SCRIPTS

Neus Cháfer Sanjuán

Scripts del proyecto de fin de grado

2024-2025

1. INTRODUCCIÓN

Para garantizar una guía coherente y documentación completa para el proyecto de PowerShell, se realiza una recopilación exhaustiva de todos los scripts utilizados. Estos scripts están debidamente comentados y documentados para facilitar la comprensión. La documentación incluye descripciones detalladas de la funcionalidad de cada script.

Esta información sirve como una referencia asegurando consistencia en el desarrollo, mantenimiento y consulta de los scripts utilizados en el proyecto.

INDICE

2. Scripts para el Servidor Principal (Windows Server 2022)

Estos scripts están diseñados para ejecutarse en el servidor principal controlador de dominio. Aquí está la lista de los scripts en el orden en que deben ejecutarse:

1. **ConfiguraciónRed.ps1:** Configura la red del servidor.
2. **ConfiguraciónAD.ps1:** Realiza la configuración inicial del Active Directory.
3. **CrearOU.ps1:** Crea unidades organizativas para organizar objetos en el Active Directory.
4. **CrearGRUPO.ps1:** Crea grupos de seguridad para gestionar los permisos de acceso.
5. **CrearUSUARIOS.ps1:** Crea usuarios en el Active Directory.
6. **AñadirUserGroup.ps1**: Añade usuarios a grupos de seguridad.
7. **HabilitarCuenta.ps1:** Habilita cuentas de usuario en el dominio.
8. **InstalarGroupPolicy.ps1:** Instala políticas de grupo para gestionar la configuración del sistema.
9. **DirectivaBloqueoCuenta.ps1:** Establece directivas para bloquear cuentas después de varios intentos de inicio de sesión fallidos.
10. **PoliticaContraseña.ps1:** Configura políticas de contraseña para garantizar la seguridad de las cuentas.
11. **PoliticaFirewall.ps1:** Define políticas de firewall para proteger la red.
12. **DirectivaAuditoria.ps1:** Establece directivas de auditoría para realizar un seguimiento de los eventos del sistema.
13. **PerfilesMoviles.ps1:** Configura perfiles móviles para usuarios.
14. **InstalarSMB.ps1:** Instala el protocolo SMB para compartir archivos e impresoras.
15. **CrearCarpetas.ps1:** Crea carpetas compartidas en el servidor.
16. **PermisosCarpetas.ps1:** Configura permisos de acceso para las carpetas compartidas.

Estos scripts se han organizado cuidadosamente en este orden para garantizar una configuración coherente y segura del servidor. Cada script será explicado detalladamente para comprender su función y asegurar una implementación segura.

También tenemos un script adicional "**Asistente_CreaciónObjetos_AD.ps1**" que proporciona funciones interactivas para guiar al usuario a través del proceso de creación de objetos AD (Unidades organizativas, grupos y usuarios)

2.1. Configuración de Red del Servidor

El script **“ConfiguraciónRed.ps1”** configura una dirección IP estática, establece un servidor DNS y cambia el nombre del PC. Primero, asigna la dirección IP "**192.168.56.100**" a través de la interfaz **Ethernet**, luego configura el servidor DNS a "**8.8.8.8**", y finalmente cambia el nombre del PC a "**WinServer-AD**" con un reinicio del sistema.

```
# Configurar la dirección IP estática
New-NetIPAddress -InterfaceAlias "Ethernet" -IPAddress "192.168.56.100" -PrefixLength
24 -DefaultGateway "192.168.56.1"

# Configurar la dirección de servidor DNS
Set-DNSClientServerAddress -InterfaceAlias "Ethernet" -ServerAddresses "8.8.8.8"

# Cambiar el nombre del PC
Rename-Computer -NewName "WinServer-AD" -Restart
```

Comando	Explicación
New-NetIPAddress -InterfaceAlias “” -IPAddress “”	Establece la dirección IP, en la interfaz especificada.
-PrefixLength	Define el tamaño de la máscara de subred.
-DefaultGateway	Especifica la puerta de enlace.
Set-DNSClientServerAddress **-InterfaceAlias “”-ServerAddresses “”**	Establece la dirección IP del servidor DNS en la Interfaz especificada.
Rename-Computer -NewName “”	Establece el nuevo nombre del equipo.

2.2. Configuración de Active Directory

El script **“ConfiguraciónAD.ps1”** instala el Rol de Active Directory en el servidor, incluyendo las herramientas de administración. Luego, promueve el servidor a Controlador de Dominio, estableciendo un nuevo dominio llamado "**Dom.SST**" y configurando una contraseña de administrador en modo seguro.

```
# Instalación del Rol de Active Directory
Install-WindowsFeature -Name AD-Domain-Services -IncludeManagementTools

# Promoción del Servidor a Controlador de Dominio
Install-ADDSForest -DomainName "Dom.SST" -SafeModeAdministratorPassword (ConvertTo-SecureString "20012506N." -AsPlainText -Force)
```

Comando	Explicación
Install-WindowsFeature -Name AD-Domain-Services -IncludeManagementTools	Instala el Rol de Servicios de Dominio de Active Directory en el servidor.
-Name AD-Domain-Services	Especifica el nombre del rol que se instalará, que en este caso es “AD-Domain-Services”.
-IncludeManagementTools	Se deben instalar también las herramientas de administración relacionadas con el rol.
Install-ADDSForest	Inicio del proceso de promover un servidor a un Controlador de Dominio en un nuevo bosque de Active Directory.
-DomainName “”	Especifica el nombre del dominio que se creará.
-SafeModeAdministratorPassword “”	Establece la contraseña del administrador en modo seguro.
(ConvertTo-SecureString “” -AsPlainText -Force)	Covierte una cadena de texto en un objeto de tipo SecureString, que es un tipo de datos seguro diseñado para contener información confidencial, como contraseñas. **AsPlainText -Force ;** Esto significa que la cadena de texto se proporciona directamente tal como está, sin cifrado adicional y debe realizarse incluso si se detecta una advertencia de seguridad.

2.3. Crear Unidades Organizativas

El script **“CrearOU.ps1”** crea unidades organizativas (OUs) en un entorno de Active Directory. Comienza creando la OU principal llamada "**ou.SST**" y luego establece varias OUs secundarias dentro de ella para organizar distintos departamentos.

Para entender mejor cómo funciona el script, es importante comprender que cada OU se encuentra ubicada dentro de una estructura jerárquica en Active Directory. Por lo tanto, al crear una OU, debemos especificar la ruta completa donde se ubicará dentro del árbol de directorios de Active Directory.

Por ejemplo, para crear una OU llamada "**ou.Desarrollo**" dentro de la OU principal “**ou.SST**”, la ruta completa sería:
OU=ou.Desarrollo,OU=ou.SST,DC=dom,DC=SST
Ahora, explicando cada comando:
New-ADOrganizationalUnit -Name "ou.SST" -Path "DC=dom,DC=SST"
Este comando crea la OU principal llamada "**ou.SST**" en la ruta especificada "DC=**dom**,DC=**SST**".

La OU principal "ou.SST" se encuentra dentro del dominio "dom.SST" y contiene las siguientes OUs secundarias: "ou.Desarrollo", "ou.SoporteTecnico", "ou.VentasyMarketing", "ou.AdminyFinanzas" y "ou.AdminSistemas".

```
# Crear la OU principal ou.SST
New-ADOrganizationalUnit -Name "ou.SST" -Path "DC=dom,DC=SST"

# Crear las OUs secundarias dentro de ou.SST
New-ADOrganizationalUnit -Name "ou.Desarrollo" -Path "OU=ou.SST,DC=dom,DC=SST"
New-ADOrganizationalUnit -Name "ou.SoporteTecnico" -Path "OU=ou.SST,DC=dom,DC=SST"
New-ADOrganizationalUnit -Name "ou.VentasyMarketing" -Path "OU=ou.SST,DC=dom,DC=SST"
New-ADOrganizationalUnit -Name "ou.AdminyFinanzas" -Path "OU=ou.SST,DC=dom,DC=SST"
New-ADOrganizationalUnit -Name "ou.AdminSistemas" -Path "OU=ou.SST,DC=dom,DC=SST"
```

Comando	Explicación
New-ADOrganizationalUnit -Name “” **-Path “”**	Se utiliza para crear una nueva unidad organizativa. Se especifica el nombre de la OU que se va a crear y especifica la ruta donde se creará la OU dentro de la estructura de Active Directory.

2.4. Crear grupos

El script **“CrearGRUPO.ps1”** crea grupos en un entorno de Active Directory, asignándolos a unidades organizativas específicas. Cada grupo se crea con un nombre designado para los distintos departamentos.

```
# Definir los nombres completos de las OU
$ouDesarrollo = "OU=ou.Desarrollo,OU=ou.SST,DC=dom,DC=SST"
$ouSoporteTecnico = "OU=ou.SoporteTecnico,OU=ou.SST,DC=dom,DC=SST"
$ouVentasyMarketing = "OU=ou.VentasyMarketing,OU=ou.SST,DC=dom,DC=SST"
$ouAdminyFinanzas = "OU=ou.AdminyFinanzas,OU=ou.SST,DC=dom,DC=SST"
$ouAdminSistemas = "OU=ou.AdminSistemas,OU=ou.SST,DC=dom,DC=SST"

# Definir los nombres de los grupos
$grupoDesarrollo = "g.Desarrollo"
$grupoSoporteTecnico = "g.SoporteTecnico"
$grupoVentasyMarketing = "g.VentasyMarketing"
$grupoAdminyFinanzas = "g.AdminyFinanzas"
$grupoAdminSistemas = "g.AdminSistemas"

# Crear los grupos para cada OU
New-ADGroup -Name $grupoDesarrollo -GroupCategory Security -GroupScope Global -Path
$ouDesarrollo
New-ADGroup -Name $grupoSoporteTecnico -GroupCategory Security -GroupScope Global -Path
$ouSoporteTecnico
New-ADGroup -Name $grupoVentasyMarketing -GroupCategory Security -GroupScope Global -
Path $ouVentasyMarketing
New-ADGroup -Name $grupoAdminyFinanzas -GroupCategory Security -GroupScope Global -Path
$ouAdminyFinanzas
New-ADGroup -Name $grupoAdminSistemas -GroupCategory Security -GroupScope Global -Path
$ouAdminSistemas
```

Comando	Explicación
`$ouDesarrollo = "OU=ou.Desarrollo,OU=ou.SST,DC=dom,DC=SST"`	En este bloque de código se define la ruta completa de una OU utilizando una variable. Esto permite almacenar la ruta en una variable para luego poder reutilizarla.
`$grupoDesarrollo = "g.Desarrollo"`	Lo mismo ocurre con el nombre del gupo. Se almacena el nombre del grupo en una variable para luego reutilizarla.
`New-ADGroup -Name $grupoDesarrollo -GroupCategory Security -GroupScope Global -Path $ouDesarrollo`	Se crea el grupo del nombre especificado en la variable `$grupoDesarrollo`. Luego al establecer **GroupCategory como Security** y **GroupScope como Global**, se esta creando un nuevo grupo de seguridad que se puede utilizar para asignar permisos y derechos a los usuarios dentro del dominio. Finalmente el grupo se almaena en la variable `$ouDesarrollo` que especifica la ruta completa donde se creará el grupo

2.5. Crear usuarios

El script **“CrearUSUARIOS.ps1”** crea usuarios en un entorno de Active Directory, asignándolos a unidades organizativas específicas. Los usuarios son creados para diferentes departamentos. Cada usuario tiene un nombre completo y un nombre de cuenta único (SamAccoutName).

```
# Definir los nombres completos de las OUs
$ouDesarrollo = "OU=ou.Desarrollo,OU=ou.SST,DC=dom,DC=SST"
$ouSoporteTecnico = "OU=ou.SoporteTecnico,OU=ou.SST,DC=dom,DC=SST"
$ouVentasyMarketing = "OU=ou.VentasyMarketing,OU=ou.SST,DC=dom,DC=SST"
$ouAdminyFinanzas = "OU=ou.AdminyFinanzas,OU=ou.SST,DC=dom,DC=SST"
$ouAdminSistemas = "OU=ou.AdminSistemas,OU=ou.SST,DC=dom,DC=SST"

# Crear los usuarios para AdminSistemas
New-ADUser -Name "Neus Cháfer Sanjuan" -SamAccountName "nchafer" -Path $ouAdminSistemas

# Crear los usuarios para Desarrollo
New-ADUser -Name "Juan Pablo Lopez" -SamAccountName "jlopez" -Path $ouDesarrollo
New-ADUser -Name "Marta Garcia Miravalles" -SamAccountName "mgarcia" -Path
$ouDesarrollo
New-ADUser -Name "Lorena Girona Marquez" -SamAccountName "lmarquez" -Path $ouDesarrollo
New-ADUser -Name "Estela Gimenez Pastor" -SamAccountName "egimenez" -Path $ouDesarrollo
New-ADUser -Name "Jorge Mateu Estubeny" -SamAccountName "jmateu" -Path $ouDesarrollo
New-ADUser -Name "Macarena Quesada Colomar" -SamAccountName "mquesada" -Path
$ouDesarrollo
New-ADUser -Name "Oscar Juarez Moreno" -SamAccountName "ojuarez" -Path $ouDesarrollo

# Crear los usuarios para SoporteTecnico
New-ADUser -Name "Javier Gómez Martinez" -SamAccountName "jgomez" -Path
$ouSoporteTecnico
New-ADUser -Name "Noelia Molina Sanjuán" -SamAccountName "nmolina" -Path
$ouSoporteTecnico
New-ADUser -Name "Pablo Cháfer Rey" -SamAccountName "pcháfer" -Path $ouSoporteTecnico
New-ADUser -Name "Sergio Climente Valles" -SamAccountName "sclimente" -Path
$ouSoporteTecnico
New-ADUser -Name "Julia Vidal Hernández" -SamAccountName "jvidal" -Path
$ouSoporteTecnico
New-ADUser -Name "Gimeno Ramir Aguado" -SamAccountName "gramir" -Path $ouSoporteTecnico
New-ADUser -Name "Rafa Nadal Perales" -SamAccountName "rnadal" -Path $ouSoporteTecnico
```

```
# Crear los usuarios para VentasyMarketing
New-ADUser -Name "Esmeralda Virginia Bataller" -SamAccountName "evbataller" -Path
$ouVentasyMarketing
New-ADUser -Name "Victor Ramos Alvino" -SamAccountName "vramos" -Path
$ouVentasyMarketing
New-ADUser -Name "Marta Carpi Estocolmo" -SamAccountName "mcarpi" -Path
$ouVentasyMarketing
New-ADUser -Name "Raúl Ferrer Mateu" -SamAccountName "rferrer" -Path
$ouVentasyMarketing
New-ADUser -Name "Perla Bilal Molina" -SamAccountName "pbilal" -Path
$ouVentasyMarketing
New-ADUser -Name "David Requena Morella" -SamAccountName "drequena" -Path
$ouVentasyMarketing
New-ADUser -Name "Pedro Sanchez Cucarella" -SamAccountName "psanchez" -Path
$ouVentasyMarketing
New-ADUser -Name "Silvia Gimenez Sanchis" -SamAccountName "sgimenez" -Path
$ouVentasyMarketing

# Crear los usuarios para AdminyFinanzas
New-ADUser -Name "Esther Llutx Llorens" -SamAccountName "ellutx" -Path
$ouAdminyFinanzas
New-ADUser -Name "Carlos Balaguer Corber" -SamAccountName "cbalaguer" -Path
$ouAdminyFinanzas
New-ADUser -Name "Virginia Bataller Alventosa" -SamAccountName "vbalaller" -Path
$ouAdminyFinanzas
New-ADUser -Name "Rafa Murillo Carpio" -SamAccountName "rmurillo" -Path
$ouAdminyFinanzas
New-ADUser -Name "Enrique Iglesias Perez" -SamAccountName "eiglesias" -Path
$ouAdminyFinanzas
```

Comando	Explicación
`$ouDesarrollo = "OU=ou.Desarrollo,OU=ou.SST,DC=dom,DC=SST"`	En este bloque de código se define la ruta completa de una OU utilizando una variable. Esto permite almacenar la ruta en una variable para luego poder reutilizarla.
`New-ADUser -Name "Neus Cháfer Sanjuan" -SamAccountName "nchafer" -Path $ouAdminSistemas`	Especifica el nombre completo del usuario que se va a crear. Luego se establece el SamAccountName (especifica el nombre de cuenta único), se utiliza para iniciar sesión en el dominio de Active Directory. En cuanto a la ruta la obtenemos de la varable en la que se ha almacenado.

2.6. Añadir el usuario a un grupo

El script **“AñadirUserGroup.ps1”** agrega usuarios a sus grupos corresondientes en un entorno de Active Directory. Utiliza el cmdlet *Add-ADGroupMember* para asignar usuarios a grupos específicos. Esto permite una gestión eficiente de los permisos y accesos dentro del sistema.

```
# Agregar los usuarios a sus respectivos grupos

# Desarrollo
Add-ADGroupMember -Identity "g.Desarrollo" -Members "jlopez", "mgarcia", "lmarquez", "egimenez", "jmateu", "mquesada", "ojuarez"

# Soporte Técnico
Add-ADGroupMember -Identity "g.SoporteTecnico" -Members "jgomez", "nmolina", "pcháfer", "sclimente", "jvidal", "gramir", "rnadal"

# Ventas y Marketing
Add-ADGroupMember -Identity "g.VentasyMarketing" -Members "evbataller", "vramos", "mcarpi", "rferrer", "pbilal", "drequena", "psanchez", "sgimenez"

# Administración y Finanzas
Add-ADGroupMember -Identity "g.AdminyFinanzas" -Members "ellutx", "cbalaguer", "vbalaller", "rmurillo", "eiglesias"

# AdminSistemas
Add-ADGroupMember -Identity "g.AdminSistemas" -Members "nchafer"
```

Comando	Explicación
Add-ADGroupMember	Se utiliza para agregar miembros a un grupo en Active Directory
-Identity “g.Desarrollo”	Especifica el nombre del grupo al que se van a agregar los miembros.
-Members "jlopez", "mgarcia", "lmarquez", "egimenez", "jmateu", "mquesada", "ojuarez"	Este parámetro especifica los nombres de los usuarios que se van a agregar como miembros del grupo.

2.7. Habilitar la cuenta de usuaro

El script **“HabilitarCuenta.ps1”** se encarga de habilitar las cuentas de usuario en Active Directory y establecer una contraseña inicial para cada una de ellas.

Primero, el script define las credenciales del administrador del dominio utilizando el cmdlet “*Get-Credential” .* Luego, se crea un listado que contiene los nombres completos de los usuarios y sus cuentas SAM.

Después, itera sobre cada usuario en el listado y establece una contraseña inicial utilizando el valor "**20012506N**." en el cmdlet “*ConvertTo-SecureString”*. Posteriormente, utiliza el cmdlet "*Set-ADAccountPassword*" para establecer la nueva contraseña y el parámetro "-*Reset*" para forzar el cambio de contraseña en el próximo inicio de sesión.

Finalmente, utiliza el cmdlet "*Enable-ADAccount*" para habilitar la cuenta de usuario. Durante el proceso, el script imprime un mensaje indicando que se ha configurado la cuenta de usuario para iniciar sesión con la contraseña inicial.

```powershell
# Definir las credenciales del administrador del dominio
$credential = Get-Credential

# Lista de usuarios y sus cuentas SAM
$usuarios = @{
    "Neus Cháfer Sanjuan" = "nchafer";
    "Juan Pablo Lopez" = "jlopez";
    "Marta Garcia Miravalles" = "mgarcia";
    "Lorena Girona Marquez" = "lmarquez";
    "Estela Gimenez Pastor" = "egimenez";
    "Jorge Mateu Estubeny" = "jmateu";
    "Macarena Quesada Colomar" = "mquesada";
    "Oscar Juarez Moreno" = "ojuarez";
    "Javier Gómez Martinez" = "jgomez";
    "Noelia Molina Sanjuán" = "nmolina";
    "Pablo Cháfer Rey" = "pcháfer";
    "Sergio Climente Valles" = "sclimente";
    "Julia Vidal Hernández" = "jvidal";
    "Gimeno Ramir Aguado" = "gramir";
    "Rafa Nadal Perales" = "rnadal";
    "Esmeralda Virginia Bataller" = "evbataller";
    "Victor Ramos Alvino" = "vramos";
    "Marta Carpi Estocolmo" = "mcarpi";
    "Raúl Ferrer Mateu" = "rferrer";
    "Perla Bilal Molina" = "pbilal";
    "David Requena Morella" = "drequena";
    "Pedro Sanchez Cucarella" = "psanchez";
    "Silvia Gimenez Sanchis" = "sgimenez";
    "Esther Llutx Llorens" = "ellutx";
    "Carlos Balaguer Corber" = "cbalaguer";
    "Virginia Bataller Alventosa" = "vbalaller";
    "Rafa Murillo Carpio" = "rmurillo";
    "Enrique Iglesias Perez" = "eiglesias"
}

# Iterar sobre cada usuario y establecer la contraseña
foreach ($userFullName in $usuarios.Keys) {
    $samAccountName = $usuarios[$userFullName]
    $newPassword = ConvertTo-SecureString "20012506N." -AsPlainText -Force
    # Establecer la contraseña inicial
    Set-ADAccountPassword -Identity $samAccountName -NewPassword $newPassword  -Reset

    # Habilitar la cuenta de usuario
    Enable-ADAccount -Identity $samAccountName

    Write-Host "Se ha configurado la cuenta de $userFullName ($samAccountName) para
iniciar sesión con la contraseña inicial."

}
```

Comando	Explicación
`$credential = Get-Credential`	La variable **`$credential`** está presente pero no se utiliza en ninguna parte del script. Puede ser eliminada sin afectar el funcionamiento del script, a menos que se planeen implementar operaciones que requieran las credenciales del administrador del dominio.
`$usuarios = @{...}`	Este es un hashtable que mapea los nombres completos de los usuarios a sus nombres de cuenta SAM. Cada clave es el nombre completo de un usuario, y cada valor es el nombre de su cuenta SAM.
`foreach ($userFullName in $usuarios.Keys) { ... }`	Este es un bucle foreach que itera sobre cada clave (nombre completo de usuario) en el hashtable **$usuarios** La variable **$userFullName** se utiliza como clave para buscar el nombre de cuenta SAM asociado en el *hashtable*. El hashtable son los nombres completos de los usuarios, y los valores son sus respectivos nombres de cuenta SAM. En el contexto del script **$usuarios.Keys** se utiliza para obtener una colección de todas las claves (en este caso, los nombres completos de los usuarios) presentes en el hashtable **$usuarios**.

`$samAccountName = $usuarios[$userFullName]`
 	
Seria lo mismo que:
`$usuarios["Neus Cháfer Sanjuan"]`
	El script itera sobre cada nombre completo de usuario almacenado en la variable **$usuarios** y asigna el nombre de cuenta SAM correspondiente a cada uno a la variable **$samAccountName.**

`$newPassword = ConvertTo-SecureString “20012506N” -AsPlainText -Force`	Se establece una contraseña inicial para el usuario. Se convierte la cadena "**20012506N**." en un SecureString, que es un formato seguro para almacenar contraseñas. (esto se almacena en la variable **$newPassword**) **-AsPlainText** indica que la cadena de texto proporcionada se trata como texto sin formato.
`Set-ADAccountPassword -Identity $samAccountName -NewPassword $newPassword -Reset`	Establece la nueva contraseña para la cuenta de usuario especificada por **$samAccountName**. El parámetro **-Reset** indica que se debe forzar el cambio de contraseña en el próximo inicio de sesión.
`Enable-ADAccount -Identity $samAccountName`	Habilita la cuenta de usuario especificada por **$samAccountName**.
`Write-Host ""`	Imprime un mensaje con el nombre completo de usuario y su SamAccountName indicando que se a configurado la cuenta para inciar sesión.

2.8. Instalar politicas de grupo

El script **“InstalarGroupPolicy.ps1”** se utiliza para instalar la característica *RSAT (Herramientas de administración remota del servidor)* para PowerShell relacionadas con Active Directory en un equipo Windows.

```
# Para configurar las GPOs en PowerShell, primero nos debemos asegurar de tener instalado el módulo GroupPolicy en el equipo.

Install-WindowsFeature -Name RSAT-AD-PowerShell
```

Comando	Explicación
Install-WindowsFeature	Se utiliza para instalar una o varias características en un servidor que ejecute el sistema operativo Windows Server.
-Name RSAT-AD-PowerShell	Especifica el nombre de la característica que se quiere instalar. En este caso, *RSAT-AD-PowerShell* se refiere a las Herramientas de administración remota del servidor para PowerShell relacionadas con Active Directory. Estas herramientas proporcionan cmdlets que permiten administrar varios aspectos de Active Directory desde la línea de comandos de PowerShell, como para configurar las GPOs (Objetos de directiva de grupo) y realizar otras tareas de administración relacionadas con Active Directory de forma remota desde PowerShell.

2.9. Directiva de bloqueo de cuenta

El script **“DirectivaBloqueoCuenta.ps1”** se encarga de aplicar la configuración de la directiva “*Default Domain Controllers Policy*” a la unidad organizativa (OU) principal “**ou.SST**” del dominio “**Dom.SST**”.

```
try {
    Write-Output "Aplicando la configuración de la directiva Default Domain Controllers Policy a la OU principal ou.SST del dominio Dom.SST..."

    # Obtener la OU principal ou.SST del dominio Dom.SST
    $ouPrincipal = Get-ADOrganizationalUnit -Filter "Name -eq 'ou.SST'" -SearchBase "DC=Dom,DC=SST"

    # Verificar si se encontró la OU principal
    if ($ouPrincipal -eq $null) {
        Write-Output "Error: No se pudo encontrar la OU principal ou.SST del dominio Dom.SST."
        return
    }

    # Obtener la directiva Default Domain Controllers Policy
    $dcPolicy = Get-GPO -Name "Default Domain Controllers Policy"

    # Verificar si se encontró la directiva
    if ($dcPolicy -eq $null) {
        Write-Output "Error: No se pudo encontrar la directiva Default Domain Controllers Policy."
        return
    }

    # Aplicar la directiva a la OU principal ou.SST
    New-GPLink -Name $dcPolicy.DisplayName -Target $ouPrincipal .DistinguishedName

    # Habilitar la herencia de GPOs en la OU principal ou.SST
    Set-ADOrganizationalUnit -Identity $ouPrincipal .DistinguishedName -ProtectedFromAccidentalDeletion $false

    Write-Output "Configuración de la directiva aplicada correctamente a la OU principal ou.SST del dominio Dom.SST y la herencia de GPOs habilitada."
} catch {
    Write-Output "Se produjo un error al aplicar la configuración de la directiva: $_"
}
```

Comando	Explicación
`Try {...} catch {...}`	**Try** y **catch** permiten anticipar y gestionar problemas mientras el script está en ejecución. Si algo sale mal dentro del bloque **Try**, el programa pasa automáticamente al bloque **catch**, donde se imprime un mensaje de error para manejar la situación de forma segura.
`Write-Output “”`	Imprime un mensaje.
`$ouPrincipal = Get-ADOrganizationalUnit -Filter "Name -eq 'ou.SST'" -SearchBase "DC=Dom,DC=SST"`	Este comando busca la *OU principal* llamada "**ou.SST**" dentro del dominio "**Dom.SST**" y almacena el resultado en la variable `$ouPrincipal`
`if ($ouPrincipal -eq $null)`	Verifica si la variable `$ouPrincipal` es nula, lo que indica que la OU principal no fue encontrada.
`$dcPolicy = Get-GPO -Name "Default Domain Controllers Policy"`	Busca la directiva llamada **“Default Domain Controllers Policy”** y almacena el resultado en la variable **$dcPolicy.**
`New-GPLink -Name $dcPolicy.DisplayName -Target $ouPrincipal .DistinguishedName`	Vincula la directiva obtenida en el paso anterior a la *OU principal*. Esto significa que la configuración de la directiva se aplicará a todos los objetos dentro de esta OU. **.DistinguishedName** ruta de acceso completa de la Unidad Organizativa (OU) principal que se obtuvo previamente con el comando Get-ADOrganizationalUnit.
`Set-ADOrganizationalUnit -Identity $ouPrincipal .DistinguishedName -ProtectedFromAccidentalDeletion $false`	Deshabilita la protección contra eliminación accidental de la OU principal. Esto es necesario para permitir que las directivas de grupo se hereden en esta OU.

2.10. Directiva de contraseña

El script **“PoliticaContraseña.ps1”** establece una política de contraseña compleja.

Primero, crea una nueva Directiva de Grupo (GPO) llamada **"PoliticaContraseña"**. Luego, configura la política de contraseña compleja estableciendo un valor del Registro en la clave "*HKLM\Software\Policies\Microsoft Services\AdmPwd*" con un valor de 1 (indicando complejidad de contraseña).

Posteriormente, asigna esta GPO a todas las Unidades Organizativas (OUs) disponibles en el dominio. Si alguna operación falla, el script captura la excepción y muestra un mensaje de error.

```
Try {
    # Crear la GPO
    New-GPO -Name "PoliticaContraseña" -ErrorAction Stop

    # Configurar la política de contraseña compleja
    Set-GPRegistryValue -Name "PoliticaContraseña" -Key "HKLM\Software\Policies\
Microsoft Services\AdmPwd" -ValueName "PasswordComplexity" -Type DWord -Value 1 -
ErrorAction Stop

    # Asignar la GPO a todas las OUs
    Get-ADOrganizationalUnit -Filter * | ForEach-Object {
        New-GPLink -Name "PoliticaContraseña" -Target $_ -ErrorAction Stop
    }
    Write-Host "La GPO 'PoliticaContraseña' se ha establecido correctamente."
} catch {
    Write-Host "Error al establecer la GPO: $_"
}
```

Comando	Explicación
`Try { ... } catch { ... }`	Dentro del bloque **Try** se ejecutará y cualquier excepción que ocurra se manejará en el bloque **catch**. Si ocurre una excepción dentro del bloque **Try**, se ejecutará el bloque **catch** para manejarla.
`New-GPO -Name "PoliticaContraseña" -ErrorAction Stop`	Se crea una nueva Directiva de Grupo con el nombre "**PoliticaContraseña".** El parámetro **-ErrorAction Stop** se asegura de que si ocurre algún error al crear la GPO, se detenga la ejecución del script y se maneje en el bloque **catch**.
Set-GPRegistryValue `-Name "PoliticaContraseña" -Key "HKLM\Software\Policies\Microsoft Services\AdmPwd" -ValueName "PasswordComplexity" -Type DWord -Value 1 -ErrorAction Stop`	Configura una ***política de registro*** en la GPO recién creada. Establece la complejidad de la contraseña en el Registro del sistema operativo. Si ocurre algún error, se manejará en el bloque **catch**.

`Get-ADOrganizationalUnit -Filter * | ForEach-Object { ... }`
 	
El codigo entre llaves es el siguiente:
 	
`New-GPLink -Name "PoliticaContraseña" -Target $_ -ErrorAction Stop`
	Se obtiene todas las Unidades Organizativas (OUs) en el dominio y luego itera sobre cada una de ellas. Para cada OU, se ejecuta el bloque de código entre llaves.
	
	El codigo entre llaves vincula la GPO **"PoliticaContraseña"** a la OU actual en la iteración.
	
	$_. se refiere a la OU actual en la iteración. En este caso, como estamos iterando sobre *todas las Unidades Organizativas* del dominio utilizando ***Get-ADOrganizationalUnit -Filter ****, **$_** representará cada OU individualmente en cada iteración.

2.11. Directiva firewall

El script “**PoliticaFirewall.ps1”** define reglas de firewall para permitir el acceso a Internet y las aplica a todas las unidades organizativas del dominio. Primero, define las reglas de firewall en un *hashtable* (colección de pares clave-valor, donde cada clave única se asocia con un valor. Esto permite acceder a los valores de manera eficiente utilizando sus claves) y crea una nueva GPO llamada “**PoliticaFirewall**.

Luego, asigna las reglas de firewall a la nueva GPO y vincula la GPO a todas las OUs específicas del dominio. Si hay errores durante el proceso, muestra mensajes correspondientes. Al final, notifica que se ha aplicado la política de firewall.

```
# Definir las reglas de firewall para permitir acceso a Internet
$FirewallRules = @{
    "AllowInternetTCP" = @{
        DisplayName = "Allow Internet TCP"
        Protocol = "TCP"
        RemoteAddress = "Any"
        RemotePort = "Any"
        Direction = "Outbound"
        Action = "Allow"
    }
    "AllowInternetUDP" = @{
        DisplayName = "Allow Internet UDP"
        Protocol = "UDP"
        RemoteAddress = "Any"
        RemotePort = "Any"
        Direction = "Outbound"
        Action = "Allow"
    }
}

# Crear la nueva GPO llamada "PoliticaFirewall"
$NewGPO = New-GPO -Name "PoliticaFirewall"

# Asignar las reglas de firewall a la nueva GPO
foreach ($rule in $FirewallRules.GetEnumerator()) {
    $RuleName= $rule.Key
    $RuleProperties = $rule.Value

  # Verificar si la regla de firewall ya existe
    $existingRule = Get-NetFirewallRule -DisplayName  $RulePropertie.DisplayName -
ErrorAction SilentlyContinue
```

```
    if (-not $existingRule) {
        # Si la regla no existe, crearla
        New-NetFirewallRule -DisplayName  $RulePropertie.DisplayName -Name $RuleName -
Protocol  $RulePropertie.Protocol -RemoteAddress  $RulePropertie.RemoteAddress -
RemotePort  $RulePropertie.RemotePort -Direction  $RulePropertie.Direction -Action
$RulePropertie.Action -Profile Any -Enabled True

    } else {
        Write-Host "La regla de firewall '$($RulePropertie.DisplayName)' ya existe."
    }

    # Configurar la política de GPO
        Set-GPRegistryValue -Name $NewGPO.DisplayName -Key "HKLM\SYSTEM\
CurrentControlSet\Services\SharedAccess\Parameters\FirewallPolicy\StandardProfile" -
ValueName "EnableFirewall" -Type DWord -Value 1
}

# Obtener las OUs específicas en el dominio
$OUs = Get-ADOrganizationalUnit -Filter * -SearchBase "OU=ou.SST,DC=dom,DC=SST"

# Vincular la nueva GPO a todas las OUs del dominio
foreach ($OU in $OUs) {
    Try {
        New-GPLink -Name "PoliticaFirewall" -Target $OU.DistinguishedName -LinkEnabled
Yes -ErrorAction Stop
        Write-Host "La GPO 'PoliticaFirewall' se ha vinculado correctamente a la OU $
($OU.DistinguishedName)."
    } catch {
        Write-Host "Error al vincular la GPO 'PoliticaFirewall' a la OU $
($OU.DistinguishedName): $_"
    }
}

Write-Host "Se ha aplicado la política de Firewall para permitir acceso a Internet a
todas las unidades organizativas del dominio."
```

Comando	Explicación

`$FirewallRules = @{`
`"AllowInternetTCP" = @{..}`
`"AllowInternetUDP" = @{..}`
`}`
`DisplayName = "UDP o TCP"`
`Protocol = "TCP" o "UDP"`
`RemoteAddress = "Any"`
`RemotePort = "Any"`
`Direction = "Outbound"`
`Action = "Allow"`
	Se define un hashtable llamado **$FirewallRules** que contiene dos reglas de firewall: una para permitir tráfico TCP saliente hacia Internet y otra para permitir tráfico UDP saliente hacia Internet. Cada **regla** está representada por un **nombre único (clave)** y un conjunto de **propiedades (valor**) que describen la configuración de la regla.
	La propiedad ***DisplayName*** especifica el nombre descriptivo de la regla de firewall, mientras que ***Protocol*** indica el protocolo de red permitido.
	RemoteAddress especifica la dirección remota desde la que se permite el acceso, ***RemotePort*** especifica el puerto remoto al que se permite el acceso. ***Direction*** indica la dirección del tráfico que la regla permite y ***Action*** determina si se permite o se bloquea el tráfico que coincide con la regla.

`$NewGPO = New-GPO -Name "PoliticaFirewall"`	Crea una nueva Directiva de Grupo llamada **"PoliticaFirewall"** utilizando el cmdlet New-GPO. Esta GPO se utilizará para aplicar las reglas de firewall a las unidades organizativas del dominio (se almacena enla variable `$NewGPO`).

`foreach ($rule in $FirewallRules.GetEnumerator()) {`
`$RuleName = $rule.Key`
`$RuleProperties = $rule.Value`
`New-NetFirewallRule..`
`Set-GPRegistryValue..`
`}`
	Este bucle foreach itera sobre cada regla definida en `$FirewallRules` . Para cada regla, crea una nueva regla de firewall utilizando el cmdlet **New-NetFirewallRule** (ingresa el **valor** de cada **clave). Cada propiedad se obtiene del `$FirewallRules`. a través de la variable `$RuleProperties` que representa los valores asociados con la regla actual en el bucle foreach.** Luego configura la GPO recién creada para habilitar el firewall en el sistema utilizando **Set-GPRegistryValue.**
	En el bucle foreach, **$rule** representa cada *elemento individual* dentro de `$FirewallRules`. Cuando usamos **`$rule`.Key** y **`$rule`.Value**, estamos accediendo a la **clave** y al **valor** de cada elemento dentro de `$FirewallRules`.

`$existingRule = Get-NetFirewallRule -DisplayName $RuleProperties.DisplayName -ErrorAction SilentlyContinue`	**`Get-NetFirewallRule:`** se utiliza para recuperar reglas de firewall en el sistema. `DisplayName` **`$RuleProperties.DisplayName:`** Especifica que queremos buscar una regla de firewall con el nombre almacenado en la propiedad **DisplayName** de la regla actual del bucle foreach. (Contiene el nombre de la regla actual)
`$OUs = Get-ADOrganizationalUnit -Filter * ..`	Se obtienen todas las OUs del dominio.

`foreach ($OU in $OUs) {`
`New-GPLink -Name "PoliticaFirewall" -Target $OU.DistinguishedName -LinkEnabled Yes..`
`}`
	Se vincula la politica a la OU actual en el bucle.
	`Target $OU.DistinguishedName` especifica la ruta completa de la OU a la que se vinculará la directiva.
	$OU representa cada unidad organizativa individual dentro de la lista de OUs . Mientras que `$OUs` es la lista completa que contiene todas las unidades organizativas obtenidas mediante el comando **`Get-ADOrganizationalUnit`**

2.12. Directiva auditoria

El script **“DirectivaAuditoria.ps1”** define una función llamada que toma dos parámetros: **$NombreGPO** y **$PathOU.** La función crea una nueva GPO con el nombre especificado y la vincula a la Unidad Organizativa. Luego, configura las políticas de auditoría en la GPO para registrar eventos de inicio de sesión, acceso a objetos y cambios de política, registrando tanto éxitos como fallos. Si todo se ejecuta correctamente, se muestra un mensaje confirmando la creación y configuración de la GPO. Finalmente, la función es llamada con los valores PoliticaAuditoria para **$NombreGPO** y OU=**ou.AdminSistemas**,OU=**ou.SST**,DC=**dom**,DC=**SST** para **$PathOU**.

```
# Función para crear y configurar la política de auditoría
function ConfigurarPoliticaAuditoria {
    param (
        [string]$NombreGPO,
        [string]$PathOU
    )

    Try {

        # Crear la GPO
        New-GPO -Name $NombreGPO -ErrorAction Stop

        # Vincular la GPO a la OU
        New-GPLink -Name $NombreGPO -Target $PathOU -LinkEnabled Yes -ErrorAction Stop

        # Configurar la auditoría en la GPO
        $AuditSettings = @{
            'AuditLogon' = 'Success,Failure':
            'AuditObjectAccess' = 'Success,Failure'
            'AuditPolicyChange' = 'Success,Failure'
        }

        foreach ($key in $AuditSettings.Keys) {
            Set-GPRegistryValue -Name $NombreGPO -Key "HKLM\SOFTWARE\Microsoft\Windows\
CurrentVersion\Policies\System\Audit"    -ValueName    $key    -Type    String    -Value
$AuditSettings[$key] -ErrorAction Stop
        }

         Write-Host "La GPO '$NombreGPO' se ha creado correctamente y se ha configurado
la auditoría."
    }
    catch {
        Write-Host "Error al crear la GPO: $_"
    }
}

# Llamar a la función para crear y configurar la política de auditoría
$NombreGPO = "PoliticaAuditoria"
$PathOU  =  "OU=ou.AdminSistemas,OU=ou.SST,DC=dom,DC=SST"ConfigurarPoliticaAuditoria  -
NombreGPO $NombreGPO -PathOU $PathOU
```

Comando	Explicación
`function ConfigurarPoliticaAuditoria { param ( [string]$NombreGPO, [string]$PathOU ) }`	La función "ConfigurarPoliticaAuditoria" configura la política de auditoría en un entorno de Active Directory. Toma dos parámetros: **$NombreGPO** y **$PathOU.** **$NombreGPO** es el nombre de la Política de Grupo que se utilizará para configurar la política de auditoría. **$PathOU** es la ruta de la Unidad Organizativa en la que se aplicará la configuración de la política de auditoría. La función configurará la política de auditoría según los parámetros proporcionados.
New-GPO -Name $NombreGPO -ErrorAction Stop	Crea una nueva GPO con el nombre especificado.
New-GPLink -Name $NombreGPO -Target $PathOU -LinkEnabled Yes -ErrorAction Stop	**-Name:** Especifica el nombre de la GPO que se va a vincular. **-Target:** Especifica la ubicación en la que se va a vincular la GPO (sitio, dominio o OU). **-LinkEnabled:** Especifica si el vínculo está habilitado o no.
`$AuditSettings = @{ 'AuditLogon' = 'Success,Failure' 'AuditObjectAccess'='Success,Failure' 'AuditPolicyChange'='Success,Failure' }`	Establece la configuración de auditoría en un sistema. `'AuditLogon' = 'Success,Failure':` Configura la auditoría de inicio de sesión para registrar tanto los inicios de sesión exitosos como los fallidos. `'AuditObjectAccess' = 'Success,Failure' :` Configura la auditoría de acceso a objetos para registrar tanto los accesos exitosos como los fallidos a los objetos del sistema. `'AuditPolicyChange' = 'Success,Failure':` **C**onfigura la auditoría de cambios de políticas para registrar tanto los cambios exitosos como los fallidos en las políticas del sistema. Todo lo guarda en la variabe `$AuditSettings`

`foreach ($key in $AuditSettings.Keys) Set-GPRegistryValue -Name $NombreGPO -Key "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System\Audit"`
 	
`-ValueName $key -Type String -Value $AuditSettings[$key] -ErrorAction Stop`
	Configura los valores del Registro de Windows para la auditoría en la GPO. Se establecen los valores de auditoría para eventos como el inicio de sesión, el acceso a objetos y el cambio de políticas.
	-Key: La ruta de la clave de registro en la que se establecerá el valor.
	La variable **$key** contiene los valores de las claves del diccionario `$AuditSettings`. Estos valores se utilizan en el comando **Set-GPRegistryValue** para establecer los valores del registro en una directiva de grupo (GPO)
	El código utiliza el diccionario `$AuditSettings` para obtener los valores que se establecerán en el registro. Cada **clave** del diccionario representa el nombre del valor de registro y el valor asociado representa el valor que se establecerá.

`Write-Host ""`	Muestra un mensaje en la consola

2.13. Perfiles Móviles

El script **“PerfilesMoviles.ps1”** define y configura perfiles móviles para una lista de usuarios en un entorno de Active Directory. Primero, se establece una ruta base donde se almacenarán los perfiles móviles. Luego, se crea una lista de nombres de usuarios. Para cada usuario en la lista, el script obtiene el objeto de usuario correspondiente en Active Directory utilizando el *SamAccountName*. Si el usuario existe, se define la ruta específica del perfil móvil del usuario y se configura esta ruta en Active Directory.
Se imprime un mensaje en la consola confirmando la creación del perfil móvil y especificando la ruta. Si el usuario no se encuentra, se muestra un mensaje de error. Finalmente, se imprime un mensaje indicando que los perfiles móviles han sido creados para los usuarios especificados.

```
# Definir la ruta base donde se almacenarán los perfiles móviles
$ProfilePath = "\\WINSERVER-AD\Users\Administrador\Documents\PerfilesMoviles"

# Lista de nombres de usuarios
$usuarios = "nchafer", "jlopez", "mgarcia", "lmarquez", "egimenez", "jmateu",
"mquesada", "ojuarez", "jgomez", "nmolina", "pcháfer", "sclimente", "jvidal", "gramir",
"rnadal", "evbataller", "vramos", "mcarpi", "rferrer", "pbilal", "drequena",
"psanchez", "sgimenez", "ellutx", "cbalaguer", "vbalaller", "rmurillo", "eiglesias"

foreach ($usuario in $usuarios) {

    # Obtener el usuario por su SamAccountName
    $User = Get-ADUser -Filter { SamAccountName -eq $usuario }

    if ($User -ne $null) {
        $UserProfilePath = "$ProfilePath\$usuario"

        # Configurar el perfil móvil para el usuario
        Set-ADUser $User -ProfilePath $UserProfilePath
          Write-Host "Se ha creado el perfil móvil para $($User.SamAccountName) en la
ruta  $UserProfilePath"
    } else {
         Write-Host "El usuario con el nombre de cuenta $usuario no se encontró en el
directorio activo."
    }
}

Write-Host "Se han creado perfiles móviles para los usuarios especificados."
```

La variable `$usuario` contiene una lista de nombres de usuarios específicos que se desea procesar en el script. Estos nombres de usuarios se proporcionan manualmente como una lista estática en el script.
La razón por la que se incluye esta lista es para especificar explícitamente los usuarios que se quieran procesar en el script. Aunque el script podría haber sido diseñado para obtener automáticamente todos los usuarios del Active Directory y configurar los perfiles móviles para cada uno de ellos, en este caso, se ha optado por proporcionar una lista predefinida de nombres de usuarios.

Esto se debe a varias razones:

1. **Control de procesamiento:** Al especificar manualmente los nombres de usuarios en la lista `$usuarios`, se tiene un mayor control sobre qué usuarios se procesan en el script. Esto puede ser útil si solo se desea configurar los perfiles móviles para un subconjunto específico de usuarios en lugar de todos los usuarios del Active Directory.

2. **Personalización**: Es posible que se quiera aplicar una configuración diferente o realizar un procesamiento especial para ciertos usuarios específicos, y proporcionar una lista manual de nombres de usuarios permite esa flexibilidad.

Comando	Explicación
`$ProfilePath = “..”`	Define la ruta base donde se crearán los perfiles móviles.
`$usuarios = “..”`	Define una lista de nombres de usuarios.

`foreach ($usuario in $usuarios) {`
 	
`}`
	Inicia un bucle que itera sobre cada nombre de usuario en la lista `$usuarios`. `$usuario` es una variable temporal que contiene el valor actual de la lista `$usuario` en cada iteración.

`$User = Get-ADUser -Filter` `{ SamAccountName -eq $usuario }`	Obtiene el objeto de usuario correspondiente en Active Directory utilizando el SamAccountName y la guarda en la variable `$User`
`if ($User -ne $null) {` `}`	Verifica si el usuario existe en Active Directory.
`$UserProfilePath = "$ProfilePath\` `$usuario "`	Define la ruta específica del perfil móvil del usuario concatenando la ruta base con el nombre de usuario y la almacena en la variable `$UserProfilePath`
`Set-ADUser $User -ProfilePath` `$UserProfilePath`	Configura el perfil móvil del usuario en Active Directory estableciendo la propiedad `-ProfilePath` .

`Write-Host “Se ha creado el perfil móvil..”`
`} else {`
`Write-Host “El usuario con el nombre de cuenta $usuario no se encontró..”`
	Imprime un mensaje en la consola confirmando la creación del perfil móvil para el usuario y especificando la ruta del perfil.
	
	Si el usuario no existe se muestra el mensaje indicando que el usuario no se encontró

2.14. Instalar SMB

El script **“InstalarAMB.ps1”** se utiliza para instalar la característica de Windows Server que habilita el protocolo SMB (Server Message Block) versión 1

```
# Instalar SMB para poder utilizar New-SmbShare y compartir la carpeta de PerfilesMoviles si todavía no esta compartida.

Install-WindowsFeature FS-SMB1
```

Comando	Explicación
Install-WindowsFeature	Es un cmdlet de PowerShell que se utiliza para instalar características y roles en Windows Server.
FS-SMB1	Es el nombre de la característica específica que se va a instalar, en este caso, el protocolo SMB versión 1.

Este comando instala el soporte para el protocolo SMB versión 1 en el servidor donde se ejecuta. SMB es un protocolo de red utilizado principalmente para compartir archivos, impresoras y otros recursos entre dispositivos en una red.

Contexto y Uso:

Instalación de SMB v1: SMB v1 es una versión antigua del protocolo SMB. Aunque ha sido reemplazada por versiones más nuevas como SMB v2 y SMB v3, algunas aplicaciones y dispositivos antiguos todavía dependen de SMB v1. Este comando garantiza que el servidor pueda utilizar SMB v1 para compartir recursos. (en mi caso probé con versiones nuevas pero daba problemas)

Preparación para Compartir Carpetas: En el contexto del script, la instalación de SMB v1 es un paso previo necesario para asegurar que el servidor tenga la capacidad de crear y gestionar recursos compartidos utilizando el cmdlet *New-SmbShare*.

2.15. Crear estructura de carpetas

El script **“CrearCarpetas.ps1”** crea y comparte carpetas para perfiles móviles de usuarios en un servidor. Verifica si la carpeta base para perfiles móviles existe; si no, la crea. Luego, comparte esta carpeta si aún no está compartida. Después, obtiene todos los usuarios del dominio y crea carpetas para sus perfiles dentro de la carpeta base. Cada usuario tiene una carpeta con su nombre de cuenta. Al final, se notifica que se han creado las carpetas para los perfiles móviles de los usuarios del dominio.

```
# Definir la ruta base para los perfiles móviles
$BasePath = "\\WINSERVER-AD\Users\Administrador\Documents\PerfilesMoviles"

# Nombre de la carpeta compartida
$ShareName = "PerfilesMoviles"

# Verificar y crear la carpeta base para los perfiles móviles si no existe
if (-not (Test-Path -Path $BasePath  -PathType Container)) {
    New-Item -Path $BasePath  -ItemType Directory
    Write-Host "Se ha creado la carpeta base para perfiles móviles en '$BasePath'."
} else {
    Write-Host "La carpeta base para perfiles móviles en '$BasePath' ya existe."
}
# Compartir la carpeta base para los perfiles móviles si no está compartida
if (-not (Get-SmbShare -Name $ShareName  -ErrorAction SilentlyContinue)) {
    New-SmbShare -Name $ShareName  -Path $BasePath  -FullAccess Everyone
    Write-Host "Se ha compartido la carpeta '$ShareName' como '$BasePath'."
} else {
    Write-Host "La carpeta '$ShareName' ya está compartida."
}
# Obtener todos los usuarios del dominio para los cuales se crearán perfiles móviles
$Usuarios = Get-ADUser -Filter * | Select-Object -ExpandProperty SamAccountName

# Iterar sobre la lista de usuarios y crear perfiles móviles
foreach ($Usuario in $Usuarios) {
    $ProfilePath = Join-Path -Path $BasePath  -ChildPath $Usuario

    # Verificar y crear la carpeta para el perfil móvil del usuario si no existe
    if (-not (Test-Path -Path $ProfilePath -PathType Container)) {
        New-Item -Path $ProfilePath -ItemType Directory
        Write-Host "Se ha creado la carpeta para el perfil móvil de $Usuario en
'$ProfilePath'."
    } else {
        Write-Host "La carpeta para el perfil móvil de $Usuario ya existe en
'$ProfilePath'."
    }
}
Write-Host "Se han creado las carpetas para los perfiles móviles de los usuarios del
dominio."
```

Comando	Explicación
`$BasePath = “..”`	Define la ruta base donde se crearán los perfiles móviles.
`$ShareName = “..”`	Define el nombre de la carpeta compartida.
`if (-not (Test-Path -Path $BasePath -PathType Container)) {`	Verifica si la carpeta base para los perfiles móviles no existe. El parámetro **`-PathType Container`** se utiliza en el comando **`Test-Path`** para especificar qué tipo de objeto se está buscando en la ruta especificada. En este caso, **`Container`** se refiere a un contenedor, es decir, un directorio o carpeta.
`New-Item -Path $BasePath -ItemType Directory`	Crea la carpeta base para los perfiles móviles si no existe.
`if (-not (Get-SmbShare -Name $ShareName -ErrorAction SilentlyContinue)) {`	Verifica si la carpeta base para los perfiles móviles no está compartida.
`New-SmbShare -Name $ShareName -Path $BasePath -FullAccess Everyone`	Comparte la carpeta base para los perfiles móviles con acceso completo para todos.
`$Usuarios = Get-ADUser -Filter * | Select-Object -ExpandProperty SamAccountName`	Obtiene todos los usuarios del dominio y *extrae sus SamAccountName*.
`foreach ($Usuario in $Usuarios) {`	Itera sobre cada usuario en la lista de usuarios, igual que en el script para crear los perfiles móviles inicia un bucle que itera sobre cada nombre de usuario en la variable `$Usuarios` (donde se obtiene todos los usuarios del dominio). `$Usuario` es una variable temporal que contiene el valor actual de la lista `$Usuarios` en cada iteración.
`$ProfilePath = Join-Path -Path $BasePath -ChildPath $Usuario`	Define la ruta específica del perfil del usuario concatenando la ruta base con el nombre de usuario y se almacena en **`$ProfilePath.`**
`if (-not (Test-Path -Path $ProfilePath -PathType Container)) {`	Verifica si la carpeta para el perfil del usuario no existe.
`New-Item -Path $ProfilePath -ItemType Directory`	Crea la carpeta para el perfil del usuario si no existe.

2.16. Establecer permisos en las carpetas

El script **“PermisosCarpetas.ps1”** establece permisos en una carpeta principal compartida para perfiles móviles de usuarios en Active Directory. Define la ruta base y los usuarios con control total y agrega permisos totales a la carpeta principal para todos los usuarios. Luego se itera sobre cada usuario, asignando permisos individuales a sus carpetas. Aplica los permisos y muestra mensajes de confirmación.

```
# Definir la ruta base para los perfiles móviles
$BasePath = "\\WINSERVER-AD\Users\Administrador\Documents\PerfilesMoviles"

# Lista de usuarios que tendrán control total en la carpeta principal
$UsuariosConPermisosTotales = "nchafer", "jlopez", "mgarcia", "lmarquez", "egimenez",
"jmateu", "mquesada", "ojuarez", "jgomez", "nmolina", "pcháfer", "sclimente", "jvidal",
"gramir", "rnadal", "evbataller", "vramos", "mcarpi", "rferrer", "pbilal", "drequena",
"psanchez", "sgimenez", "ellutx", "cbalaguer", "vbalaller", "rmurillo", "eiglesias"

# Agregar permisos a la carpeta principal compartida
$Acl  = Get-Acl -Path $BasePath
$Permission = "Everyone", "FullControl", "Allow"
$Rule = New-Object -TypeName System.Security.AccessControl.FileSystemAccessRule -
ArgumentList $Permission
$Acl .SetAccessRule($Rule)
Set-Acl -Path $BasePath -AclObject $Acl

# Iterar sobre la lista de usuarios y asignar permisos
foreach ($Usuario in $UsuariosConPermisosTotales) {
    $ProfilePath = Join-Path -Path $BasePath -ChildPath $Usuario

    # Verificar si la carpeta del usuario existe
    if (Test-Path -Path $ProfilePath -PathType Container) {
        # Agregar permisos totales al usuario
        $AclUser = Get-Acl -Path $ProfilePath
        $PermissionUser = "Dom\$Usuario", "FullControl", "Allow"
        $RuleUser = New-Object -TypeName
System.Security.AccessControl.FileSystemAccessRule -ArgumentList $PermissionUser
        $AclUser.SetAccessRule( $RuleUser)

        # Deshabilitar la herencia de permisos
        $AclUser.SetAccessRuleProtection($true, $false)

        Set-Acl -Path $ProfilePath -AclObject $AclUser
        Write-Host "Se han configurado los permisos para la carpeta del usuario
$Usuario."
    } else {
        Write-Host "La carpeta del usuario $Usuario no existe en '$ProfilePath'."
    }
}
Write-Host "Se han configurado los permisos para las carpetas personales de los
usuarios del dominio en la carpeta principal compartida."
```

Comando	Explicación
`$BasePath = “..”`	Define la ruta base donde se almacenarán los perfiles móviles.
`$UsuariosConPermisosTotales = “..”`	Crea una lista de usuarios que tendrán control total en la carpeta principal compartida.
`$Acl = Get-Acl -Path $BasePath`	Obtiene la lista de control de acceso (ACL) actual de la carpeta base. ACL es un conjunto de permisos asociados a un objeto, Estos permisos determinan quién puede acceder al objeto y qué tipo de acciones pueden realizar.
`$Permission = "Everyone", "FullControl", "Allow"`	Define el permiso que se otorgará a todos los usuarios (**Everyone**) con control total (**FullControl**) y se almacena en la variable **$Permission.**
`$Rule = New-Object -TypeName System.Security.AccessControl.FileSystemAccessRule -ArgumentList $Permission`	Crea un nuevo objeto de regla de acceso al sistema de archivos con los permisos definidos, esto se almacena en la variable **$Rule**. Especifica el tipo de objeto que se creará. En este caso, **FileSystemAccessRule** es un tipo de objeto que representa una regla de acceso a un sistema de archivos en el espacio de nombres **System.Security.AccessControl.** En este caso, **$Permission** es una lista que contiene los permisos que se otorgarán en la regla de acceso.
`$Acl.SetAccessRule($Rule)`	Agrega la nueva regla de acceso a la lista de control de acceso de la carpeta base.
`Set-Acl -Path $BasePath -AclObject $Acl`	Aplica la nueva lista de control de acceso a la carpeta base.
`foreach ($Usuario in $UsuariosConPermisosTotales) { ... }`	Itera sobre cada usuario en la lista de usuarios con permisos totales.
`$ProfilePath = Join-Path -Path $BasePath -ChildPath $Usuario`	Construye la ruta completa del perfil móvil del usuario y se almacena en **$ProfilePath**
`$AclUser = Get-Acl -Path $ProfilePath`	Obtiene la lista de control de acceso (ACL) actual de la carpeta del usuario.
`$PermissionUser = "Dom\$Usuario", "FullControl", "Allow"`	Define el permiso de control total para el usuario específico.
`$RuleUser = New-Object -TypeName System.Security.AccessControl.FileSystemAccessRule -ArgumentList $PermissionUser`	Crea un nuevo objeto de regla de acceso al sistema de archivos con los permisos definidos para el usuario específico.
`$AclUser.SetAccessRule($RuleUser)`	Agrega la nueva regla de acceso a la lista de control de acceso de la carpeta del usuario.
`$AclUser.SetAccessRuleProtection($true, $false)`	Deshabilita la herencia de permisos para la carpeta del usuario.
`Set-Acl -Path $ProfilePath -AclObject $AclUser`	Aplica la nueva lista de control de acceso a la carpeta del usuario.

2.17. Asistente para la creación de objetos de Active Directory

El script **“Asistente_CreaciónObjetos_AD.ps1”** es un asistente interactivo para crear objetos en Active Directory, incluyendo OUs (unidades organizativas), grupos y usuarios. Proporciona opciones para crear, administrar y organizar objetos de manera eficiente dentro del entorno de Active Directory. Utiliza funciones PowerShell para cada tarea, permitiendo al usuario realizar operaciones específicas según sus necesidades, como crear una OU, un grupo, un usuario, o agregar un usuario a un grupo existente. El script también incluye una opción para salir del asistente.

```
# Función para crear una OU
function CrearOU {
    $NombreOU = Read-Host "Introduce el nombre de la OU"
    $PathPadre = Read-Host "Introduce el path de la OU padre (o 'Dominio' si es en el
nivel de dominio)"

    try {
        New-ADOrganizationalUnit -Name $NombreOU -Path $PathPadre -ErrorAction Stop
        Write-Host "La OU $NombreOU ha sido creada en $PathPadre"
    } catch {
        Write-Host "Error al crear la OU: $_"
    }
}

# Función para crear un grupo
function CrearGrupo {
    $NombreGrupo = Read-Host "Introduce el nombre del grupo"
    $PathOU = Read-Host "Introduce el path de la OU donde deseas crear el grupo"

    try {
        New-ADGroup -Name $NombreGrupo -GroupCategory Security -GroupScope Global -Path
$PathOU -ErrorAction Stop
        Write-Host "El grupo $NombreGrupo ha sido creado en $PathOU"
    } catch {
        Write-Host "Error al crear el grupo: $_"
    }
}

# Función para crear un usuario
function CrearUsuario {
    $Nombre = Read-Host "Introduce el nombre del usuario"
    $SamAccountName = Read-Host "Introduce el SamAccountName del usuario"
    $OU = Read-Host "Introduce la OU donde deseas crear el usuario"

    try {
        New-ADUser -Name $Nombre -SamAccountName $SamAccountName -Path $OU -ErrorAction
Stop
        Write-Host "El usuario $Nombre ha sido creado en $OU"
```

```
    } catch {
        Write-Host "Error al crear el usuario: $_"
    }
}

# Función para agregar un usuario a un grupo
function AgregarUsuarioAGrupo {
    $Usuario = Read-Host "Introduce el SamAccountName del usuario"
    $Grupo = Read-Host "Introduce el nombre del grupo"

    try {
        Add-ADGroupMember -Identity $Grupo -Members $Usuario -ErrorAction Stop
        Write-Host "El usuario $Usuario ha sido agregado al grupo $Grupo"
    }catch {
        Write-Host "Error al agregar el usuario al grupo: $_"
    }
}

# Función para salir del script
function Salir {
    Write-Host "Saliendo del script..."
    exit
}

# Menú principal
do {
    Write-Host "Bienvenido al asistente de creación de objetos en Active Directory"
    Write-Host "Selecciona una opción:"
    Write-Host "1. Crear OU"
    Write-Host "2. Crear Grupo"
    Write-Host "3. Crear Usuario"
    Write-Host "4. Agregar Usuario a Grupo"
    Write-Host "5. Salir del script"

    $opcion = Read-Host "Introduce el número de la opción deseada"

    switch ($opcion) {
        '1' { CrearOU }
        '2' { CrearGrupo }
        '3' { CrearUsuario }
        '4' { AgregarUsuarioAGrupo }
        '5' { Salir }
        default { Write-Host "Opción no válida" }
    }
} while ($true)
```

Comando	Explicación
`Read-Host "Introduce el nombre de la OU"`	Solicita al usuario que introduzca el nombre de la (OU) que quiere crear.
`Read-Host "Introduce el path de la OU padre (o 'Dominio'..)"`	Solicita al usuario que introduzca el path de la OU padre donde se creará la nueva OU, o “Dominio” si es en el nivel de dominio.
`New-ADOrganizationalUnit -Name $NombreOU -Path $PathPadre -ErrorAction Stop`	Crea una nueva Unidad Organizativa (OU) en Active Directory con el nombre y path especificados.
`Read-Host "Introduce el nombre del grupo"`	Solicita al usuario que introduzca el nombre del grupo que quiere crear.
`Read-Host "Introduce el path de la OU donde deseas crear el grupo"`	Solicita al usuario que introduzca el path de la OU donde se creará el nuevo grupo.
`New-ADGroup -Name $NombreGrupo -GroupCategory Security -GroupScope Global -Path $PathOU -ErrorAction Stop`	Crea un nuevo grupo de seguridad en Active Directory con el nombre y path especificados.
`Read-Host "Introduce el nombre del usuario"`	Solicita al usuario que introduzca el nombre del usuario que quiere crear.
`Read-Host "Introduce el SamAccountName del usuario"`	Solicita al usuario que introduzca el SamAccountName del usuario que quiere crear.
`Read-Host "Introduce la OU donde deseas crear el usuario"`	Solicita al usuario que introduzca el path de la OU donde se creará el nuevo usuario.
`New-ADUser -Name $Nombre -SamAccountName $SamAccountName -Path $OU -ErrorAction Stop`	Crea un nuevo usuario en Active Directory con el nombre, SamAccountName y path especificados.
`Read-Host "Introduce el SamAccountName del usuario"`	Solicita al usuario que introduzca el SamAccountName del usuario que se agregará al grupo.
`Read-Host "Introduce el nombre del grupo"`	Solicita al usuario que introduzca el nombre del grupo al que se agregará el usuario.
`Add-ADGroupMember -Identity $Grupo -Members $Usuario -ErrorAction Stop`	Agrega un usuario al grupo especificado en Active Directory.
`$_`	Se utiliza para mostrar el mensaje de error específico que ocurrió durante la ejecución del comando dentro del bloque catch. Esto facilita la depuración y el manejo de errores al proporcionar detalles sobre lo que salió mal en el script.
`do { ... } while ($true)`	Esta es una estructura de bucle **do-while**. Ejecuta repetidamente el bloque de código dentro de las llaves **{ ... }** mientras la condición **$true** sea verdadera.
`switch ($opcion) { ... } ‘1’,’2’,’3’..`	Esta es una estructura de control switch. Evalúa el valor de **$opcion** y ejecuta el bloque de código correspondiente según el número seleccionado por el usuario.

3. Scripts para el Cliente (Windows 10)

Estos scripts están diseñados para ejecutarse en cliente que se unirá al servidor controlador de dominio. Aquí está la lista de los scripts en el orden en que deben ejecutarse:

1. **ConfiguraciónRedCliente.ps1:** Configura la red en el cliente, incluyendo la configuración de direcciones IP, puertas de enlace y servidores DNS.
2. **UnirCliente-Dominio.ps1:** Une el cliente al dominio configurado en el servidor principal controlador de dominio, estableciendo así una relación de confianza y permitiendo la autenticación de usuarios a través del Active Directory del dominio.

3.1. Configuración de Red del Cliente

El script **“ConfiguraciónRedCliente.ps1”** configura la dirección IP y otros parámetros de red para un cliente específico. Define variables para la dirección IP, máscara de subred, puerta de enlace y servidor DNS. Luego, utiliza cmdlets de PowerShell para configurar la dirección IP y la puerta de enlace en el adaptador de red activo, así como para establecer el servidor DNS. También elimina una dirección IP específica si está presente. Finalmente, se incluye un comando para cambiar el nombre del equipo.

Si se necesita eliminar una dirección IP específica, se proporciona un comando para hacerlo, pero se deja comentado por si no es necesario. La inclusión del parámetro **-Confirm:$false** en el cmdlet **Remove-NetIPAddress** evita la solicitud de confirmación para eliminar la dirección IP.

```
# Configuración de la dirección IP y otros parámetros de red
$ipAddress = "192.168.0.31"
$subnetMask = "255.255.255.0"
$gateway = "192.168.0.30"  # Dirección IP del adaptador de red interno del servidor
$dnsServer = "192.168.0.30"  # Dirección IP del servidor DNS del dominio

# Configuración de la dirección IP
$networkAdapter = Get-NetAdapter | Where-Object { $_.Status -eq 'Up' }
New-NetIPAddress -InterfaceIndex $networkAdapter.InterfaceIndex -IPAddress $ipAddress -
PrefixLength 24 -DefaultGateway $gateway

# Configuración del servidor DNS
Set-DnsClientServerAddress -InterfaceIndex $networkAdapter.InterfaceIndex -
ServerAddresses $dnsServer

# Eliminar la dirección IP (..) si está presente
Get-NetIPAddress -IPAddress "192.168.0.3" | Remove-NetIPAddress -Confirm:$false

# Cambio de nombre del equipo
Rename-Computer -NewName "Win10-Cliente1" -Restart
```

Comando	Explicación
`$ipAddress = "192.168.0.31"`	Define la dirección IP que se asignará al cliente.
`$subnetMask = "255.255.255.0"`	Define la máscara de subred para la red.
`$gateway = "192.168.0.30"`	Define la puerta de enlace predeterminada, que es la dirección IP del adaptador de red interno del servidor.
`$dnsServer = "192.168.0.30"`	Define la dirección IP del servidor DNS del dominio.
`$networkAdapter = Get-NetAdapter | Where-Object { $_.Status -eq 'Up' }`	Se utiliza para obtener información sobre los adaptadores de red en un sistema y filtrar aquellos que están activos (es decir, aquellos que están “arriba” o “Up”). **`Where-Object`** : Cmdlet que selecciona los objetos de una colección en función de un criterio. **`$_`**: Representa el objeto actual en la colección que se está procesando (en este caso, cada adaptador de red). **`$_.Status`**: Accede a la propiedad Status del objeto actual (adaptador de red).
`New-NetIPAddress -InterfaceIndex $networkAdapter.InterfaceIndex -IPAddress $ipAddress -PrefixLength 24 -DefaultGateway $gateway`	Crea una nueva dirección IP en una interfaz de red. Especifica el índice de la interfaz de red en la cual se va a configurar la nueva dirección IP. **`$networkAdapter.InterfaceIndex`** es una variable que contiene el índice de la interfaz de red. El índice de la interfaz es un identificador único para cada adaptador de red en el sistema. Define la nueva dirección IP que se asignará a la interfaz de red. `$ipAddress` es una variable que contiene la dirección IP que se desea configurar. En el prefijo de longittud se especifica la longitud del prefijo de la subred y se establece la puerta de enlace predeterminada para la interfaz de red. `$gateway` es una variable que contiene la dirección IP de la puerta de enlace.
`Set-DnsClientServerAddress -InterfaceIndex $networkAdapter.InterfaceIndex -ServerAddresses $dnsServer`	Especifica el índice de la interfaz de red para la cual se van a configurar las direcciones de los servidores DNS. **`$networkAdapter.InterfaceIndex`** es una variable que contiene el índice de la interfaz de red. `$dnsServer` define las direcciones de los servidores DNS que se van a asignar a la interfaz de red. Es una variable que podria contener una o más direcciones IP.
`Get-NetIPAddress -IPAddress "192.168.0.3" | Remove-NetIPAddress -Confirm:$false`	Se recupera la información de las direcciones IP configuradas en el sistema y uego se especifica que solo se debe obtener la dirección IP “**192.168.0.3”.** **`Remove-NetIPAddress`** elimina una dirección IP de una interfaz de red. **`Confirm:$false`** es el parámetro que indica que no se debe pedir confirmación antes de realizar la acción de eliminación.
`Rename-Computer -NewName "Win10-Cliente1" -Restart`	Cambia el nombre del equipo al especificado ("Win10-Cliente1") y reinicia el equipo para que el cambio tenga efecto

3.2. Unión del cliente al controlador de dominio

El script **“UnirCliente-Dominio.ps1”** se utiliza para unir un cliente a un dominio específico en Active Directory. Primero, se define el nombre del dominio al que se quiere unir el cliente. Luego, se utiliza el cmdlet **“Get-Credential”** para solicitar las credenciales del usuario que tiene permiso para unir el equipo al dominio. Después, se utiliza el cmdlet **“Add-Computer”** para realizar la unión al dominio especificado, utilizando el nombre del dominio y las credenciales proporcionadas. Finalmente, se reinicia el equipo para aplicar los cambios.

```
# Unión al dominio
$domainName  = "Dom.SST"
$credential = Get-Credential
Add-Computer -DomainName $domainName-Credential $credential -Restart
```

Comando	Explicación
`$domainName = "Dom.SST"`	Se asigna el nombre de dominio **“Dom.SST”** a la variable $domainName. Este será el dominio al cual el equipo será unido.
`$credential = Get-Credential`	Se solicita al usuario que ingrese sus credenciales, nombre de usuario y contraseña. Estas credenciales son almacenadas de forma segura en la variable **$credential**
Add-Computer -DomainName $domainName-Credential $credential -Restart	**Add-Computer**: Este cmdlet se utiliza para agregar un equipo a un dominio o grupo de trabajo en PowerShell. **-DomainName $domainName**: Parámetro que especifica el nombre del dominio al cual el equipo será unido. Aquí, **$domainName** contiene el nombre de dominio **“Dom.SST”**. **-Credential $credential**: Parámetro que especifica las credenciales de usuario que tienen los permisos necesarios para unir el equipo al dominio. $credential contiene estas credenciales. Después de esto el equipo se reiniciará autoáticamente después de unirse al dominio.