File Server Management with Windows PowerShell

Make a tedious task easier—without a lot of scripting

Jeffery Hicks
is a Windows PowerShell MVP with almost 20 years of IT experience, spending much of his time helping IT pros work more effectively. He works as an independent consultant, trainer, and author.

Email

Twitter

LinkedIn

Google+

Website

Blog

Managing file servers can be a tedious and thankless task for many IT pros. But it doesn't need to be that way. By incorporating Windows PowerShell, you can easily get a handle on shared file resources—and maybe even have a little fun. And none of it really requires any scripting. Everything I want to show you should take no more than a few interactive commands. Of course, if you find yourself running these commands often, then turning them into a script or function is the next logical step.

The beauty of PowerShell is that you can take the results of the commands that I'm going to demonstrate and do whatever you want. Need to save the results to a comma-separated value (CSV) file? Pipe them to Export-CSV. Need an HTML report? Use ConvertTo-HTML. And everything I'm going to show you scales; if you can use a command for one computer, you can use it for 10, 100, or 1,000 systems.

First, let me show you what you can do to manage what you have in your file shares today. Then we'll look at provisioning file sharing.

Get File Shares

Let's begin by identifying what's being shared. This task is easy: Simply query the Win32_Share class to use Windows Management Instrumentation (WMI). You don't even need to be logged on to the file server. You can run this command from the comfort of your cubicle:

```
Get-WmiObject -class -Win32_Share -computername MyFile
```

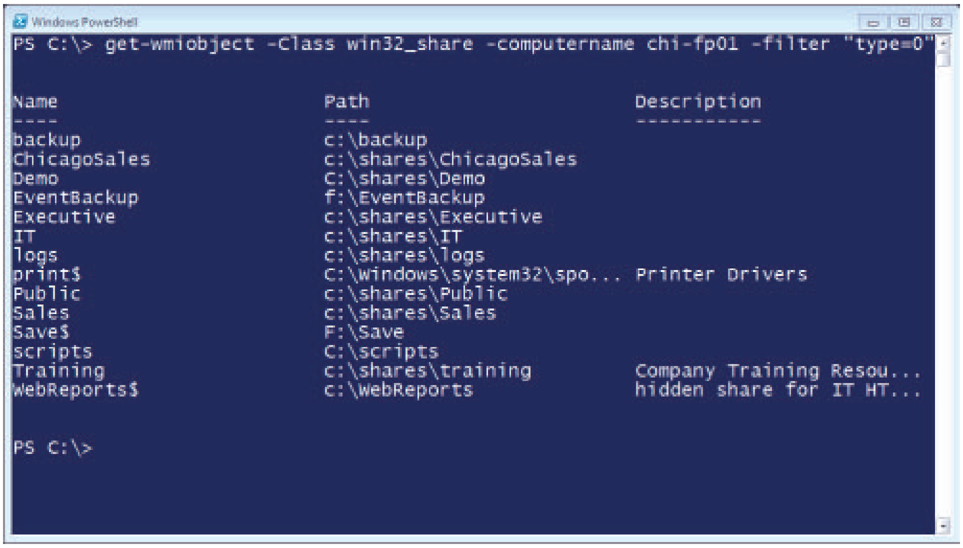
Now, when you run this command, you'll get all shares, including printers (if any). Because we're talking about file shares, let's limit the query. All Win32_Share instances have a Type property, as Table 1 shows. Thus, to limit the search, we can add a filter to our original command:

Table 1: Share Instances and Their Type Properties

Share Type	Value
File share	0
Print share	1
Administrative share	2147483648
IPC	2147483651

```
Get-WmiObject -class -Win32_Share -computername MyFile -filter
  "Type=0"
```

This approach gets rid of the administrative shares. You can see an example in Figure 1.



```
PS C:\> get-wmiobject -Class win32_share -computername chi-fp01 -filter "type=0"

Name                          Path                          Description
----                          ----                          -----------
backup                        c:\backup
ChicagoSales                  c:\shares\ChicagoSales
Demo                          C:\shares\Demo
EventBackup                   f:\EventBackup
Executive                     c:\shares\Executive
IT                            c:\shares\IT
logs                          c:\shares\logs
print$                        C:\Windows\system32\spo...    Printer Drivers
Public                        c:\shares\Public
Sales                         c:\shares\Sales
Save$                         F:\Save
scripts                       C:\scripts
Training                      c:\shares\training            Company Training Resou...
WebReports$                   c:\WebReports                 hidden share for IT HT...


PS C:\>
```

Figure 1
Listing Non-Administrative Shares with WMI

But if you're looking for other hidden shares—that is, those that end in a dollar sign ($)—all you need is a slight tweak to the filter:

It doesn't take much more effort to build a comprehensive usage report for all shares on a file server.

```
Get-WmiObject -Class win32_share -computername MyFile -filter
  "Type=0 AND name like '%$'"
```

In WMI, the percent character (%) is used as a wildcard. Returning all shares except those that are hidden is a little trickier. You'll need to use a compound comparison using a wildcard:

```
Get-WmiObject -Class win32_share -computername MyFile -filter
  "type=0 AND name like '%[^$]'"
```

This command returns all Win32_Share objects that have a Type property of 0 and a name that doesn't end in $.

Get Folder Size

A typical task that's probably on your plate is creating reports about how much disk space a folder is consuming. The quick approach is to simply use Get-ChildItem, or its alias *dir*, and pipe the results to Measure-Object:

```
dir c:\shares\public -recurse |
  where {-Not $_.PSIsContainer}|
  Measure-Object -Property length -Sum -Minimum -Maximum
```

You'll end up with a measurement object that shows the total number of objects, the total size in bytes, and the smallest and largest file sizes. In the previous command, I've filtered out folders. PowerShell 3.0 has better ways of doing this, but the command that I've used works in both PowerShell 2.0 and 3.0. This is the type of command that is best run locally (a great reason to use PowerShell remoting). The code in Listing 1 combines this command with our WMI technique to get a size report for top-level folders. You can format or process $results any way you like. How about an easy-to-read table? Just use this command:

```
$results | Format-Table Computername,Fullname,SizeKB,
  NumberFiles -autosize
```

Figure 2 illustrates the output that you can expect.

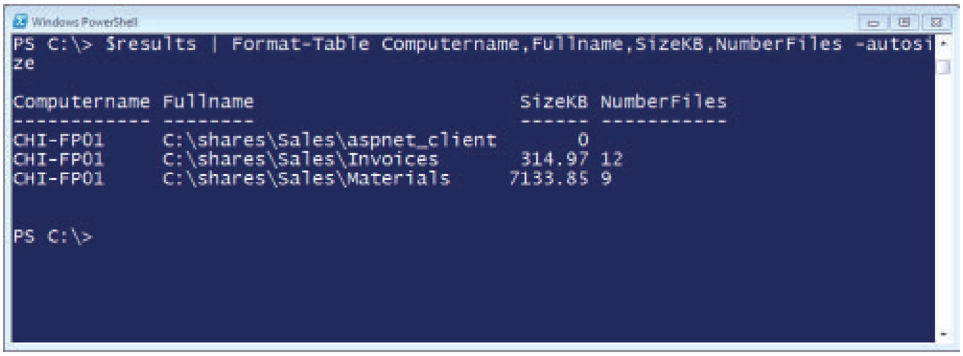
It doesn't take much more effort to build a comprehensive usage report for all shares on a file server. I'll save you the time: Take a look at Listing 2. Again, I can slice and dice $results any way I need. Figure 3 shows one approach.

Listing 1: Getting a Size Report for Top-Level Folders

```
$share=Get-WmiObject -Class Win32_Share -ComputerName
  CHI-FP01 -filter "name='Sales'"
$sb={
Param ($path)
dir $path | where {$_.PSIscontainer} |
foreach {
  $stats=dir $_.Fullname -recurse -errorAction
  "SilentlyContinue" | where {-NOT $_.PSIscontainer} |
  Measure-object -Property Length -sum
  New-Object -TypeName PSObject -Property @{
    Computername=$env:Computername
    Path=$_.Name
    Fullname=$_.Fullname
    SizeKB=[math]::Round(($stats.sum/1KB),2)
    NumberFiles=$stats.count
  } #property
  } #foreach
} #sb

$results=Invoke-Command -ScriptBlock $sb -ComputerName
  $share.__SERVER -ArgumentList @($share.path)
  -HideComputerName
```

Figure 2
Easy-to-Read Output



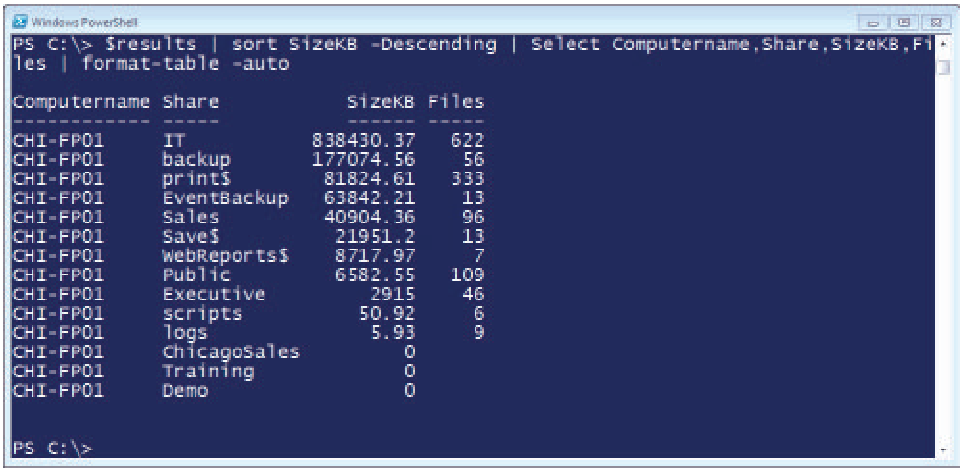
```
PS C:\> $results | Format-Table Computername,Fullname,SizeKB,NumberFiles -autosi
ze

Computername Fullname                          SizeKB NumberFiles
------------ --------                          ------ -----------
CHI-FP01     C:\shares\Sales\aspnet_client          0
CHI-FP01     C:\shares\Sales\Invoices          314.97 12
CHI-FP01     C:\shares\Sales\Materials        7133.85 9


PS C:\>
```

Listing 2: Building a Usage Report for File Server Shares

```
$sb={
#Get the file shares locally.
$shares=Get-WmiObject -Class Win32_Share -filter "type=0"
foreach ($share in $shares) {
    #Suppress any access denied error messages.
    Write-Host "Measuring $($share.path)" -ForegroundColor Green
    $stats=dir $share.path -Recurse -ErrorAction SilentlyContinue
      Where {-Not $_.PSIscontainer} |
      Measure-Object -Property Length -Sum
      $hash=@{
      Computername=$env:Computername
      Path=$share.path
      Share=$share.Name
      SizeKB=[math]::Round(($stats.sum/1KB),2)
      Files=$stats.count
      }
      #Write a custom object to the pipeline for each share.
      New-Object -TypeName PSObject -Property $hash
    } #foreach $share
}#sb
$results = Invoke-Command -ScriptBlock $sb -ComputerName
  CHI-FP01 -HideComputerName
```



```
PS C:\> $results | sort SizeKB -Descending | Select Computername,Share,SizeKB,Fi
les | format-table -auto

Computername Share          SizeKB Files
------------ -----          ------ -----
CHI-FP01     IT          838430.37   622
CHI-FP01     backup      177074.56    56
CHI-FP01     print$       81824.61   333
CHI-FP01     EventBackup  63842.21    13
CHI-FP01     Sales        40904.36    96
CHI-FP01     Save$         21951.2    13
CHI-FP01     WebReports$   8717.97     7
CHI-FP01     Public        6582.55   109
CHI-FP01     Executive        2915    46
CHI-FP01     scripts         50.92     6
CHI-FP01     logs             5.93     9
CHI-FP01     ChicagoSales        0
CHI-FP01     Training            0
CHI-FP01     Demo                0


PS C:\>
```

Figure 3
Usage Report for File Server Shares

Get Files by Owner

A variation on this theme is to find file usage by owner. If you use quotas, you most likely already have reporting in place. Otherwise, all you need to do is retrieve the file ACL, which includes the owner, and aggregate the results. I find that the best approach is to add the owner as a custom property:

```
$data=dir |
  where {-not $_.PSIsContainer} |
  select name, @{Name="Owner";Expression=
  {(Get-ACL $_.fullname).Owner}}, length
```

We can group this output by the new owner property, and then process the new object:

```
$data | group owner |
  Select Name,Count,@{Name="Size";Expression=
  {($_.Group | Measure-Object -Property Length -sum).Sum}}
```

With just a little effort, you can apply this approach to a file share, as the code in Listing 3 does. I should also point out that you might

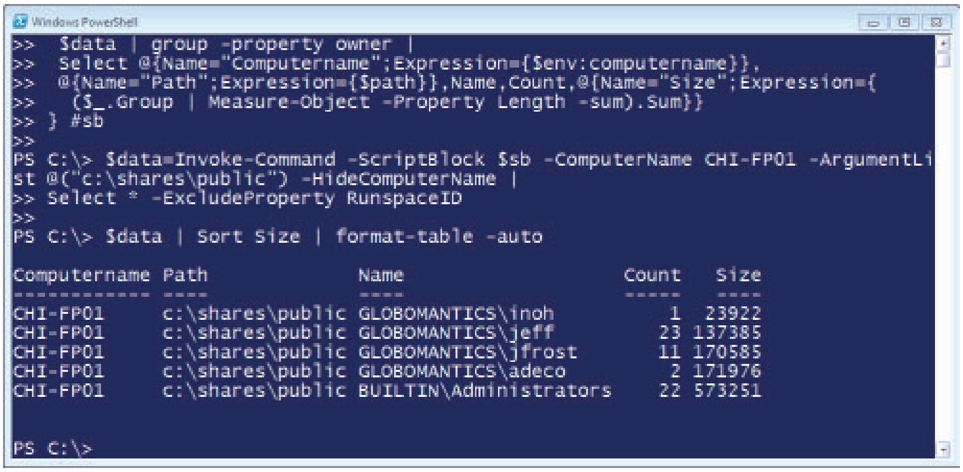
Listing 3: Grouping Output by Owner Property

```
$sb={
Param($path)
$data=dir $path |
  where {-not $_.PSIsContainer} |
  select name, @{Name="Owner";Expression=
  {(Get-ACL $_.fullname).Owner}},length
  $data | group -property owner |
  Select @{Name="Computername";Expression={$env:computername}},
  @{Name="Path";Expression={$path}},Name,Count,@{Name=
  "Size";Expression={
  ($_.Group | Measure-Object -Property Length -sum).Sum}}
} #sb

<#
Run the command remotely and suppress the RunspaceID since we
  don't really need it.
#>
Invoke-Command -ScriptBlock $sb -ComputerName CHI-FP01
  -ArgumentList @("c:\shares\public") -HideComputerName |
  Select * -ExcludeProperty RunspaceID
```

run into issues if filename paths are longer than 260 characters or if filenames contain any odd characters, especially if you attempt to run Get-ACL. In PowerShell 3.0, this cmdlet supports -LiteralPath, which helps.

I'll admit that some of these examples are getting to be a bit much to type—and they aren't the only way to accomplish these tasks, but the point is that you *can*. You can also get by with less detail and structure for ad hoc style reporting. Figure 4 illustrates this code sample with the results formatted as an easy-to-read table.



```
>>   $data | group -property owner |
>>   Select @{Name="Computername";Expression={$env:computername}},
>>   @{Name="Path";Expression={$path}},Name,Count,@{Name="Size";Expression={
>>    ($_.Group | Measure-Object -Property Length -sum).Sum}}
>> } #sb
>>
PS C:\> $data=Invoke-Command -ScriptBlock $sb -ComputerName CHI-FP01 -ArgumentLi
st @("c:\shares\public") -HideComputerName |
>> Select * -ExcludeProperty RunspaceID
>>
PS C:\> $data | Sort Size | format-table -auto

Computername Path              Name                        Count   Size
------------ ----              ----                        -----   ----
CHI-FP01     c:\shares\public GLOBOMANTICS\inoh               1  23922
CHI-FP01     c:\shares\public GLOBOMANTICS\jeff              23 137385
CHI-FP01     c:\shares\public GLOBOMANTICS\jfrost            11 170585
CHI-FP01     c:\shares\public GLOBOMANTICS\adeco              2 171976
CHI-FP01     c:\shares\public BUILTIN\Administrators         22 573251


PS C:\>
```

Figure 4
Ad Hoc Reporting

Get Files by Age

The last reporting technique that I want to demonstrate is building a file aging report. Actually, what we're creating is a collection of objects that we can re-use in several ways. You might want to use the objects to delete or move files, or you might want to build a report that can be emailed to management. Always construct PowerShell commands with maximum reuse and flexibility in mind.

Capturing file aging is a tricky thing. In PowerShell, the file object has several properties that you might want to use. For example,

```
get-item c:\work\wishlist.txt | Format-List Name,*time
```

produces the output in Figure 5.

I prefer to use LastWriteTime to indicate when a file was last touched. I've seen situations in which LastAccessTime is updated through third-party tools such as virus scanners, which can lead to erroneous conclusions. And LastAccessTime has been disabled by

Figure 5: File Aging Data

```
Name           : wishlist.txt
CreationTime   : 11/23/2010 10:31:10 PM
LastAccessTime : 11/23/2010 10:31:10 PM
LastWriteTime  : 2/15/2011 7:36:34 AM
```

default since the days of Windows Vista, although you can re-enable it. You also need to be careful because these values can change depending on whether you're copying or moving a file between volumes. Using this file as an example, we can have PowerShell tell us how old the file is, as Listing 4 shows.

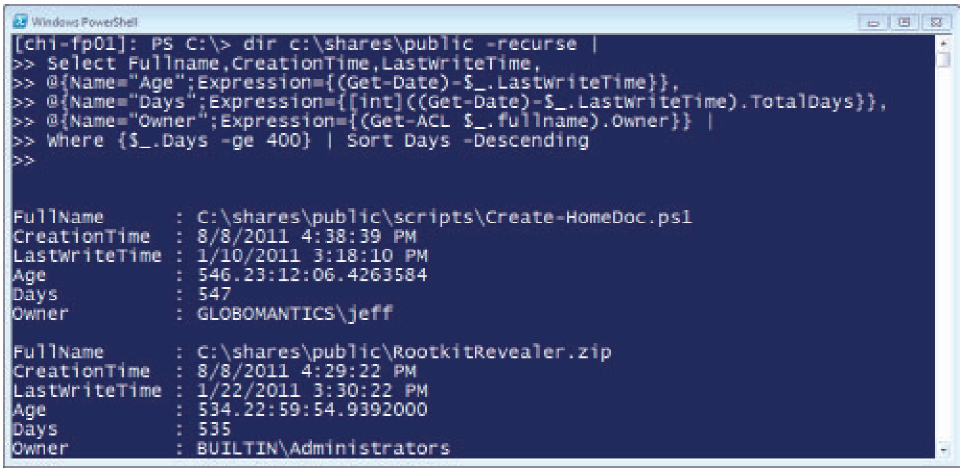
Listing 4: Determining File Age

```
PS C:\work> get-item wishlist.txt | format-list name,
  CreationTime,LastWriteTime,
>> @{Name="Age";Expression={(Get-Date)-$_.LastWriteTime}},
>> @{Name="Days";Expression={((Get-Date)
  -$_.LastWriteTime).TotalDays}}
>>

Name          : wishlist.txt
CreationTime  : 11/23/2010 10:31:10 PM
LastWriteTime : 2/15/2011 7:36:34 AM
Age           : 511.06:42:17.4251748
Days          : 511.279372271039
```

The Age property is a TimeSpan object, and the Days property is merely the TotalDays property of that object. But, because we can do this for a single file, we can do it for all files. Let's look at my public share and find all the files that haven't been modified in 400 days.

```
dir c:\shares\public -recurse |
  Select Fullname,CreationTime,LastWriteTime,
  @{Name="Age";Expression={(Get-Date)-$_.LastWriteTime}},
  @{Name="Days";Expression={[int]((Get-Date)
  -$_.LastWriteTime).TotalDays}},
  @{Name="Owner";Expression={(Get-ACL $_.fullname).Owner}} |
  Where {$_.Days -ge 400} | Sort Days -Descending
```



```
[chi-fp01]: PS C:\> dir c:\shares\public -recurse |
>> Select FullName,CreationTime,LastWriteTime,
>> @{Name="Age";Expression={(Get-Date)-$_.LastWriteTime}},
>> @{Name="Days";Expression={[int]((Get-Date)-$_.LastWriteTime).TotalDays}},
>> @{Name="Owner";Expression={(Get-ACL $_.fullname).Owner}} |
>> Where {$_.Days -ge 400} | Sort Days -Descending
>>


FullName      : C:\shares\public\scripts\Create-HomeDoc.ps1
CreationTime  : 8/8/2011 4:38:39 PM
LastWriteTime : 1/10/2011 3:18:10 PM
Age           : 546.23:12:06.4263584
Days          : 547
Owner         : GLOBOMANTICS\jeff

FullName      : C:\shares\public\RootkitRevealer.zip
CreationTime  : 8/8/2011 4:29:22 PM
LastWriteTime : 1/22/2011 3:30:22 PM
Age           : 534.22:59:54.9392000
Days          : 535
Owner         : BUILTIN\Administrators
```

Figure 6
Running Code in a Remote Session

I went ahead and included the file owner. Figure 6 shows the results from running this code in a remote session on my file server.

I could save these results to a variable and reuse them however I want. Because I have the full filename, piping the variable to a command such as Remove-Item wouldn't be too difficult.

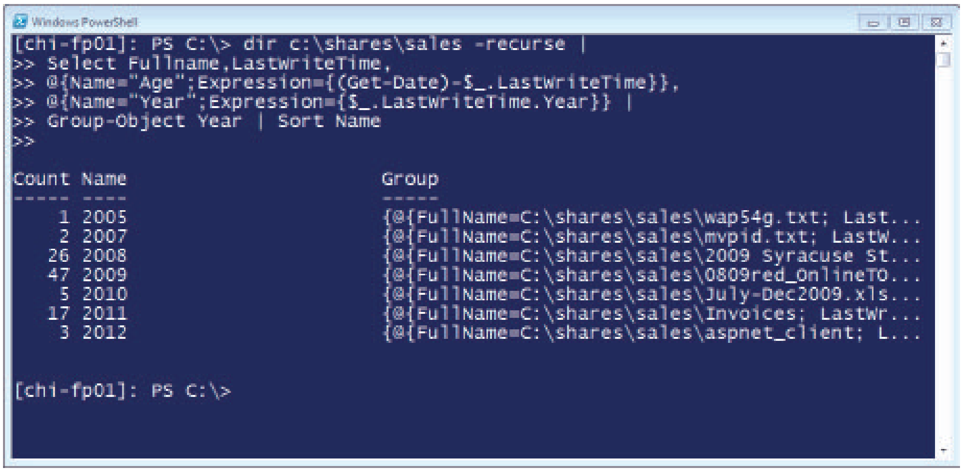
Another approach to file aging is to build a report, or object collection, based on file age buckets. A little more effort is involved; after we calculate or determine the age element, we need to add some logic to do something with it.

One of my favorite techniques is to find out how many files were last modified, by year. Again, I'll use the interactive remoting session on my file server to demonstrate:

```
dir c:\shares\sales -recurse | Select Fullname,LastWriteTime,
  @{Name="Age";Expression={(Get-Date)-$_.LastWriteTime}},
  @{Name="Year";Expression={$_.LastWriteTime.Year}} |
  Group-Object Year | Sort Name
```

As you can see in Figure 7, it looks as though some cleanup is in order. If I need more detail, I can always analyze the Group property, which is the collection of files.

Figure 7
Discovering Modified Files by Year


```
[chi-fp01]: PS C:\> dir c:\shares\sales -recurse |
>> Select Fullname,LastWriteTime,
>> @{Name="Age";Expression={(Get-Date)-$_.LastWriteTime}},
>> @{Name="Year";Expression={$_.LastWriteTime.Year}} |
>> Group-Object Year | Sort Name
>>

Count Name                      Group
----- ----                      -----
    1 2005                      {@{FullName=C:\shares\sales\wap54g.txt; Last...
    2 2007                      {@{FullName=C:\shares\sales\mvpid.txt; LastW...
   26 2008                      {@{FullName=C:\shares\sales\2009 Syracuse St...
   47 2009                      {@{FullName=C:\shares\sales\0809red_OnlineTO...
    5 2010                      {@{FullName=C:\shares\sales\July-Dec2009.xls...
   17 2011                      {@{FullName=C:\shares\sales\Invoices; LastWr...
    3 2012                      {@{FullName=C:\shares\sales\aspnet_client; L...


[chi-fp01]: PS C:\>
```

Finally, what about aging buckets? It might be useful to know how many files haven't been modified in 30, 90, or 180 days. Unfortunately, there isn't an easy way to use Group-Object for this, so I need to take a more brute-force approach; take a look at Listing 5. Figure 8 shows the result when I run this code against my scripts folder, which I know has a decent age distribution. My code doesn't include the actual files, but it wouldn't be too difficult to modify my example.

Listing 5: Determining File Modification Ages

```
$path="c:\scripts"

#Get some other properties in case we want to further
#break down each bucket.
$files=dir $path -recurse |
Select Fullname,CreationTime,LastWriteTime,Length,
@{Name="Age";Expression={(Get-Date)-$_.LastWriteTime}},
@{Name=”Days";Expression={[int]((Get-Date)
  -$_.LastWriteTime).TotalDays}}

$hash=@{
```

Listing 5: *continued*

```
Path=$path
Over=      ($files | Where {$_.Days -gt 365} | Measure-Object).Count
'365Days'= ($files | Where {$_.Days -gt 18Ø -AND
  $_.Days -le 365} | Measure-Object).Count
'18ØDays'= ($files | Where {$_.Days -gt 9Ø -AND
  $_.Days -le 18Ø} | Measure-Object).Count
'9ØDays'=  ($files | Where {$_.Days -gt 3Ø -AND
  $_.Days -le 9Ø} | Measure-Object).Count
'3ØDays'=  ($files | Where {$_.Days -gt 7 -AND
  $_.Days -le 3Ø} | Measure-Object).Count
'7Days'=   ($files | Where {$_.Days -gt Ø -AND
  $_.Days -le 7} | Measure-Object).Count
}
New-Object -TypeName PSObject -Property $hash |
Select Path,Over,365Days,18ØDays,9ØDays,3ØDays,7Days
```

Figure 8: File Modification Ages

```
Path     : c:\scripts
Over     : 1418
365Days  : 295
18ØDays  : 183
9ØDays   : 191
3ØDays   : 88
7Days    : 18
```

Creating File Shares

Next, let's look at how we can use PowerShell to create and manage files and shares. Everything I've demonstrated up to now can work in both PowerShell 2.0 and 3.0 (although in PowerShell 3.0, you could simplify my examples in a few places). File server management in PowerShell 2.0 requires WMI and complicated scripting. But in PowerShell 3.0, especially if you have Windows Server 2012, this

Always construct PowerShell commands with maximum reuse and flexibility in mind.

type of management is a thing of beauty. I'm going to switch gears and manage a Windows Server 2012 file server from a Windows 8 desktop so that I can take advantage of some new features.

Everything we want is bundled into the SMBShare module, which is installed on Windows 8 by default. The commands in this module let us easily manage file shares locally or remotely. I'm not going to cover every command, but they all follow a similar format and I strongly encourage you to read the Help and examples. We'll begin by using the New-SMBShare command to create a new file share.

We need to specify a file path, and unlike what happens when we use the GUI, the command doesn't create the folder if it doesn't exist. So we'll create one and set the NTFS permissions. Doing so takes several steps. Since they must be done on the remote server, I'll set up a PowerShell remoting session:

```
$session=New-PSSession -ComputerName SRV2K12RC
```

I could use this session interactively, but I'm going to run the commands by using Invoke-Command, which is preferable when you're building an automated process. First, I'll create the new folder:

```
invoke-command -ScriptBlock {mkdir c:\shares\companyfiles}
  -Session $session
```

Now for the tricky part. I want to set the NTFS permissions so that \JDHLAB\Domain Users have Change permission. This goal requires creating a new access rule, modifying the access rule list, and re-applying it to the folder. I'll put the commands in the scriptblock that Listing 6 shows. I wrote this scriptblock so that it takes a parameter for the path, which makes it re-usable:

```
Invoke-Command -ScriptBlock $sb -Session $session -ArgumentList
  c:\shares\companyfiles
```

Listing 6: Creating, Modifying, and Applying an Access Rule

```
$sb={
Param($path)
$du=new-object System.Security.AccessControl.FileSystem
  AccessRule "jdhlab\domain users","Modify","allow"
$acl = Get-ACL $path
$acl.AddAccessRule($du)
Set-Acl -Path $path -AclObject $acl
}
```

There are ways to streamline this process, but I kept the steps distinct for the sake of clarity. Now we're ready to create the share.

I could use the session, but I want to demonstrate how you might use the New-SmbShare command to remotely connect to a file server:

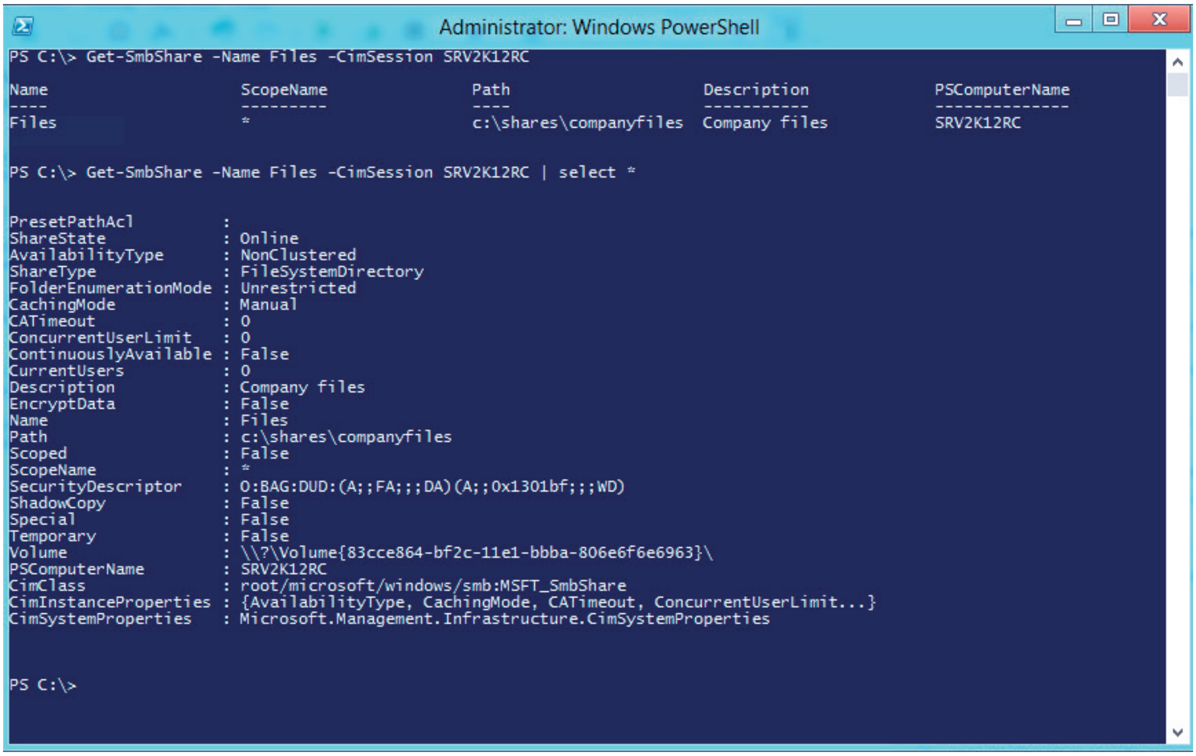
```
New-SmbShare -Name Files -Path c:\shares\companyfiles
  -CimSession SRV2K12RC -FullAccess "jdhlab\domain admins"
  -ChangeAccess Everyone -Description "Company files"
```

The default share-access permission is ReadOnly. I've granted domain admins Full Control on the share, and everyone else has the Change permission. The path is relative to the remote computer, which must be running PowerShell 3.0.

Advanced Share Settings

We can look at our share any time by using the Get-SMBShare command, as you can see in Figure 9. We can set a few extras with our file shares, such as whether to encrypt the SMB connection, which type of folder enumeration mode to use, and which type of caching mode to use. I'm going to use Set-SMBShare to fine-tune the share that I just created, as Listing 7 shows, because I neglected to define these properties during the share's creation.

When you're looking for specialized reporting or provisioning automation, PowerShell is your best bet.



```
PS C:\> Get-SmbShare -Name Files -CimSession SRV2K12RC

Name                  ScopeName             Path                  Description           PSComputerName
----                  ---------             ----                  -----------           --------------
Files                 *                     c:\shares\companyfiles Company files        SRV2K12RC


PS C:\> Get-SmbShare -Name Files -CimSession SRV2K12RC | select *


PresetPathAcl         :
ShareState            : Online
AvailabilityType      : NonClustered
ShareType             : FileSystemDirectory
FolderEnumerationMode : Unrestricted
CachingMode           : Manual
CATimeout             : 0
ConcurrentUserLimit   : 0
ContinuouslyAvailable : False
CurrentUsers          : 0
Description           : Company files
EncryptData           : False
Name                  : Files
Path                  : c:\shares\companyfiles
Scoped                : False
ScopeName             : *
SecurityDescriptor    : O:BAG:DUD:(A;;FA;;;DA)(A;;0x1301bf;;;WD)
ShadowCopy            : False
Special               : False
Temporary             : False
Volume                : \\?\Volume{83cce864-bf2c-11e1-bbba-806e6f6e6963}\
PSComputerName        : SRV2K12RC
CimClass              : root/microsoft/windows/smb:MSFT_SmbShare
CimInstanceProperties : {AvailabilityType, CachingMode, CATimeout, ConcurrentUserLimit...}
CimSystemProperties   : Microsoft.Management.Infrastructure.CimSystemProperties


PS C:\>
```

Figure 9
Using Get-SMBShare to Review a Share

Listing 7: Fine-Tuning a Share

```
PS C:\> Set-SmbShare -Name Files -EncryptData $True
  -FolderEnumerationMode AccessBased -CachingMode
  Documents -CimSession SRV2K12RC

Confirm
Are you sure you want to perform this action?
SRV2K12RC: Performing operation 'Modify' on Target '*,Files'.
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend  [?]
  Help (default is "Y"):
```

I applied this change to a single file share, but it would have been just as easy to use Get-SMBShare to retrieve all file shares and then pipe them to Set-SMBShare and apply a change to all of them:

```
Get-SMBShare -CimSession SRV2K12RC -Special $False |
  Set-SmbShare -EncryptData $True -Confirm:$false
```

This command retrieved all the shares (except administrative shares) on SRV2K12RC and set the EncryptData property to True. I didn't want to answer a prompt for each share, so I set the -Confirm switch to False. Set-SMBshare won't write anything to the pipeline unless you use -Passthru. But I was able to modify everything with one simple command.

Removing Shares

The last thing to look at is removing a share. The code in Listing 8 stops sharing the share that I just created. Could it get any easier? Of course, the folder structure is still in place on the file server.

Listing 8: Stopping a File Share

```
PS C:\> Remove-SmbShare -Name Files -CimSession SRV2K12RC

Confirm
Are you sure you want to perform this action?
SRV2K12RC: Performing operation 'Remove-Share' on Target
  '*,Files'.
[Y] Yes  [A] Yes to All  [N] No  [L] No to All  [S] Suspend
[?]
  Help (default is "Y"):
```

Putting It All Together

Let's wrap this up by putting all the management pieces together in one place: a PowerShell workflow. I don't have space to cover this terrific addition to PowerShell or to go into my code sample, but the great thing about a workflow is that I can have some commands run in parallel. For example, after the folder is created, I can create the share and set the NTFS permissions at the same time, as Listing 9 shows.

Listing 9: Creating a Share and Setting NTFS Permissions

```
Workflow New-FileShare {
Param(
[string]$Name,
[string]$Path,
[string]$Principal,
[string]$Right="Modify"
)

#This must be done first.
Sequence {
    #create the folder
    Write-Verbose -Message  "Creating new folder $path on
      $pscomputername"
    $newfolder = New-Item -Path $path -ItemType Directory
}

#Then this step can happen.
Sequence {
    Parallel {
        #these commands can happen in parallel
        InlineScript {
            Write-Verbose -Message "Modifying NTFS permissions"
            Write-Verbose -Message "Creating entry for
              $using:principal with a right of $using:Right"
            $entry=New-Object -typename System.Security
              .AccessControl.FileSystemAccessRule -argumentlist
              $using:Principal,$using:Right,"allow"
            #get the existing ACL
            $acl = Get-ACL -path $using:path
            #add the new entry
            $acl.AddAccessRule($entry)
            Write-Verbose -Message "Applying the new ACL"
            Set-Acl -Path $using:path -AclObject $acl
        } #inline
       #Create the share.
```

Listing 9: *continued*

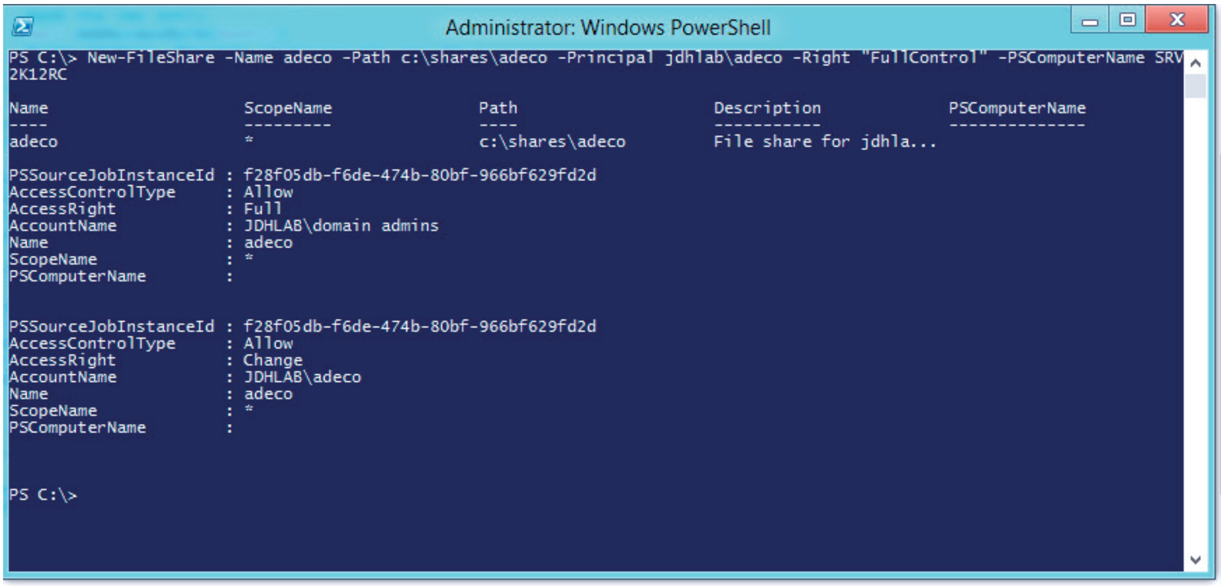
```
        Write-Verbose -message "Creating the file share $name"
        $newshare = New-SmbShare -Name $name -Path $path
          -Description "File share for $principal" -EncryptData $True
          -FolderEnumerationMode AccessBased -CachingMode
          Documents -FullAccess "$env:userdomain\domain admins"
          -ChangeAccess $Principal
    } #Parallel
} #sequence

#Get results.
Sequence {
    Parallel {
        Write-Verbose -Message "Getting the new share"
        Get-SmbShare -Name $name
        Write-Verbose -Message "Getting the new share access"
        Get-SmbShareAccess -Name $name
    }
}
```

This workflow creates a new folder and file share, assigning permissions to a user or group. I can execute this workflow from my Windows 8 desktop and have it run on my Windows Server 2012 file server by using the following command (which should be entered on one line).

```
New-FileShare -Name adeco`
  -Path c:\shares\adeco `
  -Principal jdhlab\adeco `
  -Right "FullControl" `
  -PSComputerName SRV2K12RC
```

The process takes only a few seconds. You can see the results in Figure 10.



```
PS C:\> New-FileShare -Name adeco -Path c:\shares\adeco -Principal jdhlab\adeco -Right "FullControl" -PSComputerName SRV
2K12RC

Name                    ScopeName               Path                    Description             PSComputerName
----                    ---------               ----                    -----------             --------------
adeco                   *                       c:\shares\adeco         File share for jdhla...

PSSourceJobInstanceId : f28f05db-f6de-474b-80bf-966bf629fd2d
AccessControlType     : Allow
AccessRight           : Full
AccountName           : JDHLAB\domain admins
Name                  : adeco
ScopeName             : *
PSComputerName        :


PSSourceJobInstanceId : f28f05db-f6de-474b-80bf-966bf629fd2d
AccessControlType     : Allow
AccessRight           : Change
AccountName           : JDHLAB\adeco
Name                  : adeco
ScopeName             : *
PSComputerName        :


PS C:\>
```

Figure 10
Creating File Shares with a PowerShell Workflow

Download

Download the code

There's nothing wrong with using the GUI to manage your file servers. But when you're looking for specialized reporting or provisioning automation, PowerShell is your best bet.

You can download a copy of my code samples from the *Windows IT Pro* website. If you have questions about any of these examples (or other PowerShell issues), use the forums at PowerShell.org. ■

InstantDoc ID 143789