

File System - Windows

- [List all Permissions in Share Recursively](#)
- [SMB ACL Group Creation Script](#)
- [Take Ownership and Change Admins to Full Rights](#)
- [Compress WinSxS Folder](#)
- [Create Directories From List](#)
- [Create-WorkloadFolder](#)

List all Permissions in Share Recursively

```
Get-childitem \\network\share\ -recurse | Where-Object{$_ .psiscontainer} |  
Get-Acl | ForEach-Object {  
    $path = $_.Path  
    $_.Access | ForEach-Object {  
        New-Object PSObject -Property @{  
            Folder = $path.Replace("Microsoft.PowerShell.Core\FileSystem:", "")  
            Access = $_.FileSystemRights  
            Control = $_.AccessControlType  
            User = $_.IdentityReference  
            Inheritance = $_.IsInherited  
        }  
    }  
}  
} | select-object -Property User, Access, Folder | export-csv output.csv -force
```

SMB ACL Group Creation Script

Asks you for a root folder, then allows you to enter subfolders until you leave an entry empty. Next it asks for the server name. Will create all four permission types and also apply default server, domain, and enterprise admins to the full access group.

```
param(
    [Parameter(Mandatory=$True)][String[]]$FolderName,
    [Parameter(Mandatory=$True)][String[]]$ServerName,
    [Parameter(Mandatory=$False)][String[]]$SubFolderOf,
    [Parameter(Mandatory=$False)][String]$OrganizationUnit="OU=SMB,OU=Security
    Groups,DC=corp,DC=mutschlerhome,DC=com"
)

$ServerNameUpper = $ServerName.ToUpper()
$ServerName = $ServerName.ToLower()

$Name = "SMB_{$ServerNameUpper}_{$FolderName}"
$Description = "\\{$ServerName}\$FolderName"
if($SubFolderOf) {

    if( ($SubFolderOf.EndsWith("\")) ) {
        $SubFolderOf = $SubFolderOf.substring(0,($SubFolderOf.Length-1))
    }
    $Description = "\\{$ServerName}\{$SubFolderOf}\{$FolderName}"
}

New-ADGroup -Name "{$Name}_Read" -Description "{$Description} [Read Only]" -groupscope
DomainLocal -path $OrganizationUnit
New-ADGroup -Name "{$Name}_Modify" -Description "{$Description} [Modify]" -groupscope
DomainLocal -path $OrganizationUnit
New-ADGroup -Name "{$Name}_Full" -Description "{$Description} [FullControl]" -groupscope
DomainLocal -path $OrganizationUnit
New-ADGroup -Name "{$Name}_List" -Description "{$Description} [List Only]" -groupscope
DomainLocal -path $OrganizationUnit

Add-ADGroupMember -Identity "{$Name}_Full" -Members "Domain Admins"
```

```
Add-ADGroupMember -Identity "$($Name)_Full" -Members "Enterprise Admins"
```

```
Add-ADGroupMember -Identity "$($Name)_Full" -Members "Server Admins"
```

Take Ownership and Change Admins to Full Rights

```
$folderMask = "C:\Program Files\WindowsApps\Microsoft.DesktopAppInstaller_*"
$folders = Get-ChildItem -Path $folderMask -Directory | Where-Object { $_.Name -like "*_x64_*"
}
foreach ($folder in $folders) {
    $folderPath = $folder.FullName
    TAKEOWN /F $folderPath /R /A /D Y
    ICACLS $folderPath /grant Administrators:F /T
}
```

Compress WinSxS Folder

1. Query service status. Track its state.

```
sc query msiserver
sc query TrustedInstaller
```

2. Stop/disable Windows Installer and Windows Module Installer services

```
sc stop msiserver
sc config msiserver start= disabled
sc stop TrustedInstaller
sc config TrustedInstaller start= disabled
```

3. Backup ACLs for WinSxS folder.

```
icacls "%WINDIR%\WinSxS" /save "%WINDIR%\WinSxS.acl" /t
```

4. Take ownership of WinSxS folder

```
takeown /f "%WINDIR%\WinSxS" /r
```

5. Grant full rights on WinSxS to user

```
icacls "%WINDIR%\WinSxS" /grant "USERNAME7":(F) /t
```

6. Compress Folders

```
compact /s:"%WINDIR%\WinSxS" /c /a /i *
```

7. Restore ownership

```
icacls "%WINDIR%\WinSxS" /setowner "NT SERVICE\TrustedInstaller" /t
```

8. Restore ACLs

```
icacls "%WINDIR%" /restore "%WINDIR%\WinSxS.acl"
del "%WINDIR%\WinSxS.acl"
```

9. Restore services, replace "demand" and "start" with the right state

```
sc config msiserver start= auto
sc start msiserver
sc config TrustedInstaller start= auto
sc start TrustedInstaller
```

From <http://blog.aslanbrooke.com/2015/07/25/compressing-winsxs-and-installer-on-windows.html>

Create Directories From List

Script

```
set list=\\192.168.1.41\Emulation\OrganizedRoms\directories.txt
set location=\\192.168.1.41\Emulation\OrganizedRoms

for /f "delims== tokens=1,2" %%G in (%list%) do (
    md "%location%\%%G"
    if errorlevel 1 (echo Error creating folder : %location%\%%G >> Error.txt)
)
```

List Example

```
1292-advanced-programmable-video-system
3do
abc-80
apf
ay-3-8500
ay-3-8603
ay-3-8605
ay-3-8606
ay-3-8607
ay-3-8610
ay-3-8710
ay-3-8760
acorn-archimedes
acorn-electron
adventure-vision
airconsole
alice-3290
altair-680
altair-8800
amazon-alexa
```

amazon-fire-tv
amiga
amiga-cd32
acpc
amstrad-pcw
android
antstream
apple-i
appleii
apple2gs
apple-iigs
apple-pippin
arcade
arcadia-2001
arduboy
astral-2000
atari2600
atari5200
atari7800
atari8bit
jaguar
atari-jaguar-cd
lynx
atari-st
atari-vcs
atom
bbcmicro
bbcmicro
brew
astrocade
beos
blackberry
blacknut
blu-ray-player
bubble
philips-cd-i
cdccyber70
commodore-cdtv
fred-cosmac
cpm

call-a-computer
computers-lynx
casio-loopy
casio-pv-1000
casio-programmable-calculator
champion-2711
fairchild-channel-f
clickstart
colecoadam
colecovision
colour-genie
c128
c16
c64
commodore-cdtv
cpet
cpet
c-plus-4
vic-20
compal-80
compucolor-i
compucolor-ii
compucorp-programmable-calculator
creativision
cybervision
dos
dvd-player
danger-os
dedicated-console
dedicated-handheld
didj
doja
donner30
dragon-32-slash-64
dc
ecd-micromind
edsac--1
acorn-electron
enterprise
epoch-cassette-vision

epoch-game-pocket-computer
epoch-super-cassette-vision
evercade
exen
exelvision
exidy-sorcerer
fm-towns
fm-7
fairchild-channel-f
famicom
fds
mobile-custom
nimrod
amazon-fire-tv
freebox
g-cluster
gimini
gnex
gp2x
gp2x-wiz
gp32
gvm
galaksija
gamate
game-and-watch
gb
gba
gbc
gamegear
game-wave
game-dot-com
game-dot-com
ngc
gamestick
gear-vr
genesis-slash-megadrive
gizmondo
gloud
glulx
stadia

hd-dvd-player
hp2100
hp3000
hp-9800
hp-programmable-calculator
handheld-electronic-lcd
heathzenith
heathkit-h11
hitachi-s1
hugo
hyper-neo-geo-64
hyperscan
ibm-5100
ideal-computer
intel-8008
intel-8080
intel-8086
intellivision
intellivision-amico
interact-model-one
interton-video-2000
j2me
jolt
jupiter-ace
kim-1
kaios
kindle
laser200
laseractive
leaptv
leapster
leapster-explorer-slash-leadpad-explorer
leapster-explorer-slash-leadpad-explorer
legacy-computer
mobile
linux
luna
mos-technology-6502
mre
msx

msx2
mac
mac
maemo
mainframe
matsushitapanasonic-jr
mattel-aquarius
meego
mega-duck-slash-cougar-boy
memotech-mtx
meritum
meta-quest-2
meta-quest-3
microbee
microtan-65
microvision--1
mophun
motorola-6800
motorola-68k
ngage
ngage2
nec-pc-6000-series
nascom
neogeoaes
neogeoaes
neo-geo-cd
neogeomvs
neo-geo-pocket
neo-geo-pocket-color
neo-geo-x
new-nintendo-3ds
newbrain
newton
3ds
n64
nintendo-64dd
nds
nintendo-dsi
nes
ngc

nintendo-playstation
switch
northstar
noval-760
nuon
ooparts
os2
oculus-go
oculus-quest
oculus-rift
odyssey--1
odyssey-2-slash-videopac-g7000
ohio-scientific
onlive-game-system
orao
oric
ouya
win
pc-booter
supergrafx
pc-50x-family
pc-6001
pc-8000
pc-8800-series
pc-9800-series
pc-fx
pdp1
pdp10
pdp11
pdp-8--1
pico
plato--1
psvita
palm-os
panasonic-jungle
panasonic-m2
pandora
pebble
philips-cd-i
philips-vg-5000

photocd
pippin
ps
ps2
ps3
ps4--1
ps5
playstation-now
psp
psvr
psvr2
psvita
playdate
playdia
plex-arcade
plug-and-play
pocketstation
pokitto
pokemon-mini
poly-88
r-zone
rca-studio-ii
research-machines-380z
roku
sam-coupe
scmp
sd-200270290
sdssigma7
sega-32x
segacd
sega-master-system
saturn
sg1000
sk-vm
smc-777
sri-5001000
swtpc-6800
satellaview
sega32
segacd

gamegear
sms
genesis-slash-megadrive
sega-pico
saturn
sharp-mz-2200
sharp-mz-80b20002500
sharp-mz-80k7008001500
x1
sharp-x68000
sharp-zaurus
signetics-2650
sinclair-ql
sinclair-zx81
socrates
sol-20
sord-m5
spectravideo
super-acan
sfam
snes
super-vision-8000
supervision
sure-shot-hd
swanycrystal
symbian
tads
ti-programmable-calculator
ti-994a
tim
trs-80
trs-80-color-computer
trs-80-mc-10
trs-80-model-100
taito-x-55
zod
tattung-einstein
tektronix-4050
tele-spiel
telstar-arcade

terebikko-slash-see-n-say-video-phone
terminal
ti-99
thomson-mo5
thomson-to
tiki-100
timex-sinclair-2068
tizen
tomahawk-f1
tomy-tutor
triton
turbografx-16-slash-pc-engine-cd
turbografx16--1
turbografx16--1
turbografx-16-slash-pc-engine-cd
vflash
vsmile
vc-4000
vis
vectrex
versatile
videobrain
videopac-g7400
virtualboy
vc
visual-memory-unit-slash-visual-memory-system
wipi
wang2200
watara-slash-quickshot-supervision
browser
wii
wiiu
win
win3x
windows-apps
windows-mobile
winphone
wonderswan
wonderswan-color
xavixport

xbox
xbox360
xboxcloudgaming
xboxone
series-x
xerox-alto
z-machine
zxs
zx-spectrum-next
zx80
sinclair-zx81
zeebo
z80
zilog-z8000
zodiac
zune
bada
digiblast
ios
ipad
ipod-classic
iircade
tvos
visionos
watchos
webos

Create-WorkloadFolder

Location Agnostic Version v1.3

```
#####  
# AUTHOR      : Ryan Mutschler  
# DATE        : 3-19-2025  
# EDIT        : 3-21-2025  
# PURPOSE     : This script creates folders for Changes, Incidents, or Projects following the  
specified naming convention  
# REPOSITORY: https://github.com/MutschlerHomeTech/Public-  
Scripts/blob/master/1.%20Full%20Scripts/Windows/File%20System/Create-WorkloadFolder.ps1  
# WIKIPEDIA  : https://wikipedia.mutschlerhome.com/books/windows-scripts/page/create-  
workloadfolder-v12  
#  
# VERSION    : 1.0      (Initial release)  
# VERSION    : 1.1      (URL subfolder creation)  
# VERSION    : 1.2      (Communications subfolder creation)  
# VERSION    : 1.3      (Added illegal character handling)  
#####  
  
# Function to check for illegal Windows folder name characters  
function Test-IllegalCharacters {  
    param (  
        [string]$FolderName  
    )  
  
    return $FolderName -match '[\\\/\:\*\?\"<\>\\|]'  
}  
  
# Function to remove illegal Windows folder name characters  
function Remove-IllegalCharacters {  
    param (  
        [string]$FolderName  
    )
```

```

# Replace illegal characters with underscores or appropriate alternatives
$cleanName = $FolderName -replace '[\\\/\:\*\?\"<\>|]', '_'
return $cleanName
}

# Get the base workload folder path by reading from the Windows Registry
try {
    # Get the Documents folder path from Registry
    $documentsPath = (Get-ItemProperty -Path
"HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders" -Name
"Personal").Personal

    # Expand environment variables if they exist in the path
    $documentsPath = [System.Environment]::ExpandEnvironmentVariables($documentsPath)

    # Build the workload path
    $workloadBasePath = Join-Path -Path $documentsPath -ChildPath "Workload"
}
catch {
    Write-Host "Error accessing Registry. Falling back to default Documents location." -
ForegroundColor Yellow
    $workloadBasePath = Join-Path -Path $env:USERPROFILE -ChildPath "Documents\Workload"
}

# Check if the workload folder exists, create it if not
if (-not (Test-Path -Path $workloadBasePath)) {
    Write-Host "Creating base Workload folder structure..."
    New-Item -Path $workloadBasePath -ItemType Directory | Out-Null

    # Create main category folders
    $categories = @("Changes", "Incidents", "Projects")
    foreach ($category in $categories) {
        $categoryPath = Join-Path -Path $workloadBasePath -ChildPath $category
        New-Item -Path $categoryPath -ItemType Directory | Out-Null

        # Create subcategories based on the parent folder
        $subcategories = @()
        switch ($category) {
            "Changes" {
                $subcategories = @(

```

```

        "1. Discovery",
        "2. Testing",
        "3. Implementation",
        "4. Completed",
        "99. Uncategorized"
    )
}
"Incidents" {
    $subcategories = @(
        "1. Investigation",
        "2. On Hold",
        "3. Resolved",
        "99. Uncategorized"
    )
}
"Projects" {
    $subcategories = @(
        "1. Discovery",
        "2. Implementation",
        "3. Maintenance",
        "4. Decommissioned",
        "99. Uncategorized"
    )
}
}

# Create each subcategory folder
foreach ($subcategory in $subcategories) {
    $subcategoryPath = Join-Path -Path $categoryPath -ChildPath $subcategory
    New-Item -Path $subcategoryPath -ItemType Directory | Out-Null
}
}
Write-Host "Base folder structure created successfully."
}

# Main script execution starts here
Write-Host "=== Workload Folder Creation Tool ===" -ForegroundColor Cyan
Write-Host "This script will create a new folder for your workload item."

# Prompt for workload type

```

```

$validSelection = $false
$workloadType = ""
$prefix = ""

while (-not $validSelection) {
    Write-Host "`nSelect the type of workload:" -ForegroundColor Yellow
    Write-Host "1. Change"
    Write-Host "2. Incident"
    Write-Host "3. Project"
    $selection = Read-Host "Enter your selection (1-3)"

    switch ($selection) {
        "1" {
            $workloadType = "Changes"
            $prefix = "RMCHG"
            $validSelection = $true
        }
        "2" {
            $workloadType = "Incidents"
            $prefix = "RMINC"
            $validSelection = $true
        }
        "3" {
            $workloadType = "Projects"
            $prefix = "RMPRJ"
            $validSelection = $true
        }
        default {
            Write-Host "Invalid selection. Please enter a number between 1 and 3." -
ForegroundColor Red
        }
    }
}

# Get the workload name
$workloadName = ""
$validName = $false
while (-not $validName) {
    $workloadName = Read-Host "Enter the name of the $($workloadType.TrimEnd('s'))"
}

```

```

if ([string]::IsNullOrEmpty($workloadName)) {
    Write-Host "Name cannot be empty. Please enter a valid name." -ForegroundColor Red
}
elseif ($workloadName -match "\/") {
    # Found forward slash, ask user what to do
    Write-Host "`nThe name contains a forward slash (/), which can create subfolders." -
ForegroundColor Yellow
    Write-Host "1. Create subfolders (e.g., 'folder/subfolder' creates 'folder' with
'subfolder' inside)"
    Write-Host "2. Remove illegal characters and use a single folder"
    $slashChoice = Read-Host "Enter your choice (1-2)"

    if ($slashChoice -eq "1") {
        # User wants subfolders, proceed with the name as is
        $validName = $true
    }
    else {
        # User wants to remove illegal characters
        #$originalName = $workloadName
        $workloadName = Remove-IllegalCharacters -FolderName $workloadName
        Write-Host "Cleaned name: '$workloadName'" -ForegroundColor Cyan
        $confirmName = Read-Host "Is this acceptable? (Y/N)"
        if ($confirmName.ToUpper() -eq "Y") {
            $validName = $true
        }
    }
}
elseif (Test-IllegalCharacters -FolderName $workloadName) {
    # Other illegal characters found
    Write-Host "The name contains illegal Windows folder characters." -ForegroundColor
Yellow
    #$originalName = $workloadName
    $workloadName = Remove-IllegalCharacters -FolderName $workloadName
    Write-Host "Cleaned name: '$workloadName'" -ForegroundColor Cyan
    $confirmName = Read-Host "Is this acceptable? (Y/N)"
    if ($confirmName.ToUpper() -eq "Y") {
        $validName = $true
    }
}
else {

```

```

        $validName = $true
    }
}

# Get current date in the required format (YYMMdd)
$currentDate = Get-Date -Format "yyMMdd"

# Determine the next available sequence number
function Get-NextSequenceNumber {
    param (
        [string]$TargetPath,
        [string]$Prefix,
        [string]$Date
    )

    # Get all folders that match the pattern for today in the target path only
    $pattern = "$Prefix$Date*"
    $existingFolders = Get-ChildItem -Path $TargetPath -Directory |
        Where-Object { $_.Name -like $pattern }

    if ($existingFolders.Count -eq 0) {
        # No folders exist for today, start with 001
        return "001"
    }

    # Extract sequence numbers from existing folders
    $sequenceNumbers = @()
    foreach ($folder in $existingFolders) {
        if ($folder.Name -match "$Prefix$Date(\d{3})") {
            $sequenceNumbers += [int]$matches[1]
        }
    }

    if ($sequenceNumbers.Count -eq 0) {
        # No sequence numbers found, start with 001
        return "001"
    }

    # Find the highest sequence number and increment by 1
    $nextNumber = ($sequenceNumbers | Measure-Object -Maximum).Maximum + 1

```

```
    return $nextNumber.ToString("000")
}

# Get the next sequence number for the specific subcategory folder
$sequenceNumber = Get-NextSequenceNumber -TargetPath $subcategoryPath -Prefix $prefix -Date
$currentDate

# Build the folder name with sequence number
$folderName = "$prefix$currentDate$sequenceNumber - $workloadName"

# Prompt for subcategory
Write-Host "`nSelect the subcategory:" -ForegroundColor Yellow
$subcategories = @()
switch ($workloadType) {
    "Changes" {
        Write-Host "1. Discovery"
        Write-Host "2. Testing"
        Write-Host "3. Implementation"
        Write-Host "4. Completed"
        Write-Host "5. Uncategorized"
        $subcategories = @(
            "1. Discovery",
            "2. Testing",
            "3. Implementation",
            "4. Completed",
            "99. Uncategorized"
        )
    }
    "Incidents" {
        Write-Host "1. Investigation"
        Write-Host "2. On Hold"
        Write-Host "3. Resolved"
        Write-Host "4. Uncategorized"
        $subcategories = @(
            "1. Investigation",
            "2. On Hold",
            "3. Resolved",
            "99. Uncategorized"
        )
    }
}
```

```

"Projects" {
    Write-Host "1. Discovery"
    Write-Host "2. Implementation"
    Write-Host "3. Maintenance"
    Write-Host "4. Decommissioned"
    Write-Host "5. Uncategorized"
    $subcategories = @(
        "1. Discovery",
        "2. Implementation",
        "3. Maintenance",
        "4. Decommissioned",
        "99. Uncategorized"
    )
}

}

$validSubcategory = $false
$subcategoryPath = ""

while (-not $validSubcategory) {
    $maxOption = if ($workloadType -eq "Incidents") { 4 } else { 5 }
    $subcategorySelection = Read-Host "Enter your selection (1-$maxOption)"

    $subcategoryIndex = [int]$subcategorySelection - 1

    # Handle "Uncategorized" as special case
    if ($subcategorySelection -eq $maxOption.ToString()) {
        $subcategoryIndex = $subcategories.Count - 1
    }

    if ($subcategoryIndex -ge 0 -and $subcategoryIndex -lt $subcategories.Count) {
        $subcategoryPath = Join-Path -Path $workloadBasePath -ChildPath
"$workloadType\$($subcategories[$subcategoryIndex])"
        $validSubcategory = $true
    }
    else {
        Write-Host "Invalid selection. Please enter a number between 1 and $maxOption." -
ForegroundColor Red
    }
}
}

```

```
# Create the full path for the new folder
$newFolderPath = Join-Path -Path $subcategoryPath -ChildPath $folderName

# Check if the folder already exists
if (Test-Path -Path $newFolderPath) {
    Write-Host "`nWarning: A folder with this name already exists at this location." -
ForegroundColor Yellow
    $confirmation = Read-Host "Would you like to create it anyway? (Y/N)"

    if ($confirmation.ToUpper() -ne "Y") {
        Write-Host "Operation cancelled by user." -ForegroundColor Red
        exit
    }
}

# Create the folder
try {
    # Create the main workload folder
    New-Item -Path $newFolderPath -ItemType Directory | Out-Null

    # Create subfolders within the new workload folder
    $urlFolderPath = Join-Path -Path $newFolderPath -ChildPath "URL"
    $commsFolderPath = Join-Path -Path $newFolderPath -ChildPath "Communications"

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

    Write-Host "`nSuccess! Created folder:" -ForegroundColor Green
    Write-Host $newFolderPath
    Write-Host "Subfolders created:"
    Write-Host "- URL: $urlFolderPath"
    Write-Host "- Communications: $commsFolderPath"

    # Open the folder in Explorer
    $openFolder = Read-Host "Would you like to open the folder now? (Y/N)"
    if ($openFolder.ToUpper() -eq "Y") {
        Invoke-Item -Path $newFolderPath
    }
}
```

```
catch {  
    Write-Host "`nError creating folder: $_" -ForegroundColor Red  
}
```

Version 0.1

Features:

- 1. Base Folder Structure Setup:**
 - Creates the Workload folder structure if it doesn't already exist
 - Establishes the correct hierarchy for Changes, Incidents, and Projects with appropriate subcategories
- 2. User Interaction:**
 - Prompts the user to select the workload type (Change, Incident, or Project)
 - Asks for the name of the workload item
 - Lets the user select which subcategory the item belongs in
- 3. Naming Convention:**
 - Automatically applies the correct prefix (RMCHG, RMINC, or RMPRJ)
 - Includes the current date in YYMMDD format
 - Follows the specified format: [PREFIX][DATE] - [NAME]
- 4. Error Handling:**
 - Validates user input
 - Checks if folders already exist
 - Provides appropriate error messages
- 5. Convenience Features:**
 - Option to open the newly created folder directly after creation
 - Clear user interface with color-coded prompts

Usage:

1. Simply run the script in PowerShell
2. Follow the prompts to select workload type, name, and subcategory
3. The script will create the folder with the proper naming convention in the correct location

Version 0.2

Updated the script to retrieve the user's Documents folder location from the Windows Registry instead of assuming the default location. This change makes the script more robust by:

1. Reading the actual Documents folder path from the Registry key:

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders\Personal
```

2. Expanding any environment variables that might be in the path
3. Including error handling that falls back to the default location if the Registry read fails for any reason

This modification ensures that the script will correctly place the Workload folder in the user's actual Documents folder location, even if they've customized it to be somewhere other than the default `%USERPROFILE%\Documents`.

The rest of the script's functionality remains unchanged - it will still create the proper folder structure according to the requirements and follow the specified naming convention.

Version 0.3

I've updated the script to include logic for sequential numbering of folders created on the same day. Here's how the new functionality works:

1. Sequential Numbering System:

- Added a 3-digit sequence number (starting with "001") between the date and the folder name
- The format is now: `[PREFIX][DATE][SEQUENCE] - [NAME]` (e.g., "RMPRJ250319001 - Project Name")

2. Automatic Sequence Detection:

- The script searches all subfolders in the Workload structure for folders with the same prefix and date
- It identifies the highest existing sequence number and increments it by 1 for the new folder

3. Implementation Details:

- Created a new function `Get-NextSequenceNumber` that:
 - Searches recursively through all subfolders in the Workload structure
 - Uses regex pattern matching to extract existing sequence numbers
 - Determines the next available sequence number
 - Formats the number with leading zeros (e.g., "001", "002", etc.)

4. Handling Edge Cases:

- If no folders exist for the current date, it starts with "001"
- If folders exist but don't follow the sequence numbering pattern, it still starts with "001"
- Handles multiple folders across different subcategories (e.g., if there's already a "001" in Discovery and a "002" in Testing, the next folder will be "003")

This enhancement ensures that each folder has a unique identifier, even if multiple folders are created on the same day with the same type and name.

Version 1.0

I've updated the script to only scan for existing folders within the specific subcategory folder where the new folder will be created, rather than searching the entire Workload structure. Here are the key changes:

1. Limited Scope for Sequence Number Search:

- Modified the `Get-NextSequenceNumber` function to only check within the target subcategory folder
- Removed the `-Recurse` parameter from the `Get-ChildItem` command to prevent searching through subfolders
- Changed the parameter name from `BasePath` to `TargetPath` to better reflect its purpose

2. Sequence Number Generation:

- Now the sequence numbers will only be unique within each subcategory folder
- For example, you could have "RMPRJ250319001 - Test Project" in the "1. Discovery" folder and another "RMPRJ250319001 - Different Project" in the "2. Implementation" folder

3. Function Call Update:

- Updated the function call to pass the specific subcategory path rather than the workload base path

This change makes the sequence numbering more compartmentalized, ensuring that folders in different subcategories don't affect each other's numbering. The sequence will now only increment based on existing folders within the same subcategory.

Version 1.1

I've updated the script to automatically create a URL subfolder within each new workload folder. Here's what the script does now:

1. When a new workload folder is created (e.g., "RMPRJ250319001 - Test Project"), the script will:
 - First create the main workload folder as before
 - Then create a subfolder named "URL" inside that workload folder
2. The success message has been enhanced to show:
 - The path of the main workload folder that was created
 - Confirmation that the URL subfolder was created, along with its path
3. When you choose to open the folder after creation, it will still open the main workload folder (not the URL subfolder directly)

All other functionality remains the same. This enhancement ensures that each workload folder automatically has a URL subfolder ready for use, eliminating the need to manually create it

later.

Version 1.2

I've updated the script to create both "URL" and "Communications" subfolders within each new workload folder. Here's what the script now includes:

1. When a new workload folder is created, the script will automatically:
 - Create the main workload folder (e.g., "RMPRJ250319001 - Test Project")
 - Create a "URL" subfolder inside the workload folder
 - Create a "Communications" subfolder inside the workload folder
2. The success message has been improved to:
 - Show the path of the main workload folder
 - List all created subfolders with their paths in a clear, bulleted format
3. All other functionality remains the same, including:
 - The option to open the main folder in Explorer after creation
 - Error handling for any issues during folder creation

This update ensures that each workload folder is created with a consistent internal structure, with both URL and Communications subfolders ready for use from the start.

Version 1.3

I've updated the script to handle forward slashes and other illegal characters in folder names. Here's how the new functionality works:

Forward Slash Handling:

1. **Detection and User Options:**
 - When a user enters a folder name containing a forward slash ("/"), the script now detects this and presents two options:
 1. Create subfolders (using the slash as a directory separator)
 2. Remove illegal characters and use a single folder
2. **User Experience:**
 - If the user chooses option 1 (create subfolders), the script will use the name as-is, allowing Windows to create the nested folder structure
 - If the user chooses option 2 (remove illegal characters), the script will:
 - Replace all illegal characters (including slashes) with underscores
 - Show the "cleaned" folder name to the user
 - Ask for confirmation before proceeding

Handling Other Illegal Characters:

The script now also detects and handles other illegal Windows folder name characters:

- Backslash (\)
- Forward slash (/)
- Colon (:)
- Asterisk (*)
- Question mark (?)
- Double quote (")
- Less than (<)
- Greater than (>)
- Pipe (|)

If any of these characters are detected (except forward slashes, which get the special treatment described above), the script automatically:

1. Suggests a cleaned version of the name with illegal characters replaced by underscores
2. Asks the user to confirm if the cleaned name is acceptable
3. Loops back to prompt for a new name if the user rejects the cleaned version

This update ensures that the script can handle a wide variety of folder name inputs while giving users control over how forward slashes specifically are handled.

MutschlerHome Version v1.6

```
#####  
# AUTHOR      : Ryan Mutschler  
# DATE        : 3-19-2025  
# EDIT        : 3-21-2025  
# PURPOSE     : This script creates folders for Changes, Incidents, or Projects following the  
specified naming convention  
# REPOSITORY: https://github.com/MutschlerHomeTech/Public-  
Scripts/blob/master/1.%20Full%20Scripts/Windows/File%20System/Create-WorkloadFolder.ps1  
# WIKIPEDIA  : https://wikipedia.mutschlerhome.com/books/windows-scripts/page/create-  
workloadfolder-v12
```

```

#
# VERSION : 1.0 (Initial release)
# VERSION : 1.1 (URL subfolder creation)
# VERSION : 1.2 (Communications subfolder creation)
# VERSION : 1.3 (Added illegal character handling)
# VERSION : 1.4 (Changed to MutschlerHome folder structure)
# VERSION : 1.5 (Corrected subfolder structure)
# VERSION : 1.6 (Corrected subfolders to have numeric values as the prefix)
#####

# Function to check for illegal Windows folder name characters
function Test-IllegalCharacters {
    param (
        [string]$FolderName
    )

    return $FolderName -match '[\\\/\:\*\?\"<>|]'
}

# Function to remove illegal Windows folder name characters
function Remove-IllegalCharacters {
    param (
        [string]$FolderName
    )

    # Replace illegal characters with underscores or appropriate alternatives
    $cleanName = $FolderName -replace '[\\\/\:\*\?\"<>|]', '_'
    return $cleanName
}

# Get the base workload folder path by reading from the Windows Registry
try {
    # Get the Documents folder path from Registry
    $documentsPath = (Get-ItemProperty -Path
"HKCU:\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders" -Name
"Personal").Personal

    # Expand environment variables if they exist in the path
    $documentsPath = [System.Environment]::ExpandEnvironmentVariables($documentsPath)
}

```

```

# Build the workload path
$workloadBasePath = Join-Path -Path $documentsPath -ChildPath "MutschlerHome"
}
catch {
    Write-Host "Error accessing Registry. Falling back to default Documents location." -
ForegroundColor Yellow
    $workloadBasePath = Join-Path -Path $env:USERPROFILE -ChildPath "Documents\MutschlerHome"
}

# Check if the workload folder exists, create it if not
if (-not (Test-Path -Path $workloadBasePath)) {
    Write-Host "Creating base MutschlerHome folder structure..."
    New-Item -Path $workloadBasePath -ItemType Directory | Out-Null

    # Create main category folders
    $categories = @("1. Public Projects", "2. Internal Projects", "3. Work Projects", "4.
Incidents")
    foreach ($category in $categories) {
        $categoryPath = Join-Path -Path $workloadBasePath -ChildPath $category
        New-Item -Path $categoryPath -ItemType Directory | Out-Null

        # Create subcategories based on the parent folder
        $subcategories = @(
switch ($category) {
    "1. Public Projects" {
        $subcategories = @(
            "1. Discovery",
            "2. Implementation",
            "3. Maintenance",
            "4. Decommissioned",
            "99. Uncategorized"
        )
    }
    "2. Internal Projects" {
        $subcategories = @(
            "1. Discovery",
            "2. Implementation",
            "3. Maintenance",
            "4. Decommissioned",
            "99. Uncategorized"

```

```

        )
    }
    "3. Work Projects" {
        $subcategories = @(
            "1. Discovery",
            "2. Implementation",
            "3. Maintenance",
            "4. Decommissioned",
            "99. Uncategorized"
        )
    }
    "4. Incidents" {
        $subcategories = @(
            "1. Investigation",
            "2. On Hold",
            "3. Resolved",
            "99. Uncategorized"
        )
    }
}

# Create each subcategory folder
foreach ($subcategory in $subcategories) {
    $subcategoryPath = Join-Path -Path $categoryPath -ChildPath $subcategory
    New-Item -Path $subcategoryPath -ItemType Directory | Out-Null
}

}

Write-Host "Base folder structure created successfully."
}

# Main script execution starts here
Write-Host "=== MutschlerHome Folder Creation Tool ===" -ForegroundColor Cyan
Write-Host "This script will create a new folder for your workload item."

# Prompt for workload type
$validSelection = $false
$workloadType = ""
$prefix = ""

while (-not $validSelection) {

```

```

Write-Host "`nSelect the type of workload:" -ForegroundColor Yellow
Write-Host "1. Public Project"
Write-Host "2. Internal Project"
Write-Host "3. Work Project"
Write-Host "4. Incident"
$selection = Read-Host "Enter your selection (1-4)"

switch ($selection) {
    "1" {
        $workloadType = "1. Public Projects"
        $prefix = "RMPPRJ"
        $validSelection = $true
    }
    "2" {
        $workloadType = "2. Internal Projects"
        $prefix = "RMIPRJ"
        $validSelection = $true
    }
    "3" {
        $workloadType = "3. Work Projects"
        $prefix = "RMWPRJ"
        $validSelection = $true
    }
    "4" {
        $workloadType = "4. Incidents"
        $prefix = "RMINC"
        $validSelection = $true
    }
    default {
        Write-Host "Invalid selection. Please enter a number between 1 and 4." -
ForegroundColor Red
    }
}

# Get the workload name
$workloadName = ""
$validName = $false
while (-not $validName) {
    $workloadName = Read-Host "Enter the name of the $($workloadType.TrimEnd('s'))"
}

```

```

if ([string]::IsNullOrEmpty($workloadName)) {
    Write-Host "Name cannot be empty. Please enter a valid name." -ForegroundColor Red
}
elseif ($workloadName -match "\/") {
    # Found forward slash, ask user what to do
    Write-Host "`nThe name contains a forward slash (/), which can create subfolders." -
ForegroundColor Yellow
    Write-Host "1. Create subfolders (e.g., 'folder/subfolder' creates 'folder' with
'subfolder' inside)"
    Write-Host "2. Remove illegal characters and use a single folder"
    $slashChoice = Read-Host "Enter your choice (1-2)"

    if ($slashChoice -eq "1") {
        # User wants subfolders, proceed with the name as is
        $validName = $true
    }
    else {
        # User wants to remove illegal characters
        #$originalName = $workloadName
        $workloadName = Remove-IllegalCharacters -FolderName $workloadName
        Write-Host "Cleaned name: '$workloadName'" -ForegroundColor Cyan
        $confirmName = Read-Host "Is this acceptable? (Y/N)"
        if ($confirmName.ToUpper() -eq "Y") {
            $validName = $true
        }
    }
}
elseif (Test-IllegalCharacters -FolderName $workloadName) {
    # Other illegal characters found
    Write-Host "The name contains illegal Windows folder characters." -ForegroundColor
Yellow
    #$originalName = $workloadName
    $workloadName = Remove-IllegalCharacters -FolderName $workloadName
    Write-Host "Cleaned name: '$workloadName'" -ForegroundColor Cyan
    $confirmName = Read-Host "Is this acceptable? (Y/N)"
    if ($confirmName.ToUpper() -eq "Y") {
        $validName = $true
    }
}
}

```

```

else {
    $validName = $true
}
}

# Get current date in the required format (YYMMdd)
$currentDate = Get-Date -Format "yyMMdd"

# Determine the next available sequence number
function Get-NextSequenceNumber {
    param (
        [string]$TargetPath,
        [string]$Prefix,
        [string]$Date
    )

    # Get all folders that match the pattern for today in the target path only
    $pattern = "$Prefix$Date*"
    $existingFolders = Get-ChildItem -Path $TargetPath -Directory |
        Where-Object { $_.Name -like $pattern }

    if ($existingFolders.Count -eq 0) {
        # No folders exist for today, start with 001
        return "001"
    }

    # Extract sequence numbers from existing folders
    $sequenceNumbers = @()
    foreach ($folder in $existingFolders) {
        if ($folder.Name -match "$Prefix$Date(\d{3})") {
            $sequenceNumbers += [int]$matches[1]
        }
    }

    if ($sequenceNumbers.Count -eq 0) {
        # No sequence numbers found, start with 001
        return "001"
    }

    # Find the highest sequence number and increment by 1

```

```

    $nextNumber = ($sequenceNumbers | Measure-Object -Maximum).Maximum + 1
    return $nextNumber.ToString("000")
}

# Get the next sequence number for the specific subcategory folder
$sequenceNumber = Get-NextSequenceNumber -TargetPath $subcategoryPath -Prefix $prefix -Date
$currentDate

# Build the folder name with sequence number
$folderName = "$prefix$currentDate$sequenceNumber - $workloadName"

# Prompt for subcategory
Write-Host "`nSelect the subcategory:" -ForegroundColor Yellow
$subcategories = @(
    switch ($workloadType) {
        "1. Public Projects" {
            Write-Host "1. Discovery"
            Write-Host "2. Implementation"
            Write-Host "3. Maintenance"
            Write-Host "4. Decommissioned"
            Write-Host "5. Uncategorized"
            $subcategories = @(
                "1. Discovery",
                "2. Implementation",
                "3. Maintenance",
                "4. Decommissioned",
                "99. Uncategorized"
            )
        }
        "2. Internal Projects" {
            Write-Host "1. Discovery"
            Write-Host "2. Implementation"
            Write-Host "3. Maintenance"
            Write-Host "4. Decommissioned"
            Write-Host "5. Uncategorized"
            $subcategories = @(
                "1. Discovery",
                "2. Implementation",
                "3. Maintenance",
                "4. Decommissioned",
            )
        }
    }
)

```

```

        "99. Uncategorized"
    )
}
"3. Work Projects" {
    Write-Host "1. Discovery"
    Write-Host "2. Implementation"
    Write-Host "3. Maintenance"
    Write-Host "4. Decommissioned"
    Write-Host "5. Uncategorized"
    $subcategories = @(
        "1. Discovery",
        "2. Implementation",
        "3. Maintenance",
        "4. Decommissioned",
        "99. Uncategorized"
    )
}
"4. Incidents" {
    Write-Host "1. Investigation"
    Write-Host "2. On Hold"
    Write-Host "3. Resolved"
    Write-Host "4. Uncategorized"
    $subcategories = @(
        "1. Investigation",
        "2. On Hold",
        "3. Resolved",
        "99. Uncategorized"
    )
}
}

$validSubcategory = $false
$subcategoryPath = ""

while (-not $validSubcategory) {
    $maxOption = if ($workloadType -eq "Incidents") { 4 } else { 5 }
    $subcategorySelection = Read-Host "Enter your selection (1-$maxOption)"

    $subcategoryIndex = [int]$subcategorySelection - 1

```

```

# Handle "Uncategorized" as special case
if ($subcategorySelection -eq $maxOption.ToString()) {
    $subcategoryIndex = $subcategories.Count - 1
}

if ($subcategoryIndex -ge 0 -and $subcategoryIndex -lt $subcategories.Count) {
    $subcategoryPath = Join-Path -Path $workloadBasePath -ChildPath
"$workloadType\$($subcategories[$subcategoryIndex])"
    $validSubcategory = $true
}
else {
    Write-Host "Invalid selection. Please enter a number between 1 and $maxOption." -
ForegroundColor Red
}
}

# Create the full path for the new folder
$newFolderPath = Join-Path -Path $subcategoryPath -ChildPath $folderName

# Check if the folder already exists
if (Test-Path -Path $newFolderPath) {
    Write-Host "`nWarning: A folder with this name already exists at this location." -
ForegroundColor Yellow
    $confirmation = Read-Host "Would you like to create it anyway? (Y/N)"

    if ($confirmation.ToUpper() -ne "Y") {
        Write-Host "Operation cancelled by user." -ForegroundColor Red
        exit
    }
}

# Create the folder
try {
    # Create the main workload folder
    New-Item -Path $newFolderPath -ItemType Directory | Out-Null

    # Create subfolders within the new workload folder
    $urlFolderPath = Join-Path -Path $newFolderPath -ChildPath "URL"
    $commsFolderPath = Join-Path -Path $newFolderPath -ChildPath "Communications"
}

```

```

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

Write-Host "`nSuccess! Created folder:" -ForegroundColor Green
Write-Host $newFolderPath
Write-Host "Subfolders created:"
Write-Host "- URL: $urlFolderPath"
Write-Host "- Communications: $commsFolderPath"

# Open the folder in Explorer
$openFolder = Read-Host "Would you like to open the folder now? (Y/N)"
if ($openFolder.ToUpper() -eq "Y") {
    Invoke-Item -Path $newFolderPath
}
}
catch {
    Write-Host "`nError creating folder: $_" -ForegroundColor Red
}

```

Version 0.1

Features:

1. Base Folder Structure Setup:

- Creates the Workload folder structure if it doesn't already exist
- Establishes the correct hierarchy for Changes, Incidents, and Projects with appropriate subcategories

2. User Interaction:

- Prompts the user to select the workload type (Change, Incident, or Project)
- Asks for the name of the workload item
- Lets the user select which subcategory the item belongs in

3. Naming Convention:

- Automatically applies the correct prefix (RMCHG, RMINC, or RMPRJ)
- Includes the current date in YYMMDD format
- Follows the specified format: [PREFIX][DATE] - [NAME]

4. Error Handling:

- Validates user input
- Checks if folders already exist
- Provides appropriate error messages

5. Convenience Features:

- Option to open the newly created folder directly after creation
- Clear user interface with color-coded prompts

Usage:

1. Simply run the script in PowerShell
2. Follow the prompts to select workload type, name, and subcategory
3. The script will create the folder with the proper naming convention in the correct location

Version 0.2

Updated the script to retrieve the user's Documents folder location from the Windows Registry instead of assuming the default location. This change makes the script more robust by:

1. Reading the actual Documents folder path from the Registry key:

```
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Explorer\User Shell Folders\Personal
```

2. Expanding any environment variables that might be in the path
3. Including error handling that falls back to the default location if the Registry read fails for any reason

This modification ensures that the script will correctly place the Workload folder in the user's actual Documents folder location, even if they've customized it to be somewhere other than the default `%USERPROFILE%\Documents`.

The rest of the script's functionality remains unchanged - it will still create the proper folder structure according to the requirements and follow the specified naming convention.

Version 0.3

I've updated the script to include logic for sequential numbering of folders created on the same day. Here's how the new functionality works:

1. Sequential Numbering System:

- Added a 3-digit sequence number (starting with "001") between the date and the folder name
- The format is now: `[PREFIX][DATE][SEQUENCE] - [NAME]` (e.g., "RMPRJ250319001 - Project Name")

2. Automatic Sequence Detection:

- The script searches all subfolders in the Workload structure for folders with the same prefix and date
- It identifies the highest existing sequence number and increments it by 1 for the new folder

3. Implementation Details:

- Created a new function `Get-NextSequenceNumber` that:
 - Searches recursively through all subfolders in the Workload structure
 - Uses regex pattern matching to extract existing sequence numbers
 - Determines the next available sequence number
 - Formats the number with leading zeros (e.g., "001", "002", etc.)

4. Handling Edge Cases:

- If no folders exist for the current date, it starts with "001"
- If folders exist but don't follow the sequence numbering pattern, it still starts with "001"
- Handles multiple folders across different subcategories (e.g., if there's already a "001" in Discovery and a "002" in Testing, the next folder will be "003")

This enhancement ensures that each folder has a unique identifier, even if multiple folders are created on the same day with the same type and name.

Version 1.0

I've updated the script to only scan for existing folders within the specific subcategory folder where the new folder will be created, rather than searching the entire Workload structure. Here are the key changes:

1. Limited Scope for Sequence Number Search:

- Modified the `Get-NextSequenceNumber` function to only check within the target subcategory folder
- Removed the `-Recurse` parameter from the `Get-ChildItem` command to prevent searching through subfolders
- Changed the parameter name from `BasePath` to `TargetPath` to better reflect its purpose

2. Sequence Number Generation:

- Now the sequence numbers will only be unique within each subcategory folder
- For example, you could have "RMPRJ250319001 - Test Project" in the "1. Discovery" folder and another "RMPRJ250319001 - Different Project" in the "2. Implementation" folder

3. Function Call Update:

- Updated the function call to pass the specific subcategory path rather than the workload base path

This change makes the sequence numbering more compartmentalized, ensuring that folders in different subcategories don't affect each other's numbering. The sequence will now only increment based on existing folders within the same subcategory.

Version 1.1

I've updated the script to automatically create a URL subfolder within each new workload folder. Here's what the script does now:

1. When a new workload folder is created (e.g., "RMPRJ250319001 - Test Project"), the script will:
 - First create the main workload folder as before
 - Then create a subfolder named "URL" inside that workload folder
2. The success message has been enhanced to show:
 - The path of the main workload folder that was created
 - Confirmation that the URL subfolder was created, along with its path
3. When you choose to open the folder after creation, it will still open the main workload folder (not the URL subfolder directly)

All other functionality remains the same. This enhancement ensures that each workload folder automatically has a URL subfolder ready for use, eliminating the need to manually create it later.

Version 1.2

I've updated the script to create both "URL" and "Communications" subfolders within each new workload folder. Here's what the script now includes:

1. When a new workload folder is created, the script will automatically:
 - Create the main workload folder (e.g., "RMPRJ250319001 - Test Project")
 - Create a "URL" subfolder inside the workload folder
 - Create a "Communications" subfolder inside the workload folder
2. The success message has been improved to:
 - Show the path of the main workload folder
 - List all created subfolders with their paths in a clear, bulleted format
3. All other functionality remains the same, including:
 - The option to open the main folder in Explorer after creation
 - Error handling for any issues during folder creation

This update ensures that each workload folder is created with a consistent internal structure, with both URL and Communications subfolders ready for use from the start.

I've updated the script to handle forward slashes and other illegal characters in folder names. Here's how the new functionality works:

Forward Slash Handling:

1. Detection and User Options:

- When a user enters a folder name containing a forward slash ("/"), the script now detects this and presents two options:
 1. Create subfolders (using the slash as a directory separator)
 2. Remove illegal characters and use a single folder

2. User Experience:

- If the user chooses option 1 (create subfolders), the script will use the name as-is, allowing Windows to create the nested folder structure
- If the user chooses option 2 (remove illegal characters), the script will:
 - Replace all illegal characters (including slashes) with underscores
 - Show the "cleaned" folder name to the user
 - Ask for confirmation before proceeding

Handling Other Illegal Characters:

The script now also detects and handles other illegal Windows folder name characters:

- Backslash (\)
- Forward slash (/)
- Colon (:)
- Asterisk (*)
- Question mark (?)
- Double quote (")
- Less than (<)
- Greater than (>)
- Pipe (|)

If any of these characters are detected (except forward slashes, which get the special treatment described above), the script automatically:

1. Suggests a cleaned version of the name with illegal characters replaced by underscores
2. Asks the user to confirm if the cleaned name is acceptable
3. Loops back to prompt for a new name if the user rejects the cleaned version

This update ensures that the script can handle a wide variety of folder name inputs while giving users control over how forward slashes specifically are handled.

Version 1.4

I've updated the script to use "MutschlerHome" as the root folder name instead of "Workload". The changes include:

1. Changed the base path construction to use "MutschlerHome" instead of "Workload":
 - Primary path (using Registry): `[Documents Path]\MutschlerHome`
 - Fallback path: `%USERPROFILE%\Documents\MutschlerHome`
2. Updated the title displayed in the console to "MutschlerHome Folder Creation Tool" to reflect the new root folder name

All the functionality remains the same, but now the script will create and work with folders under the "MutschlerHome" directory instead of "Workload". The internal folder structure (Changes, Incidents, Projects) and the naming conventions remain unchanged.

Is there anything else you'd like me to modify in the script?

Version 1.5

I've updated the script according to your specifications with the following changes:

1. Changed the main folder categories to:
 - Public Projects
 - Internal Projects
 - Work Projects
 - Incidents
2. Updated the prefix codes for each category:
 - RMPPRJ for Public Projects
 - RMIPRJ for Internal Projects
 - RMWPRJ for Work Projects
 - RMINC for Incidents (kept the same)
3. Modified the user selection prompts to reflect the new categories
4. Maintained the appropriate subcategory structure for each folder type:
 - All project types (Public, Internal, Work) use the project subcategories:
 - 1. Discovery
 - 2. Implementation
 - 3. Maintenance
 - 4. Decommissioned
 - 99. Uncategorized

- Incidents continue to use:
 - 1. Investigation
 - 2. On Hold
 - 3. Resolved
 - 99. Uncategorized

The script will now create the appropriate folder structures with the correct prefixes for each workload type you specified.

Version 1.6

I've updated the script to correct the folder names within the MutschlerHome root folder. All references to the subfolders now include their numerical prefixes:

1. Changed folder names in the categories array to:
 - "1. Public Projects"
 - "2. Internal Projects"
 - "3. Work Projects"
 - "4. Incidents"
2. Updated all references to these folders throughout the script, including:
 - In the switch statements for creating subcategories
 - In the menu selection logic
 - In the workload type assignment

The script will now create and reference folders with the correct numbered prefixes in their names. All functionality remains the same, with the proper prefixes for folder creation (RMPPRJ, RMIPRJ, RMWPRJ, and RMINC) maintained as specified in your previous request.