Find and remove duplicate files with PowerShell

https://4sysops.com/archives/find-and-remove-duplicate-files-with-powershell/

You can use PowerShell to find and remove duplicate files that are only wasting valuable storage. Once you identify duplicate files, you can move them to another location, or you might even want to permanently remove all duplicates.

Contents

Admins who need to manage file servers and storage understand the hassle that duplicate files may cause. When storage disks start running out of space, admins tend to look for various tools to find and delete duplicate files that unnecessarily waste storage. However, PowerShell provides all you need to deal with duplicate files.

Data loss warning

The procedure described in this post involves the removal of files, which could cause accidental data loss. Therefore, it is highly recommended to create a good backup of the source directory before proceeding.

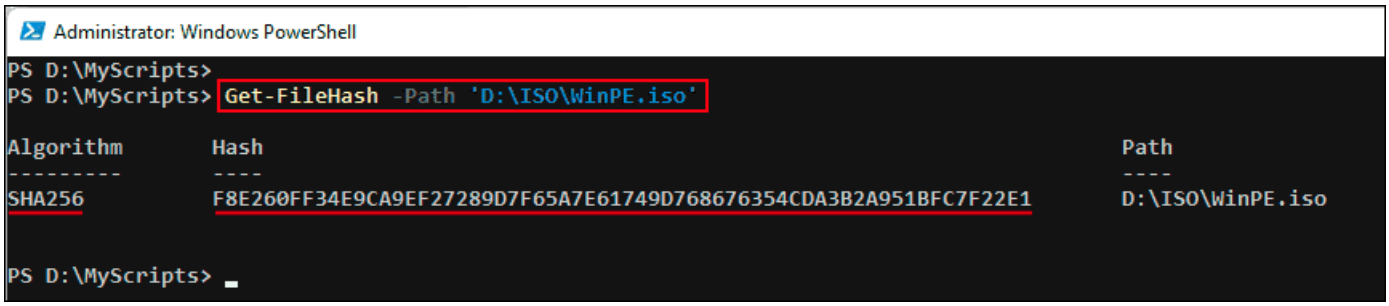
Get-FileHash cmdlet

PowerShell offers the *Get-FileHash* cmdlet to compute the hash (or checksum) of one or more files. This hash can be used to uniquely identify a file. In this post, we will use the hash value to identify duplicate files. The syntax of the command is as follows:

```
Get-FileHash -Path [file_path] -Algorithm [hashing_algorithm]
```

To calculate the hash of a single file, you can run the command shown below:

```
Get-FileHash -Path 'D:\ISO\WinPE.iso' -Algorithm SHA512
```



Calculate the hash or checksum of a file using PowerShell

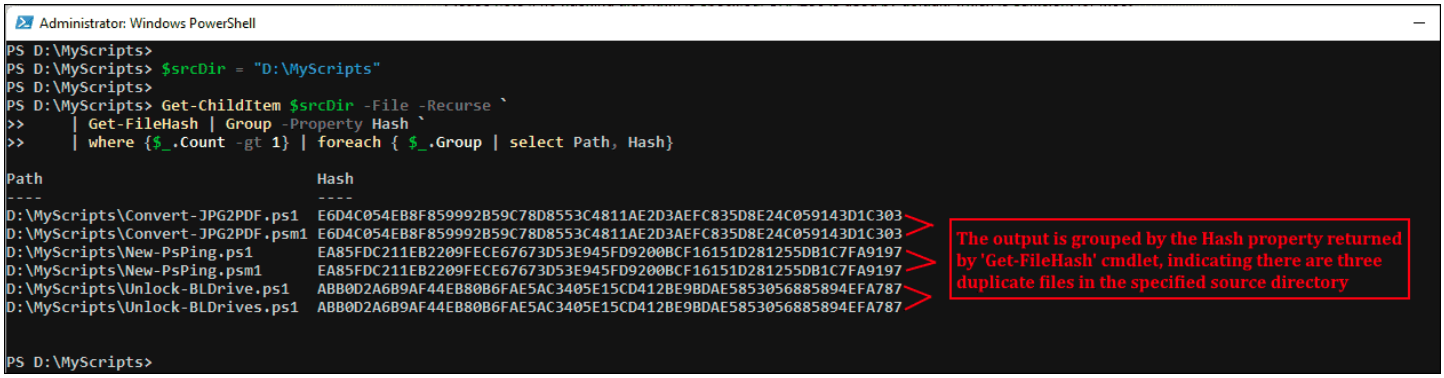
When no hashing algorithm is specified, SHA256 is used by default, which is sufficient in most cases. The hash is used to verify file integrity, particularly for files downloaded from the Internet.

The good thing about *Get-FileHash* is that it accepts the input from the pipeline, which allows us to use it with the *Get-ChildItem* cmdlet to compute the hash of each item returned by *Get-ChildItem*.

Find duplicate files based on the hash

The following command gets all the files in a folder recursively, passes them to *Get-FileHash* to calculate their hash, and finally groups them based on matching hash values.

```
$srcDir = "D:\MyScripts"
Get-ChildItem $srcDir -File -Recurse `
| Get-FileHash | Group -Property Hash `
| where {$_.Count -gt 1} | foreach { $_.Group | select Path, Hash}
```



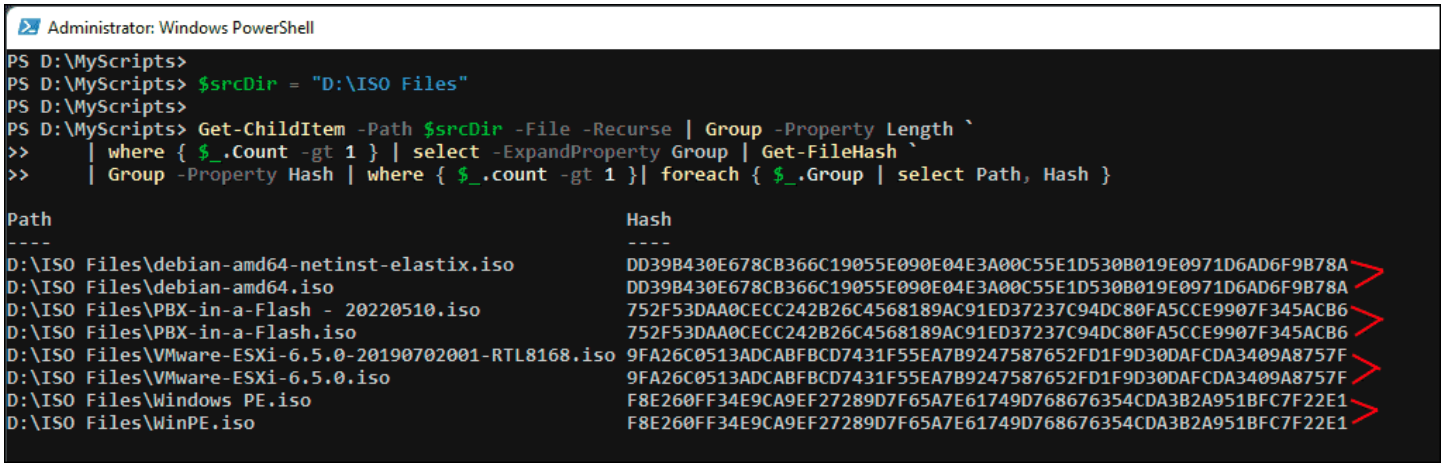
Find duplicate files with PowerShell

The command has one serious flaw, though. It works well for a few small files, but you will run into trouble if the source directory contains many large files. The hash computation is a resource-intensive operation, and the aforementioned command computes it for each file, regardless of its size. The command will therefore take ages to find duplicates when used against a directory with a large number of large files. In the next section, we further optimize this command.

Find duplicate files based on length and hash

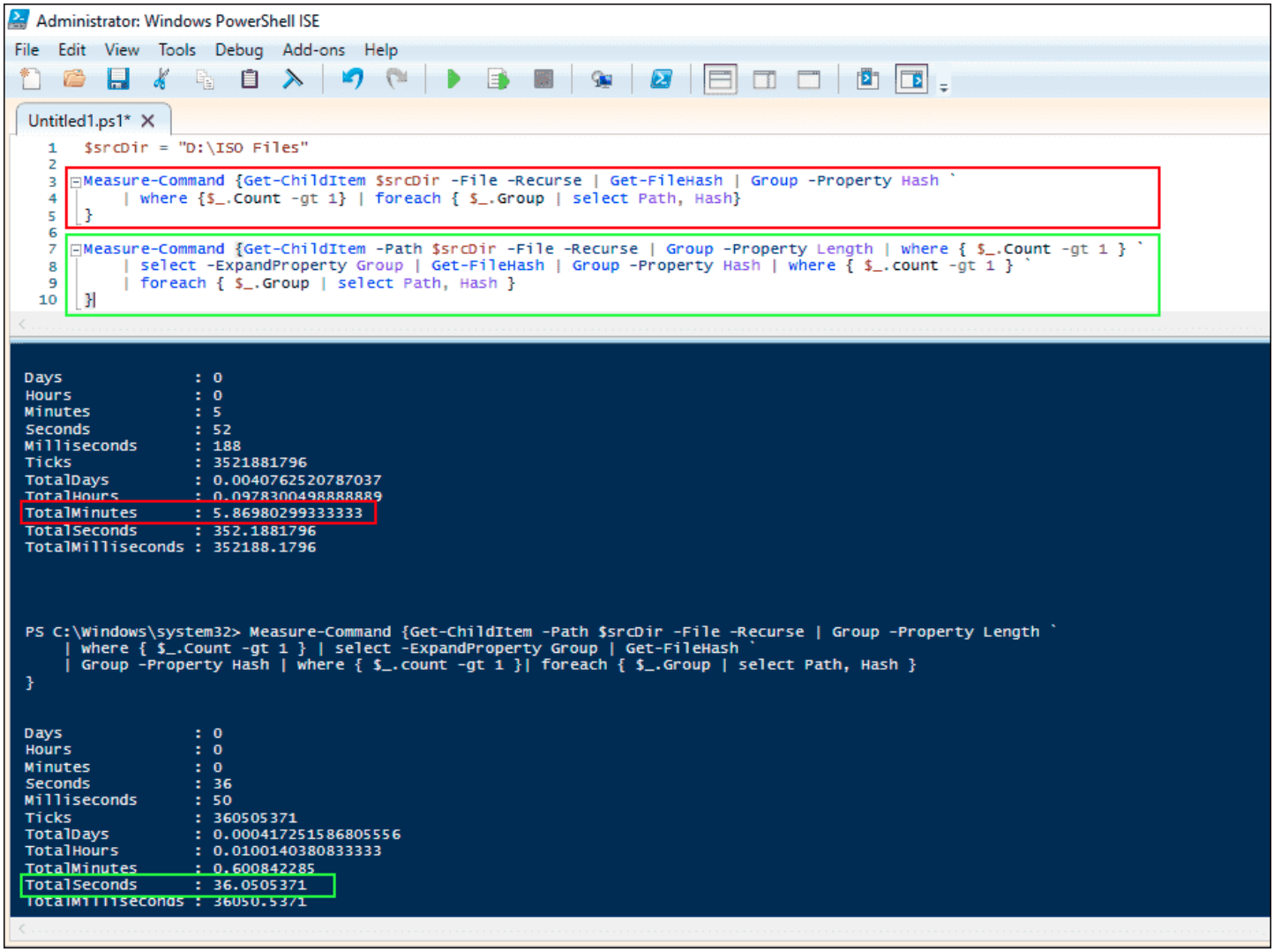
A necessary condition for duplicate files is that their size must match, which means that files where the size does not match cannot be duplicates. You probably know the *Length* property of the *Get-ChildItem* PowerShell cmdlet. Because the *Length* value is retrieved from the directory, no computation is required. The trick is to only compute the hashes of files having the same length because we already know that files with different lengths can't be duplicates. In this way, the overall time of the command is significantly reduced. This is accomplished by the PowerShell commands below:

```
$srcDir = "D:\ISO Files"
Get-ChildItem -Path $srcDir -File -Recurse | Group -Property Length `
| where { $_.Count -gt 1 } | select -ExpandProperty Group | Get-FileHash `
| Group -Property Hash | where { $_.count -gt 1 }| foreach { $_.Group | select Path, Hash }
```



Fast way to find duplicate files with PowerShell

With the help of the *Group* cmdlet, we first group the files that match in size, and then pass those files to *Get-FileHash* to calculate their hash. The output is identical to the previous command, but we needed significantly less execution time to get the result. The screenshot below shows a performance comparison of both solutions:



Using Measure Command to compare the performance of two commands

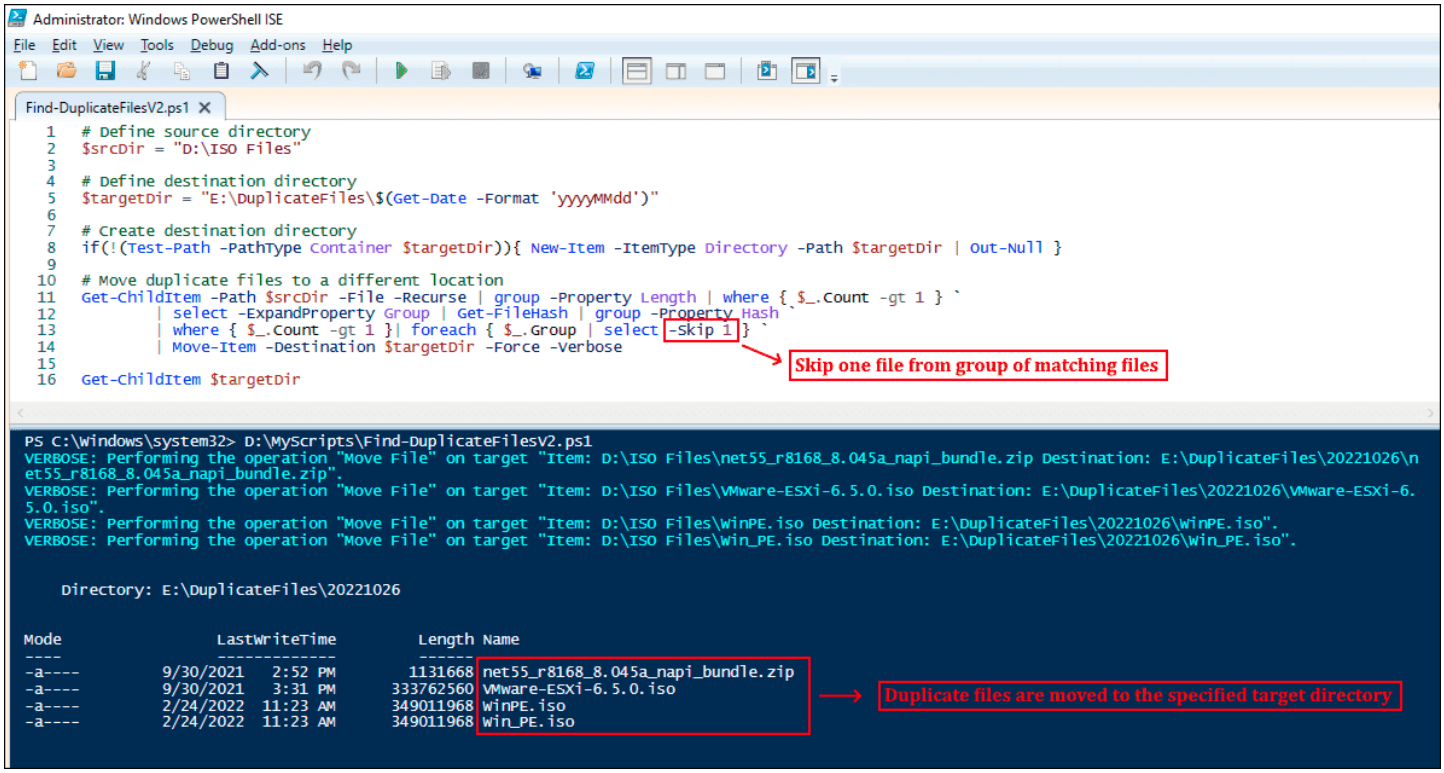
You can see that the second command took just **36** seconds, whereas the first one took about **5** minutes. This difference could increase significantly, depending on the size and number of duplicate files in your source directory.

To speed it up even further, you could use the *MD5* algorithm with the *GetFileHash* cmdlet, which is way faster than the default (SHA256). MD5 isn't good for hashing secret information, but we are only using it here to compare files. The hash-collision risk is relatively higher in MD5, but it doesn't make much difference unless you're comparing billions of files.

Removing duplicate files

Now that we have come up with the right command to identify duplicate files in PowerShell, we need to handle them in a suitable manner. If you are working with important files, it is not recommended to delete duplicates straight away. Instead, you can move them to another directory, probably on a different drive, with enough free space. This is very easy with our new command. We just need to pipe the output of the above command and pass it to the *Move-Item* cmdlet. See the updated code shown below.

```
# Define source directory
$srcDir = "D:\ISO Files"
# Define destination directory
$targetDir = "E:\DuplicateFiles\$(Get-Date -Format 'yyyyMMdd')"
# Create destination directory
if(!(Test-Path -PathType Container $targetDir)){ New-Item -ItemType Directory -Path $targetDir | Out-Null }
# Move duplicate files to a different location
Get-ChildItem -Path $srcDir -File -Recurse | group -Property Length | where { $_.Count -gt 1 } `
| select -ExpandProperty Group | Get-FileHash | group -Property Hash `
| where { $_.Count -gt 1 }| foreach { $_.Group | select -Skip 1 } `
| Move-Item -Destination $targetDir -Force -Verbose
```

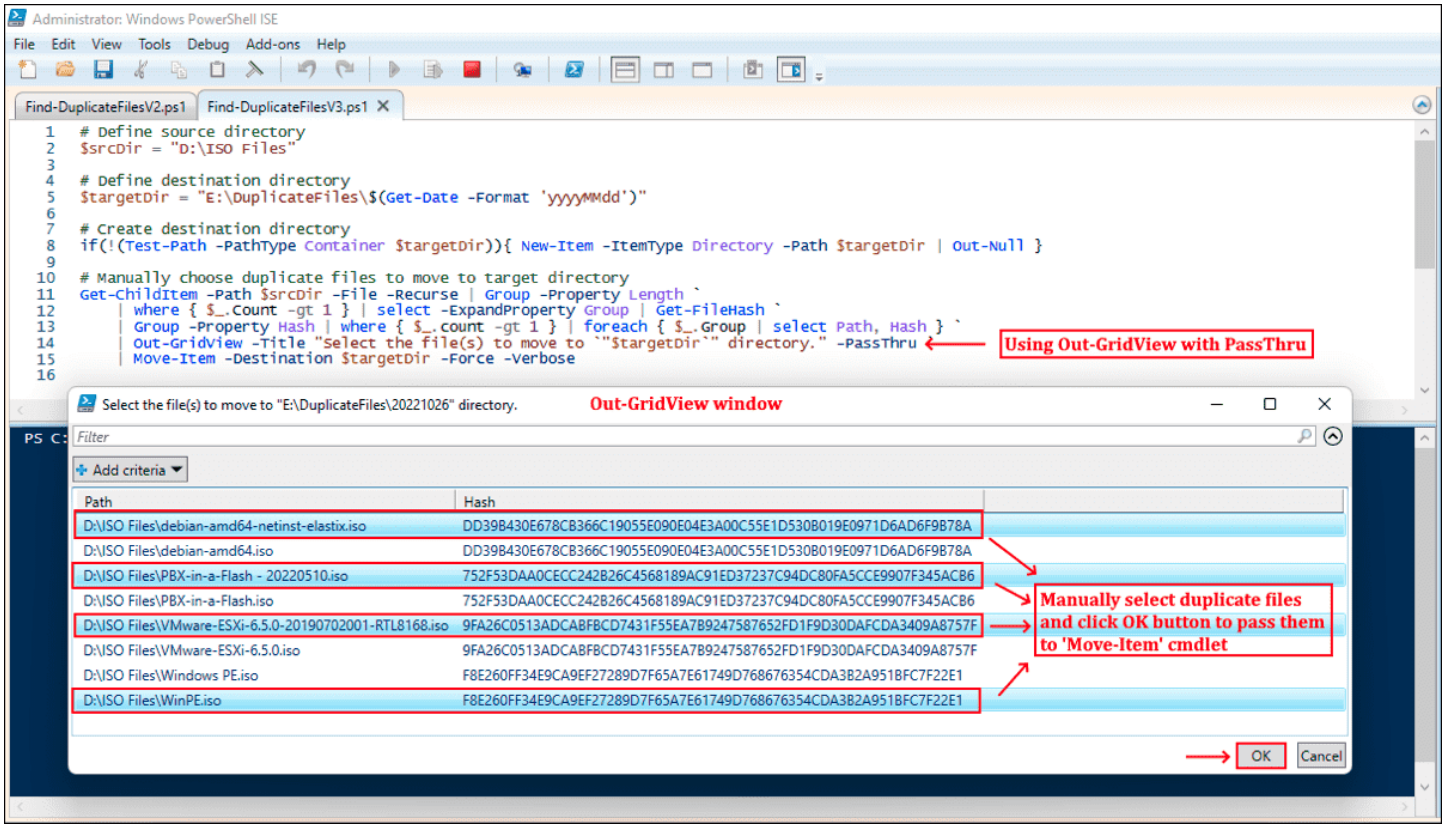


Find and move duplicate files to another drive automatically using PowerShell

The only change here is that I used *-Skip 1* to leave one file in the source directory while moving other duplicates to the specified target directory. Once moved, you can manually review them later on and remove them, if necessary. If you're working on a huge source directory with millions of files, it is a good idea to avoid using the *-verbose* parameter with *Move-Item*.

If your directory contains a handful of files, you may want to manually select which file to move and which to leave in the source directory. You can modify the above code as follows:

```
# Define source directory
$srcDir = "D:\ISO Files"
# Define destination directory
$targetDir = "E:\DuplicateFiles\$(Get-Date -Format 'yyyyMMdd')"
# Create destination directory
if(!(Test-Path -PathType Container $targetDir)){ New-Item -ItemType Directory -Path $targetDir | Out-Null }
# Manually choose duplicate files to move to target directory
Get-ChildItem -Path $srcDir -File -Recurse | Group -Property Length `
| where { $_.Count -gt 1 } | select -ExpandProperty Group | Get-FileHash `
| Group -Property Hash | where { $_.count -gt 1 } | foreach { $_.Group | select Path, Hash } `
| Out-GridView -Title "Select the file(s) to move to `"$targetDir`" directory." -PassThru `
| Move-Item -Destination $targetDir -Force -Verbose
```

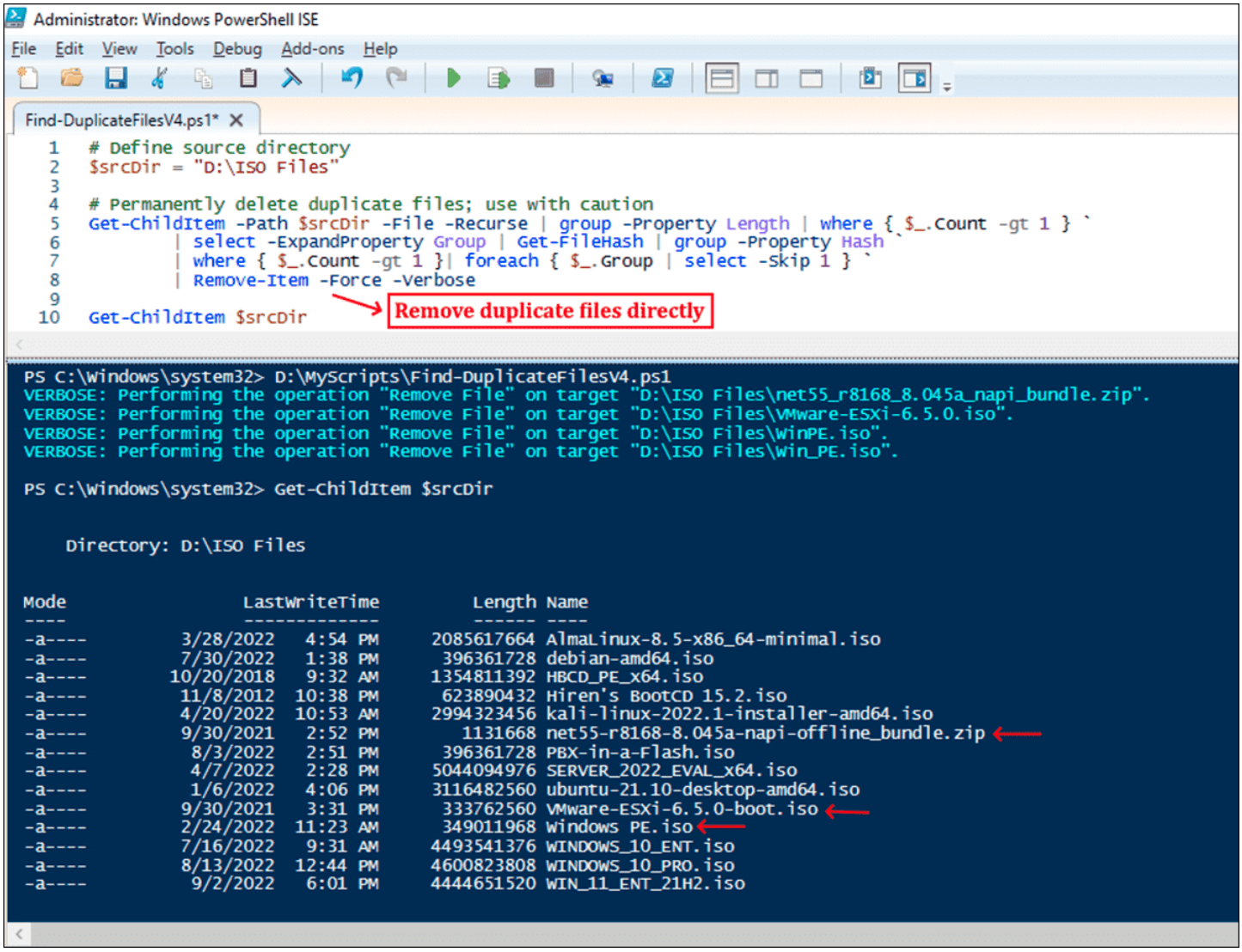


Find and move duplicate files to another drive manually using PowerShell

Here, we are using the *Out-GridView* cmdlet with the *-PassThru* switch to display duplicate files in a new window, so you can manually control what to leave in the source directory and what to move. To select multiple files, you would press and hold the *Ctrl* key while using the mouse to click through the files. When you click OK, the selected files will be moved to the target directory.

Furthermore, if you already have a good backup of your source directory and want to remove the duplicate files permanently, use the following code instead:

```
# Define source directory
$srcDir = "D:\ISO Files"
# Permanently delete duplicate files; use with caution
Get-ChildItem -Path $srcDir -File -Recurse | group -Property Length | where { $_.Count -gt 1 } `
| select -ExpandProperty Group | Get-FileHash | group -Property Hash `
| where { $_.Count -gt 1 }| foreach { $_.Group | select -Skip 1 } `
| Remove-Item -Force -Verbose
```



Find and remove duplicate files with PowerShell

You can see in the screenshot that four duplicate files were deleted successfully, and in the end, our source directory has only unique files.