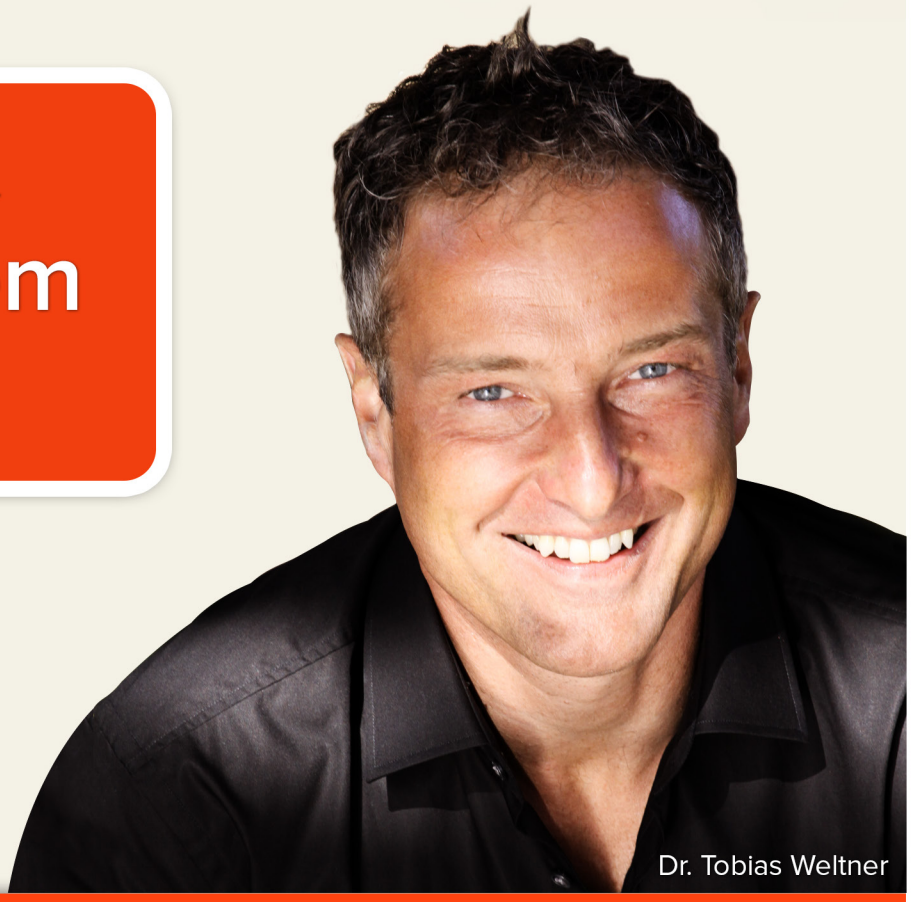
# PowerTips MONTHLY

Part of the PowerShell.com reference library, brought to you by **Dr. Tobias Weltner**

Vol. 1 | June 2013

This Month's Topic:

## File System Tasks



Dr. Tobias Weltner

## Table of Contents



## 1. Resolving Paths

Paths can be relative, such as ". \file.txt". To resolve such a path and display its full path, you could use Resolve-Path:

```
PS> Resolve-Path .\file.txt
```

Unfortunately, though, Resolve-Path expects the file to really exist, so you cannot use it on hypothetical paths. To resolve all paths, whether they exist or not, use this line instead:

```
PS> $ExecutionContext.SessionState.Path.GetUnresolvedProviderPathFromPSPath('.\file.txt')
C:\Users\Tobias\file.txt
```

## 2. Sophisticated Directory Filtering in PowerShell 3.0

In PowerShell 3.0, Get-ChildItem now supports sophisticated filtering through its -Attributes parameter. To get all files in your Windows folder or one of its subfolders that are not system files but are encrypted or compressed, use something like this:

```
Get-ChildItem $env:windir -Attributes !Directory+!System+Encrypted,!Directory+!System+Compressed
-Recurse -ErrorAction SilentlyContinue
```

(Note how "`!`" negates the filter.)

The `-Attributes` parameter supports these attributes: Archive, Compressed, Device, Directory, Encrypted, Hidden, Normal, NotContentIndexed, Offline, ReadOnly, ReparsePoint, SparseFile, System, and Temporary.

## 3. Finding Files Only or Folders Only

In PowerShell 2.0, to list only files or only folders you had to do filtering yourself:

```
Get-ChildItem $env:windir | Where-Object { $_.PSIsContainer -eq $true }
Get-ChildItem $env:windir | Where-Object { $_.PSIsContainer -eq $false }
```

In PowerShell 3.0, Get-ChildItem is smart enough to do that for you:

```
Get-ChildItem $env:windir -File
Get-ChildItem $env:windir -Directory
```

## 4. Adding Personal Drives in PowerShell

PowerShell enables you to create custom drives with custom drive names (New-PSDrive). Here's a piece of code that you can place into your profile script (path to this script is found in $profile, and the profile script may be created first).

It adds useful system folders as named PowerShell drives to your PowerShell environment:

```
Function Add-PersonalDrive
{
  [System.Enum]::GetNames([System.Environment+SpecialFolder]) |
  ForEach-Object {
    $name = $_
    $target = [System.Environment]::GetFolderPath($_)
    New-PSDrive $name FileSystem $target -Scope Global
  }
}
```

Once you run the function, you get a whole lot of new drives, for example Desktop: or MyDocuments:

```
PS> Add-PersonalDrive

Name                   Used (GB) Free (GB) Provider   Root
----                   --------- --------- --------   ----
Desktop                              2,39 FileSystem C:\Users\Tobias\Desktop
Programs                             2,39 FileSystem C:\Users\Tobias\AppData\Roaming\...
MyDocuments                          2,39 FileSystem C:\Users\Tobias\Documents
Personal                             2,39 FileSystem C:\Users\Tobias\Documents
(...)
```

## 5. Discovering File System-related Cmdlets

Here's a simple line that returns all file system-related cmdlets:

```
Get-Command -Noun item*, path
```

Many of these cmdlets have historic aliases that will help you guess what they are doing:

```
Get-Alias -Definition *-item*, *-path* |
  Select-Object Name, Definition |
  Out-GridView
```

---

## Author Bio

Tobias Weltner is a long-term Microsoft PowerShell MVP, located in Germany. Weltner offers entry-level and advanced PowerShell classes throughout Europe, targeting mid- to large-sized enterprises. He just organized the first German PowerShell Community conference which was a great success and will be repeated next year (more on www.pscommunity.de).. His latest 950-page "PowerShell 3.0 Workshop" was recently released by Microsoft Press.

To find out more about public and in-house training, get in touch with him at tobias.weltner@email.de.

## 6. Clean Your Temp Folder with PowerShell

When disk space gets low, you may want to clean up your temporary folder. The code deletes all files that are older than 30 days to make sure you're not dumping anything that's still needed:

```
$cutoff = (Get-Date) - (New-TimeSpan -Days 30)

$before = (Get-ChildItem $env:temp | Measure-Object Length -Sum).Sum

Get-ChildItem $env:temp |
  Where-Object { $_.Length -ne $null } |
  Where-Object { $_.LastWriteTime -lt $cutoff } |
  # simulation only, no files and folders will be deleted
  # replace -WhatIf with -Confirm to confirm each delete
  # remove -WhatIf altogether to delete without confirmation (at your own risk)
  Remove-Item -Force -ErrorAction SilentlyContinue -Recurse -WhatIf

$after = (Get-ChildItem $env:temp | Measure-Object Length -Sum).Sum
$freed = $before - $after

'Cleanup freed {0:0.0} MB.' -f ($freed/1MB)
```

Since deleting stuff is always risky, we left a -WhatIf in the code so you can check that you are actually deleting your temp folder and not anything else (due to a typo for example). Once you are comfortable, remove -WhatIf to invoke the cleanup process automatically, or replace -WhatIf by -Confirm to confirm each delete manually.

You may be surprised how much garbage can be removed.

```
PS> C:\Users\Tobias\Documents\temp\Untitled2.ps1
WARNING: Size of TEMP folder is currently 879,02 MB
WARNING: Freed 329,27 MB disk space
```

## 7. Unblocking and Unpacking ZIP Files

Before you can unpack ZIP files downloaded from the Internet, you will want to unblock them - or else executable files like *.exe or *.dll will not work. Here is a script that uses PowerShell 3.0's new Unblock-File cmdlet to first unblock a ZIP file, then uses the File Explorer-integrated ZIP functionality to unpack its contents:

Note that this requires the built-in Windows ZIP support to be present and not replaced with other ZIP tools.

```
$Source = 'c:\some\zipfile.zip'
$Destination = 'c:\unpackfolder'  # this folder MUST exist

Unblock-File $Destination     # use this in PowerShell 3.0 to unblock downloaded data
                              # remove this in PowerShell 2.0 and unblock manually if needed

$helper = New-Object -ComObject Shell.Application
$files = $helper.NameSpace($Source).Items()
$helper.NameSpace($Destination).CopyHere($files)
```

## 8. Find Open Files

To find open files on a remote system, use openfiles.exe and convert the results to rich objects. Here is a sample (replace “storage1” with the name of a remote computer you have access permissions):

```
PS> openfiles /Query /S storage1 /FO CSV /V | ConvertFrom-Csv | Out-GridView
```

Openfiles.exe cannot find open files on the local machine (unless you configure your local machine to monitor open files).

## 9. Finding Newest or Oldest Files

In PowerShell 3.0, Measure-Object can be applied not just to numeric data but to anything that is comparable. In PowerShell 2.0, this line would return the smallest and largest file size in your Windows folder:

```
Get-ChildItem $env:windir |
  Measure-Object -Property Length -Minimum -Maximum |
  Select-Object -Property Minimum,Maximum
```

In PowerShell 3.0, you could also measure properties like LastWriteTime, telling you the oldest and newest dates:

```
Get-ChildItem $env:windir |
  Measure-Object -Property LastWriteTime -Minimum -Maximum |
  Select-Object -Property Minimum,Maximum
```

Or, you could get the minimum and maximum start times of all the running processes. Make sure you use Where-Object to exclude any process that has no StartTime value:

```
Get-Process |
  Where-Object StartTime |
  Measure-Object -Property StartTime -Minimum -Maximum |
  Select-Object -Property Minimum,Maximum
```

## 10. Finding Duplicate Files

Hash tables are a great way to find duplicates. Simply use the hash table as lookup to see if the file (or element) was already added to the hash table. The following script would find all files and folder with same name in the current folder, the Windows folder and its System32 subfolder.

```
Function Find-DuplicateName
{
  $Input | ForEach-Object {
    if ($lookup.ContainsKey($_.Name))
    {
      '{0} ({1}) already exists in {2}.' -f $_.Name, $_.FullName, $lookup.$($_.Name)
    }
    else
    {
      $lookup.Add($_.Name, $_.FullName)
    }
  }
}
```

## Technical Editor Bio

Aleksandar Nikolic, Microsoft MVP for Windows PowerShell, a frequent speaker at the conferences (Microsoft Sinergija, PowerShell Deep Dive, NYC Techstravaganza, KulenDayz, PowerShell Summit) and the cofounder and editor of the PowerShell Magazine (http://powershellmagazine.com). You can find him on Twitter: https://twitter.com/alexandair

```
$lookup = @{}
Get-ChildItem $home | Find-DuplicateName
Get-ChildItem $env:windir | Find-DuplicateName
Get-ChildItem $env:windir\system32 | Find-DuplicateName
```

## 11. Finding Old Files

Occasionally, you might want to find files that are older than a give number of days to delete or backup those. A simple filter can provide that functionality:

```
Filter Filter-Age($Days=30)
{
  if (($_.CreationTime -le (Get-Date).AddDays($days * -1) )) {$_}
}
```

Pipe the result of a Dir into the Filter-Age filter, and it will only let those files and folders pass that are at least the specified number of days old. The following line finds all logfiles in your Windows folder that are at least 10 days old:

```
Dir $env:windir *.log -Recurse -ea 0 | Filter-Age -Days 10
```

You could easily delete or backup the resulting files. This would delete them:

```
Dir $env:windir *.log -Recurse -ea 0 | Filter-Age -Days 10 | Del -WhatIf
```

## 12. Finding System Folders

You may want to know where special folders such as MyPictures or Documents are located. The Environment .NET class provides a static method named GetFolderPath() which provides this information. To find the location of your desktop, for example, use this:

```
[Environment]::GetFolderPath('Desktop')
```

Simply specify an invalid keyword, and the exception will list all valid keywords:

```
PS> [Environment]::GetFolderPath('give me more!')
Cannot convert argument "folder", with value: "give me more!", for "GetFolderPath" to type "System.
Environment+SpecialFolder": "Cannot convert value
"give me more!" to type "System.Environment+SpecialFolder". Error: "Unable to match the identifier
name give me more! to a valid enumerator name.
Specify one of the following enumerator names and try again: Desktop, Programs, MyDocuments,
Personal, Favorites, Startup, Recent, SendTo, StartMenu,
MyMusic, MyVideos, DesktopDirectory, MyComputer, NetworkShortcuts, Fonts, Templates,
CommonStartMenu, CommonPrograms, CommonStartup,
CommonDesktopDirectory, ApplicationData, PrinterShortcuts, LocalApplicationData, InternetCache,
Cookies, History, CommonApplicationData, Windows,
System, ProgramFiles, MyPictures, UserProfile, SystemX86, ProgramFilesX86, CommonProgramFiles,
CommonProgramFilesX86, CommonTemplates,
CommonDocuments, CommonAdminTools, AdminTools, CommonMusic, CommonPictures, CommonVideos, Resources,
LocalizedResources, CommonOemLinks, CDBurning""
```

## 13. Hiding Drive Letters

Sometimes you may want to hide drive letters in Explorer from users. There's a Registry key that can do this for you. It takes a bit mask where each drive has a bit. When the bit is set, the drive is hidden. Here's the function that automatically manages the bitmasks and registry entries to hide selected drive letters:

```
Function Hide-Drive
{
param
  (
$DriveLetter
  )

$key = @{
```

```
    Path = 'HKCU:\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer'
    Name = 'NoDrives'
  }

if ($DriveLetter -eq $null)
  {
Remove-ItemProperty @key
  }
else
  {
$mask = 0
$DriveLetter |
ForEach-Object { $_.toUpper()[0] } |
Sort-Object |
ForEach-Object { $mask += [Math]::Pow(2,(([Byte]$_) -65)) }

Set-ItemProperty @key -Value $mask -type DWORD
  }
}
```

For example, to hide drives A, B, E, and Z, you would use it like this:

```
PS> Hide-Drive A,B,E,Z
```

To display all drives, call Hide-Drive without any argument:

```
PS> Hide-Drive
```

Note that you need to have administrative privileges to change policies, and that policies may be overridden by group policy settings set up by your corporate IT.

For the changes to take effect, you need to log off and on again or kill your explorer.exe and restart it.

## 14. Checking Total Size of Downloads-Folder

Whenever you download something with IE, by default the files get stored in your personal download folder. Often, over time a lot of garbage can accumulate. This line tells you just how much data you store in this folder:

```
$folder = "$env:userprofile\Downloads"
Get-ChildItem -Path $folder -Recurse -Force -ea 0 |
  Measure-Object -Property Length -Sum |
  ForEach-Object {
    $sum = $_.Sum / 1MB
"Your Downloads folder currently occupies {0:#,##0.0} MB storage" -f $sum
}
```

You may be surprised just how much stuff has been collected there.

## 15. Sharing Folders

Console commands are first class PowerShell citizens, so sometimes it may be easier to use classic console commands to solve a problem. Here is a function that creates a local folder and also shares it so others can use it via network. Just be aware that the net.exe used to share the folder requires you to have full admin privileges:

```
Function New-Share
{
  param
  (
    $Path,

    $Name
  )

try {
$ErrorActionPreference = 'Stop'

if ((Test-Path $Path) -eq $false)
```

```
  {
    $null = New-Item -Path $Path -ItemType Directory
  }

  net.exe share $Name=$Path
}
catch {
   Write-Warning "Create Share: Failed, $_"
 }
}
```

And this is how you share new folders:
PS> New-Share c:\myfolder sharedplace
sharedplace was shared successfully.

## 16. Shrinking Paths

Many file-related .NET Framework methods fail when the overall path length exceeds a certain length. Use low-level methods to convert lengthy paths to the old 8.3 notation which is a lot shorter in many cases:

```
Function Get-ShortPath
{
param
(
  [Parameter(Mandatory=$true)]
  $Path
)
  $code = @'
[DllImport("kernel32.dll", CharSet = CharSet.Auto, SetLastError=true)]
public static extern uint GetShortPathName(string longPath,
StringBuildershortPath,uint bufferSize);
'@
  $API = Add-Type -MemberDefinition $code -Name Path -UsingNamespace System.Text -PassThru
  $shortBuffer = New-Object Text.StringBuilder ($Path.Length * 2)
  $rv = $API::GetShortPathName( $Path, $shortBuffer, $shortBuffer.Capacity )

  if ($rv -ne 0)
  {
    $shortBuffer.ToString()
  }
  else
  {
    Write-Warning "Path '$path' not found."
  }
}
```

Here is how you can use the new function:

```
PS> $null = md c:\thisIsALongName\VeryLongPath\MayExceed260chars -ea 0

PS> Get-ShortPath
'c:\thisIsALongName\VeryLongPath\MayExceed260chars'

c:\THISIS~1\VERYLO~1\MAYEXC~1
```

## 17. How Large Are My Folders?

To find out the total size of all subfolders in a directory, try this function:

```
Function Get-FolderSize
{
param
  (
$Path=$home
  )
```

```
$code = { ('{0:#,##0.0} MB' -f ($this/1MB)) }

Get-ChildItem -Path $Path|
Where-Object { $_.PSIsContainer } |
ForEach-Object {
Write-Progress -Activity 'Calculating Total Size for:' -Status $_.FullName
$sum = Get-ChildItem $_.FullName -Recurse -ErrorAction SilentlyContinue |
Measure-Object -Property Length -Sum -ErrorAction SilentlyContinue
$bytes = $sum.Sum

if ($bytes -eq $null)
    {
  $bytes = 0
    }

$result = 1 | Select-Object -Property Path, TotalSize
$result.Path = $_.FullName
$result.TotalSize = $bytes |
Add-Member -MemberType ScriptMethod -Name toString -Value $code -Force -PassThru
$result
  }
}
```

And this is what a result would typically look like:

```
PS> Get-FolderSize $env:windir

Path                                          TotalSize
----                                          ----
C:\Windows\addins                             0,0 MB
C:\Windows\AppCompat                          0,0 MB
C:\Windows\apppatch                           11,8 MB
C:\Windows\assembly                           2.279,4 MB
C:\Windows\AUInstallAgent                     0,0 MB
C:\Windows\Boot                               36,8 MB
C:\Windows\Branding                           2,2 MB
C:\Windows\BrowserChoice                      1,1 MB
(...)
```

## 18. Bulk-Renaming Files

Rename-Item can rename hundreds of files in one step. Have a look:

```
$global:i = 1
Get-ChildItem -Path c:\test1\ -Filter *.jpg |
  Rename-Item -NewName { "picture_$i.jpg"; $global:i++}
```

This code assumes there is a folder called `c:\test1` that contains a number of `*.jpg` files. Get-ChildItem gets all the `*.jpg` files and pipes them to Rename-Item. Rename-Item then renames all of these files as "`picture_X.jpg`" where X is an incrementing number.

## 19. Monitoring Folder Content

You can use a FileSystemWatcher object to monitor a folder and write a log for all newly created files. Here is a script that demonstrates this.

```
# make sure this folder exists. Script will monitor changes to this folder:
$folder = 'C:\SomeFolder'
$timeout = 1000

$FileSystemWatcher = New-Object System.IO.FileSystemWatcher $folder

Write-Host "Press CTRL+C to abort monitoring $folder"
while ($true) {
   $result = $FileSystemWatcher.WaitForChanged('all', $timeout)

   if ($result.TimedOut -eq $false)
  {
    Write-Warning ('File {0} : {1}' -f $result.ChangeType, $result.name)
  }
}

Write-Host 'Monitoring aborted.'
```

It monitors the folder c:\somefolder (which you should create before you run this script). Next, while the script is running, any change to the folder is reported. For example, if you open the monitored folder in Windows Explorer, add a new text file, rename it and then delete it, here's the result:

```
PS> C:\testscript.ps1
Press CTRL+C to abort monitoring C:\Users\Tobias
WARNING: File Created : New Text Document.txt
WARNING: File Renamed : test.txt
WARNING: File Deleted : test.txt
```

## 20. Open File Exclusively

To open a file in a locked state so no one else can open, access, read, or write the file, you can use the low-level .NET methods like this:

```
$path = "$env:temp\somefile.txt" # MUST EXIST!

if ( (Test-Path$path) -eq$false)
{
Set-Content-Value'test'-Path$path
}

$file = [System.io.File]::Open($path, 'Open', 'Read', 'None')
$reader = New-Object System.IO.StreamReader($file)
$text = $reader.ReadToEnd()
$reader.Close()
Read-Host 'Press ENTER to release file!'
$file.Close()
```

This will lock a file and read its content. To illustrate the lock, the file will remain locked until you press ENTER.

## 21. Removing File Extensions

To remove file extensions, use .NET methods:

```
PS> [system.io.path]::GetFileNameWithoutExtension('c:\test\report.txt')
```

## 22. Quickly Changing File Extensions

If you want to quickly exchange a file extension to create a “bak” backup version or generate a new file name for output, you should use the ChangeExtension() method:

```
$oldpath = ‘c:\test\datei.csv’
$newpath = [System.IO.Path]::ChangeExtension($oldpath, ‘.xls’)
$oldpath
$newpath
```

## 23. Grouping Files Based on Size

Group-Object can auto-create hash tables so that you can easily create groups of objects of a kind.
Here is an example:

```
$criteria = {
if ($_.Length -lt 1KB) {
    ‘tiny’
} elseif ($_.length -lt 1MB) {
    ‘average’
} else {
    ‘huge’ }
}

$myFiles = dir $env:windir | Group-Object -Property $criteria -asHash -asString
```

Next, you can output the results, or use your keywords to access the file groups:

```
PS> $myFiles

Name                           Value
----                           -----
huge                           {explorer.exe, WindowsUpdate.log}
tiny                           {addins, AppCompat, AppPatch, assembly...}
average                        {bfsvc.exe, bootstat.dat, ColorPicker for PowerPoint Setup Log.txt,
DPINST.LOG...}



PS> $myFiles.huge


    Directory: C:\Windows


Mode                 LastWriteTime     Length Name
----                 -------------     ------ ----
-a---        25.02.2011     07:19    2871808 explorer.exe
-a---        20.02.2013     13:21    1975208 WindowsUpdate.log
PS>
```

## 24. Open Many Files with One Command

To quickly open all files of a kind, such as all text or script files found in a folder, you should try the Open-File function. It will accept a simple pattern, such as *.txt which opens all text files, or a*.ps1 which opens all PowerShell scripts that start with “a”:

```
function Open-file{
param(
[Parameter(Mandatory=$true)]
$path
  )
$paths = Resolve-Path $path -ea SilentlyContinue
```

```
if ($paths -ne $null) {
$paths | Foreach-Object { Invoke-Item $_ }
  } else {
“No file matched $path.”
  }
}
```

The next command then would open all text files in your current folder with the program that is associated with .txt-files (like Notepad editor):

PS > Open-File *.txt

## 25. Use Multiple Wildcards

Did you know that you can use multiple wildcards in paths? This will give you a lot of control.
This line will find all DLL files in all subfolders up to two levels below your Windows folder (it will take a couple of seconds for Resolve-Path to gather all files):

```
Resolve-Path $env:windir\*\*\*.dll -ea 0
```

And this line will lists the Desktop folder content for all user accounts on your computer—provided that you have sufficient privileges:

```
dir c:\users\*\desktop\*
```

## 26. Filtering Multiple File Types

If you want to filter files based on multiple extensions, you can use this filter:

```
Filter where-Extension
{
  param
  (
   [String[]]
   $extension = (‘.bmp’, ’.jpg’, ’.wmv’)
  )

$_ |
Where-Object {
  $extension -contains $_.Extension
  }
}
```

To find all *.log-files and all *.txt-files in your Windows folder, use it like this:

Dir $env:windir -Recurse -ea 0 | Where-Extension .log,.txt

## 27. Creating Shortcuts on your Desktop

PowerShell and .NET can do amazing things, but they are not good at creating shortcuts. However, you can create a shortcut on your Desktop with just a few lines of code using COM objects instead:

```
$shell = New-Object -ComObject WScript.Shell
$desktop = [System.Environment]::GetFolderPath(‘Desktop’)
$shortcut = $shell.CreateShortcut(“$desktop\clickme.lnk”)
$shortcut.TargetPath = ”notepad.exe”
$shortcut.IconLocation = ”shell32.dll,23”
$shortcut.Save()
```

## 28. Create Files and Folders in One Step

Use New-Item like this when you want to create a file plus all the folders necessary to host the file:

```
PS> New-Item -Path c:\subfolder\anothersubfolder\yetanotherone\test1.txt -Type File -Force
```

This will create the necessary folders first and then insert a blank file into the last folder. You can then edit the file easily using Notepad. The magic is done by -Force, but this parameter will also overwrite the file if it already exists. You may want to use Test-Path first to check whether the file exists if you must make sure it won't get overwritten.

## 29. Remove Recents Folder

Windows uses the special recent folder to remember which files you have opened. You can have a look at the information Windows has accumulated. Remove the -WhatIf parameter if you want to get rid of the files:

```
Get-ChildItem ([Environment]::GetFolderPath("Recent")) | Remove-Item -Recurse -WhatIf
```

## 30. Displaying Hex Dumps

PowerShell can read plain text, but it can also read binary content. Here is a little function that creates a "hex dump" of any binary file:

```
Function Get-HexDump
{
param
  (
$path,

$width=10,

$bytes=-1
  )

$OFS=''
Get-Content -Encoding byte $path -ReadCount $width `
-TotalCount $bytes | ForEach-Object {
$byte = $_

if (($byte -eq 0).count -ne $width)
    {
$hex = $byte | ForEach-Object {
' ' + ('{0:x}'-f $_).PadLeft(2,'0')}
$char = $byte | ForEach-Object {
if ([char]::IsLetterOrDigit($_))
{
  [char] $_
 }
 else
 {
        '.'
 }
}
"$hex $char"
    }
  }
}
```

And this is an example on how to call the function:

```
PS> Get-HexDump $env:windir\explorer.exe -width 15 -bytes 150
 4d 5a 90 00 03 00 00 00 04 00 00 00 ffff 00 MZ.....ÿÿ.
00 b8 00 00 00 00 00 00 00 40 00 00 00 00 00 .........@.....
 e0 00 00 00 0e 1f ba 0e 00 b4 09 cd 21 b8 01 à.....°....Í...
4c cd 21 54 68 69 73 20 70 72 6f 67 72 61 6d LÍ.This.program
20 63 61 6e 6e 6f 74 20 62 65 20 72 75 6e 20 .cannot.be.run.
69 6e 20 44 4f 53 20 6d 6f 64 65 2e 0d 0d 0a in.DOS.mode....
 24 00 00 00 00 00 00 00 93 83 28 37 d7 e2 46 .........7.âF
 64 d7 e2 46 64 d7 e2 46 64 de 9a c2 64 9d e2 d.âFd.âFdÞ.Âd.â
PS>
```

## 31. Reading File "Magic Number"

File types are not entirely dependent on file extension. Rather, binary files have internal ID numbers called "magic numbers" that tell Windows what type of file it is. Here is a function to read and display the magic number:

```
Function Get-MagicNumber
{
param
  (
$path
  )

Resolve-Path $path | ForEach-Object {
$magicnumber = Get-Content -Encoding byte $_ -ReadCount 4 -TotalCount 4
$hex1 = ('{0:x}' -f ($magicnumber[0] * 256 + $magicnumber[1])).PadLeft(4,'0')
$hex2 = ('{0:x}' -f ($magicnumber[2] * 256 + $magicnumber[3])).PadLeft(4,'0')
[string] $chars = $magicnumber|ForEach-Object{
if ([char]::IsLetterOrDigit($_))
{
  [char] $_
}
else
{
  '.'
}
}
"{0} {1} '{2}'" -f $hex1, $hex2, $chars }
}
```

Executable files use the two letters "MZ" (which coincidentally are the initials of Mark Zbikowski, one of the original developers of MS DOS). Data files use different signatures.

```
PS> Get-MagicNumber C:\Windows\explorer.exe
4d5a 9000 'M Z . .'

PS> Get-MagicNumber C:\Windows\bootstat.dat
2000 0000 '. . . .'

PS> Get-MagicNumber C:\Windows\notepad.exe
4d5a 9000 'M Z . .'
```

## 32. Rename Drive Label

WMI can read any drive label (the name that appears next to a drive inside Explorer), and you can change the drive label, too—provided you have administrator privileges. This code renames drive c:\ to "My Harddrive":

```
$drive = [wmi]"Win32_LogicalDisk='C:'"
$drive.VolumeName = "My Harddrive"
$drive.Put()
```

Just make sure to call Put() so your changes will get written back to the original object.

## 33. No Need for Virtual Drives

You do not need to use drive letters to access information provided by PowerShell providers. For example, use this to list the HKEY_CLASSES_ROOT in your Registry:

```
Dir Registry::HKEY_CLASSES_ROOT
```

By pre-pending the path with the provider name that PowerShell should use, you no longer need drive letters. This is basically the same as adding a new virtual drive and specifying the provider PowerShell should use:

```
New-PSDrive HKCR Registry HKEY_CLASSES_ROOT
Dir HKCR:
```

## 34. Temporary File Name

Thanks to Get-Date, you can easily create unique temporary file names with a timestamp:

```
(Get-Date -format 'yyyy-MM-ddhh-mm-ss') + '.tmp'
```

## 35. Creating Large Dummy Files with .NET

If you need to create large dummy files, the following code is a very fast way to generate really large test files. This for example creates a 1GB file in a fraction of a second:

```
$path = "$env:temp\testfile.txt"
$file = [io.file]::Create($path)
$file.SetLength(1gb)
$file.Close()
Get-Item $path
```

## 36. Does a Folder Contain a Specific File?

Test-Path supports wildcards so if you'd like to know whether there are any PowerShell script files located in your home folder, try this:

```
Test-Path $home\*.ps1
```

To get the actual number of PowerShell scripts, use Get-Childitem and count the result:

```
@(Get-ChildItem $home\*.ps1).Count
```

Note the @() converts any result into an array so even if there is only one file or none, the result will be an array so you can safely query its Count property. This step isn't necessary anymore in PowerShell 3.0, but it won't do any harm there and keeps your code compatible.

## 37. File or Folder? Find Out!

Test-Path can check whether a file or folder exists, but this does not tell you whether the path specified was actually a file or a folder. If you'd like to check whether the folder c:\test exists, use this:

```
Test-Path c:\test -PathType Container
```

If c:\test was a file (without extension), Test-Path would still return false because with the -PathType parameter, you explicitly looked for folders. Likewise, if you want to explicitly check for files, use the parameter -PathType leaf.

## 38. List Hidden Files

Did you notice that Dir, ls or Get-ChildItem do not return hidden files?
To see hidden files, you need to specify the -Force parameter:

```
Get-ChildItem $env:windir -Force
```

But what if you just wanted to see hidden files only? Filter the result, for example like this:

```
Get-ChildItem $env:windir -Force | Where-Object { $_.Mode -like '*h*' }
```

In PowerShell 3.0, you can also use:

```
Get-ChildItem $env:windir-Attributes h
```

## 39. Converting File System to NTFS

You probably know about the convert.exe utility which can convert a file system to NTFS without data loss. This tool cannot be run automatically though because it wants a manual confirmation and asks for the drive label of the drive you want to convert.
Here is an example how PowerShell can embrace convert.exe and turn it into a more sophisticated tool that converts drives without manual confirmation. Of course, this imposes risk because converting a file system cannot be undone.

```
Function ConvertTo-NTFS
{
param
  (
$letter='C:'
  )
if (!(Test-Path $letter))
  {
Throw "Drive $letter does not exist."
  }

$drive = [wmi]"Win32_LogicalDisk='$letter'"
$label = $drive.VolumeName
$filesystem = $drive.FileSystem

if ($filesystem -eq 'NTFS')
  {
    Throw 'Drive already uses NTFS filesystem'
}

"Label is $label"
$label |
convert.exe $letter /FS:NTFS /X
}
```

Make sure the drive you want to convert is not in use or else the conversion may be scheduled to take place at next reboot.

## 40. Reading and Writing Drive Labels

Drive labels are the names attached to logical disks. Using WMI, you can both read and write (change) drive labels.
To read the existing drive label, use this function:

```
Function Get-DriveLabel
{
param
  (
$letter='C:'
  )
```

```
if (!(Test-Path $letter))
{
  Throw "Drive $letter does not exist."
}

  ([wmi]"Win32_LogicalDisk='$letter'").VolumeName
}
```

```
PS> Get-DriveLabel
BOOTCAMP
PS>
```

To actually change the drive label, try this one:

```
Function Set-DriveLabel
{
param
  (
$letter='C:',

$label='New Label'
  )

if (!(Test-Path $letter))
  {
Throw "Drive $letter does not exist."
  }

$instance= ([wmi]"Win32_LogicalDisk='$letter'")
$instance.VolumeName = $label
$instance.Put()
}
```

So to set a new drive label for disk D:, use this:

```
Set-DriveLabel D: 'A new label'
```

Be aware that changing a drive label requires Admin privileges.

## 41. Determining a Drives' File System

If you ever needed a tool to find out the type of file system for any drive, take a look at this simple PowerShell function:

```
Function Get-FileSystem
{
  param
  (
    $letter='C:'
  )

if (!(Test-Path $letter))
  {
Throw "Drive $letter does not exist."
  }

  ([wmi]"Win32_LogicalDisk='$letter'").FileSystem
}
```

Get-FileSystem accepts a drive letter such as "D:", defaults to drive "C:" otherwise, and returns the file system used by that drive.

```
PS> Get-FileSystem
NTFS
PS>
```

## 42. Removing Illegal File Name Characters

If you need to batch-process hundreds of file names, you may want to make sure all file names are legal and automatically remove all illegal characters. Here is how you can check for illegal file names (and optionally remove illegal characters from it):

```
$file = 'this*file\\is_illegal<>.txt'
$file
$pattern = '[{0}]' -f ([Regex]::Escape([String] `
    [System.IO.Path]::GetInvalidFileNameChars()))
$newfile = [Regex]::Replace($file, $pattern,'')
$newfile
```

## 43. Using Simple Path Analysis

PowerShell comes with the Split-Path cmdlet, which helps you disassemble paths and find the interesting information. Take a look:

```
Split-Path c:\test\file.txt
Split-Path c:\test\file.txt -IsAbsolute
Split-Path c:\test\file.txt -Leaf
Split-Path c:\test\file.txt -NoQualifier
Split-Path c:\test\file.txt -Parent
Split-Path c:\test\file.txt -Qualifier
```

## 44. Getting Real Paths

PowerShell uses virtual drives, which sometimes have a close mapping to the "real" drives you see in Windows Explorer. However, sometimes these drives have no apparent real-world relation. Let's say you have created the following drive "test":

```
New-PSDrive test FileSystem $env:windir
dir test:
```

The virtual drive "test" now points to your Windows folder. To "translate" virtual paths to real paths, use Convert-Path:

```
PS> Convert-Path test:\system32
C:\Windows\system32
PS>
```

## 45. Select Folder-Dialog

Want to provide your users with a neat dialog to select folders? Simply use a COM object called Shell.Application, which provides the BrowseForFolder method:

```
Function Select-Folder
{
param
  (
$message='Select a folder',

$path = 0
  )

$object = New-Object -ComObject Shell.Application
$folder = $object.BrowseForFolder(0, $message, 0, $path)
if ($folder -ne $null) {    $folder.self.Path  }
}

Select-Folder 'Select the folder you want!'Select-Folder -message 'Select some folder!' -path
$env:windir
```

## 46. Quick Drive Info

Want to quickly get a number of interesting details for any drive? Use the .NET System.IO.DriveInfo class like this:

```
PS> $drive = New-Object System.io.DriveInfo 'C:'

PS> $drive.DriveFormat
NTFS

PS> $drive.VolumeLabel
BOOTCAMP

PS> $drive


Name                : C:\
DriveType           : Fixed
DriveFormat         : NTFS
IsReady             : True
AvailableFreeSpace  : 6212796416
TotalFreeSpace      : 6212796416
TotalSize           : 125139152896
RootDirectory       : C:\
VolumeLabel         : BOOTCAMP

PS>
```