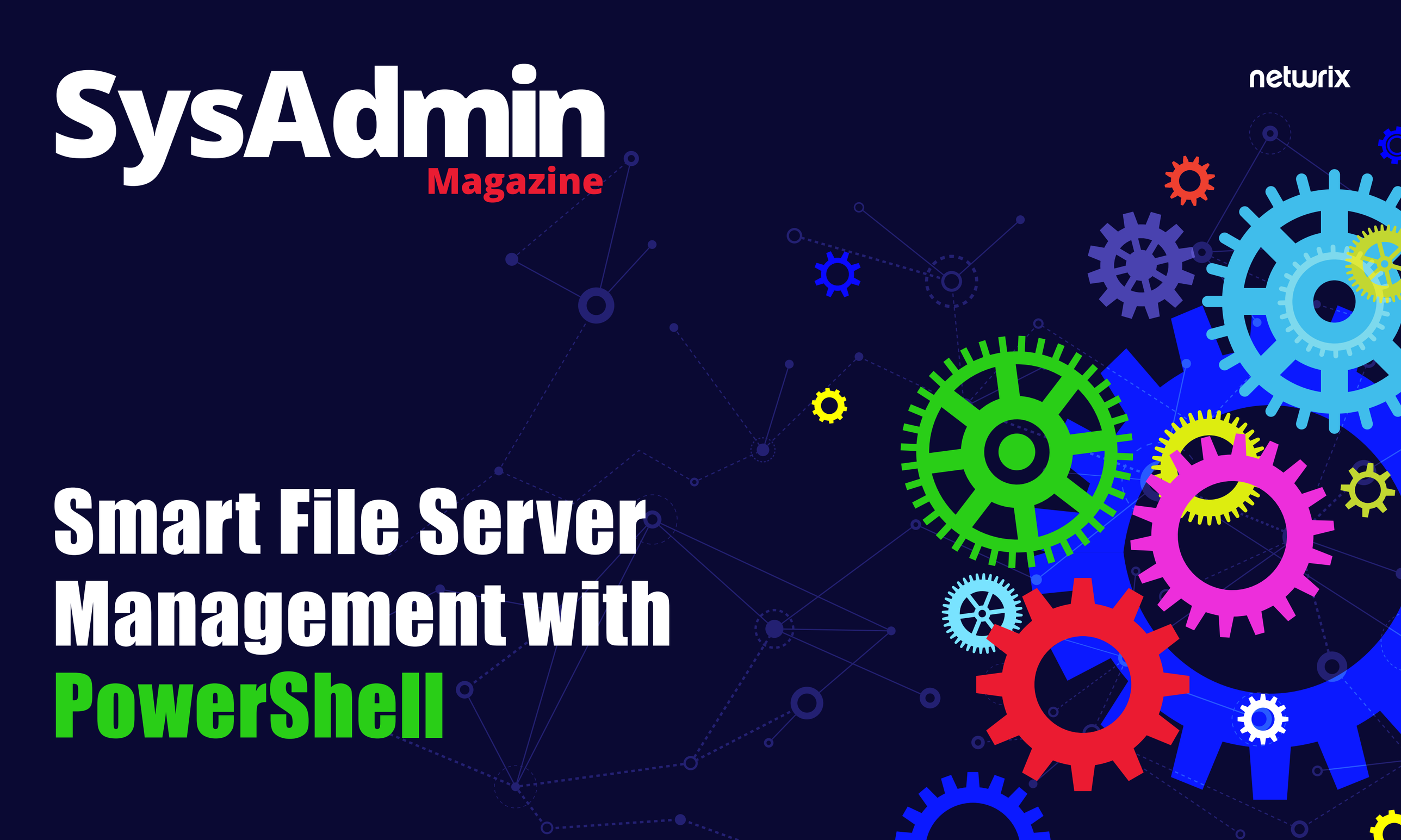
SysAdmin
Magazine

Smart File Server Management with PowerShell



SysAdmin
Magazine

№ 46

March ‘19

SysAdmin Magazine is a free source of knowledge for IT Pros who are eager to keep a tight grip on network security and do the job faster.

The Sysadmin Magazine team
sysadmin.magazine@netwrix.com

Contents

7 tricks of managing file system ACLs with PowerShell

Jeff Melnick
IT Security Expert, Blogger

Many organizations with a Microsoft Windows environment rely on NTFS as the main file system for their storage devices that contain sensitive data. It is the easiest way for users to work with files. In order to implement a least-privilege model, which is a best practice for system security, IT security specialists and system administrators configure NTFS access control lists (ACLs) by adding access control entries (ACEs) on NTFS file servers.

NTFS permissions types for files and folders

There are both basic and advanced NTFS permissions. You can set each of the permissions to "Allow" or "Deny". Here are the basic permissions:

- **Full Control**: Users can modify, add, move and delete files and directories, as well as their associated properties. In addition, users can change permissions settings for all files and subdirectories.
- **Modify**: Users can view and modify files and file properties, including deleting and adding files to a directory or file properties to a file.
- **Read & Execute:** Users can run executable files, including script
- **Read**: Users can view files, file properties and directories.
- **Write**: Users can write to a file and add files to directories.

Here is the list of **advanced permissions**:

- **Traverse Folder/Execute File:** Users can navigate through folders to reach other files or folders, even if they have no permissions for these files or folders. Users can also run executable files. The Traverse Folder permission takes effect only when the group or user doesn't have the "Bypass Traverse Checking" right in the Group Policy snap-in.
- **List Folder/Read Data:** Users can view a list of files and subfolders within the folder as well as the content of the files.
- **Read Attributes:** Users can view the attributes of a file or folder, such as whether it is read-only or hidden.
- **Write Attributes:** Users can change the attributes of a file or folder.
- **Read Extended Attributes:** Users can view the extended attributes of a file or folder, such as permissions and creation and modification times.
- **Write Extended Attributes:** Users can change the extended attributes of a file or folder.
- **Create Files/Write Data:** The "Create Files" permission allows users to create files within the folder. (This permission applies to folders only.) The "Write Data" permission allows users to make changes to the file and overwrite existing content. (This permission applies to files only.)
- **Create Folders/Append Data:** The "Create Folders" permission allows users to create folders within a folder. (This permission applies to folders only.) The "Append Data" permission allows users to make changes

- to the end of the file, but they can't change, delete or overwrite existing data. (This permission applies to files only.)
- **Delete:** Users can delete the file or folder. (If users don't have the "Delete" permission on a file or folder, they can still delete it if they have the "Delete Subfolders And Files" permission on the parent folder.)
- **Read Permissions:** Users can read the permissions of a file or folder, such as "Full Control", "Read", and "Write".
- **Change Permissions:** Users can change the permissions of a file or folder.
- **Take Ownership:** Users can take ownership of the file or folder. The owner of a file or folder can always change permissions on it, regardless of any existing permissions that protect the file or folder.
- **Synchronize:** Users can use the object for synchronization. This enables a thread to wait until the object is in the signaled state. This right is not presented in ACL Editor. You can read more about it here.

You can find all these user permissions by running the following PowerShell script:

```
[system.enum]::getnames([System.Security.
AccessControl.FileSystemRights])
```

NTFS permissions can be either explicit or inherited. Explicit permissions are permissions that are configured individually, while inherited permissions are inherited from the parent folder. The hierarchy for permissions is as follows:

- Explicit Deny
- Explicit Allow
- Inherited Deny
- Inherited Allow

Now that we know NTFS permissions are, let's explore how to manage them.

Get ACL for files and folders

The first PowerShell cmdlet used to manage file and folder permissions is "get-acl"; it lists all object permissions. For example, let's get the list of all permissions for the folder with the object path "\\fs1\shared\sales":

```
get-acl \\fs1\shared\sales | fl
```

```
PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\J.Brown Allow  FullControl
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
Audit  :
```

If you want to get a full NTFS permissions report via PowerShell, you can follow this easy how-to about exporting NTFS permissions to CSV.

Copy file and folder permissions

To copy permissions, a user must own both the source and target folders. The following command will copy the permissions from the "Accounting" folder to the "Sales" folder:

```
get-acl \\fs1\shared\accounting | Set-Acl
\\fs1\shared\sales
```

```
PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl

Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\J.Brown Allow  FullControl
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
Audit  :
Sddl   : O:BAG:S-1-5-21-210521867-2639090965-1213260628-513D:AI(A;OICI;FA;;;S-1-5-21-6
         928-1670731417-1160)(A;OICIID;FA;;;SY)(A;OICIID;FA;;;S-1-5-21-611411812-38042



PS C:\Users\j.carter> get-acl \\fs1\shared\accounting | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\T.Simpson Deny  FullControl
         ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
```

As we can see from the output of the “get-acl” commands before and after the permissions copy, the “Sales” shared folder permissions have been changed.

Set file and folder permissions

The PowerShell “set-acl” cmdlet is used to change the security descriptor of a specified item, such as a file, folder or a registry key; in other words, it is used to modify file or folder permissions. The following script sets the “FullControl” permission to “Allow” for the user “ENTERPRISE\T.Simpson” to the folder “Sales”:

```
$acl = Get-Acl \\fs1\shared\sales

$AccessRule = New-Object System.Security.
AccessControl.FileSystemAccessRule("ENTER-
PRISE\T.Simpson","FullControl","Allow")

$acl.SetAccessRule($AccessRule)

$acl | Set-Acl \\fs1\shared\sales$acl | Set-
Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$AccessRule = New-Object System.Security.AccessControl.FileSystemAccessRule("ENTERPRISE\T.Simpson","FullControl","Allow")
$acl.SetAccessRule($AccessRule)
$acl | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\T.Simpson Deny  FullControl
         ENTERPRISE\T.Simpson Allow  FullControl
         ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
```

If you want to set other permissions to users or security groups, choose them from the table below:

Access Right	Access Right's Name in PowerShell
Full Control	FullControl
Traverse Folder / Execute File	ExecuteFile
List Folder / Read Data	ReadData
Read Attributes	ReadAttributes
Read Extended Attributes	ReadExtendedAttributes
Create Files / Write Data	CreateFiles
Create Folders / Append Data	AppendData
Write Attributes	WriteAttributes
Write Extended Attributes	WriteExtendedAttributes
Delete Subfolders and Files	DeleteSubdirectoriesAndFiles
Delete	Delete
Read Permissions	ReadPermissions

Access Right	Access Right's Name in PowerShell	Name of the Set in PowerShell
Read	List Folder / Read Data Read Attributes Read Extended Attributes Read Permissions	Read
Write	Create Files / Write Data Create Folders / Append Data Write Attributes Write Extended Attributes	Write
Read and Execute	Traverse folder / Execute File List Folder / Read Data Read Attributes Read Extended Attributes Read Permissions	ReadAndExecute
Modify	Traverse folder / Execute File List Folder / Read Data Read Attributes Read Extended Attributes Create Files / Write Data Create Folders / Append Data Write Attributes Write Extended Attributes Delete Read Permissions	Modify

Remove user permissions

To remove a permission, use the “RemoveAccessRule” parameter. Let’s delete the “Allow FullControl” permission for T.Simpson to the “Sales” folder:

```
$acl = Get-Acl \\fs1\shared\sales

$AccessRule = New-Object System.Security.
AccessControl.FileSystemAccessRule("ENTER-
PRISE\T.Simpson","FullControl","Allow")

$acl.RemoveAccessRule($AccessRule)

$acl | Set-Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$AccessRule = New-Object System.Security.AccessControl.FileSystemAccessRule ("ENTERPRISE\T.Simpson","FullControl","Allow")
$acl.RemoveAccessRule($AccessRule)
$acl | Set-Acl \\fs1\shared\sales
True

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\T.Simpson Deny  FullControl
         ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
```

Notice that T.Simpson still has the “Deny FullControl” permission. To remove it, let’s use the command “PurgeAccessRules”, which will completely wipe T.Simpson’s permissions to the “Sales” folder:

```
$acl = Get-Acl \\fs1\shared\sales

$usersid = New-Object System.Security.Principal.Ntaccount ("ENTERPRISE\T.Simpson")

$acl.PurgeAccessRules($usersid)

$acl | Set-Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$usersid = New-Object System.Security.Principal.Ntaccount ("ENTERPRISE\T.Simpson")
$acl.PurgeAccessRules($usersid)
$acl | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
```

Note that “PurgeAccessRules” doesn’t work with a string user name; it works only with SIDs. Therefore, we used the “Ntaccount” class to convert the user account name from a string into a SID. Also note that “PurgeAccessRules” works only with explicit permissions; it does not purge inherited ones.

Disable or enable permissions inheritance

To manage inheritance, we use the “SetAccessRuleProtection” method. It has two parameters:

- The first parameter is responsible for blocking inheritance from the parent folder. It has two states: “$true” and “$false”.
- The second parameter determines whether the current inherited permissions are retained or removed. It has the same two states: “$true” and “$false”.

Let’s disable inheritance for the “Sales” folder and delete all inherited permissions as well:

```
$acl = Get-Acl \\fs1\shared\sales

$acl.SetAccessRuleProtection($true,$false)

$acl | Set-Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$acl.SetAccessRuleProtection($true,$false)
$acl | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\ceo Allow  Modify, Synchronize
```

Now we have only one access permission left (because it was added explicitly); all inherited permissions were removed.

Let’s revert this change and enable inheritance for the folder “Sales” again:

```
$acl = Get-Acl \\fs1\shared\sales
$acl.SetAccessRuleProtection($false,$true)
$acl | Set-Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$acl.SetAccessRuleProtection($false,$true)
$acl | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : BUILTIN\Administrators
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
```

Change file and folder ownership

If you want to set an owner for a folder, you need to run the "SetOwner" method. Let's make "ENTERPRISE\J.Carter" the owner of the "Sales" folder:

```
$acl = Get-Acl \\fs1\shared\sales
$object = New-Object System.Security.Principal.Ntaccount("ENTERPRISE\J.Carter")
$acl.SetOwner($object)
$acl | Set-Acl \\fs1\shared\sales
```

```
PS C:\Users\j.carter> $acl = Get-Acl \\fs1\shared\sales
$object = New-Object System.Security.Principal.Ntaccount("ENTERPRISE\J.Carter")
$acl.SetOwner($object)
$acl | Set-Acl \\fs1\shared\sales

PS C:\Users\j.carter> get-acl \\fs1\shared\sales | fl


Path   : Microsoft.PowerShell.Core\FileSystem::\\fs1\shared\sales
Owner  : ENTERPRISE\j.carter
Group  : G:S-1-5-21-210521867-2639090965-1213260628-513
Access : ENTERPRISE\ceo Allow  Modify, Synchronize
         ENTERPRISE\ceo Allow  ReadAndExecute, Synchronize
         NT AUTHORITY\SYSTEM Allow  FullControl
         ENTERPRISE\I.Scur Allow  FullControl
         ENTERPRISE\Domain Admins Allow  FullControl
         BUILTIN\Administrators Allow  FullControl
```

Notice that we again used the "Ntaccount" class to convert the user account name from a string into a SID.

Note that the "SetOwner" method does not enable you to change the owner to any account you want; the account must have the "Take Ownership", "Read" and "Change Permissions" rights.

As you can see, it is very easy to manage NTFS permissions with PowerShell. But don't forget to audit NTFS permissions as well — it's critical for security to track all changes made to your file servers in order to reduce data leakage and combat the insider threat and other IT security risks. Here is a basic guide on how to audit NTFS permissions with PowerShell.

The ultimate guide to file management with PowerShell

Ian Skur
PowerShell Expert

Every day, sysadmins have to perform various standard operations on the numerous files and folders on their Windows servers. These tasks often include managing users' data on shared resources and maintaining backups properly. You can use PowerShell to reduce amount of manual work involved.

Before you start, make sure your system policy allows running PowerShell scripts as described in "Windows PowerShell Scripting Tutorial for Beginners."

Viewing the objects in a directory

To view the content of a directory on a Windows file server, use the Get-ChildItem cmdlet. To show all hidden files, add the -Force parameter. The command below shows all root objects in the Shared folder:

```
Get-ChildItem -Force \\fs\Shared
```

If you want to also check all subfolders and their content, add -Recurse parameter:

```
Get-ChildItem -Force \\fs\Shared -Recurse
```

To filter the output, add the Filter, Exclude, Include and Path parameters to the Get-ChildItem cmdlet. For advanced object filtering, use the Where-Object cmdlet. The script below searches for all executable files in the IT folder that were modified after April 1, 2018:

```
Get-ChildItem -Path \\fs\Shared\IT -Recurse
-Include *.exe | Where-Object -FilterScript
{($_.LastWriteTime -gt '2018-04-01')}
```

Creating files and folders with PowerShell

To create new objects with Windows PowerShell, you can use the New-Item cmdlet and specify the type of item you want to create, such as a directory, file or registry key.

For example, this command creates a folder:

```
New-Item -Path '\\fs\Shared\NewFolder'
-ItemType Directory
```

And this command creates an empty file:

```
New-Item -Path '\\fs\Shared\NewFolder\newfi-
le.txt' -ItemType File
```

Creating files and writing data to them

There are at least two built-in methods to create a file and write data to it. The first is to use the Out-File cmdlet:

```
$text = 'Hello World!' | Out-File $text -Fi-
lePath C:\data\text.txt
```

To overwrite an existing file, use the –Force switch parameter.

You can also create files using the Export-Csv cmdlet, which exports the output into a csv file that can be opened in Excel:

configuration settings and is available only from the Share-Point administration center.

```
Get-ADuser -Filter * | Export-Csv -Path C:\
data\ADusers.csv
```

Creating files after checking that they don't already exist

The following script checks whether a specific file (pc.txt) already exists in a particular folder; if not, it generates a list of all AD computers and saves it to a new file named pc.txt:

```
#create array of text files
$files=Get-ChildItem C:\data\*.txt | select
-expand fullname
#check if file exists inside the array
$files -match "pc.txt"
#if matching return "True" key then exit,
if "False" then create a report
if($files -eq 'False'){
Get-ADComputer -Filter * | Export-Csv -Path C:\
data\pc.txt
}
else{exit}
```

Deleting files and folders with PowerShell

To delete objects, use the Remove-Item cmdlet. Please note that it requires your confirmation upon execution if the object is not empty. The example below demonstrates how to delete the IT folder and all the subfolders and files inside it:

```
Remove-Item -Path '\\fs\shared\it\'
Confirm
The item at \\pdc\shared\it has children and
the Recurse parameter was not
specified. If you continue, all children will be
removed with the item. Are you
sure you want to continue?
[Y] Yes  [A] Yes to All  [N] No  [L] No to All
[S] Suspend  [?] Help
(default is "Y"):
```

If you have already made sure that every object inside the folder should be deleted, you can use the ?Recurse switch to skip the confirmation step:

```
Remove-Item -Path '\\fs\shared\it\' -Recurse
```

Deleting files and folders older than X days

Sometimes you need to clean up old files from a certain directory. Here's the way to accomplish that:

```
$Folder = "C:\Backups"

#delete files older than 30 days
Get-ChildItem $Folder -Recurse -Force -ea 0 |
? {!$_.PsIsContainer -and $_.LastWriteTime -lt
(Get-Date).AddDays(-30)} |
ForEach-Object {
   $_ | del -Force
   $_.FullName | Out-File C:\log\deletedback-
ups.txt -Append
}

#delete empty folders and subfolders if any
exist
Get-ChildItem $Folder -Recurse -Force -ea 0 |
```

```
? {$_.PsIsContainer -eq $True} |
? {$_.getfiles().count -eq 0} |
ForEach-Object {
    $_ | del -Force
    $_.FullName | Out-File C:\log\deletedback-
ups.txt -Append
}
```

Deleting files after checking they exist

Here's how to check whether a file exists and delete it if it does:

```
$FileName = 'C:\data\log.txt'
If (Test-Path $FileName){
   Remove-Item $FileName
}
```

Deleting files from multiple computers in one script

To delete files from remote PCs, you must have the appropriate security permissions to access them. Be sure to use UNC paths so the script will correctly resolve the file locations.

```
$filelist = @(" \c$\Temp", "\c$\Backups") #vari-
able to delete files and folder
$computerlist = Get-Content  C:\data\pc.txt
#get list of remote pc's
    foreach ($computer in $computerlist){
        foreach ($file in $filelist){
            $filepath= Join-Path "\\$computer\"
"$filelist" #generate unc paths to files or fol-
ders
            if (Test-Path $filepath)
            {
```

Copying files and folders with PowerShell

The Copy-Item cmdlet enables you to copy objects from one path to another. The following command creates a backup by copying the file users.xlsx from one remote computer (fs) and saving it to another (fs2) over the network:

```
Copy-Item -Path \\fs\Shared\it\users.xlsx
-Destination \\fs2\Backups\it\users.xlsx
```

If the target file already exists, the copy attempt will fail. To overwrite the existing file, even if it is in Read-Only mode, use the -Force parameter:

```
Copy-Item -Path \\fs\Shared\it\users.xlsx
-Destination \\fs2\Backups\it\users.xlsx -Force
```

Copying files with PowerShell to or from a remote computer

If you're copying files to or from remote computers, be sure to use UNC paths.

For example, use this command to copy files from a remote file server to the local C: directory:

```
Copy-Item \\fs\c$\temp -Recurse C:\data\
```

To copy files from your local directory to the remote folder, simply reverse the source and destination locations:

```
Copy-Item C:\data\ -Recurse \\fs\c$\temp
```

Copying multiple files from one server to another over the network in one script

You can also copy files from one remote server to another. The following script recursively copies the \\fs\Shared\temp folder to \\fs\Shared\test:

```
Copy-Item \\fs\Shared\temp -Recurse \\fs\Shared\test
```

Copying only certain types of files

To copy only certain files from the source content to the destination, use the -Filter parameter. For instance, the following command copies only txt files from one folder to another:

```
Copy-Item -Filter *.txt -Path \\fs\Shared\it
-Recurse -Destination \\fs2\Shared\text
```

Copying files using XCOPY and ROBOCOPY commands or COM objects

You can also run XCOPY and ROBOCOPY commands to copy files, or use COM objects as in the example below:

```
(New-Object -ComObject Scripting.FileSystemObject).CopyFile('\\fs\Shared', 'fs2\Backup')
```

Moving files and directories with PowerShell

The Move-Item cmdlet moves an item, including its properties, contents, and child items, from one location to another. It can also move a file or subdirectory from one directory to another location.

The following command moves a specific backup file from one location to another:

```
Move-Item -Path \\fs\Shared\Backups\1.bak
-Destination \\fs2\Backups\archive\1.bak
```

This script moves the entire Backups folder and its content to another location:

```
Move-Item -Path \\fs\Shared\Backups -Destination \\fs2\Backups\archive
```

The Backups directory and all its files and subfolders will then appear in the archive directory.

Renaming files with PowerShell

The Rename-Item cmdlet enables you to change the name of an object while leaving its content intact. It's not possible to move items with the Rename-Item command; for that functionality, you should use the Move-Item cmdlet as described above.

The following command renames a file:

```
Rename-Item -Path "\\fs\Shared\temp.txt" -NewName "new_temp.txt"
```

Renaming multiple files

To rename multiple files at once, use a script like this:

```
$files = Get-ChildItem -Path C:\Temp #create
list of files
foreach ($file in $files)
{
    $newFileName=$file.Name.Replace("A","B")
#replace "A" with "B"
    Rename-Item $file $newFileName
}               -Begin {$total=0;} `
               -Process {$total+=[int]$_.
File.Length} `
               -End {$total} `
          }
```

Changing file extensions with PowerShell

You can also use the Rename-Item to change file extensions. If you want to change the extensions of multiple files at once, use the Rename-Item cmdlet with the Get-ChildItem cmdlet.

The following script changes all “txt” file extensions to “bak”. The wildcard character (*)ensures that all text files are included:

```
Get-ChildItem \\fs\Shared\Logs\*.txt | Re-
name-Item -NewName { $_.name -Replace
'\.txt$','.bak' }           } |
   format-table -AutoSize
```

Using the information in this article, you can automate a variety of simple operations related to file management on your file storages and save time for more important tasks. Good luck!

Using PowerShell to create ZIP archives and unzip files

Russell Smith
IT Consultant, PowerShell Expert

Sometimes it can be useful to programmatically create zip archives or extract files from existing archives. Windows PowerShell 5.0 added two cmdlets for doing just that. The Compress-Archive cmdlet enables you to create new archives from folders or individual files and to add files to archives; Extract-Archive can be used to unzip files.

If you don't already have PowerShell 5.0 or later installed on your systems, you can download the latest version of the Windows Management Framework from Microsoft's website here.

Using PowerShell to create zip files

Let's start by using PowerShell to compress files in a new zip archive. All you need to do is use the **-Path** parameter to specify the folder you want to compress and the **–DestinationPath** parameter to specify the name of the archive you want to create. The command below will zip the Invoices folder in the root C directory and create an archive called Invoices.zip in the Archives folder:

```
Compress-Archive -Path C:\Invoices -DestinationPath C:\Archives\Invoices
```

Alternatively, we could zip the files in the Invoices folder individually using **-LiteralPath** instead of **–Path**. This command creates an archive with just the two files explicitly listed in the –LiteralPath parameter:

```
Compress-Archive -LiteralPath C:\ Invoices\File1.txt, C:\Invoices\File2.txt -DestinationPath C:\Archives\Invoices -Force
```

Note that I added the **-Force** parameter to overwrite the archive that I created using the first command. Without the **-Force** parameter, you cannot overwrite existing archives and PowerShell will prompt you to add files to the archive instead.

To add files to an archive, use the **-Update** parameter. The command below adds all the files in the Invoices folder to my existing Invoices.zip archive:

```
Compress-Archive -Path C:\Invoices\* -Update -DestinationPath C:\Archives\Invoices
```

Optionally, you can use the **-CompressionLevel** parameter with one of three values: Optimal, NoCompression or Fastest. Optimal is the default setting if the **-CompressionLevel** parameter is not set; it uses the best compression available, but it might take longer than using Fastest. To create an archive with no compression, use the NoCompression value.

Using PowerShell to unzip files

Extracting files from an archive is even easier than creating one. All you need to do is specify the name of the archive and the destination folder for the unzipped files. The command below extracts the contents of the Invoices.zip archive to a folder named InvoicesUnzipped using the Expand-Archive cmdlet.

```
Expand-Archive -LiteralPath C:\Archives\Invoi-
ces.Zip -DestinationPath C:\ InvoicesUnzipped
```

The folder where you want to unzip the files doesn't need to exist; **Expand-Archive** will automatically create the folder if needed. But if the files you want to unzip already exist in the destination folder, **Expand-Archive** will return an error. You can overwrite files in the destination folder by adding the **-Force** parameter to the command.

How to copy files from one server to another

Larry Glusman
Windows Network Administrator, Blogger

Copying files from one server to another is a fairly frequent task that system administrators face. As with most things in IT, there are many ways to accomplish this task. The method you choose will depend on your situation and your personal preferences. The information below assumes that both of your servers are on the same domain.

If you don't like the command line, you could just drag and drop the files to the new server. But Windows Explorer doesn't have any real error handling or logging, so if anything goes wrong, you end up having to guess what caused the problem. There are several good third-party tools out there to enhance Explorer, but most admins prefer to keep their servers as lean and clean as possible.

In the modern world of Virtual Machines, another option is to just make a copy of the entire VM file and then attach that as a secondary drive to the destination server. This is only viable if you want all of the files and all of the same permission settings. If that is your goal, it's easy to do and works well.

How to copy files to a server using robocopy

If you only want to copy some of the files to a server, you will probably want to use Robocopy. It's been around since Windows Server 2003 and is a very powerful tool. With Robocopy you can copy a single file, stripping all permissions, mirror an entire drive while keeping all NTFS permissions in place, and pretty much anything in between. You can run Robocopy from your workstation, specifying the source and destination file servers, but running it directly on one of the servers will be faster and will generate less network traffic.

If you want to copy folder DATA on Server1 to Server2, for instance, you could use this command:

- **Robocopy \\server1\data \\server2\data /mir /copyall /dcopy:T**

This will create an exact copy of the source folder structure, including all permissions and time stamps. Because it's creating a mirror, it will also delete anything in the destination that doesn't match the source. It is perfect for copying to blank destinations but is potentially hazardous if there is already data there.

If you want to copy an entire folder, but don't want to keep the permissions or delete anything from the destination folder, you could use the following command:

- **Robocopy \\server1\data \\server2\data /e**

But Robocopy doesn't just do simple copying. As mentioned earlier, it's a very powerful tool. You can set it to monitor (/mon) the source folder and copy it again if changes are detected. You can set it to list only (/l) to preview results without actually copying anything. You can set it to only copy files with the hidden attribute set (/ia:H) or to exclude (/xa:H) hidden files.

You can also do some things that don't involve copying files. Perhaps you used the /copyall option because you wanted all permissions copied over but then had to change some of the permissions on the source. You can use /sec /nocopy to update the permissions without having to copy the files from the server again.

Tool of the Month

Free Community Edition

Netwrix Auditor for Windows File Servers

Download Free Tool

Freeware tool that keeps you aware of changes and data access on your file servers.

It collects and analyzes log events on file server and sends daily activity summaries that detail all read attempts and changes that were made during the last 24 hours, with the before and after values.

Netwrix Auditor for File Servers

Activity Summary

Added	1
Modified	1
Renamed	1

Action	Object type	What	Item	Where	When	Workstation
Added	File	\\fs3\shared\Documents\Orders\1875409723.pdf	\\fs3\shared	fs3	4/17/2017 3:04:27 AM	atl-mkt021.enterprise.com
Renamed	File	\\fs3\shared\Documents\Contractors\payroll2017.docx	\\fs3\shared	fs3	4/17/2017 3:04:43 AM	atl-mkt021.enterprise.com
Name changed from "payroll2017.docx" to "payroll2017_Q1.docx"						
Modified	File	\\fs3\shared\Documents\Accounting\payroll.rtf	\\fs3\shared	fs3	4/17/2017 3:05:14 AM	atl-mkt021.enterprise.com
Added **User permissions:** "ENTERPRISE\B.Hops (Allow Inherited: All access)"						

How-to for IT Pro

How to Find Duplicate Files

1. Open the PowerShell ISE Run the following script, adjusting the directory path:

```
$Path = '\\PDC\Shared\Accounting' #define
path to folders to find duplicate files
$Files=gci -File -Recurse -path $Path | Se-
lect-Object -property FullName,Length
$Count=1
$TotalFiles=$Files.Count
$MatchedSourceFiles=@()
ForEach ($SourceFile in $Files)
{
  Write-Progress -Activity "Processing Fi-
les" -status "Processing File $Count / $To-
talFiles" -PercentComplete ($Count / $To-
talFiles * 100)
  $MatchingFiles=@()
  $MatchingFiles=$Files |Where-Object {$_.
Length -eq $SourceFile.Length}
  Foreach ($TargetFile in $MatchingFiles)
  {
    if (($SourceFile.FullName -ne $Target-
File.FullName) -and !(($MatchedSourceFiles |
      Select-Object -ExpandProperty File)
-contains $TargetFile.FullName))
    {
      Write-Verbose "Matching $($SourceFile.
FullName) and $($TargetFile.FullName)"
      Write-Verbose "File sizes match."
      if ((fc.exe /A $SourceFile.FullName
$TargetFile.FullName) -contains "FC: no dif-
ferences encountered")
      {
        Write-Verbose "Match found."
        $MatchingFiles+=$TargetFile.FullName
      }
    }
  }
  if ($MatchingFiles.Count -gt 0)
  {
    $NewObject=[pscustomobject][ordered]@{
      File=$SourceFile.FullName
      MatchingFiles=$MatchingFiles
    }
    $MatchedSourceFiles+=$NewObject
  }
  $Count+=1
}
$MatchedSourceFilesGroup,Role,Inherited" |
out-file $ExportFile
```

2. Review the results:

File	MatchingFiles
\\PDC\Shared\Accounting\Q4 report.pdf	{@{FullName=\\PDC\Shared\Accounting\Q4 report.pdf; Length=223344}, @{FullName=\\PDC\Shared\Accounting\Old Records\2016\Q4 report.pdf; Length= 223344}, \\PDC\Shared\Accounting\Old Records\2016\Q4 report.pdf}
\\PDC\Shared\Accounting\Taxes.xlsx	{@{FullName=},\\PDC\Shared\Accounting\Taxes.xlsx; Length=229958}, @{FullName=},\\PDC\Shared\Accounting\Old Records\2016\Taxes.xlsx;Length=229958}, },\\PDC\Shared\Accounting\Old Records\2016\Taxes.xlsx}
\\PDC\Shared\Accounting\Old Records\2016\Q4 report.pdf	{@{FullName=},\\PDC\Shared\Accounting\Q4 report.pdf; Length=223344}, @{FullName=},\\PDC\Shared\Accounting\Old Records\2016\Q4 report.pdf; Length=223344}}

[On-Demand Webinar]

4 Handy PowerShell commands for managing the file system

Despite the increasing popularity of cloud storage and SharePoint, Windows file servers still play an important role in the enterprise. Understanding who has access to data and keeping file servers secure isn't an easy task.

Watch this webinar with Russell Smith, IT consultant and PowerShell expert for a pocketful of scripts that will enable you to:

- Modify permissions on files and folders
- Get permissions on files and folders
- Find folders with excessive permissions
- Audit permission changes

Russell Smith
IT Consultant, PowerShell Expert

Watch Now

Corporate Headquarters:
300 Spectrum Center Drive,
Suite 200 Irvine, CA 92618

Phone: 1-949-407-5125
Toll-free: 888-638-9749
EMEA: +44 (0) 203-318-02