

```
#!/bin/bash
# Script Name: generate_users.sh
# Author: [Your Name]
# Date: [Current Date]
# Description: A Bash script that generates usernames and passwords based on user input. It includes options to get events, create new events, or delete/backup existing events.

# Function to get user input
get_user_input() {
    echo -e "\n--- User Input Menu ---"
    echo -e "1 = Get Events"
    echo -e "2 = New Event"
    echo -e "3 = Delete or Backup Event"
}

# Main script logic
main() {
    echo -e "\n--- Please Enter a Number ---"

    # Get user input
    read -p "Enter a number (1-3): " user_input

    # Process user input
    case $user_input in
        1)
            # Get Events
            echo -e "\n--- Get Events ---"
            # Set location for events
            events_dir="/var/log/events"
            # Set location for event users
            event_users_dir="/var/log/event-users"
            # Get child files
            find $events_dir -type f | while read file; do
                echo $file
            done
        ;;
        2)
            # New Event
            echo -e "\n--- New Event ---"
            event_name=$(read -p "Enter event name: ")
            event_start=$(read -p "Enter event start (dd.mm.yyyy hh:mm:ss): ")
            event_end=$(read -p "Enter event end (dd.mm.yyyy hh:mm:ss): ")
            input_file=$(read -p "Enter the path of user file: ")
            output_file=$(read -p "Where do you want to save generated usernames and passwords?: ")

            # Generate event data
            event_start=$(date -d "$event_start" +%s)
            event_end=$(date -d "$event_end" +%s)
            event_duration=$((event_end - event_start))

            # Create output file
            echo -e "Creating output file: $output_file"
            echo -e "Set location: $output_file"

            # Get content of input file
            users=$(cat $input_file)

            # Process each user
            for user in $users; do
                # Generate password
                temp_password=$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 12 | xargs -n1 sha256sum | cut -d' ' -f1 | tr -d '\n')
                secure_temp_password=$(echo -n $temp_password | openssl dgst -sha256 -hex | cut -d' ' -f2 | tr -d '\n')
                secure_temp_password=$(echo -n $secure_temp_password | openssl dgst -sha256 -hex | cut -d' ' -f2 | tr -d '\n')

                # Create user
                echo -e "New user: $user"
                echo -e "Name: $user"
                echo -e "Secure password: $secure_temp_password"
            done
        ;;
        3)
            # Delete or Backup Event
            echo -e "\n--- Delete or Backup Event ---"
            # Set location for events
            events_dir="/var/log/events"
            # Set location for event users
            event_users_dir="/var/log/event-users"
            # Get child files
            find $events_dir -type f | while read file; do
                echo $file
            done
        ;;
    esac
}

# Run the main function
main
```

Event Manager (M159)

Tim Bühler

ST13a

Inhaltsverzeichnis

Inhaltsverzeichnis.....	1
Einleitung	1
Dokumentation Powershell Script	2
Meta, Definitionen	2
Menü	2
Events anzeigen	2
Events erstellen.....	3
Events löschen und Backupen	3
Error	3
Anhang	4
250 Benutzer generieren.	4

Einleitung

Die Aufgabe war es, Ein Powershell Script für die Verwaltung von Usern an einem Event zu erstellen. Dieses habe ich Event Manager genannt, da es so konzipiert wurde, dass es universell, also nicht nur für dieses eine Event, einsetzbar ist. Das Tool ist also nicht auf die Messe Basel spezifiziert, sondern kann nach minimalen Anpassungen auf anderen Systemen (Andere Domäne) eingesetzt werden.

Dokumentation Powershell Script

Das Script verfügt über Inline Comments. Sie Beschreiben das wichtigste knapp. Hier wird es etwas ausführlicher dokumentiert.

Meta, Definitionen

<pre><# Meta Info Name: Event Manager Version: 0.1 Date: 2016.04.10 By: Tim Buehler Module : 159 School/Class: TBZ/St13a #></pre>	Meta Informationen. Darin werden Name, Version und Datum sowie Urheber definiert.
<pre>clear-host</pre>	Damit die Konsole vor dem Ausführen gesäubert wird.
<pre>Import-Module ActiveDirectory</pre>	Für das Administrieren einer AD wird ein Powershell AD Modul importiert. Mit diesem Modul können Active Directory Pfade wie normale Verzeichnisse angesprochen werden.
<pre>#Definitions \$Domain = 'schoollab' \$TLD = 'local'</pre>	Möchte man das Script in einer anderen Domäne verwenden, muss man jediglich \$Domain und \$TLD anders setzen.

Menü

<pre>Set-Location \$ADRootDirectory Create-OU -ou Event-Users Set-Location OU=Event-Users</pre>	Nun beginnt das eigentliche Script. Wir erstellen die OU Event-Users und wechseln in dieses Verzeichnis. In dieser OU werden dann die OUs für die Events angelegt.
<pre>Write-Host 'Welcome to the Event Manager' Write-Host '1 = Get Events' Write-Host '2 = New Event' Write-Host '3 = Delete or Backup Event' Write-Host 'x = Exit' \$action = Read-Host -Prompt 'Please Enter a Number'</pre>	Das Menü fragt den User nach der gewünschten Aktion. Über dieses und das folgende if statement ist eine DO UNTIL Schlafe gemacht, damit sich das Menü nach einer getätigten Aktion wiederholt. Bis „x“ eingegeben wird.

Events anzeigen

<pre>Set-Location \$AdRootDirectory Set-Location 'OU=Event-Users' Get-ChildItem Write-Host</pre>	Bei einer „1“ werden die Unterverzeichnisse der OU Event-Units angezeigt. Also die aktuellen Events, welche als OU da definiert sind.
--	---

Events erstellen

<code>\$EventName = Read-Host -Prompt 'Enter event name'</code>	Bei einer „2“ Wird ein neues Event erstellt. Erst werden Infos zum Event vom Benutzer verlangt. (Nur <u>ein</u> Bsp. in der Linken Box)
<code>Create-OU -ou \$EventName Set-Location OU=\$EventName</code>	Danach wird Das Event (OU) erstellt.
<code>[array]\$users = Get-Content \$InputFile</code>	Das Userfile in eine Variable abgespeichert
<code>foreach (\$item in \$users)</code>	Folgende Änderungen werden nun für jeden user (welcher vom Userfile importiert wurde) vorgenommen.
<code>\$TempPassword = GET-Temppassword -length 8 -sourcedata \$ascii</code>	Es wird ein Passwort generiert mit 8 Zeichen und dem ASCII Zeichensatz.
<code>\$SecureTempPassword = ConvertTo-SecureString \$TempPassword -AsPlainText -Force</code>	Das Passwort wird als SecureString in einer anderen Variable abgespeichert.
<code>New-ADUser ` -Name \$item ` -AccountPassword \$currentPassword ` -CannotChangePassword \$true ` -PasswordNeverExpires \$true ` -AccountExpirationDate \$EventEnd ` -ChangePasswordAtLogon \$false ` -Enabled \$true ` -KerberosEncryptionType AES256</code>	Danach wird ein Benutzer in der AD angelegt. Die Linksstehenden Parameter werden verwendet.
<code>[array]\$UserLoginData = \$item,\$temppassword \$UserLoginData Out-file \$OutputFiles\\$item.txt</code>	Danach wird für den User ein Textfile mit seinem Usernamen und Passwort generiert.

Events löschen und Backups

<code>\$DeleteEvent = Read-Host -Prompt 'Enter the OU you want to Delete' \$OutputFile = Read-Host -Prompt 'where do you want to Save you backup?'</code>	Bei einer „3“ wird eine OU gelöscht und die Namen der OU und User gespeichert. Dafür werden zuerst wieder ein paar Variablen des Benutzers gebraucht.
<code>\$users = get-aduser \$DeleteEvent \$users Out-File \$OutputFile</code>	Die User auslesen und als Output speichern.
<code>Remove-ADOrganizationalUnit \$DeleteEvent -Recursive</code>	Die OU mitsamt Unterobjekten löschen

Error

<code>Write-Host 'Ungültige Eingabe'</code>	Wird keine Option angegeben, wird Ungültige Eingabe in die Konsole geschrieben und durch die DO UNTIL Funktion wieder das Menü aufgerufen.
---	--

Anhang

250 Benutzer generieren.

Zum testen des Scripts brauchte ich 250 Users in einem File. Dieses Habe ich mit folgendem Script generiert.

#Creates 250 named user in ascii encoded text file

```
#definitions
$path = 'C:\Temp'
$file = '250users.txt'
$fullpath = $path + '\' + $file
[array]$users = $null
$x = 0 #reset var X

#counting and adding users to variable users
do
{
    $currentuser = "user$x"
    $users += $currentuser

    $x += 1
}
until ($x -eq 250)

#writes users to file
$users | out-file $fullpath -Encoding ascii
```