**Working with Files and Folders**

* Article
* 10/08/2021
* 4 minutes to read
* 4 contributors

Navigating through Windows PowerShell drives and manipulating the items on them is similar to manipulating files and folders on Windows physical disk drives. This article discusses how to deal with specific file and folder manipulation tasks using PowerShell.

**Listing All the Files and Folders Within a Folder**

You can get all items directly within a folder by using Get-ChildItem. Add the optional **Force** parameter to display hidden or system items. For example, this command displays the direct contents of Windows PowerShell Drive C (which is the same as the Windows physical drive C):

PowerShellCopy

Get-ChildItem -Path C:\ -Force

The command lists only the directly contained items, much like using Cmd.exe's DIR command or ls in a UNIX shell. In order to show contained items, you need to specify the -Recurse parameter as well. (This can take an extremely long time to complete.) To list everything on the C drive:

PowerShellCopy

Get-ChildItem -Path C:\ -Force -Recurse

Get-ChildItem can filter items with its **Path**, **Filter**, **Include**, and **Exclude** parameters, but those are typically based only on name. You can perform complex filtering based on other properties of items by using Where-Object.

The following command finds all executables within the Program Files folder that were last modified after October 1, 2005 and which are neither smaller than 1 megabyte nor larger than 10 megabytes:

PowerShellCopy

Get-ChildItem -Path $env:ProgramFiles -Recurse -Include \*.exe | Where-Object -FilterScript {($\_.LastWriteTime -gt '2005-10-01') -and ($\_.Length -ge 1mb) -and ($\_.Length -le 10mb)}

**Copying Files and Folders**

Copying is done with Copy-Item. The following command backs up C:\boot.ini to C:\boot.bak:

PowerShellCopy

Copy-Item -Path C:\boot.ini -Destination C:\boot.bak

If the destination file already exists, the copy attempt fails. To overwrite a pre-existing destination, use the **Force** parameter:

PowerShellCopy

Copy-Item -Path C:\boot.ini -Destination C:\boot.bak -Force

This command works even when the destination is read-only.

Folder copying works the same way. This command copies the folder C:\temp\test1 to the new folder C:\temp\DeleteMe recursively:

PowerShellCopy

Copy-Item C:\temp\test1 -Recurse C:\temp\DeleteMe

You can also copy a selection of items. The following command copies all .txt files contained anywhere in C:\data to C:\temp\text:

PowerShellCopy

Copy-Item -Filter \*.txt -Path c:\data -Recurse -Destination C:\temp\text

You can still use other tools to perform file system copies. XCOPY, ROBOCOPY, and COM objects, such as the **Scripting.FileSystemObject,** all work in Windows PowerShell. For example, you can use the Windows Script Host **Scripting.FileSystem COM** class to back up C:\boot.ini to C:\boot.bak:

PowerShellCopy

(New-Object -ComObject Scripting.FileSystemObject).CopyFile('C:\boot.ini', 'C:\boot.bak')

**Creating Files and Folders**

Creating new items works the same on all Windows PowerShell providers. If a Windows PowerShell provider has more than one type of item—for example, the FileSystem Windows PowerShell provider distinguishes between directories and files—you need to specify the item type.

This command creates a new folder C:\temp\New Folder:

PowerShellCopy

New-Item -Path 'C:\temp\New Folder' -ItemType Directory

This command creates a new empty file C:\temp\New Folder\file.txt

PowerShellCopy

New-Item -Path 'C:\temp\New Folder\file.txt' -ItemType File

**Important**

When using the **Force** switch with the New-Item command to create a folder, and the folder already exists, it *won't* overwrite or replace the folder. It will simply return the existing folder object. However, if you use New-Item -Force on a file that already exists, the file *will* be completely overwritten.

**Removing All Files and Folders Within a Folder**

You can remove contained items using Remove-Item, but you will be prompted to confirm the removal if the item contains anything else. For example, if you attempt to delete the folder C:\temp\DeleteMe that contains other items, Windows PowerShell prompts you for confirmation before deleting the folder:

Copy

Remove-Item -Path C:\temp\DeleteMe

Confirm

The item at C:\temp\DeleteMe 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 do not want to be prompted for each contained item, specify the **Recurse** parameter:

PowerShellCopy

Remove-Item -Path C:\temp\DeleteMe -Recurse

**Mapping a Local Folder as a drive**

You can also map a local folder, using the New-PSDrive command. The following command creates a local drive P: rooted in the local Program Files directory, visible only from the PowerShell session:

PowerShellCopy

New-PSDrive -Name P -Root $env:ProgramFiles -PSProvider FileSystem

Just as with network drives, drives mapped within Windows PowerShell are immediately visible to the Windows PowerShell shell. In order to create a mapped drive visible from File Explorer, the parameter -Persist is needed. However, only remote paths can be used with Persist.

**Reading a Text File into an Array**

One of the more common storage formats for text data is in a file with separate lines treated as distinct data elements. The Get-Content cmdlet can be used to read an entire file in one step, as shown here:

Copy

PS> Get-Content -Path C:\boot.ini

[boot loader]

timeout=5

default=multi(0)disk(0)rdisk(0)partition(1)\WINDOWS

[operating systems]

multi(0)disk(0)rdisk(0)partition(1)\WINDOWS="Microsoft Windows XP Professional"

/noexecute=AlwaysOff /fastdetect

multi(0)disk(0)rdisk(0)partition(1)\WINDOWS=" Microsoft Windows XP Professional

with Data Execution Prevention" /noexecute=optin /fastdetect

Get-Content already treats the data read from the file as an array, with one element per line of file content. You can confirm this by checking the **Length** of the returned content:

Copy

PS> (Get-Content -Path C:\boot.ini).Length

6

This command is most useful for getting lists of information into Windows PowerShell directly. For example, you might store a list of computer names or IP addresses in a file C:\temp\domainMembers.txt, with one name on each line of the file. You can use Get-Content to retrieve the file contents and put them in the variable $Computers:

PowerShellCopy

$Computers = Get-Content -Path C:\temp\DomainMembers.txt

$Computers is now an array containing a computer name in each element.