This lesson illustrates ways to manage the file system, including file creation, reads, writes, deletes, copies, moves, and searches, as well as directory creation, listing and moves.

# Creating Files

## Creating Files Using Out-File (From Cmdlet Output)

Example:

Get-Process | Format-Table Name, Description | Out-File Proc1.txt

Get-Content Proc1.txt

Returns:

The contents of the newly created Proc1.txt file will be displayed.

This file contains the names and descriptions for all currently running processes.

## Creating Files Using Out-File (From String Output)

Example:

"Systems Checklist" | Out-File List.txt

Get-Content List.txt

Returns:

Systems Checklist

## Creating Files Using Add-Content (From Cmdlet Output)

Example:

Add-Content -Path Date.txt -Value (Get-Date)

Get-Content Date.txt

Returns:

The contents of the newly created Date.txt file will be displayed.

This file contains the output from the Get-Date cmdlet.

## Creating Files Using Add-Content (From String Output)

Example:

Add-Content -Path Private.txt -Value "Personal and Confidential"

Get-Content Private.txt

Returns:

Personal and Confidential

## Creating Files Using Set-Content

Example:

Set-Content -Path NewFile.txt -Value "This is a new file."

Get-Content NewFile.txt

Returns:

This is a new file.

## Key Observations

* Piping any command line screen output into Out-File will redirect this output from the screen to the file specified after the Out-File cmdlet. By default, if the specified file already exists its content will be overwritten by this new information.
* The Add-Content cmdlet will add the specified data to an existing file and if the specified file name does not exist then Add-Content creates it.
* When the screen output being redirected to a file is generated by a cmdlet (e.g. Get-Date) the cmdlet syntax must be enclosed in parentheses to ensure the cmdlet output is generated prior to being redirected to the file.
* Set-Content is used to create a new file and always overwrites any existing file.

# Appending Data to an Existing Text File

NOTE: To illustrate ways to append data to an existing text file the file must first be created. In the first example the command line Get-Date | Out-File is used to create a file. In the second example there doesn't need to be an existing file because Add-Content will create the file if it doesn't already exist.

## Appending Using Out-File

Example:

Get-Date | Out-File Date1.txt

"Above is the date information." | Out-File Date1.txt -Append

Get-Content Date1.txt

Returns:

October 31, 2013 12:46:47 AM

Above is the date information.

## Appending Using Add-Content

Example:

Add-Content -Path Date2.txt -Value (Get-Date)

“End of File” | Add-Content –path Date2.txt

Get-Content Date2.txt

Returns:

5/7/2010 5:46:55 PM

End of File

## Appending Using Out-File and Add-Content (Specific Text Encoding)

**Note: PowerShell version 2 behaves differently than PowerShell version 3 and up.**

Example:

“Some text” | Out-File text.txt

“Some more text” | Add-Content text.txt

Get-Content text.txt

PowerShell version 3 and 4 Returns:

Some text

Some more text

PowerShell version 2 Returns:

Some text

????????

Example:

“ASCII encoding” | Out-File text.txt –Encoding ASCII

“Unicode encoding” | Add-Content text.txt –Encoding Unicode

Get-Content text.txt

PowerShell version 3 and 4 Returns:

ASCII encoding

U n i c o d e e n c o d i n g

PowerShell version 2 Returns:

???????Unicode encoding

Example:

“ASCII encoding" | Out-File text.txt -Encoding ASCII

Get-Date | add-content text.txt

Get-Content text.txt

Returns:

ASCII encoding

2014-03-09 10:21:28 PM

Example:

Get-Date | Out-File text.txt -Append -Encoding ASCII

Get-Content text.txt

Returns:

ASCII encoding

2014-03-09 10:21:28 PM

March 9, 2014 10:22:24 PM

## Key Observations

* In the first example the -Append parameter makes Out-File append the new data to the file rather than overwrite it which is the Out-File default.
* When Add-Content is used, the data is automatically appended and does not overwrite existing data. If the specified file does not exist, then the first time Add-Content is used, it creates the specified file.
* Both the Out-File and Add-Content cmdlets have an –Encoding parameter that can be used to force a specific text encoding method.
* Switching between encoding methods can result in unreadable content.

# Displaying content and files

## Directory Listings of Files

Example:

Get-ChildItem

Returns:

Lists the files and subdirectories located in the current directory.

Example:

Get-ChildItem C:\

Returns:

Lists the files and subdirectories located in the root directory of C drive.

Example:

Get-ChildItem C:\ -Recurse

Returns:

Lists the files and subdirectories in and below the root directory of C drive.

Press Ctrl C if you wish to terminate the listing before it is finished.

Example:

Get-ChildItem C:\w\*

Returns:

Lists the files and subdirectories in the root directory of C drive that start with the letter w.

## Key Observations

* Standard wildcards can be used to control which files and directories are listed.

## Clearing (i.e. Erasing) File Content

Example:

Set-Content -Path Budget.txt -Value "Budget file"

Get-Content Budget.txt

Returns:

Budget file

Example:

Clear-Content -Path Budget.txt

Get-Content Budget.txt

Returns:

Nothing is returned because the file is empty.

Example:

Get-ChildItem Budget.txt

Returns:

Note the file Budget.txt with a 0 byte size is still in the directory listing.

## Key Observations

* Clear-Content simply deletes the contents of the specified file but does not delete the file. Clear-Content creates an empty file.
* The period at the end of the Get-ChildItem command line indicates that the current directory is the specified location. If a double dot (i.e. ..) was used then it would mean the directory immediately above the current directory.

# Deleting Files

## Deleting a Single File

Example:

Set-Content -Path OldFile.txt -Value "This is an old file."

Get-ChildItem OldFile.txt

Remove-Item OldFile.txt

Get-ChildItem OldFile.txt

Returns:

An error message stating the path does not exist.

## Deleting Multiple Files Using Wildcards

Example:

Set-Content -Path Blue.org -Value "This is the first file."

Set-Content -Path Yellow.net -Value "This is the second file."

Set-Content -Path Brown.net -Value "This is the third file."

Set-Content -Path Black.org -Value "This is the fourth file."

Get-ChildItem

Remove-Item B\*.org

Get-ChildItem

Returns:

Of the 4 files that were just created, only Blue.org and Black.org are deleted.

## Key Observations

* Standard wildcards can be used to control the deletion of multiple files.

# Creating Subdirectories

## Creating a Subdirectory Using an Absolute Path

Example:

New-Item -ItemType Directory -Path C:\Subdir1

Get-ChildItem C:\

Returns:

The C:\ directory listing should show that a subdirectory called Subdir1 was created.

## Creating a Subdirectory Using a Relative Path

Example:

New-Item -ItemType Directory -Path Subdir2

Get-ChildItem

Returns:

Get-ChildItem shows that the Subdir2 directory was created in the current directory.

## Key Observations

* In addition to creating directories, New-Item can be used to create a variety of items such as text files, logfiles, registry keys and registry values so it is important to include the -ItemType parameter to ensure the correct type of item is being created.

# Copy, Rename and Move files

## Copying a Single File

Example:

New-Item -ItemType Directory -Path C:\NewDir

“This is the file to be copied.” | Out-File File2Copy.txt

Copy-Item -Path File2Copy.txt -Destination C:\NewDir

Get-ChildItem C:\NewDir

Returns:

The C:\NewDir directory will contain a file called File2Copy.txt.

## Renaming a Copied File

Example:

New-Item -ItemType Directory -Path C:\Test

“File to be copied and renamed.” | Out-File Original.txt

Copy-Item -Path Original.txt -Destination C:\Test\NewVersion.txt

Get-ChildItem C:\Test

Returns:

The C:\Test directory will contain a file called NewVersion.txt.

## Copying Multiple Files

Example:

New-Item -ItemType Directory -Path C:\LogsBackup

Copy-Item -Path C:\Windows\\*.log -Destination C:\LogsBackup

Get-ChildItem C:\LogsBackup

Returns:

The C:\LogsBackup directory will contain a copy of all the files in the C:\Windows directory that have the file name extension of log.

## Key Observations

* Standard wildcards can be used to allow multiple files to be copied. The original file being copied is not altered in any way.
* As the second example shows, a file can be renamed during the copy process.

## Moving a Single File

Example:

New-Item -ItemType Directory -Path C:\NewLocation

“This is the file to be moved.” | Out-File File2Move.txt

Get-ChildItem

Move-Item -Path File2Move.txt -Destination C:\NewLocation

Get-ChildItem

Get-ChildItem C:\NewLocation

Returns:

File2Move.txt will no longer be listed in the current directory but will be listed in the C:\NewLocation directory.

## Moving a Directory

Example:

New-Item -ItemType Directory -Path C:\MovedLogs

New-Item -ItemType Directory -Path C:\TempLogs

Copy-Item C:\Windows\\*.log -Destination C:\TempLogs

Get-ChildItem C:\TempLogs

Get-ChildItem C:\MovedLogs

Move-Item -Path C:\TempLogs -Destination C:\MovedLogs

Get-ChildItem C:\TempLogs

Get-ChildItem C:\MovedLogs

Get-ChildItem C:\MovedLogs\TempLogs

Returns:

When a directory is moved as shown in the second example, all of the files and directories in and below that directory are also moved to the specified destination.

## Key Observations

* Standard wildcards can be used to allow multiple files to be moved. The original files and directories that are moved will no longer exist in their original location.
* Remember that moving a directory will always result in all of that directory's content being moved as well.

# NTFS Alternate Data Streams

NTFS files can contain more than one data attribute. The primary data attribute is called $data and contains the content of the file. Alternate data streams have been used to hide data and even conceal malware. They are frequently used to identify the source of a downloaded file.

In order to demonstrate alternate data streams, open Chrome, Internet Explorer, Edge or FireFox and download the following file. Save the file to your Downloads folder in your user profile.

<https://download.sysinternals.com/files/SysinternalsSuite.zip>

## Listing Alternate Data Streams

Example:

Get-Item $env:UserProfile\Downloads\SysinternalsSuite.zip

Returns:

The Get-Item command will list details about the SysinternalsSuite.zip file you downloaded.

Example:

Get-Item -stream \* $env:UserProfile\Downloads\SysinternalsSuite.zip

Returns:

The –stream \* parameter will list details of the file’s data streams. This file contains the $Data stream, the content of the file and a stream called Zone.Identifier with a size of 97 bytes.

Example: (The following must be entered as a single command)

Invoke-WebRequest -Uri “https://download.sysinternals.com/files/SysinternalsSuite.zip” -OutFile $env:\UserProfile\Downloads\outfile.zip

Example:

Get-Item -stream \* $env:UserProfile\Downloads\outfile.zip

Returns:

This file contains only the $Data stream. The web browser adds the Zone.Identifier stream to the file.

## Reading Alternate Data Stream Content

Example: (The following must be entered as a single command)

Get-Content -Stream Zone.Identifier $env:UserProfile\Downloads\SysinternalsSuite.zip

Returns:

[ZoneTransfer]

ZoneId=3

HostUrl=https://download.sysinternals.com/files/SysinternalsSuite.zip

## Creating Alternate Data Streams

Example:

$StreamData=Get-Content -Stream Zone.Identifier '$Env:UserProfile\Downlaods\SysinternalsSuite.zip'

Add-Content –Stream Zone.Identifier $Env:UserProfile\Downloads\outfile.zip –Value $StreamData

Get-Item –stream \* $Env:UserProfile\Downloads\outfile.zip

Get-Content -Stream Zone.Identifier $Env:UserProfile\Downloads\outfile.zip

Returns:

The first command will store the contents of the Zone.Identifier data stream from the SystinternalsSuite.zip file downloaded using a web browser to the $StreamData variable.

The Add-Content command will create a data stream called Zone.Identifier for the outfile.zip file downloaded using the Invoke-WebRequest cmdlet and write the content of the $StreamData variable to the data stream.

The Get-Item cmdlet will list the details of the new data stream and the Get-Content cmdlet will display the contents of the new Zone.Identifier data stream.

## Deleting Alternate Data Streams

Example:

Remove-Item -Stream Zone.Identifier $Env:Userprofile\Downloads\outfile.zip

Get-Item –stream \* $Env:UserProfile\Downloads\outfile.zip

Returns:

The Zone.Identifier data stream has been deleted.

## Key Observations

* NTFS file system supports multiple data streams. Alternate data streams are not visible by default to Windows built-in commands but can be manipulated using PowerShell and some third party tools.
* The Get-Item –stream parameter is used to list a file’s data streams.
* The Get-Content and Add-Content cmdlets can be used to read and write data streams.
* Remove-Content can be used to delete data streams.
* It is important to note, data streams are not returned by the Get-ChildItem cmdlet.

# Searching File Content

## Searching Using Get-Content (Case-Insensitive)

Example:

Get-Service | Out-File CurrentServices.txt

Get-Content CurrentServices.txt | select-string Windows

Returns:

All lines in the CurrentServices.txt file that contain the string Windows are listed.

## Searching Using Get-ChildItem (Case-Insensitive)

Example:

Get-Service | Out-File NewServices.txt

Get-Content NewServices.txt | select-string windows

Get-ChildItem NewServices.txt | select-string windows

Returns:

All lines in the NewServices.txt file that contain the string windows are listed. Note that each line is preceded by the file name and the line number where the string is located using the format Filename:LineNumber:.

## Case-Sensitive Searching

Example:

Get-Service | Out-File ServiceUpdate.txt

Get-Content ServiceUpdate.txt | select-string Windows -CaseSensitive

Get-Content ServiceUpdate.txt | select-string windows -CaseSensitive

Returns:

All lines in the ServiceUpdate.txt file that contain the string Windows with an exact case match are listed. The example where Windows is entirely in lowercase will not list the lines where the string Windows contains any capital letters.

## Key Observations

* Select-String is case-insensitive by default. Add the -CaseSensitive parameter if case is important.
* There will be times when you want to know the line number where the string is located. For example, you may want to store the file into an array and knowing the line number would allow the contents of specific lines to be manipulated since you can obtain the line's index number.
* In such cases using Select-String with Get-ChildItem is the best option.
* There will also be times when just the matching data is required and the filename and line number are not required. In such cases using Select-String with Get-Content is the best option.

# Working with CSV Files

## Export-CSV (Example 1)

Example:

Get-Service | Export-CSV Services.csv

Returns:

Start Microsoft Excel and open Services.csv. You will see the same data that is displayed when you enter Get-Service. CSV stands for Comma Separated Value. Close Excel and open Services.csv in a basic text editor such as WordPad. Note that information about each service is displayed with each property separated by a comma. (thus the name Comma Separated Value)

## Export-CSV (Example 2)

Example:

Get-WMIObject Win32\_SystemDriver | Select-Object -Property DisplayName,`

Status, PathName | Export-CSV Drivers.csv -NoTypeInformation

Returns:

When you open Drivers.csv in Excel you will see the three properties for each driver with each property in its own column.

The system object time information will not be output to the file.

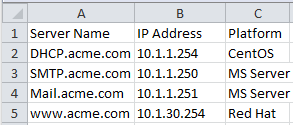
## Key Observations

* The CSV (Comma Separated Value) file format is a very popular format for data that is to be transferred from one type of application to another because this format strips away everything except the raw data which minimizes complications when the data is imported into other applications.
* Excel is one of the applications that can work with .CSV files. Many PowerShell cmdlets access resources that contain numeric values. Export-CSV makes it easy to transfer this data to Excel where powerful spreadsheet tools can be used to process this data (e.g. create graphs, perform computations).
* By default Export-CSV places a comma between each property value which acts as a delimiter to indicate the end of that property value. Export-CSV has a -Delimiter parameter that allows you to change the delimiter to another keystroke or series of keystrokes if need be.
* The –NoTypeInformation parameter will strip out the object type information. Without this parameter the first line of the file will contain a description of the object. For example,

#TYPE Selected.System.Management.ManagementObject

## Importing CSV Files

Before demonstrating how the Import-CSV cmdlet is used you need to create a CSV file using Excel. Start Excel and create a file called ServerInfo.csv. Be sure to save the file in the same drive and directory where you are running this script. Save the file in CSV (Comma delimited) format. The row and column locations are shown here:



## Importing A CSV File

Example:

Import-Csv ServerInfo.csv

Returns:

The screen output will match the data that you entered into the spreadsheet.

## Storing A CSV File To A Variable

Example:

$ServerData = Import-Csv ServerInfo.csv

Returns:

Like all other situations where a variable is created there is no screen output.

## Accessing Imported CSV Data

Example:

$ServerData | Get-Member

$ServerData | Select-Object "Server Name"

Returns:

Again like all other situations the variable has properties which can be managed in all of the standard ways.

## Key Observations

* If you look at the output from the $ServerData | Get-Member example you will see that all of the properties are strings.
* If a property heading of an imported CSV file contains spaces as in the case with the last example above (i.e. Server Name) the property name must be placed in quotes.
* CSV files are very useful for the storage and subsequent retrieval of system information. A system administrator will find all sorts of uses for both the Import-CSV and Export-CSV cmdlets.