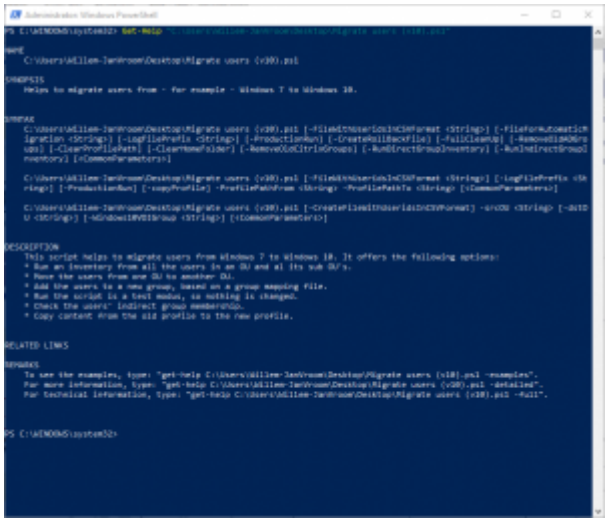
Improved script to migrate users from Windows 7 to Windows 10 automatically



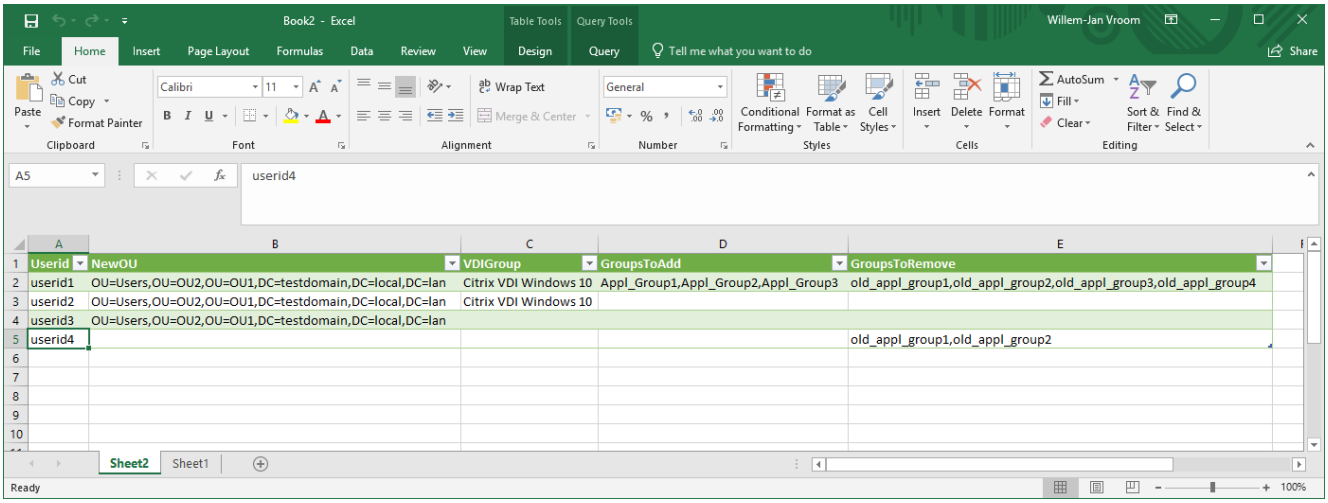
In my article, I wrote about a script to perform a user migration. In the meantime, that script has been improved to perform a more efficient and errorless migration.

The following improvements have been implemented:

- Provide a mappingsfile for an automatic migration. That mappingsfile contains both the old and the new group name.
- Perform a rollback in case of inpredicted results.
- In case of different profile locations: copy the old desktop and the old favorites to a new location.
- Improved help

To do this, I created a Powershell script.

The users who need to be migrated from Windows 7 to Windows 10 are in a csv file:



Userid	NewOU	VDIGroup	GroupsToAdd	GroupsToRemove
userid1	OU=Users,OU=OU2,OU=OU1,DC=testdomain,DC=local,DC=lan	Citrix VDI Windows 10	Appl_Group1,Appl_Group2,Appl_Group3	old_appl_group1,old_appl_group2,old_appl_group3,old_appl_group4
userid2	OU=Users,OU=OU2,OU=OU1,DC=testdomain,DC=local,DC=lan	Citrix VDI Windows 10		
userid3	OU=Users,OU=OU2,OU=OU1,DC=testdomain,DC=local,DC=lan			
userid4				old_appl_group1,old_appl_group2

- The column **Userid** contains the userid
- The column **NewOU** contains the OU where the user should be moved to. This column can be empty.
- The colum **VDIGroup** contains the new VDI group. This column can be empty.
- The colum **GroupsToAdd** contains the groups the user should be added to. Split the groups with a comma. This column can be empty.
- The colum **GroupsToRemove** contains the groups the user should be removed from. Split the groups with a comma. This column can be empty.

An example:

```
"Userid","NewOU","VDIGroup","GroupsToAdd","GroupsToRemove"
"userid1","OU=Users,OU=OU2,OU=OU1,DC=testdomain,DC=local,DC=lan","Citrix VDI Windows 10","Appl_Group1,Appl_Group2,Appl_Group3","old_appl_group1,old_appl_group2,old_appl_group3,old_appl_group4"
"userid2","OU=Users,OU=OU2,OU=OU1,DC=testdomain,DC=local,DC=lan","Citrix VDI Windows 10","",""
"userid3","OU=Users,OU=OU2,OU=OU1,DC=testdomain,DC=local,DC=lan","","","",""
"userid4","","","","old_appl_group1,old_appl_group2"
```

Parameters for a migration:

- **FileWithUseridsInCSVFormat**: CSV Filename that contains

all the userids that should be migrated. Default = the script name, with the csv extension.

- **FileForAutomaticMigration**: CSV filename that contains the old and new group name for a fully automated migration. If not specified, then there is no automatic migration. Default is empty.
- **LogFilePrefix**: The name the logfile starts with. So the logfiles are grouped together.
- **ProductionRun**: Use this swith to update Active Directory. If not specified, the script is run in test mode.
- **CreateRollBackFile**: Use this switch to create a rollback file. That makes it possible to perform a rollback in case of unpredicted behavior.
- **FullCleanUp**: Use this swith to remove all unneeded groups.
- **RemoveOldADGroups**: Use this switch to remove the old AD groups in case of an automated migration.
- **ClearProfilePath**: Use this switch to clear the profile path.
- **ClearHomeFolder**: Use this swicth to clear the drive mapping and home folder.
- **RemoveOldCitrixGroups**: Use this switch to remove all the old Citrix groups from the users' account.
- **RunDirectGroupInventory**: Use this switch to display all the direct group membership in the result file.
- **RunIndirectGroupInventory**: Use this switch to display all the indirect group membership in the result file.

Parameters for an user inventory, based on an OU:

- **CreateFileWithUseridsInCSVFormat**: Use this switch to create a File with all the userids to migrate based on an OU.
- **srcOU**: The OU (distinguised name) where all the userids are found that needs to be migrated. If the OU contains spaces, then add quotes around the OU name.

- **dstOU**: The destination OU (distinguised name). If this one can is empty then the users are not moved to another OU. If the OU contains spaces, then add quotes around the OU name.
- **Windows10VDIGroup**: The new Windows 10 VDI group. If the VDI group contains spaces, then add quotes around it.

Parameters to copy the desktop and internet explorer favorites to another location:

- **FileWithUseridsInCSVFormat**: CSV Filename that contains all the userids that should be migrated. Default = the script name, with the csv extension.
- **ProfilePathFrom**: The old profile path (exluding userid). If the profile path contains spaces, then add quotes around it.
- **ProfilePathTo**: The new profile path (exluding userid). If the profile path contains spaces, then add quotes around it.
- **ProductionRun**: Use this switch to update Active Directory. If not specified, the script is run in test mode.

The script:

```
<#
.SYNOPSIS
    Helps to migrate users from - for example - Windows 7 to Windows 10.

.DESCRIPTION
     This script helps to migrate users from Windows 7 to Windows 10. It offers the following options:
    * Run an inventory from all the users in an OU and al its sub OU's.
    * Move the users from one OU to another OU.
    * Add the users to a new group, based on a group mapping file.
    * Run the script is a test modus, so nothing is changed.
```

```
    * Check the users' indirect group membership.
    * Copy content from the old profile to the new profile.

.EXAMPLE
     Perform the actions as described in the CSV Input File 'Userids-to-migrate.csv' in test mode:
     ."\Migrate users (v10).ps1" -FileWithUseridsInCSVFormat Userids-to-migrate.csv

.EXAMPLE
     Perform the actions as described in the CSV Input File 'Userids-to-migrate.csv' and add the groups as per mappingsfile 'MappingOldGroupToNewGroup.csv' in test mode:
     ."\Migrate users (v10).ps1" -FileWithUseridsInCSVFormat Userids-to-migrate.csv      -FileForAutomaticMigration MappingOldGroupToNewGroup.csv

.EXAMPLE
     Perform the actions as described in the CSV Input File 'Userids-to-migrate.csv' and add the groups as per mappingsfile 'MappingOldGroupToNewGroup.csv' in production:
     ."\Migrate users (v10).ps1" -FileWithUseridsInCSVFormat Userids-to-migrate.csv      -FileForAutomaticMigration MappingOldGroupToNewGroup.csv -ProductionRun

.EXAMPLE
     Perform the actions as described in the CSV Input File 'Userids-to-migrate.csv', add the groups as per mappingsfile 'MappingOldGroupToNewGroup.csv' in production and create a rollback file:
     ."\Migrate users (v10).ps1" -FileWithUseridsInCSVFormat Userids-to-migrate.csv      -FileForAutomaticMigration MappingOldGroupToNewGroup.csv    -ProductionRun    -CreateRollBackFile

.EXAMPLE
     Perform a rollback in production
     ."\Migrate users (v10).ps1" -FileWithUseridsInCSVFormat RBF_.csv -ProductionRun

.EXAMPLE
```

Perform the actions as described in the CSV Input File 'Userids-to-migrate.csv' and add the groups as per mappingsfile 'MappingOldGroupToNewGroup.csv' in production.

Also run both the direct- and indirect group membership inventory:

."\Migrate users (v10).ps1" -FileWithUseridsInCSVFormat Userids-to-migrate.csv -FileForAutomaticMigration MappingOldGroupToNewGroup.csv -ProductionRun -RunDirectGroupInventory -RunIndirectGroupInventory

.EXAMPLE

Perform the actions as described in the CSV Input File 'Userids-to-migrate.csv', add the groups as per mappingsfile 'MappingOldGroupToNewGroup.csv' and remove the old group in production:

."\Migrate users (v10).ps1" -FileWithUseridsInCSVFormat Userids-to-migrate.csv -FileForAutomaticMigration MappingOldGroupToNewGroup.csv -ProductionRun -RemoveOldADGroups

.EXAMPLE

Perform the actions as described in the CSV Input File 'Userids-to-migrate.csv', add the groups as per mappingsfile 'MappingOldGroupToNewGroup.csv', remove the old group in production and run the inventory for indirect group membership:

."\Migrate users (v10).ps1" -FileWithUseridsInCSVFormat Userids-to-migrate.csv -FileForAutomaticMigration MappingOldGroupToNewGroup.csv -ProductionRun -RemoveOldADGroups -RunIndirectGroupInventory

.EXAMPLE

Create a csv file with all the userids to migrate from the OU testdomain.local.lan\OU1\Old OU

."\Migrate users (v10).ps1" -CreateFileWithUseridsInCSVFormat -srcOU "OU=Old OU,OU=OU1,DC=testdomain,DC=local,DC=lan"

.EXAMPLE

Create a csv file with all the userids to migrate from the OU testdomain.local.lan\OU1\Ould OU. The new Citrix VDI group is 'Citrix10VDI'

```
                ."\Migrate users (v10).ps1" -CreateFileWithUseridsInCSVFormat -srcOU "OU=Old OU,OU=OU1,DC=testdomain,DC=local,DC=lan" -Windows10VDIGroup Citrix10VDI

.EXAMPLE
     Create a csv file with all the userids to migrate from the OU testdomain.local.lan\OU1\Ould OU. The new OU is testdomain.local.lan\OU1\OU2\Users. The new Citrix VDI group is 'Citrix10VDI'
                ."\Migrate users (v10).ps1" -CreateFileWithUseridsInCSVFormat -srcOU "OU=Old OU,OU=OU1,DC=testdomain,DC=local,DC=lan"  -dstOU "OU=Users,OU=OU2,OU=OU1,DC=testdomain,DC=local,DC=lan" -Windows10VDIGroup Citrix10VDI

.EXAMPLE
     Migrate the IE favorites and the desktop folder from the old profile to the new profile in test mode:
      ."\Migrate users (v10).ps1" -FileWithUseridsInCSVFormat Userids-to-migrate.csv -copyProfile -ProfilePathFrom "\\server\share\w 7" -ProfilePathTo \\server\share\w10

.EXAMPLE
     Migrate the IE favorites and the desktop folder from the old profile to the new profile in production:
      ."\Migrate users (v10).ps1" -FileWithUseridsInCSVFormat Userids-to-migrate.csv -copyProfile -ProfilePathFrom "\\server\share\w 7" -ProfilePathTo \\server\share\w10 -ProductionRun

.NOTES
    Author:  Willem-Jan Vroom
    Website: https://www.vroom.cc/
    Twitter: @TheStingPilot

 v0.1:
   * Initial version.

 v0.2:
   * The logfilename has been changed.
```

```
 v0.3:
   * The logfile mentions 'RUNNING IN TEST MODE' in case the
swith -ProductionRun is not used.

 v1.0:
   * Introduction automatic migration.
   * Introduction Indirect Group Inventory
    * Automatic creation of the File with the userids to
migrate in CSV format.
   * Cleanup old Citrix groups
   * The results log file has a different lay-out
   * Parametersetnames
   * Improved help.
   * Copy desktop and favorites from the old profile to the
new profile.
   * Clear the homedir path in Active Directory.
   * Rollback scenario has been added.
#>

[CmdLetBinding()]

param
(
[CmdletBinding(DefaultParameterSetName = "Default")]

# CSV Filename that contains all the userids that should be
migrated. Default = the script name, with the csv extension.
# The filename with the users to be modified.
#
# This file has the following layout:
#
# "Userid","NewOU","VDIGroup","GroupsToAdd","GroupsToRemove"
#
"userid1","OU=Users,OU=OU2,OU=OU1,DC=testdomain,DC=local,DC=la
n","Citrix                  VDI                  Windows
10","Appl_Group1,Appl_Group2,Appl_Group3","old_appl_group1,old
_appl_group2,old_appl_group3,old_appl_group4"
#
"userid2","OU=Users,OU=OU2,OU=OU1,DC=testdomain,DC=local,DC=la
n","Citrix VDI Windows 10","",""
#
```

```
"userid3","OU=Users,OU=OU2,OU=OU1,DC=testdomain,DC=local,DC=la
n","","",""
# "userid4","","","","old_appl_group1,old_appl_group2"
#
# You can add more columns, but these columns are ignored.
[Parameter(Mandatory=$False,
           ParameterSetName="CopyProfile")]
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[String] $FileWithUseridsInCSVFormat = "",

# CSV filename that contains the old and new group name for a
fully automated migration. If not specified, then there is no
automatic migration. Defualt is empty.
#
# This file has the following layout:
#
# "OldGroup","NewGroup"
# "gg_appl_old1","appl_new1"
# "gg_appl_old2","appl_new2"
# "gg_appl_old1","appl_new3"
# "gg_appl_old4","appl_new4"
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[String] $FileForAutomaticMigration = "",

# The name the logfile starts with. So the logfiles are
grouped together.
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[Parameter(Mandatory=$False,
           ParameterSetName="CopyProfile")]

[String] $LogFilePrefix = "ZZZ-Logfile_",

# Use this swith to update Active Directory. If not specified,
the script is run in test mode.
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[Parameter(Mandatory=$False,
           ParameterSetName="CopyProfile")]
```

```
[Switch] $ProductionRun,

# Use this swith to create a rollback file. That makes it
possible to perform a rollback in case of unpredicted
behavior.
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[Switch] $CreateRollBackFile,

# Use this swith to remove all unneeded groups.
# If this switch is used then the following groups will be
removed from the users' account:
# * All gg_appl groups
# * WM-Users
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[Switch] $FullCleanUp,

# Use this swith to remove the old AD groups in case of an
automated migration.
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[Switch] $RemoveOldADGroups,

# Use this swith to clear the profile path.
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[Switch] $ClearProfilePath,

# Use this swith to clear the drive mapping and home folder.
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[Switch] $ClearHomeFolder,

# Use this swith to remove all the old Citrix groups from the
users' account.
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[Switch] $RemoveOldCitrixGroups,

# Use this swith to display all the direct group membership in
```

```
the result file.
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[Switch] $RunDirectGroupInventory,

# Use this swith to display all the indirect group membership
in the result file.
[Parameter(Mandatory=$False,
           ParameterSetName="Default")]
[Switch] $RunIndirectGroupInventory,

# Use this swith to create a File with all the userids to
migrate based on an OU.
[Parameter(Mandatory=$False,
ParameterSetName="CreateFileWithUseridsInCSVFormat")]
[Switch] $CreateFileWithUseridsInCSVFormat,

# The OU (distinguised name) where all the userids are found
that needs to be migrated. If the OU contains spaces, then add
quotes around the OU name.
[Parameter(Mandatory=$True,
ParameterSetName="CreateFileWithUseridsInCSVFormat")]
[string] $srcOU,

# The destination OU (distinguised name). If this one can is
empty then the users are not moved to another OU. If the OU
contains spaces, then add quotes around the OU name.
[Parameter(Mandatory=$False,
ParameterSetName="CreateFileWithUseridsInCSVFormat")]
[string] $dstOU,

# The new Windows 10 VDI group. If the VDI group contains
spaces, then add quotes around it.
[Parameter(Mandatory=$False,
ParameterSetName="CreateFileWithUseridsInCSVFormat")]
[string] $Windows10VDIGroup="",

# Use this switch if you want to copy desktop and favorites
from the users' old profile to the users' new profile.
[Parameter(Mandatory=$False,
           ParameterSetName="CopyProfile")]
```

```
[switch] $copyProfile,

# The old profile path (exluding userid). If the profile path
contains spaces, then add quotes around it.
[Parameter(Mandatory=$True,
           ParameterSetName="CopyProfile")]
[string] $ProfilePathFrom="",

# The new profile path (exluding userid). If the profile path
contains spaces, then add quotes around it.
[Parameter(Mandatory=$True,
           ParameterSetName="CopyProfile")]
[string] $ProfilePathTo=""

)

#
================================================================
================================================================
================
# Function block
#
================================================================
================================================================
================

Function Write-EntryToResultsFile
{

<#
.NOTES
================================================================
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       03-August-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Write-EntryToResultsFile
================================================================
================================================================
```

```
================
.SYNOPSIS

This function adds the success or failure information to the
array that contains the log
information.

#>
param
 (
  [string] $strUserid,
  [string] $Result  = "",
  [string] $Action  = "",
  [string] $Message = ""
 )
  $Record                                    =  [ordered]
@{"Timestamp"="";"Username"  =  "";"Result"=  "";"Action"=
"";"Message"= ""}
 $Record."Timestamp"    = (Get-Date -UFormat "%a %e %b %Y
%X").ToString()
 $Record."Username"     = $strUserid
 $Record."Result"       = $Result
 $Record."Action"       = $Action
 $Record."Information"  = $Message
 $objRecord               = New-Object PSObject -Property
$Record
 $Global:arrTable      += $objRecord
}

Function Export-ResultsLogFileToCSV
{

<#
.NOTES
================================================================
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       06-September-2018
Created by:       Willem-Jan Vroom
Organization:
```

```
Functionname:     Export-ResultsLogFileToCSV
=============================================================
=============================================================
================
.SYNOPSIS

This function writes the logfile content to a CSV file.

#>

 if($Global:arrTable.Count -gt 0)
   {
     $Global:arrTable | Export-Csv $strCSVLogFileSucces -
NoTypeInformation
   }
    Else
   {
    Write-Host "Something went wrong while writing the logfile
$strCSVLogFileSucces. Maybe nothing to report..."
   }
}

Function Export-RollBackFileToCSV
{

<#
.NOTES
=============================================================
=============================================================
================
Created with:     Windows PowerShell ISE
Created on:       29-November-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Export-RollBackFileToCSV
=============================================================
=============================================================
================
.SYNOPSIS

This function writes the rollbackfile content to a CSV file.
```

```
#>

 if($Global:arrTableWithRollBackRecords.Count -gt 0)
   {
      $Global:arrTableWithRollBackRecords | Export-Csv
$strCSVRollBackFile -NoTypeInformation
   }
    Else
   {
     Write-Host "Something went wrong while writing the file
$strCSVRollBackFile. Maybe nothing to report..."
   }
}

Function Remove-ProfilePathFromUserProfileInAD
{

<#
.NOTES
================================================================
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       06-September-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Remove-ProfilePathFromUserProfileInAD
================================================================
================================================================
================
.SYNOPSIS

This function clears the ProfilePath from AD.

#>

  param
   (
    [string] $strUserid
   )
```

```
    Write-EntryToResultsFile -strUserid $strUserid -Message
"Profilepath: $((Get-ADUser -Identity $strUserid -Properties
profilePath).profilePath)"  -Action  "Inventory"  -Result
"Success"

   Try
      {
                $strUserDN  =  (Get-ADUser  -Identity
$strUserid).distinguishedName
        Set-ADUser -Identity $strUserDN -Clear profilePath -
WhatIf:(-not($ProductionRun)) -Confirm:$false
       Write-EntryToResultsFile -strUserid $strUserid -Action
"Clear profile path" -Result "Success"
      }
       Catch
      {
       Write-EntryToResultsFile -strUserid $strUserid -Message
$_.Exception.Message  -Action "Clear profile path" -Result
"Error"
       Continue
      }
}

Function Remove-HomeFolderPathFromUserProfileInAD
{

<#
.NOTES
================================================================
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       06-September-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Remove-HomeFolderPathFromUserProfileInAD
================================================================
================================================================
================
.SYNOPSIS
```

```
This function clears the Home Folder Path from AD.

#>

  param
   (
    [string] $strUserid
   )

    Write-EntryToResultsFile -strUserid $strUserid -Message
"Homedrive:     $((Get-ADUser -Identity $strUserid -Properties
homeDrive).homeDrive)" -Action "Inventory" -Result "Success"
    Write-EntryToResultsFile -strUserid $strUserid -Message
"Homedirectory: $((Get-ADUser -Identity $strUserid -Properties
homeDirectory).homeDirectory)" -Action "Inventory" -Result
"Success"

   Try
      {
                $strUserDN = (Get-ADUser -Identity
$strUserid).distinguishedName
       Set-ADUser -Identity $strUserDN -Clear homeDirectory -
WhatIf:(-not($ProductionRun)) -Confirm:$false
       Write-EntryToResultsFile -strUserid $strUserid -Action
"Clear homedirectory path" -Result "Success"
      }
       Catch
      {
       Write-EntryToResultsFile -strUserid $strUserid -Message
$_.Exception.Message -Action "Clear homedirectory path" -
Result "Error"
       Continue
      }

    Try
      {
                $strUserDN = (Get-ADUser -Identity
$strUserid).distinguishedName
        Set-ADUser -Identity $strUserDN -Clear homeDrive -
WhatIf:(-not($ProductionRun)) -Confirm:$false
       Write-EntryToResultsFile -strUserid $strUserid -Action
```

```
"Clear homedrive" -Result "Success"
      }
       Catch
      {
       Write-EntryToResultsFile -strUserid $strUserid -Message
$_.Exception.Message -Action "Clear homedrive" -Result "Error"
       Continue
      }
}

Function Move-ADUserToOtherOU
{

<#
.NOTES
================================================================
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       06-September-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Move-ADUserToOtherOU
================================================================
================================================================
================
.SYNOPSIS

This function moves the user to another OU.

#>

  param
   (
    [string] $strUserid,
    [string] $strDestinationOU
   )
   Try
     {
      if($strDestinationOU.Length -gt 0)
      {
```

```
                $strUserDN = (Get-ADUser -Identity
$strUserid).distinguishedName
         Move-ADObject -Identity $strUserDN -TargetPath
$strDestinationOU -WhatIf:(-not($ProductionRun)) -
Confirm:$false
       Write-EntryToResultsFile -strUserid $strUserid -Action
"Move AD User to OU $strDestinationOU." -Result "Success"
      }
     }
      Catch
     {
      Write-EntryToResultsFile -strUserid $strUserid -Message
$_.Exception.Message -Action "Move AD User to OU
$strDestinationOU." -Result "Error"
      Continue
     }
}

Function Add-ADMemberToGroup
{

<#
.NOTES
================================================================
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       06-September-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Add-ADMemberToGroup
================================================================
================================================================
================
.SYNOPSIS

This function adds a user to an AD group.

#>

  param
```

```
   (
    [string] $strUserid,
    [string] $strADGroupName
   )
   Try
     {
      if($strADGroupName.Length -gt 0)
      {
                $strUserDN  =  (Get-ADUser  -Identity
$strUserid).distinguishedName
        Add-ADGroupMember -Identity $strADGroupName -Members
$strUserDN -WhatIf:(-not($ProductionRun)) -Confirm:$false
       Write-EntryToResultsFile -strUserid $strUserid -Action
"Add AD group $strADGroupName to user $strUserid." -Result
"Success"
       if($Global:strForRollBackGroupsToRemove.Length -eq 0)
       {
        $Global:strForRollBackGroupsToRemove = $strADGroupName
       }
       else
       {
              $Global:strForRollBackGroupsToRemove  =
$Global:strForRollBackGroupsToRemove + "," + $strADGroupName
       }
      }
     }
      Catch
     {
      Write-EntryToResultsFile -strUserid $strUserid -Message
$_.Exception.Message -Action "Add AD group $strADGroupName to
user $strUserid." -Result "Error"
      Continue
     }
}

Function Remove-ADMemberFromGroup
{

<#
.NOTES
============================================================
```

```
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       06-September-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Remove-ADMemberFromGroup
================================================================
================================================================
================
.SYNOPSIS

This function removes a user from an AD group.

#>

  param
   (
    [string] $strUserid,
    [string] $strADGroupName
   )
   Try
     {
      if($strADGroupName.Length -gt 0)
      {
                $strUserDN  =  (Get-ADUser  -Identity
$strUserid).distinguishedName
       Remove-ADGroupMember -Identity $strADGroupName -Members
$strUserDN -WhatIf:(-not($ProductionRun)) -Confirm:$false
       Write-EntryToResultsFile -strUserid $strUserid -Action
"Remove AD group $strADGroupName from user $strUserid." -
Result "Success"
       if($Global:strForRollBackGroupsToAdd.Length -eq 0)
       {
        $Global:strForRollBackGroupsToAdd = $strADGroupName
       }
       else
       {
                 $Global:strForRollBackGroupsToAdd  =
$Global:strForRollBackGroupsToAdd + "," + $strADGroupName
       }
```

```
      }
     }
      Catch
     {
      Write-EntryToResultsFile -strUserid $strUserid -Message
$_.Exception.Message -Action "Remove AD group $strADGroupName
from user $strUserid." -Result "Error"
      Continue
     }
}

Function Remove-UserFromMultipleGroups
{

<#
.NOTES
===========================================================
===========================================================
=================
Created with:     Windows PowerShell ISE
Created on:       12-October-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Remove-UserFromMultipleGroups
===========================================================
===========================================================
=================
.SYNOPSIS

Delete multiple groups from the users' account. The group
names are in an array.

#>

  param
   (
    [string] $arrGroupsToSearchFor,
    [string] $strUserid
   )

      $arrGroups = @(Get-ADUser $strUserid -Properties
```

```
MemberOf).MemberOf
    forEach($objGroup in $arrGroups)
    {
    $strGroup=(Get-ADGroup $objGroup).Name
    forEach($objGroupMustContain in $arrGroupsToSearchFor)
     {
                                $strGroupMustContain     =
$objGroupMustContain.ToString().ToLower()
      if($strGroup.ToLower().IndexOf($strGroupMustContain) -eq
0)
       {
          Remove-ADMemberFromGroup -strUserid $strUserid -
strADGroupName $strGroup
       }
     }
    }


}

Function Migrate-ADUserToNewGroup
{

<#
.NOTES
================================================================
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       11-October-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Migrate-ADUserToNewGroup
================================================================
================================================================
================
.SYNOPSIS

Perform the translating from the old group name to the new
group name.
```

```
#>

  param
   (
    [string] $strUserid,
    [string] $strADGroupName
   )
   $bolOldGroupNameHasBeenFound = $False
       ForEach($objGroupNameForAutomaticMigration  in
$arrGroupNamesForAutomaticMigration)
    {
     Try
     {
                                          $strOldGroup      =
$objGroupNameForAutomaticMigration.OldGroup
                                          $strNewGroup      =
$objGroupNameForAutomaticMigration.NewGroup
     }
      Catch
     {
       Write-Host "There is something wrong with the CSV
filename  $FileForAutomaticMigration  while  processing
$strUserid."
      Exit 1
     }
     if ($strADGroupName -eq $strOldGroup)
     {
      $bolOldGroupNameHasBeenFound = $true
      if ($strNewGroup.Length -gt 0)
       {
           Add-ADMemberToGroup -strUserid $strUserid -
strADGroupName $strNewGroup
        if ($RemoveOldADGroups)
         {
           Remove-ADMemberFromGroup -strUserid $strUserid -
strADGroupName $strOldGroup
         }
       }
        Else
       {
        Write-EntryToResultsFile -strUserid $strUserid -Action
```

```
"Migrate users" -Message "No new groupname specified for $strADGroupName in $FileForAutomaticMigration." -Result "Error"
       }
     }

    }
    if(-not($bolOldGroupNameHasBeenFound))
    {
      Write-EntryToResultsFile -strUserid $strUserid -Action "Migrate users" -Message "The group $strADGroupName has not been found in $FileForAutomaticMigration." -Result "Error"
    }
  }

Function Find-InArry
{
<#
.NOTES
================================================================================================================
Created with:     Windows PowerShell ISE
Created on:       17-October-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Find-InArry
================================================================================================================
.SYNOPSIS

This function checks two arrays for content. I had to write my own function as the build-in functions like Compare-Object or $arrOnlyIndirectMembers | Where {$arrDirectAndIndirectMembers -notcontains $arrDirectMembers} did not did their work properly.

The swith 'NotContains' can be $True or $False. If not specified on the command line, then it is $False.
```

```
#>


param
(
 [string[]] $SearchIn,
 [string[]] $LookFor,
 [switch] $NotContains
)

$Contains = -not $NotContains

$tmpArray = @()
ForEach ($objSearchIn in $SearchIn)
 {
  $bolFound = $false
  ForEach ($objLookFor in $LookFor)
  {
   if($objSearchIn -eq $objLookFor)
    {
     $bolFound = $True
     if ($bolFound -and $Contains)
     {
      $tmpArray+=$objLookFor
     }
    }
  }
  if (-not($bolFound) -and -not $Contains)
  {
   $tmpArray+=$objSearchIn
  }
 }
 Return $tmpArray
}

Function Show-IndirectGroupsInResultsFile
{

<#
.NOTES
============================================================
```

```
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       18-October-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Show-IndirectGroupsInResultsFile
================================================================
================================================================
================
.SYNOPSIS

This function shows only the indirect group membership in the
results file.

#>

  param
   (
    [string] $strUserid
   )

    $dn                                    = (Get-ADUser
$strUserid).DistinguishedName
   $arrOnlyDirectMembers        = @()
   $arrDirectAndIndirectMembers = @()
   $arrOnlyIndirectMembers      = @()
    $arrDirectAndIndirectMembers = Get-ADGroup   -LDAPFilter
("(member:1.2.840.113556.1.4.1941:={0})" -f $dn)    | Select-
Object sAMAccountName | Sort-Object sAMAccountName
       $arrOnlyDirectMembers                    =   Get-
ADPrincipalGroupMembership                          $dn
| Select-Object sAMAccountName | Sort-Object sAMAccountName
    $arrOnlyIndirectMembers       = Find-InArry -SearchIn
$arrDirectAndIndirectMembers   -NotContains   -LookFor
$arrOnlyDirectMembers

   $arrOnlyIndirectMembers | ForEach($_) {
    $tmpValue = $_
    $tmpValue = $TmpValue -Replace("@{sAMAccountName=","")
    $tmpValue = $TmpValue -Replace("}","")
```

```
     Write-EntryToResultsFile -strUserid $strUserid -Result
"Information" -Action "Indirect group member" -Message
$TmpValue
   }

}

Function Show-DirectGroupsInResultsFile
{

<#
.NOTES
================================================================
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       15-November-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Show-DirectGroupsInResultsFile
================================================================
================================================================
================
.SYNOPSIS

This function shows only the direct group membership in the
results file.

#>

  param
   (
    [string] $strUserid
   )

   $arrOnlyDirectMembers        = @()
       $arrOnlyDirectMembers                    =  Get-
ADPrincipalGroupMembership              ((Get-ADUser
$strUserid).DistinguishedName)  | Select-Object sAMAccountName
| Sort-Object sAMAccountName
   $arrOnlyDirectMembers | ForEach($_) {
```

```
    $tmpValue = $_.sAMAccountName
     Write-EntryToResultsFile -strUserid $strUserid -Result
"Information" -Action "Direct group member" -Message $TmpValue
   }

}

Function Create-FileWithUseridsInCSVFormat
{

<#
.NOTES
================================================================
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       18-October-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Create-FileWithUseridsInCSVFormat
================================================================
================================================================
================
.SYNOPSIS

This function cretes the file with all the userids to migrate,
based on OU.

#>
  Param
    (
     [string] $srcOU ="",
     [string] $dstOU= "",
     [string] $Windows10VDIGroup= ""
    )

     $strBaseFileName = "UseridsToMigrate (" + (Get-
Date).ToString('G') + ").csv"
   $strBaseFileName = $strBaseFileName -replace ":","-"
   $strBaseFileName = $strBaseFileName -replace "/","-"
   $strFileName     = $strCurrentPath + "\" + $strBaseFileName
```

```
   if($strFileName.Length -gt 260)
   {
    $valLength       = ($strCurrentPath.Length)-4
      $strBaseFileName = ($strBaseFileName.Substring(0,260-
$valLength)) + ".csv"
       $strFileName        =  $strCurrentPath  +  "\"  +
$strBaseFileName
   }

   $arrTableWithUserids = @()
    $arrUserids           = (Get-ADUser -Filter {Enabled -eq
"True"} -SearchScope Subtree -SearchBase $srcOU)
   $valCounter          = 1

   $Users = ForEach ($objUserid in $arrUserids)
   {
     $strUserDetails = Get-ADUser $objUserid -Properties
Name,sAMAccountName,department,cn, DistinguishedName, mail
    Write-Progress -Activity "Create CSV File with the userids
for    migration"    -Status    "Processing    user
$($strUserDetails.cn)."  -PercentComplete  ($valCounter  /
$arrUserids.Count * 100)
    $UserRecord                    = [ordered] @{"Userid" =
"";"Full  name"=  "";"Mail"="";"Department"=  "";"CurrentOU"=
"";"NewOU"="";"VDIGroup"=          "";"GroupsToAdd"=
"";"GroupsToRemove"= ""}
         $UserRecord."Userid"                          =
$strUserDetails.sAMAccountName
    $UserRecord."Full name"       = $strUserDetails.cn
    $UserRecord."Mail"            = $strUserDetails.mail
    $UserRecord."Department"      = $strUserDetails.department
        $UserRecord."CurrentOU"              =  "OU="+
($strUserDetails.DistinguishedName -split "=",3)[-1]
    $UserRecord."NewOU"           = $dstOU
    $UserRecord."VDIGroup"        = $Windows10VDIGroup
    $UserRecord."GroupsToAdd"     = ""
    $UserRecord."GroupsToRemove"  = ""
    $objRecordWithUserids          = New-Object PSObject -
Property $UserRecord
    $arrTableWithUserids        += $objRecordWithUserids
    $valCounter++
```

```
   }

   If($arrTableWithUserids.Count -gt 0)
   {
      $arrTableWithUserids | Export-Csv $strFileName -
NoTypeInformation
   }
    Else
   {
     Write-Host "Something went wrong while writing the file
$strFileName. Maybe nothing to report..."
   }
}

Function Add-TrailingBackSlash
{

<#
.NOTES
===========================================================
===========================================================
================
Created with:     Windows PowerShell ISE
Created on:       06-September-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Add-TrailingBackSlash
===========================================================
===========================================================
================
.SYNOPSIS

This function adds a trailing backslash to a string
#>

param
(
 [string] $AddBackslashTo =""
)

if($AddBackslashTo.Length -gt 0)
```

```
 {
  if(($AddBackslashTo.SubString($AddBackslashTo.Length-1,1)) -
ne "\")
  {
   $AddBackslashTo = $AddBackslashTo + "\"
  }
 }
Else
 {
  $AddBackslashTo = "\"
 }

 Return $AddBackslashTo

}

Function Copy-UserProfileFiles
{
<#
.NOTES
================================================================
================================================================
================
Created with:     Windows PowerShell ISE
Created on:       06-September-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Copy-UserProfileFiles
Source:
https://techblog.dorogin.com/powershell-how-to-recursively-cop
y-a-folder-structure-excluding-some-child-folders-and-files-
a1de7e70f1b
================================================================
================================================================
================
.SYNOPSIS

This function really copies the files.
#>

param
```

```
 (
  [string] $OldFilesDirectory,
  [string] $NewFilesDirectory
 )

 $arrExclude       = @("desktop.ini","$RECYCLE.BIN","Google
Chrome.lnk","thumbs.db")
 $arrExcludeMatch = @(".pst")
 New-Item -Path $NewFilesDirectory -ItemType Directory -Force
-WhatIf:(-not($ProductionRun)) | Out-Null
  Try
   {
    Get-ChildItem -Path $OldFilesDirectory -Recurse -Exclude
$arrExclude |
         where  {  $arrExcludeMatch  -eq  $null  -or
$_.FullName.Replace($OldFilesDirectory,  "")  -notmatch
$arrExcludeMatch } |
    Copy-Item -Destination {
        if ($_.PSIsContainer)
        {
                        Join-Path  $NewFilesDirectory
$_.Parent.FullName.Substring($OldFilesDirectory.length)
        }
         else
        {
                        Join-Path  $NewFilesDirectory
$_.FullName.Substring($OldFilesDirectory.length)
        }
         }  -Force  -Exclude  $arrExclude  -WhatIf:(-
not($ProductionRun))  -Confirm:$false  -Erroraction
SilentlyContinue
     Write-EntryToResultsFile -strUserid $Userid -Result
"Success"  -Action  "Copy  from  $OldFilesDirectory  to
$NewFilesDirectory."
   }
    Catch
   {
     Write-EntryToResultsFile -strUserid $Userid -Result
"Error"  -Action  "Copy  from  $OldFilesDirectory  to
$NewFilesDirectory." -Message $_.Exception.Message
    Continue
```

```
   }

}

Function Copy-UserProfile
{
<#
.NOTES
===========================================================================
===========================================================================
================
Created with:     Windows PowerShell ISE
Created on:       06-September-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Copy-UserProfile
===========================================================================
===========================================================================
================
.SYNOPSIS

This function copies the favorites and desktop from the old
profile to the new profile

#>

param
 (
  [string] $Userid="",
  [string] $OldProfilePath="",
  [string] $NewProfilePath=""
 )

 # Perform some checking
 $bolErrorsFound = $False

  $OldProfilePath = (Add-TrailingBackSlash -AddBackslashTo
$OldProfilePath) + $Userid
  $NewProfilePath = (Add-TrailingBackSlash -AddBackslashTo
$NewProfilePath) + $Userid
```

```
 if(-not(test-path($OldProfilePath)))
 {
  $bolErrorsFound = $True
  Write-EntryToResultsFile -strUserid $Userid -Result "Error"
-Action "Check directory" -Message "The directory
$OldProfilePath does not exists."
 }

 if(-not(test-path($NewProfilePath)))
 {
  $bolErrorsFound = $True
  Write-EntryToResultsFile -strUserid $Userid -Result "Error"
-Action "Check directory" -Message "The directory
$NewProfilePath does not exists."
 }

 if(test-path($NewProfilePath + "\Desktop\Oude desktop Windows
7"))
 {
  $bolErrorsFound = $True
  Write-EntryToResultsFile -strUserid $Userid -Result "Error"
-Action "Check directory" -Message "The directory
$($NewProfilePath + "\Desktop\Oude desktop Windows 7") exists.
That means that the profile has already been copied."
 }

 If($bolErrorsFound)
 {
  Return
 }

 # Copy the Favorites and desktop

   Copy-UserProfileFiles -OldFilesDirectory ($strOldFolder =
$OldProfilePath   +"\Favorites")   -NewFilesDirectory
($strNewFolder = $NewProfilePath +"\Favorites")
   Copy-UserProfileFiles -OldFilesDirectory ($strOldFolder =
$OldProfilePath  +"\Desktop")      -NewFilesDirectory
($strNewFolder = $NewProfilePath +"\Desktop\Oude desktop
Windows 7")
```

```
}

Function Add-RollBackRecord
{

<#
.NOTES
===========================================================
===========================================================
================
Created with:     Windows PowerShell ISE
Created on:       06-September-2018
Created by:       Willem-Jan Vroom
Organization:
Functionname:     Add-RollBackRecord
===========================================================
===========================================================
================
.SYNOPSIS

This function adds an item to the rollback file

#>

  param
   (
    [string] $Userid,
    [string] $NewOU,
    [string] $VDIGroup,
    [string] $GroupsToAdd,
    [string] $GroupsToRemove
   )
  $RollBackRecord                        = [ordered] @{"Userid"
=     "";"NewOU"="";"VDIGroup"=     "";"GroupsToAdd"=
"";"GroupsToRemove"= ""}
  $RollBackRecord."Userid"              = $Userid
  $RollBackRecord."NewOU"               = $NewOU
  $RollBackRecord."VDIGroup"            = $VDIGroup
  $RollBackRecord."GroupsToAdd"         = $GroupsToAdd
  $RollBackRecord."GroupsToRemove"      = $GroupsToRemove
  $objRollBackRecord                     = New-Object PSObject
```

```
-Property $RollBackRecord
  $Global:arrTableWithRollBackRecords  += $objRollBackRecord
  }

#
================================================================
================================================================
================
# End function block
#
================================================================
================================================================
================

#
================================================================
================================================================
================
# Declares the variables.
# Modify $strPrefixOldGroupName for your own enfironment.
#
================================================================
================================================================
================

  $valCounter                         = 1
  $Global:arrTable                    = @()
  $Global:arrTableWithRollBackRecords = @()
  $strCurrentPath                        = Split-Path -parent
$MyInvocation.MyCommand.Definition
    $strCurrentFile                                    =
$MyInvocation.MyCommand.Name
  $strPrefixOldGroupName              = "gg_appl_"
  $strPrefixNewGroupName              = "Appl_"
         $arrGroupMustContainForFullCleanUp          =
@($strPrefixOldGroupName,"WM-Users")

#
================================================================
================================================================
================
```

```
# check the length of the current directory. Stop the script if more that 248 characters. Otherwise the results cannot be written to the given
# folder.
#
================================================================================
================================================================================

  If($strCurrentPath.Length -gt 248)
  {
    Write-Host "The current directory $strCurrentPath has a length of more than 248 characters. The script will end with exit code 999 now."
   Exit 999
  }

#
================================================================================
================================================================================
# Check if the parameter CreateFileWithUseridsInCSVFormat is used. In that case, create the file and stop.
#
================================================================================
================================================================================

  if($CreateFileWithUseridsInCSVFormat)
  {
    Create-FileWithUseridsInCSVFormat -srcOU $srcOU -dstOU $dstOU -Windows10VDIGroup $Windows10VDIGroup
   Exit 0
  }
#
================================================================================
================================================================================
# Define the CSV Import File.
#
================================================================================
```

```
================================================================
================

  if($FileWithUseridsInCSVFormat.Length -eq 0)
   {
    $strCSVFileName = $strCurrentFile -Replace ".ps1",".csv"
   }
    else
   {
    if($FileWithUseridsInCSVFormat.ToLower().IndexOf(".csv") -
eq -1)
    {
     $FileWithUseridsInCSVFormat = $FileWithUseridsInCSVFormat
+ ".csv"
    }
    $strCSVFileName = $FileWithUseridsInCSVFormat
   }

#
================================================================
================================================================
================
# Check if the string $strCSVFileName is a path. In that case,
nothing has to be done.
# In case it is not a path, then the current location should
be added.
#
================================================================
================================================================
================

  if (-not(Split-Path($strCSVFileName)))
  {
   $strCSVFileName = $strCurrentPath + "\" + $strCSVFileName
  }

#
================================================================
================================================================
================
# In case of an automated migration check if the string
```

```
$FileForAutomaticMigration is a path. In that case, nothing
has to be done.
# In case it is not a path, then the current location should
be added.
# If the file exists then read all the content, otherwise
quit.
#
================================================================
================================================================
================

  if($FileForAutomaticMigration.Length -gt 0)
   {
    if (-not(Split-Path($FileForAutomaticMigration)))
    {
      $FileForAutomaticMigration = $strCurrentPath + "\" +
$FileForAutomaticMigration
    }
    if(Test-Path($FileForAutomaticMigration))
     {
        $arrGroupNamesForAutomaticMigration = @(Import-Csv
$FileForAutomaticMigration)
      if ($arrGroupNamesForAutomaticMigration.Count -eq 0)
      {
      Write-Host "The file $FileForAutomaticMigration seems to
be empty..."
      Exit 1
      }
     }
      else
     {
      Write-Host "The file $FileForAutomaticMigration does not
exists. Thus quitting."
      Exit 1
     }
   }

#
================================================================
================================================================
================
```

```
# Define the log file. This log file contains all the results.
#
================================================================
================================================================
================

  $strLastPartOfFileName = " (" + (Get-Date).ToString('G') +
").csv"
   $strLastPartOfFileName = $strLastPartOfFileName -replace
":","-"
   $strLastPartOfFileName = $strLastPartOfFileName -replace
"/","-"
  If(-not($ProductionRun))
  {
    $strLastPartOfFileName = " (RUNNING IN TEST MODE)" +
$strLastPartOfFileName
  }

  $strCSVLogFileSucces   = $strCSVFileName -Replace ".csv",
$strLastPartOfFileName

  $strPathName          = (Split-Path $strCSVLogFileSucces) +
"\"
       $strFileName                                        =
$strCSVLogFileSucces.Substring($strPathName.Length,($strCSVLog
FileSucces.Length - $strPathName.Length))

  $strCSVLogFileSucces   = $strPathName + $LogFilePrefix +
$strFileName
   $strCSVRollBackFile     = $strPathName + "RBF_" +
$strFileName

  $valFileLengthLogFile = $strCSVLogFileSucces.Length

  if($valFileLengthLogFile -gt 260)
  {
   Write-Host "The file name $strCSVLogFileSucces is too long.
The maximum file length is 260 characters. This one is
$valFileLengthLogFile long.`n No log file is generated, and
therefore, this application will quit."
   Exit 2
```

```
  }

#
================================================================
================================================================
================
# Read the CSV file.
#
================================================================
================================================================
================
  If(Test-Path $strCSVFileName)
  {
   $arrUserids = @(Import-Csv $strCSVFileName)
  }
   Else
  {
    Write-Host "The import file $strCSVFileName does not
exists."
   Exit 1
  }
#
================================================================
================================================================
================
# Find all the arguments and puthem in the log file
# Source: https://ss64.com/ps/psboundparameters.html
#
================================================================
================================================================
================

  Write-EntryToResultsFile -strUserid "" -Result "Information"
-Action "Used script" -Message $strCurrentPath + "\" +
$strCurrentFile

  foreach($boundparam in $PSBoundParameters.GetEnumerator())
  {
     Write-EntryToResultsFile -strUserid "" -Result
"Information" -Action "Key: $($boundparam.Key)" -Message
"Value: $($boundparam.Value)"
```

```
  }
#
================================================================
================================================================
================
# Process the users in the CSV file.
#
================================================================
================================================================
================

  Clear-Host

  $strActivity                          = "Modifying users in
Active Directory."

  If(-not($ProductionRun))
  {
  $strActivity = $strActivity + " (RUNNING IN TEST MODE)"
  }

  ForEach($objUser in $arrUserids)
  {
   $strUserid = $objUser.Userid
    Write-Progress -Activity $strActivity -Status "Processing
user  $strUserid"  -PercentComplete  ($valCounter  /
$arrUserids.Count * 100)

   if($CreateRollBackFile)
   {
    $Global:strForRollBackGroupsToAdd    = ""
    $Global:strForRollBackGroupsToRemove = ""
     $strUserDetails                          = Get-ADUser
$strUserid -Properties cn, DistinguishedName
     $strCurrentUserOU                         = "OU=" +
($strUserDetails.DistinguishedName -split "=",3)[-1]
   }

   Try
   {
    $strNewOU          = $objUser.NewOU
```

```
    $strVDIGroup       = $objUser.VDIGroup
    $arrGroupsToAdd    = $objUser.GroupsToAdd.Split(",")
    $arrGroupsToRemove = $objUser.GroupsToRemove.Split(",")
   }
    Catch
   {
     Write-EntryToResultsFile -strUserid $strUserid -Result
"Error" -Message "There is something wrong with the CSV
filename $strCSVFileName while processing $strUserid."
    Export-ResultsLogFileToCSV
    Exit 1
   }
   # Copy Profile
   if($copyProfile)
   {
      Copy-UserProfile -Userid $strUserid -OldProfilePath
$ProfilePathFrom -NewProfilePath $ProfilePathTo
   }

   if(-not $copyProfile)
   {

    # Clear the profile path
    If($ClearProfilePath)
     {
         Remove-ProfilePathFromUserProfileInAD -strUserid
$strUserid
     }
    # Clear the homedrive and home directory
    If($ClearHomeFolder)
     {
        Remove-HomeFolderPathFromUserProfileInAD -strUserid
$strUserid
     }
    # Move the user to another OU:
       Move-ADUserToOtherOU -strUserid $strUserid -
strDestinationOU $strNewOU
    # Add the user to the new VDI group:
     Add-ADMemberToGroup -strUserid $strUserid -strADGroupName
$strVDIGroup
    # Add the user to various groups:
```

```
    ForEach($objGroupToAdd in $arrGroupsToAdd)
    {
     Add-ADMemberToGroup -strUserid $strUserid -strADGroupName
$objGroupToAdd
    }
    # Remove the user to various groups:
    ForEach($objGroupToRemove in $arrGroupsToRemove)
    {
       Remove-ADMemberFromGroup -strUserid $strUserid -
strADGroupName $objGroupToRemove
    }
   }

   # Automatic migration

    if($FileForAutomaticMigration.Length -gt 0 -and -not
$copyProfile)
   {
      $arrGroups = @(Get-ADUser $strUserid -Properties
MemberOf).MemberOf
    forEach($objGroup in $arrGroups)
    {
     $strGroupName = (Get-ADGroup $objGroup).Name
                                                          if
($strGroupName.ToLower().IndexOf($strPrefixOldGroupName)  -eq
0)
     {
        Migrate-ADUserToNewGroup -strUserid $strUserid -
strADGroupName $strGroupName
     }
    }
   }

   # Full Clean Up
   if($FullCleanUp -and -not $copyProfile)
   {
     Remove-UserFromMultipleGroups  -arrGroupsToSearchFor
$arrGroupMustContainForFullCleanUp -strUserid $strUserid
   }

   # Remove old Citrix groups
```

```
   if($RemoveOldCitrixGroups -and -not $copyProfile)
   {
                                    $arrCitrixOU          =
@("OU=Citrix,OU=OU1,DC=testdomain,DC=local,DC=lan")

   ForEach($objCitrixOU in $arrCitrixOU)
     {
       $arrWithCitrixGroups = get-adgroup -filter "*" -
SearchBase $objCitrixOU | Where-Object Name -Match "Citrix
VDI*"
     ForEach ($objWithCitrixGroups in $arrWithCitrixGroups)
      {
        Remove-UserFromMultipleGroups -arrGroupsToSearchFor
$objWithCitrixGroups.Name -strUserid $strUserid
      }
     }

       Remove-UserFromMultipleGroups -arrGroupsToSearchFor
@("CitrixTestUsers","CitrixProductionUsers")    -strUserid
$strUserid
   }

   # Perform the Direct Group Inventory
   if($RunDirectGroupInventory -and -not $ProductionRun -and -
not $copyProfile)
   {
    Show-DirectGroupsInResultsFile -strUserid $strUserid
   }

   # Perform the Indirect Group Inventory
   if($RunIndirectGroupInventory -and -not $ProductionRun -and
-not $copyProfile)
   {
    Show-IndirectGroupsInResultsFile -strUserid $strUserid
   }

   if($CreateRollBackFile)
   {
       Add-RollBackRecord -Userid $strUserid -NewOU
$strCurrentUserOU    -VDIGroup    ""    -GroupsToAdd
$Global:strForRollBackGroupsToAdd      -GroupsToRemove
```

```
$Global:strForRollBackGroupsToRemove
   }

   $valCounter++
  }
   if($ProductionRun -and ($RunDirectGroupInventory -or
$RunIndirectGroupInventory) -and -not $copyProfile)
  {
   $valCounter = 1
   $maxCounter = 30
         For($valCounter = 1; $valCounter -le
$maxCounter;$valCounter++)
    {
     Write-Progress -Activity "Processing changes in Active
Directory" -Status "Waiting $valCounter of $maxCounter
seconds." -PercentComplete ($valCounter / $maxCounter * 100)
     Sleep 1
    }

                                                              #
================================================================
================================================================
================
   # Inventory the indirect user groups the user belongs to.
   # This can only be done after updating Active Directory,
otherwise incorrect results are shown.
                                                              #
================================================================
================================================================
================

     $strActivity                          = "Inventory the
indirect and / or direct groups the user belongs to."

     If(-not($ProductionRun))
      {
       $strActivity = $strActivity + " (RUNNING IN TEST MODE)"
      }

     $valCounter = 1
     ForEach($objUser in $arrUserids)
```

```
      {
       $strUserid = $objUser.Userid
          Write-Progress -Activity $strActivity -Status
"Processing user $strUserid" -PercentComplete ($valCounter /
$arrUserids.Count * 100)
       if ($RunDirectGroupInventory)
        {
         Show-DirectGroupsInResultsFile -strUserid $strUserid
        }
       if ($RunIndirectGroupInventory)
        {
            Show-IndirectGroupsInResultsFile -strUserid
$strUserid
        }
       $valCounter++
      }
  }

#
================================================================
================================================================
================
# Write the results to the csv file.
#
================================================================
================================================================
================
  Export-ResultsLogFileToCSV
#
================================================================
================================================================
================
# Write the Rollback file to a csv file.
#
================================================================
================================================================
================

  if($CreateRollBackFile)
  {
   Export-RollBackFileToCSV
```

```
}
```

Current version: Migrate users (v10)

Any feedback to improve this script is appreciated. You can download the previous versions here:

1. Migrate users (v01)
2. Migrate users (v02)
3. Migrate users (v03)