|  |  |
| --- | --- |
| **TABLE OF CONTENTS** | **PAGE NUMBER** |
| * PowerShell Script to Copy files from SharePoint Application to Local Server | 2 |
| * Console Application to fetch pdf files from Local Server to SharePoint Application | 4 |

**PowerShell Script to Copy files from SharePoint Application to Local Server**

This Script copies the files from SharePoint application to local windows server and logs the details in a log file. We can automate this process by adding it to scheduler task. This reduces the manual work of constant file update on local server.

**Estimated Time Saved: 30 Hrs/month** manual work (This may vary based on the action frequency & files count/size)

**SCRIPT:**

$RootPath = Split-Path -Path $PSScriptRoot -Parent

$Path = $RootPath+" **<Log Folder Name>** "

$PreviousDay = (Get-Date).AddDays(-1)

$FileServerPath = " **<Server Path>** "

$DriveName = " **<Drive Name>** "

$DavWWWRoot = " **<SharePoint Application Path>** "

#Get-Credential | Export-Clixml -Path $PSScriptRoot\${env:USERNAME}\_cred.xml

$cred = Import-Clixml -Path $PSScriptRoot\${env:USERNAME}\_cred.xml

$domain = " **<Domain Name>** "

$UserName = $cred.UserName

$Password = $cred.GetNetworkCredential().Password

$UnsupportedExtensions = @("\*.aspx","\*.asp","\*.db")

$UnsupportedChars = @("\*``[\*","\*``!\*","\*``$\*","\*``&\*","\*``{\*","``\*}\*","\*``~\*","\*``#\*","\*``%\*","\*``]\*")

try

{

if(!(Test-Path $Path))

{

New-Item -ItemType Directory -Force -Path $Path

}

if(!(Test-Path $DriveName))

{

$net = New-Object -ComObject WScript.Network

$net.MapNetworkDrive($DriveName, $DavWWWRoot, $false, $UserName, $Password)

}

############### Delete Previous Day Logs ##########

Get-ChildItem -Path $Path -Recurse -Force | Where-Object { !$\_.PSIsContainer –and $\_.CreationTime -lt $(Get-Date -Format MM-dd-yyyy) } | Remove-Item -Force

############### Copying the Files #####################

$source = "Z:\ **<Source Folder Name>** \"

$destination = "$FileServerPath\ **<Destination Folder Name>** \"

$ExtensionIgnored = Get-ChildItem $destination -Include $UnsupportedExtensions -Recurse

$FilesIgnored = Get-ChildItem $destination -Include $UnsupportedChars -Recurse | ?{!$\_.PSIsContainer }

$FolderIgnored = Get-ChildItem $destination -Include $UnsupportedChars -Recurse | ?{ $\_.PSIsContainer }

$IgnoreFormsFolder = $source+'\Forms'

robocopy $source $destination /MIR /XF $ExtensionIgnored $FilesIgnored /XD $FolderIgnored $IgnoreFormsFolder | Out-File $Path"atlas\_manuals\_$(Get-Date -Format MM-dd-yyyy).log" -Append

Write-Host "Execution Completed" -ForegroundColor Yellow

}

catch

{

$exception = $\_.Exception.Message

Out-File -FilePath $Path"Exception.log" -Append -InputObject $exception

Write-Host $exception

}

**Console Application to fetch pdf files from Local Server to SharePoint Application**

This Console application uploads pdf files from local windows server to SharePoint Application and logs the details in a Task List. It will eliminate other extensions from copying to SharePoint and also logs the error to the error log list with year folder structure for future troubleshoot. We can automate this process by adding it as a task in windows task scheduler. This reduces the manual work of uploading the files in multiple places.

**Estimated Time Saved: 15 Hrs/month** manual work (This may vary based on the action frequency & files count/size)

**Code:**

using System;

using System.Data;

using System.Linq;

using System.IO;

using System.Web.Configuration;

using System.Collections.Generic;

using System.Text.RegularExpressions;

using System.Diagnostics;

using Microsoft.SharePoint;

using Microsoft.CSharp;

namespace **<Namespace Name>**

{

class **<Class Name>**

{

**#region Constant Declaration**

static string destSite = string.Empty;

static string sourceSite = string.Empty;

static string userName = string.Empty;

static string domainName = string.Empty;

static string passWord = string.Empty;

static string title = string.Empty;

static string logListName = string.Empty;

static string configList = string.Empty;

static string Source = string.Empty;

static string Destination = string.Empty;

static int filesAddedCount = 0;

static string filesAdded = string.Empty;

static int foldersAddedCount = 0;

static string foldersAdded = string.Empty;

static int filesDeletedCount = 0;

static string filesDeleted = string.Empty;

static int foldersDeletedCount = 0;

static string foldersDeleted = string.Empty;

static int filesNotAddedCount = 0;

static string filesNotAdded = string.Empty;

static string lblErrorMsg = string.Empty;

static string[] ext = new string[] { };

static string[] extension = new string[] { };

static string webUrl = string.Empty;

static string subSite = string.Empty;

**#endregion**

**#region Deleting the files**

static void DeleteFiles(SPList targetLDDocLib, List<Int32> ids)

{

try

{

foreach (Int32 id in ids)

{

SPListItem item = targetLDDocLib.GetItemById(id);

filesDeletedCount = ids.Count;

filesDeleted += item.Name.ToString() + "<br>";

lblErrorMsg = "Files are updated";

item.Delete();

}

targetLDDocLib.Update();

}

catch (Exception ex)

{

Console.WriteLine(ex.Message);

}

}

**#endregion**

**#region Deleting the Folders**

static void DeleteFolders(SPList targetLDDocLib, List<Int32> ids)

{

try

{

foreach (Int32 id in ids)

{

SPListItem item = targetLDDocLib.GetItemById(id);

foldersDeletedCount += 1;

foldersDeleted += item.Name.ToString() + "<br>";

lblErrorMsg = "Files are updated";

item.Delete();

}

targetLDDocLib.Update();

}

catch (Exception ex)

{

if (Convert.ToString(ex.Message).ToLower() == "item does not exist. it may have been deleted by another user.")

{

Console.WriteLine("item does not exist");

}

else

{

Console.WriteLine(ex.Message);

}

}

}

**#endregion**

**#region Uploading the Files and Folders**

static void UploadFolders(string path, SPFolderCollection oFolderCollection, string[] restrictedExt)

{

try

{

**#region Upload Multiple Files from the server file path**

foreach (FileInfo oFI in new DirectoryInfo(path).GetFiles())

{

if (oFI.Extension == ".pdf")

{

string fileName = string.Empty;

fileName = GetSharePointFriendlyName(oFI.Name, oFI.FullName.Length + 1, MAXFILELENGTH);

Boolean replaceExistingFiles = true;

FileStream fileStream = File.OpenRead(oFI.FullName);

SPFile spfile = oFolderCollection.Folder.Files.Add(fileName, fileStream, replaceExistingFiles);

spfile.Item["Modified On"] = oFI.LastWriteTime.ToString();

spfile.Item.Update();

filesAddedCount += 1;

filesAdded += "<br/>" + oFI.Name;

}

else

{

filesNotAddedCount += 1;

filesNotAdded += "<br/>" + oFI.Name;

}

}

lblErrorMsg = "Files are updated";

**#endregion**

**#region Upload Multiple Folders from the server file path**

foreach (DirectoryInfo oDI in new DirectoryInfo(path).GetDirectories())

{

string sFolderName = oDI.FullName.Split('\\')

[oDI.FullName.Split('\\').Length - 1];

SPFolder spNewFolder = oFolderCollection.Add(sFolderName);

spNewFolder.Item["Modified On"] = oDI.LastWriteTime.ToString();

spNewFolder.Item.Update();

foldersAddedCount += 1;

foldersAdded += "<br/>" + oDI.Name;

**//Recursive call to create child folder**

UploadFolders(oDI.FullName, spNewFolder.SubFolders, restrictedExt);

}

#endregion

}

catch (Exception)

{

throw;

}

}

**#endregion**

**#region Updating the Sharepoint Destination**

static void UpdateFiles()

{

try

{

List<KeyValuePair<string, string>> mappingItems = new List<KeyValuePair<string, string>>();

using (SPSite objSite = new SPSite(webUrl))

{

using (SPWeb targetWeb = objSite.OpenWeb(subSite))

{

**#region Read filer server credentials from config list**

**//reading config list to get filer credentials, local path and sharepoint library by title**

configList = WebConfigurationManager.AppSettings["configList"];

title = WebConfigurationManager.AppSettings["title"];

SPList oSpList = targetWeb.Lists[configList];

SPListItemCollection oSpListCln = oSpList.Items;

foreach (SPListItem item in oSpListCln)

{

if (item.Title == "FileServerUserName")

{

userName = item["ConfigValue"].ToString();

}

if (item.Title == "FileServerDomainName")

{

domainName = item["ConfigValue"].ToString();

}

if (item.Title == "FileServerPassword")

{

passWord = item["ConfigValue"].ToString();

}

if (item.Title == title)

{

Destination = item["ConfigKey"].ToString();

Source = item["ConfigValue"].ToString();

mappingItems.Add(new KeyValuePair<string, string>(Destination, Source));

}

}

**#endregion**

foreach (KeyValuePair<string, string> kvp in mappingItems)

{

string libName = kvp.Key;

string localPath = kvp.Value;

**//checking whether local path exist or not**

if (Directory.Exists(localPath))

{

**//checking whether the source folder is empty or not**

if (Directory.GetDirectories(localPath).Length == 0 && Directory.GetFiles(localPath, "\*", SearchOption.AllDirectories).Length == 0)

{

throw new Exception("Error in uploading files from, " + localPath + "<br>" + "Source folder is empty or it does not contain any files/folders");

}

else

{

**#region Delete all the files from SharePoint Library**

**// listing all the files from the SharePoint library to delete**

List<Int32> listFiles = new List<Int32>();

SPList targetLDDocLib = targetWeb.Lists[libName];

SPQuery query = new SPQuery();

query.ViewAttributes = "Scope=\"Recursive\"";

SPListItemCollection myItems = targetLDDocLib.GetItems(query);

foreach (SPListItem oLibraryItem in myItems)

{

SPFile file = targetWeb.GetFile(oLibraryItem.Url);

listFiles.Add(file.Item.ID);

}

**//calling the DeleteFiles method by passing target SharePoint library and files list to be deleted**

DeleteFiles(targetLDDocLib, listFiles);

**#endregion**

**#region Delete all folders from SharePoint Library**

**//listing all the folders from the SharePoint library to delet**e

List<Int32> listFolder = new List<Int32>();

SPQuery folderquery = new SPQuery();

folderquery.Query = @"<Where><BeginsWith><FieldRef Name='ContentTypeId' /><Value Type='ContentTypeId'>0x0120</Value></BeginsWith></Where>";

folderquery.ViewAttributes = "Scope=\"RecursiveAll\"";

SPListItemCollection items = targetLDDocLib.GetItems(folderquery);

foreach (SPListItem i in items)

{

listFolder.Add(i.Folder.Item.ID);

}

**//calling the DeleteFolders method by passing target SharePoint library and folders list to be deleted**

DeleteFolders(targetLDDocLib, listFolder);

**#endregion**

**#region Add New file from the file server**

**//uploading the files and folders from local server to SharePoint**

using (new Impersonator(userName, domainName, passWord))

{

SPFolder folderRoot = targetLDDocLib.RootFolder;

SPFolderCollection subFolder = folderRoot.SubFolders;

UploadFolders(localPath, subFolder, extension);

}

**#endregion**

}

}

else

{

throw new Exception(" Error while connecting to the source path" + "<br>" + "Unable to connect to " + localPath + "<br>" + "The source path is incorrect or the source folder does not exists");

}

}

}

}

}

catch (Exception)

{

throw;

}

}

**#endregion**

static void Main(string[] args)

{

try

{

webUrl = WebConfigurationManager.AppSettings["webUrl"];

subSite = WebConfigurationManager.AppSettings["subSite"];

logListName = WebConfigurationManager.AppSettings["logListName"];

SPSite objSite = new SPSite(webUrl);

SPWeb targetWeb = objSite.OpenWeb(subSite);

SPListItem newTask = targetWeb.Lists.TryGetList(logListName).Items.Add();

newTask["Title"] = " **<Your Job Title>** ";

newTask["Priority"] = "(2) Normal";

newTask["Status"] = "Completed";

newTask["StartDate"] = DateTime.Now.ToString();

UpdateFiles();

newTask["DueDate"] = DateTime.Now.ToString();

newTask["Body"] = "Job Completed Successfully" + "<br>" + "<br>" + "<b>Files Added Count: " + filesAddedCount + "</b>" + filesAdded + "<br>" + "<br>"+" Files Updated";

newTask.Update();

}

catch (Exception ex)

{

SPSite objSite = new SPSite(webUrl);

SPWeb targetWeb = objSite.OpenWeb("/flightops/");

SPListItem newTask = targetWeb.Lists.TryGetList(logListName).Items.Add();

newTask["Title"] = " **<Your Job Title>** ";

newTask["Priority"] = "(1) High";

newTask["Status"] = "Not Started";

newTask["StartDate"] = DateTime.Now.ToString();

newTask["Body"] = "Error while updating..." + "<br>" + "<b>Error Message: </b>" + ex.Message;

newTask.Update();

}

}

**#region Regex Declaration**

public const int MAXFOLDERLENGTH = 128, MAXFILELENGTH = 123;

public static int MAXURLLENGTH = 259;

public static Regex invalidCharsRegex = new Regex(@"[\\*\?\|\\\t/:""'<>#{}%~&]", RegexOptions.Compiled);

public static Regex invalidRulesRegex = new Regex(@"\.{2,}", RegexOptions.Compiled);

public static Regex startEndRegex = new Regex(@"^[\. ]|[\. ]$", RegexOptions.Compiled);

public static Regex extraSpacesRegex = new Regex(" {2,}", RegexOptions.Compiled);

**#endregion**

**#region Creating Sharepoint friendly name**

public static string GetSharePointFriendlyName(string original, int currentPathLength, int maxItemLength)

{

**// remove invalid characters and some initial replacements**

string friendlyName = extraSpacesRegex.Replace(invalidRulesRegex.Replace(invalidCharsRegex.Replace(original, String.Empty).Trim(), "."), " ");

**// assign maximum item length**

int maxLength = (currentPathLength + maxItemLength > MAXURLLENGTH) ? MAXURLLENGTH - currentPathLength : maxItemLength;

if (maxLength <= 0)

throw new ApplicationException("Current path is too long for importing into SharePoint");

**// return truncated name if length exceeds maximum**

if (friendlyName.Length > maxLength) friendlyName = friendlyName.Substring(0, maxLength - 1).Trim();

**// finally, check beginning and end for periods and spaces**

while (startEndRegex.IsMatch(friendlyName)) friendlyName = startEndRegex.Replace(friendlyName, String.Empty);

return friendlyName;

}

**#endregion**

}

}