Using PowerShell to Update an AD User from a CSV file

lazyadmin.nl/powershell/using-powershell-to-update-an-ad-user-from-a-csv-file

January 7, 2019

I needed to update a bunch of AD Users with their new job titles. A new year has started and some made promotions and got a new job title. So I received an Excel file with their full name and their new title from HR. Because this is happening every year, is it time to create a PowerShell script for it.

As a bonus, we are also going to send the user an email to notify them that their job title is changed. So they don't have to contact IT for it anymore.

Importing the CSV File in PowerShell

To update the AD User we are going to use a CSV file. This allows us to use the **Import-CSV** cmdlet in PowerShell. I have used the following Excel table that I have saved a CSV.

name	jobtitle
John Doe	senior accountmanager
Jane Doe	accountmanager
Salina Scott	engineer
Jasper Rempel	junior engineer

With the parameter `csvPath`, we can specify the location of the CSV file that we want to import.

Set-UserPromotions.ps1 -csvPath c:\temp\promotions.csv
Now to import the CSV file into PowerShell we use the following command later on:

$promotions = Import-Csv -Delimiter ";" -Path $csvPath

> Tip
>
> *By default, the **Delimiter** is a comma (,), but you can change to by using the **-Delimiter** parameter. As you can see I am using the $rootPath to reference to the script location. You can also fill in the absolute path to the file here.*

The result of **$promotions** is:

name jobtitle

John Doe senior accountmanager
Jane Doe accountmanager

Salina Scott Engineer
Jasper Rempel junior engineer

Finding the AD user to update

The next step is to find the user in the Active Directory. We need to find the users based on their full names. We are going to use the `Get-ADUser` cmdlet for this and filter the results on the display name.

We don't have one user, but a whole list. So we are going to use a foreach loop to walk through the list of users.

```
foreach($user in $promotions){
# Find user
$ADUser = Get-ADUser -Filter "displayname -eq '$($user.name)'"
}
```

So we are trying to get each user in the table promotions. Now a good practice is to implement a catch in case the user doesn't exist. Maybe HR made a typo which can result in the user not being found.

Also, we need the user's email address later on to send him the notification. Add the property mail to the `Get-ADUser` cmd.

```
foreach($user in $promotions){
#find user
$ADUser = Get-ADUser -Filter "displayname -eq '$($user.user)'" -Properties mail
if ($ADUser){
# <update the user>
}else{
Write-Warning ("Failed to update " + $($user.user))
}
}
```

Updating the AD User

If the users exist in the Active Directory we can use the `Set-ADUser` to update an attribute. In this case the title attribute, but that can be any AD User-related attribute.

> Tip
>
> *Read more about the Set-ADUser cmdlet in this article.*

```
foreach($user in $promotions){
# Find user
$ADUser = Get-ADUser -Filter "displayname -eq '$($user.name)'" -Properties mail
if ($ADUser){
Set-ADUser -Identity $ADUser -Title $user.jobtitle
}else{
Write-Warning ("Failed to update " + $($user.name))
}
}
```

If you only want to update AD User from a CSV file then you are done. If you also want to send them an email to notify them about the change, then keep reading.

Sending the User an Email

I am using an email template to send the user an email. The template is an HTML file that you can download here on GitHub along with this script. Before we can the email we need to replace some placeholders with the user’s name and a new title. Also, we need an SMTP server for sending the email.

Setting the STMP details

Below is a simple array with the SMTP details, like the server and from address.

```
#SMPT Details to send the email
$smtp = @{
"address" = "stonegrovebank.mail.protection.outlook.com"
"from" = "itdept@stonegrovebank.com>"
"subject" = "Jobtitle updated."
}
```

Creating the Email body

I have a function that I use for creating the email body. Using functions allows me to reuse parts of code in different scripts. The first function will get the email template and replace the placeholders with the correct data.

```
Function Get-EmailTemplate {
<#
.SYNOPSIS
Get the eamil template which is located in the same location as the script
#>
PARAM(
[parameter(Mandatory=$true)]
$user,
[parameter(Mandatory=$true)]
$jobtitle
)
PROCESS
{
#Get the mailtemplate
$mailTemplate = (Get-Content ($rootPath + '\MailTemplate.html')) | ForEach-Object {
```

```
$_ -replace '{{user.jobtitle}}', $jobtitle`
-replace '{{user.firstname}}', $user.givenName
} | Out-String
return $mailTemplate
}
}
```

In the foreach loop, after the Set-ADuser cmd we add the following line to call the function and build the email body:

```
$emailBody = Get-EmailTemplate -user $ADUser -JobTitle $user.jobtitle
```

Sending the email

The last function is for sending the email. It takes the SMTP details from the SMTP array and uses the supplied information to send the email. I added a what-if variable in it so you can run a test before sending the actual email.

```
Function Send-Mail {
<#
.SYNOPSIS
Send the user a mail.
#>
PARAM(
[parameter(Mandatory=$true)]
$emailBody,
[parameter(Mandatory=$true)]
$user,
[parameter(Mandatory=$false)]
[bool]$whatIf
)
PROCESS
{
#Set encoding
$encoding = [System.Text.Encoding]::UTF8
Try
{
if ($whatIf -ne $true)
{
send-MailMessage -SmtpServer $smtp.address -To $user.mail -From $smtp.from -
Subject $smtp.subject -Encoding $encoding -Body $emailBody -BodyAsHtml
}
else
{
Write-host ("Send mail to -SmtpServer " + $smtp.address + " -To " + $user.mail + " -From
" + $smtp.from + " -Subject $smtp.subject")
}
}
Catch
{
Write-Error "Failed to send email to, $_"
}
}
}
```
We call this function with the following cmd that we add below the email body:

```
Send-Mail -user $ADUser -EmailBody $emailBody
```
So the complete foreach loop now looks like this:

```
foreach($user in $promotions){
#find user
$ADUser = Get-ADUser -Filter "displayname -eq '$($user.user)'" -Properties mail
if ($ADUser){
Set-ADUser -Identity $ADUser -Title $user.jobtitle -WhatIf
$emailBody = Get-EmailTemplate -user $ADUser -JobTitle $user.jobtitle
Send-Mail -user $ADUser -EmailBody $emailBody -whatIf $true
}else{
Write-Warning ("Failed to update " + $($user.user))
}
}
```
Notice that I added two **WhatIf** flags. One for the Set-ADUser and one for the Send-mail. This way you can run your script to test it before actually changing or sending anything.

Conclusion

I hope the code above helped you update your AD User from a CSV file with PowerShell. You can find the complete script and email template here in my GitHub repository.

If you have any questions just add a comment below.

Did you **Liked** this **Article**?
Get the latest articles like this **in your mailbox**
or share this article

I hate spam to, so you can unsubscribe at any time.