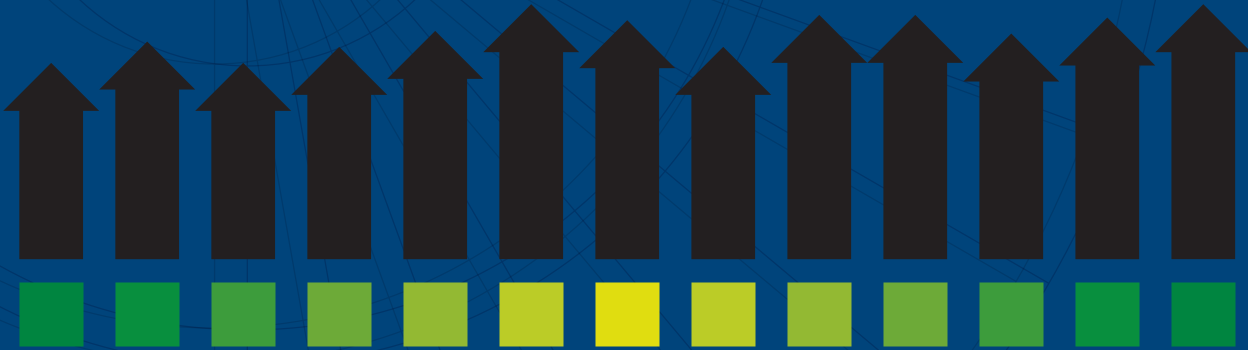
Microsoft

Windows PowerShell



BEST PRACTICES

Ed Wilson

Microsoft

Windows PowerShell Best Practices

Ed Wilson

Published with the authorization of Microsoft Corporation by:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, California 95472

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: 978-0-7356-6649-8

Second Printing: April 2014

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, O'Reilly Media, Inc., 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: Christopher Hearse

Editorial Production: nSight, Inc.

Technical Reviewer: Brian Wilhite

Cover Design: Twist Creative • Seattle

Cover Composition: Ellie Volckhausen

Illustrator: nSight, Inc.

This book is dedicated to Teresa. You make each day feel like it is filled with infinite possibilities.

—Ed Wilson

Contents at a glance

Foreword *xix*

Introduction *xxi*

PART I UNDERSTANDING THE BASICS OF WINDOWS POWERSHELL

CHAPTER 1 Survey of Windows PowerShell capabilities 3

CHAPTER 2 Using the CIM cmdlets 27

PART II PLANNING FOR SCRIPTING

CHAPTER 3 Using the Active Directory module 45

CHAPTER 4 Identifying scripting opportunities 73

CHAPTER 5 Configuring the script environment 111

CHAPTER 6 Avoiding scripting pitfalls 151

CHAPTER 7 Tracking scripting opportunities 195

PART III DESIGNING THE SCRIPT

CHAPTER 8 Designing the script 233

CHAPTER 9 Designing help for scripts 277

CHAPTER 10 Designing modules 311

CHAPTER 11 Handling input and output 339

CHAPTER 12 Handling errors 397

CHAPTER 13 Testing scripts 433

CHAPTER 14 Documenting scripts 475

PART IV DEPLOYING THE SCRIPT

CHAPTER 15 Managing the execution policy 491

CHAPTER 16 Running scripts 507

CHAPTER 17 Versioning scripts 521

CHAPTER 18 Logging results 531

CHAPTER 19 Troubleshooting scripts 559

CHAPTER 20 **Using the Windows PowerShell ISE** **605**
CHAPTER 21 **Using Windows PowerShell remoting and jobs** **615**
CHAPTER 22 **Using Windows PowerShell Workflow** **643**
CHAPTER 23 **Using the Windows PowerShell DSC** **659**

Index *675*

About the Author *705*

Contents

Foreword **xix**

Introduction **xxi**

PART I UNDERSTANDING THE BASICS OF WINDOWS POWERSHELL

Chapter 1 Survey of Windows PowerShell capabilities **3**

Understanding Windows PowerShell 3

Installing Windows PowerShell. 6

Deploying Windows PowerShell 7

- Using cmdlets 7

Using command-line utilities 9

Security issues with Windows PowerShell 11

- Controlling execution of Windows PowerShell cmdlets 11
- Confirming commands 12
- Suspending confirmation of cmdlets 12

Working with Windows PowerShell. 13

- Accessing Windows PowerShell 14
- Configuring Windows PowerShell 15

Supplying options for cmdlets 16

Working with the help options. 17

- Updating help information 17
- Discovering information in help 21

Additional resources. 26

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:

www.microsoft.com/learning/booksurvey/

Chapter 2 Using the CIM cmdlets 27

Using the CIM cmdlets to explore WMI classes. 27

Using the *classname* parameter 27

Finding WMI class methods 29

Filtering classes by qualifier 30

Retrieving WMI instances 33

Reduce returned properties and instances 33

Clean up output from the command 34

Working with Association classes. 35

Additional resources. 41

PART II PLANNING FOR SCRIPTING

Chapter 3 Using the Active Directory module 45

Understanding the Active Directory module. 45

Installing the Active Directory module 47

Getting started with the Active Directory module 47

Using the Active Directory module 48

Finding the FSMO role holders 50

Documenting Active Directory 56

Renaming Active Directory sites 59

Managing users 60

Creating a user 63

Finding and unlocking AD user accounts 64

Finding disabled users 66

Finding unused user accounts 68

Additional resources. 72

Chapter 4 Identifying scripting opportunities 73

Automating routine tasks 73

Automation interface 75

Using *RegRead* to read the registry 77

Using WMI to read the registry 77

Using .NET to read the registry 78

Using intrinsic Windows PowerShell techniques 79

Structured requirements 83

Security requirements 83

Detecting the current user 84

Detecting the user role 96

.NET Framework version requirements 100

Operating system requirements 102

Application requirements 106

Module requirements 108

Additional resources. 109

Chapter 5 Configuring the script environment 111

Configuring a profile 111

Creating aliases 112

Creating functions 116

Passing multiple parameters 120

Creating variables 126

Creating PSDrives 133

Enabling scripting 139

Creating a profile 141

Choosing the correct profile 141

Creating other profiles 143

Accessing functions in other scripts. 147

Creating a function library 147

Using an include file 148

Additional resources. 150

Chapter 6 Avoiding scripting pitfalls 151

Lack of cmdlet support 151

Complicated constructors 153

Version compatibility issues 155

Trapping the operating system version 160

Lack of WMI support 162

Working with objects and namespaces 163

Listing WMI providers 168

Working with WMI classes. 169

Changing settings 173

Modifying values through the registry 175

Lack of .NET Framework support 179

Use of static methods and properties 180

Version dependencies 182

Lack of COM support 182

Lack of external application support 189

Additional resources. 193

Chapter 7 Tracking scripting opportunities 195

Evaluating the need for the script 195

Reading a text file 196

Export command history 203

Fan-out commands 205

Query Active Directory 208

Just use the command line 214

Calculating the benefit from the script 217

Repeatability 219

Documentability 223

Adaptability 225

Script collaboration. 229

Additional resources. 230

PART III DESIGNING THE SCRIPT

Chapter 8 Designing the script 233

Understanding functions. 233

Using functions to provide ease of code reuse 244

Using two input parameters 248

Using a type constraint 253

Using more than two input parameters 257

Using functions to encapsulate business logic 259

Using functions to provide ease of modification .261

Understanding filters 270

Additional resources. 276

Chapter 9 Designing help for scripts 277

Adding help documentation to a script with single-line comments 277

Working with temporary folders 285

Using multiple-line comment tags in Windows PowerShell 4.0. 287

Creating multiple-line comments with comment tags 287

Creating single-line comments with comment tags 288

Using comment-based help 289

The 13 rules for writing effective comments 295

Update documentation when a script is updated 295

Add comments during the development process 296

Write for an international audience 297

Consistent header information 298

Document prerequisites 299

Document deficiencies 300

Avoid useless information 302

Document the reason for the code 303

Use of one-line comments 303

Avoid end-of-line comments 304

Document nested structures 305

Use a standard set of keywords 306

Document the strange and bizarre 307

Additional resources. 310

Chapter 10 Designing modules 311

Understanding modules 311

Locate and load modules 311

Listing available modules 312

Loading modules 316

Install modules 319

Creating a modules folder 319

Working with the *$modulePath* variable 322

Creating a module drive 324

Checking for module dependencies 326

Using a module from a share 330

Creating a module 331

Additional resources. 338

Chapter 11 Handling input and output 339

Choosing the best input method 340

Reading from the command line 340

Using the *Param* statement 348

Working with passwords as input 362

Working with connection strings as input 372

Prompting for input 373

Choosing the best output method 374

Output to the screen 376

Output to file 382

Splitting the output to both the screen and the file 383

Output to email 387

Output from functions 388

Additional resources. 395

Chapter 12 Handling errors 397

Handling missing parameters. 398

Creating a default value for the parameter 398

Making the parameter mandatory 399

Limiting choices. 400

Using *PromptForChoice* to Limit Selections 401

Using ping to identify accessible computers 402

Using the −*contains* Operator to examine the contents of an array 404

Using the −*contains* operator to test for properties 406

Handling missing rights 408
Attempting and failing 410
Checking for rights and exiting gracefully 412
Using *#Requires* *413*

Handling missing WMI providers. 415

Incorrect data types 423

Out of bounds errors 429
Using a boundary checking function 429
Placing limits on the parameter 430

Additional resources. 431

Chapter 13 Testing scripts 433

Using basic syntax checking techniques. 433
Looking for errors 438
Running the script 440
Documenting what you did 442

Conducting performance testing of scripts 444
Using the store and forward approach 445
Using the Windows PowerShell pipeline 446
Evaluating the performance of different versions of a script 450

Using standard parameters. 460
Using the *debug* parameter 460
Using the *Verbose* parameter 462
Using the *whatif* parameter 464

Using *Start-Transcript* to produce a log 468

Advanced script testing 470

Additional resources. 473

Chapter 14 Documenting scripts 475

Getting documentation from help 475

Getting documentation from comments 481

Using the AST parser 484

Additional resources. 487

PART IV DEPLOYING THE SCRIPT

Chapter 15 Managing the execution policy 491

Selecting the appropriate script execution policy. 491
The purpose of script execution policies 492
Understanding the different script execution policies 492
Understanding the Internet zone 493

Deploying the script execution policy. 495
Modifying the registry 495
Using the Set-ExecutionPolicy cmdlet 496
Using Group Policy to deploy the script execution policy 499

Understanding code signing. 504

Additional resources. 505

Chapter 16 Running scripts 507

Logon scripts 507
What to include in logon scripts 509
Methods of calling the logon scripts 512

Script folder 515
Deploy locally 515
Deploy an MSI package locally 515

Stand-alone scripts 515
Diagnostics 516
Reporting and auditing 516

Help desk scripts 517
Avoid editing 517
Provide a good level of help interaction 517

Additional resources. 520

Chapter 17 Versioning scripts 521

Why version control? 521
Avoid introducing errors 522
Enable accurate troubleshooting 523
Track changes 523

Maintain a master listing 523
Maintain compatibility with other scripts 523
Internal version number in the comments 525

Version control software 529

Additional resources. 530

Chapter 18 Logging results 531

Logging to a text file 531
Designing a logging approach 532
Text location 542
Networked log files 548

Logging to the event log. 552
Using the Application log 554
Creating a custom event log 555

Logging to the registry 556

Additional resources. 558

Chapter 19 Troubleshooting scripts 559

Understanding debugging in Windows PowerShell. 559
Working with syntax errors 560
Working with runtime errors 560
Working with logic errors 564

Using the Set-PSDebug cmdlet 567
Tracing the script 568
Stepping through the script 572
Enabling strict mode 581

Debugging scripts. 585
Setting breakpoints 587
Responding to breakpoints 596
Listing breakpoints 597
Enabling and disabling breakpoints 599
Deleting breakpoints 601

Additional resources. 603

Chapter 20 Using the Windows PowerShell ISE 605

Running the Windows PowerShell ISE. 605

Navigating the Windows PowerShell ISE 606

Working with the script pane 608

Tab expansion and IntelliSense 610

Working with Windows PowerShell ISE snippets 611

Using Windows PowerShell ISE snippets to create code 611

Creating new Windows PowerShell ISE snippets 612

Removing user-defined Windows PowerShell ISE snippets 613

Additional resources. 614

Chapter 21 Using Windows PowerShell remoting and jobs 615

Understanding Windows PowerShell remoting 615

Classic remoting 615

WinRM—Windows Remote Management 626

Using Windows PowerShell jobs 634

Additional resources. 641

Chapter 22 Using Windows PowerShell Workflow 643

Why use Windows PowerShell Workflow 643

Workflow requirements 644

A simple workflow 644

Parallel PowerShell 645

Workflow activities 648

Windows PowerShell cmdlets as activities 649

Disallowed core cmdlets 650

Non-automatic cmdlet activities 650

Parallel activities 651

Checkpointing a Windows PowerShell workflow .652

Understanding checkpoints 652

Placing checkpoints 652

Adding checkpoints 653

Adding a sequence activity to a workflow 656

Additional resources. 658

Chapter 23 Using the Windows PowerShell DSC 659

Understanding Desired State Configuration 659
- The DSC process 660
- Configuration parameters 663
- Setting dependencies 665
- Configuration data 666

Controlling configuration drift 671

Additional resources. 673

Index 675

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:

www.microsoft.com/learning/booksurvey/

Foreword

In April 2003, Microsoft's Jeffrey Snover gave me an early peek at PowerShell or, as it was known in its beta days, "Monad." I must admit that, while I fell in love with PoSH at first sight, I was just too darned busy with other work to really get my hands dirty with it for another five years, and I soon realized that boy, had I missed a few memos. "Objects in a pipeline? Is that anything like snakes on a plane?" "Hash tables? Can I get mine with a fried egg?"

Yup, there was a lot to learn, and I nearly wore out Google looking up PoSH-y things. Just about every one of those searches, however, seemed to lead me to the same place: the Hey, Scripting Guy! Blog. I quickly noticed that the blog delivered new articles daily, and so I was very surprised to see that the vast majority of those articles were penned by one guy: Ed Wilson. Since then, I've gotten to know Ed personally, and trust me, he's even funnier and more entertaining in person than he is in print, which brings me to this volume.

If you're a Windows admin, learning Windows PowerShell is an essential (as in you need to do this if you want to remain a Windows admin) task. It's not always an easy one, though, and you will often find yourself wishing for the "answers in the back of the book" so to speak. Well, Ed's written that book, and you're holding the latest edition. Work your way through *Windows PowerShell Best Practices*, actually take the time to try out the examples, and soon you, too, will be automating, scripting, and workflow-ing like mad. Happy PowerShelling!

—Mark Minasi, author of the *Mastering Windows Server* books

P.S. In case you don't already know, objects in a pipeline are way cooler than snakes on a plane. Really.

Introduction

Welcome to *Windows PowerShell Best Practices*, a book that was developed together with the Microsoft Windows PowerShell product group to provide in-depth information about Windows PowerShell and best practices based on real-life experiences with the product in use in different environments. Numerous sidebars are also included that detail experiences from skilled industry professionals such as Enterprise Admins and Windows PowerShell Most Valuable Professionals (MVPs).

The book is largely based on Windows PowerShell 4.0 as it exists on Windows 8.1 and on Windows Server 2012 R2. Because Windows PowerShell introduced Desired State Configuration in Windows PowerShell 4.0, Chapter 23, "Using the Windows PowerShell DSC," must be run on a computer with Windows PowerShell 4.0 installed on it. Nearly all of the material in the other chapters will work without modification on Windows PowerShell 3.0 (on Windows 8 or on Windows Server 2012). A large part of the book also applies to Windows PowerShell 2.0 running on any version of Windows that it installs upon.

Who is this book for?

Microsoft *Windows PowerShell Best Practices* is for anyone tasked with designing, implementing or managing enterprise products. This includes Active Directory Domain Services, System Center, Exchange, and SharePoint products. In addition, it is designed for anyone who either teaches or trains others on Windows PowerShell or even for the MCSE track of courseware. Lastly, power users who want to automate their desktops will also benefit from the explanations, scenarios, and sample scripts.

How is this book organized?

This book is organized into four parts:

- Part I: Understanding the basics of Windows PowerShell
- Part II: Planning for scripting
- Part III: Designing the script
- Part IV: Deploying the script

The first part of this book consists of two chapters that focus on the basics of Windows PowerShell capabilities. This portion of the book is a level setting and would be ideal for anyone just learning Windows PowerShell.

The second part of the book discusses identifying scripting opportunities, the scripting environment, and avoiding scripting pitfalls. This part is also ideal for people learning Windows PowerShell, but it is also a great section for admins experienced with the fundamentals of Windows PowerShell but who need to write new scripts.

The third section of the book talks about how you actually design a script—how you plan for inputs and outputs to the script and how you document your scripts. This is a more advanced section, and it is appropriate for advanced students and for people who write scripts that others are expected to utilize.

The last section of the book talks about deploying scripts—how you run them; how you handle versioning; and how you use remote, workflow, and DSC capabilities in your script. This is appropriate for enterprise admins who are firmly entrenched in DevOps.

System requirements

This book is designed to be used with the following Exchange 2010 software:

- Windows Server 2008 or Windows Server 2008 R2
- 1 GB of RAM
- x64 architecture-based computer with Intel or AMD processor that supports 64 bit
- 1.2 GB of available disk space
- Display monitor capable of 800 × 600 resolution

The following list details the minimum system requirements needed to run the content in the book's companion website:

- Windows XP with the latest service pack installed and the latest updates from Microsoft Update Service
- Display monitor capable of 1024 × 768 resolution
- CD-ROM drive
- Microsoft mouse or compatible pointing device

The companion website

This book features a companion website that makes available to you additional information such as job aids, quick reference guides, and additional Windows PowerShell resources. These elements are included to help you plan and manage your Windows PowerShell organization and apply the book's recommended best practices. The companion website includes the following:

- **Job Aids** Additional documents on most of the chapters that help you to collect and structure your work through the book.
- **Quick Reference Guides** These guides provide an overview of all best practice recommendations in the book as well as a collection of all Internet links referenced in the book.

You can download these files from the companion website, which is located at *http://gallery.technet.microsoft.com/scriptcenter/PowerShell-40-Best-d9e16039*.

Acknowledgements

A book of this scope does not happen without assistance. First I must thank my wife, Teresa Wilson, aka the Scripting Wife. She not only coordinated the acquisition of sidebars, but she also read the entire book at least three times. My technical reviewer, Microsoft PFE Brian Wilhite, was great at catching things that would have made me look silly. He also made numerous suggestions for improving not only the clarity of the writing, but in some cases the accuracy of the code. Brian absolutely rocks. Luckily, the Windows PowerShell community is very enthusiastic and as a result was receptive for my call for sidebars. The high quality of the sidebars, and the diversity of content was fun to read, and in the end makes for a much better book. If you run across one of the authors of the sidebars, make sure you tell them "hi." Lastly, I want to thank Jeffrey Snover, Ken Hansen and the rest of the Windows PowerShell team. They made an awesome product that just keeps getting better and better each year. Windows PowerShell for the win!

Support & feedback

The following sections provide information on errata, book support, feedback, and contact information.

Errata

We have made every effort to ensure the accuracy of this book. If you do find an error, please report it on our Microsoft Press site:

http://aka.ms/PowershellBestPractices/errata

You will find additional information and services for your book on its catalog page. If you need additional support, please e-mail 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 us keep the conversation going! We are on Twitter: *http://twitter.com/MicrosoftPress*.

CHAPTER 3

Using the Active Directory module

- Understanding the Active Directory module
- Using the Active Directory module
- Additional resources

Understanding the Active Directory module

Microsoft made Active Directory Domain Services (AD DS) Windows PowerShell cmdlets available beginning with Windows Server 2008 R2. You can also download and install the Active Directory Management Gateway Service (ADMGS) that provides a web service interface to Active Directory domains or Active Directory Lightweight Directory Services that are running on the same server as the ADMGS. The ADMGS can run on Windows Server 2003 with Service Pack 2 or on Windows Server 2008. On Windows Server 2008 R2 and above, the ADMGS installs as a role and does not require an additional download. When you have one domain controller running Windows Server 2008 R2 (or later) in your domain, you can use the new cmdlets to manage your AD DS installation. Installing the ADMGS Windows Server 2003 or Windows Server 2008 does not make it possible to load the Active Directory module on those machines, but it does permit you to use the Active Directory module from another machine to manage those servers.

INSIDE TRACK

Ashley McGlone, Senior Premier Field Engineer
Microsoft Corporation

Some of us have been using Active Directory since the release candidates in 1999. Others have gotten started with it recently. But we all have one thing in common. We need to automate tasks across hundreds or thousands of users, computers, groups, OUs, and so on.

Over the years, our tools were basically VBScript with ADSI or canned command-line utilities like CSVDE or DSQUERY. (A few of us even used WMI or ADODB to

interface with the directory.) Those legacy scripting techniques faithfully carried us through many implementations and change controls.

But in the autumn of 2009, our tools made a giant leap forward. Windows Server 2008 R2 and the RSAT for Windows 7 introduced the Active Directory PowerShell Module. Wow! What would take 20 lines of VBScript can now be done in a single line of Windows PowerShell.

Here are some great examples of AD PowerShell one-liners:

```
Spreadsheet of stale accounts past 30 days:
Search-ADAccount -AccountInactive -TimeSpan 30 | Export-CSV .\Stale_
Accts.csv

Helpdesk prompt for a user password reset:
Set-ADAccountPassword (Read-Host 'Username') -Reset

Target list of global catalog domain controllers:
(Get-ADForest).GlobalCatalogs
```

In my field experience, I have written some large-scale scripts as well, such as the following:

- Active Directory SID history cleanup and file server ACL migrations
- DNS reorganization and migration to AD-integrated zones
- Security delegation reporting across OUs and GPOs

On the domain controller, this magic is made possible by the Active Directory Web Service (ADWS). The ADWS listens on port 9389 and answers to the Windows PowerShell cmdlets. Whether you're running a quick one-liner or automating across thousands of accounts, this service enables you to read and write directory data with ease.

With each release of Windows Server, the Active Directory module (and now companion modules) grows to support new features. The latest releases offer additional functionality to replace trusty utilities like DCPROMO and REPADMIN. Additionally, the Group Policy module enables further automation of workstation management through Active Directory.

The Active Directory module for Windows PowerShell is no longer new technology. This is a mature product that every administrator needs in their bag of tricks to make the work day go faster. Get started today with one simple Windows PowerShell command: `Import-Module ActiveDirectory`.

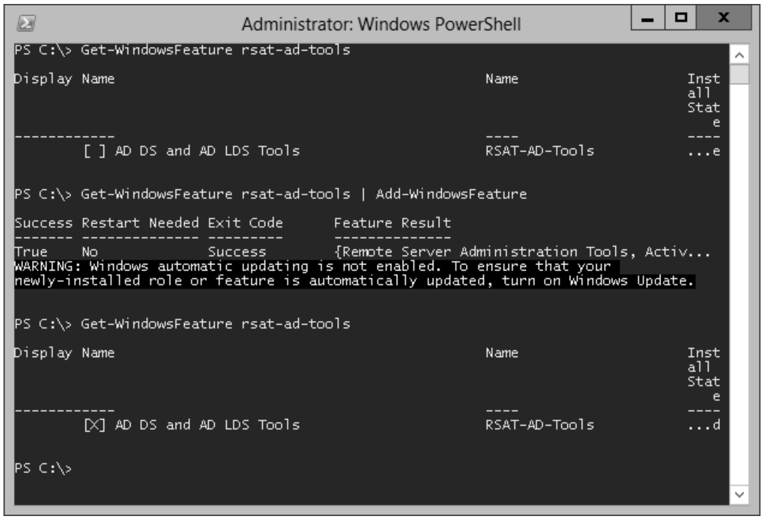
Installing the Active Directory module

The Active Directory module is available beginning with Windows 7 on the client side and with Windows 2008 R2 on servers. To make the cmdlets available on the desktop operating system requires downloading and installing the Remote Server Administration Tools (RSAT).

To install the Active Directory module on either a Windows Server 2012 or Windows Server 2012 R2 machine, you can use the `Add-WindowsFeature` cmdlet because the Active Directory module is directly available to the operating system as an optional Windows feature. Therefore, installation on a server operating system does not require downloading the RSAT tools. To install the RSAT tools for Active Directory, first use the `Get-WindowsFeature` cmdlet to get the rsat-ad-tools, and then pipeline it to the `Add-WindowsFeature` cmdlet. This technique is shown here:

```
Get-WindowsFeature rsat-ad-tools | Add-WindowsFeature
```

The output associated with getting and installing the rsat-ad-tools feature is shown in Figure 3-1.



```
Administrator: Windows PowerShell
PS C:\> Get-WindowsFeature rsat-ad-tools

Display Name                                            Name                Inst
                                                                            all
                                                                            Stat
                                                                               e
------------                                            ----                ----
        [ ] AD DS and AD LDS Tools                      RSAT-AD-Tools       ...e

PS C:\> Get-WindowsFeature rsat-ad-tools | Add-WindowsFeature

Success Restart Needed Exit Code      Feature Result
------- -------------- ---------      --------------
True    No             Success        {Remote Server Administration Tools, Activ...
WARNING: Windows automatic updating is not enabled. To ensure that your
newly-installed role or feature is automatically updated, turn on Windows Update.

PS C:\> Get-WindowsFeature rsat-ad-tools

Display Name                                            Name                Inst
                                                                            all
                                                                            Stat
                                                                               e
------------                                            ----                ----
        [X] AD DS and AD LDS Tools                      RSAT-AD-Tools       ...d

PS C:\>
```

FIGURE 3-1 Installing the RSAT tools provides access to the Active Directory module.

Getting started with the Active Directory module

After you have installed the RSAT tools, you will want to verify that the Active Directory is present and that it loads properly. To do this, use the `Get-Module` cmdlet with the *ListAvailable* switch to verify that the *ActiveDirectory* module is present. Here is the command to do this:

```
Get-Module -ListAvailable ActiveDirectory
```

After the *ActiveDirectory* module loads, you can obtain a listing of the Active Directory cmdlets by using the `Get-Command` cmdlet and specifying the *module* parameter. Because Windows PowerShell 4.0 automatically loads modules, you do not need to use the `Import-Module` cmdlet to import the *ActiveDirectory* module if you do not want to do so. This command is shown here:

```
Get-Command -Module ActiveDirectory
```

Using the Active Directory module

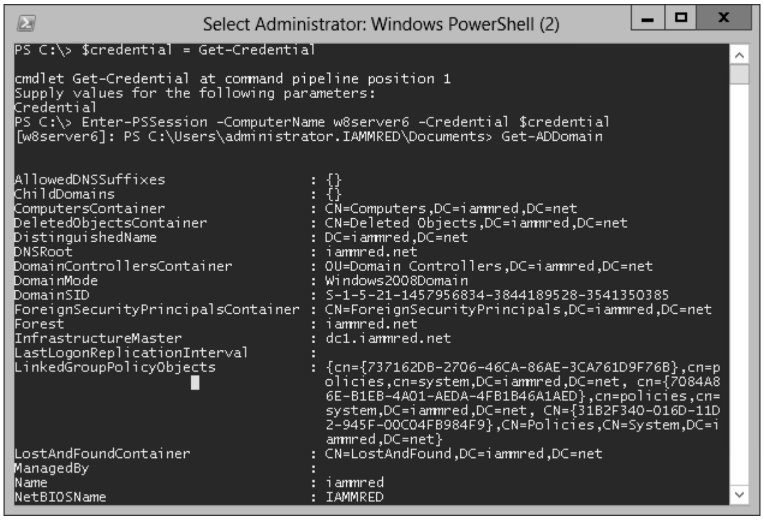
It is not necessary to always load the Active Directory module (or for that matter any module) because Windows PowerShell 3.0 and 4.0 automatically load the module containing a referenced cmdlet. The location searched by Windows PowerShell for modules comes from the environmental variable *PSModulePath*. To view the value of this environmental variable, preface the variable name with the environmental drive. The following command retrieves the default module locations and displays the associated paths:

```
PS C:\> $env:PSModulePath
C:\Users\ed.IAMMRED\Documents\WindowsPowerShell\Modules;C:\Program Files\WindowsPowe
rShell\Modules;C:\Windows\system32\WindowsPowerShell\v1.0\Modules\
```

If you do not want to install the Active Directory module on your client operating systems, all you need to do is to add the rsat-ad-tools feature to at least one server. When installed on the server, use Windows PowerShell remoting to connect to the server hosting the rsat-ad-tools feature from your client workstation. When in the remote session, if the remote server is Windows 8, all you need to do is call one of the Active Directory cmdlets. The *ActiveDirectory* module automatically loads, and the information returns. The following commands illustrate this technique:

```
$credential = get-credential
Enter-PSSession -ComputerName w8Server6 -Credential $credential
Get-ADDomain
```

The technique to use Windows PowerShell remoting to connect to a server that contains the Active Directory module and to automatically load that module while using a cmdlet from that module on Windows PowerShell 4.0 is shown in Figure 3-2.



```
PS C:\> $credential = Get-Credential

cmdlet Get-Credential at command pipeline position 1
Supply values for the following parameters:
Credential
PS C:\> Enter-PSSession -ComputerName w8server6 -Credential $credential
[w8server6]: PS C:\Users\administrator.IAMMRED\Documents> Get-ADDomain


AllowedDNSSuffixes                 : {}
ChildDomains                       : {}
ComputersContainer                 : CN=Computers,DC=iammred,DC=net
DeletedObjectsContainer            : CN=Deleted Objects,DC=iammred,DC=net
DistinguishedName                  : DC=iammred,DC=net
DNSRoot                            : iammred.net
DomainControllersContainer         : OU=Domain Controllers,DC=iammred,DC=net
DomainMode                         : Windows2008Domain
DomainSID                          : S-1-5-21-1457956834-3844189528-3541350385
ForeignSecurityPrincipalsContainer : CN=ForeignSecurityPrincipals,DC=iammred,DC=net
Forest                             : iammred.net
InfrastructureMaster               : dc1.iammred.net
LastLogonReplicationInterval       :
LinkedGroupPolicyObjects           : {cn={737162DB-2706-46CA-86AE-3CA761D9F76B},cn=p
                                     olicies,cn=system,DC=iammred,DC=net, cn={7084A8
                                     6E-B1EB-4A01-AEDA-4FB1B46A1AED},cn=policies,cn=
                                     system,DC=iammred,DC=net, CN={31B2F340-016D-11D
                                     2-945F-00C04FB984F9},CN=Policies,CN=System,DC=i
                                     ammred,DC=net}
LostAndFoundContainer              : CN=LostAndFound,DC=iammred,DC=net
ManagedBy                          :
Name                               : iammred
NetBIOSName                        : IAMMRED
```

FIGURE 3-2 Using Windows PowerShell 4.0 remoting to obtain Active Directory information without first loading the module.

NOTES FROM THE FIELD

Brian Wilhite, Premier Field Engineer (PFE)
Microsoft Corporation

Like most Windows administrators, you probably work with Active Directory on a weekly, if not daily, basis. With Windows PowerShell, working with Active Directory is so much easier than it used to be. In fact, I've forgotten how complex structuring ADSI code can be. When installing a fresh copy of Windows, usually after customizing my profile, I will download and install the Remote Server Administration Tools (RSAT), to ensure that I have the *ActiveDirectory* Module for use within Windows PowerShell. From time to time, my manager has asked me to run a query against Active Directory to determine what computers have been enabled for delegation, for compliance reasons, and to possibly execute a task on those systems. So I turn to Windows PowerShell, with the *ActiveDirectory* module, for the answer.

First, you need to determine what Active Directory attributes to filter for. In my case, I'm looking for any computer object that has a value present for the *msDS-AllowedToDelegateTo* attribute or the *TrustedForDelegation* attribute value set to *true*. The Active Directory module has a cmdlet that will allow me to query Active Directory for these attributes and their settings. Consider the following example:

```
Get-ADComputer '
-Filter {msDS-AllowedToDelegateTo -like "*" -or TrustedForDelegation
-eq "True"} '
-Properties TrustedForDelegation, msDS-AllowedToDelegateTo |
Select Name, TrustedForDelegation, msDS-AllowedToDelegateTo
```

This will return any computer object that is trusted for delegation to any service or specific services. Finally, let's assume that you want to take those computers and query the Windows Updates that have been applied to them. You can run the following one-liner, assuming Windows PowerShell remoting is enabled on the targets, to pipe the results into the `Invoke-Command` cmdlet, launching *Get-HotFix* on the target machine, and storing the results in a variable:

```
$Results = Get-ADComputer '
-Filter {msDS-AllowedToDelegateTo -like "*" -or TrustedForDelegation -eq
"True"} '
-Properties TrustedForDelegation, msDS-AllowedToDelegateTo |
Select Name, TrustedForDelegation, msDS-AllowedToDelegateTo |
ForEach-Object {Invoke-Command -Command {Get-HotFix} -ComputerName
$_.Name}
```

After this runs, which might take a few minutes, given the number of computers, you will have a nice report that you can review. If you wanted to take it a step further, you could take the results variable and pipe it to a CSV file:

```
$Results | Export-Csv -Path C:\Temp\DelegationPatchReport.csv
```

Windows PowerShell with the *ActiveDirectory* module will make a Windows administrator's life easy when given a task big or small.

Finding the FSMO role holders

To find information about domain controllers and FSMO roles, you do not have to write a Windows PowerShell script; you can do it directly from the Windows PowerShell console or ISE by using the Active Directory cmdlets. The first thing that needs to done, more than likely, is to load the *ActiveDirectory* module into the current Windows PowerShell session. While it is possible to add the `import-module` command to your Windows PowerShell profile, in general it is not a good idea to load a bunch of modules that you might or might not use on a regular basis. In fact, you can load all the modules at once by piping the results of the `Get-Module` *–ListAvailable* command to the `Import-Module` cmdlet. This is shown here:

```
PS C:\> Get-Module -ListAvailable | Import-Module
PS C:\> Get-Module
```

```
ModuleType Name                      ExportedCommands
---------- ----                      ----------------
Script     BasicFunctions            {Get-ComputerInfo, Get-OptimalSize}
Script     ConversionModuleV6        {ConvertTo-Feet, ConvertTo-Miles, ConvertTo-...
Script     PowerShellPack            {New-ByteAnimationUsingKeyFrames, New-TiffBi...
Script     PSCodeGen                 {New-Enum, New-ScriptCmdlet, New-PInvoke}
Script     PSImageTools              {Add-CropFilter, Add-RotateFlipFilter, Add-O...
Script     PSRss                     {Read-Article, New-Feed, Remove-Article, Rem...
Script     PSSystemTools             {Test-32Bit, Get-USB, Get-OSVersion, Get-Mul...
Script     PSUserTools               {Start-ProcessAsAdministrator, Get-CurrentUs...
Script     TaskScheduler             {Remove-Task, Get-ScheduledTask, Stop-Task, ...
Script     WPK                       {Get-DependencyProperty, New-ModelVisual3D, ...
Manifest   ActiveDirectory           {Set-ADOrganizationalUnit, Get-ADDomainContr...
Manifest   AppLocker                 {Get-AppLockerPolicy, Get-AppLockerFileInfor...
Manifest   BitsTransfer              {Start-BitsTransfer, Remove-BitsTransfer, Re...
Manifest   FailoverClusters          {Set-ClusterParameter, Get-ClusterParameter,...
Manifest   GroupPolicy               {Get-GPStarterGPO, Get-GPOReport, Set-GPInhe...
Manifest   NetworkLoadBalancingCl... {Stop-NlbClusterNode, Remove-NlbClusterVip, ...
Script     PSDiagnostics             {Enable-PSTrace, Enable-WSManTrace, Start-Tr...
Manifest   TroubleshootingPack       {Get-TroubleshootingPack, Invoke-Troubleshoo...


PS C:\>
```

After you have loaded the Active Directory module, you will want to use the `Get-Command` cmdlet to see the cmdlets that are exported by the module. This is shown here:

```
PS C:\> Get-Module -ListAvailable

ModuleType Name                      ExportedCommands
---------- ----                      ----------------
Script     BasicFunctions            {}
Script     ConversionModuleV6        {}
Script     DotNet                    {}
Manifest   FileSystem                {}
Manifest   IsePack                   {}
Manifest   PowerShellPack            {}
Manifest   PSCodeGen                 {}
Manifest   PSImageTools              {}
Manifest   PSRSS                     {}
Manifest   PSSystemTools             {}
Manifest   PSUserTools               {}
Manifest   TaskScheduler             {}
Manifest   WPK                       {}
Manifest   ActiveDirectory           {}
Manifest   AppLocker                 {}
Manifest   BitsTransfer              {}
Manifest   FailoverClusters          {}
Manifest   GroupPolicy               {}
Manifest   NetworkLoadBalancingCl... {}
Manifest   PSDiagnostics             {}
Manifest   TroubleshootingPack       {}


PS C:\> Import-Module active*
PS C:\> Get-Command -Module active*

CommandType     Name                               Definition
-----------     ----                               ----------
Cmdlet          Add-ADComputerServiceAccount       Add-ADComputerServiceAccount [...
Cmdlet          Add-ADDomainControllerPasswordR... Add-ADDomainControllerPassword...
Cmdlet          Add-ADFineGrainedPasswordPolicy... Add-ADFineGrainedPasswordPolic...
Cmdlet          Add-ADGroupMember                  Add-ADGroupMember [-Identity] ...
Cmdlet          Add-ADPrincipalGroupMembership     Add-ADPrincipalGroupMembership...
Cmdlet          Clear-ADAccountExpiration          Clear-ADAccountExpiration [-Id...
Cmdlet          Disable-ADAccount                  Disable-ADAccount [-Identity] ...
Cmdlet          Disable-ADOptionalFeature          Disable-ADOptionalFeature [-Id...
Cmdlet          Enable-ADAccount                   Enable-ADAccount [-Identity] <...
Cmdlet          Enable-ADOptionalFeature           Enable-ADOptionalFeature [-Ide...
Cmdlet          Get-ADAccountAuthorizationGroup    Get-ADAccountAuthorizationGrou...
Cmdlet          Get-ADAccountResultantPasswordR... Get-ADAccountResultantPassword...
Cmdlet          Get-ADComputer                     Get-ADComputer -Filter <String...
<output truncated>
```

To find a single domain controller, if you are not sure of one in your site, you can use the *discover* switch on the `Get-ADDomainController` cmdlet. One thing to keep in mind is that the *discover* parameter could return information from the cache. If you want to ensure that a fresh *discover* command is sent, use the *forceDiscover* switch in addition to the *–discover* switch. These techniques are shown here:

```
PS C:\> Get-ADDomainController -Discover


Domain      : NWTraders.Com
Forest      : NWTraders.Com
HostName    : {HyperV.NWTraders.Com}
IPv4Address : 192.168.1.100
IPv6Address :
Name        : HYPERV
Site        : NewBerlinSite



PS C:\> Get-ADDomainController -Discover -ForceDiscover


Domain      : NWTraders.Com
Forest      : NWTraders.Com
HostName    : {HyperV.NWTraders.Com}
IPv4Address : 192.168.1.100
IPv6Address :
Name        : HYPERV
Site        : NewBerlinSite



PS C:\>
```

When using the `Get-ADDomainController` cmdlet, a minimal amount of information returns. If you want to see additional information from the domain controller you discovered, you would need to connect to it by using the *identity* parameter. The value of the *identity* property can be an IP address, GUID, host name, or even a NetBIOS sort of name. This technique is shown here:

```
PS C:\> Get-ADDomainController -Identity hyperv


ComputerObjectDN           : CN=HYPERV,OU=Domain Controllers,DC=NWTraders,DC=Com
DefaultPartition           : DC=NWTraders,DC=Com
Domain                     : NWTraders.Com
Enabled                    : True
Forest                     : NWTraders.Com
HostName                   : HyperV.NWTraders.Com
InvocationId               : 6835f51f-2c77-463f-8775-b3404f2748b2
IPv4Address                : 192.168.1.100
IPv6Address                :
IsGlobalCatalog            : True
IsReadOnly                 : False
LdapPort                   : 389
Name                       : HYPERV
NTDSSettingsObjectDN       : CN=NTDS Settings,CN=HYPERV,CN=Servers,CN=NewBerlinSite,
                             CN=Sites,CN=Configuration,DC=NWTraders,DC=Com
OperatingSystem            : Windows Server 2008 R2 Standard
OperatingSystemHotfix      :
OperatingSystemServicePack :
OperatingSystemVersion     : 6.1 (7600)
OperationMasterRoles       : {SchemaMaster, DomainNamingMaster}
Partitions                 : {DC=ForestDnsZones,DC=NWTraders,DC=Com, DC=DomainDnsZon
                             es,DC=NWTraders,DC=Com, CN=Schema,CN=Configuration,DC=N
                             WTraders,DC=Com, CN=Configuration,DC=NWTraders,DC=Com...}
ServerObjectDN             : CN=HYPERV,CN=Servers,CN=NewBerlinSite,CN=Sites,CN=Confi
                             guration,DC=NWTraders,DC=Com
ServerObjectGuid           : ab5e2830-a4d6-47f8-b2b4-25757153653c
Site                       : NewBerlinSite
SslPort                    : 636



PS C:\>
```

As shown in the preceding output, the server named Hyperv is a Global Catalog server. It also holds the SchemaMaster and the DomainNamingMaster FSMO roles. It is running Windows Server 2008 R2 Standard edition, which shows that the cmdlet works with down-level versions of the operating system. The `Get-ADDomainController` cmdlet accepts a *filter* parameter that can be used to perform a search and retrieve operation. It uses a special search syntax that is discussed in the online help about files. Unfortunately, it does not accept LDAP syntax.

Luckily, you do not have to learn the special filter syntax, because the `Get-ADObject` cmdlet will accept a LDAP dialect filter. You can simply pipeline the results of the `Get-ADObject` cmdlet to the `Get-ADDomainController` cmdlet. This technique is shown here:

```
PS C:\> Get-ADObject -LDAPFilter "(objectclass=computer)" -searchbase "ou=domain
controllers,dc=nwtraders,dc=com" | Get-ADDomainController


ComputerObjectDN           : CN=HYPERV,OU=Domain Controllers,DC=NWTraders,DC=Com
DefaultPartition           : DC=NWTraders,DC=Com
Domain                     : NWTraders.Com
Enabled                    : True
Forest                     : NWTraders.Com
HostName                   : HyperV.NWTraders.Com
InvocationId               : 6835f51f-2c77-463f-8775-b3404f2748b2
IPv4Address                : 192.168.1.100
IPv6Address                :
IsGlobalCatalog            : True
IsReadOnly                 : False
LdapPort                   : 389
Name                       : HYPERV
NTDSSettingsObjectDN       : CN=NTDS Settings,CN=HYPERV,CN=Servers,CN=NewBerlinSite,
                             CN=Sites,CN=Configuration,DC=NWTraders,DC=Com
OperatingSystem            : Windows Server 2008 R2 Standard
OperatingSystemHotfix      :
OperatingSystemServicePack :
OperatingSystemVersion     : 6.1 (7600)
OperationMasterRoles       : {SchemaMaster, DomainNamingMaster}
Partitions                 : {DC=ForestDnsZones,DC=NWTraders,DC=Com, DC=DomainDnsZones,
                             DC=NWTraders,DC=Com, CN=Schema,CN=Configuration,DC
                             =NWTraders,DC=Com, CN=Configuration,DC=NWTraders,DC=Com...}
ServerObjectDN             : CN=HYPERV,CN=Servers,CN=NewBerlinSite,CN=Sites,CN=Confi
                             guration,DC=NWTraders,DC=Com
ServerObjectGuid           : ab5e2830-a4d6-47f8-b2b4-25757153653c
Site                       : NewBerlinSite
SslPort                    : 636

ComputerObjectDN           : CN=DC1,OU=Domain Controllers,DC=NWTraders,DC=Com
DefaultPartition           : DC=NWTraders,DC=Com
Domain                     : NWTraders.Com
Enabled                    : True
Forest                     : NWTraders.Com
HostName                   : DC1.NWTraders.Com
InvocationId               : fb324ced-bd3f-4977-ae69-d6763e7e029a
IPv4Address                : 192.168.1.101
IPv6Address                :
IsGlobalCatalog            : True
IsReadOnly                 : False
LdapPort                   : 389
Name                       : DC1
NTDSSettingsObjectDN       : CN=NTDS Settings,CN=DC1,CN=Servers,CN=NewBerlinSite,CN=
                             Sites,CN=Configuration,DC=NWTraders,DC=Com
OperatingSystem            : Windows Serverr 2008 Standard without Hyper-V
```

```
OperatingSystemHotfix      :
OperatingSystemServicePack : Service Pack 2
OperatingSystemVersion     : 6.0 (6002)
OperationMasterRoles       : {PDCEmulator, RIDMaster, InfrastructureMaster}
Partitions                 : {DC=ForestDnsZones,DC=NWTraders,DC=Com, DC=DomainDnsZones,
                             DC=NWTraders,DC=Com, CN=Schema,CN=Configuration,DC
                             =NWTraders,DC=Com, CN=Configuration,DC=NWTraders,DC=Com...}
ServerObjectDN             : CN=DC1,CN=Servers,CN=NewBerlinSite,CN=Sites,CN=Configur
                             ation,DC=NWTraders,DC=Com
ServerObjectGuid           : 80885b47-5a51-4679-9922-d6f41228f211
Site                       : NewBerlinSite
SslPort                    : 636


PS C:\>
```

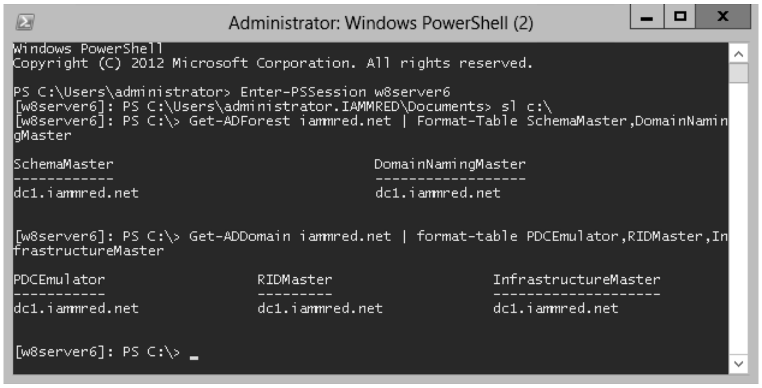
If it returns too much information, the Active Directory cmdlets work just like any other Windows PowerShell cmdlet and therefore permit using the pipeline to choose the information you want to display. To obtain only the FSMO information, it comes down to two commands—three commands if you want to include importing the Active Directory module in your count, or four commands if you need to make a remote connection to a domain controller to run the commands. One cool thing about using Windows PowerShell remoting is that you specify the credentials that you need to run the command. If your normal account is a standard user, you use an elevated account only when you require performing actions with elevated rights. If you have already started the Windows PowerShell console with elevated credentials, you can skip typing in credentials when you enter the remote Windows PowerShell session (assuming that the elevated account also has rights on the remote server). The first two commands seen here create a remote session on a remote domain controller and load the *ActiveDirectory* module:

```
Enter-PSSession w8Server6
```

When the Active Directory module loads, you type a one-line command to get the Forest FSMO roles, and you type another one-line command to get the domain FSMO roles. These two commands are shown here:

```
Get-ADForest iammred.net | Format-Table SchemaMaster,DomainNamingMaster
Get-ADDomain iammred.net | format-table PDCEmulator,RIDMaster,InfrastructureMaster
```

That is it—two or three one-line commands, depending on how you want to count. Even at worst case, three one-line commands are much easier to type than 33 lines of code that would be required if you did not have access to the Active Directory module. In addition, the Windows PowerShell code is much easier to read and to understand. The commands and the associated output from the Windows PowerShell commands appear in Figure 3-3.



```
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

PS C:\Users\administrator> Enter-PSSession w8server6
[w8server6]: PS C:\Users\administrator.IAMMRED\Documents> sl c:\
[w8server6]: PS C:\> Get-ADForest iammred.net | Format-Table SchemaMaster,DomainNamin
gMaster

SchemaMaster                                  DomainNamingMaster
------------                                  ------------------
dc1.iammred.net                               dc1.iammred.net


[w8server6]: PS C:\> Get-ADDomain iammred.net | format-table PDCEmulator,RIDMaster,In
frastructureMaster

PDCEmulator                  RIDMaster                   InfrastructureMaster
-----------                  ---------                   --------------------
dc1.iammred.net              dc1.iammred.net             dc1.iammred.net


[w8server6]: PS C:\> _
```

FIGURE 3-3 Using Windows PowerShell remoting to obtain FSMO information.

Documenting Active Directory

Using the Microsoft Active Directory Windows PowerShell cmdlets and remoting, you can easily discover information about the forest and the domain. The first thing you need to do is to enter a *PSSession* on the remote computer. To do this you use the `Enter-PSSession` cmdlet. Next, you import the Active Directory module and set the working location to the root of the C drive. The reason for setting the working location to the root of the C drive is to regain valuable command-line space. These commands are shown here:

```
PS C:\Users\Administrator.NWTRADERS> Enter-PSSession dc1
[dc1]: PS C:\Users\Administrator\Documents> Import-Module activedirectory
[dc1]: PS C:\Users\Administrator\Documents> Set-Location c:\
```

After you have connected to the remote domain controller, you can use the `Get-WmiObject` cmdlet to verify the operating system on that computer. This command and associated output are shown here:

```
[dc1]: PS C:\> Get-WmiObject win32_operatingsystem
SystemDirectory : C:\Windows\system32
Organization    :
BuildNumber     : 7601
RegisteredUser  : Windows User
SerialNumber    : 55041-507-0212466-84005
Version         : 6.1.7601
```

Now you want to get information about the forest. To do this, you use the `Get-ADForest` cmdlet. The output from the `Get-ADForest` cmdlet includes lots of great information, such as the Domain Naming Master, Forest Mode, Schema Master, and Domain Controllers. This command and associated output appears here:

```
[dc1]: PS C:\> Get-ADForest
ApplicationPartitions : {DC=DomainDnsZones,DC=nwtraders,DC=com, DC=ForestDnsZones,DC
=nwtraders,DC=com}
CrossForestReferences : {}
```

```
DomainNamingMaster    : DC1.nwtraders.com
Domains               : {nwtraders.com}
ForestMode            : Windows2008Forest
GlobalCatalogs        : {DC1.nwtraders.com}
Name                  : nwtraders.com
PartitionsContainer   : CN=Partitions,CN=Configuration,DC=nwtraders,DC=com
RootDomain            : nwtraders.com
SchemaMaster          : DC1.nwtraders.com
Sites                 : {Default-First-Site-Name}
SPNSuffixes           : {}
UPNSuffixes           : {}
```

Now, to obtain information about the domain, use the `Get-ADDomain` cmdlet. The command returns important information such as the location of the default domain controller OU, the PDC emulator, and the RID master. The command and associated output are shown here:

```
[dc1]: PS C:\> Get-ADDomain
AllowedDNSSuffixes                 : {}
ChildDomains                       : {}
ComputersContainer                 : CN=Computers,DC=nwtraders,DC=com
DeletedObjectsContainer            : CN=Deleted Objects,DC=nwtraders,DC=com
DistinguishedName                  : DC=nwtraders,DC=com
DNSRoot                            : nwtraders.com
DomainControllersContainer         : OU=Domain Controllers,DC=nwtraders,DC=com
DomainMode                         : Windows2008Domain
DomainSID                          : S-1-5-21-909705514-2746778377-2082649206
ForeignSecurityPrincipalsContainer : CN=ForeignSecurityPrincipals,DC=nwtraders,DC=com
Forest                             : nwtraders.com
InfrastructureMaster               : DC1.nwtraders.com
LastLogonReplicationInterval       :
LinkedGroupPolicyObjects           : {CN={31B2F340-016D-11D2-945F-00C04FB984F9},CN
                                     =Policies,CN=System,DC=nwtraders,DC=com}
LostAndFoundContainer              : CN=LostAndFound,DC=nwtraders,DC=com
ManagedBy                          :
Name                               : nwtraders
NetBIOSName                        : NWTRADERS
ObjectClass                        : domainDNS
ObjectGUID                         : 0026d1fc-2e4d-4c35-96ce-b900e9d67e7c
ParentDomain                       :
PDCEmulator                        : DC1.nwtraders.com
QuotasContainer                    : CN=NTDS Quotas,DC=nwtraders,DC=com
ReadOnlyReplicaDirectoryServers    : {}
ReplicaDirectoryServers            : {DC1.nwtraders.com}
RIDMaster                          : DC1.nwtraders.com
SubordinateReferences              : {DC=ForestDnsZones,DC=nwtraders,DC=com,
                                     DC=DomainDnsZones,DC=nwtraders,DC=com,
                                     CN=Configuration,DC=nwtraders,DC=com}
SystemsContainer                   : CN=System,DC=nwtraders,DC=com
UsersContainer                     : CN=Users,DC=nwtraders,DC=com
```

From a security perspective, you should always check the domain password policy. To do this, use the `Get-ADDefaultDomainPasswordPolicy` cmdlet. Things you want to pay attention to are the use of complex passwords, minimum password length, password age, and password retention. You also need to check the account lockout policy. This policy is especially important to review closely when inheriting a new network. Here is the command and associated output that does that very thing:

```
[dc1]: PS C:\> Get-ADDefaultDomainPasswordPolicy
ComplexityEnabled           : True
DistinguishedName           : DC=nwtraders,DC=com
LockoutDuration             : 00:30:00
LockoutObservationWindow    : 00:30:00
LockoutThreshold            : 0
MaxPasswordAge              : 42.00:00:00
MinPasswordAge              : 1.00:00:00
MinPasswordLength           : 7
objectClass                 : {domainDNS}
objectGuid                  : 0026d1fc-2e4d-4c35-96ce-b900e9d67e7c
PasswordHistoryCount        : 24
ReversibleEncryptionEnabled : False
```

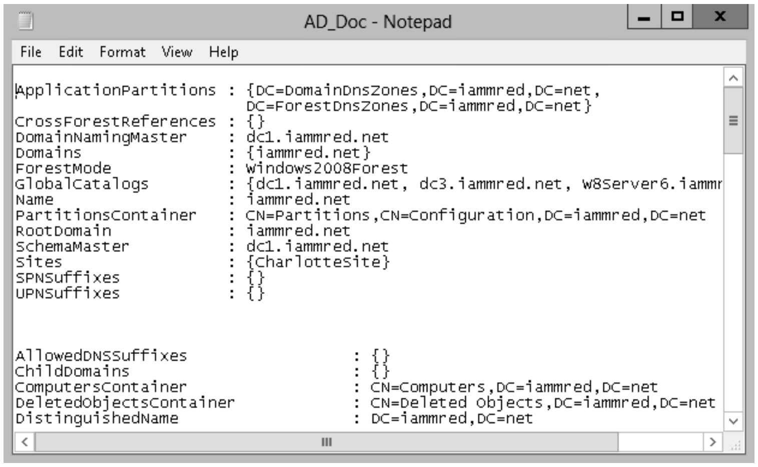
The last things to check are the domain controllers themselves. To do this, use the `Get-ADDomainController` cmdlet. This command returns important information, such as whether the domain controller is read-only, a global catalog server, operations master roles held, and operating system information. Here is the command and associated output:

```
 [dc1]: PS C:\> Get-ADDomainController -Identity dc1
ComputerObjectDN           : CN=DC1,OU=Domain Controllers,DC=nwtraders,DC=com
DefaultPartition           : DC=nwtraders,DC=com
Domain                     : nwtraders.com
Enabled                    : True
Forest                     : nwtraders.com
HostName                   : DC1.nwtraders.com
InvocationId               : b51f625f-3f60-44e7-8577-8918f7396c2a
IPv4Address                : 10.0.0.1
IPv6Address                :
IsGlobalCatalog            : True
IsReadOnly                 : False
LdapPort                   : 389
Name                       : DC1
NTDSSettingsObjectDN       : CN=NTDS Settings,CN=DC1,CN=Servers,CN=Default-First-Site-Na
me,CN=Sites,CN=Configuration,DC=nwtraders,DC=com
OperatingSystem            : Windows Server 2008 R2 Enterprise
OperatingSystemHotfix      :
OperatingSystemServicePack : Service Pack 1
OperatingSystemVersion     : 6.1 (7601)
OperationMasterRoles       : {SchemaMaster, DomainNamingMaster, PDCEmulator,
                             RIDMaster...}
Partitions                 : {DC=ForestDnsZones,DC=nwtraders,DC=com, DC=DomainDnsZones,
                             DC=nwtraders,DC=com, CN=Schema,CN=Configuration,
                             DC=nwtraders,DC=com, CN=Configuration,DC=nwtraders,
                             DC=com...}
ServerObjectDN             : CN=DC1,CN=Servers,CN=Default-First-Site-Name,CN=Sites,
                             CN=Configuration,DC=nwtraders,DC=com
ServerObjectGuid           : 5ae1fd0e-bc2f-42a7-af62-24377114e03d
Site                       : Default-First-Site-Name
SslPort                    : 636
```

To produce a report is as easy as redirecting the output to a text file. These commands gather the information discussed earlier in this section and store the retrieved information in a file named AD_Doc.txt. The commands also illustrate that it is possible to redirect the information to a file stored in a network share.

```
Get-ADForest >> \\dc1\shared\AD_Doc.txt
Get-ADDomain >> \\dc1\shared\AD_Doc.txt
Get-ADDefaultDomainPasswordPolicy >> \\dc1\shared\AD_Doc.txt
Get-ADDomainController -Identity dc1 >>\\dc1\shared\AD_Doc.txt
```

The file as viewed in Notepad appears in Figure 3-4.



```
ApplicationPartitions : {DC=DomainDnsZones,DC=iammred,DC=net,
                        DC=ForestDnsZones,DC=iammred,DC=net}
CrossForestReferences : {}
DomainNamingMaster    : dc1.iammred.net
Domains               : {iammred.net}
ForestMode            : Windows2008Forest
GlobalCatalogs        : {dc1.iammred.net, dc3.iammred.net, W8Server6.iammr
Name                  : iammred.net
PartitionsContainer   : CN=Partitions,CN=Configuration,DC=iammred,DC=net
RootDomain            : iammred.net
SchemaMaster          : dc1.iammred.net
Sites                 : {CharlotteSite}
SPNSuffixes           : {}
UPNSuffixes           : {}



AllowedDNSSuffixes                 : {}
ChildDomains                       : {}
ComputersContainer                 : CN=Computers,DC=iammred,DC=net
DeletedObjectsContainer            : CN=Deleted Objects,DC=iammred,DC=net
DistinguishedName                  : DC=iammred,DC=net
```

FIGURE 3-4 Active Directory documentation displayed in Notepad.

Renaming Active Directory sites

It is easy to rename a site. All you need to do is to right-click the site and select Rename from the action menu. By default, the first site is called Default-First-Site-Name, which is not too illuminating. To work with Active Directory sites, it is necessary to understand that they are a bit strange. First, they reside in the configuration naming context. Connecting to this context by using the Active Directory module is rather simple. Just use the `Get-ADRootDSE` cmdlet, and then select the *ConfigurationNamingContext* property. First, you have to make a connection to the domain controller and import the Active Directory Module (assuming that you do not have the RSAT tools installed on your client computer). This is shown here:

```
Enter-PSSession -ComputerName dc3 -Credential iammred\administrator
Import-Module activedirectory
```

Here is the code that will retrieve all of the sites. It uses the `Get-ADObject` cmdlet to search the configuration naming context for objects that have the object class of *site*.

```
Get-ADObject -SearchBase (Get-ADRootDSE).ConfigurationNamingContext -filter "objectclass
-eq 'site'"
```

When you have the site you want to work with, you first change the *DisplayName* attribute. To do this, you pipeline the site object to the `Set-ADOObject` cmdlet. The `Set-ADOObject` cmdlet allows me to set a variety of attributes on an object. This command is shown here. (This is a single command that is broken into two pieces at the pipeline character.)

```
Get-ADObject -SearchBase (Get-ADRootDSE).ConfigurationNamingContext -filter "objectclass
-eq 'site'" | Set-ADObject -DisplayName CharlotteSite
```

When you have set the *DisplayName* attribute, you decide to rename the object itself. To do this, you use another cmdlet called `Rename-ADObject`. Again, to simplify things, you pipeline the site object to the cmdlet and you assign a new name for the site. This command is shown here. (This is also a one-line command broken at the pipe.)

```
Get-ADObject -SearchBase (Get-ADRootDSE).ConfigurationNamingContext -filter "objectclass
-eq 'site'" | Rename-ADObject -NewName CharlotteSite
```

Managing users

To create a new Organizational Unit, you use the `New-ADOrganizationalUnit` cmdlet as shown here:

```
New-ADOrganizationalUnit -Name TestOU -Path "dc=nwtraders,dc=com"
```

If you want to create a child Organizational Unit (OU), you use the `New-ADOrganizationalUnit` cmdlet, but in the path, you list the location that will serve as the parent, as shown here:

```
New-ADOrganizationalUnit -Name TestOU1 -Path "ou=TestOU,dc=nwtraders,dc=com"
```

If you want to make several child OUs in the same location, use the up arrow to retrieve the previous command and edit the name of the child. You can use the home key to move to the beginning of the line, the end key to move to the end of the line, and the left and right arrow keys to find your place on the line so that you can edit it. A second child OU is created here:

```
New-ADOrganizationalUnit -Name TestOU2 -Path "ou=TestOU,dc=nwtraders,dc=com"
```

To create a computer account in one of the newly created child Organizational Units, you must type the complete path to the OU that will house the new computer account. The `New-ADComputer` cmdlet is used to create new computer accounts in AD DS. In this example, the TestOU1 OU is a child of the TestOU OU, and therefore, both OUs must appear in the path parameter. Keep in mind that the path that is supplied to the *path* parameter must be contained inside quotation marks, as shown here:

```
New-ADComputer -Name Test -Path "ou=TestOU1,ou=TestOU,dc=nwtraders,dc=com"
```

To create a user account, you use the `New-ADUser` cmdlet as shown here:

```
New-ADUser -Name TestChild -Path "ou=TestOU1,ou=TestOU,dc=nwtraders,dc=com"
```

Because there could be a bit of typing involved that tends to become redundant, you might want to write a script to create the OUs at the same time that the computer and user accounts are created. A sample script that creates OUs, users, and computers is the UseADCmdletsToCreateOuComputerAndUser.ps1 script shown here.

UseADCmdletsToCreateOuComputerAndUser.ps1

```
Import-Module -Name ActiveDirectory
$Name = "ScriptTest"
$DomainName = "dc=nwtraders,dc=com"
$OUPath = "ou={0},{1}" -f $Name, $DomainName

New-ADOrganizationalUnit -Name $Name -Path $DomainName
-ProtectedFromAccidentalDeletion $false

For($you = 0; $you -le 5; $you++)
{
 New-ADOrganizationalUnit -Name $Name$you -Path $OUPath
-ProtectedFromAccidentalDeletion $false
}

For($you = 0 ; $you -le 5; $you++)
{
 New-ADComputer -Name  "TestComputer$you" -Path $OUPath
 New-ADUser -Name "TestUser$you" -Path $OUPath
}
```

The UseADCmdletsToCreateOuComputerAndUser.ps1 script begins by importing the Active Directory module. It then creates the first OU. When testing a script, it is important to disable the deletion protection by using the *ProtectedFromAccidentalDeletion* parameter. This will allow you to easily delete the OU and avoid having to go into the advanced view in Active Directory Users And Computers and changing the protected status on each OU.

After the ScriptTest OU is created, the other OUs, users, and computer accounts can be created inside the new location. It seems obvious that you cannot create a child OU inside the parent OU if the parent has not yet been created, but it is easy to make a logic error like this.

To create a new global security group, use the `New-ADGroup` Windows PowerShell AD DS cmdlet. The `New-ADGroup` Windows PowerShell cmdlet requires three parameters: the *name* of the group, a *path* to the location where the group will be stored, and the *groupscope*, which can be global, universal, or domain local. Before running the command shown here, remember that you must import the Active Directory module into your current Windows PowerShell session.

```
New-ADGroup -Name TestGroup -Path "ou=TestOU,dc=nwtraders,dc=com" -groupScope global
```

To create a new universal group, you need to change only the *groupscope* parameter value as shown here:

```
New-ADGroup -Name TestGroup1 -Path "ou=TestOU,dc=nwtraders,dc=com" -groupScope universal
```

To add a user to a group, you must supply values for the *identity* parameter and for the *members* parameter. The value that you use for the identity parameter is the name of the group. You do not need to use the LDAP syntax of *cn=groupname*; you need to supply only the name. Use ADSI Edit to examine the requisite LDAP attributes needed for a group in ADSI Edit.

It is a bit unusual that the *members* parameter is named members and not member because most Windows PowerShell cmdlet parameter names are singular and not plural. The parameters are singular even when they accept an array of values (such as the *computername* parameter). The command to add a new group named TestGroup1 to the UserGroupTest group is shown here:

```
Add-ADGroupMember -Identity TestGroup1 -Members UserGroupTest
```

To remove a user from a group, use the `Remove-ADGroupMember` cmdlet with the name of the user and group. The *identity* and the *members* parameters are required, but the command will not execute without confirmation, as shown here:

```
PS C:\> Remove-ADGroupMember -Identity TestGroup1 -Members UserGroupTest

Confirm
Are you sure you want to perform this action?
Performing operation "Set" on Target "CN=TestGroup1,OU=TestOU,DC=NWTraders,DC=Com".
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?] Help (default is "Y"):
y
PS C:\>
```

If you are sure that you want to remove the user from the group and that you want to suppress the query, you use the *confirm* parameter and assign the value *$false* to it. The problem is that you will need to supply a colon between the parameter and *$false* value.

NOTE **The use of the colon after the *confirm* parameter is not documented, but the technique works on several different cmdlets.**

The command is shown here:

```
Remove-ADGroupMember -Identity TestGroup1 -Members UserGroupTest -Confirm:$false
```

You need the ability to suppress the confirmation prompt to be able to use the `Remove-ADGroupMember` cmdlet in a script. The first thing the RemoveUserFromGroup.ps1 script does is load the Active Directory module. When the module is loaded, the `Remove-ADGroupMember` cmdlet is used to remove the user from the group. To suppress the confirmation prompt, the *–confirm:$false* command is used. The RemoveUserFromGroup.ps1 script is shown here.

RemoveUserFromGroup.ps1

```
import-module activedirectory
Remove-ADGroupMember -Identity TestGroup1 -Members UserGroupTest -Confirm:$false
```

Creating a user

Now create a new user in Active Directory. You will name the user "ed." The command to create a new user is simple; it is `New-Aduser` and the user name. The command to create a disabled user account in the *users* container in the default domain is shown here:

```
new-aduser -name ed
```

When the preceding command that creates a new user completes, nothing is returned to the Windows PowerShell console. To check to ensure that the user is created, use the `Get-Aduser` cmdlet to retrieve the user object. This command is shown here:

```
Get-aduser ed
```

When you are certain that your new user is created, you decide to create an organizational unit to store the user account. The command to create a new organizational unit off the root of the domain is shown here:

```
new-ADOrganizationalUnit scripting
```

Just like the previously used `New-Aduser` cmdlet, nothing returns to the Windows PowerShell console. If you use the `Get-ADOrganizationalUnit` cmdlet, you must use a different methodology. A simple `Get-AdOrganizationalUnit` command returns an error; therefore, you use an *LDAPFilter* parameter to find the OU. The command using the *LDAPFilter* parameter to find my newly created OU is shown here:

```
Get-ADOrganizationalUnit –LDAPFilter "(name=scripting)"
```

Now that you have a new user and a new OU, you need to move the user from the *users* container to the newly created *scripting* OU. To do that, you use the `Move-ADObject` cmdlet. You first get the *distinguishedname* attribute for the scripting OU and store it in a variable called *$oupath*. Next, you use the `Move-ADObject` cmdlet to move the *ed* user to the new OU. The trick here is that where the `Get-AdUser` cmdlet can find a user with the name of *ed*, the `Move-ADObject` cmdlet must have the *distinguishedname* of the *ed* user object to move it. The error that occurs when not supplying the *distinguishedname* appears in the figure that follows. You could use the `Get-AdUser` cmdlet to retrieve the *distinguishedname* in a similar method as you did with the scripting OU.

The next thing you need to do is to enable the user account. To do this, you need to assign a password to the user account. The password must be a secure string. To do this, you can use the `ConvertTo-SecureString` cmdlet. By default, warnings display about converting text to a secure string, but these prompts are suppressible by using the *force* parameter. Here is the command you use to create a secure string for a password:

```
$pwd = ConvertTo-SecureString -String "P@ssword1" -AsPlainText –Force
```

Now that you have created a secure string to use for a password for my user account, you call the `Set-ADAccountPassword` cmdlet to set the password. Because this is a new password, you need to use the *newpassword* parameter. In addition, because you do not have a previous password, you use the *reset* parameter. This command is shown here:

```
Set-ADAccountPassword -Identity ed -NewPassword $pwd –Reset
```

After the account has a password, you can enable the account. To do this, you use the `Enable-ADAccount` cmdlet and specify the user name to enable. This command is shown here:

```
Enable-ADAccount -Identity ed
```

As with the previous commands, none of the cmdlets return any information. To ensure that you have actually enabled the *ed* user account, you use the `Get-ADUser` cmdlet. In the output, you are looking for the value of the *enabled* property. The *enabled* property is a Boolean, so expect the value to be true.

Finding and unlocking AD user accounts

When using the Microsoft Active Directory cmdlets, locating locked out users is a snap. In fact, the `Search-ADAccount` cmdlet even has a *LockedOut* switch. Use the `Search-ADAccount` cmdlet with the *LockedOut* parameter. This command is shown here:

```
Search-ADAccount –LockedOut
```

NOTE **Many network administrators who spend the majority of their time working with Active Directory import the Active Directory module via their Windows PowerShell profile. This way, they never need to worry about the initial performance hit that occurs due to autoloading the Active Directory module.**

The `Search-ADAccount` command and the associated output are shown here:

```
[w8server6]: PS C:\> Search-ADAccount -LockedOut

AccountExpirationDate :
DistinguishedName     : CN=kimakers,OU=test,DC=iammred,DC=net
Enabled               : True
LastLogonDate         : 1/24/2012 8:40:29 AM
LockedOut             : True
Name                  : kimakers
ObjectClass           : user
ObjectGUID            : d907fa99-cd08-435f-97de-1e99d0eb485d
PasswordExpired       : False
PasswordNeverExpires  : False
```

```
SamAccountName          : kimakers
SID                     : S-1-5-21-1457956834-3844189528-3541350385-1608
UserPrincipalName       : kimakers@iammred.net

[w8server6]: PS C:\>
```

You can unlock the locked out user account as well—assuming that you have permission. In Figure 3-5, you attempt to unlock the user account with an account that is a normal user, and an error arises.

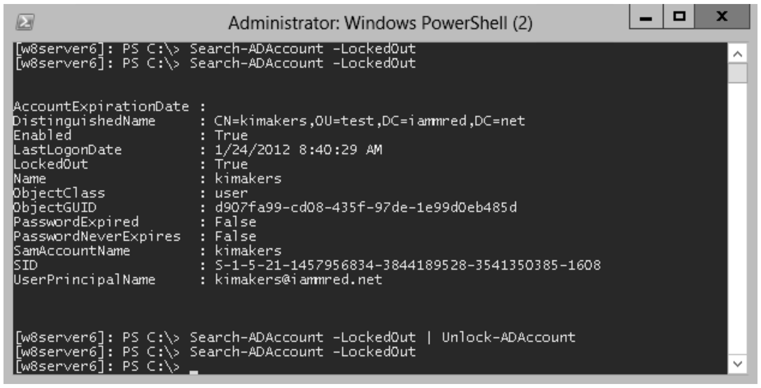
NOTE **People are often worried about Windows PowerShell from a security perspective. Windows PowerShell is only an application, and therefore a user cannot do anything that they do not have the rights or permission to accomplish. This is a case in point.**

If your user account does not have admin rights, you need to start Windows PowerShell with an account that has the ability to unlock a user account. To do this, you right-click the Windows PowerShell icon while holding down the Shift key; this allows you to select Run As Different User from the quick action menu.

When you start Windows PowerShell back up with an account that has rights to unlock users, the Active Directory module needs to load once again. You then check to ensure that you can still locate the locked out user accounts. After you have proven you can do that, you pipeline the results of the `Search-ADAccount` cmdlet to the `Unlock-ADAccount` cmdlet. A quick check ensures that you have unlocked all the locked out accounts. The series of commands is shown here:

```
Search-ADAccount –LockedOut
Search-ADAccount -LockedOut | Unlock-ADAccount
Search-ADAccount –LockedOut
```

The commands and associated output are shown in Figure 3-5.



```
[w8server6]: PS C:\> Search-ADAccount -LockedOut
[w8server6]: PS C:\> Search-ADAccount -LockedOut


AccountExpirationDate :
DistinguishedName     : CN=kimakers,OU=test,DC=iammred,DC=net
Enabled               : True
LastLogonDate         : 1/24/2012 8:40:29 AM
LockedOut             : True
Name                  : kimakers
ObjectClass           : user
ObjectGUID            : d907fa99-cd08-435f-97de-1e99d0eb485d
PasswordExpired       : False
PasswordNeverExpires  : False
SamAccountName        : kimakers
SID                   : S-1-5-21-1457956834-3844189528-3541350385-1608
UserPrincipalName     : kimakers@iammred.net


[w8server6]: PS C:\> Search-ADAccount -LockedOut | Unlock-ADAccount
[w8server6]: PS C:\> Search-ADAccount -LockedOut
[w8server6]: PS C:\> _
```

FIGURE 3-5 Using the Active Directory module to find and to unlock user accounts.

NOTE **Keep in mind that the command** `Search-ADAccount -LockedOut | Unlock-ADAccount` **will unlock every account that you have permission to unlock. In most cases, you will want to investigate prior to unlocking all locked out accounts. If you do not want to unlock all locked out accounts, use the *confirm* switch to be prompted prior to unlocking an account.**

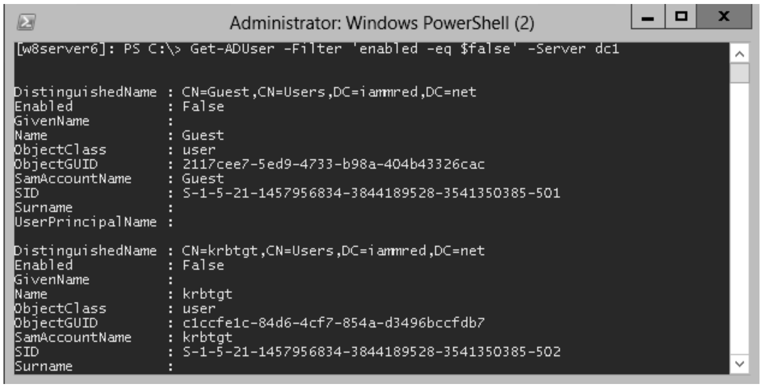
If you do not want to unlock all users, you use the *confirm* parameter from the `Unlock-ADAccount` cmdlet. For example, you first check to see what users are locked out by using the `Search-ADAccount` cmdlet—but you do not want to see everything, only their name. Next, you pipeline the locked out users to the `Unlock-ADAccount` cmdlet with the *confirm* parameter. You are then prompted for each of the three locked out users; choose to unlock the first and third users, but not the second user. You then use the `Search-ADAccount` cmdlet one last time to ensure that the second user is still locked out.

Finding disabled users

Luckily, by using Windows PowerShell and the Microsoft Active Directory cmdlets, it takes a single line of code to retrieve the disabled users from your domain. The command is shown here. (Keep in mind that running this command automatically imports the Active Directory module into the current Windows PowerShell host.)

```
Get-ADUser -Filter 'enabled -eq $false' -Server dc3
```

Not only is the command a single line of code, but it is also a single line of readable code. You get users from AD DS; you use a filter that looks for the enabled property set to false. You also specify that you want to query a server named dc3 (the name of one of the domain controllers on my network). The command and the associated output appear in Figure 3-6.



```
[w8server6]: PS C:\> Get-ADUser -Filter 'enabled -eq $false' -Server dc1


DistinguishedName : CN=Guest,CN=Users,DC=iammred,DC=net
Enabled           : False
GivenName         :
Name              : Guest
ObjectClass       : user
ObjectGUID        : 2117cee7-5ed9-4733-b98a-404b43326cac
SamAccountName    : Guest
SID               : S-1-5-21-1457956834-3844189528-3541350385-501
Surname           :
UserPrincipalName :

DistinguishedName : CN=krbtgt,CN=Users,DC=iammred,DC=net
Enabled           : False
GivenName         :
Name              : krbtgt
ObjectClass       : user
ObjectGUID        : c1ccfe1c-84d6-4cf7-854a-d3496bccfdb7
SamAccountName    : krbtgt
SID               : S-1-5-21-1457956834-3844189528-3541350385-502
Surname           :
```

FIGURE 3-6 Finding disabled user accounts.

If you want to work with a specific user, you can use the *identity* parameter. The *identity* parameter accepts several things: distinguishedname, sid, guid, or SamAccountName. Probably the easiest one to use is the SamAccountName. This command and associated output are shown here:

```
PS C:\Users\ed.IAMMRED>    Get-ADUser -Server dc3 -Identity teresa
DistinguishedName : CN=Teresa Wilson,OU=Charlotte,DC=iammred,DC=net
Enabled           : True
GivenName         : Teresa
Name              : Teresa Wilson
ObjectClass       : user
ObjectGUID        : 75f12010-b952-4d3-9b22-3ada7d26eed8
SamAccountName    : Teresa
SID               : S-1-5-21-1457956834-3844189528-3541350385-1104
Surname           : Wilson
UserPrincipalName : Teresa@iammred.net
```

To use the DistinguishedName value for the *identity* parameter, you need to supply it inside a pair of quotation marks—either single or double. This command and associated output are shown here:

```
PS C:\Users\ed.IAMMRED>    Get-ADUser -Server dc3 -Identity 'CN=Teresa Wilson,OU
=Charlotte,DC=iammred,DC=net'
DistinguishedName : CN=Teresa Wilson,OU=Charlotte,DC=iammred,DC=net
Enabled           : True
GivenName         : Teresa
Name              : Teresa Wilson
ObjectClass       : user
ObjectGUID        : 75f12010-b952-4d3-9b22-3ada7d26eed8
SamAccountName    : Teresa
SID               : S-1-5-21-1457956834-3844189528-3541350385-1104
Surname           : Wilson
UserPrincipalName : Teresa@iammred.net
```

It is not necessary to use quotation marks when using the SID for the value of the *identity* parameter. This command and associated output are shown here:

```
PS C:\Users\ed.IAMMRED>    Get-ADUser -Server dc3 -Identity S-1-5-21-1457956834-
3844189528-3541350385-1104


DistinguishedName : CN=Teresa Wilson,OU=Charlotte,DC=iammred,DC=net
Enabled           : True
GivenName         : Teresa
Name              : Teresa Wilson
ObjectClass       : user
ObjectGUID        : 75f12010-b952-4d3-9b22-3ada7d26eed8
SamAccountName    : Teresa
SID               : S-1-5-21-1457956834-3844189528-3541350385-1104
Surname           : Wilson
UserPrincipalName : Teresa@iammred.net
```

Again, you can also use *ObjectGUID* for the *identity* parameter value. It does not require quotation marks either. This command and associated output are shown here:

```
PS C:\Users\ed.IAMMRED>    Get-ADUser -Server dc3 -Identity 75f12010-b952-4d3-9
b22-3ada7d26eed8
DistinguishedName : CN=Teresa Wilson,OU=Charlotte,DC=iammred,DC=net
Enabled           : True
GivenName         : Teresa
Name              : Teresa Wilson
ObjectClass       : user
ObjectGUID        : 75f12010-b952-4d3-9b22-3ada7d26eed8
SamAccountName    : Teresa
SID               : S-1-5-21-1457956834-3844189528-3541350385-1104
Surname           : Wilson
UserPrincipalName : Teresa@iammred.net
```

Finding unused user accounts

To obtain a listing of all the users in Active Directory, supply a wildcard to the *filter* parameter of the `Get-ADUser` cmdlet. This technique is shown here:

```
Get-ADUser -Filter *
```

If you want to change the base of the search operations, use the *searchbase* parameter. The *searchbase* parameter accepts an LDAP style of naming. The following command changes the search base to the TestOU:

```
Get-ADUser -Filter * -SearchBase "ou=TestOU,dc=nwtraders,dc=com"
```

When using the `Get-ADUser` cmdlet, only a certain subset of user properties are displayed (10 properties to be exact). These properties will be displayed when you pipeline the results to `Format-List` and use a wildcard and the *force* parameter, as shown here:

```
PS C:\> Get-ADUser -Identity bob | format-list -Property * -Force


DistinguishedName : CN=bob,OU=TestOU,DC=NWTraders,DC=Com
Enabled           : True
GivenName         : bob
Name              : bob
ObjectClass       : user
ObjectGUID        : 5cae3acf-f194-4e07-a466-789f9ad5c84a
SamAccountName    : bob
SID               : S-1-5-21-3746122405-834892460-3960030898-3601
Surname           :
UserPrincipalName : bob@NWTraders.Com
PropertyNames     : {DistinguishedName, Enabled, GivenName, Name...}
PropertyCount     : 10



PS C:\>
```

Anyone who knows very much about Active Directory Domain Services (AD DS) knows that there are certainly more than 10 properties associated with a user object. If you try to display a property that is not returned by the `Get-ADUser` cmdlet, such as the *whenCreated* property, an error is not returned—the value of the property is not returned. This is shown here:

```
PS C:\> Get-ADUser -Identity bob | Format-List -Property name, whenCreated


name        : bob
whencreated :
```

The *whenCreated* property for the user object has a value—it just is not displayed. However, suppose you were looking for users who had never logged on to the system? Suppose you used a query such as the one seen here, and you were going to base a delete operation on the results? The results could be disastrous.

```
PS C:\> Get-ADUser -Filter * | Format-Table -Property name, LastLogonDate

name                                       LastLogonDate
----                                       -------------
Administrator
Guest
krbtgt
testuser2
ed
SystemMailbox{1f05a927-a261-4eb4-8360-8...
SystemMailbox{e0dc1c29-89c3-4034-b678-e...
FederatedEmail.4c1f4d8b-8179-4148-93bf-...
Test
TestChild
<results truncated>
```

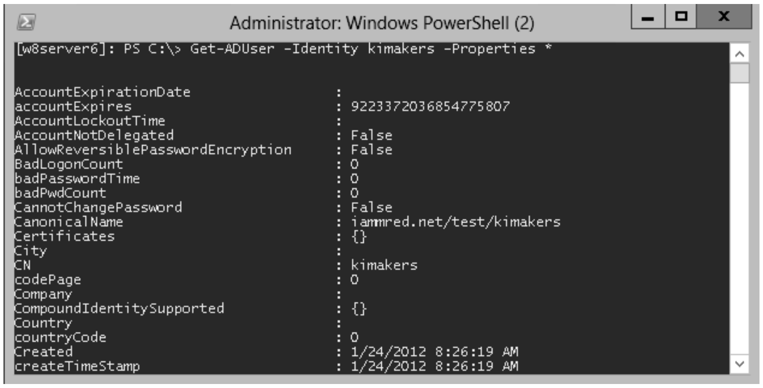
To retrieve a property that is not a member of the default 10 properties, you must select it by using the *property* parameter. The reason that `Get-ADUser` does not automatically return all properties and their associated values is because of performance reasons on large networks—there is no reason to return a large dataset when a small dataset will perfectly suffice. To display the *name* and the *whenCreated* date for the user named *bob*, the following command can be used:

```
PS C:\> Get-ADUser -Identity bob -Properties whencreated | Format-List -Property name,
whencreated


name        : bob
whencreated : 6/11/2010 8:19:52 AM



PS C:\>
```

To retrieve all of the properties associated with a user object, use the wildcard "*" for the properties parameter value. You would use a command similar to the one shown here:

```
Get-ADUser -Identity kimakers -Properties *
```

Both the command and the results associated with the command to return all user properties are shown in Figure 3-7.



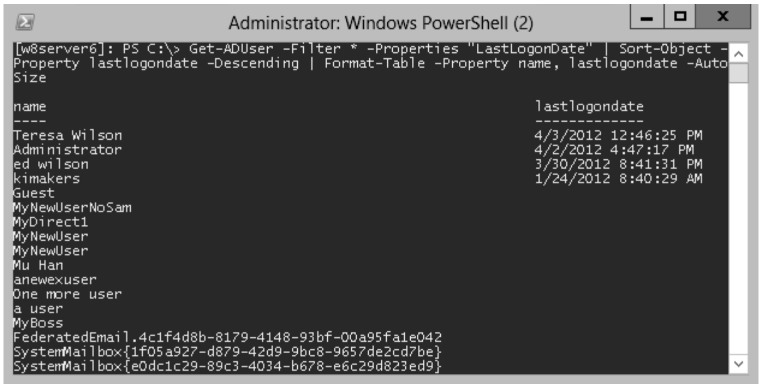
```
[w8server6]: PS C:\> Get-ADUser -Identity kimakers -Properties *


AccountExpirationDate                :
accountExpires                       : 9223372036854775807
AccountLockoutTime                   :
AccountNotDelegated                  : False
AllowReversiblePasswordEncryption    : False
BadLogonCount                        : 0
badPasswordTime                      : 0
badPwdCount                          : 0
CannotChangePassword                 : False
CanonicalName                        : iammred.net/test/kimakers
Certificates                         : {}
City                                 :
CN                                   : kimakers
codePage                             : 0
Company                              :
CompoundIdentitySupported            : {}
Country                              :
countryCode                          : 0
Created                              : 1/24/2012 8:26:19 AM
createTimeStamp                      : 1/24/2012 8:26:19 AM
```

FIGURE 3-7 Using the Get-ADUser cmdlet to display all user properties.

To produce a listing of all the users and their last logon date, you can use a command similar to the one shown here. This is a single command that might wrap the line, depending on your screen resolution.

```
Get-ADUser -Filter * -Properties "LastLogonDate" |
sort-object -property lastlogondate -descending |
Format-Table -property name, lastlogondate -AutoSize
```

The output produces a nice table. Both the command and the output associated with the command to obtain the time a user last logged on are shown in Figure 3-8.



```
[w8server6]: PS C:\> Get-ADUser -Filter * -Properties "LastLogonDate" | Sort-Object -
Property lastlogondate -Descending | Format-Table -Property name, lastlogondate -Auto
Size

name                                                           lastlogondate
----                                                           -------------
Teresa Wilson                                                  4/3/2012 12:46:25 PM
Administrator                                                  4/2/2012 4:47:17 PM
ed wilson                                                      3/30/2012 8:41:31 PM
kimakers                                                       1/24/2012 8:40:29 AM
Guest
MyNewUserNoSam
MyDirect1
MyNewUser
MyNewUser
Mu Han
anewexuser
One more user
a user
MyBoss
FederatedEmail.4c1f4d8b-8179-4148-93bf-00a95fa1e042
SystemMailbox{1f05a927-d879-42d9-9bc8-9657de2cd7be}
SystemMailbox{e0dc1c29-89c3-4034-b678-e6c29d823ed9}
```

FIGURE 3-8 Using the Get-ADUser cmdlet to identify the last logon times for users.

NOTES FROM THE FIELD

Jeff Wouters
Microsoft PowerShell MVP

"Write tools, not scripts" is one of my favorite phrases from the Windows PowerShell community. When I had just started to write some Windows PowerShell code, I was (and still am!) crazy about one-liners.

The ease with which the pipeline allows you to connect commands to each other and make them work together were unheard of in the VBS world.

But then I got a customer who wanted me to leave some code with them when I left. So I did, and only one week later, I got a call from that customer, in panic, saying that my script had deleted half their Active Directory!

I asked them to send me the code of the script. After only a few seconds, I noticed that this wasn't my code. There was a whole lot more in there that didn't come from me. So I connected to my home environment and looked in the backup I had made of all the scripts and documentation I had left with the customer, and yes, the script I had left them had a lot less code in there. So someone had changed my script!

Luckily, this customer had the Active Directory Recycle Bin enabled, so restoring the objects in Active Directory wasn't that hard. But for me, this was a wake-up call. Sign your scripts! Or at least make sure that you can verify the integrity of your scripts.

I also found that the person who changed my script, and basically was the cause of the problem, was a member of the service desk at that company. This is where "write tools, not scripts" comes into play.

So I rewrote my script, added a GUI, and signed it. This way, the help desk would have a nice clickable interface, and the script itself would be safe from malicious editors causing all kinds of issues. Because there were a whole bunch of scripts, I've created a module for them called "<CompanyName>Administration." To finish things off, I've introduced them to the concept of a centralized store for their modules.

For me, this was a learning curve, and these days I prefer a six steps approach:

1. Log everything; what it does and who executes it.
2. Support the common parameters, such as –*Whatif* and -*Confirm*.
3. Create an interface for the appropriate user—a command line for people who understand PowerShell and a GUI for those who don't.
4. Sign your script!

5. **Group scripts into modules.**
6. **Use a centralized module repository, preferably with read-only rights for everyone who is not responsible for the modules.**

These steps will make your life a whole lot easier when people start messing with your scripts.

Additional resources

- The TechNet Script Center at *http://www.microsoft.com/technet/scriptcenter* contains numerous script examples.
- All scripts from this chapter are available via the TechNet Script Center Script Repository at *http://gallery.technet.microsoft.com/scriptcenter/PowerShell-40-Best-d9e16039*.

Index

Symbols

* (asterisk), as wildcard character, 24
` (backtick) character, for line continuation, 305
: (colon), confirm parameter for $false value, 62
{ } (curly brackets)
  for ForEach-Object cmdlet, 191
  for function code block, 117
  for opening and closing script block, 242
  pairing comment with closing, 282, 305
$_ automatic variable, 255, 435
$? automatic variable, for testing errors, 188
$ (dollar sign), variable name and, 341–342
$_ variable, for current object, 220
! (not) operator, 188, 346
( ) (parentheses), in functions, 116
; (semicolon), to separate commands on single line, 10
<# and #> for comment tag, 287–289
! SET keyword, to preface variable assignments, 572

A

Abort, Retry, Ignore dialog box, 132
Abs method of System.Math class, 180
Abstract Syntax Tree (AST), 480
abstract WMI classes, 32
access denied message, 83
Access Is Denied error message, 561
Access property of Win32_LogicalDisk class, 249
AccountDomainSid method of SecurityIdentifier class, 87
account lockout policy, 57
Acos method of System.Math class, 180
Active Directory
  cmdlets, 13
  documenting, 56–59
  exporting portions for script testing, 471
  query, 208–217
    [ADSISearcher] for, 209
    cmdlets for, 213–214
    command line for, 214–217
  renaming sites, 59–60
Active Directory Domain Services, password storage in, 366–367
Active Directory Management Gateway Service (AD-MGS), 45
Active Directory module
  basics, 45–48
  Get-Command for listing cmdlets exported by, 51–52
  installing, 47
  user management, 60–63
  using, 48–72
  verifying presence, 47
Active Directory Users And Computers, 512
activities in workflow, 648–652
  cmdlets as, 649
adaptability of scripts, 225–229
Add-Computer cmdlet, 616, 623
Add-Content cmdlet, 93
AddDays method, 445
Add-History cmdlet, 204, 650
Add-Member cmdlet, 544
AddOne.ps1 script, 388–389
AddOne2.ps1 script, 389
AddOne3.ps1 script, 390
AddOne5.ps1 script, 391
AddOne6.ps1 script, 392
AddOne function, 271
Add-Printer cmdlet, 616
Add-PrinterDriver cmdlet, 616
Add-PrinterPort cmdlet, 616
Add-PSSnapin cmdlet, 650

Add-RegistryValue function, 556, 568–570, 572
AddToRecent method of Shell.Application object, 186
AddTwoError.ps1 script, 583–584
Add-WindowsFeature cmdlet, 47
administrative rights, 408
  for creating profile, 144
  for module install, 319
  requiring, 414
  for unlocking user account, 65
administrator, launching Windows PowerShell as, 96
[ADSISearcher] type accelerator, 209, 216
Alert (`a), 93
aliases, 136
  avoiding in profile, 146
  creating, 112–116
    for Get-Help cmdlet, 25
  for data types, 253–254
  for functions, 118
  making read-only, 114–115
  naming convention for, 145
  for parameters, 354
  predefined, 112
  in profile, 134
Alias parameter attribute argument, 354
AllowEmptyCollection parameter attribute, 360
AllowEmptyString parameter attribute, 360
AllowNull parameter attribute, 360
AllSigned script execution policy, 140, 493, 499, 502
AllUsersAllHosts profile, 141, 143
AllUsersCurrentHost profile, 141-142
  creating, 144
APIs, testing, 455
AppActivate method of WshShell object, 129
appending to log file, 532, 536–539
Application log, 554–555
Application property of Shell.Application object, 187
applications
  lack of external support, 189–193
  scripting requirements, 106–108
Archive resource provider, 660, 665
$args automatic variable, 340–342
  indexing directly to, 343
  passing multiple parameters with, 120–122
arguments. *See also* parameters
  positional, 239
arrays
  – contains operator to examine contents, 404
  indexing, 39
  looping, 119
  pipelining to Get-WmiObject cmdlet, 344
$arycomputer variable, 219–220
ASCII file. *See also* text files
  .ddf file as, 190
Asin method of System.Math class, 180
–as operator, 87, 93
Association classes, 35–41
  exploring returned, 40
  finding, 31
asterisk (*), as wildcard character, 24
AST parser, 484–486
Atan2 method of System.Math class, 180
Atan method of System.Math class, 180
authentication, 152
AuthenticationType property of WindowsIdentity
    object, 84
authorized verb list, warning to check, 317
Autoexec.bat file, 141
automation interface, 75–82
automation of routine tasks, 73–74
Availability property of Win32_LogicalDisk class, 249

B

background job
  checking status, 640
  Windows PowerShell as, 82
backslash, escaping, 158
backspace (`b), 93
backtick (`) character, for line continuation, 305
backtick n, 93
backtick r, 93
backtick t (`t), 265
BackUpFiles.ps1 script, 278–280
backup of scripts, 547
backups of domain controllers, for script testing, 471
BadGetRandomObject.ps1 script, 154
BadParam.ps1 script, 348
BadScript.ps1 script, 573–581
basename property of System.Io.FileInfo class, 322
BasicFunctions.psm1 module, 331–333
batch file, 4
Begin block in filter, 274
–Begin parameter, 270
Bellée, Chris, 229, 483–484
BigMul method of System.Math class, 180

BinaryLength method of SecurityIdentifier class, 87
BIOS information
  retrieving from remote system, 630
  returning from local computer, 33
  saving to CSV file, 152
black box approach, 247
BlockSize property of Win32_LogicalDisk class, 249
boundary-checking functions, 357, 429–430
BoundParameters property of System.Management.Automation.InvocationInfo class, 178
Brasser, Jaap, 355
breakpoints, 586–587
  deleting, 601
  enabling and disabling, 599
  listing, 597–598
  responding to, 596–597
  setting, 587–595
    on command, 593–595
    on line number, 588–589
    on variable, 589–593
BrowseForFolder method of Shell.Application object, 186
Brundage, James, 145–146, 410–412, 439, 455–456, 458
build number for operating system version, 102
Burley, James Craig, 267–270
business logic, 564
  functions for encapsulating, 259–261
BusinessLogicDemo.ps1 script, 259, 260
buttons, values and meanings for pop-up dialog box, 131
Bypass script execution policy, 140, 493, 503

C

cabinet files, 183
  adding files to, 184
  closing, 184
  expanding, 186
calling functions, 238
Canastreiro, Luís, 101–102, 159
CanStartStopService method of Shell.Application object, 186
Caption property of Win32_LogicalDisk class, 249
Carter, Marc, 329
CascadeWindows method of Shell.Application object, 186
case-sensitive operator (-ceq), 114
case-sensitivity of –contains operator, 404
catalog file, 222
–ccontains operator, 404
cd command, 3
Cedeno, Enrique, 467–468, 471
Ceiling method of System.Math class, 181
central file share, using module from, 330–331
centralized module repository, 72
-ceq (case-sensitive operator), 114
Certificate Enrollment Wizard, 504–505
Certificate Manager utility, 504
certificates for script signature, 524
ChangeVue, 522
Check-AllowedValue function, 429
CheckEventLog.ps1 script, 306, 307
CheckForPdfAndCreateMarker.ps1 script, 296
Check-Number function, 356–357
CheckNumberRange.ps1 script, 356
checkpoints in workflow, 652–655
  adding, 653–655
  disabling, 655–656
  placement in workflow, 652–653
CheckPoint-Workflow activity, 648, 655–656
CheckProviderThenQuery.ps1 script, 417
child functions, variables from parent available to, 241
child Organizational Unit (OU)
  creating, 60
Chkdsk method of Win32_LogicalDisk class, 249
Christopher, Jim, 318–319
CIM cmdlets, WMI classes for exploring, 27–33
classes in WMI, 169–179
classname parameter, for Get-CimClass cmdlet, 27–29
__CLASS property of Win32_Logical Disk class, 250
$class variable, 425
Clear-EventLog cmdlet, 616, 623
Clear-History cmdlet, 650
Clear-Host function, 394
clearing
  $Error array, 256
  error stack, 435
clear method, 18
Clear-Variable cmdlet, 650
client workstations, determining service pack level, 28
closecab method, 184
cloud technologies, 136
CMD (command) shell, 3
cmdlets, 7, 483–484
  Active Directory, 13

for Active Directory query, 213–214
controlling execution of, 11
disallowed in workflow from core modules, 650
help for, 280–281
as InlineScript activity, 650–651
lack of support, 151–153
options for, 16–17
problem from number of, 152
redirecting output to text file, 59
setting breakpoint on, 593–595
suspending confirmation of, 12–13
as workflow activities, 649
CmdLineArgumentsTime.ps1 script, 301
code
reducing complexity, 449–450
reusing, 147, 226, 244–246
verbose style for, 247
signing, 504–505
snippets to create, 611–612
code block for function, 117
collaboration in scripting, 229–230
tools for, 73
colon (:), confirm parameter for $false value, 62
CombinationFormatGetIPDemo.ps1 script, 266
COM (Component Object Model) components
lack of support, 182–189
object creation, 184
Command add-on, 607
script pane and, 608
turning off, 609
in Windows PowerShell ISE, 606
CommandAST class, 487
command-line
Active Directory query from, 214–217
input from, 340–348
$args automatic variable for, 340–342
moving through, 60
path length on, 133
script writing vs., 195
command-line parameters, 349–350
creating, 183, 186
default values for, 398–399
validating, 356
command-line utilities, 7, 9–10
CommandOrigin property of System.Management.Automation.InvocationInfo class, 178
commands
confirming, 12
entering multiple, 10
exporting history, 203–205
locating, 607
overriding existing, 117–118
running in script pane, 609
setting breakpoint on, 593–595
in text file, 196
CommandType property of System.Management.Automation.ScriptInfo object, 178
comment-based help, 289–294, 477
comments, 256, 468
documentation from, 481–483
documenting nested structures with, 305–306
in functions, 117
internal version number in, 525–528
multiple-line, 277, 287–288
one-line, 277–287, 303–304
creating, 288–289
pairing with curly bracket, 282
rules for writing, 295–310
adding during development process, 296–297
avoiding useless information, 302
consistency in header information, 298–299
international audience and, 297–298
on document deficiencies, 300–301
prerequisite information, 299–300
reason for code, 303
updating for updated script, 295–296
on script changes, 527
standard set of keywords for, 306–307
on strange items in code, 307–308
Commercial Certificate Authority, 524
common classes, 169
Common Language Runtime (CLR), vs. .NET Framework, 101
–ComObject parameter, 182
CompareTo method of SecurityIdentifier class, 86
compatibility of scripts, version control and, 523
Complete-Transaction cmdlet, 650
complicated constructors, 153–154
Component help tag, 290
compressed file, unzipping, 665
Compressed property of Win32_LogicalDisk class, 249
computers
account creation, 60
ping to identify accessible, 402–403
ConfigManagerErrorCode property of Win32_LogicalDisk class, 249

ConfigManagerUserConfig property of Win32_LogicalDisk class, 249
Configuration keyword, 661–662
ConfigurationNamingContext property, 59
confirmation prompt, suppressing, 62
–confirm parameter, 11-12, 16
  colon (:) for $false value, 62
  for Unlock-ADAccount cmdlet, 66
connection strings, 372–373
Connect-PSSession cmdlet, 616
Connect-WSMan cmdlet, 616
consistency
  in comment header information, 298–299
  in Windows PowerShell, 74
Console configuration file, exporting, 15
constant alias, creating, 115
constants, 126
constructors
  complicated, 153–154
  and error, 154
  for .NET Framework classes, 179
consumers in WMI, 163
–contains operator, 88–89
  testing properties with, 406–408
  to examine array contents, 404
Continue command (c), 596, 600
ControlPanelItem method of Shell.Application object, 186
ConversionFunctions.ps1 script, 147–148, 244–245
ConvertFrom-Cab function, 186, 188
ConvertFromDateTime script method, 250
ConvertFrom-SecureString cmdlet, 368–371
converting System.IO.FileInfo object FullName property to string, 191
ConvertToDateTime script method, 250
ConvertToFahrenheit_Include.ps1 script, 148, 300
ConvertToFahrenheit.ps1 script, 147
ConvertTo-Html cmdlet, 387, 535
ConvertToMeters.ps1 script, 244
ConvertTo-SecureString cmdlet, 63, 370–371
ConvertUseFunctions.ps1 script, 149–150
Copy-Item cmdlet, 198–199, 323
Copy-Module function, 321
Copy-Modules.ps1 script, 320–324, 330
core classes, 169
core cmdlets, 7
Cosh method of System.Math class, 181
Cos method of System.Math class, 181
Costantini, Peter, 309
CreateCab.ps1 script, 185
CreateCab2.ps1 script, 192
CreateFileNameFromDate.ps1 script, 278
CreateFilePath function, 227
CreateRegistryKey.ps1 script, 556–557, 568–569, 571
CreateScriptingRegistryKey.ps1 script, 225
CreateSelection function, 226
CreateShortcut method of WshShell object, 129
CreateWordDoc function, 226
CreationClassName property of Win32_LogicalDisk class, 249
credential object, obtaining, 160
credentials
  cmdlets to specify, 624–625
  for remote PowerShell session, 55
  importing and exporting, 370–372
  passing, 158
  running cmdlets with different, 626
$credential variable, 370
CSVDE, 471–472, 473
CSV files, 535–536
  piping command variables to, 50
  saving BIOS information to, 152
curly brackets ({ })
  for ForEach-Object cmdlet, 191
  for function code block, 117
  for opening and closing script block, 242
  pairing comment with closing, 282, 305
CurrentDirectory method of WshShell object, 130
current state, checkpoint as snapshot, 652
CurrentUserAllHosts profile, 141, 144
CurrentUserCurrentHost profile, 141, 144
  creating, 142–143
current user, detecting, 84–95
Current User profiles, 142
custom event log, 555

D

data
  handling large amounts, 166–168
  presentation functions separated from gathering, 264–265
  testing scripts against known, 471
database
  for script logging, 542

of scripts, 229
testing script connecting to, 467–468
data types
aliases for, 253–254
incorrect, 423–429
DateTime object, 435, 445
.ddf file
as ASCII file, 190
creating, 189
debugging. *See also* errors
basics, 559–567
breakpoints for, 586–587
setting, 587–595
PowerShell cmdlets for, 585–603
quitting session, 588
recommendations for, 585
script-level tracing for, 568–572
Set-PSDebug cmdlet for, 567–585
stepping through script, 572–581
syntax errors, 560
debug mode, exiting, 600
debug parameter, 16, 183, 460–462, 562–564
$DebugPreference variable, 177, 184, 562
Debug-Process cmdlet, 650
DebugRemoteWMISession.ps1 script, 562–563
$debug variable, 188
checking presence, 192
default script execution policy, 492
default value for parameter, 247, 398–399
default WMI namespace, 27
Definition property System.Management.Automation.
ScriptInfo object, 178
DefragAnalysis method, 219, 220
DefragAnalysisReport.ps1 script, 219, 221
defragmentation report, redirection to produce, 220
Dekens, Luc, 565–567
DeleteScriptingRegistryKey.ps1 script, 225
deleting
breakpoints, 601
read-only cmdlet, 115
remnants of completed jobs, 636
risk of deleting wrong script version, 527
snippets, 613
DemoAddOneFilter.ps1 script, 272
DemoAddOneFunction.ps1 script, 272–273
DemoAddOneR2Function.ps1 script, 273
DemoConsoleBeep.ps1 script, 302
DemoConsoleBeep2.ps1 script, 303
Demo-MultilineComment.ps1 script, 287–288
DemoTrapSystemException.ps1 script, 254, 255
DemoUserConfig.ps1 configuration script, 666-667
dependencies
checking modules for, 326–329
in DSC, 665
between scripts, 523
deploying
help files, 26
script execution policies, 495–500
Windows PowerShell, 7
deprecated qualifier, 31
__DERIVATION property of Win32_Logical Disk
class, 250
Description help tag, 290
–description parameter, 128
of New-Alias cmdlet, 114
for PowerShell drives, 134
Description property of Win32_LogicalDisk class, 249
deserialized object
methods, 637
storing in variable, 639
Desired State Configuration (DSC), 76, 659–671
configuration, 663–664, 666–671
controlling configuration drift, 671–673
dependencies, 665
group provider for group creation, 669–671
process, 660–662
resource providers and properties, 660
user provider for user creation, 669
desktop settings, Win32_Desktop WMI class for, 173
development, 236
adding comments furing, 296–297
DeviceID property of Win32_LogicalDisk class, 249
diagnostic scripts, 516
dialog box, 130–132
dir command, 3
directories. *See* folders
DirectoryEntry object, 216
disabled user accounts, 63
finding, 66–68
Disable-PSBreakpoint cmdlet, 586, 599, 601, 650
Disconnect-PSSession cmdlet, 616
Disconnect-WSMan cmdlet, 616
DisplayName attribute, changing, 60
DisplayProcessor.ps1 script, 517–520
DistinguishedName value, for identity parameter, 67
divide-by-zero error, 597

DivRem method of System.Math class, 181
documentability of scripts, 223–225
documentation
  Active Directory, 56–59
  comments for, 256, 481–483
    listing deficiencies, 300–301
  from help, 475–480
  PSParser class Tokenize method for, 484–486
  script testing process, 442–444
document files
  finding, 260
  Standard Operating Procedure (SOP), 76
Documents folder, Join-Path cmdlet for location to, 127–128
dollar sign ($), variable name and, 341–342
domain controllers, 58
  backup for script testing, 471
domain FSMO roles, command for obtaining, 55
domain password policy, 57
dotnettypes.format.ps1xml file, 378
DotSourceScripts.ps1 script, 263
dot-sourcing scripts, 244–245
double-clicking, to run script, 499
download, managing for help files, 20
$driveData variable, 248, 251
DriveType property of Win32_LogicalDisk class, 249
DSQuery.exe, 216–217
dynamic classes, 169
dynamic qualifier, 32
__DYNASTY property of Win32_Logical Disk class, 250

E

EjectPC method of Shell.Application object, 186
elevated permissions, 83
email
  output to, 387–388
  sending logging information by, 551
Enable-ADAccount cmdlet, 64
enabled property, for user account, 64
Enable-PSBreakpoint cmdlet, 586, 599, 601, 650
Enable-PSRemoting function, 627–628
Encoding parameter, 540
Encrypting File System (EFS), 363
End block in block, 274
End parameter, 221, 270
Enterprise Certificate Authority, 524
Enter-PSSession cmdlet, 56, 616, 629, 631, 650
Enum class, GetValues static method, 98
Environment method of WshShell object, 130
Environment .NET Framework class, GetFolderPath static method, 544
Environment resource provider, 660
environment variables, in profile, 135
E property of System.Math class, 182
−eq operator, 114
Equals method
  of SecurityIdentifier class, 86
  of System.Math class, 181
-ErrorAction parameter, 16, 18
$errorActionPreference variable, 424–425, 428, 434, 436, 532, 537
$Error array, clearing, 256
$error automatic variable, 435
ErrorCleared property of Win32_LogicalDisk class, 249
ErrorDescription property of Win32_LogicalDisk class, 249
error handling
  basics, 397
  – contains operator to examine array contents, 404
  for detecting operating system version, 220
  incorrect data types, 423–429
  learning, 427–429
  limiting choices, 400–408
    PromptForChoice for, 401–402
  missing parameters, 398–400
  missing rights, 408–415
  missing WMI providers, 415–423
  out of bounds, 429–431
  Try/Catch/Finally structure, 409–410
error message
  for constant alias, 115–116
  from dollar sign in variable, 342
  from failed efforts to create profile, 144
  from missing parentheses, 127
  ParameterBindingException, 343
  suppressing display, 255
  type mismatch, 343
  from updating help, 18
  VariableNotFound, 342
ErrorMethodology property of Win32_LogicalDisk class, 249
$error object, 347
ErrorRecord class, 255

errors
- constructors and, 154
- divide-by-zero, 597
- from handling large amounts of data, 166–168
- Invalid parameter, 122
- from Invoke-Command, 197
- from job script block, 638
- logic, 564–565
- looking for, 438–439
- from missing closing bracket, 282
- from multiple instances for inputobject parameter, 38–39
- from not supplying distinguishedname, 63
- from null value as argument, 345–346
- from PowerShell workflow, 647
- runtime, 560–564
- syntax, 560
- testing, $? automatic variable for, 188
- from Throw statement, 346
- from type constraint deviation, 254–255
- from version incompatibility, 155
- WinRM and, 628–629
- writing to log file, 468
- from wrong placement of Param, 348

error stack, clearing, 435
-ErrorVariable parameter, 16
escaping backslash, 158
$etime variable, 436
evaluating scripts, 479–480
event log, 509–510
- custom, 555
- logging to, 552–555
- monitoring, 552

Example help tag, 290
-examples argument, 24
examples, in cmdlet documentation, 281
Exec method of WshShell object, 130
exiting debug mode, 600
Exit-PSSession cmdlet, 650
Expand-Cab function, 223
ExpandCab.ps1 script, 189
ExpandEnvironmentStrings method of WshShell object, 130
expanding cabinet file, 186
ExpectingInput property of System.Management.Automation.InvocationInfo class, 178
Explore method of Shell.Application object, 186
ExplorerPolicy method of Shell.Application object, 186
Exp method of System.Math class, 181
Export-Alias cmdlet, 650
ExportBiosToCsv.ps1 script, 152
Export-CliXml cmdlet, 370, 375
Export-Console cmdlet, 15, 650
Export-CSV cmdlet, 387, 535-536
Export-Excel function, 352
exporting
- command history, 203–205
- credentials, 370–372
- to XML, 386

Extended Type System (ETS), 135
external dependency, 326
ExternalHelp help tag, 290

F

fan-out commands, 205–208
Farr, Ian, 524–525, 541, 546–547
Favorites folder, path to, 128
feedback, on help documentation, 281
file hashes, counting, 8
$filepath property, 221
File resource provider, 660
FileRun method of Shell.Application object, 186
files
- output to, 382–383
- searching for string pattern, 243–244
- splitting output to screen and, 383–387

filesystemobject, 152
FileSystem property of Win32_LogicalDisk class, 249
FilterHasMessage.ps1 script, 274
filter parameter of Get-CimInstance cmdlet, 33, 39
filters, 270–275
- to remove folders from cabinet file collection, 191
- for WMI classes by qualifier, 30–33

FilterToday.ps1 script, 274
FindAll method of DirectorySearcher object, 209
FindComputer method of Shell.Application object, 186
FindDisabledUserAccounts.ps1 script, 295–296
FindFiles method of Shell.Application object, 186
finding
- Association classes, 31
- document files, 260
- FSMO role holders, 50–56
- unused user accounts, 68–72
- user accounts, 64–66

FindLargeDocs.ps1 script, 260–261
FindPrinter method of Shell.Application object, 186
Finke, Douglas, 243
Floor method of System.Math class, 181
folders
  changing, 11
  displaying listing, 3
  for scripts, 515
  for modules, 311–312, 319–322
  temporary, 285–286
  for text file log, 542
Foldershare, 135
forceDiscover switch, for Get-ADDomainController cmdlet, 52
force parameter, 17
  for Format-List cmdlet, 68
  to suppress prompts, 64
ForEach-Object cmdlet, 87, 113, 344, 447, 449
ForEach -Parallel workflow activity, 646, 648
Forest FSMO roles, command for obtaining, 55
Format-IPOutput function, 266
Format-List cmdlet, force parameter, 68
Format-NonIPOutput function, 266
format.ps1xml files, 377–378
Format-Table cmdlet
  -autosize parameter, 208
  for Get-PSBreakpoint results, 598
  piplining fan-out command results to, 206–207
Format-Table cmdlet (ft alias), 34
Form feed (`f), 93
for statement, 119, 195
ForwardHelpCategory help tag, 290
ForwardHelpTargetName help tag, 290
FreeSpace property of Win32_LogicalDisk class, 249
FSMO role holders
  finding, 50–56
FSOBiosToCsv.ps1 script, 152
fsutil, 4
FullName property of System.IO.FileInfo object
  converting to string, 191
Functionality help tag, 290
FunctionGetIPDemo.ps1 script, 264
Function keyword, 116, 189, 234, 237, 248
function library, 244
  creating, 147–148
functions, 95, 233–244
  accessing in other scripts, 147–150
  alias for, 118
  benefits of changing code to, 263
  business logic encapsulation, 259–261
  calling, 238
  comments for, 117
  creating, 116–120, 234
  for code reuse, 244–246
  for ease of modification, 261–270
  guidelines for writing, 246
  include file and, 148–150
  looping array, 119
  moving inline code into, 262
  naming convention, 145
    verb-noun, 116
  output from, 388–392
  overriding existing commands, 117–118
  passing multiple parameters, 120–126, 248–253, 257–258
    named, 122–126
    with $args, 120–122
  positional arguments, 239
  priority of, 117
  signature of, 260
  type constraints for parameters, 253–255

G

__GENUS property of Win32_Logical Disk class, 250
Get-ADDefaultDomainPasswordPolicy cmdlet, 57
Get-ADDomain cmdlet, 57
Get-ADDomainController cmdlet, 52–53, 58
Get-ADForest cmdlet, 56
GetAdminFunction.ps1 script, 98
Get-ADObject cmdlet, 59
Get-ADOrganizationalUnit cmdlet, 63, 216
Get-ADRootDSE cmdlet, 59
Get-ADUser cmdlet, 63-64, 66
  wildcard for filter parameter, 68
Get-Alias cmdlet, 25, 112-113, 124–125, 650
Get-AllowedComputerAndProperty.ps1 script, 408
Get-AllowedComputer function, 407–408
Get-AllowedComputer.ps1 script, 404–406
Get-ASTScriptProfile.ps1 script, 480
Get-AuthenticodeSignature cmdlet, 504
GetBinaryForm method of SecurityIdentifier class, 86
Get-BiosArgsCheck2.ps1 script, 346
Get-BiosArgsTrap1.ps1 script, 347
Get-BiosArray1.ps1 script, 343

Get-BiosArray2.ps1 script, 344
Get-Bios function, 469
Get-BiosInformationDefaultParam.ps1 script, 399
Get-BiosInformation.ps1 script, 398
Get-BiosMandatoryParameter.ps1 script, 351
Get-BiosMandatoryParameterWithAlias.ps1 script, 354
Get-BiosParam.ps1 script, 348–350
Get-Bios.ps1 script, 341, 397, 409, 412
GetBiosTryCatchFinally.ps1 script, 347–348
Get-Change function, 452
Get-ChildItem cmdlet, 184, 260, 377, 434, 445
Get-Choice function, 401
Get-ChoiceFunction.ps1 script, 401–402
Get-CimAssociatedInstance cmdlet, 35, 616
  inputobject, 38
  resultclassname parameter, 38
  viewing objects returned, 40
Get-CimClass cmdlet, 27, 616
  classname parameter of, 27–29
Get-CimInstance cmdlet (gcim), 33, 616, 646
  filter parameter, 39
  property parameter, 33
Get-CimSession cmdlet, 617
GetCmdletsWithMoreThanTwoAliases.ps1 script, 112
Get-Command cmdlet, 214, 234, 649
  cmdlets exported by Active Directory module, 51–52
  module parameter, 48, 335
Get-CommandLineOptions function, 441
GetCommentsFromScript.ps1 script, 481–483
Get-Comments function, 482
Get-CompInfo cmdlet, –PSPersist parameter, 653
Get-CompInfoWorkflowCheckPointWorkflow.ps1 script, 653, 655–656
Get-CompInfoWorkflowPersist.ps1 cmdlet, 655–656
Get-ComputerInfo function, 333, 335
GetComputerInfoWorkFlow.ps1 script, 647–648
Get-ComputerWmiInformation.ps1 script, 440, 442–444
Get-Content cmdlet, 196, 198, 219, 382
Get-Counter cmdlet, 617
Get-CountryByIP function, 544
Get-CountryByIP.ps1 script, 543–546
Get-Credential cmdlet, 160, 370, 514
GetCurrent method, 84
GetCurrent method of Security.Principal.WindowsIdentity class, 90, 92, 96, 553
Get-Date cmdlet, 274, 445, 645
  for log time stamp, 469
Get-DirectoryListing function, 257, 258
Get-DirectoryListingToday.ps1 script, 258
Get-Discount function, 259
Get-DiskInformation function, 430
Get-DiskSpace.ps1 script, 251–252
Get-Doc function, 260
GetDrivesCheckAllowedValue.ps1 script, 430–431
GetDrivesValidRange.ps1 script, 431
Get-EnabledBreakpointsFunction.ps1 script, 598
Get-EventLog cmdlet, 555, 617, 623
Get-EventLogData.ps1 script, 651–658
Get-ExecutionPolicy cmdlet, 140, 612
Get-FileHash cmdlet, 8
Get-FileName function, 482
GetFolderPath static method from Environment .NET Framework class, 544
Get-FreeDiskSpace function, 248
Get-FreeDiskSpace.ps1 script, 248
GetGeoIP method, 544
GetHashCode method of SecurityIdentifier class, 87
Get-Help cmdlet, 17, 116, 292, 335–336, 615
  cmdlet discovery with, 5
  help for, 21–23
Get-History cmdlet, 203, 650
Get-HotFix cmdlet, 618, 624
GetInfoByZip method, 254
GetIPDemoSingleFunction.ps1 script, 262–263
Get-Ipinfo cmdlet, 382
Get-IPObjectDefaultEnabledFormatNonIPOutput.ps1 script, 265
Get-IPObjectDefaultEnabled.ps1 script, 264–265
Get-IPObject function, 266
Get-IseSnippet cmdlet, 613
Get-ItemProperty cmdlet, 100
Get-Job cmdlet, 635, 636, 639
  –Newest, 640
Get-Mailbox cmdlet, 387
Get-Member cmdlet, 35, 40, 180, 388
  for properties of Win32_Desktop WMI class, 174–175
Get-MemberOf.ps1 script, 90–91
Get-MicrosoftUpdates.ps1 script, 305
Get-ModifiedFiles.ps1 script, 445
Get-ModifiedFilesUsePipeline2.ps1 script, 449–450
Get-Module cmdlet, 47, 50, 312, 316, 328
  –ListAvailable parameter, 313, 317, 334
Get-MoreHelp2.ps1 script, 119

Get-MoreHelp function, 116
Get-MoreHelpWithAlias.ps1 script, 118
Get-MyModule function, 327–329
GetNames method of System.Enum .NET Framework class, 98
Get-NetAdapter cmdlet, 640
Get-NetIPConfiguration cmdlet, 5
Get-OperatingSystemVersion function, 237, 320
Get-OperatingSystemVersion.ps1 script, 237–238
Get-OptimalSize function, 337
GetOsVersionFunction.ps1 script, 103–105
Get-OsVersion.ps1 script, 160
Get-PowerShellRequirements.ps1 script, 6
Get-PrintConfiguration cmdlet, 618
Get-Printer cmdlet, 618
Get-PrinterDriver cmdlet, 618
Get-PrinterPort cmdlet, 618
Get-PrinterProperty cmdlet, 618
Get-PrintJob cmdlet, 618
Get-Process cmdlet, 106, 151-152, 376–377, 618, 624, 637
GetProcessesDisplayTempFile.ps1 script, 308
Get-PSBreakpoint cmdlet, 586, 597–598, 650
Get-PSCallStack cmdlet (k), 586, 595-596, 650
Get-PSDrive cmdlet, 421
Get-PSSession cmdlet, 618, 631
Get-PSSnapin cmdlet, 650
Get-PsVersionNet.ps1 script, 78, 78–79
Get-PsVersionRegistry.ps1 script, 75
Get-PsVersionRegRead.ps1 script, 77
Get-PSVersionRemoting.ps1 script, 81
Get-PsVersionWmi.ps1 script, 78
Get-PSVersionWorkflow.ps1 script, 80
Get-Random cmdlet, 154
GetRandomObject.ps1 script, 153–154
GetRunningProcess.ps1 script, 107–108
GetRunningService.ps1 script, 106–107
Get-ScriptHelp.ps1 script, 475–477, 478
Get-ScriptVersion.ps1 script, 525–527
Get-Service cmdlet, 106-107, 151-152, 205–206, 618, 624
GetServicesInSvchost.ps1 script, 303–304
GetSetieStartPage.ps1 script, 283–285
GetSetting method of Shell.Application object, 186
GetStringValue method, 78
GetSystemInformation method of Shell.Application object, 186
GetSystemVersion method of System.Runtime.InteropServices.RunTimeEnvironment .NET Framework class, 100
Get-TempFile function, 452
GetTempFileName method, 308, 434, 548–549
GetTempPath method, 285
Get-TextStatisticsCallChildFunction-DoesNOTWork-MissingClosingBracket.ps1 script, 242–243
Get-TextStatisticsCallChildFunction.ps1 script, 241–242
Get-TextStatistics function, 238, 241
  $path variable in, 242
Get-TextStatistics.ps1 script, 239
Get-Transaction cmdlet, 650
Get-Type function, 146
GetType method, 87, 424
Get-ValidWmiClass function, 423-426
Get-ValidWmiClassFunction.ps1 script, 423–426
GetValue method (.NET), 78
GetValues method of Enum class, 98
Get-Variable cmdlet, 341, 650
get-verb pattern for cmdlets, 7
Get-Version.ps1 script, 160–161
get-VM.ps1 script, 414
Get-Volume function, 158
GetVolume.ps1 script, 158
GetVolumeWithCredentials.ps1 script, 158
Get-WebServiceProxy cmdlet, 543
Get-WindowsEdition.ps1 script, 527–528
Get-WindowsFeature cmdlet, 47, 656
Get-WinEvent cmdlet, 618
Get-WinFeatureServersWorkflow.ps1 script, 656
Get-WmiClass2.ps1 script, 122, 124
Get-WmiClass2WithAlias.ps1 script, 125–126
GetWmiClassesFunction1.ps1 script, 291–293
Get-WmiClass function, 121, 123
  alias for, 124
Get-WmiClass.ps1 script, 122
GetWmiData function, 226
Get-WmiInformation function, 426
Get-WmiObject cmdlet, 168–169, 251, 469, 618, 624
  for BIOS information, 348
    from remote system, 630
  for connection into WMI, 164
  pipelining array to, 344
  to query for __provider WMI class instances, 416, 418
  to query Win32_LogicalDisk WMI class, 248
  to query Win32_Volume WMI class, 220

to retrieve Win32_NetworkAdapterConfiguration WMI class instances, 261
to retrieve Win32_OperatingSystem WMI class instance, 237
to verify operating system, 56
for WMI class listing, 121, 170
Get-WmiProvider function, 417–418, 422
Get-WmiProviderFunction.ps1 script, 422–423
Get-WMIProviders.ps1 script, 168–169
Get-WSManInstance cmdlet, 618
gip, 5
global security group, creating, 61
global variable, 390–392
namespace in, 392–393
Goude, Niklas, 513–514, 646
graphical applications, testing, 458–459
Group-Object cmdlet, 234–236
Group Policy
assigning scripts from within, 513
script execution policy deployment with, 499–500
script execution policy modified with, 495
script execution policy setting with, 412–413
Set the default source path for Update-Help, 20
Group Policy Object (GPO)
for logon/logoff script, 507–508
for PowerShell deployment, 7
Group Policy Preferences, scheduled task for Update-Help, 20–21
Group Policy templates (ADM files), 502
Group resource provider, 660, 669–671
groups
adding user, 62
determining match, 93
removing user from, 62
Groups property of WindowsIdentity object, 84, 85–86
GUIs, 105, 337
Gusev, Vasily, 599–601

H

hash of password, storing in text file, 368–369
hash table of permissible values, 429
HasMessage filter, 274
header for script, 278, 477–478
HelloUserTimeworkflow.ps1 script, 645
HelloUserworkflow.ps1 script, 644
Helmick, Jason, 5–6
help. *See also* comments
for cmdlets, 17, 280–281
comment-based, 289–294
deploying with scheduled task, 26
discovering information in, 21–25
documentation from, 475–480
keeping up-to-date, 19–20
options for, 17
updating, 17–19
Help command (debugger), 596
Help desk scripts, 517–520
helpdesk staff, permissions and, 513–514
help function, 24
HelpMessage parameter attribute argument, 354
Help method of Shell.Application object, 186
$help variable, 291
here-string, 190
Hicks, Jeffery, 479–480
hierarchical namespace, 162
Hill, Keith, 314
hive, 77
HKEY_CURRENT_USER hive, 556
new registry key in, 224–225
HKEY_LOCAL_MACHINE registry key, 77
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft
\NET Framework Setup\NDP\v4\Full\Release key value, 100
\PowerShell\1\ShellIDs\Microsoft.PowerShell\ExecutionPolicy, 495
HKLM moniker, 77
HKLM PowerShell drive, 75
$hklm variable, 78
Hoey, Shane, 375
Hofferle, Jason, 73–74, 252–253
Holmes, Lee, 95, 370
Home directory, variable pointing to, 127
Horizontal tab (`t), 93
$host.ui.PromptForChoice class, 374
Huffman, Clint, 170
Hyper-V, 434

I

identity parameter, 67–68
for Get-ADDomainController cmdlet, 53
IEEERemainder method of System.Math class, 181

ImpersonationLevel property of WindowsIdentity object, 84
Import-Alias cmdlet, 650
Import-CliXml cmdlet, 204, 370
importing credentials, 370–372
Import-Module cmdlet, 50, 213, 316, 328
 –Verbose parameter, 317
Import-PSSession cmdlet, 95
include file, functions and, 148–150
incrementing version control numbers, 527
indexing, array, 39
infrastructure of WMI, 163
InLineGetIPDemo.ps1 script, 261
InlineScript activity, 650–651, 656, 657
in-progress scripts, documenting, 300–301
input, 339. *See also* passwords
 out of bounds errors from, 429
 prompt for, 373–374
$input.current property, 271
input methods, 340–373. *See also* Param statement
 best practices, 340
 connection strings, 372–373
 reading from command line, 340–348
  $args automatic variable for, 340–342
  multiple values for $args, 343–347
inputobject, for Get-CimAssociatedInstance cmdlet, 38
Inputs help tag, 290
$input variable, 271
InstallDate property of Win32_LogicalDisk class, 249
installed modules, list of, 328
installing
 Active Directory module, 47–48
 modules, 319–331, 330
 Remote Server Administration Tools (RSAT), 47
 Windows PowerShell, 6
instancename parameter, for DSC configuration, 663
Integrated Scripting Environment (ISE), 329
Intellisense, 610–611
interactive use of Windows PowerShell, 4
international audience, comments and, 297–298
Internet Explorer Developer Tools, 458
Internet Information Services (IIS)
 as PWA requirement, 26
 Windows PowerShell as application, 82
Internet zone, 493–495
InvalidOperation CategoryInfo field, 157
Invalid parameter error, 122
InvocationName property of System.Management.Automation.InvocationInfo class, 178
Invoke-CimMethod cmdlet, 618
Invoke-Command cmdlet, 80, 82, 105, 152, 205, 510, 619, 632–633
 [ADSISearcher] and, 212
Invoke-Expression cmdlet, 196, 268, 435
Invoke-History cmdlet, 204, 650
Invoke-WmiMethod cmdlet, 202, 619, 624
Invoke-WSManAction cmdlet, 619
Io.Path .NET Framework class, 548
IPConfig.exe, 5, 381
 running commands, 10
IsAccountSid method of SecurityIdentifier class, 87
IsAnonymous property of WindowsIdentity object, 84
IsAuthenticated property of WindowsIdentity object, 84
ISE. *See* Windows PowerShell ISE
IsEqualDomainSid method of SecurityIdentifier class, 87
ISE Snippets, 257
IsGuest property of WindowsIdentity object, 84
IsInRole method of WindowsPrincipal class, 97
IsRestricted method of Shell.Application object, 186
IsServiceRunning method of Shell.Application object, 187
IsSystem property of WindowsIdentity object, 84
IsValidTargetType method of SecurityIdentifier class, 87
IsWellKnown method of SecurityIdentifier class, 87

J

jobs in PowerShell, 634–641
 creating, 20
Join-DtcDiagnosticResourceManager cmdlet, 619
Join-Path cmdlet, 127-128, 189–190, 285, 322, 533, 544
Jones, Don, 26, 516, 521, 585

K

k command (debugging), 595
Kearney, Sean, 23
$key variable, 78
keywords, standard set for comments, 306–307
Kinect for Windows, 409
Klindt, Todd, 108–109

L

lab environment, 470
language statement, breakpoint and, 593
LastErrorCode property of Win32_LogicalDisk class, 249
LDAP attributes, 62
LDAPFilter parameter, 63
LDAP search filter syntax, 209–210
  resources, 230
LDIFDE, 471–473
learning, 106
  to script, 230
  Windows PowerShell, 109
legacy code, migrating to PowerShell, 79
–like operator, 89
Limit-EventLog cmdlet, 619, 624
line continuation character, 191, 305
line number, setting breakpoint on, 588–589
Line property of System.Management.Automation.InvocationInfo class, 178
Link help tag, 290
List command, 596, 600
loaded modules, listing, 312
loading modules, 108
$localappdata environment variable, 126
local computer
  BIOS information from, 33
  storing scripts in, 515
  workflow on, 644–645
LocalMachine, as default for execution policy setting, 503
locked-out users, locating, 64–66
Log10 method of System.Math class, 181
LogChartProcessWorkingSet.ps1 script, 537, 538–539, 542
LogEvent method of WshShell object, 130
log files
  appending to, 532, 536–539
  Application log, 554–555
  networked, 548–549
  from script testing, 456–457
  from Start-Transcript function, 468–470
  writing errors to, 468
logging
  benefits, 550–552
  designing approach for, 532–542
  to event log, 552–555
  for building maintainable scripts, 541–542
  Out-File cmdlet for, 539–541
  to registry, 556–557
  to text files, 531–552
    directory for storing, 542–546
  for troubleshooting, 537
logic errors, 564–565, 585
logic operators, in LDAP search filter, 210
Log method of System.Math class, 181
logoff script, 507
logon date, listing all users with last, 70
Logon.ps1 logon script, 510–511
logon scripts, 507–514
  items included, 509–510
  methods of calling, 512–513
  for Set-ExecutionPolicy cmdlet, 497–498
LogonScriptWithLogging.ps1 script, 532–534
Log resource provider, 660
looping, 247, 595
looping array, 119
Lopez, Juan Carlos Ruiz, 236–237, 256, 595

M

Maheu, Georges, 162, 193
mailing list, 74
Major part of operating system version, 102
major version of script, 527
MakeCab.exe utility, 189
makecab.expandcab object, 185
makecab.makecab object, 183, 185
Managed Object Format (MOF) file, 660–661
MandatoryParameter.ps1 script, 400
mandatory parameters, 350–351, 353, 399–400
manifest for module, 333
–match operator, 89–90, 93
MaximumComponentLength property of Win32_LogicalDisk class, 249
$MaximumHistoryCount variable, 135
Max method of System.Math class, 181
Maxvalue property, 362
Mayer, Keith, 76
McGlone, Ashley, 45–46
MD5, hash creation with, 8
MeasureAddOneFilter.ps1 script, 271
MeasureAddOneR2Function.ps1 script, 273
Measure-Command cmdlet, 448–449, 451–455
Measure-Object cmdlet, 113

MediaType property of Win32_LogicalDisk class, 249
Mell, Bill, 13
memory, requirements for script, 167
menu hierarchy, script to create, 146
message box from WshShell object, 130–132
messages, printing, 221
methods
  overload display, 484
  static, 179, 180–182
metrics for shared drives, 511–512
Microsoft.ActiveDirectory.Management.ADOrganizationalUnit object, 216
Microsoft Baseline Security Analyzer (MBSA), 222
Microsoft Download center, 26
Microsoft Fix It blog, 276
Microsoft.Office.Interop.Word.WdSaveFormat enumeration, 227
Microsoft.PowerShell.Management module, 313
Microsoft.PowerShell.Utility module, 313
Microsoft SharePoint Portal product, 73
Microsoft SkyDrive, for profile, 135
Microsoft Systems Center Configuration Manager package, 7
Microsoft Visual Studio, 387
Microsoft.Win32.Registry .NET Framework class, 78
migrating legacy code to PowerShell, 79
Minasi, Mark, 381
MinimizeAll method of Shell.Application object, 187
Min method of System.Math class, 181
Minor part of operating system version, 102
minor version of script, 527
missing data exception, 345
missing parameters, 398–400
modular code, 226
module drive, creating, 324–326
ModuleName System.Management.Automation.ScriptInfo object, 178
module parameter, 17
  for Get-Command cmdlet, 48
$modulePath variable, 322–324
module repository, centralized, 72
modules
  basics, 311
  checking for dependencies, 326–329
  creating, 331–337
  folder for, 311–312, 319–322
  for packaging workflow, 643
  installing, 319–331, 330
  listing available, 312–314
  list of installed, 328
  loading, 316–318
  for profiles, 146
  requiring specific, 415
  structured requirements, 108–109
  uses for, 314–315
  using from share, 330–331
Module System.Management.Automation.ScriptInfo object, 178
MOF files, 663–664
Moravec, David, 8–9
More.com executable, 117
more function, 116
Move-ADObject cmdlet, 63
MSDN reference library, 193
MSDN website, 109
MSI packages, local deployment, 515
multiple-line comments, 277, 287–288
MultiplyNumbersCheckParameters.ps1 script, 361–362
Muscetta, Daniele, 501–503
MyCommand property of System.Management.Automation.InvocationInfo class, 178
My-Function.ps1 script, 564–565
My-Test function, 254

N

named parameters
  multiple, 122–126
  vs. positional, 239
–name parameter, dollar sign ($) and, 128
Name property
  of Win32_LogicalDisk class, 249
  of WindowsIdentity object, 84
names
  for functions, 237
    verb-noun convention for, 116
  for jobs, 636
  for parameters, 62
  for returned job object, 638
NameSpace method of Shell.Application object, 187, 188
__NAMESPACE property of Win32_Logical Disk class, 250
namespaces, 27, 163–166
  generating list for machine, 164–166

in global variable, 392–393
hierarchical, 162
Name System.Management.Automation.ScriptInfo object, 178
naming conventions, 153
for cmdlets, 7
standard, 145
nesting, comments for documenting, 305–306
.NET assemblies, 135
.NET Framework, 229
classes, 159, 395
cmdlets and, 483
lack of support, 179–193
namespace, 7
for reading registry, 78–79
.NET Framework System.Security.SecureString class, 372
.NET Framework versions
dependencies, 182
determining, 101–102
history, 109, 505
in registry, 100
requirements, 100–102
NETSH, 189
NetSh Advanced Firewall commands, 230
network drives, mapping with WshNetwork object, 533
networked log files, 548–549
network file share
for script repository, 546–547
outputting log text file to, 551
New-ADComputer cmdlet, 60
New-ADGroup cmdlet, 61
New-ADOrganizationalUnit cmdlet, 60
New-ADUser cmdlet, 60, 63
New-Alias cmdlet, 25, 114, 650
–description parameter, 114
New-Cab function, 183, 191
New-CimInstance cmdlet, 619
New-CimSession cmdlet, 158, 620
New-DDF cmdlet, 192
New-DDF function, 189
New-EventLog cmdlet, 552–553, 555, 620, 624
New-IseSnippet cmdlet, 612
New-Item cmdlet, 114, 142, 323, 533, 556, 571
for profile, 144
for variables, 126
New-ItemProperty cmdlet, 556, 571
new line (`n), 93
New-LocalGroupFunction.ps1 script, 465–466
New-LocalUserFunction.ps1 script, 460–461
New-ModulesDrive.ps1 script, 325–326
New-Object cmdlet, 182, 209, 455, 483
WshShell object instance created with, 77
New-PSDrive cmdlet, 133, 324, 421
New-PSSession cmdlet, 620, 631
New-PSSessionConfigurationFile cmdlet, 513
New-PSWorkflowSession cmdlet, 620
New-TempFile.ps1 script, 548–549
New-TestConnection function, 358–359
New-Variable cmdlet, 127-128, 650
New-WebServiceProxy cmdlet, 254, 544
New-WSManInstance cmdlet, 620
non-terminating errors, 410, 428
–noprofile switch, 313
Norman, Richard, 393
Notepad, 106, 229, 457, 540–541, 569–570
editing profile in, 142
temporary file display in, 308
Notes help tag, 290
not (!) operator, 188, 346
–not operator, 328
noun in function name, 240
noun name, changing SID into, 86
NTFS File System (NTFS) permissions, 363
null (`0), 93
$null variable, 448
NumberOfBlocks property of Win32_LogicalDisk class, 249

O

ObjectGUID for the identity parameter, 68
objects, 163–166
storing, 166–167
OffsetInLine property of System.Management.Automation.InvocationInfo class, 178
one-line comments, 303–304
On Error Resume Next statement (VBScript), 427
Open method of Shell.Application object, 187
OpenPasswordProtectedExcel.ps1 script, 372
OpenPasswordProtectedWord.ps1 script, 372–373
operating system
scripting requirements, 102–106
verifying for remote domain controller, 56
versions, 102, 104
compatibility issues, 155–158

error handling for detecting, 220
trapping, 160–161
WMI classes and, 424
Operations Manager, links for Windows Updates downloads, 222
options for cmdlets, 16–17
[ordered] tag, 565
organizational units (OU)
creating, 60, 63
names using special characters, 211
retrieving listing of, 216
OSVersion property of System.Environment .NET Framework class, 102, 103
$oupath variable, 63
-OutBuffer parameter, 17
Out-Default cmdlet, 379–381
Out-File cmdlet, 190, 220, 382, 548
–append parameter, 435
for logging, 539–541
Out-GridView cmdlet, 650
Out-Host cmdlet, 379–381
Out-Null cmdlet, 308, 323, 556
out of bounds errors, 429–431
out-of-range loop, 595
output, 339, 374–395
consistency of, 247
to email, 387–388
to file, 382–383
from functions, 388–392
to screen, 376–382
splitting to screen and file, 383–387
outputpath parameter, for DSC configuration, 663
Outputs help tag, 290
-Outvariable parameter, 17
overriding existing commands, 117–118
overwriting log file, 532–534
Owner property of WindowsIdentity object, 84

P

Package resource provider, 660
package.xml file, 222
paging
more function for, 116
Windows PowerShell ISE and, 25
–paging parameter, 380
parallel activities in workflow, 645–648, 651–652
parameter binding error, Trap statement to catch, 346–347
ParameterBindingException, 343, 346–347
Parameter help tag, 290
parameters, 237
aliases for, 354
attributes, 353–354
creating, 241
documenting, 278
Intellisense and, 610
limiting number, 258
listing in ISE, 607
mandatory, 350–351, 399–400
missing, 398–400
multiple named, 122–126
names, 62
supplying full, 217
passing multiple, 120–126, 248–253, 257–258
with $args, 120–122
placing limits on, 430–431
standard for testing scripts, 460–466
type constraints for, 122–123, 253–255
validating input, 356–361
vs. hard-coding, 247
well-defined, 247
ParameterSetName parameter attribute argument, 353
ParameterSets System.Management.Automation.ScriptInfo object, 179
Parameters System.Management.Automation.ScriptInfo object, 179
Param statement, 183, 248, 257, 348–362
default value assignment in, 399
for DSC configuration, 663–673
location in script, 348
multiple arguments in, 361–362
parentheses
error message from missing, 127
in functions, 116
Parent property of Shell.Application object, 187
ParseScriptCommands.ps1 script, 485–486
parsing script files, 480
partial parameter completion, 183
passwords, 362–372
domain policy for, 57
in virtual machine, 472
in MOF file, 668
prompt for, 367–369
as PSCredential object, 669

storing
Active Directory Domain Services for, 366–367
registry for, 365–366
in script, 363–364
text file for, 364–365
for user accounts, 63
pasting functions into Function library script, 244
path, 499
adding script directory to, 293–294
to Favorites folder, 128
length on command line, 133
in profile, 134–135
to Windows Update Log, 126–127
path parameter, for Get-Content cmdlet, 196
__PATH property of Win32_Logical Disk class, 250
$path variable, 77, 434
pause in script execution. *See* breakpoints
PDF document, creating, 376
performance, process block and, 273–274
permissions, 513
errors from, 561
listing required in documentation, 281
NTFS File System (NTFS), 363
to unlock user account, 65
persistence points in workflow, 643
persisting a workflow, 652–658
Pfeiffer, Mike, 535–536
ping command, 195, 402–403, 629
PingComputers.ps1 script, 358–359
PingIpAddress.ps1 script, 359–360
pinning Windows PowerShell
to desktop taskbar, 605
to Start screen and taskbar, 14
$pinToStart, 14
PinToStartAndTaskBar.ps1 script, 14
$pinToTaskBar, 14
pipeline, 71, 352, 393, 435, 444–445, 446–450
function in, vs. filter, 270–272
for job object, 640
performance improvements from technique, 220
scope of, 356
PipelineLength property of System.Management.Automation.InvocationInfo class, 178
PipelinePosition property of System.Management.Automation.InvocationInfo class, 178
PipelineVariable parameter, 355
piping ipconfig / results to text files, 10
PI property of System.Math class, 182
PNPDeviceID property of Win32_LogicalDisk class, 249
Popup method of WshShell object, 130
icon values, 132
portability of functions, 248
Portable Script Center, 229
PoshPAIG (PowerShell Audit/Install GUI), 337
positional arguments, 239
PositionMessage property of System.Management.Automation.InvocationInfo class, 178
Position parameter attribute argument, 353
PowerManagementCapabilities property of Win32_LogicalDisk class, 249
PowerManagementSupported property of Win32_LogicalDisk class, 249
PowerShell drives, 133–134
PowerShell ISE. *See* Windows PowerShell ISE
Pow method of System.Math class, 181
prerequisites, listing in comments, 299–300
presentation, data-gathering functions separated from, 264–265
printer, sending information to, 387
printing
information to script user, 251
messages, 221
production code, 275. *See also* code
profile configuration, 111–140
alias creation, 112–116
function creation, 116–120
PSDrives, 133–134
variables creation, 126–133
profiles
choosing correct, 141–143
creating, 141–145
modules for, 146
as script, 139
uses, 145–146
working with, 134–139
$profile variable, 142
program ID, from COM object, 182
program logic, 259
programming, 236. *See also* code
programming language, 309
progress indicator
for workflow, 653–654
Write-Debug statement for, 188
prompt
for input, 373
for password, 367–369

PromptForChoice method, 401–402
prompt function, in profile, 134
properties
 –contains operator for testing, 406–408
 retrieving all associated with user object, 70
 static, 180–182
 variables to store, 35
 Windows PowerShell display of, 221
__PROPERTY_COUNT property of Win32_Logical Disk class, 250
property parameter of Get-CimInstance cmdlet, 33
ProtectedFromAccidentalDeletion parameter, 61
ProviderName property of Win32_LogicalDisk class, 249
providers in WMI, 168–169
__provider WMI class, 416–417
 properties, 418–419
Prox, Boe, 337
$PSBoundParameters automatic variable, 160
PSBreakpoints, 585
psc1 file extension, 15
–PSComputerName parameter, for PowerShell workflow, 647
PSConsole file, 15
PSCredential object, 368, 370
 password as, 669
PSCX (PowerShell Community Extensions), 315
PS drives, 133–134
 creating, 421
 naming convention for, 145
PSDscAllowPlainTextPassword setting, 666
$pshome automatic variable, 377
.psm1 extension, 331
$PSModulePath, 324
PSModulePath environment variable, 48, 321
PSParser class, 487
 Tokenize method, 484–486
PSScheduledJob module, 20
PSSession, on remote computer, 56
PSStatus propertyset of Win32_Logical Disk class, 250
$PSVersionTable variable, 79, 81, 313
Purpose property of Win32_LogicalDisk class, 249

Q

qualifier, dynamic, 32
query, Active Directory, 208–217
 [ADSISearcher] for, 209
 cmdlets for, 213–214
 command line for, 214–217
QueryComputersPromptForPassword.ps1 script, 367–368
QueryComputersUseCredentialsFromADDS.ps1 script, 366–367
QueryComputersUseCredentialsFromRegistry.ps1 script, 365–366
QueryComputersUseCredentialsFromText.ps1 script, 364–365
QueryComputersUseCredentials.ps1 script, 363–364
Query parameter, 171
Quit command, 600
quitting debugging session, 588
QuotasDisabled property of Win32_LogicalDisk class, 250
QuotasIncomplete property of Win32_Logical Disk class, 250
QuotasRebuilding property of Win32_Logical Disk class, 250
quotation marks for strings, 124

R

Rahim, Ibrahim Abdul, 439, 455, 458
range operator (..), 113
readability of scripts, 309–310
Read-Host cmdlet, 238, 367, 373
 AsSecureString parameter, 368
ReadHostQueryDrive.ps1 script, 373–374
ReadHostSecureStringQueryWmi.ps1 script, 368
reading text file, 196–203
read mode, 590–592
read-only aliases, making, 114–115
read-only cmdlet, deleting, 115
read-only variables, 126
readwrite mode, 591–592
Receive-DtcDiagnosticTransaction cmdlet, 620
Receive-Job cmdlet, 621, 635, 638
 –Keep, 641
Receive-PSSession cmdlet, 621
RecursiveWMINameSpaceListing.ps1 script, 164–166
Recycle Bin, for Active Directory, 71
redirection operators, 531
ReferenceEquals method of System.Math class, 181
RefreshMenu method of Shell.Application object, 187

RegDelete method of WshShell object, 130
Register-CimIndicationEvent cmdlet, 621
Register-PSSessionConfiguration cmdlet, 514
Register-WmiEvent cmdlet, 621, 624
registry
  logging to, 556–557
  logon scripts writing to, 509
  modifying to enable or disable script execution policy, 495–496
  modifying values through, 175–179
  .NET Framework version in, 100
  for password storage, 365–366
  PowerShellVersion value, 75
  reading, 75
    .NET for, 78–79
    RegRead for, 77
    WMI for, 77–78
  reading and writing to, 159
  searching for WMI provider registration, 420–421
registry resource provider, 660, 664
RegRead method of WshShell object, 130
regular expression
  for IP address, 359
  pattern, 358
RegWrite method of WshShell object, 130
__RELPATH property of Win32_Logical Disk class, 250
remote computer
  checking status of services, 152
  for cross-domain situation, 158
  fan-out commands for, 205–208
  help desk scripts for troubleshooting, 517
  limiting data returned from, 33–34
  PSSession on, 56
  retrieving information from, 107
  running single PowerShell command, 632–634
Remote Desktop, 199–200
  Windows Firewall and, 201
remote domain controller, verifying operating system for, 56
RemoteHelpRunSpace help tag, 290
remote PowerShell session, credentials for, 55
Remote Procedure Call (RPC), 82
Remote Server Administration Tools (RSAT), 49
  installing, 47
remote servers, restarting services on, 173
RemoteSigned script execution policy, 140, 493, 495, 502
remote Windows PowerShell session, creating, 629–632
RemoteWMISessionNoDebug.ps1 script, 561–562
remoting, 81–82, 315
  classic, 615–626
  server connection for Active Directory module access, 48–49
  Windows PowerShell Workflow for, 80
Remove-ADGroupMember cmdlet, 62
Remove-CimInstance cmdlet, 621–622
Remove-CimSession cmdlet, 622
Remove-Computer cmdlet, 622, 624
Remove-EventLog cmdlet, 622, 624
Remove-Item cmdlet, 115
Remove-Item snippet, 613
Remove-Job cmdlet, 636
Remove-OutPutFile function, 286
Remove-Printer cmdlet, 622
Remove-PrinterDriver cmdlet, 622
Remove-PrinterPort cmdlet, 622
Remove-PrintJob cmdlet, 622
Remove-PSBreakpoint cmdlet, 586, 601, 650
Remove-PSSession cmdlet, 213, 622, 631–632
Remove-PSSnapin cmdlet, 650
Remove-Variable cmdlet, 128, 391, 650
Remove-WmiObject cmdlet, 622, 624
Remove-WSManInstance cmdlet, 622
Rename-ADObject cmdlet, 60
Rename-Computer cmdlet, 622, 624
renaming sites in Active Directory, 59–60
repeatability
  and decision to script, 73
  of scripts, 219–223
Repeat command, 596
reporting scripts, 516
$report variable, 434
repository for scripts, 524, 546–547
Representational State Transfer (REST) web services, 455
RequireModuleVersion.ps1 script, 415
#Requires directive, 108
RequiresModule.ps1 script, 108
#requires statement, 83–84, 326–327, 413–415
Reset method of Win32_LogicalDisk class, 249
Resolve-ZipCode function, 254
Resolve-ZipCode.ps1 script, 254
resource management, 409
resources in WMI, 162
Restart-Computer cmdlet, 622, 624
restarting services, on remote servers, 173

Restart-PrintJob cmdlet, 622
restricted endpoint, 513
Restricted script execution policy, 140, 412, 492, 501
Resultant Set of Profiles (RSOP), 144
resultclassname parameter for Get-CimAssociatedInstance cmdlet, 38
$Results variable, eliminating from script, 167
Resume-PrintJob cmdlet, 622
Retrieve-Job cmdlet, 639
return
  behavior of, 267–270
  code capture from dialog box interaction, 131–132
return (r`), 93
Return statement, 465
ReturnValue property, 283
reusing code, 147, 226
  functions for, 244–246
  verbose style for, 247
revision of operating system, 102
Rich Text Content Controls, 376
Riedel, Alexander, 529
rights
  checking for, 412–413
  error handling for missing, 408–415
  for module install, 319
  to run query, 156
Ring, Jan Egil, 19, 26
Role help tag, 290
Rottenberg, Hal, 134–139
Round method of System.Math class, 181
RSAT (Remote Server Administration Tools), 49
  installing, 47
RSS-feed, on new help releases, 20
Run As Different User, 65
Run As Different User dialog box, 626
Run method of WshShell object, 130
runtime errors, 560–564
RuntimeException class, 346

S

Sajid, Osama, 602
SamAccountName for Get-ADUser cmdlet, 67
SAPIEN Technologies, 529
saveas method of Word.Document object, 227
Save-Help cmdlet, 20
SaveWmiInformationAsDocument.ps1 script, 225–229, 227
SaveWordData function, 227
saving BIOS information to CSV file, 152
scheduled task, deploying help with, 26
Schwinn, Dave, 386
scope
  for execution policy, 503
  for pipeline, 356
  for PowerShell drive, 325
  for script execution policy, 140
  for variable, 390
screen
  output to, 376–382
  splitting output to file and, 383–387
script block, curly bracket for opening and closing, 242
ScriptBlock System.Management.Automation.ScriptInfo object, 179
Script Center Script Repository, 26
Script Encoder for the Windows Script Host, 529
script execution policies, 139, 150, 330, 412, 491–495
  default value, 496
  deploying, 495–500
  Group Policy for, 499–500
  purpose of, 492
  settings options, 492–493
  and snippets, 612
ScriptFolderConfig.ps1 script, 661–662
ScriptFolderVersion.ps1 script, 664
ScriptFolderVersionUnzipCreateUsersAndProfile.ps1 script, 669–671
ScriptFolderVersionUnzip.ps1 script, 665–666
scripting
  collaboration in, 229–230
  enabling, 139–140
  methodology, 159
  structured requirements
    applications, 106–108
    modules, 108–109
    operating system, 102–106
    security, 83
  tracking opportunities, 229
“scripting guy” role, 73–74
Scripting Guys Script Repository, 478
scripting pitfalls
  cmdlet support absence, 151–153
  complicated constructors, 153–154
  listing WMI providers, 168–169

.NET Framework support absence, 179–193
objects and namespaces, 163–166
version compatibility issues, 155–158
WMI classes, 169–179
WMI support absence, 162–163
Script-level scope, 391
script-level tracing, 568–572
ScriptLineNumber property of System.Management.Automation.InvocationInfo class, 178
ScriptName property of System.Management.Automation.InvocationInfo class, 178
Script resource provider, 660
$scriptRoot variable, 556
scripts
accessing functions in other, 147–150
benefits, 217–229
adaptability, 225–229
documentability, 223–225
repeatability, 219–223
database of, 229
evaluating, 479–480
evaluating need for, 195–217
folder for, 515
header section, 278
for organization unit creation, 61
parsing, 480
readability of, 309–310
reasons for failure, 551
reasons for writing, 105–106
risk of damage from, 433
risk of deleting wrong version, 527
running, disabled by default, 11
signing, 71
suspending execution, 578
testing, 61
tracking and coordinating development, 73
verifying integrity, 71
writing process, 95
script types
diagnostic, 516
Help desk scripts, 517–520
logon scripts, 507–514
reporting scripts, 516
stand-alone scripts, 515–516
Search-ADAccount cmdlet, 64–65
SearchAllComputersInDomain.ps1 script, 305
searchbase parameter, for Get-ADUser cmdlet, 68
SearchForWordImages.ps1 script, 297
searching files for string pattern, 243–244
secure string, 368
security, 498–499, 524, 547
issues, 11–13
Security.Principal.WindowsBuiltInRole .NET Framework enumeration, 97–98
Security.Principal.WindowsIdentity .NET Framework class, 84
GetCurrent method, 96, 553
selection object, 226
Select-Object cmdlet, 316
self-containment of function, 247
semicolon (;), to separate commands on single line, 10
Send-DtcDiagnosticTransaction cmdlet, 622
SendKeys method of WshShell object, 130
Send-MailMessage cmdlet, 387, 388
Sequence workflow activity, 649, 656–658
__SERVER property of Win32_Logical Disk class, 250
service pack level, determining for client workstations, 28
Service resource provider, 660
services, 170–173
restarting on remote servers, 173
script for stopping and starting, 170
ServiceStart method of Shell.Application object, 187
ServiceStop method of Shell.Application object, 187
sessions, removing unused, 213
Set-ADAccountPassword cmdlet, 64
Set-ADOObject cmdlet, 60
Set-Alias cmdlet, 114, 650
Set-AuthenticodeSignature cmdlet, 504
Set-CimInstance cmdlet, 622
Set-ExecutionPolicy cmdlet, 139, 324, 413, 495, 496–498, 502
logon script for, 497–498
Set-ieStartPage function, 282
Set-Item cmdlet, 126
! SET keyword, to preface variable assignments, 572
Set-LocalGroupFunction.ps1 script, 462–464
Set-Location cmdlet, 133, 629
SetMultiStringValue method, 283
Set-Number function, 356, 357
SetPowerState property of Win32_LogicalDisk class, 249
Set-PrintConfiguration cmdlet, 623
Set-Printer cmdlet, 623
Set-PrinterProperty cmdlet, 623
Set-PSBreakpoint cmdlet, 586, 588–591, 600, 602, 650

–Action parameter, 592
–Command parameter, 593
Set-PSDebug cmdlet, 565, 567–585, 568-569, 650
step parameter, 574, 578
-strict mode, 581–582
tracing problems in, 580–581
Set-PSSessionConfiguration cmdlet, 514
Set-SaverTimeout.ps1 script, 175, 179
Set-ScreenSaverTimeout function, 175
SetScriptExecutionPolicy.vbs script, 498
Set-Service cmdlet, 623, 624
SetServicesConfig.ps1 script, 672
Set-StrictMode cmdlet, 479, 583–584, 650
SetStringValue method, 283
SetTime method of Shell.Application object, 187
settings, 173–175
Set-TraceMode cmdlet, 650
Set-Variable cmdlet, 126, 128, 650
set-verb pattern for cmdlets, 7
Set-WmiInstance cmdlet, 623, 624
Set-WSManInstance cmdlet, 623
SharePoint page, 74
sharing automated fixes, 253
Shell.Application object, 186–187-188, 458
Shell, Brandon, 246, 276
ShellExecute method of Shell.Application object, 187
shortcut keystroke. *See* aliases
ShowBrowserBar method of Shell.Application object, 187
Show-Command cmdlet, 25
Show-EventLog cmdlet, 623-624
shutdown scripts, 513
ShutdownWindows method of Shell.Application object, 187
Siddaway, Richard, 498–499
SID value
of group, obtaining, 86
for identity parameter, 67
Siepser, Gary, 352, 409
signature of function, 260
signing scripts, 71, 504–505, 524
Sign method of System.Math class, 182
Simple Object Access Protocol (SOAP), 455
SimpleTypingErrorNotReported.ps1 script, 582
SimpleTypingError.ps1 script, 581–582
single-line comments, 277–287
creating, 288–289
Sinh method of System.Math class, 182
Sin method of System.Math class, 182
sites, renaming in Active Directory, 59–60
Size property of Win32_LogicalDisk class, 250
SkyDrive, 136
snippets, 611–614
creating, 612–613
user-defined, removing, 613–614
Snover, Jeffrey, 6, 23, 81–82, 95, 105–106
software
removing from servers, 201–202
version control, 529–530
Software Update Services (SUS), for PowerShell deployment, 7
SolarWinds Network Configuration Manager, 15
source control, 521–522
source control repository, 522
special characters
avoiding in organizational unit names, 212
LDAP search filter, 211
SpecialFolders method of WshShell object, 130
SpecialFolders property of WshShell object, 129
split method of System.String class, 321–322
Split-Path cmdlet, 184
Split-Path function, 285
Sqrt method of System.Math class, 182
StackOverflow, 315
Stahler, Wes, 511–512
stand-alone scripts, 515–516
Standard Operating Procedure (SOP) documents, 76
Start-DscConfiguration cmdlet, 663–665, 671
starting services, script for, 170
Start-Job cmdlet, 634, 638-639
startmode of service, 7
Start | Run | PowerShell, 7
Start screen, pinning Windows PowerShell to, 14
Start-Transaction cmdlet, 176, 650
Start-Transcript cmdlet, 550, 630–631, 650
Start-Transcript function, log file from, 468–470
static methods, 179, 180–182
static properties, 180–182
StatusInfo property of Win32_Logical Disk class, 250
Status property of Win32_Logical Disk class, 250
stdRegProv WMI class, 77, 78
Step-into command (s), 596, 600
Step-out command (o), 596
Step-over command (v), 596
stepping through script, 572–581
turning off, 581

Stewart, Bill, 427–429
$stime variable, 434
Stop command (q), 596
Stop-Computer cmdlet, 623, 624
stopping services, script for, 170
Stop-Process cmdlet, 151, 352
Stop-Transcript cmdlet, 469, 550, 650
storing objects, 166–167
Stranger, Stefan, 222–223
Streams.exe Windows SysInternals utility, 494
strict mode, 256, 581–584
    disabling, 582
string
    converting to WMI class, 424
    displaying on screen, 378–379
    wildcard search of, 89
StringArgsArray1.ps1 script, 344–345
StringArgs.ps1 script, 343–344
string pattern, searching files for, 243–244
subroutines, vs. functions, 233–234
__SUPERCLASS property of Win32_Logical Disk class, 250
SupportsDiskQuotas property of Win32_Logical Disk class, 250
SupportsFileBasedCompression property of Win32_Logical Disk class, 250
suppressing error message display, 255
suspending script execution, 578
Suspend method of Shell.Application object, 187
suspend parameter, 11
Suspend-PrintJob cmdlet, 623
Suspend-Workflow workflow activity, 649
Switch_DebugRemoteWMISession.ps1 script, 563–564
switch statement, 401, 564
Syncplicity, 135
Synctoy, 135
Synopsis help tag, 290
syntax errors, debugging, 560
SystemCreationClassName property of Win32_Logical Disk class, 250
System.Diagnostics.Process .NET Framework object, 637
System.DirectoryServices.DirectorySearcher class, 208
System.Enum .NET Framework class, GetNames method of, 98
System.Environment .NET Framework class, 104
    OSVersion property, 102-103
System.IO.FileInfo object, 135
    basename property, 322
    FullName property, converting to string, 191
System.Io.Path .NET Framework class, 285, 434
System.Management.Automation.InvocationInfo class
    properties, 178
System.Management.Automation.LineBreak .NET Framework class, 588, 589
System.Management.Automation.ScriptInfo object, 177–178
    properties, 178–179
System.Management.ManagementClass .NET Framework class, 423
    object creation, 425
System.Math class
    obtaining members of, 180
    static members, 180–182
SystemName property of Win32_Logical Disk class, 250
System Properties dialog box (Windows 8), 199
System.Runtime.InteropServices.RunTimeEnvironment .NET Framework class, GetSystemVersion method of, 100
System.Security.Cryptography.HashAlgorithm class, 8
System.Security.Principal.NTAccount class, 87
System.Security.Principal.WindowsIdentity class
    GetCurrent static method, 90, 92
System.Security.SecureString .NET Framework class, 368
system state, cmdlets to change, 11
System.String class, split method, 321–322
System.SystemException class, 255
System.TimeSpan .NET Framework class, 448
System.Version class, 103

T

tab character, 265
Tabdilio, Mark, 275–276
tab expansion, 561, 610–611
    for module name completion, 316–317
Tanh method of System.Math class, 182
Tan method of System.Math class, 182
taskbar, pinning Windows PowerShell to, 14
tasks, automation of routine, 73–74
TechNet Script Center, 26, 41, 109
Tee-Object cmdlet, 383–385, 543–546, 550
    variable parameter for, 385
temporary file
    for log file, 548

Notepad for displaying, 308
temporary folders, 285–286
terminating errors, 410, 428
TestAdminCreateEventLog.ps1 script, 553–554
Test-ComputerPath.ps1 script, 403
Test-Connection cmdlet, 358, 562, 623, 624
Test-DscConfiguration function, 671
testing
code, 99
errors, $? automatic variable for, 188
for .NET Framework version, 101
software in PowerShell, 439–440
testing scripts, 61
advanced, 470–472
basic syntax checking, 433–444
comparing speed of two, 447–449
displaying results, 456
documenting process, 442–444
evaluating performance of different versions, 450–457
against known data, 471
log from Start-Transcript function, 468–470
looking for errors, 438–439
performance of, 444–459
reducing code complexity, 449–450
running script, 440–441
standard parameters for, 460–466
store and forward approach, 445–446
Test-Path cmdlet for, 149
total running time for, 436
Test-IsAdminFunction.ps1 script, 96–97
Test-IsAdministrator function, 553
Test-IsInRole.ps1 script, 99
Test-ModulePath function, 320
Test-Path cmdlet, 93, 149, 286, 320, 420, 421, 533
Test-ScriptHarness.ps1 script, 433, 436–437
Test-Script.ps1 script, 497
Test-Scripts function, 451
Test-TwoScripts.ps1 script, 451–455
log from, 457
Test-WSMan cmdlet, 623, 628
text files, 340
hash of password stored in, 368–369
logging to, 531–552
directory for storing, 542–546
output to, 382
for password storage, 364–365
piping ipconfig / results to, 10
reading, 196–203
redirecting cmdlet output to, 59
width parameter for output, 540
Throw statement, 188, 346
TileHorizontally method of Shell.Application object, 187
TileVertically method of Shell.Application object, 187
TimeSpan object, 450
time stamp
converting, 394
in log file, 469, 533
TODO: tags, 307
ToggleDesktop method of Shell.Application object, 187
Tokenize method of PSParser class, 484–486
Token property of WindowsIdentity object, 84
ToString method, 86, 87
Trace-Command cmdlet, 650
$trace variable, 538
tracing, script-level, 568–572
tracking
changes with version control, 523
script changes, 527–528
transactions, registry modification with, 176
TranscriptBios.ps1 script, 469–470
transcript tool in PowerShell, 630–631
Translate method, 86-87, 93
trapping operating system version, 160–161
Trap statement, 254-255, 346–347, 410
TrayProperties method of Shell.Application object, 187
TroubleShoot.bat script, 10
troubleshooting. *See also* debugging; errors; testing scripts
logging for, 537, 541
missed closing bracket, 242
version control and, 523
Truman, Jeff, 15–16
Truncate method of System.Math class, 182
Trusted Internet zone, 515
Trusted Sites zone, adding script share to, 495
Try/Catch/Finally construction, 347, 410–412
Tsaltas, Dean, 280
Turn On Script Execution Group Policy setting, 139
two-letter aliases, 113
Tyler, Jonathan, 99–100
type constraints for parameters, 122–123, 253–255
type mismatch error message, 343
TypeText method, 226

U

unapproved verbs, 317–318
UnboundArguments property of System.Management.Automation.InvocationInfo class, 178
Undefined execution policy, 503
UndoMinimizeALL method of Shell.Application object, 187
Undo-Transaction cmdlet, 650
uninitialized variable, 582
universal group, creating, 62
Universal Naming Convention (UNC) path, 548
Universal Naming Convention (UNC) shares, Internet zone and, 495
Unlock-ADAccount cmdlet, 65
  confirm parameter, 66
unlocking user accounts, 64–66
Unrestricted script execution policy, 140, 493, 499, 502
unused sessions, removing, 213
unused user accounts, finding, 68–72
unzipping compressed file, 665
Update-Help cmdlet, 17
UpdateHelpTrackErrors.ps1 script, 18–19
updating help, 17–19
UseADCmdletsToCreateOuComputerAndUser.ps1 script, 61
use case scenario, 397
UseGetMemberOf.ps1 script, 91, 92–95
User Account Control (UAC), 83, 408
user accounts
  creating, 60
  disabled, 63
    finding, 66–68
  enabling, 63–64
  finding and unlocking, 64–66
  finding unused, 68–72
user-defined snippets, 612
  removing, 613–614
user interfaces, automating tests in, 459
user management in Active Directory module, 60–63
user objects
  properties for, 69
  retrieving all properties associated with, 70
  whenCreated property, 69
user preferences for restricted execution policy, 139
%UserProfile% location, 330
User property of WindowsIdentity object, 84
User resource provider, 660, 669
user rights, 155
users
  adding to group, 62
  detecting current, 84–95
  detecting roles, 96–100
  interaction with cmdlet help, 281
  listing all with last logon date, 70
  removing from group, 62
Use-Transaction cmdlet, 650
–UseTransaction switch, 176

V

ValidateCount parameter attribute, 360
ValidateLength parameter attribute, 360
ValidateNotNullOrEmpty parameter attribute, 361-362
ValidateNotNull parameter attribute, 361
ValidatePattern attribute, 358–359, 360
ValidateRange parameter, 360, 430
ValidateRange.ps1 script, 357
ValidateScript parameter attribute, 360
ValidateSet parameter attribute, 361
validating parameter input, 356–361
ValueFromPipelineByPropertyName parameter attribute argument, 353
ValueFromPipeline parameter attribute argument, 353
ValueFromRemainingArguments parameter attribute argument, 353
Value method of SecurityIdentifier class, 87
values, passing to function, 238
$value variable, 78
VariableNotFound error message, 342
variables
  assigning returned job object to, 638–639
  assigning value to, 126
  changing value in suspended script, 579
  CIM instance in, 35
  creating, 126–133
  global, 390–392
  names, 145
    dollar sign and, 341–342
  property selection in, 35
  ! SET keyword to preface assignments, 572
  setting breakpoint on, 589–593
  uninitialized, 582
VBScript, 159, 394
  classic function example, 233

error handling, 427
Verb menu, 146
Verb-Noun pattern, 7, 116
–Verbose parameter, 16, 176, 462–464
$verbosePreference variable, 420
verbose style, code reuse, 247
verbs
for cmdlets, 7
for functions, 234
choosing, 240
unapproved, 317–318
verifying script integrity, 71
version control
avoiding new errors, 522
incrementing numbers, 527
internal number in comments, 525–528
reasons for, 521–528
version control software, 529–530
versions of operating system
compatibility issues, 155–158
trapping, 160–161
versions of script, evaluating performance of different, 450–457
Vertical tab (`v), 93
virtual machine, 5, 434
passwords in, 472
for script testing, 471
Visibility System.Management.Automation.ScriptInfo object, 179
Visual SourceSafe (VSS), 529
VMware plugin, 13
VolumeDirty property of Win32_Logical Disk class, 250
VolumeName property of Win32_Logical Disk class, 250
VolumeSerialNumber property of Win32_Logical Disk class, 250

W

Wait-Job cmdlet, 639
Walker, Jason, 166–168
Wbemtest.exe, 415
Web Application Services Platform (WASP), 459
Web Services Description Language (WSDL), 254, 544
web services, testing, 455
websites, testing, 458
–whatif parameter, 11, 16, 247, 460, 464–467
whenCreated property, for user object, 69
Where-Object cmdlet (?), 107, 121, 191, 449, 598, 614
width parameter, for text file output, 540
wiki page, 74
wildcards, 607
* (asterisk), 24
for filter parameter of Get-ADUser cmdlet, 68
Get-CimInstance cmdlet and, 35
for Get-Command cmdlet search, 214
for module name completion, 316–317
for string search, 89
Wilhite, Brian, 28, 49–50, 201
Willett, Andrew, 550–552
Wilson, Ed, 23
Win32_Bios WMI class, 343
properties available, 33
Win32_ComputerSystem Windows Management Instrumentation (WMI) class, 434
Win32_DefragAnalysis management object, 221
Win32_Desktop WMI class
for desktop settings, 173
properties, 174–175
Win32_LogicalDisk class, 248–249
Win32_NetworkAdapterConfiguration WMI class
retrieving instances, 261
Win32_OperatingSystem management object, 160
Win32_PingStatus WMI class, 402–403
Win32_Process class, 35
Win32_Product WMI class, 417
Win32_UserAccount WMI class, 35, 37
WIN32_Volume Windows Management Instrumentation (WMI) class, 155
Windows 7, 6
Windows 8, 14, 381
Remote Settings, 199
Windows 8.1 client, 6
Windows Automation Snap-in, 459
WindowsFeature resource provider, 660
Windows Firewall, Remote Desktop and, 201
WindowsIdentity object, 553
creating, 90
Groups property, 85–86
properties, 84
returning instance of, 84
Windows Internet Explorer, 493–495
Windows Management Instrumentation (WMI), 201
coded values for registry tree, 77–78
connecting to namespace, 415
for reading registry, 77–78

lack of support, 162–163
listing providers, 168–169
obtaining specific data, 251
remoting, 82
Windows Management Instrumentation (WMI) classes, 169–179
CIM cmdlets exporation with, 27–33
deprecated, 31
filtering classes by qualifier, 30–33
filtering out unwanted names, 121
output clean-up, 34
retrieving instances, 33
Windows Management Instrumentation Tester (Wbem-Test), 415–423
Windows Media Player scripting object model, 182
Windows method of Shell.Application object, 187
Windows, names and versions, 104
Windows PowerShell
accessing, 14
basics, 3–4
benefits of using, 13, 23–24, 318–319
configuring, 15–16
deploying, 7
identifying version, 75
installing, 6
intrinsic techniques, 79–82
jobs, 634–641
learning, 5, 109
location for module search, 48
payback from automation, 76
script for checking prerequisites, 6
syntax and WMI Query Language (WQL) syntax, 158
vs. other scripting languages, 394
Windows PowerShell Community Extensions (PSCX), 134, 315
Windows PowerShell console, 606
Windows PowerShell debugger, 590. *See also* debugging
commands for, 596
Windows PowerShell drives. *See* PS drives
Windows PowerShell formatter, 540
Windows PowerShell ISE
debugging with, 602–603
loading workflow in, 644–645
module creation in, 331
navigating, 606–608
pager and, 25
profile and, 141
running, 605–611
script pane, 608–609
snippets, 611–614
syntax error detection, 560
tab expansion and Intellisense, 610–611
for testing, 440
Windows PowerShell remoting. *See* remoting
Windows PowerShell Web Access (PWA), 26
Windows PowerShell workflow, 643. *See also* workflow
WindowsPrincipal class, IsInRole method of, 97
WindowsProcess resource provider, 660
Windows Remote Management (WinRM), 626–634
WindowsSecurity method of Shell.Application object, 187
Windows Server 2012, installing Active Directory module on, 47
Windows Server 2012 R2, 6
running in core mode, 3
Windows Update, 222
Windows Update Log, storing path to, 126–127
Windows Vista
user Personal folder on, 142
user rights, 155
WindowSwitcher method of Shell.Application object, 187
Windows Workflow Foundation, 643
WinRM (Windows Remote Management), 626–634
WMI providers
missing, 415–423
searching registry for registration, 420–421
WMI Query Language (WQL), 170
Windows PowerShell syntax and, 158
Word.Application object, 226
Word.Document object, saveas method, 227
workarounds, 157
workflow, 516
activities in, 648–652
checkpoint in, 652–655
cmdlets
as activities, 649
disallowed from core modules, 650
as InlineScript activities, 650–651
cool features, 657
for parallel PowerShell, 645–648
on local computer, 644–645
parallel activities, 651–652
persistence points in, 643
reasons to use, 643–645

requirements, 644
sequence activity in, 656–658
workflow keyword, 644
working directory, changing, 3
Wouters, Jeff, 71
–Wrap parameter, 208
WriteBiosInfoToWord.ps1 script, 298–299
Write-Debug cmdlet, 177, 190, 460, 561-562, 565
for progress indicator, 188
Write-EventLog cmdlet, 555, 623-624
Write-Host cmdlet, 265, 347, 389, 592, 650
write mode for breakpoint, 589–590
Write-Output cmdlet, 247
Write-Path function, 241
Write verb, 238
Write-Verbose cmdlet, 176, 420, 421–422, 464, 585
WshNetwork object, mapping network drives with, 533
WshShell object, 79, 128–130
New-Object cmdlet for creating, 77
SpecialFolders property, 129
$wshShell variable, 77
WshSpecialFolders object, 129
WS-Management Protocol, 626
WS-Management (WSMan) cmdlets
remoting with, 82

X

XML file, 395
for console file, 15
debugger to dump variables into, 600
exporting commands to, 203
exporting Lync server configuration to, 375
exporting to, 386
formatted for screen output, 377–378
for snippet, 612
XPath, 395
XQuery statement, 395

Z

Zone.Identifier tag, 494