Introduction to Windows PowerShell Scripting for Microsoft SharePoint Server 2010

Contents

Before You Begin

Estimated time to complete this lab

60 minutes

Objectives

After completing this lab, you will be able to:

- Find your way around the SharePoint 4.0 Management Console and interact with SharePoint Web applications, site collections, and sites.
- Use scripting techniques in the Windows PowerShell™ command-line interface, such as pipes, filters, wildcards, and enumerations, for administration of Microsoft® SharePoint® Server 2010.
- Explain how to create and assign variables and use the SharePoint object model from Windows PowerShell.

Prerequisites

Before working on this lab, you must have some experience of working with previous versions of SharePoint Products and Technologies.

Lab scenario

The objective of this hands-on lab is to introduce you to using Windows PowerShell to administer a SharePoint Server 2010 environment. The lab begins with a beginners' introduction to the SharePoint 4.0 Management Console, and then explores how you can use progressively more advanced Windows PowerShell scripting techniques to administer your SharePoint server farm.

Exercise 1: Explore the SharePoint 4.0 Management Console

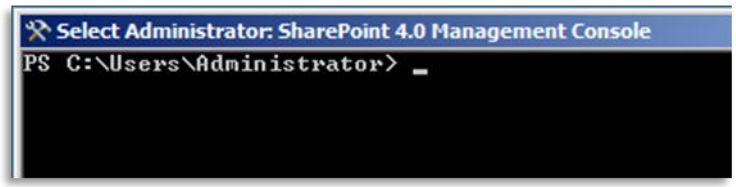
If you are familar with previous versions of SharePoint Products and Technologies, you have probably used the stsadm command-line tool to perform various administrative tasks. You may have used batch files to automate sequences of stsadm commands. You may even have developed your own tool applications by programming against the SharePoint object model.

Windows PowerShell offers a new approach to administration in SharePoint Server 2010 through the SharePoint 4.0 Management Console. Windows PowerShell combines the immediacy of the command line with the power and flexibility of managed code. In this exercise, you will explore the SharePoint 4.0 Management Console and learn how to use Windows PowerShell to accomplish some simple administrative tasks.

Getting Started with the Management Console

In this task, you will familiarize yourself with the SharePoint 4.0 Management Console, learn how to find help and information, and start using Windows PowerShell commands (known as "cmdlets").

1. Ensure that you are logged on to the SP2010 virtual machine as **CONTOSO\Administrator** with a password of **pass@word1**
2. Launch the SharePoint 4.0 Management Console. To do this, on the **Start** menu, point to **Administrative Tools**, and then click **SharePoint 4.0 Management Console**.



1. At the **PS >** prompt, type the command in the following code example, and then press ENTER.

```
Get-Command -pssnapin "Microsoft.SharePoint.PowerShell" |more
```

This command displays all of the available cmdlets in the **Microsoft.SharePoint.PowerShell** namespace. Press the SPACEBAR to page down the list. Notice that there are hundreds of cmdlets that relate to a broad range of administrative tasks.

*Note: You can also type **gcm** as shorthand for **Get-Command**.*

To get help and information about a particular cmdlet, you can use the **get-help <cmdlet-name>** command.

2. Type the command in the following code example, and then press ENTER.

```
get-help Get-SPSite
```

The Command Prompt window displays information about how to use the **Get-SPSite** cmdlet.

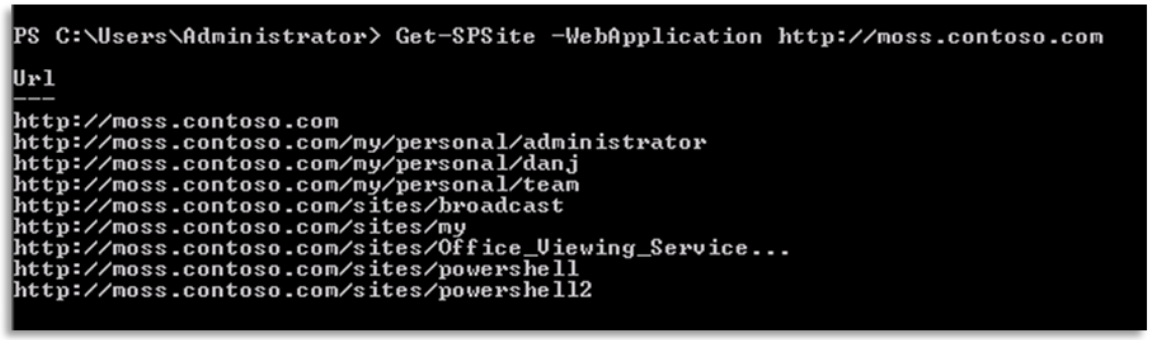
As you might have guessed, cmdlets that include the **Get** prefix return information. You can use these cmdlets to write information to the console, or to provide information to other cmdlets.

You can also use the **get-help <cmdlet-name> -detailed** or **get-help <cmdlet-name> -full** command options to retrieve more detailed technical information about a specific cmdlet. Some of these help files can be very long, so remember that you can append **|more** to any command if you want to page the command output.

3. Type the command in the following code example, and then press ENTER.

```
Get-SPSite –WebApplication http://moss.contoso.com
```

This command returns all of the site collections that exist in the http://moss.contoso.com Web application.



You can use the same approach to retrieve information at each level of the SharePoint architecture. For example, you can retrieve all of the Web applications in your server farm, all of the site collections within a specific Web application, or all of the individual sites within a site collection.

Retrieving Information from the Server Farm

Windows PowerShell provides a highly sophisticated and very powerful interactive scripting language. This can be somewhat daunting to new users. Fortunately, the SharePoint snap-in for Windows PowerShell includes many straightforward cmdlets that can retrieve information from your SharePoint server farm without requiring the ability to write sophisticated scripts. In addition to being an invaluable tool in your day-to-day administrative tasks, these cmdlets also provide a great starting point to learn more about Windows PowerShell. In this task, you will explore cmdlets that retrieve information from a SharePoint server farm.

When you are not sure which cmdlet you need to perform a particular task, you can use the **Get-Command** cmdlet to discover what is available. For example, suppose you want to review which cmdlets can retrieve information from SharePoint Server 2010.

1. Type the command in the following code example, and then press ENTER.

```
Get-Command Get-SP* |more
```

The Command Prompt window displays the names of all of the cmdlets that start with **Get-SP**. The verb **Get** is common to all cmdlets that return information, and the noun prefix **SP** is common to all SharePoint cmdlets. The asterisk is a wildcard character.

*Note: You can use the same approach to find cmdlets relating to specific areas of functionality. For example, **Get-Command *Service*** returns all service-related cmdlets.*

We will now explore some of the more commonly used **Get** cmdlets.

2. Type the command in the following code example, and then press ENTER.

```
Get-SPServiceApplication | Select ID, Name
```

The Command Prompt window displays the **ID** and **Name** properties of each service application that is running in the server farm. Don't worry about the pipe character and the **Select** syntax for now; we will study this later in the lab. Let's consider a more realistic example. Suppose the event logs identify a problem with a service application and provide a globally unique identifier (GUID). You need to find out which service application is causing the problem.

3. Type the command in the following code example, and then press ENTER.

```
Get-SPServiceApplication -Identity 25b17ac2-4c52-4887-8ed6-f066d50ea7e3
```

The Command Prompt window provides details of the service application and identifies the corresponding Internet Information Services (IIS) application pool.

Now let's take a look at feature management. In the course of many routine deployment and troubleshooting tasks, you will need to find the GUID that corresponds to a particular SharePoint feature. This was a somewhat laborious task in previous versions of SharePoint Products and Technologies, and administrators would often resort to full-text searches of the Features folder on the file system. SharePoint Server 2010 provides a far more straightforward approach to this problem through Windows PowerShell.

4. Type the command in the following code example, and then press ENTER.

```
Get-SPFeature
```

The Command Prompt window provides a list of all of the installed features in the server farm, together with an ID and scope for each feature.

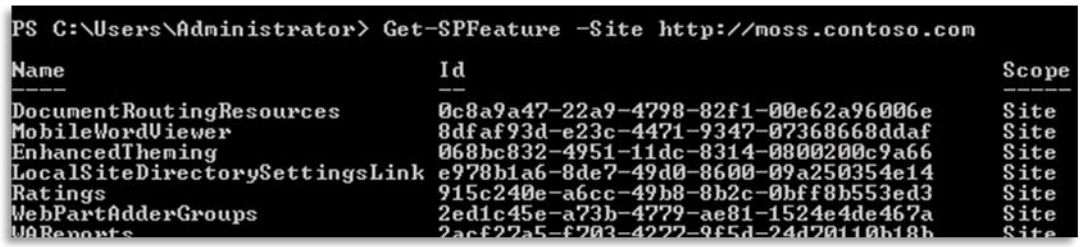
In addition to retrieving a list of all of the installed features in a server farm, you may also want to retrieve a list of all of the features that are activated to a particular scope. To do

this, you can add switch parameters to the **Get-SPFeature** cmdlet (**Site**, **Web**, **WebApplication**, or **Farm**).

5. Type the command in the following code example, and then press ENTER.

```
Get-SPFeature –Site http://moss.contoso.com
```

The Command Prompt window provides a list of all of the features that are activated on the moss.contoso.com site collection.

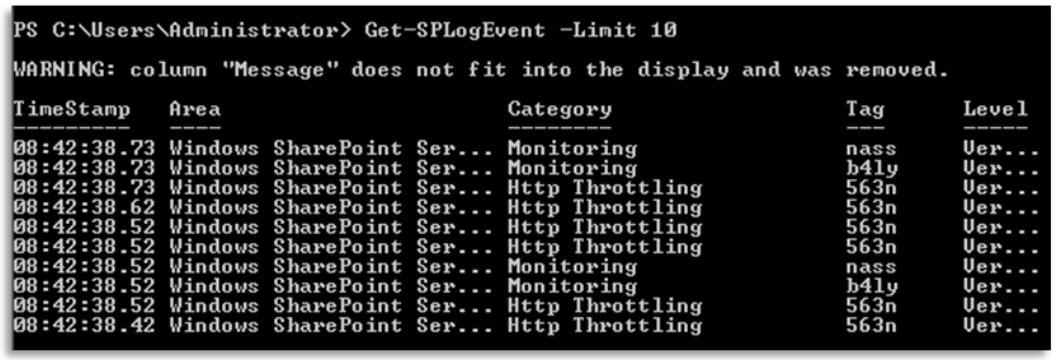


One particularly useful application for administrators of Windows PowerShell for SharePoint Server 2010 is the built-in ability to parse SharePoint log files.

6. Type the command in the following code example, and then press ENTER.

```
Get-SPLogEvent –Limit 10
```

The Command Prompt window displays the last 10 messages that were written to the log files.



The *Limit* parameter is common to many **Get** cmdlets. Where the *Limit* parameter is available, the cmdlet returns 20 rows by default if you do not specify a value. Alternatively, to return all results, you can specify *–Limit All*.

In its current format, this information is not particularly informative. As you work through the remaining exercises in this lab, you will see how you can use increasingly sophisticated techniques to filter and manipulate the information that these cmdlets return.

Creating Sites and Setting Properties

In addition to retrieving information, the SharePoint snap-in for Windows PowerShell includes cmdlets that can set properties, create new items, and perform a wide range of other

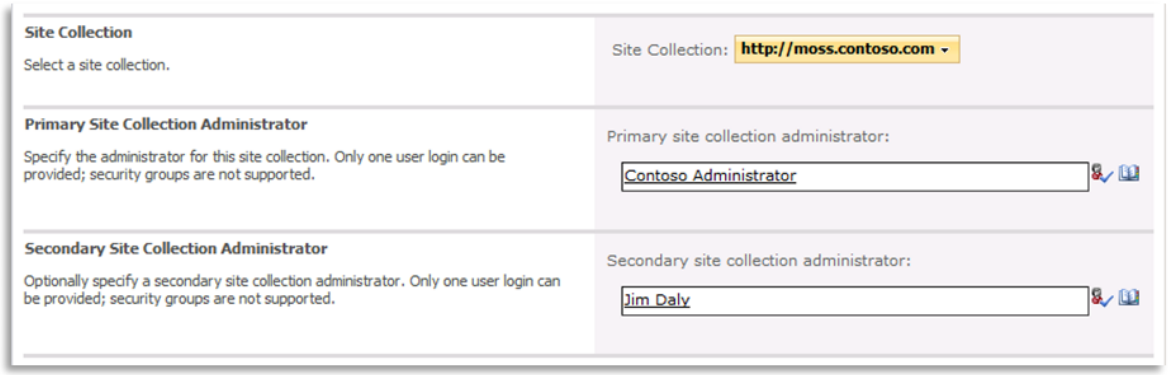
actions. In this task, you will explore how you can put these cmdlets to use in your SharePoint server farm.

Let's start by examining the **Set-SPSite** cmdlet. You can use this cmdlet to configure a range of properties on a specific site collection.

1. Type the command in the following code example, and then press ENTER.

```
Set-SPSite -Identity http://moss.contoso.com -SecondaryOwnerAlias
CONTOSO\jimd
```

2. Launch the Central Administration Web site. To do this, on the **Start** menu, point to **Administrative Tools**, and then click **SharePoint 4.0 Central Administration**.
3. On the Central Administration home page, click the **Application Management** heading.
4. Under **Site Collections**, click **Change site collection administrators**.
5. On the **Site Collection Administrators** page, verify that Jim Daly has been added as a secondary site collection administrator.



In isolation, it may have been quicker to specify the secondary site collection administrator through the Central Administration user interface. However, imagine that you want to add a secondary site collection administrator to 200 site collections. Rather than configuring two hundred site collections manually in Central Administration, you could write a few lines of Windows PowerShell script to loop through the site collections and change the site collection administrator on each one.

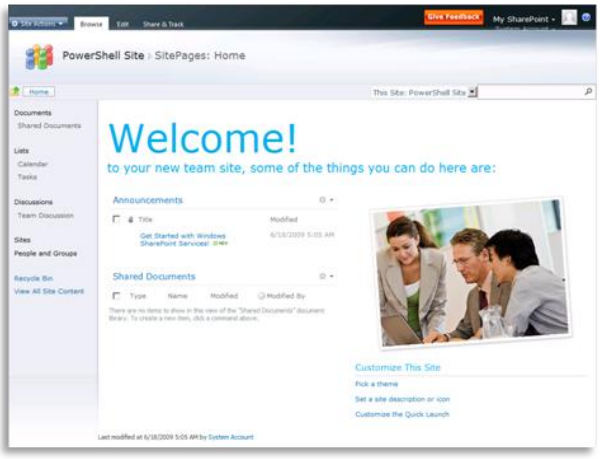
For the final part of this task, let's take a look at how you can use Windows PowerShell to create and delete sites.

6. Switch back to the SharePoint 4.0 Management Console. Type the command in the following code example, and then press ENTER.

```
New-SPSite -Url http://moss.contoso.com/sites/powershell -OwnerAlias
CONTOSO\Administrator -Name "PowerShell Site" -Template STS#0
```

Note: If you do not specify a template here, you will be directed to a template picker page when you browse to the site.

7. When the command returns, switch to the browser window and browse to **http://moss.contoso.com/sites/powershell**. Verify that a new team site has been created at the URL.



8. Switch back to the SharePoint 4.0 Management Console. Type the command in the following code example, and then press ENTER.

```
Remove-SPSite –Url http://moss.contoso.com/sites/powershell –OwnerAlias
CONTOSO\Administrator –Name "PowerShell Site" -Template STS#0
```

9. At the confirm prompt, type **Y** and then press ENTER.
10. When the command returns, switch to the browser window and browse to **http://moss.contoso.com/sites/powershell**. Verify that the site no longer exists.

Exercise 2: Get More from Windows PowerShell

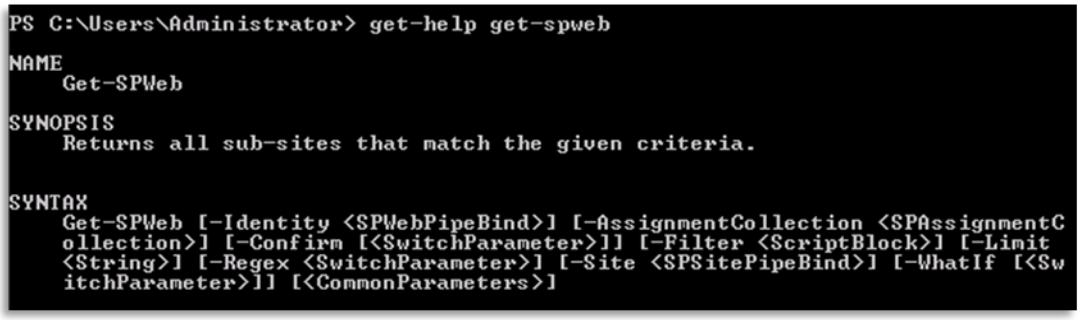
Up to this point, you have used simple SharePoint cmdlets in isolation. When you use cmdlets in this way, it can be hard to see the advantages of Windows PowerShell over standard, noninteractive command-line tools such as stsadm. The real benefits to administrators become apparent when you start to use some of the more sophisticated features of Windows PowerShell, such as combining cmdlets, creating filters, and using wildcard characters. In this exercise, you will explore how to use these features to your advantage when you perform administrative tasks in the SharePoint 4.0 Management Console.

Combining Cmdlets

In this task, you will learn how to combine cmdlets and how to pass data from one cmdlet to another.

1. In the SharePoint 4.0 Management Console, type the command in the following code example, and then press ENTER.

```
Get-Help Get-SPWeb
```



Notice that the *Site* parameter accepts an object of type **SPSitePipeBind**. This indicates that, in addition to providing a value for the parameter directly, you can also use the pipe character "|" to pass in the value from the output of another cmdlet.

2. Type the command in the following code example, and then press ENTER.

```
Get-SPSite http://moss.contoso.com | Get-SPWeb
```

What does this command represent? You are using the **Get-SPSite** cmdlet to get a reference to the site collection at moss.contoso.com, and then passing this reference to the **Get-SPWeb** cmdlet. The **Get-SPWeb** cmdlet then returns all of the sites at the passed-in

site collection. This is functionally equivalent to the command in the following code example.

```
Get-SPWeb –Site http://moss.contoso.com
```

Why use the pipe character, if you can achieve the same result without it? As your Windows PowerShell scripts become more sophisticated, you may not always know the values that you need to pass into a cmdlet in advance. As you progress through the remainder of this lab, you will see examples where this is the case.

You can also use cmdlets directly to provide parameter values for other cmdlets. Consider the **Get-SPLogEvent** cmdlet that we examined earlier. Suppose you wanted to retrieve all events that happened in the last five minutes.

3. Type the command in the following code example, and then press ENTER.

```
Get-SPLogEvent –Limit all –StartTime (Get-Date).AddMinutes(-5)
```

The Command Prompt window displays a list of the events that occurred in the last five minutes. You have yet to configure event log throttling in your server farm, so you will notice that the list is rather long.

What is actually happening here? The **Get-Date** cmdlet returns an object of type **DateTime**. The **DateTime** class includes a method named **AddMinutes**, which returns the current date and time adjusted by the number of minutes that you pass in. The **Get-SPLogEvent** cmdlet uses this value as the input for the *StartTime* parameter.

If you use the same approach to specify the *EndTime* parameter, you can retrieve all of the events that occurred in a specific time window.

4. Type the command in the following code example, and then press ENTER.

```
Get-SPLogEvent –Limit all –StartTime (Get-Date).AddMinutes(-10) –EndTime
(Get-Date).AddMinutes(-5)
```

The Command Prompt window displays a list of the events that occurred between 10 minutes ago and 5 minutes ago.

One final technique to consider in this task is how you can use the **select** keyword to process the results that a cmdlet returns. For example, suppose you want to retrieve a list of all of the event categories in which an event of **Warning** severity has occurred.

5. Type the command in the following code example, and then press ENTER.

```
Get-SPLogEvent –Limit All –MinimumLevel Warning | Select Category -
Unique
```

The Command Prompt window indicates that the only warning has occurred in the Claims Authentication category.

```
PS C:\Users\Administrator> Get-SPLogEvent -Limit All -MinimumLevel Warning | Sel
ect Category -Unique

Category
--------
Claims Authentication
```

You can use the **select** keyword in conjunction with any property of the objects that you are retrieving. In this case, you are retrieving objects of type **SPLogEvent**. The **SPLogEvent** includes a property named **Category**.

Using Filters and Wildcard Characters

In this task, you will explore the use of filters and wildcard characters to refine the functionality of SharePoint cmdlets.

To start with, let's look at the use of a simple wildcard character, the asterisk (*). Suppose you want to retrieve a list of site collections on the managed path http://moss.contoso.com/sites/.

1. Type the command in the following code example, and then press ENTER.

```
Get-SPSite http://moss.contoso.com/sites/*
```

Note: This expression is shorthand for the command in the following code example.

```
Get-SPSite -Identity http://moss.contoso.com/sites/*
```

The Command Prompt window displays a list of site collections that lie on the managed path that you specified.

```
PS C:\Users\Administrator> Get-SPSite http://moss.contoso.com/sites/*

Url
---
http://moss.contoso.com/sites/unitedkingdom
http://moss.contoso.com/sites/broadcast
http://moss.contoso.com/sites/my
http://moss.contoso.com/sites/Office_Viewing_Service...
```

The ability to use wildcard characters can be particularly useful when you combine cmdlets. For example, suppose you want to make Sanjay Shah the secondary site collection administrator for all site collections on the managed path http://moss.contoso.com/sites/.

2. Type the command in the following code example, and then press ENTER.

```
Get-SPSite http://moss.contoso.com/sites/* -Limit All | Set-SPSite -
SecondaryOwnerAlias CONTOSO\sanjays
```

At this point, the efficiency of Windows PowerShell should start to become apparent. Using a single command, you have changed the secondary site collection administrator on multiple sites.

*Note: You can only use the **Get-SPSite** cmdlet if you are already a site owner. If this is not the case, you can use the **Get-SPSiteAdministration** cmdlet to retrieve the site object.*

To verify that the command has worked, you can use another cmdlet with the **select** keyword that we discussed earlier.

3. Type the command in the following code example, and then press ENTER.

```
Get-SPSite | Select Url, SecondaryContact
```

The Command Prompt window displays a list of site collections by URL and secondary site collection administrator. You can see that all of the sites on the http://moss.contoso.com/sites/ managed path now have Sanjay Shah as the secondary site collection administrator.

A few cmdlets also provide a *Filter* parameter that you can use to constrain the results that are returned.

4. Type the command in the following code example, and then press ENTER.

```
Get-SPSite -Filter {$_.SecondaryOwner -eq "CONTOSO\sanjays"}
```

The Command Prompt window displays a list of all site collections for which Sanjay Shah is a secondary site collection administrator. Filter strings always take the format **{$_.PropertyName <operator> "FilterValue"}**. The **$_** variable represents the data in the pipe; in other words, it is a reference to the type that the query returns, which, in this case, is **SPSite**. The operator can take several forms:

- **-lt** (less than)
- **-le** (less than or equal to)
- **-eq** (equal to)
- **-ge** (greater than or equal to)
- **-gt** (greater than)
- **-ne** (not equal to)
- **-like** (matches a wildcard character pattern)
- **-notlike** (does not match a wildcard character pattern)

Another approach to filtering is to pass your cmdlet output to the **Where-Object** cmdlet. You can use the question mark (**?**) as shorthand for **Where-Object**. This cmdlet filters the objects that are passed along the command pipeline according to a filter string that you specify. You can use **Where-Object** to filter the output from any cmdlet.

Note: Where available, the Filter *parameter provides a more efficient way to filter data than the **Where-Object** cmdlet. When you use the* Filter *parameter, the entire query is executed on the server, whereas piping data to the **Where-Object** cmdlet invokes a round-trip to the Windows PowerShell client. However, the **Where-Object** cmdlet is universally available, whereas the* Filter *parameter is only available on a few cmdlets. The **Where-Object** cmdlet also provides extra functionality, such as enabling you to use **match** and **notmatch** operators with regular expressions in the filter string, which are unavailable when you use the* Filter *parameter.*

5. Type the command in the following code example, and then press ENTER.

```
Get-SPLogEvent | ?{$_.Level -eq "High"} | Select Category, Message
```

The Command Prompt window displays a list of all log events where the value of the **Level** property is **High**.

You can also use Regular Expressions to constrain the results that a cmdlet returns. The **Get-SPSite** cmdlet includes a *RegEx* parameter that causes the URL provided for the *Identity* parameter to be treated as a Regular Expression. Suppose you want to retrieve a list of all site collections on the http://moss.contoso.com/sites/ and the http://moss.contoso.com/my/ managed paths.

6. Type the command in the following code example, and then press ENTER.

```
Get-SPSite http://moss.contoso.com/(sites|my)/.* -RegEx -Limit All |
Select Url
```

The Command Prompt window displays a list of all of the site collections that lie on the managed paths that you specified.

Enumerating Collections

When it comes to repetitive tasks, the ability to loop through collections of objects is an invaluable tool in the armory of the Windows PowerShell scriptwriter. In this task, you will learn how to enumerate collections and perform actions on each object in the collection.

Windows PowerShell includes the **ForEach-Object** cmdlet (or **foreach** in shorthand) that you can use to enumerate over a result set. To use the **ForEach-Object** cmdlet, you must pass in a collection of objects, and you must provide the actions to perform on each object within curly brackets. Suppose you want to provision a team blog site on every site collection on the http://moss.contoso.com/sites/ managed path.

1. Type the command in the following code example, and then press ENTER.

```
Get-SPSite http://moss.contoso.com/sites/* -Limit All | foreach{New-
SPWeb -Url ($_.Url + "/blog") -Template BLOG#0}
```

The Command Prompt window returns a list of URLs at which new blog sites have been created.

2. Browse to one or more of the URLs to verify that the blog sites were created correctly.

In the final exercise of this lab, when we discuss assigning variables and using the SharePoint object model, you will see more sophisticated ways of working with the **ForEach-Object** cmdlet.

Exercise 3: Perform Advanced Tasks by Using Windows PowerShell

In the previous exercise, you saw how some of the features of Windows PowerShell enable you to create scripts that go way beyond the capabilities of previous command-line tools such as stsadm. In this exercise, you will take this a step further and learn how you can use the full power of the SharePoint object model from the Windows PowerShell command prompt.

Using Local Variables

The ability to create and assign values to local variables is a powerful benefit when you use Windows PowerShell. However, if you have previously worked with the SharePoint object model, you will know that certain objects, such as **SPSite** and **SPWeb**, need to be managed carefully to avoid excessive memory use. In this task, you will learn how to assign and dispose of local variables in your Windows PowerShell scripts for SharePoint.

The SharePoint snap-in for Windows PowerShell provides two key cmdlets that you can use to manage variable assignment: **Start-SPAssignment** and **Stop-SPAssignment**.

1. In the SharePoint 4.0 Management Console, type the command in the following code example, and then press ENTER.

```
Start-SPAssignment -Global
```

2. Type the command in the following code example, and then press ENTER.

```
$Web = Get-SPWeb http://moss.contoso.com
```

3. Type the command in the following code example, and then press ENTER.

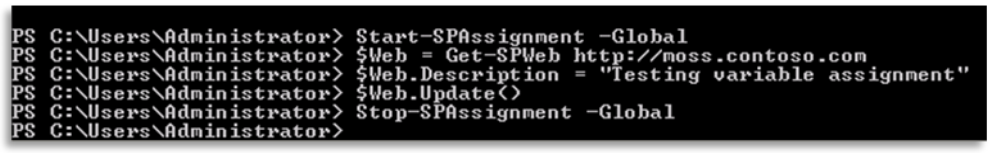
```
$Web.Description = "Testing variable assignment!"
```

4. Type the command in the following code example, and then press ENTER.

```
$Web.Update()
```

5. Type the command in the following code example, and then press ENTER.

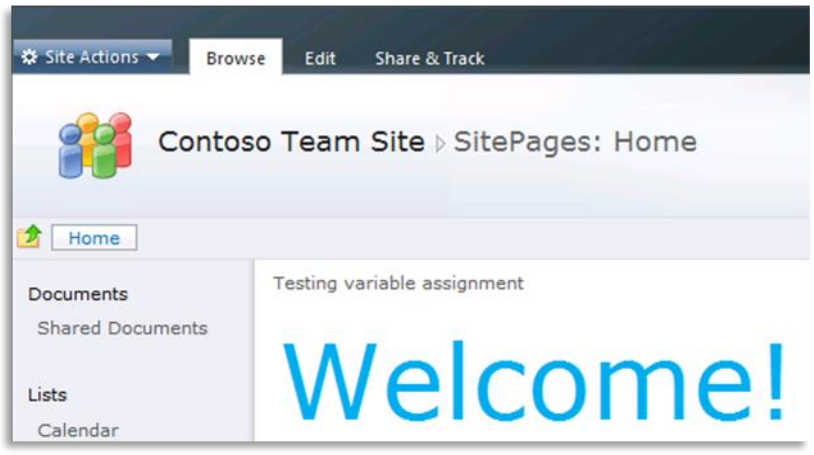
```
Stop-SPAssignment -Global
```



In this case, we are using the global assignment model. After we call **Start-SPAssignment –Global**, any objects that we create are assigned to a global assignment store. This ensures that the objects remain available while we use the object model to set properties and call methods (more on this later). When we call **Stop-SPAssignment –Global**, any objects in the global assignment store are disposed of and the memory is released.

*Note: Use global assignment with caution! For example, if you use **Start-SPAssignment –Global** and then call **Get-SPSite –Limit ALL**, every site collection object will be loaded into memory. In a live server farm, this is likely to cause serious performance issues.*

6. Open a browser window and browse to **http://moss.contoso.com**.
7. Verify that the site description reflects your changes.



When you write complex, long-running scripts, you may require more granular control over how memory is assigned and released. Rather than assigning all of your variables to a global assignment store, you can use named assignment stores that enable you to assign and release different objects at different points in your script.

Suppose you want to create a news site on every site collection on a particular managed path, and then set a few properties on each news site.

8. Type the command in the following code example, and then press ENTER.

```
$SiteScope = Start-SPAssignment
```

9. Type the text in the following code example, and then press ENTER.

```
Foreach($Site in ($SiteScope | Get-SPSite
http://moss.contoso.com/sites/*))
```

Notice that, when you press ENTER, the Windows PowerShell prompt changes to a double angle bracket (>>). This indicates that Windows PowerShell recognizes that your procedure is incomplete, and will accept multiline input without attempting to execute every time you press ENTER.

10. Type the text in the following code example and press ENTER after each line.

```
{
Get-SPFeature PublishingSite | Enable-SPFeature –Url $Site.Url
New-SPWeb –Url ($Site.Url + "/news")
$WebScope = Start-SPAssignment
$Web = $WebScope | Get-SPWeb ($Site.Url + "/news")
$Web.Title = "Team News"
$Web.Description = "All the latest news and information"
$Web.Update()
```

```
Stop-SPAssignment $WebScope
}
Stop-SPAssignment $SiteScope
```

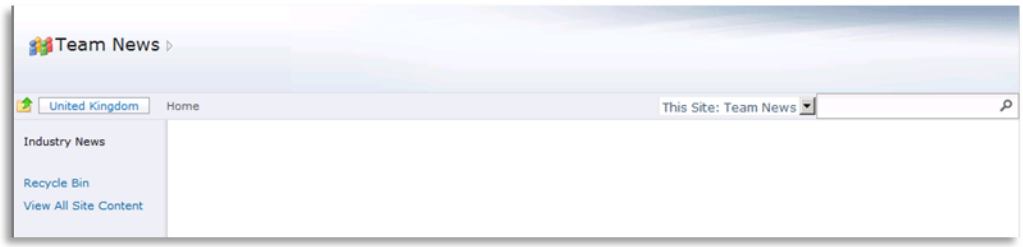
11. Press ENTER again. The entire command should now execute.

```
PS C:\Users\Administrator> $SiteScope = Start-SPAssignment
PS C:\Users\Administrator> foreach($Site in ($SiteScope | Get-SPSite http://moss
.contoso.com/sites/*))
>> {
>> Get-SPFeature PublishingSite | Enable-SPFeature -Url $Site.Url
>> New-SPWeb -Url ($Site.Url + "/news") -Template SPSNEWS#0
>> $WebScope = Start-SPAssignment
>> $Web = $WebScope | Get-SPWeb ($Site.Url + "/news")
>> $Web.Title = "Team News"
>> $Web.Description = "All the latest news and information"
>> $Web.Update()
>> Stop-SPAssignment $WebScope
>> }
>> Stop-SPAssignment $SiteScope
>>

Url
---
http://moss.contoso.com/sites/unitedkingdom/news
http://moss.contoso.com/sites/broadcast/news
http://moss.contoso.com/sites/my/news
http://moss.contoso.com/sites/Office_Viewing_Service...
```

In this case, you are creating two different assignment scopes. The scope named **$SiteScope** is used to store references to site collections and is not cleared until the entire procedure is complete. The scope named **$WebScope** is used to store references to individual sites and is cleared after each individual site is processed.

12. Browse to one of the URLs that the Command Prompt window returns and verify that the News site has been created and configured properly.



*Note: In addition to enabling you to create and assign values to local variables, Windows PowerShell also provides access to environment variables. For example, you can use **$env:os** to retrieve the local operating system, or **$env:computername** to retrieve the local computer name.*

Using the Object Model

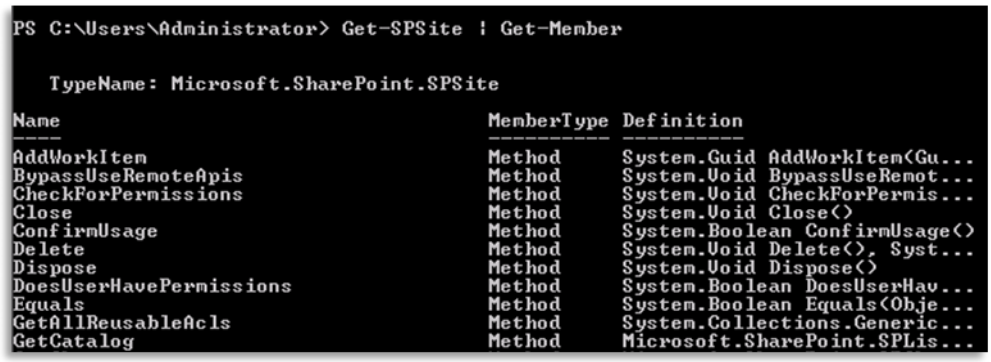
In the previous task, you saw some examples of how you can assign local variables and then use the SharePoint object model to set properties and call methods on those variables. In this task, you will learn more about how to use the SharePoint object model in your Windows PowerShell scripts.

When you start to use the object model, it is essential to be able to find out what methods and properties a particular object supports. Windows PowerShell includes a cmdlet named **Get-Member** that can help. You can pass any object into the **Get-Member** cmdlet to return a list of the available properties and methods on that object.

1. In the SharePoint 4.0 Management Console, type the command in the following code example, and then press ENTER.

```
Get-SPSite | Get-Member
```

   The Command Prompt window displays a long list of the methods and properties that you can access on an **SPSite** object.



   You can experiment with the object model by writing property values to the screen. Remember to create a memory assignment before you create a local variable.

2. Type the command in the following code example, and then press ENTER.

```
Start-SPAssignment -Global
```

3. Type the command in the following code example, and then press ENTER.

```
$Site = Get-SPSite http://moss.contoso.com
```

4. Type the command in the following code example, and then press ENTER.

```
Write-Host $Site.Url
```

   The Command Prompt window displays the URL of the site collection.

5. Type the command in the following code example, and then press ENTER.

```
Write-Host $Site.Zone
```

   The Command Prompt window indicates that the site collection is in the Default zone.

6. Type the command in the following code example, and then press ENTER.

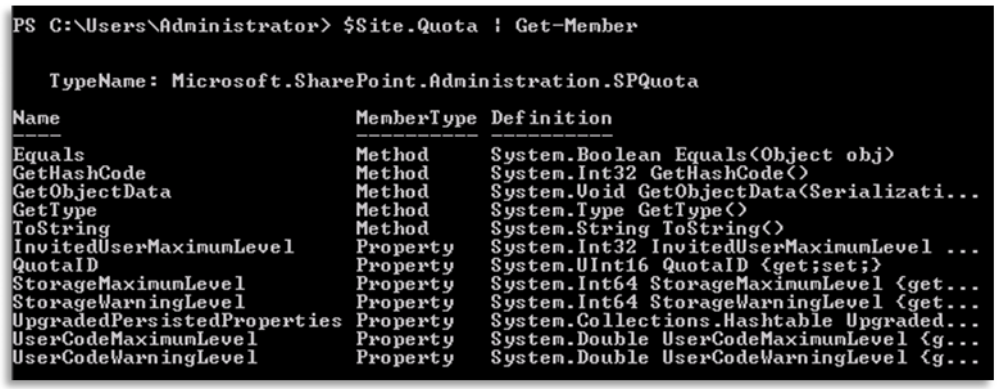
```
Write-Host $Site.Quota
```

   The site quota is a complex type that is not well represented by a string, so the Command Prompt window simply displays the type name of the **SPQuota** object.

As before, you can use the **Get-Member** cmdlet to find out more about the **SPQuota** object.

7. Type the command in the following code example, and then press ENTER.

```
$Site.Quota | Get-Member
```



8. Type the command in the following code example, and then press ENTER.

```
$Site.Quota | Select StorageMaximumLevel
```

The Command Prompt window shows a maximum site storage of zero, which indicates that the site does not have a usage quota assigned. As an administrator, you find this disturbing. You immediately establish a quota of 50 MB, together with a warning level of 40 MB.

9. Type the command in the following code example, and then press ENTER.

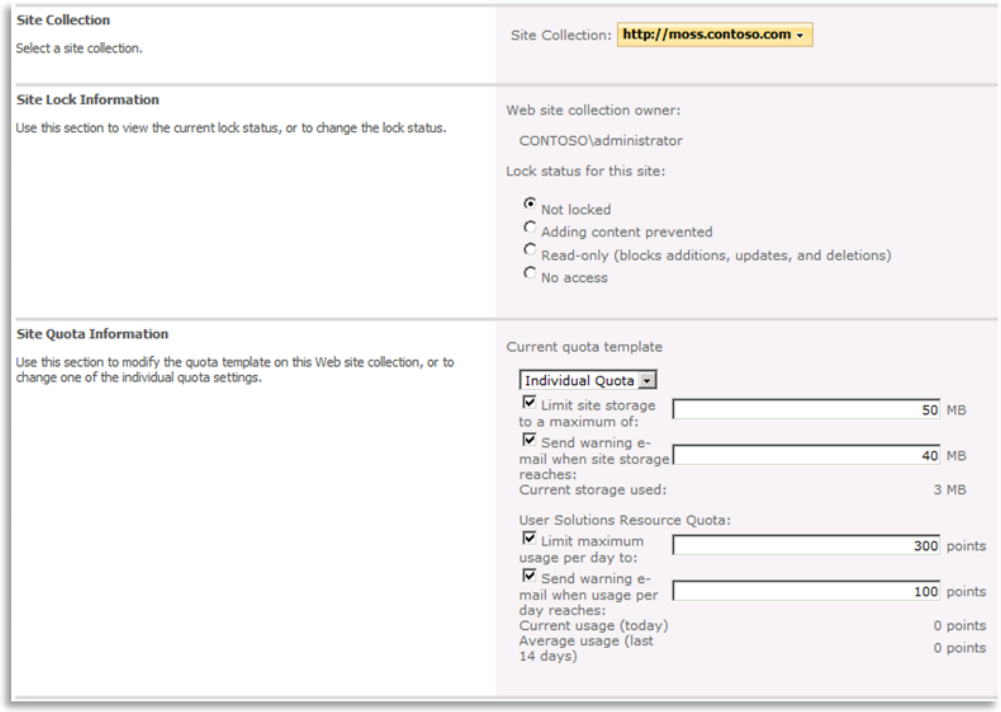
```
$Site.Quota.StorageMaximumLevel = 52428800
```

Note: 52,428,800 represents 50 MB in bytes.

10. Type the command in the following code example, and then press ENTER.

```
$Site.Quota.StorageWarningLevel = 41943040
```

11. Open a browser window, browse to **http://moss:6147/_admin/sitequota.aspx** (the **Site Collection Quotas and Locks** page in the Central Administration Web site), and then verify that the site quotas have been properly set.



Finally, when you have finished using the $Site variable, remember to clear the memory assignment.

12. Type the command in the following code example, and then press ENTER.

```
Stop-SPAssignment -Global
```

Conclusion

This lab provided an overview of how to use Windows PowerShell to script administrative tasks in SharePoint Server 2010. In particular, the lab explored:

- How to get started with the SharePoint 4.0 Management Console.
- How to use Windows PowerShell scripting techniques, such as pipes, filters, wildcard characters, and enumerations, for administration of SharePoint Server 2010.
- How to assign variables and use the SharePoint object model from Windows PowerShell.

For more information about SharePoint Server 2010, visit the SharePoint Products and Technologies team blog at http://blogs.msdn.com/sharepoint for regular announcements and updates.