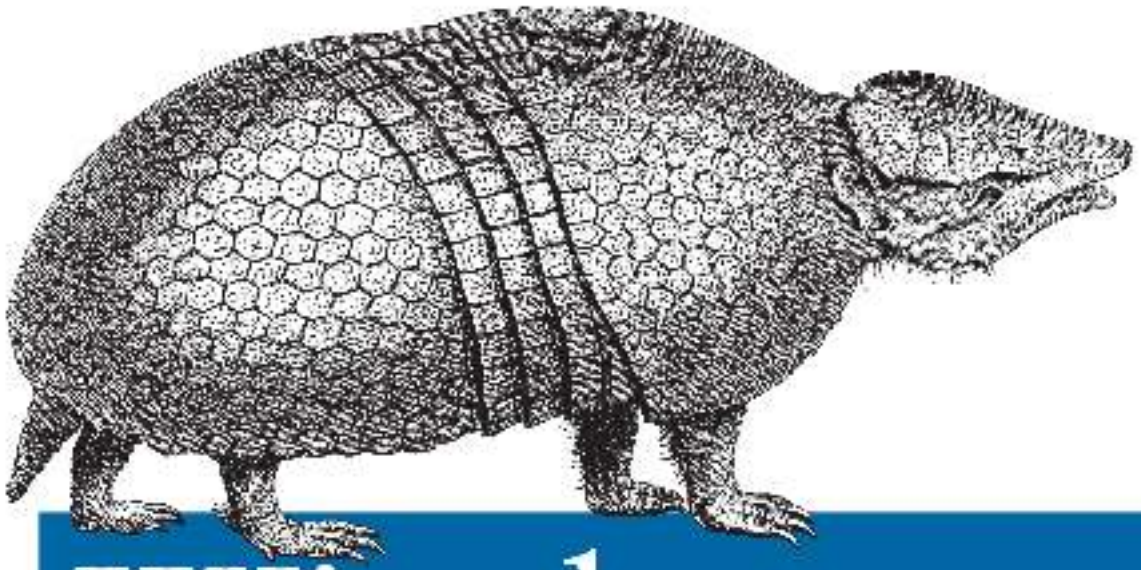
Enhance Your Productivity and Enable Rapid Application Development



Windows PowerShell

for Developers

O'REILLY®

Douglas Finke

Windows PowerShell for Developers

Want to perform programming tasks better, faster, simpler, and make them repeatable? Take a deep dive into Windows PowerShell and discover what this distributed automation platform can do. Whether you're a .NET developer or IT pro, this concise guide will show you how PowerShell's scripting language can help you be more productive with everyday tasks.

Quickly learn how to create PowerShell scripts and embed them into your existing applications, write "little languages" to solve specific problems, and take charge of your code. This book includes example scripts that you can easily pull apart, tweak, and then use in your own PowerShell and .NET solutions.

- Slice and dice text, XML, CSV, and JSON with ease
- Embed PowerShell to provide scripting capabilities for your C# apps
- Create GUI applications five to ten times faster with less code
- Leverage PowerShell's capabilities to work with the Internet
- Interact with DLLs and create objects, automatically display properties, and call methods in live interactive sessions
- Build domain-specific languages (DSLs) and vocabularies to express solutions more clearly
- Work with Microsoft Office via the Component Object Model (COM)
- Discover PowerShell v3 features included with Windows 8 and Windows Server 2012

US $24.99 CAN $26.99
ISBN: 978-1-449-32270-0

Twitter: @oreillymedia
facebook.com/oreilly

O'REILLY®
oreilly.com

Windows Powershell for Developers

Douglas Finke

O'REILLY®
Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

Windows Powershell for Developers
by Douglas Finke

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

Editor: Rachel Roumeliotis
Production Editor: Iris Febres
Proofreader: Iris Febres
Cover Designer: Karen Montgomery
Interior Designer: David Futato
Illustrator: Robert Romano

Revision History for the First Edition:
2012-07-03 First release
See *http://oreilly.com/catalog/errata.csp?isbn=9781449322700* for release details.

ISBN: 978-1-449-32270-0

[LSI]

1341351046

To my daughter, Elizabeth, with love

Table of Contents

Preface

Windows PowerShell is a successful, compelling, and integrated tool that all good .NET developers, IT pros, and anyone working with Windows should have in their toolboxes.

It can be used for making unit tests more powerful, scripting tasks such as reading XML or data imports, providing integration points in your .NET applications for end users to customize or extend using their own scripts, and defining little languages to express readable and concise business rules.

PowerShell simplifies your life, opening doors not previously accessible to you, by providing a .NET-based scripting language filled with useful features and *application programming interfaces* (APIs) for all the common programming tasks you take on daily.

You'll quickly learn the basic concepts using the interactive command line, and you'll move rapidly to creating scripts and embedding PowerShell into your existing .NET applications.

Audience

This book is for anyone who wants to know more about PowerShell. If you're serious about PowerShell, it's a must read. This book walks you through what is possible with PowerShell—helping you answer questions such as "can this be done better, faster, or simpler, or can I make it repeatable?"—and planting the seeds for you to creatively apply this new distributed automation platform on your own.

Assumptions This Book Makes

This book is not a beginner's guide to PowerShell. If you are an experienced developer or IT pro, this book gives you insight into what PowerShell can do.

The examples in this book are runnable out of the box. You can study how and what the scripts do—this is one of the tried-and-true ways of learning a new paradigm. While some examples include C# .NET, it is not required that you understand C#.

The examples are self-contained. Run them; see what they do. Then you can pull them apart, tweak them, and incorporate them into your PowerShell and .NET solutions.

Contents of This Book

Chapter 1 gives an overview of the platform and answers the question "Why PowerShell?"

Chapter 2 steps you through getting PowerShell prepped for running.

Chapter 3 offers a walkthrough of things you probably didn't even know the PowerShell platform could do.

Chapter 4 covers writing a template engine and using the new PowerShell v3 abstract syntax tree interface to extract information from PowerShell scripts.

Chapter 5 kicks it up a notch and shows you how easy it is to provide scripting abilities for your C# (WPF) apps by embedding PowerShell into them.

Chapter 6 demonstrates PowerShell's excellent capabilities for working with the Internet. JSON, XML, HTTP, Twitter? No problem.

Chapter 7 demonstrates how PowerShell is based on .NET. Want to build GUIs with less code? This is the chapter for you.

Chapter 8 further explores PowerShell's relationship to .NET and shows you how to leverage this seamless integration with other Microsoft frameworks.

Chapter 9 covers one of my favorite topics—building "little languages"—and shows how PowerShell makes this easy. Whether you prefer domain-specific languages (DSL) or domain-specific vocabularies (DSV), you'll want to check out what PowerShell has to offer.

Chapter 10 shows you how to really leverage applications like Microsoft Excel and by extension, Microsoft COM (Component Object Model) applications.

Chapter 11 is an excursion through some of the new and exciting features of PowerShell v3, set to ship with Windows 8 and Windows Server 2012, and available in beta for Windows 7.

Appendix A is all about programmer productivity. This is a PowerShell sweet spot, and this chapter shows you how to get the most out of the platform.

Appendix B shows you how to enable PowerShell v2 to load and work with .NET 4.0 DLLs. This is the default mode in PowerShell v3.

Conventions Used in This Book

The following typographical conventions are used in this book:

Plain text
: Indicates menu titles, menu options, menu buttons, and keyboard accelerators (such as Alt and Ctrl).

Italic
: Indicates new terms, URLs, email addresses, filenames, file extensions, pathnames, directories, and Unix utilities.

`Constant width`
: Indicates commands, options, switches, variables, attributes, keys, functions, types, classes, namespaces, methods, modules, properties, parameters, values, objects, events, event handlers, XML tags, HTML tags, macros, the contents of files, or the output from commands.

`Constant width bold`
: Shows commands or other text that should be typed literally by the user.

`Constant width italic`
: Shows text that should be replaced with user-supplied values.

This icon signifies a tip, suggestion, or general note.

This icon indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*PowerShell for Developers* by Douglas Finke (O'Reilly). Copyright 2012 Douglas Finke, 978-1-4493-2270-0."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

Available for Download

The code examples in the following chapters are available for download from GitHub at *https://github.com/dfinke/powershell-for-developers*.

We'd Like to Hear from You

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
(800) 998-9938 (in the United States or Canada)
(707) 829-0515 (international or local)
(707) 829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://oreilly.com/catalog/9781449322700

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, courses, conferences, and news, see our website at *http://www.oreilly.com*.

Find us on Facebook: *http://facebook.com/oreilly*

Follow us on Twitter: *http://twitter.com/oreillymedia*

Watch us on YouTube: *http://www.youtube.com/oreillymedia*

Safari® Books Online

Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at *http://my.safaribooksonline.com*.

Acknowledgments

Writing a book is an interesting journey. Now that it's completed, looking back over the last several months I'm amazed at how lucky I've been to come in contact with so many terrific people.

I'd like to thank my editor at O'Reilly, Rachel Roumeliotis, who was absolutely amazing to work with.

Thank you to Elizabeth, my daughter, who has just finished another year at university and continues to be my inspiration.

I was fortunate to have three great guys as reviewers for my book. They spent countless hours providing feedback and examples, researching specific content, offering lots of encouragement, and engaging with me in great discussions about PowerShell.

A special thanks to **Daniel Moore**. His passion for computing has earned him the nickname Beaver (as in "eager beaver"). He jumps in deep-end first and starts building dams like nobody's business. He's responsible for the WPF GUI in Chapter 5, a.k.a. the "Beaver Music application." He helped save me tons of time prepping the code for NuGet and the other examples for GitHub. Thanks, Daniel!

Thank you very much, **Aleksandar Nikolic´** and **Steve Murawski**, fellow PowerShell MVPs and cofounders of *PowerShell Magazine* (*http://www.powershellmagazine.com/*).

Aleksandar's incredible attention to detail was a significant asset in helping to finalize the book. He has a passion for PowerShell and is extremely generous with the time that he spends with the PowerShell community. Catch him at the next PowerShell Deep Dive.

Steve's depth of knowledge on PowerShell let him plow through these chapters and provide great feedback throughout the process.

When Steve signed on to review the book, his family was about to increase by one. He reviewed the chapters, did speaking gigs (including PowerShell Deep Dive), went to his day job, and took care of a newborn. Makes me tired just writing about it.

Gentlemen, it was an honor and privilege working with you.

And Now, the Small Village of Folks Who Helped, Inspired, and Supported Me

Allyson Chisholm—you have my heart.

Sal Mangano—fellow author, how you wrote a 1,000-page book is beyond me.

Thank you to this gang, a group of smart, supportive people whom I continue to learn from: Jeffrey Snover, James Brundage, Bruce Payette, Lee Holmes, Jason Shirk, Ed Wilson, Davin Tanabe, Jason Dolinger, Ravikanth Chaganti, Shay Levy, Ajay Kalras, Joel Bennett, Oisín Grehan, Keith Hill, Karl Prosser, Will Steele, Justin Rich, Lance Arlaus, Peter Coates, Ronald Lindtag, Sivabalan Muthukumar, Bailey Ling, Josh Einstein, and last but not least, Caleb and Ebony Finke (the furry ones).

CHAPTER 1
Introduction

There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of a new order of things. Because the innovator has for enemies all those who have done well under the old conditions and lukewarm defenders in those who may do well under the new.

—Nicolo Machiavelli, *The Prince*

PowerShell is the next-generation platform for distributed automation in the Microsoft Windows environment. It provides significant benefits to developers, testers, power users, and administrators. PowerShell works by leveraging the .NET Framework, and provides significant benefits to developers, testers, power users, and administrators. PowerShell leverages .NET to provide a powerful, consistent, intuitive, extensible, and useful set of tools that drive down costs, and make it easier to program for and automate Windows.

PowerShell was developed in 2002 under the code name *Monad*. In 2006, Microsoft published Release Candidate 1 of the platform, simultaneously announcing its new name, Windows PowerShell. Today PowerShell v3 is being delivered with Windows 8 and Windows Server 2012 and is available for Windows 7.

For a slightly reworked version of inventor Jeffrey Snover's opening to the *Monad Manifesto* whitepaper, which outlined the core ideas behind what would eventually become PowerShell, see http://bit.ly/n68k1X.

New PowerShell developers can often create timesaving scripts after just a few hours of learning. There are numerous accounts of people seeing huge reductions in time spent solving problems using PowerShell, compared to traditional system programming languages.

Another distinguishing feature of PowerShell is the fact that you can embed it into .NET applications. Adding the PowerShell scripting engine to a Windows .NET application allows you to provide a full-featured configuration and macro language to that application. This is roughly analogous to adding Visual Basic for Applications (VBA) to automate your work in Microsoft Excel.

This Is Just the Beginning

Once you learn PowerShell, you'll be able to write scripts for any PowerShell-enabled system. Windows Server 2012 is shipping with over 2,300 *cmdlets* (the basic unit of PowerShell functionality), up from the 400 cmdlets that shipped with Windows 2008 R2.

On top of this, the number of PowerShell solutions provided from third parties and the user community is growing by leaps and bounds. To get an idea of what PowerShell's future might hold, check out the sidebar "PowerShell Score Card, Ten Years On" to see what it has accomplished already in its 10-year history.

PowerShell Score Card, Ten Years On

- PowerShell v3 will ship with both Windows 8 and Windows Server 2012. There will be a version available for older Windows platforms.
- PowerShell v1 shipped in 2006.
- PowerShell v2 in 2009.
- WS2012 ships with over 2,300 cmdlets ready to use.
- PowerShell is integrated with SQL Server, IIS, Hyper-V, Microsoft Exchange, SharePoint, Server Manager, and much more.
- PowerShell is a *stop ship event*, meaning no Microsoft server product ships if it does not have a PowerShell interface.
- PowerShell supports running background jobs, running PowerShell scripts remotely, workflows, and much more.
- PowerShell is integrated with third-party companies like VMWare, Intel, Cisco, Citrix, Red Hat, and NetApp, among others.
- PowerShell has third-party tools like IDEs, Quest Software, Devfarm Software, Software/FX, and many more.
- The PowerShell console window runs in the browser, a.k.a. PowerShell Web Access (PWA).
- PowerShell has a thriving community with over 50 PowerShell MVPs (most valuable professionals), bloggers, podcasts, script repositories, active forums, and much more.

Why Use PowerShell

I use PowerShell for a number of reasons. It makes me fast, it's easy to use, and it's comprehensive. While PowerShell will never win a race with compiled .NET code, it's fast enough.

You can include .NET code directly in a PowerShell script and compile it on the fly.

PowerShell is an astonishing *glue language* because it is rooted in .NET. The .NET Framework, and the applications built on it, provides a set of powerful components that PowerShell can connect together. This includes the .NET applications I am building today. PowerShell pipes objects—not text—across the pipeline, enabling programming scenarios, in few lines of code, that were not possible before.

PowerShell is easy to learn and extremely powerful. It has all the elements you'd expect in a systems language—variables, loops, data structures, file I/O—and more. In addition, it has complete access to the .NET Framework, and the ability to seamlessly load .NET DLLs, instantiate objects, and retrieve metadata—either on your local box or via PowerShell remoting.

Finally, PowerShell is fun, satisfying, and rewarding to use. Whether you're using it to automate a tedious task, to simplify an implementation complicated by traditional means, or to create GUIs (WPF or WinForms based), PowerShell reduces both the effort and time you spend to get to a completed program.

There's a New Game in Town

Think of PowerShell as a new pinball game. We can continue to play the old one—we know how to jiggle the machine just right so as not to tilt it, we understand all the ins and outs, and we know the tricks to get extra plays—but this new game has great potential.

But there is a wrinkle here: in order to get good at PowerShell, you need to experience a short, frustrating period of being bad at it (i.e., the valley of the *s*-curve shown in Figure 1-1). That means you'll be looking things up, wrapping your head around new ideas, and getting comfortable with the fact that when you jiggle PowerShell, sometimes it's going to tilt.

Usually you see declines in performance before significant improvements.

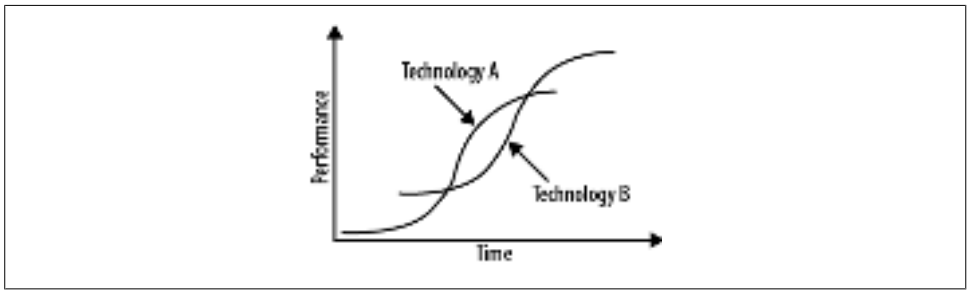


Figure 1-1. S-curve of innovation

An Underutilized Development Tool

Scripted versions of applications require less code, effort, and development time compared to traditional approaches. The interesting discussion is not static languages versus dynamic languages, but rather when and where to use both for delivering solutions. As John Ousterhout, creator of Tcl/Tk, put it:

> Scripting languages are higher level than system programming languages in the sense that a single statement does more work on average. A typical statement in a scripting language executes hundreds or thousands of machine instructions, whereas a typical statement in a system programming language executes about five machine instructions.

In summary, you owe it to yourself to try out this new pinball machine.

CHAPTER 2

Getting Started

Installing PowerShell

Installing PowerShell is as simple as installing any other application. Even better, it comes preinstalled with Windows 7, Windows Server 2008 R2, Windows 8, and Windows Server 2012. PowerShell is also available for previous versions of Windows XP, 2003, and Vista.

As noted, PowerShell v3 comes preinstalled with Windows 8 (and as I am writing this, there is a RC release for Windows 7; you can download it at *http://bit.ly/MdfXPo*). New cmdlets and language features are abundant in this more robust version, all designed to make you more productive and lower the barrier of entry to using PowerShell.

If you are running an older Microsoft Windows OS, I encourage you to update that; however, PowerShell v2 can run on these boxes. You can get v2 at *http://bit.ly/2QfKYT*; make sure to download the right PowerShell for your OS and architecture.

While there is no PowerShell version for UNIX, Linux, or Mac, Microsoft did license the PowerShell language under the Community Promise (*http://bit.ly/dLIHJ8*). We'll see if any developers pick up from here and implement PowerShell on non-Windows boxes.

Checking the PowerShell Version

Depending on your Windows OS, you can navigate to PowerShell in many ways. First, get to the command prompt and type:

```
PS C:\> $PSVersionTable

Name                           Value
----                           -----
WSManStackVersion              3.0
PSCompatibleVersions           {1.0, 2.0, 3.0}
SerializationVersion           1.1.0.1
BuildVersion                   6.2.8158.0
```

```
PSVersion                      3.0
CLRVersion                     4.0.30319.239
PSRemotingProtocolVersion      2.103
```

This gives you lots of good information about the PowerShell version running on your box—including what version of .NET you are going against, noted as CLRVersion in PowerShell. I'm running PowerShell v3 CTP3. I can run PowerShell in version 2 mode; if possible, you should too.

Here is what I get when I look at the `$PSVersionTable` variable. Notice I have only two compatible versions and am using .NET 2.0, CLRVersion. When PowerShell v2 was delivered, only .NET 2.0 was released. PowerShell v3 works with .NET Framework 4.0.

```
PS C:\> $PSVersionTable

Name                           Value
----                           -----
CLRVersion                     2.0.50727.5448
BuildVersion                   6.1.7601.17514
PSVersion                      2.0
WSManStackVersion              2.0
PSCompatibleVersions           {1.0, 2.0}
SerializationVersion           1.1.0.1
PSRemotingProtocolVersion      2.1
```

Interactivity, the Key to PowerShell

The prompt is up, so let's work the PowerShell REPL. A *REPL* (pronounced "repple") is a read-eval-print loop. This means that when you type some PowerShell command and press Enter, those commands are read and evaluated, results (or errors) are printed, and the console loops back and waits to do it again. Let's try it:

```
PS C:\> 2 + 2 * 3
8
PS C:\>
```

So, PowerShell is just a big calculator? Not exactly. If you try that example in a DOS prompt, what happens? You get an error. Here, the result is printed and we get the prompt back, ready to complete your next request.

Now type in the `"Hello World"` quoted string. Press Enter, and you get back the same thing you typed, without the quotes. PowerShell *evaluated* that for you, demonstrating the *E* in REPL. Also, we didn't have to explicitly specify that we wanted it to be printed; PowerShell just "knew" to do that. These are great timesaving aspects of PowerShell—not to mention, they cut down on keystrokes too.

```
PS C:\> "Hello World"
Hello World
```

Let's tap into the .NET Framework now. Type in:

```
PS C:\> [System.Math]::Pow(2, 3)
8
```