**Script Documentation: Browser Settings Backup Script**

**Overview**

This script is designed to back up browser settings for Google Chrome and Microsoft Edge from multiple user profiles on a back office workstation. The script performs the following tasks:

1. Validates the hostname to ensure it is correctly formatted.
2. Logs operations and errors.
3. Handles network interruptions during file operations.
4. Checks directories for corruption.
5. Creates necessary backup directories.
6. Backs up specified files and folders for each user.

**Prerequisites**

* PowerShell 5.0 or higher.
* Sufficient permissions to access user directories and network paths.
* The script should be executed on a back office workstation with a correctly formatted hostname.

**Execution Instructions**

1. Ensure the script is saved with a .ps1 extension.
2. Open PowerShell with administrative privileges.
3. Navigate to the directory where the script is saved.
4. Execute the script by typing .\scriptname.ps1 (replace scriptname.ps1 with the actual name of the script).

**Notes**

* Ensure network paths and permissions are correctly configured.
* Monitor the log file for detailed information and troubleshooting.
* The script will exit if it encounters significant errors or if the hostname is not correctly formatted.

**Detailed Code Breakdown:**

**1. Set Global Variables**

The script initializes global variables including the hostname and site number, and sets the name of the log file.

$hostname = hostname

if ($hostname.Substring(0, 4) -match '^\d{4}') {

$siteNumber = $hostname.Substring(0, 4)

} else {

Write-Error "This computer, '${hostname}', is either named incorrectly or is not a back office workstation. Exiting script..."

Start-Sleep -Seconds 10

Exit

}

$logFileName = "${hostname}\_Backup.log"

$scriptName = [System.IO.Path]::GetFileName($MyInvocation.ScriptName)

**2. Log Messages**

A function to log messages with a timestamp.

function LogMessage {

param (

[string]$Message

)

$TimeStamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"

try {

Add-Content -Path $logFileName -Value "$TimeStamp - $Message"

} catch {

Write-Error "$scriptName failed to write $Message to $logFileName on $hostname. Error: $\_"

}

}

**3. Handle Network Interruptions**

A function to retry operations in case of network interruptions.

function HandleNetworkInterruptions {

param (

[ScriptBlock]$Operation

)

$retryCount = 3

for ($i = 1; $i -le $retryCount; $i++) {

try {

& $Operation

LogMessage "Operation succeeded on attempt $i."

return $true

} catch [System.IO.IOException] {

LogMessage "Network interruption occurred. Retrying operation ($i/$retryCount)..."

Start-Sleep -Seconds 5

} catch {

LogMessage "Unexpected error occurred: $\_. Exiting operation." "ERROR"

return $false

}

}

LogMessage "Operation failed after $retryCount retries due to network interruptions."

return $false

}

**4. Check Directory Corruption**

A function to check if a directory is corrupted or inaccessible.

function CheckDirCorruption {

param (

[string]$Dir

)

if (!(Test-Path -Path $Dir -ErrorAction SilentlyContinue)) {

LogMessage "$'{Dir}' doesn't exist, assumed corrupted" "ERROR"

return $true

}

try {

Get-ChildItem $Dir -ErrorAction Stop | Out-Null

LogMessage "'${Dir}' is intact and accessible"

return $false

} catch {

LogMessage "'${Dir}' is corrupted or inaccessible" "ERROR"

return $true

}

}

**5. Create Backup Directory**

A function to create backup directories if they do not already exist.

function CreateBackupDirectory {

param (

[string]$Path

)

try {

if (!(Test-Path -Path $Path)) {

New-Item -ItemType Directory -Path $Path | Out-Null

if (Test-Path -Path $Path) {

LogMessage "Backup directory created: $Path"

} else {

LogMessage "Failed to create backup directory: $Path" "ERROR"

}

} else {

LogMessage "Backup directory already exists: $Path"

}

} catch {

LogMessage "Failed to create or verify backup directory: $Path. Error: $\_" "ERROR"

}

}

**6. Initialize Lists**

Initialize lists to track successful and failed backups.

$successfulChromeBackup = @()

$failedChromeBackup = @()

$successfulEdgeBackup = @()

$failedEdgeBackup = @()

**7. Validate Pre-Existing Document Repository Path**

Check if the pre-existing document repository network path is accessible.

$ExistingDocumentRepository = "\\${siteNumber}-pcName\Redacted\"

if (!(Test-Path -Path $ExistingDocumentRepository)) {

LogMessage "$ExistingDocumentRepository is inaccessible, exiting script..."

Exit

}

**8. Create Backup Directory**

Create the main backup directory in "Theatre Documents".

$backupDir = "\\${siteNumber}-pcName\Redacted\Browser Settings Backups\${hostname}"

if (-not (HandleNetworkInterruptions { CreateBackupDirectory -Path $backupDir })) {

LogMessage "Failed to create backup directory $backupDir after multiple attempts. Exiting script." "ERROR"

Exit

}

**9. Get User Directories**

Retrieve the list of user directories to back up.

$usersDir = Get-ChildItem -Path "C:\Users" -Directory

$userList = $usersDir.Name | Where-Object { $\_ -notin @("Administrator", "Public") }

**10. Back Up Browser Settings**

Define and execute the function to back up browser settings for each user.

function BackupBrowserSettings {

param (

[string]$ProfileDir,

[string]$BackupDir,

[string[]]$FilesToBackup,

[string[]]$FoldersToBackup

)

$backupSuccess = $true

if (Test-Path -Path $ProfileDir) {

if (-not (HandleNetworkInterruptions { CreateBackupDirectory -Path $BackupDir })) {

LogMessage "Failed to create backup directory $BackupDir after multiple attempts. Skipping backup for $ProfileDir." "ERROR"

return

}

foreach ($file in $FilesToBackup) {

$sourcePath = Join-Path -Path $ProfileDir -ChildPath $file

if (Test-Path -Path $sourcePath) {

$copyOperation = {

Copy-Item -Path $sourcePath -Destination $BackupDir -Recurse -Force

}

if (-not (HandleNetworkInterruptions $copyOperation)) {

LogMessage "Failed to copy file $sourcePath to $BackupDir after multiple attempts." "ERROR"

$backupSuccess = $false

} else {

LogMessage "Successfully backed up file $sourcePath to $BackupDir."

}

}

}

foreach ($folder in $FoldersToBackup) {

$sourcePath = Join-Path -Path $ProfileDir -ChildPath $folder

if (Test-Path -Path $sourcePath) {

$destinationPath = Join-Path -Path $BackupDir -ChildPath $folder

$copyOperation = {

Copy-Item -Path $sourcePath -Destination $destinationPath -Recurse -Force

}

if (-not (HandleNetworkInterruptions $copyOperation)) {

LogMessage "Failed to copy folder $sourcePath to $destinationPath after multiple attempts." "ERROR"

$backupSuccess = $false

} else {

LogMessage "Successfully backed up folder $sourcePath to $destinationPath."

}

}

}

} else {

LogMessage "Profile directory $ProfileDir does not exist. Skipping backup." "ERROR"

$backupSuccess = $false

}

return $backupSuccess

}

**11. Iterate Through Users**

Perform backup operations for each user.

foreach ($user in $userList) {

$userBackupDir = Join-Path -Path $backupDir -ChildPath $user

if (-not (HandleNetworkInterruptions { CreateBackupDirectory -Path $userBackupDir })) {

LogMessage "Failed to create user backup directory $userBackupDir after multiple attempts. Skipping user $user." "ERROR"

continue

}

$chromeBackupDir = Join-Path -Path $userBackupDir -ChildPath "Chrome"

$edgeBackupDir = Join-Path -Path $userBackupDir -ChildPath "Edge"

$chromeProfileDir = "C:\Users\${user}\AppData\Local\Google\Chrome\User Data\Default"

$edgeProfileDir = "C:\Users\${user}\AppData\Local\Microsoft\Edge\User Data\Default"

if (-not (CheckDirCorruption -Dir $chromeProfileDir)) {

if (BackupBrowserSettings -User $user -ProfileDir $chromeProfileDir -BackupDir $chromeBackupDir -FilesToBackup @("Bookmarks", "Preferences", "Login Data", "History", "Cookies", "Web Data") -FoldersToBackup @("Extensions", "Local Storage", "Session Storage", "Sync Data")) {

$successfulChromeBackup += $user

} else {

$failedChromeBackup += $user

}

} else {

LogMessage "Chrome profile directory $chromeProfileDir is corrupted or inaccessible. Skipping backup." "ERROR"

$failedChromeBackup += $user

}

if (-not (CheckDirCorruption -Dir $edgeProfileDir)) {

if (BackupBrowserSettings -User $user -ProfileDir $edgeProfileDir -BackupDir $edgeBackupDir -FilesToBackup @("Bookmarks", "Preferences", "Login Data", "History", "Cookies", "Web Data") -FoldersToBackup @("Extensions", "Local Storage", "Session Storage", "Sync Data")) {

$successfulEdgeBackup += $user

} else {

$failedEdgeBackup += $user

}

} else {

LogMessage "Edge profile directory $edgeProfileDir is corrupted or inaccessible. Skipping backup." "ERROR"

$failedEdgeBackup += $user

}

}

**12. Log and Display Results**

Combine results and log the final outcomes.

$allSuccessful = $successfulChromeBackup + $successfulEdgeBackup

$allFailed = $failedChromeBackup + $failedEdgeBackup

$successfulTotalBackup = $allSuccessful | Where-Object {

$user = $\_

$allFailed -notcontains $user

}

$successfulTotalBackup = $successfulTotalBackup | Select-Object -Unique

if ($successfulTotalBackup.Count -gt 0) {

$successMessage = "Successfully backed up Chrome and Edge settings for: $($successfulTotalBackup -join ', ')."

LogMessage $successMessage

Write-Host $successMessage

}

if ($failedChromeBackup.Count -gt 0) {

$failedMessage = "Failed to back up Chrome settings for: $($failedChromeBackup -join ', ')."

LogMessage $failedMessage

Write-Host $failedMessage

}

if ($failedEdgeBackup.Count -gt 0) {

$failedMessage = "Failed to back up Edge settings for: $($failedEdgeBackup -join ', ')."

LogMessage $failedMessage

Write-Host $failedMessage

}

LogMessage "Browser settings have been backed up!"

Start-Sleep -Seconds 10