

Covers PowerShell v2



Windows PowerShell IN ACTION

SECOND EDITION

Bruce Payette

 MANNING

contents

preface xix
acknowledgments xxi
about this book xxiii
about the cover illustration xxix

Part 1 Learning PowerShell 1

- 1 Welcome to PowerShell 3*
 - 1.1 What is PowerShell? 5
 - Shells, command lines, and scripting languages 6 ♦ Why a new shell? Why now? 7 ♦ The last mile problem 8
 - 1.2 Soul of a new language 9
 - Learning from history 9 ♦ Leveraging .NET 10
 - 1.3 Brushing up on objects 11
 - Reviewing object-oriented programming 11 ♦ Objects in PowerShell 13
 - 1.4 Up and running with PowerShell 13
 - PowerShell 14 ♦ Starting PowerShell 14 ♦ The PowerShell console host 14 ♦ The PowerShell Integrated Scripting Environment 17 ♦ Command completion 20
 - 1.5 Dude! Where's my code? 22
 - Navigation and basic operations 22 ♦ Basic expressions and variables 23 ♦ Processing data 25 ♦ Flow control statements 30 ♦ Scripts and functions 31 ♦ Remoting and the Universal Execution Model 32
 - 1.6 Summary 35

2	<i>Foundations of PowerShell</i>	36
2.1	Getting a sense of the PowerShell language	37
2.2	The core concepts	38
	Command concepts and terminology	38 ♦ Commands and cmdlets 38 ♦ Command categories 42
2.3	Aliases and elastic syntax	46
2.4	Parsing and PowerShell	50
	How PowerShell parses	51 ♦ Quoting 51 ♦ Expression-mode and command-mode parsing 54 ♦ Statement termination 56 ♦ Comment syntax in PowerShell 58
2.5	How the pipeline works	60
	Pipelines and streaming behavior	61 ♦ Parameters and parameter binding 62
2.6	Formatting and output	64
	The formatting cmdlets	64 ♦ The outputter cmdlets 67
2.7	Summary	70
3	<i>Working with types</i>	72
3.1	Type management in the wild, wild West	72
	PowerShell: a type-promiscuous language	73 ♦ The type system and type adaptation 75
3.2	Basic types and literals	77
	String literals	77 ♦ Numbers and numeric literals 82
3.3	Collections: dictionaries and hash tables	85
	Creating and inspecting hash tables	85 ♦ Modifying and manipulating hash tables 88 ♦ Hash tables as reference types 90
3.4	Collections: arrays and sequences	91
	Collecting pipeline output as an array	91 ♦ Array indexing 92 ♦ Polymorphism in arrays 92 ♦ Arrays as reference types 93 ♦ Singleton arrays and empty arrays 94
3.5	Type literals	96
	Type name aliases	96 ♦ Generic type literals 98 ♦ Accessing static members with type literals 99
3.6	Type conversions	101
	How type conversion works	101 ♦ PowerShell's type-conversion algorithm 104 ♦ Special type conversions in parameter binding 107
3.7	Summary	109

4	<i>Operators and expressions</i>	110
4.1	Arithmetic operators	112
	The addition operator	113 ♦ The multiplication operator
	Subtraction, division, and the modulus operator	117
4.2	The assignment operators	119
	Multiple assignments	120 ♦ Multiple assignments with type
	qualifiers	121 ♦ Assignment operations as
	value expressions	123
4.3	Comparison operators	124
	Scalar comparisons	125 ♦ Comparisons and case
	sensitivity	127 ♦ Using comparison operators
	with collections	129
4.4	Pattern matching and text manipulation	131
	Wildcard patterns and the -like operator	132 ♦ Regular
	expressions	133 ♦ The -match operator
	operator	134 ♦ The -replace
	operator	137 ♦ The -join operator
	operator	139 ♦ The -split
	operator	143
4.5	Logical and bitwise operators	148
4.6	Summary	150
5	<i>Advanced operators and variables</i>	151
5.1	Operators for working with types	152
5.2	The unary operators	154
5.3	Grouping and subexpressions	157
	Subexpressions \$(...)	159 ♦ Array subexpressions @(...)
		160
5.4	Array operators	162
	The comma operator	162 ♦ The range operator
	Array indexing and slicing	167 ♦ Using the range
	operator with arrays	170 ♦ Working with
	multidimensional arrays	171
5.5	Property and method operators	173
	The dot operator	174 ♦ Static methods and the double-colon
	operator	177 ♦ Indirect method invocation
		178
5.6	The format operator	179
5.7	Redirection and the redirection operators	181
5.8	Working with variables	184
	Creating variables	185 ♦ Variable name syntax
	Working with the variable cmdlets	188
	Splatting a variable	193
5.9	Summary	196

6	<i>Flow control in scripts</i>	198
6.1	The conditional statement	200
6.2	Looping statements	203
	The while loop	203 ♦ The do-while loop
	The for loop	205 ♦ The foreach loop
	The foreach loop	207
6.3	Labels, break, and continue	212
6.4	The switch statement	215
	Basic use of the switch statement	215 ♦ Using wildcard patterns with the switch statement
	Using regular expressions with the switch statement	216 ♦ Using regular expressions with the switch statement
	Processing files with the switch statement	217 ♦ Using the \$switch loop enumerator in the switch statement
	Using the \$switch loop enumerator in the switch statement	222
6.5	Flow control using cmdlets	223
	The ForEach-Object cmdlet	223 ♦ The Where-Object cmdlet
	The Where-Object cmdlet	228
6.6	Statements as values	231
6.7	A word about performance	233
6.8	Summary	234
7	<i>PowerShell functions</i>	236
7.1	Fundamentals of PowerShell functions	237
	Passing arguments using \$args	237 ♦ Example functions: ql and qs
	Simplifying \$args processing with multiple assignment	239 ♦ Simplifying \$args processing with multiple assignment
	Simplifying \$args processing with multiple assignment	240
7.2	Declaring formal parameters for a function	241
	Mixing named and positional parameters	242 ♦ Adding type constraints to parameters
	Handling variable numbers of arguments	243 ♦ Handling variable numbers of arguments
	Initializing function parameters with default values	245 ♦ Initializing function parameters with default values
	Handling mandatory parameters, v1-style	246 ♦ Handling mandatory parameters, v1-style
	Handling mandatory parameters, v1-style	248
	Using switch parameters to define command switches	248
	Switch parameters vs. Boolean parameters	252
7.3	Returning values from functions	257
	Debugging problems in function output	259 ♦ The return statement
	The return statement	262
7.4	Using simple functions in a pipeline	263
	Filters and functions	265 ♦ Functions with begin, process, and end blocks
	Functions with begin, process, and end blocks	266
7.5	Managing function definitions in a session	267

7.6	Variable scoping in functions	269
	Declaring variables	270 ♦ Using variable scope modifiers 272
7.7	Summary	273
8	<i>Advanced functions and scripts</i>	275
8.1	PowerShell scripts	276
	Script execution policy	276 ♦ Passing arguments to scripts 278 ♦ Exiting scripts and the exit statement 280
	Scopes and scripts	281 ♦ Managing your scripts 284
	Running PowerShell scripts from other applications	285
8.2	Writing advanced functions and scripts	287
	Specifying script and function attributes	288 ♦ The CmdletBinding attribute 289 ♦ The OutputType attribute 293 ♦ Specifying parameter attributes 296
	Creating parameter aliases with the Alias attribute	303
	Parameter validation attributes	305
8.3	Dynamic parameters and dynamicParam	311
	Steps for adding a dynamic parameter	312 ♦ When should dynamic parameters be used? 314
8.4	Documenting functions and scripts	314
	Automatically generated help fields	315 ♦ Creating manual help content 315 ♦ Comment-based help 316 ♦ Tags used in documentation comments 318
8.5	Summary	321
9	<i>Using and authoring modules</i>	322
9.1	The role of a module system	323
	Module roles in PowerShell	324 ♦ Module mashups: composing an application 324
9.2	Module basics	325
	Module terminology	326 ♦ Modules are single-instance objects 326
9.3	Working with modules	327
	Finding modules on the system	327 ♦ Loading a module 331
	Removing a loaded module	335
9.4	Writing script modules	337
	A quick review of scripts	338 ♦ Turning a script into a module 340 ♦ Controlling member visibility with Export-ModuleMember 343 ♦ Installing a module 347 ♦ How scopes work in script modules 348 ♦ Nested modules 350

9.5	Binary modules	353
	Binary modules versus snap-ins	354 ♦ Creating a binary module 355 ♦ Nesting binary modules in script modules 357
9.6	Summary	360
10	<i>Module manifests and metadata</i>	361
10.1	Module folder structure	362
10.2	Module manifest structure	363
10.3	Production manifest elements	366
	Module identity	368 ♦ Runtime dependencies 368
10.4	Construction manifest elements	370
	The loader manifest elements	371 ♦ Module component load order 374
10.5	Content manifest elements	375
10.6	Language restrictions in a manifest	376
10.7	Advanced module operations	378
	The PSModuleInfo object	378 ♦ Using the PSModuleInfo methods 382 ♦ The defining module versus the calling module 384 ♦ Setting module properties from inside a script module 388 ♦ Controlling when modules can be unloaded 388
	Running an action when a module is removed	389
10.8	Summary	390
11	<i>Metaprogramming with scriptblocks and dynamic code</i>	392
11.1	Scriptblock basics	393
	Invoking commands	394 ♦ The scriptblock literal 397
	Defining functions at runtime	398
11.2	Building and manipulating objects	400
	Looking at members	400 ♦ Using Add-Member to extend objects 402 ♦ Adding note properties with New-Object 409
11.3	Using the Select-Object cmdlet	410
11.4	Dynamic modules	412
	Dynamic script modules	412 ♦ Closures in PowerShell 414
	Creating custom objects from modules	417
11.5	Steppable pipelines	418
	How steppable pipelines work	418 ♦ Creating a proxy command with steppable pipelines 420
11.6	A closer look at the type-system plumbing	423
	Adding a property	425 ♦ Shadowing an existing property 427

11.7	Extending the PowerShell language	428
	Little languages	428 ♦ Adding a CustomClass keyword to PowerShell 428 ♦ Type extension 433
11.8	Building script code at runtime	436
	The Invoke-Expression cmdlet	436 ♦ The ExecutionContext variable 437 ♦ The ExpandString() method 437 ♦ The InvokeScript() method 438 ♦ Mechanisms for creating scriptblocks 438 ♦ Creating functions using the function: drive 439
11.9	Compiling code with Add-Type	440
	Defining a new .NET class: C#	440 ♦ Defining a new enum at runtime 442 ♦ Dynamic binary modules 443
11.10	Summary	445
12	<i>Remoting and background jobs</i>	447
12.1	Getting started with remoting	448
	Commands with built-in remoting	448 ♦ The PowerShell remoting subsystem 449 ♦ Enabling remoting 450
	Additional setup steps for workgroup environments	451
	Enabling remoting in the enterprise	452
12.2	Applying PowerShell remoting	454
	Basic remoting examples	454 ♦ Adding concurrency to the examples 455 ♦ Solving a real problem: multimachine monitoring 457
12.3	Sessions and persistent connections	462
	Additional session attributes	466 ♦ Using the New-PSSession cmdlet 468 ♦ Interactive sessions 469 ♦ Managing PowerShell sessions 472
12.4	Implicit remoting	473
	Using implicit remoting	474 ♦ How implicit remoting works 476
12.5	Background jobs in PowerShell	481
	The job commands	483 ♦ Working with the job cmdlets 483
	Working with multiple jobs	487 ♦ Starting jobs on remote computers 489 ♦ Running jobs in existing sessions 492
12.6	Considerations when running commands remotely	493
	Remote session startup directory	494 ♦ Profiles and remoting 494 ♦ Issues running executables remotely 495
	Reading and writing to the console	496 ♦ Remote output vs. local output 497 ♦ Processor architecture issues 498
12.7	Summary	500

13	<i>Remoting: configuring applications and services</i>	502
13.1	Remoting infrastructure in depth	503
	The PowerShell remoting protocol stack	503 ♦ Using the WSMAN cmdlets and providers 509 ♦ Authenticating the target computer 511 ♦ Authenticating the connecting user 514
	Addressing the remoting target	518 ♦ Windows version-specific connection issues 520 ♦ Managing resource consumption 522
13.2	Building custom remoting services	527
	Remote service connection patterns	527 ♦ Working with custom configurations 530 ♦ Creating a custom configuration 531
	Access controls and endpoints	533 ♦ Constraining a PowerShell session 535 ♦ Creating a constrained execution environment 543
13.3	Summary	551
14	<i>Errors and exceptions</i>	553
14.1	Error handling	554
	ErrorRecords and the error stream	555 ♦ The \$error variable and -ErrorVariable parameter 560 ♦ Determining if a command had an error 564 ♦ Controlling the actions taken on an error 566
14.2	Dealing with errors that terminate execution	569
	The trap statement	570 ♦ The try/catch/finally statement 575
	The throw statement	578
14.3	Debugging with the host APIs	580
	Catching errors with strict mode	582 ♦ The Set-StrictMode cmdlet in PowerShell v2 584 ♦ Static analysis of scripts 589
14.4	Capturing session output	593
	Starting the transcript	593 ♦ What gets captured in the transcript 595
14.5	PowerShell and the event log	597
	The EventLog cmdlets	597 ♦ Examining the PowerShell event log 603
14.6	Summary	605
15	<i>The PowerShell ISE and debugger</i>	606
15.1	The PowerShell ISE	607
	Controlling the ISE pane layout	607 ♦ Using the ISE editor 610 ♦ Executing commands in the ISE 614
	Considerations when running scripts in the ISE	616
15.2	Using multiple PowerShell tabs	618
	Local in-memory session tabs	619 ♦ Remote session tabs in PowerShell ISE 619

- 15.3 Extending the ISE 622
 - The \$psISE variable 622 ♦ Using the Options property 624
 - Managing tabs and files 625 ♦ Working with text panes 629
 - Adding a custom menu 633
- 15.4 PowerShell script debugging features 638
 - The Set-PSDebug cmdlet 638 ♦ Nested prompts and the Suspend operation 643
- 15.5 The PowerShell v2 debugger 647
 - The graphical debugger 648
- 15.6 Command-line debugging 652
 - Working with breakpoint objects 653 ♦ Setting breakpoints on commands 656 ♦ Setting breakpoints on variable assignment 657 ♦ Debugger limitations and issues 658
- 15.7 Summary 659

Part 2 Using PowerShell 661

16 Working with files, text, and XML 663

- 16.1 PowerShell and paths 664
 - Providers and the core cmdlets 664 ♦ Working with PSDrives 665 ♦ Working with paths that contain wildcards 667 ♦ Suppressing wildcard processing in paths 668 ♦ The -LiteralPath parameter 670
 - The Registry provider 671
- 16.2 File processing 672
 - Reading and writing files 674 ♦ Writing files 679 ♦ All together now—reading and writing 680 ♦ Performance caveats with Get-Content 680
- 16.3 Processing unstructured text 681
 - Using System.String to work with text 681 ♦ Using hashtables to count unique words 684 ♦ Using regular expressions to manipulate text 686 ♦ Searching files with the Select-String cmdlet 688
- 16.4 XML structured text processing 693
 - Using XML as objects 693 ♦ Adding elements to an XML object 695 ♦ Loading and saving XML files 697
 - Processing XML documents in a pipeline 701 ♦ Processing XML with XPath 702 ♦ A hint of XQuery 709 ♦ Rendering objects as XML 711
- 16.5 Summary 717

17	<i>Extending your reach with .NET</i>	719
17.1	Using .NET from PowerShell	720
	.NET basics	720 ♦ Working with assemblies 721 ♦ Finding types 725 ♦ Creating instances of types 727 ♦ Defining new types with Add-Type 729 ♦ Working with generic types 739
17.2	PowerShell and the internet	740
	Retrieving a web page	740 ♦ Processing an RSS feed 742
17.3	PowerShell and graphical user interfaces	743
	PowerShell and WinForms	744 ♦ Creating a winforms module 750
	PowerShell and Windows Presentation Foundation	753
17.4	Summary	759
18	<i>Working with COM</i>	760
18.1	Working with COM in PowerShell	761
	Creating COM objects	761 ♦ Identifying and locating COM classes 762
18.2	Automating Windows with COM	764
	Exploring with the Shell.Application class	765 ♦ Managing browser windows using COM 767 ♦ A browser window management module 770
18.3	Working with the WScript.Shell class	777
18.4	Using COM to manage applications	779
	Looking up a word using Internet Explorer	779 ♦ Using Microsoft Word to do spell checking 781
18.5	The WSH ScriptControl class	783
	Embedding VBScript code in a PowerShell script	784
	Embedding JScript code in a PowerShell script	785
18.6	Working with the Windows Task Scheduler	786
	Getting started with the Schedule.Service class	786 ♦ Listing running tasks 787 ♦ Creating a new scheduled task 788
	Credentials and scheduled tasks	789 ♦ Viewing the life cycle of a task 792
18.7	Issues with COM	793
	64-bit vs. 32-bit issues	793 ♦ Threading model problems 793
	Interop assemblies, wrappers, and typelibs	793
18.8	Summary	795
19	<i>Management objects: WMI and WS-MAN</i>	797
19.1	Working with WMI in PowerShell	798
	Exploring WMI	798 ♦ The WMI infrastructure 799

19.2	The WMI cmdlets	801
	The WMI cmdlet common parameters	802 ♦ The Get-WmiObject cmdlet 804 ♦ The Set-WmiInstance cmdlet 813 ♦ The Invoke-WmiMethod cmdlet 819 ♦ The Remove-WmiObject cmdlet 822
19.3	The WMI object adapter	824
	The WMI type accelerators	825 ♦ Putting modified WMI objects back 828
19.4	Exploring WS-Man	830
	The WS-Man cmdlets	831 ♦ Using Get-WSManInstance to retrieve management data 832 ♦ Updating resources using Set-WSManInstance 840 ♦ Invoking methods with Invoke-WSManAction 841
19.5	Summary	845
20	<i>Responding in real time with eventing</i>	847
20.1	Foundations of event handling	848
20.2	Synchronous events	849
	Synchronous eventing in GUIs	850 ♦ Delegates and delegation 850
20.3	Asynchronous events	853
	Subscriptions, registrations, and actions	854 ♦ The eventing cmdlets 854
20.4	Working with asynchronous .NET events	855
	Writing a timer event handler	856 ♦ Managing event subscriptions 859
20.5	Asynchronous event handling with scriptblocks	860
	Automatic variables in the event handler	860 ♦ Dynamic modules and event handler state 862
20.6	Queued events and the Wait-Event cmdlet	863
20.7	Working with WMI events	866
	WMI event basics	866 ♦ Class-based WMI event registration 867 ♦ Query-based WMI event registrations 871
20.8	Engine events	875
	Predefined engine events	875 ♦ Generating events in functions and scripts 876
20.9	Remoting and event forwarding	877
	Handling remote EventLog events	879 ♦ Serialization issues with remote events 880
20.10	How eventing works	882

20.11	Summary	885
21	<i>Security, security, security</i>	888
21.1	Introduction to security	889
	What security is and what it isn't	889 ♦ Security: perception and reality 890
21.2	Security modeling	891
	Introduction to threat modeling	891 ♦ Classifying threats using the STRIDE model 892 ♦ Security basics: threats, assets, and mitigations 893
21.3	Securing the PowerShell environment	897
	Secure by default	897 ♦ Enabling scripting with execution policy 898
21.4	Signing scripts	904
	How public key encryption and one-way hashing work	904
	Signing authorities and certificates	905 ♦ Self-signed certificates 905 ♦ Using a certificate to sign a script 909
	Enabling strong private key protection	913 ♦ Using the PFX file to sign a file 915
21.5	Writing secure scripts	916
21.6	Using the SecureString class	916
	Creating a SecureString object	917 ♦ The SecureString cmdlets 918 ♦ Working with credentials 919 ♦ Avoiding Invoke-Expression 923
21.7	Summary	926

index 929

appendix A Comparing PowerShell to other languages

appendix B Examples

appendix C PowerShell Quick Reference

appendix D Additional PowerShell topics

Appendixes are available for download from
www.manning.com/WindowsPowerShellinActionSecondEdition