

Using Windows PowerShell to Manage SharePoint 2010

Kevin Laahs

(Reprinted From WindowsITPro Magazine)

The release of SharePoint 2010 (both SharePoint Foundation and SharePoint Server) offers welcome support for Windows PowerShell. Of course it has always been technically possible to use PowerShell to manage SharePoint but that involved doing a lot of work yourself to call the .NET Framework and work directly with the SharePoint Object Model or Web Services. But now it's official, and administrators get to use the power of PowerShell out of the box.

I'll show you the PowerShell implementation in SharePoint, the requirements for accessing it, and the possibilities it brings to the administration table. I assume that you have a general understanding of PowerShell 2.0 and its concepts.

Positioning STSADM and PowerShell

Before SharePoint 2010, the only command-line tool available to administer SharePoint was STSADM. STSADM still ships with SharePoint 2010 because some commands don't yet have direct PowerShell equivalents (e.g., enabling the developer dashboard), and you might still need it to support any management utilities you built that you want to continue to use against your 2010 deployment.

STSADM is itself an extensible command-line tool (182 commands were available out of the box with Microsoft Office SharePoint Server 2007—MOSS 2007) and has been put to good use in many deployments. Gary Lapointe (stsadm.blogspot.com) provides some great examples for not only extending STSADM but also building your own PowerShell cmdlets. That said, it's likely that PowerShell will ultimately be the sole command-line utility for management in the future, and STSADM will gracefully fade away as Microsoft focuses resources on a single tool that's used across multiple server applications.

PowerShell is more flexible than STSADM in its extension capabilities. It also performs far more efficiently, especially in the batch processing area, because, with STSADM, each operation has to physically invoke the STSADM executable.

Loading the SharePoint Cmdlets

The first step to using the SharePoint cmdlets is to fire up a shell and load the Microsoft.SharePoint.PowerShell snap-in. You can do this manually via the Add-PSSnapin cmdlets, but you don't actually have to, as the installation adds the SharePoint 2010 Management Shell shortcut, which does this for you, to the Start menu. If you choose to do it manually, the command behind the shortcut is

```
C:\Windows\System32\WindowsPowerShell\v1.0\
PowerShell.exe -NoExit " & ' C:\Program Files\
Common Files\ Microsoft Shared\Web Server
Extensions\14\CONFIG\ POWERSHELL\
Registration\sharepoint.ps1 ' "
```

This launches a shell, which then invokes the sharepoint.ps1 script. Inside that script, the cmdlet Add-PsSnapin Microsoft.SharePoint.PowerShell is executed.

You'll find 531 cmdlets in the snap-in that you can use for your administrative delights (SharePoint Foundation 2010 adds "only" 244, so this gives you a good feel for the added features that are in

Using Windows PowerShell to Manage SharePoint 2010

Kevin Laahs

(Reprinted From WindowsITPro Magazine)

SharePoint Server 2010). The number of cmdlets increases as you layer on other features such as Office Web Applications, FAST search, or PerformancePoint. To get a list of all the SharePoint cmdlets, simply execute the following:

```
PS> get-command -Module Microsoft.SharePoint.PowerShell | ft name
```

You can see sample output from this command in Figure 1.



```
Windows PowerShell
Get-SPWebAnalyticsServiceApplication
Get-SPWebAnalyticsServiceApplicationProxy
Get-SPWebApplication
Get-SPWebApplicationHttpThrottlingMonitor
Get-SPWebPartPack
Get-SPWebTemplate
Get-SPWorkflowConfig
Grant-SPBusinessDataCatalogMetadataObjectSecurity
Grant-SPObjectSecurity
Import-SPBusinessDataCatalogDotNetAssembly
Import-SPBusinessDataCatalogModel
Import-SPEnterpriseSearchTopology
Import-SPInfoPathAdministrationFiles
Import-SPMetadataWebServicePartitionData
Import-SPSiteSubscriptionBusinessDataCatalogConfig
Import-SPSiteSubscriptionSettings
Import-SPWeb
Initialize-SPResourceSecurity
Initialize-SPStateServiceDatabase
Install-SPApplicationContent
Install-SPDataConnectionFile
Install-SPFeature
Install-SPHelpCollection
Install-SPInfoPathFormTemplate
Install-SPService
Install-SPSolution
Install-SPUserSolution
Install-SPWebPartPack
Install-SPWebTemplate
Merge-SPLogFile
Mount-SPContentDatabase
Mount-SPStateServiceDatabase
Move-SPBlobStorageLocation
Move-SPProfileManagedMetadataProperty
```

Permissions Required

The permissions for being able to execute SharePoint cmdlets are pretty strict—indeed, they essentially mean you have to be a server administrator and a SQL Server administrator. The requirements are as follows:

1. Your account must be assigned the SharePoint_Shell_Access role on any SQL Server databases against which PowerShell will be used, both configuration and content databases. Note that this role is nested into the db_owner role, which means you have full access to the database (and all site collections held within) to do whatever you like—intentionally or not!
2. Your account, or a group that you are a member of, must be a member of the WSS_ADMIN_WPG local group on all servers in the farm.

Using Windows PowerShell to Manage SharePoint 2010

Kevin Laahs

(Reprinted From WindowsITPro Magazine)

3. You must be a site collection administrator on any site collection that you will manage via PowerShell.

These are heavy requirements for using PowerShell and will come as a disappointment to those people who had envisioned being able to script the management of their own site collections. We can only hope that support will extend to this scenario in the future. Indeed, if SharePoint follows what Microsoft has done with Exchange in this area, we should expect that role-based access control will be embedded into PowerShell so that users can perform only those actions allowable with their normal permissions to the site collection.

The cmdlet `Add-SPShellAdmin` can be used to grant the user the correct SQL Server roles and to add the user to the `WSS_ADMIN_WPG` group on all servers. You can use `Add-SPShellAdmin` to provide access to those cmdlets that touch the configuration database (such as `Set-SPFarmConfig`), and content databases, by providing the `-Database` parameter. For example, the following commands grant the user `DOMAIN\KLAHHS` shell access to the content database called `WSS_Content`:

```
$db = get-SPContentDatabase WSS_Content
Add-SPShellAdmin -Username DOMAIN\KLAHHS
-database $db
```

Accessing Cmdlets Remotely

PowerShell 2.0 supports remote execution (leveraging Windows Remote Management), which lets you execute cmdlets without having to physically log on to your SharePoint servers. Instead, you can run a local PowerShell on your workstation and execute commands remotely against your SharePoint servers. Being able to execute commands remotely is essential for managing your whole farm. It lets you execute cmdlets on all your SharePoint servers without having to physically log on to each one. There are generally three ways in which you execute remote commands:

1. Using the `Invoke-Command` cmdlet: This lets you execute one-off commands by specifying the computer on which to run the command as a parameter.
2. Creating a session on a remote computer using `New-PSSession`, then passing that session to the `Invoke-Command` cmdlet: This lets you load the SharePoint snap-in and maintain session state. You can then pass SharePoint cmdlets directly through the `Invoke-Command` cmdlet.
3. As 2 above, but rather than executing commands via `Invoke-Command`, you import the remote session to your local session, which lets you enter native SharePoint cmdlets into your local session that execute against the remote session.

As an example (assuming you have PowerShell configured for remote usage), the following commands on a workstation permit the execution of SharePoint cmdlets on a remote computer called `SPS01` (note that the first command creates a pop-up dialog box asking the user for credentials):

Using Windows PowerShell to Manage SharePoint 2010

Kevin Laahs

(Reprinted From WindowsITPro Magazine)

```
$c = Get-Credential
$s = New-PSSession SPS01 -Authentication
CredSSP -Credential $c
Invoke-Command -Session $s -ScriptBlock
{Add-PSSnapin Microsoft.SharePoint.PowerShell}
Import-PSSession $s
```

Seven Tips For Administering SharePoint through PowerShell

In spite of the greater administrative flexibility PowerShell offers, admins are always eager to make it even easier to use. The following tips detail some SharePoint-specific considerations to keep in mind when using PowerShell cmdlets.

1. Learn the parameters. With almost all cmdlets, you need to define which object is the target by using one or more parameters. The target object could be many things, such as a web application, a content database, or a site collection. Consult the Help for a cmdlet to discover the different ways in which it can be called.

For example the syntax for Get-SPSite indicates four different ways in which it can be called. One way is to specify a content database as the target object for this cmdlet, in which case all site collections within the specified content database will be listed. You use the Get-Help <cmdletname> cmdlet to access the Help for a cmdlet, and the -examples parameter is especially useful for finding out the different ways the cmdlet can be used.

2. Understand the PipeBind object. The PipeBind object type is expected for some parameters (e.g., the -Identity parameter on the Get-SPSite cmdlet). The PipeBind object type accepts multiple formats due to the fact that you can uniquely reference a SharePoint object in multiple ways. For example, you can use the URL for a site collection or its GUID (globally unique identifier). The PipeBind object type ensures that you have the flexibility to use whatever unique identifier is appropriate for the operation at hand.
3. Beware performance hungry operations. Many innocuous cmdlets can operate against a very large number of items. For example, the Get-SPSite cmdlet with no parameters enumerates all the site collections in the farm. Most cmdlets, therefore, have a built-in limit on the number of objects they return. You can explicitly override the limit by using the -Limit parameter, which takes a non-negative number or the word "ALL" as its value. As an example, the Get-SPSite inbuilt limit is 200.
4. Unsure? Use -whatif. As you execute cmdlets (especially those that do Write operations), you might want to be sure beforehand what the outcome of the operation will be. PowerShell supports a parameter called -whatif that lets you see what the operation would do if it were actually executed. This parameter is very useful for those operations where you are unsure of the target objects that will ultimately be operated on. Consider the following:

Using Windows PowerShell to Manage SharePoint 2010

Kevin Laahs

(Reprinted From WindowsITPro Magazine)

```
Get-SPSite -Limit 20 | Remove-SPSite -whatif
```

You can't know for sure what the twenty returned sites would be, so to pipe them without caution into a destructive cmdlet such as Remove-SPSite could be disastrous. The parameter -whatif is your get-out-of-jail-free card.

5. Know your scope and control the objects that you work with. Some cmdlets are global in execution and therefore can be run on any server in the farm (e.g., New-SPSite), whereas others are local and affect only the server upon which they're run (e.g., Start-SPServiceInstance).

There are various ways to work with the subset of the objects that a particular cmdlet returns. First, some cmdlets let you specify parameters that limit the scope of execution to a known entity such as the farm, web application, or content database. Second, you can use client-side filtering using the where-object cmdlet in the pipeline. This lets you look at every object that's returned and process only those that match the specified criteria. All objects are returned initially, and the filtering is done on the client, which could be wasteful as far as server processing is concerned.

Finally, server-side filtering is supported by some cmdlets—usually through, but not limited to, the -Filter parameter. The -Filter parameter performs object filtering during execution of the cmdlet on the server, before the result set is sent to the client. This is much more performant than client-side filtering and is supported by the Get-SPSite cmdlet as well as others such as Get-SPWeb. The Get-SPSite, Get-SPWeb, and Get-SPSiteAdministration cmdlets also support wildcards and regular expressions via the -Regex parameter. Both of these options result in server-side filtering, and their use is encouraged. As an example, the following commands use server-side filtering to return all the webs that have been provisioned with the Blog template and all the webs that are owned by people in the EMEA domain:

```
get-spsite | get-spweb -Filter {$_.Template  
-eq "BLOG#0"}  
get-spsite -Filter {$_.Owner  
-like "EMEA\*"} | get-spweb
```

6. Use calculated properties to extract what you need. Some cmdlets return a collection of values, and you need to use these as input to a calculated property to extract the actual data you're interested in. Calculated properties are defined using a hash table that contains the Name (key) you want to give to the calculated property and the Expression (value) you want to apply to generate the calculated property. For example, the Usage attribute of the Get-SPSite cmdlet returns details about quota and actual usage. To list just the storage that's currently being used by a site collection, you would type the following:

```
Get-SPSite | select URL, @{Name="Size";  
Expression={$_.Usage.Storage}}
```

Using Windows PowerShell to Manage SharePoint 2010

Kevin Laahs

(Reprinted From WindowsITPro Magazine)

7. Don't get exhausted. From a development perspective, it's always a best practice to dispose of objects (such as site, web, and site administration objects) to avoid server memory exhaustion. The same holds true from within PowerShell if you are assigning such objects to variables for subsequent use. (Note that this particular issue doesn't occur when you use these objects interactively in the shell in a single-line command, because they will be automatically disposed of properly after that single command is complete.) As an example, the following two commands run the risk of the structures associated with the `$sites` variable not being disposed of correctly:

```
$sites = Get-SPSite  
$sites | Get-SPWeb
```

This won't be a big issue if you're executing just a small number of commands that affect only a small number of objects, but it's an important consideration if you write scripts that run frequently and touch many objects. To properly dispose of such objects, SharePoint provides two cmdlets, `Start-SPAssignment` and `Stop-SPAssignment`. The former creates structures called assignment collectors, and the latter disposes of these structures. Your task is to let the system know when you want to create and dispose of these structures by calling the appropriate cmdlets.

The cmdlets support two modes of execution. The first uses `-Global` as the single parameter on both these cmdlets to indicate that you want to use the global assignment collector. This is the simplest option, and when the `Start-SPAssignment` cmdlet is encountered, this indicates to SharePoint to start tracking any subsequent assignments until it encounters the `Stop-SPAssignment` cmdlet. At this point, it disposes of all the objects that were assigned in the meantime, and any variables that contained objects will no longer be valid.

The disadvantage of doing this globally is that every object will be kept, whether assigned to a variable or not. So operations that generate a large number of objects (such as `Get-SPSite -Limit ALL`) will cause problems, and your shell will eventually run out of memory. SharePoint does warn you of this possibility if you attempt to execute an operation that will return a large number of objects when a global assignment collector is currently active.

The second mode of execution is to create individual assignment collectors and explicitly use these collectors when generating objects that you want to store for later manipulation. In the following example, I create an assignment collector variable called `$ac` that's subsequently used to create variable `$s`, because I know that it will contain a large number of objects that I want to refer to later in the script. Note that the objects returned here will be disposed of automatically, so no need to worry about any assignment collectors:

```
$ac = Start-SPAssignment  
$s = $ac | Get-SPSite "http://sps*"  
GetSPSite -Limit All  
$s | foreach {$_ .Owner}  
$ac | Stop-SPAssignment
```

Using Windows PowerShell to Manage SharePoint 2010

Kevin Laahs

(Reprinted From WindowsITPro Magazine)

Saving Admin Time

I hope I've helped you to see how PowerShell support in SharePoint 2010 provides administrators with the flexibility to manage SharePoint in many different ways. By building scripts using standard PowerShell techniques, many repetitive tasks can be automated, giving SharePoint administrators some time back to concentrate on other, less mundane, tasks.