Scripts

- Firewall Export to CSV
- Bildsortierung
- Downloads-Ordner sortieren

Firewall Export to CSV

```
import csv

txt_file = 'input.txt'
csv_file = 'output.csv'

with open(txt_file, 'r') as file:
    lines = file.readlines()

# Entferne doppelte Leerzeichen zwischen den Spaltenwerten
lines = [line.replace('  ', ' ') for line in lines]

data = [line.strip().split('\t') for line in lines]

with open(csv_file, 'w', newline='') as file:
    writer = csv.writer(file, delimiter=',')
    writer.writerows(data)

print('Die TXT-Datei wurde erfolgreich in eine CSV-Datei umgewandelt.')
```

Vorsicht: Nur ein schneller ChatGPT Vorgang und kurz an Beispieldaten getestet. **Testet selbst, bevor Ihr das auf Produktivdaten loslässt!**

Bildsortierung

> Kurze Vorwarnung dass dies hier überwiegend KI-generiertes Zeug ist. Hat funktioniert, muss aber nicht erneut funktionieren. Es ist wichtig dass ihr in Grundzügen nachvollziehen könnt, was die Scripts tun, auch wenn sie generiert sind.

Das hier wird sicherlich irgendwann mal kombiniert in nem großen Script dass dann gewisse Auswahl zulässt.

Nach Bildauflösung

Skript sammelt Bilder die kleiner als festgelegte Auflösung sind in einem anderen Ordner und belässt die, die größer sind im Ordner.

Das Beispiel hier trennt Bilder >4K von denen die es nicht sind.

```
param (
    [string]$SourceFolder = "C:\Path\to\SourceFolder",
    [string]$DestinationFolder = "C:\Path\to\DestinationFolder",
    [int]$MinWidth = 3840,
    [int]$MinHeight = 2160
)

# Create the destination folder if it doesn't exist
if (-not (Test-Path $DestinationFolder)) {
    New-Item -ItemType Directory -Path $DestinationFolder | Out-Null
}

# Get all image files in the source folder and its subfolders
$images = Get-ChildItem -Path $SourceFolder -Filter "*.jpg" -File -Recurse

$totalImages = $images.Count
$processedImages = 0

# Process each image file
```

```
foreach ($image in $images) {
    Write-Host "Processing $($image.FullName)"

    # Use .NET classes to read image dimensions
    $imageStream = New-Object System.IO.FileStream($image.FullName, [System.IO.FileMode]::Open)
    $imageBitmap = New-Object System.Drawing.Bitmap($imageStream)
    $width = $imageBitmap.Width
    $height = $imageBitmap.Height
    $imageStream.Close()

    # Check if the image is smaller than the specified dimensions
    if ($width -lt $MinWidth -or $height -lt $MinHeight) {
        $destinationPath = Join-Path -Path $DestinationFolder -ChildPath $image.Name
        Write-Host "Moving $($image.FullName) to $destinationPath"
        Move-Item -Path $image.FullName -Destination $destinationPath
    }

    $processedImages++
    $progress = [math]::Round(($processedImages / $totalImages) * 100, 2)
    Write-Progress -Activity "Moving images" -Status "Progress: $progress%" -PercentComplete $progress
}

Write-Progress -Activity "Moving images" -Status "Progress: 100%" -PercentComplete 100
Write-Host "Image move complete!"
```

Dedup

(falls Copy statt Move im Script vorab gemacht wurde)

> Das Ding tut noch nicht das was ich will, womöglich komme ich selbst durcheinander mit Source und Destination. Ich setz also unten noch mal anders an.

```
param (
  # Mag etwas durcheinander und verkehrt sein. KI halt... Jedenfalls ist Destination der Ordner der über bleiben soll. und source der Ordner in dem die Duplikate liegen
    # Ordner in dem gelöscht wird.
    [string]$SourceFolder = "XXXXXXXXXXXXXXXXXXXXXXXX",
```

```
    # Ordner gegen den Verglichen wird
    [string]$DestinationFolder = "XXXXXXXXXXXXXXXXXXXX",

)

# Get all image files in the destination folder
$destinationImages = Get-ChildItem -Path $DestinationFolder -Filter "*.jpg" -File

# Initialize counters for deleted images in each folder
$deletedInSourceFolderCount = 0
$deletedInDestinationFolderCount = 0

# Process each image file in the destination folder
foreach ($destinationImage in $destinationImages) {
    Write-Host "Processing $($destinationImage.FullName)"

    # Construct the source file path based on the destination file name
    $sourceFilePath = Join-Path -Path $SourceFolder -ChildPath $destinationImage.Name

    # Check if the corresponding file exists in the source folder
    if (Test-Path $sourceFilePath) {
        Write-Host "Deleting $($sourceFilePath)"
        # Remove the item (move to Recycle Bin)
        Remove-Item -Path $sourceFilePath -Force -Confirm:$false

        # Increment the counter for deleted images in the source folder
        $deletedInSourceFolderCount++
    }

    # Increment the counter for deleted images in the destination folder
    $deletedInDestinationFolderCount++
}

# Output the deletion statistics
Write-Host "Duplicate removal complete!"
Write-Host "Deleted $deletedInSourceFolderCount images in the source folder: $SourceFolder"
```

Die Abwesenheit von Bildern >4K-Auflösung nachvollziehen

```
$SourceFolder = "C:\Path\to\SourceFolder"
$MinWidth = 3840
$MinHeight = 2160

# Get all image files in the source folder and its subfolders
$images = Get-ChildItem -Path $SourceFolder -Filter "*.jpg" -File -Recurse

$totalImages = $images.Count
$processedImages = 0

# Process each image file
foreach ($image in $images) {
    # Use .NET classes to read image dimensions
    $imageStream = New-Object System.IO.FileStream($image.FullName, [System.IO.FileMode]::Open)
    $imageBitmap = New-Object System.Drawing.Bitmap($imageStream)
    $width = $imageBitmap.Width
    $height = $imageBitmap.Height
    $imageStream.Close()

    # Check if the image is larger than the specified dimensions
    if ($width -ge $MinWidth -and $height -ge $MinHeight) {
        Write-Host "Found large image: $($image.FullName)"
    }

    $processedImages++
    $progress = [math]::Round(($processedImages / $totalImages) * 100, 2)
    Write-Progress -Activity "Checking image sizes" -Status "Progress: $progress%" -PercentComplete $progress
}

Write-Progress -Activity "Checking image sizes" -Status "Progress: 100%" -PercentComplete 100
Write-Host "Image size check complete!"
```

Alle Bilder > 4K löschen

```
$SourceFolder = "C:\Path\to\SourceFolder"
$MinWidth = 3840
$MinHeight = 2160

# Get all image files in the source folder and its subfolders
$images = Get-ChildItem -Path $SourceFolder -Filter "*.jpg" -File -Recurse

$totalImages = $images.Count
$processedImages = 0

# Process each image file
foreach ($image in $images) {
    # Use .NET classes to read image dimensions
    $imageStream = New-Object System.IO.FileStream($image.FullName, [System.IO.FileMode]::Open)
    $imageBitmap = New-Object System.Drawing.Bitmap($imageStream)
    $width = $imageBitmap.Width
    $height = $imageBitmap.Height
    $imageStream.Close()

    # Check if the image is larger than the specified dimensions
    if ($width -ge $MinWidth -and $height -ge $MinHeight) {
        Write-Host "Deleting large image: $($image.FullName)"
        Remove-Item -Path $image.FullName -Force
    }

    $processedImages++
    $progress = [math]::Round(($processedImages / $totalImages) * 100, 2)
    Write-Progress -Activity "Checking and deleting large images" -Status "Progress: $progress%" -
PercentComplete $progress
}

Write-Progress -Activity "Checking and deleting large images" -Status "Progress: 100%" -PercentComplete
100
Write-Host "Image size check and deletion complete!"
```

Downloads-Ordner sortieren

> !!! WORK IN PROGRESS !!!

```
# Pfad zum Downloads-Ordner
$downloadsPath = "$env:userprofile\Downloads"

# Name des Ordners, in den alle nicht sortierten Ordner verschoben werden sollen
$folderName = "Ordner"

# Liste von Zielordnern und den zugehörigen Dateierweiterungen
$folders = @{
    "Dokumente" = @(".pdf", ".odt", ".doc", ".docx", ".rtf", ".txt")
    "Bilder" = @(".jpg", ".jpeg", ".png", ".gif", ".bmp")
    "Musik" = @(".mp3", ".flac", ".wav", ".m4a", ".aac", ".wma")
    "Archive" = @(".zip", ".rar", ".7z", ".tar", ".gz")
    "Setups" = @(".exe", ".msi")
}

# Erstellen des Ordners für nicht sortierte Dateien, falls er nicht vorhanden ist
$otherFolderPath = Join-Path $downloadsPath $folderName
if (-not (Test-Path $otherFolderPath)) {
    New-Item -ItemType Directory -Path $otherFolderPath | Out-Null
}

# Sortieren der Dateien
Get-ChildItem $downloadsPath -Exclude $folderName | Where-Object { $_.PSIsContainer -eq $false } | ForEach-
Object {
    $extension = $_.Extension
    foreach ($folder in $folders.Keys) {
        if ($folders[$folder] -contains $extension) {
            $targetPath = Join-Path $downloadsPath $folder
            Move-Item $_.FullName $targetPath
            return
        }
    }
```

```
    if ($_.Name -ne $folderName -and $_.Name -ne "Sonstige Dateien") {
        $otherFolderPath = Join-Path $downloadsPath $folderName
        Move-Item $_.FullName $otherFolderPath
    }
}

# Verschieben der verbleibenden Ordner in den Ordner für nicht sortierte Dateien
Get-ChildItem $downloadsPath -Directory | Where-Object { $_.Name -notin $folders.Keys -and $_.Name -ne $folderName -and $_.Name -ne "Sonstige Dateien" } | ForEach-Object {
    Move-Item $_.FullName $otherFolderPath
}

# Schließen des PowerShell-Fensters
Exit
```

Das Script erstellt Ordner und verschiebt Dateien darein. Alle sonstigen Ordner die in dem Downloads-Ordner verbleiben, werden in einen Ordner namens "Ordner" verschoben:

Endresultat ist ein Downloads-Ordner mit sieben Ordnern der entsprechenden Dateitypen. Deutlich aufgeräumter als alles chronologisch rumliegen zu lassen.