# brief contents

## PART 1  **POWERShell Administration**

| 1 | Diagnosing and troubleshooting PowerShell remoting | 3 |
| 2 | CIM sessions | 22 |
| 3 | Collecting and analyzing performance counter data | 31 |
| 4 | TCP port communications with PowerShell | 51 |
| 5 | Managing systems through a keyhole | 71 |
| 6 | Using PowerShell to audit user logon events | 80 |
| 7 | Managing and administering a certification authority database with PowerShell | 93 |
| 8 | Using PowerShell to reduce Active Directory token bloat | 115 |

## PART 2  **POWERShell Scripting**

| 9 | The 10 PowerShell scripting commandments | 129 |
| 10 | Avoiding the pipeline | 150 |
| 11 | A template for handling and reporting errors | 161 |
| 12 | Tips and tricks for creating complex or advanced HTML reports with PowerShell | 171 |
| 13 | Using and “abusing” dynamic parameters | 192 |
| 14 | PowerShell type formatting | 207 |
| 15 | Scalable scripting for large data sets: pipeline and database techniques | 221 |
16  ■  Building your own WMI-based cmdlets  236
17  ■  Turning command-line tools into PowerShell tools  247

PART 3  PowerShell for Developers .................................259
18  ■  Using Source Control Software with PowerShell   261
19  ■  Inline .NET code  270
20  ■  PowerShell and XML: better together  278
21  ■  Adding automatic remoting to advanced functions and cmdlets 306
22  ■  Taming software builds (and other complicated processes) with psake  326

PART 4  PowerShell Platforms .................................341
23  ■  PowerShell and the SQL Server provider  343
24  ■  Creating flexible subscriptions in SSRS  354
25  ■  Inventory database table statistics using PowerShell and SQL Server Management Objects 368
26  ■  WSUS and PowerShell  379
27  ■  Provisioning IIS web servers and sites with PowerShell  392
28  ■  Active Directory Group Management application  406
The following scenario is common if you’re an administrative web master, and here’s how it was delivered to me: “Deploy a highly available web farm (four servers) with a couple of websites, including certificates, for a new secure shopping site. Make sure to enable graphical remote management for IIS Manager so that other admins and developers can make changes; and, by the way, did we mention we’re moving to Windows Server 2012 Core?” (See figure 1.)

This isn’t a complicated project, thanks to the support of PowerShell and the Internet Information Services (IIS) cmdlets, but you may encounter tricky spots and gotchas along the way.

Initially I solved this problem by using PowerShell interactively to complete the required tasks. As a smart and lazy admin, I saved the commands to a script so that in the future I could automate similar deployments without all the typing. I even turned some of the tasks into advanced functions so that other admins could accomplish some of the trickier stuff.

In this chapter you’ll see how I interactively solved this deployment scenario, and I’ll also show you how to automate it. The entire process from beginning to end involves these tasks:

- Deploy IIS to the Windows Server 2012 Core remote servers.
- Prepare the remote servers with website files and certificates.
- Enable remote-management support for the graphical IIS Manager.
- Create a load-balanced web farm.
- Create a secure load-balanced website using Secure Sockets Layer (SSL).
- Automate the process.

Let’s get started and deploy the web servers and websites.
Setting up the lab environment

I created a lab environment to write this chapter. If you want to follow along you can create a similar environment.

Although I’m using Windows Server 2012 Core, this deployment solution also works on Windows Server 2008 R2, with or without a graphical desktop. I use some of the newer networking commands from Server 2012 for the Domain Name System (DNS) settings, but if you’re using Windows Server 2008 R2 you can work around that with the GUI. I also use the dynamic module-loading feature in PowerShell v3; if you’re using PowerShell v2 I’ll warn you when you need to import a module.

These are the items that I set up in advance:

**Deployment station**—Windows 8 Pro running Remote Server Administration Tools (RSAT). I’ll use local RSAT cmdlets in this chapter.

**Four Windows Server 2012 Core servers**—Each server is assigned an IP address and is a member of the domain, although this is not required for middle-tier web servers. You can set up the IP address through `SConfig.cmd` or the networking cmdlets.

**Remoting**—This feature is enabled for all Windows Server 2012 products; you’ll need to enable it if you’re using Windows Server 2008 R2. (This is a requirement.)
CHAPTER 27  Provisioning IIS web servers and sites with PowerShell

Rapid IIS deployment

To begin the deployment we’ll use PowerShell Remoting to connect to the remote servers. Some tasks won’t be completed over remoting, so store a list of computer names in a variable that you can pipe to commands.

1  Gather the computer names of the future web servers and store them to a variable, $Servers, using one of the following options.

   If the servers are members of the domain, use the Active Directory cmdlet Get-ADComputer:

   PS> $Servers = Get-ADComputer -Filter "name -like 's*'" | Select-Object -ExpandProperty name

   **NOTE** If you’re using PowerShell v2, be sure to import the Active Directory module first.

   You can also get the list from a CSV or TXT file:

   PS> $Servers = Import-Csv c:\servers.csv | Select-Object -ExpandProperty ComputerName
   PS> $Servers = Get-Content c:\servers.txt

2  Create a PowerShell remote session to the servers. Store the sessions in a variable $Sessions for easy access later:

   PS> $Sessions = New-PSSession -ComputerName $Servers

3  Determine what software is needed to support all of the tasks for this project.

   The remote servers require the following roles and features for this deployment solution, but you can add to the list if you need additional components to support your websites:

   - **Web Server (IIS)** *(web-server)*—The primary role for a web server. This installs the components of IIS and creates the default website.
   - **ASP.NET** *(web-asp-net)*—Provides support for ASP.NET websites.
   - **Network Load Balancing (NLB)**—I’m using Microsoft’s built-in layer-3 NLB software. You can substitute your own hardware load balancer or Microsoft’s layer-7 Application Request Routing (ARR) balancer. ARR has cmdlets for easy management and is one of my favorite products. ARR also includes additional features beyond load balancing but requires greater in-depth

**Script execution**—This should be enabled on the servers.

**SSL certificate**—For a lab environment you can use a self-signed certificate, but for production use a good web server certificate or even an Extended Validation (EV) certificate. I created a certificate in Active Directory Certificate Services (AD CS) and exported it to a Personal Information Exchange (.pfx) file.
knowledge, so I’m sticking with the straightforward, built-in, and useful Microsoft NLB.

- **Management Service (Web-Mgmt-Service)**—Required component for remote management of IIS with IIS Manager.

4 Install the required components on the remote servers with Invoke-Command:

   ```powershell
   PS> Invoke-Command -Session $Sessions {Install-WindowsFeature web-server,web-asp-net,NLB,Web-Mgmt-Service}
   ```

4 Installing the software components to all four servers, as shown in figure 2, takes only a few minutes (5 minutes to be exact).

   The IIS installation process creates the default website automatically. Let’s test this default website on each server before continuing with the next task.

   Testing ensures that the web server is functioning properly and reduces future troubleshooting if something goes wrong:

5 Use the $Servers variable to pipe the server names to Internet Explorer:

   ```powershell
   PS> $Servers | ForEach-Object {Start-Process iexplore http://$_}
   ```

   Four separate browsers automatically launch and test the default website on each individual server.

With the initial software deployment completed the next task is to deploy (copy) the website files and certificate out to the servers. PowerShell makes this a snap.

![Figure 2 Performing a rapid install of the required software on multiple servers](image-url)
Transferring website files and certificates

IIS supports storing your website files and applications on a central share from a clustered file server. Some organizations, such as small companies, don’t have this capability, so we’ll copy the websites from a central location (my computer) out to the individual web servers. Because these web servers will be load-balanced, each server needs to have the same files.

**DEPLOYING THE DEFAULT WEBSITE**

1. Copy the new default website to each web server’s c:\inetpub\wwwroot path:
   ```powershell
   PS> $Servers | ForEach-Object {Copy-Item -Path c:\sites\www\*.* -Destination "\$_\c$\inetpub\wwwroot"}
   ```

2. Test the default website after the file transfer (see figure 3):
   ```powershell
   PS> $Servers | ForEach-Object {Start-Process iexplore http://$_}
   ```

With the default website successfully deployed we can focus on the new secure shopping site.

**DEPLOYING THE SHOPPING WEBSITE**

Most of the websites that you’ll copy out to the web servers won’t be in the default path (InetPub). I prefer to use a directory called sites, with each website in its own folder:

1. Create the folder structure on the remote servers (C:\sites\shopping), and then copy the new website:
   ```powershell
   PS> Invoke-Command -Session $Sessions {New-Item -Path c:\sites\shopping -ItemType directory -Force}
   PS> $Servers | ForEach-Object {Copy-Item -Path c:\sites\shopping\*.* -Destination "\$_\c$\sites\shopping"}
   ```

2. Generate a certificate for SSL for the secure shopping site.
   (I previously generated and stored a trusted certificate on my local Windows 8 computer in c:\sites\certpfx.)

3. Copy the certificate to the remote servers, and then use CertUtil.exe to import the certificate:
Enabling remote management for IIS Manager

IIS remote management adds the capability of managing websites on remote servers from IIS Manager. It’s best to enable and configure this feature using IIS Manager run locally on each server; it’s not a friendly feature to enable through the command line or on Windows Server 2012 Core. In addition, we need to replace the temporary, self-signed certificate, which is assigned to remote management.

Let’s break this into two steps: enabling the service and replacing the certificate.

**ENABLING THE SERVICE**

1. Enable the remote management service in the registry, and then start the Web Management Service (WMSVC).

   WMSVC has a startup type of Manual, so change the startup to Automatic before starting the service:

   ```powershell
   PS> Invoke-Command -Session $Sessions {Set-ItemProperty
   ➤ –Path HKLM:\SOFTWARE\Microsoft\WebManagement\Server
   ➤ –Name EnableRemoteManagement -Value 1}
   PS> Invoke-Command -Session $Sessions {Set-Service wmsvc
   ➤ -StartupType Automatic}
   PS> Invoke-Command -Session $Sessions {Start-Service wmsvc}
   ``

   Figure 5 illustrates the successful start of WMSVC on the remote computers.

   **Figure 4 Deploying and installing a certificate for SSL**

   ```powershell
   PS> $Servers | ForEach-Object {Copy-Item -Path c:\sites\certpfx\*.*
   ➤ -Destination "\$_\c$"}
   PS> Invoke-Command -Session $Sessions {certutil -p P@ssw0rd
   ➤ -importpfx c:\company.loc.pfx}
   ```

   I sent the password in clear text because PowerShell Remoting is secure and encrypted. I wouldn’t do this in a script. The certificate imports successfully, as shown in figure 4.

   4. Remove (delete) the .pfx file from the remote servers:

   ```powershell
   PS> $Servers | ForEach-Object {Remove-Item -Path "\$_\c$\company.loc.pfx"}
   ```

   The website files are copied to the remote servers and each server has the certificate for the secure site. Before you finish creating and configuring the secure site you need to enable IIS remote management so that the websites can be managed using IIS Manager.
At this point you can connect IIS Manager to the remote computers, but you can’t use IIS Manager to manage and change the certificates for the remote service.

**REPLACING THE CERTIFICATE**

The IIS remote management service uses port 8172 and binds a temporary certificate to “all unassigned” IP addresses. You need to change this binding, and this is where things get a little strange. To remove the old SSL binding for port 8172 and add a new one you need to access the IIS: provider. Because PowerShell cmdlets and this provider don’t work together as well as they could, extra steps are required to complete the process:

1. Get the thumbprint of the trusted certificate that you imported previously and store it to a variable ($cert).
   
   Perform this step over PowerShell Remoting so that the variable can be used for later commands:
   
   ```powershell
   PS> Invoke-Command -Session $Sessions {Get-ChildItem -Path Cert:\LocalMachine\My | where {$_._subject -like '*company*'} | Select-Object -ExpandProperty Thumbprint}
   ```

2. Access the IIS: drive.
   
   When IIS is installed, a module called WebAdministration is added, which includes cmdlets and an IIS: provider. To ensure that the provider is loaded, import the WebAdministration module:
   
   ```powershell
   PS> Invoke-Command -Session $Sessions {Import-Module WebAdministration}
   ```

   ```powershell
   PS> Invoke-command -Session $Sessions {cd IIS:\SslBindings}
   ```

   Bindings are stored in IIS\SslBindings as path items.

3. Remove the binding that contains the temporary certificate:

   ```powershell
   PS> Invoke-command -Session $Sessions {Remove-Item -Path IIS:\SslBindings \0.0.0.0!8172}
   ```

**NOTE** Usually IIS binding information is entered and displayed as IPAddress:port:hostname, as in *:*80:*, but PowerShell interprets the colon (:) as a path indicator. When using the cmdlets to work with bindings for IIS, replace the colon with an exclamation mark (!), as in *!80!*.
Creating a load-balanced web farm

4. Create a new binding that uses the new trusted certificate.

Use the `Get-Item` command to retrieve the correct certificate based on the thumbprint stored in `$cert`. The certificate is piped to `New-Item`, which creates the new binding for all IP addresses on port 8172:

```
PS> Invoke-Command -Session $Sessions {Get-Item
   -Path "cert:\localmachine\my\$cert" |
   New-Item -Path IIS:\SslBindings\0.0.0.0!8172}
```

5. Start IIS Manager (PS> Start inetmgt), and create connections to the remote servers as shown in figure 6.

With the remote management capabilities of IIS enabled we can finish off our deployment and provisioning web server project with two final tasks: building the web farm and creating a new secure website. Let’s start with the web farm.

Creating a load-balanced web farm

For many companies a hardware load balancer that provides high availability is the only choice for their web farms; it’s fast, efficient, and provides certificate management. Not everyone can afford (or even needs) this level of performance, so other options are available. My favorite is the layer-7 load balancer for IIS from Microsoft called Application Request Routing (ARR). It’s free, an excellent product, can be downloaded from www.iis.net, has cmdlets for management, includes many more
features in addition to load balancing, and, did I mention, it’s free. ARR performs load balancing using URL rewrite. Because URL rewrite is complex and requires in-depth knowledge of ARR I chose to use the built-in Microsoft NLB for this example deployment situation. NLB works well and doesn’t require the additional installation and knowledge overhead to make a great solution.

For this task I’m using the cmdlets from the NLB module on my Windows 8 computer. Alternatively you could issue these commands over PowerShell Remoting:

1. Create the load balance on server S1 with the New-NlbCluster cmdlet, and create a cluster IP address for the default website:

   ```powershell
   PS> New-NlbCluster -HostName s1 -InterfaceName Ethernet -ClusterName web
   -ClusterPrimaryIP 192.168.3.200 -SubnetMask 255.255.255.0
   -OperationMode Multicast
   ```

2. Add another address with the Add-NlbClusterVip cmdlet:

   ```powershell
   PS> Get-NlbCluster -HostName s1 | Add-NlbClusterVip -IP 192.168.3.201
   -SubnetMask 255.255.255.0
   ```

   You’ll use this additional cluster IP address for the secure website that you’ll create in the next section.

3. Add the second server (S2) as a node in the load balance with the Get-NlbCluster cmdlet:

   ```powershell
   PS> Get-NlbCluster -HostName s1 | Add-NlbClusterNode -NewNodeName s2
   -NewNodeInterface Ethernet
   ```

4. Repeat step 3 for the other two servers in this scenario.

   The return information from the Get-NlbCluster cmdlet informs you if you have any problems converging the load balance.

5. Launch the graphical Network Load Balancing Manager (on a Windows 8 computer) from the Administrative Tools to verify the status (see figure 7).

---

**Figure 7** Verifying the load balance in the Network Load Balancing Manager
Test the load balance with full name resolution.

Create a www record in DNS that points to the cluster IP address, and then launch a browser using the new address:

```powershell
PS> Add-DnsServerResourceRecordA -Name www -ZoneName company.loc
   -IPv4Address 192.168.3.200 -ComputerName DC.company.loc
PS> Start-Process iexplore http://www.company.loc
```

Finally, after all this work, it's time for the final task: creating a new and secure website for the web farm. Let's make a website!

**Creating an SSL website**

To make a new website on the remote servers use the IIS (web) cmdlets from the WebAdministration module. Remember that we already copied the files for this new website to the location c:\sites\shopping:

1. Create an application pool for the new website with the New-WebAppPool cmdlet:

   ```powershell
   PS> Invoke-Command -Session $Sessions
   {New-WebAppPool -Name Shopping-Pool}
   ```

   Figure 8 shows the graphical version of creating a pool in IIS Manager.

   The new application pool is created with default settings for items such as the recycle times and identity. This is a good time to add your own application pool commands to alter those defaults, if desired. (See the sidebar for an example.)

**Changing the application pool identity**

Usually, for application pools of ApplicationPoolIdentity, the default identity is sufficient as a restricted identity. In cases where multiple customers have websites located on the same server (multitenant), isolating each pool with its own identity provides unique security for every customer. To set the pool identity IIS uses a number representing the identity. The default value is 4, but if you want to have isolation you can create individual accounts and assign those accounts to each pool as in the following example:

```powershell
LocalSystem = 0
LocalService = 1
NetworkService = 2
SpecificUser = 3
ApplicationPoolIdentity = 4
PS> Invoke-Command -Session $Sessions
{Set-ItemProperty
   -Path IIS:\AppPools\MyTest -Name processmodel.identityType -Value 3}
```
**CHAPTER 27  Provisioning IIS web servers and sites with PowerShell**

2  Create a new website named **Shopping**.
   After you create the application pool the `New-Website` cmdlet does the rest of the work:

   ```powershell
   PS> Invoke-Command -Session $Sessions {New-Website -Name Shopping
   -HostHeader shop.company.loc -PhysicalPath C:\sites\shopping
   -ApplicationPool Shopping-Pool -Port 443 -ssl -SslFlags 0}
   ```

   The website has a host header of `shop.company.loc` and points to the physical location of the website files. The new site is assigned to the correct application pool and a binding on port 443 is set. The `-SslFlags` tells the website to use a normal certificate.

3  Create another SSL binding for the new site.
   The process is the same as discussed previously, but the binding is for all IP addresses on port 443:

   ```powershell
   PS> Invoke-Command -Session $Sessions {$cert=Get-ChildItem
   -Path Cert:\LocalMachine\My | where {$_.subject -like "*company*"} | Select-Object -ExpandProperty Thumbprint}
   PS> Invoke-Command -Session $Sessions {Import-Module WebAdministration}
   PS> Invoke-Command -Session $Sessions {Get-Item
   -Path "cert:\localmachine\my\$cert" | New-Item -Path
   IIS:\SslBindings\0.0.0.0!443!Shop.company.loc}
   ```

   As shown in figure 9, the new binding is successfully created on all remote servers.

4  Test the new website.
   Add a DNS record that points to the cluster IP address previously defined for the website and then launch a browser using the address:

   ```powershell
   PS> Add-DnsServerResourceRecordA -Name shop -ZoneName company.loc
       -IPv4Address 192.168.3.201 -ComputerName DC.company.loc
   PS> Start-Process iexplore https://shop.company.loc
   ```

   ![Figure 9  Successful creation of the new SSL binding](image)
Automating the process

As shown in figure 10, the new website successfully passes the test using the trusted certificate over SSL.

Total time for this project, using PowerShell interactively, is approximately 30 minutes. Storing these commands in a .ps1 file helps me script future deployment projects. Why do all that typing again? I wrote the tricky tasks, such as enabling remote management, as advanced functions so that other admins have the tools they need without all the hassle. I increased my value to the company and managed to get a little more time on the beach.

Automating the process

Automating the deployment process is as simple as sticking the commands in a script file, but I went further and built in more flexibility. PowerShell Remoting and the Invoke-Command cmdlet make life easy. For example, have you ever tried the switch option for Invoke-Command -FilePath? This switch option eliminates the need to copy scripts to remote computers before executing them. You write a script that performs the tasks as if it were running on the local computer. To send that script to your remote computers use Invoke-Command.

In this section I’ll first show you the script that does the hard work, and then I’ll show you how I call and use the script. The only changes from the commands you’ve already seen are the following:

- I removed all of the Invoke-Command cmdlets.
- I changed how the certificate password is passed to the script. I don’t want the password hardcoded in the script, so I used a PowerShell v3 feature to pass a variable to the script with $Using:CertPassword.
- I left out the NLB commands, in case you already have a load-balance solution, but you can always add them.

Here’s the script, which I named Deploy-WebServer.ps1.
CHAPTER 27 Provisioning IIS web servers and sites with PowerShell

To use the Deploy-WebServer.ps1 script I run interactive commands to set up the remoting connections and set a few variables. Then I call the deployment script with a single Invoke-Command cmdlet:

1. Build a remote session to the computers that will become web servers.
   
   Put the server names in a variable—you’ll need that later, so don’t cheat and make this a one-liner:
   
   ```powershell
   PS> $Servers='server1','server2', 'server3'
   PS> $Sessions=New-PSSession -ComputerName $Servers
   ```

2. Set a variable to contain the password to install the certificate.
   
   This information is passed over the remoting session encrypted:
   
   ```powershell
   PS> $CertPassword="P@ssw0rd"
   ```

3. Interactively copy the website files and certificates to the remote servers:
   
   ```powershell
   PS> $servers | ForEach-Object{New-Item -Path \$_\C$\inetpub\wwwroot -ItemType Directory -Force}
   PS> $servers | ForEach-Object{Copy-Item -Path c:\sites\www\*.* -Destination \\$_\C$\inetpub\wwwroot -Force}
   PS> $servers | ForEach-Object{Copy-Item -Path c:\sites\CertPFX\*.* -Destination \\$_\C$ -Force}
   ```

   If you put these commands in the Deploy-WebServer.ps1 file you’ll run into a double-hop issue—the remote computers connecting to another remote server to get the files.

Listing 1 Deploy-WebServer.psl

```powershell
Install-WindowsFeature web-server,Web-Mgmt-Service

Set-ItemProperty -Path HKLM:\SOFTWARE\Microsoft\WebManagement\Server ` `-Name EnableRemoteManagement -Value 1

Set-Service wmsvc -StartupType Automatic
Start-Service wmsvc

certutil -p $Using:certPassword -importpfx c:\Wildcard.company.loc.pfx
Remove-Item -Path c:\Wildcard.company.loc.pfx

Import-module -Name WebAdministration
$cert = Get-ChildItem -Path Cert:\LocalMachine\My | where {$_.subject -like "*company*"}
Select-Object -ExpandProperty Thumbprint
Remove-Item -Path IIS:\SslBindings\0.0.0.0!8172

Get-Item -Path "cert:\localmachine\my\$cert" | New-Item -Path IIS:\SslBindings\0.0.0.0!8172
New-WebBinding -Name "Default Website" -Protocol https

Get-Item -Path "cert:\localmachine\my\$cert" | New-Item -Path IIS:\SslBindings\0.0.0.0!443

```

Installs required components

Enables remote management

Removes the certificate file

Creates new SSL binding

NOTE If you copy files to Windows Server 2012 Core you’ll first need to install the FS-FileServer role to access the C$ share.

4 Run the deployment script using the -FilePath parameter:

```
PS> Invoke-Command -Session $Sessions -FilePath C:\scripts\deploy-WebServer.ps1
```

All the target servers now have a web server, website, and certificates installed and are ready for action!

**Summary**

This chapter covered the deployment of multiple web servers with multiple websites, which included building a web farm and installing certificates for SSL. The concepts and tactics demonstrated here could easily be applied to other roles, features, and products, such as SharePoint web servers and Client Access Server (CAS) arrays for Microsoft Exchange. I gleaned the following takeaways during this real-life project:

- I can use PowerShell interactively to solve each task, even for a more complicated deployment.
- There may not be specific cmdlets for every situation, such as enabling the remote management of IIS, but there are ways around those issues.
- PowerShell Remoting must be enabled to permit these larger-scale management solutions. While it’s the default for Windows Server 2012, you need to enable it now even if you’re not at that version yet.

Thanks to PowerShell I get an amazing amount of work done quickly and without traveling to a cold data center. If you have any questions about the script or commands I discussed in this chapter visit the forums at http://www.powershell.org, and I’ll be happy to help!

**About the author**

Jason Helmick is a teacher, author, consultant, and 25-year IT veteran focusing on Microsoft enterprise technologies. Jason is a strong proponent of automation with PowerShell and is a board member of PowerShell.Org. He is the author of *Learn Windows IIS in a Month of Lunches* (Manning, 2013). You can follow and contact Jason on Twitter: @theJasonHelmick.
PowerShell has permanently changed Windows administration. This powerful scripting and automation tool allows you to control virtually every aspect of Windows and most Microsoft servers like IIS and SQL Server. Here's your chance to learn from the best in the business.

*PowerShell Deep Dives* is a trove of essential techniques and practical guidance. It is rich with insights from experts who won them through years of experience. The book’s 28 chapters, grouped in four parts (Administration, Scripting, Development, and Platforms), were hand-picked by four section editors: Jeffery Hicks, Richard Siddaway, Oisín Grehan, and Aleksandar Nikolić.

**What’s Inside**
- Managing systems through a keyhole
- The Ten Commandments of PowerShell scripting
- Scalable scripting for large datasets
- Adding automatic remoting
- Provisioning web servers and websites automatically to IIS 8
- And 23 more fantastic chapters

Whether you’re just getting started with PowerShell or you already use it daily, you’ll find yourself returning to this book over and over.

**The Authors**
Editors Jeffery Hicks, Richard Siddaway, Oisín Grehan, and Aleksandar Nikolić are joined by PowerShell experts Chris Bellée, Bartek Bielawski, Robert C. Cain, Jim Christopher, Adam Driscoll, Josh Gavant, Jason Helmick, Don Jones, Ashley McGlone, Jonathan Medd, Ben Miller, James O’Neill, Arnaud Petitjean, Vadims Podans, Karl Prosser, Boe Prox, Matthew Reynolds, Mike Robbins, Donabel Santos, Will Steele, Trevor Sullivan, and Jeff Wouters.

Manning Publications and the authors of this book support Save the Children at www.savethechildren.org.

To download their free eBook in PDF, ePub, and Kindle formats, owners of this book should visit manning.com/PowerShellDeepDives.