 529–530
 - overview, 529
- Tshoot.txt file, 6
- type argument, 170
- type constraints in functions, 190–191
- typename property, 378
- Type property, 315
- Types.ps1xml file, 294
- typing errors, 479–480

U

- UAC (User Account Control), 512
- UID attribute, 388
- unavailable resources, 462
- Unblock cmdlet, 584
- Unblock-File cmdlet, 577
- UNC (Universal Naming Convention), 237, 404, 462
- Undefined execution policy, 134
- UnderstandingTheRegistryProvider.txt file, 90
- UnderstandingTheVariableProvider.txt file, 97
- Undo cmdlet, 584
- Undo-Transaction cmdlet, 577

- unfocused variables, 631
- unhandled parameters, 213–214
- unique parameter, 381
- Universal Naming Convention (UNC), 237, 404
- UnloadTimeout property, 517
- Unlock-ADAccount cmdlet, 437, 438
- unlocking locked-out users, 436–437
- unnamed parameters, 628
- Unregister cmdlet, 584
- Unregister-Event cmdlet, 577
- Unrestricted execution policy, 134
- unwanted execution, preventing, 155–156
- Update cmdlet, 584
- Update-FormatData cmdlet, 577
- Update-Help cmdlet, 13–15, 98
- UpdateHelpTrackErrors.ps1 script, 14–15
- Update-List cmdlet, 577
- Update-TypeData cmdlet, 577
- UPN (user principal name), 544
- url attribute, 399
- usage patterns for profiles, 272
- UseADCmdletsToCreateOuComputerAndUser.ps1 script, 433
- use-case scenario, 501
- Use cmdlet, 584
- UserAccessLogging module, 580
- UserAccountControl attribute, 396
- User Account Control (UAC), 512
- user accounts, creating (Exchange Server 2010)
 - exercise, 565–568
 - multiple, 546–547
 - when creating mailbox, 544–546
- User class, 394
- user-defined snippets, 260
- UserDomain property, 62
- UserGroupTest group, 434
- UserNames.txt file, 565
- UserName variable, 331
- user principal name (UPN), 544
- users
 - Active Directory and
 - computer account, 395–396
 - deleting users, 411–412
 - exposing address information, 400–401
 - general user information, 398–399
 - groups, 394–395
 - modifying user profile settings, 403–405
 - modifying user properties, 397–398
 - multiple users, creating, 408–409
 - multivalued users, creating, 414–417

- organizational settings, modifying, 409–411
- overview, 393–394
- telephone settings, modifying, 405–407
- user account control, 396–397
 - soliciting input from, 594
- Use-Transaction cmdlet, 577
- UsingWhatif.txt file, 7–8
- uspendConfirmationOfCmdlets.txt file, 9

V

- ValidateRange parameter attribute, 528
- value argument, 79
- ValueFromPipelineByPropertyName property, 217, 220
- ValueFromPipeline parameter property, 217, 220–221, 246
- ValueFromRemainingArguments property, 217, 220
- value parameter, 324, 468
- values
 - passing to functions, 175
 - retrieving from registry, 89–90
- variable parameter, 485, 486
- variables
 - constants compared with, 146
 - creating, 100–101, 170
 - deleting, 101
 - grouping, 631
 - improperly initialized, 479, 481, 488
 - indicating can only contain integers, 145
 - initializing properly, 623
 - naming, 631
 - nonexistent, 479
 - provider for, 97–98
 - putting property selection into, 373
 - recycled, 631
 - retrieving, 98–100
 - scope of, 631
 - setting breakpoints on, 485–489
 - special, 142
 - storing CIM instance in, 374
 - storing remote session as, 116–117
 - unfocused, 631
 - using, 141–146
 - Windows environment variables, 330–335
- VariableValue variable, 331
- verb argument, 39
- verbose parameter, 12, 15, 94, 210–211, 227, 516, 519

verbs

- verbs, 172, 175
 - distribution of, 55–56
 - grouping of, 54–55
- version parameter, 482
- version property, 174, 517
- video classes, WMI, 380–381
- <view> configuration, 294
- VolumeDirty property, 188
- VolumeName property, 188
- VolumeSerialNumber property, 188
- VpnClient module, 580

W

- Wait cmdlet, 584
- Wait-Event cmdlet, 577
- Wait-Job cmdlet, 68, 124, 451
- Wait-Process cmdlet, 577
- WbemTest (Windows Management Instrumentation Tester), 361, 513
- Wdac module, 580
- Web Services Description Language (WSDL), 190
- Web Services Management (WSMAN), 108
- whatif parameter, 12, 261, 629
 - adding support for to function, 214–215
 - controlling execution with, 7
 - using before altering system state, 74
- Whea module, 581
- whenCreated property, 441
- where alias, 68, 70, 82
- Where clause, 325
- Where cmdlet, 585
- Where-Object cmdlet, 59, 67, 70, 108, 204, 261, 299, 493, 559
 - alias for, 68
 - compounding, 76
 - searching for aliases using, 66
- WhileDoesNotRun.ps1 script, 156
- While...Not ...Wend loop, 147
- WhileReadLine.ps1 script, 150
- WhileReadLineWend.vbs script, 147
- While statement
 - constructing, 148–149
 - example of, 150
 - looping with, 150
 - preventing unwanted execution using, 155–156
- While...Wend loop, 147
- whoami command, 128
- Width parameter, 52
- wildcards
 - asterisk (*) character, 7, 17, 21, 68, 293, 309, 442
 - in Commands add-on, 252
 - in Windows PowerShell 2.0, 226
 - loading modules using, 226
 - searching for cmdlets using, 36–39
 - searching job names, 121
- Win32_1394Controller class, 598
- Win32_1394ControllerDevice class, 598
- Win32_Account class, 614
- Win32_AccountSID class, 610
- Win32_ACE class, 610
- Win32_ActiveRoute class, 607
- Win32_AllocatedResource class, 598
- Win32_AssociatedBattery class, 601
- Win32_AssociatedProcessorMemory class, 598
- Win32_AutochkSetting class, 598
- Win32_BaseBoard class, 598
- Win32_BaseService class, 612
- Win32_Battery class, 601
- Win32_Bios WMI class, 292, 309, 343, 371, 501, 512, 514, 598
- Win32_BootConfiguration class, 608
- Win32_Bus class, 598
- Win32_CacheMemory class, 598
- Win32_CDROMDrive class, 598
- Win32_CIMLogicalDeviceCIMDataFile class, 604
- Win32_ClassicCOMApplicationClasses class, 603
- Win32_ClassicCOMClass class, 603
- Win32_ClassicCOMClassSettings class, 603
- Win32_ClientApplicationSetting class, 603
- Win32_CodecFile class, 607
- Win32_CollectionStatistics class, 605
- Win32_COMApplication class, 603
- Win32_COMApplicationClasses class, 603
- Win32_COMApplicationSettings class, 603
- Win32_COMClassAutoEmulator class, 603
- Win32_COMClass class, 603
- Win32_COMClassEmulator class, 603
- Win32_ComponentCategory class, 603
- Win32_ComputerShutdownEvent class, 607
- Win32_ComputerSystem class, 309, 319, 608
- Win32_ComputerSystemEvent class, 607
- Win32_ComputerSystemProcessor class, 608
- Win32_ComputerSystemProduct class, 608
- Win32_ComputerSystemWindows
 - ProductActivationSetting class, 615
- Win32_COMSetting class, 603
- Win32_ConnectionShare class, 612
- Win32_ControllerHasHub class, 598

Win32_CurrentProbe class, 601
 Win32_CurrentTime WMI class, 294
 Win32_DCOMApplicationAccessAllowedSetting class, 603
 Win32_DCOMApplication class, 603
 Win32_DCOMApplicationLaunchAllowedSetting class, 604
 Win32_DCOMApplicationSetting class, 604
 Win32_DependentService class, 608
 Win32_Desktop class, 296–298, 604
 Win32_DesktopMonitor class, 294, 602
 Win32_DeviceBus class, 598
 Win32_DeviceChangeEvent class, 607
 Win32_DeviceMemoryAddress class, 598
 Win32_DeviceSettings class, 598
 Win32_DFSNode class, 612
 Win32_DFSNodeTarget class, 612
 Win32_DFSTarget class, 612
 Win32_Directory class, 604
 Win32_DirectorySpecification class, 604
 Win32_DiskDrive class, 598
 Win32_DiskDriveToDiskPartition class, 604
 Win32_DiskPartition class, 604
 Win32_DiskQuota class, 604
 Win32_DisplayConfiguration class, 370, 602
 Win32_DisplayControllerConfiguration class, 602
 Win32_DMACHannel class, 598
 Win32_DriverForDevice class, 601
 Win32_DriverVXD class, 604
 Win32_Environment class, 330, 604
 Win32_Fan class, 597
 Win32_FloppyController class, 598
 Win32_FloppyDrive class, 598
 Win32_Group class, 614
 Win32_GroupInDomain class, 614
 Win32_GroupUser class, 614
 Win32_HeatPipe class, 597
 Win32_IDEController class, 599
 Win32_IDEControllerDevice class, 599
 Win32_ImplementedCategory class, 604
 Win32_InfraredDevice class, 599
 Win32_IP4PersistedRouteTable class, 607
 Win32_IP4RouteTable class, 607
 Win32_IP4RouteTableEvent class, 607
 Win32_IRQResource class, 599
 Win32_Keyboard class, 597
 Win32_LoadOrderGroup class, 608
 Win32_LoadOrderGroupServiceDependencies class, 608
 Win32_LoadOrderGroupServiceMembers class, 608
 Win32_LocalTime class, 610
 WIN32_loggedonuser WMI class, 341
 Win32_LogicalDisk class, 146, 187, 189, 318, 605
 Win32_LogicalDiskRootDirectory class, 605
 Win32_LogicalDiskToPartition class, 605
 WIN32_LogicalDisk WMI class, 312, 314
 Win32_LogicalFileAccess class, 611
 Win32_LogicalFileAuditing class, 611
 Win32_LogicalFileGroup class, 611
 Win32_LogicalFileOwner class, 611
 Win32_LogicalFileSecuritySetting class, 611
 Win32_LogicalMemoryConfiguration class, 606
 Win32_LogicalProgramGroup class, 612
 Win32_LogicalProgramGroupDirectory class, 612
 Win32_LogicalProgramGroupItem class, 613
 Win32_LogicalProgramGroupItemDataFile class, 613
 Win32_LogicalShareAccess class, 611
 Win32_LogicalShareAuditing class, 611
 Win32_LogicalShareSecuritySetting class, 611
 Win32_LogonSession class, 614
 Win32_LogonSessionMappedDisk class, 614
 Win32_LogonSession WMI class, 374
 Win32_LUIDandAttributes class, 605
 Win32_LUID class, 605
 Win32_MappedLogicalDisk class, 605
 Win32_MemoryArray class, 599
 Win32_MemoryArrayLocation class, 599
 Win32_MemoryDeviceArray class, 599
 Win32_MemoryDevice class, 599
 Win32_MemoryDeviceLocation class, 599
 Win32_ModuleLoadTrace class, 607
 Win32_ModuleTrace class, 607
 Win32_MotherboardDevice class, 599
 Win32_NamedJobObjectActgInfo class, 606
 Win32_NamedJobObject class, 605
 Win32_NamedJobObjectLimit class, 606
 Win32_NamedJobObjectLimitSetting class, 606
 Win32_NamedJobObjectProcess class, 606
 Win32_NamedJobObjectSecLimit class, 606
 Win32_NamedJobObjectSecLimitSetting class, 606
 Win32_NamedJobObjectStatistics class, 606
 Win32_NetworkAdapter class, 601
 Win32_NetworkAdapterConfiguration class, 196, 601
 Win32_NetworkAdapterSetting class, 601
 Win32_NetworkClient class, 607
 Win32_NetworkConnection class, 607
 Win32_NetworkLoginProfile class, 614
 Win32_NetworkProtocol class, 607
 Win32_NTDomain class, 607

Win32_NTEventlogFile class

Win32_NTEventlogFile class, 614
Win32_NTLogEvent class, 614
Win32_NTLogEventComputer class, 614
Win32_NTLogEventLog class, 614
Win32_NTLogEventUser class, 614
Win32_OnBoardDevice class, 599
Win32_OperatingSystemAutochkSetting class, 605
Win32_OperatingSystem class, 174, 319, 608
Win32_OperatingSystemQFE class, 608
Win32_OSRecoveryConfiguration class, 609
Win32_PageFile class, 606
Win32_PageFileElementSetting class, 606
Win32_PageFileSetting class, 606
Win32_PageFileUsage class, 606
Win32_ParallelPort class, 599
Win32_PCMCIAController class, 599
Win32_PerfFormattedData_ASP_ActiveServerPages class, 615
Win32_PerfFormattedData class, 615
Win32_PerfFormattedData_ContentFilter_IndexingServiceFilter class, 615
Win32_PerfFormattedData_ContentIndex_IndexingService class, 615
Win32_PerfFormattedData_InetInfo_InternetInformationServicesGlobal class, 615
Win32_PerfFormattedData_ISAPISearch_HttpIndexingService class, 615
Win32_PerfFormattedData_MSDDTC_DistributedTransactionCoordinator class, 615
Win32_PerfFormattedData_NTFSDRV_SMPNTFSStoreDriver class, 615
Win32_PerfFormattedData_PerfDisk_LogicalDisk class, 615
Win32_PerfFormattedData_PerfDisk_PhysicalDisk class, 615
Win32_PerfFormattedData_PerfNet_Browser class, 615
Win32_PerfFormattedData_PerfNet_Redirector class, 615
Win32_PerfFormattedData_PerfNet_Server class, 616
Win32_PerfFormattedData_PerfNet_ServerWorkQueues class, 616
Win32_PerfFormattedData_PerfOS_Cache class, 616
Win32_PerfFormattedData_PerfOS_Memory class, 616
Win32_PerfFormattedData_PerfOS_Objects class, 616
Win32_PerfFormattedData_PerfOS_PagingFile class, 616
Win32_PerfFormattedData_PerfOS_Processor class, 616
Win32_PerfFormattedData_PerfOS_System class, 616
Win32_PerfFormattedData_PerfProc_FullImage_Costly class, 616
Win32_PerfFormattedData_PerfProc_Image_Costly class, 616
Win32_PerfFormattedData_PerfProc_JobObject class, 616
Win32_PerfFormattedData_PerfProc_JobObjectDetails class, 616
Win32_PerfFormattedData_PerfProc_ProcessAddressSpace_Costly class, 616
Win32_PerfFormattedData_PerfProc_Process class, 616
Win32_PerfFormattedData_PerfProc_Thread class, 617
Win32_PerfFormattedData_PerfProc_ThreadDetails_Costly class, 617
Win32_PerfFormattedData_PSched_PSchedFlow class, 617
Win32_PerfFormattedData_PSched_PSchedPipe class, 617
Win32_PerfFormattedData_RemoteAccess_RASPort class, 617
Win32_PerfFormattedData_RemoteAccess_RASTotal class, 617
Win32_PerfFormattedData_RSVP_ACSRSVPInterfaces class, 617
Win32_PerfFormattedData_RSVP_ACSRSVPService class, 617
Win32_PerfFormattedData_SMTPSVC_SMTPServer class, 617
Win32_PerfFormattedData_Spooler_PrintQueue class, 617
Win32_PerfFormattedData_TapiSrv_Telephony class, 617
Win32_PerfFormattedData_Tcpip_ICMP class, 617
Win32_PerfFormattedData_Tcpip_IP class, 617
Win32_PerfFormattedData_Tcpip_NBTCConnection class, 617
Win32_PerfFormattedData_Tcpip_NetworkInterface class, 617
Win32_PerfFormattedData_Tcpip_TCP class, 617
Win32_PerfFormattedData_Tcpip_UDP class, 618
Win32_PerfFormattedData_TermService_TerminalServices class, 618

Win32_PerfFormattedData_W3SVC_WebService class, 618
 Win32_PerfRawData_ASP_ActiveServerPages class, 618
 Win32_PerfRawData class, 618
 Win32_PerfRawData_ContentFilter_IndexingServiceFilter class, 618
 Win32_PerfRawData_ContentIndex_IndexingService class, 618
 Win32_PerfRawData_InetInfo_InternetInformationServicesGlobal class, 618
 Win32_PerfRawData_ISAPISearch_HttpIndexingService class, 618
 Win32_PerfRawData_MSDTC_DistributedTransactionCoordinator class, 618
 Win32_PerfRawData_NTFSDRV_SMTPTNTFSStoreDriver class, 618
 Win32_PerfRawData_PerfDisk_LogicalDisk class, 618
 Win32_PerfRawData_PerfDisk_PhysicalDisk class, 618
 Win32_PerfRawData_PerfNet_Browser class, 618
 Win32_PerfRawData_PerfNet_Redirector class, 618
 Win32_PerfRawData_PerfNet_Server class, 619
 Win32_PerfRawData_PerfNet_ServerWorkQueues class, 619
 Win32_PerfRawData_PerfOS_Cache class, 619
 Win32_PerfRawData_PerfOS_Memory class, 619
 Win32_PerfRawData_PerfOS_Objects class, 619
 Win32_PerfRawData_PerfOS_PagingFile class, 619
 Win32_PerfRawData_PerfOS_Processor class, 619
 Win32_PerfRawData_PerfOS_System class, 619
 Win32_PerfRawData_PerfProc_FullImage_Costly class, 619
 Win32_PerfRawData_PerfProc_Image_Costly class, 619
 Win32_PerfRawData_PerfProc_JobObject class, 619
 Win32_PerfRawData_PerfProc_JobObjectDetails class, 619
 Win32_PerfRawData_PerfProc_ProcessAddressSpace_Costly class, 619
 Win32_PerfRawData_PerfProc_Process class, 619
 Win32_PerfRawData_PerfProc_Thread class, 619
 Win32_PerfRawData_PerfProc_ThreadDetails_Costly class, 619
 Win32_PerfRawData_PSSched_PSSchedFlow class, 620
 Win32_PerfRawData_PSSched_PSSchedPipe class, 620
 Win32_PerfRawData_RemoteAccess_RASPort class, 620
 Win32_PerfRawData_RemoteAccess_RASTotal class, 620
 Win32_PerfRawData_RSVP_ACSRSVPInterfaces class, 620
 Win32_PerfRawData_RSVP_ACSRSVPService class, 620
 Win32_PerfRawData_SMTSPVC_SMTSPServer class, 620
 Win32_PerfRawData_Spooler_PrintQueue class, 620
 Win32_PerfRawData_TapiSrv_Telephony class, 620
 Win32_PerfRawData_Tcpip_ICMP class, 620
 Win32_PerfRawData_Tcpip_IP class, 620
 Win32_PerfRawData_Tcpip_NBTCConnection class, 620
 Win32_PerfRawData_Tcpip_NetworkInterface class, 620
 Win32_PerfRawData_Tcpip_TCP class, 620
 Win32_PerfRawData_Tcpip_UDP class, 620
 Win32_PerfRawData_TermService_TerminalServices class, 620
 Win32_PerfRawData_TermService_TerminalServicesSession class, 620
 Win32_PerfRawData_W3SVC_WebService class, 620
 Win32_PhysicalMedia class, 598
 Win32_PhysicalMemoryArray class, 599
 Win32_PhysicalMemory class, 599
 Win32_PhysicalMemoryLocation class, 599
 Win32_PingStatus class, 506, 607
 Win32_PNPAllocatedResource class, 599
 Win32_PNPDevice class, 599
 Win32_PNPEntity class, 382, 599
 Win32_PointingDevice class, 597
 Win32_PortableBattery class, 601
 Win32_PortConnector class, 599
 Win32_PortResource class, 600
 Win32_POTSModem class, 602
 Win32_POTSModemToSerialPort class, 602
 Win32_PowerManagementEvent class, 601
 Win32_Printer class, 601
 Win32_PrinterConfiguration class, 601
 Win32_PrinterController class, 601
 Win32_PrinterDriver class, 601
 Win32_PrinterDriverDll class, 601
 Win32_PrinterSetting class, 602
 Win32_PrinterShare class, 612
 Win32_PrintJob class, 602
 Win32_PrivilegesStatus class, 611
 Win32_Process class, 262, 294, 326, 355, 360, 374, 610
 Win32_Processor class, 294, 600

Win32_ProcessStartTrace class

Win32_ProcessStartTrace class, 607
Win32_ProcessStartup class, 610
Win32_ProcessStopTrace class, 607
Win32_ProcessTrace class, 607
Win32_Product class, 126, 516, 518
Win32_ProgramGroup class, 613
Win32_ProgramGroupContents class, 613
Win32_ProgramGroupOrItem class, 613
Win32_ProtocolBinding class, 607
Win32_Proxy class, 615
Win32_QuickFixEngineering class, 609
Win32_QuotaSetting class, 605
Win32_Refrigeration class, 597
Win32_Registry class, 610
Win32_ScheduledJob class, 132, 610
Win32_SCSIController class, 600
Win32_SCSIControllerDevice class, 600
Win32_SecurityDescriptor class, 363, 611
Win32_SecurityDescriptorHelper class, 361, 362
Win32_SecuritySettingAccess class, 611
Win32_SecuritySettingAuditing class, 611
Win32_SecuritySetting class, 611
Win32_SecuritySettingGroup class, 611
Win32_SecuritySettingOfLogicalFile class, 611
Win32_SecuritySettingOfLogicalShare class, 611
Win32_SecuritySettingOfObject class, 611
Win32_SecuritySettingOwner class, 611
Win32_SerialPort class, 600
Win32_SerialPortConfiguration class, 600
Win32_SerialPortSetting class, 600
Win32_ServerConnection class, 612
Win32_ServerSession class, 612
Win32_Service class, 294, 301, 373, 612
Win32_SessionConnection class, 612
Win32_SessionProcess class, 612
Win32_ShadowBy class, 613
Win32_ShadowContext class, 613
Win32_ShadowCopy class, 613
Win32_ShadowDiffVolumeSupport class, 613
Win32_ShadowFor class, 613
Win32_ShadowOn class, 613
Win32_ShadowProvider class, 613
Win32_ShadowStorage class, 613
Win32_ShadowVolumeSupport class, 614
Win32_Share class, 315, 612
Win32_ShareToDirectory class, 612
Win32_ShortcutFile class, 605
Win32_SIDandAttributes class, 606
Win32_SID class, 611
Win32_SMBIOSMemory class, 600
Win32_SoundDevice class, 600
Win32_StartupCommand class, 609
Win32_SubDirectory class, 605
Win32_SystemAccount class, 614
Win32_SystemBIOS class, 600
Win32_SystemBootConfiguration class, 609
Win32_SystemConfigurationChangeEvent class, 608
Win32_SystemDesktop class, 609
Win32_SystemDevices class, 609
Win32_SystemDriver class, 604
Win32_SystemDriverPNPEntity class, 600
Win32_SystemEnclosure class, 600
Win32_SystemLoadOrderGroups class, 609
Win32_SystemLogicalMemoryConfiguration class, 606
Win32_SystemMemoryResource class, 600
Win32_SystemNetworkConnections class, 609
Win32_SystemOperatingSystem class, 609
Win32_SystemPartitions class, 605
Win32_SystemProcesses class, 609
Win32_SystemProgramGroups class, 609
Win32_SystemResources class, 609
Win32_SystemServices class, 609
Win32_SystemSetting class, 609
Win32_SystemSlot class, 600
Win32_SystemSystemDriver class, 610
Win32_SystemTimeZone class, 610
Win32_SystemTrace class, 608
Win32_SystemUsers class, 610
Win32_TapeDrive class, 598
Win32_TCPIPPrinterPort class, 602
Win32_TemperatureProbe class, 597
Win32_Thread class, 610
Win32_ThreadStartTrace class, 608
Win32_ThreadStopTrace class, 608
Win32_ThreadTrace class, 608
Win32_TimeZone class, 604
Win32-TokenGroups class, 606
Win32-TokenPrivileges class, 606
Win32_Trustee class, 611
Win32_UninterruptiblePowerSupply class, 601
Win32_USBController class, 600
Win32_USBControllerDevice class, 600
Win32_USBHub class, 600
Win32_UserAccount class, 132, 376, 614
Win32_UserDesktop class, 604
Win32_UserInDomain class, 614
Win32_VideoConfiguration class, 602
Win32_VideoController class, 602

- Win32_VideoSettings class, 602
- Win32_VoltageProbe class, 601
- Win32_VolumeChangeEvent class, 608
- Win32_Volume class, 605, 614
- Win32_VolumeQuota class, 605
- Win32_VolumeQuotaSetting class, 605
- Win32_VolumeUserQuota class, 605, 614
- Win32_WindowsProductActivation class, 615
- windir variable, 77
- Windows 7, taskbar shortcuts in, 10–11
- Windows 8
 - firewall exceptions for, 114
 - using -force parameter, 81, 82
 - prompts displayed prior to stopping certain processes, 216
 - WinRM in PowerShell Client, 112
- WindowsDeveloperLicense module, 581
- Windows environment variables, 330–335
- WindowsErrorReporting module, 581
- Windows flag key, 10
- Windows Management Framework 3.0 package, 3
- Windows Management Instrumentation. *See* WMI
- Windows Management Instrumentation Tester (WbemTest), 361
- Windows PowerShell. *See* PowerShell
- Windows PowerShell 2.0, 226
- Windows PowerShell console, 53
- Windows PowerShell ISE
 - creating modules in, 238–239
 - IntelliSense in, 256
 - navigating in, 252–254
 - running, 251
 - running commands in, 255
 - script pane in, 254–255
 - snippets in
 - creating code with, 257–259
 - creating user-defined, 259–260
 - defined, 257
 - removing user-defined, 261–262
 - Tab expansion in, 256
- Windows PowerShell remoting
 - discovering information about forest and domain, 428–431
 - obtaining FSMO information using, 428
- Windows Remote Management (WinRM), 3
- Windows Server 2003, 227
- Windows Server 2012, 112
- Windows XP, 227
- WinNT provider, 385
- WinRM (Windows Remote Management), 3
 - configuring, 112–114
 - firewall exceptions, 114
 - overview, 112
 - testing configuration, 113–114
- wjb alias, 68
- WMI classes
 - abstract, 370
 - association classes, 373–378
 - description of, 597–620
 - dynamic, 370
 - list of, 597–620
 - properties of, 597–620
 - retrieving WMI instances
 - cleaning up output from command, 373
 - overview, 371–372
 - reducing returned properties and instances, 372–373
 - using CIM cmdlets to explore
 - filtering classes by qualifier, 369–371
 - finding WMI class methods, 368–369
 - overview, 367
 - retrieving associated WMI classes, 381–382
 - using -classname parameter, 367–368
 - WMI video classes, 380–381
- [wmi] type accelerator, 523, 524
- WMI cmdlets
 - Invoke-WmiMethod cmdlet, 358–360
 - overview, 355–357
 - using terminate method directly, 357–358
- [wmi] type accelerator, 360–361
- WMI Query argument, 320
- WMI Tester (WbemTest), 513, 518
- [wmi] type accelerator, 189, 360–361
- WMI (Windows Management Instrumentation), 1. *See also* WMI classes; WMI cmdlets
 - classes in, 289–293
 - connecting to, default values for, 307–308
 - importance of, 283–284
 - missing providers, handling, 513–523
 - model for, 284
 - namespaces in, 284–288
 - obtaining operating system version using, 174
 - obtaining specific data from, 189
 - providers in, 289
 - queries from bogus users, 463
 - querying
 - eliminating WMI query argument, 320–321
 - finding installed software, 327–330

identifying service accounts

- identifying service accounts, 322–323
- logging service accounts, 323–324
- obtaining BIOS information, 308–311
- using operators, 321–322
- overview, 293
- retrieving data from specific instances of class, 319–320
- retrieving default WMI settings, 308
- retrieving every property from every instance of class, 314
- retrieving information about all shares on local machine, 315
- retrieving list of running processes, 317–318
- retrieving specific properties from class, 316
- shortening syntax, 325–326
- specific class, 293–296
- specifying maximum number of connections to server, 316–317
- substituting Where clause with variable, 325
- viewing Windows environment variables, 330–335
- Win32_Desktop class, 296–298
- working with disk drives, 312–314
- remoting
 - using CIM classes to query WMI classes, 343–344
 - disadvantages of, 341
 - using group policy to configure WMI, 337–338
 - remote results, 344–348
 - supplying alternate credentials for remote connection, 338–341
- using to work with static methods, 361–363, 365–366
- WorkingWithVariables.txt file, 97
- Wrap switch, 255
- write alias, 68
- Write cmdlet, 583
- Write-Debug cmdlet, 174, 463, 464, 464–465, 577
- Write-Error cmdlet, 174, 577
- Write-EventLog cmdlet, 577
- Write-Host cmdlet, 178, 328, 488, 577, 592
- Write mode, 485
- Write-Output cmdlet, 68, 577
- Write-Path function, 176
- Write-Progress cmdlet, 577, 629
- Write-Verbose cmdlet, 209, 519, 520, 577
- Write-Warning cmdlet, 577
- Wscript.Echo command, 133
- Wscript.Quit statement, 161
- WSDL (Web Services Description Language), 190
- wshNetwork object, 61
- wshShell object, 50–52
- WS-Management protocol, 112
- WSMAN (Web Services Management), 108

X

- [xml] alias, 146, 190