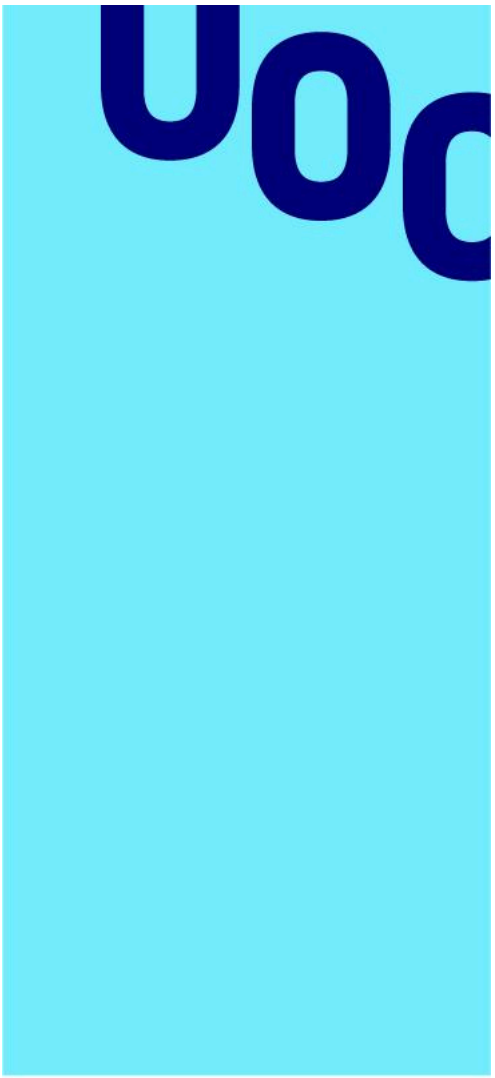
FP.048 - (P) Administración de servidores empresariales

Producto 3. Automatización de tareas y procesos del sistema

Fast&Query

- Rubén Vicente Gilabert
- Vicent Melero Escriba
- Pau Cabanillas Marin
- Andrei Vasiliu



Universitat Oberta de Catalunya

Descripción

En estos momentos, ya tenemos un dominio con el servicio de directorio en funcionamiento y los requerimientos de usuarios y recursos tal como se había previsto en nuestra planificación.

Después de unas semanas con el nuevo sistema en funcionamiento, hemos descubierto que hay una serie de tareas repetitivas que podríamos realizar de una forma más sencilla automatizándolas. Algunas de estas tareas son, por ejemplo, el alta masiva de usuarios, listado de procesos del sistema en funcionamiento, estado de la memoria física..., es por ello que vamos a preparar el sistema para tal fin.

En este caso mediante el uso de lenguajes de guion y automatización de tareas, prepararemos la automatización de las tareas de mantenimiento como son: gestión de usuarios, gestión de procesos, gestión del sistema de ficheros y monitorización del sistema.

Objetivos

Automatizar altas de usuarios del dominio mediante el uso de lenguajes de guión, monitorizando el sistema y gestionando el sistema de ficheros y procesos.

1. Realizar las siguientes especificaciones generales:

I. Generar, para todos los scripts, una cabecera donde aparezca la información del grupo (nombre, miembros…), fecha, versión y descripción breve sobre su funcionalidad y uso.

En todos los scripts del Producto 3 (gestion_usuarios.ps1, gestion_sistema.ps1, tareas_programadas.ps1 y menu_principal.ps1) se ha incluido una **cabecera estándar** al inicio del código.

Esta cabecera contiene siempre:

- Nombre del script
- Módulo / Producto (FP.048 – Producto 3)
- Empresa y grupo de trabajo (Trust SL – Administración de servidores, grupo Fast&Query)
- Miembros del grupo
- Fecha y número de versión
- Una breve descripción del script
- Indicaciones de uso (cómo ejecutarlo, desde dónde y con qué permisos)
- Requisitos previos (versión de PowerShell, necesidad de ejecutar como administrador, rutas necesarias, etc.)

El objetivo de esta cabecera es que cualquier administrador pueda identificar rápidamente quién ha desarrollado el script, para qué sirve, cómo debe ejecutarse correctamente y en qué contexto se utiliza dentro de la solución.

Modelo de cabecera (PowerShell)

gestion_usuarios.ps1

```
#############################################################
# Nombre del script  : gestion_sistema.ps1
# Módulo / Producto  : FP.048 - Producto 3
# Empresa            : Trust SL - Administración de servidores
# Grupo              : Fast&Query
# Miembros           : Rubén Vicente Gilabert
#                      Vicent Melero Escriba
#                      Pau Cabanillas Marin
#                      Andrei Vasiliu
# Fecha              : 18/11/2025
# Version            : 1.4
#
# Descripcion:
#   Script para automatizar la gestion de usuarios y grupos
#   del dominio Trust.lan. Permite comprobar si existen y
#   crearlos de forma interactiva.
#
# Uso:
#   Ejecutar en PowerShell como Administrador del dominio.
#   Ejemplo:
#       .\gestion_usuarios.ps1
#
# Requisitos:
#   - Modulo ActiveDirectory instalado.
#   - Ejecucion con privilegios de administrador.
#############################################################
```

II. Generar los scripts según las especificaciones detalladas en los siguientes puntos. El nombre de las funciones y variables deben ser significativos e identificables.

A la hora de desarrollar los scripts se ha seguido el criterio de utilizar **nombres claros y descriptivos**, tanto para los archivos como para las funciones y variables:

- Los nombres de los scripts indican su responsabilidad principal:
  - gestion_usuarios.ps1: gestión de usuarios y grupos.
  - gestion_sistema.ps1: monitorización, sistema de ficheros y procesos.
  - tareas_programadas.ps1: creación de tareas programadas.
  - menu_principal.ps1: menú general que centraliza todos los scripts anteriores.

- En las funciones se ha utilizado un estilo **verbo-sustantivo**, siguiendo las buenas prácticas de PowerShell, por ejemplo:
  - Test-Administrador: comprueba si el script se ejecuta con privilegios de administrador.
  - Mostrar-Ayuda: muestra información general de uso del script.
  - Verificar-Usuario, Verificar-Grupo: comprueban existencia de usuarios o grupos en el dominio.
  - Crear-Usuario, Crear-Grupo: automatizan la creación de estos objetos.
  - Obtener-EstadoCPU, Obtener-NumeroProcesos, Obtener-EstadoDiscos, Comprobar-EstadoSistema: funciones de monitorización.
  - Buscar-FicherosUsuario, Buscar-FicherosTipo: funciones del sistema de ficheros.
  - Mostrar-ProcesosDosUsuarios, Mostrar-ProcesosUsuariosReales: funciones de gestión de procesos.
  - Crear-Tarea-EstadoSistema, Crear-Tarea-BuscarFicheros: funciones específicas para programar tareas.
  - Menu-Monitoreo, Menu-Ficheros, Menu-Procesos, Mostrar-Menu: gestionan los distintos menús interactivos.

- Las **variables globales** también tienen nombres autoexplicativos, por ejemplo:
  - $BaseDN, $OUGrupos, $OUUsuariosDeptos: describen las rutas LDAP donde se crean usuarios y grupos.
  - $RutaGestionUsuarios, $RutaGestionSistema, $RutaTareasProgramadas: indican claramente la ruta de cada script.
  - $FicheroAlerta: almacena la ruta del fichero alerta.log donde se registran los avisos de monitorización.

Este diseño facilita la **lectura y el mantenimiento** del código, ya que cualquier persona que revise el script puede intuir la finalidad de cada elemento solo con leer su nombre, sin necesidad de interpretar la lógica completa.

III. Documentar todas las funciones dentro del código.

Además de la cabecera general, todas las funciones importantes están **documentadas dentro del propio script**.
La documentación se ha hecho de dos formas:

1. **Bloques de comentarios antes de cada función**, donde se indica:
   - Nombre de la función.
   - Objetivo principal.
   - Ejemplo de esquema utilizado:

```
#------------------------------------------------------------
# Funcion: Test-Administrador
# Objetivo: Verificar que el script se ejecuta como admin
#------------------------------------------------------------
```

2. **Comentarios puntuales dentro del código** para explicar decisiones relevantes, como:
   - Por qué se utiliza un ValidateSet para limitar opciones (evitar que el usuario escriba mal el alcance de un grupo o el departamento).
   - Por qué se guarda la actividad en alerta.log cuando se superan ciertos umbrales de CPU, memoria o disco.
   - Explicaciones sobre el tratamiento de parámetros en el script de tareas programadas (por ejemplo, cómo se construyen los argumentos que se pasan a gestion_sistema.ps1).

IV. Los videos han de ser explicativos y han de detallar la estructura que se ha seguido para desarrollar cada uno de los scripts y su código. Se ha de demostrar su funcionamiento.

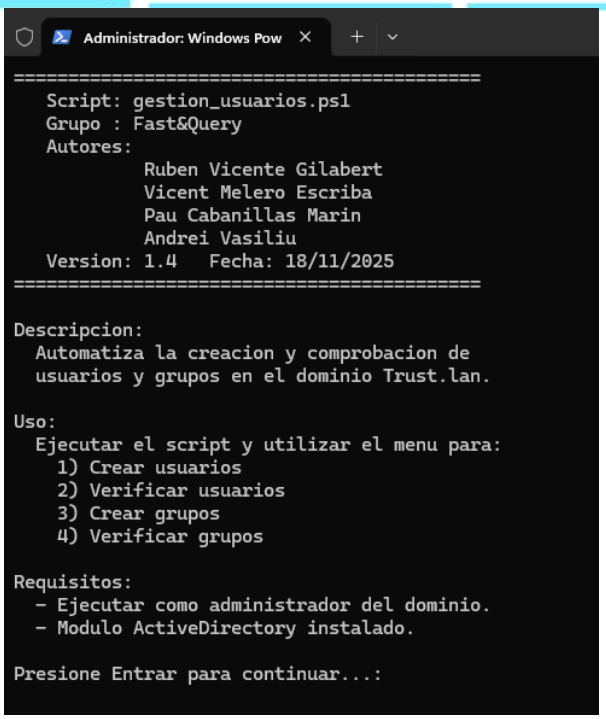
- **Gestion_usuarios.ps1:** https://youtu.be/4tMaOacjgw0
- **Gestion_sistemas.ps1:** https://youtu.be/aoefPlQ33vU
- **Tareas_programadas.ps1:** https://youtu.be/XhusALHD7tY
- **Menu_principal.ps1:** https://youtu.be/PHFk6F1tEQc

2. Crear un script para automatizar la creación de usuarios siguiendo las siguientes indicaciones:

I. Crear una función que muestre la ayuda/información de uso del script. Mostrará información sobre funcionamiento y uso del script, autor, fecha, versión…

La función Mostrar-Ayuda proporciona al administrador toda la información esencial del script en una única pantalla, incluyendo los datos de identificación (nombre del script, empresa, grupo, autores, versión y fecha), una breve descripción técnica de su propósito y funcionalidades, las instrucciones de uso y las opciones disponibles en su menú, así como los requisitos previos necesarios para su correcta ejecución, como disponer de permisos de administrador, tener instalados los módulos requeridos y ubicar los scripts en la carpeta C:\Scripts.

```
#------------------------------------------------------------
# Funcion: Mostrar-Ayuda
# Objetivo: Mostrar informacion del script, autor, uso, etc.
#------------------------------------------------------------
function Mostrar-Ayuda {
    Clear-Host
    Write-Host "========================================="
    Write-Host "   Script: gestion_usuarios.ps1"
    Write-Host "   Grupo : Fast&Query"
    Write-Host "   Autores: "
    Write-Host "            Ruben Vicente Gilabert"
    Write-Host "            Vicent Melero Escriba"
    Write-Host "            Pau Cabanillas Marin"
    Write-Host "            Andrei Vasiliu"
    Write-Host "   Version: 1.4   Fecha: 18/11/2025"
    Write-Host "========================================="
    Write-Host ""
    Write-Host "Descripcion:"
    Write-Host "  Automatiza la creacion y comprobacion de"
    Write-Host "  usuarios y grupos en el dominio Trust.lan."
    Write-Host ""
    Write-Host "Uso:"
    Write-Host "  Ejecutar el script y utilizar el menu para:"
    Write-Host "    1) Crear usuarios"
    Write-Host "    2) Verificar usuarios"
    Write-Host "    3) Crear grupos"
    Write-Host "    4) Verificar grupos"
    Write-Host ""
    Write-Host "Requisitos:"
    Write-Host "  - Ejecutar como administrador del dominio."
    Write-Host "  - Modulo ActiveDirectory instalado."
    Write-Host ""
    Pause
}
```



II. Crear una función que verifica si un usuario existe.

La función **Verificar-Usuario** sirve para comprobar si un usuario ya existe en el dominio usando su **SamAccountName**. Lo que hace es crear un filtro de búsqueda, consultar Active Directory con **Get-ADUser** y revisar si la respuesta es válida o está vacía.

Según el resultado, muestra un mensaje indicando si el usuario existe o no, usando colores para que sea más fácil de ver, y devuelve **$true** o **$false**. Gracias a esto, el resto del script puede decidir si continuar o no (por ejemplo, evitar crear un usuario que ya está registrado). Esta función es clave para evitar errores y trabajar siempre con cuentas correctas.

```powershell
#------------------------------------------------------------
# Funcion: Verificar-Usuario
# Objetivo: Comprobar si un usuario existe en el dominio
#------------------------------------------------------------
function Verificar-Usuario {
    param(
        [Parameter(Mandatory = $true)]
        [string]$SamAccountName
    )

    $filtro  = "SamAccountName -eq '$SamAccountName'"
    $usuario = Get-ADUser -Filter $filtro -ErrorAction SilentlyContinue

    if ($null -ne $usuario) {
        Write-Host "El usuario '$SamAccountName' EXISTE en el dominio." -ForegroundColor Green
        return $true
    } else {
        Write-Host "El usuario '$SamAccountName' NO existe en el dominio." -ForegroundColor Yellow
        return $false
    }
}
```

```
==========================================
   GESTION DE USUARIOS Y GRUPOS - TRUST.LAN
==========================================
1. Mostrar ayuda del script
2. Crear nuevo usuario
3. Verificar si un usuario existe
4. Crear nuevo grupo
5. Verificar si un grupo existe
0. Salir
------------------------------------------
Seleccione una opcion: 3
Introduce el samAccountName del usuario: ruben.vicente
El usuario 'ruben.vicente' EXISTE en el dominio.
True
Presione Entrar para continuar...:
```

III. Crear una función que verifica si un grupo existe.

La función sirve para comprobar si un grupo de Active Directory existe en el dominio *Trust.lan* antes de usarlo en otras partes del script, como al añadir usuarios o crear nuevos objetos. Con esto evitamos errores, duplicados o trabajar con grupos que realmente no existen.

Lo que hace es recibir el nombre exacto del grupo y buscarlo con **Get-ADGroup**. Si lo encuentra, muestra un mensaje en verde indicando que el grupo existe y devuelve **$true**. Si no lo encuentra, muestra un aviso en amarillo y devuelve **$false**, lo que permite al script decidir qué hacer (por ejemplo, no dejar crear un usuario si falta el grupo).

Esta comprobación previa es importante porque ayuda a que la automatización sea más fiable y evita fallos a la hora de gestionar usuarios y grupos dentro del dominio.

```powershell
#------------------------------------------------------------
# Funcion: Verificar-Grupo
# Objetivo: Comprobar si un grupo existe en el dominio
#------------------------------------------------------------
function Verificar-Grupo {
    param(
        [Parameter(Mandatory = $true)]
        [string]$NombreGrupo
    )

    $filtro = "Name -eq '$NombreGrupo'"
    $grupo  = Get-ADGroup -Filter $filtro -ErrorAction SilentlyContinue

    if ($null -ne $grupo) {
        Write-Host "El grupo '$NombreGrupo' EXISTE en el dominio." -ForegroundColor Green
        return $true
    } else {
        Write-Host "El grupo '$NombreGrupo' NO existe en el dominio." -ForegroundColor Yellow
        return $false
    }
}
```

```
Seleccione una opcion: 5
Introduce el nombre del grupo: grupo.tic
El grupo 'grupo.tic' EXISTE en el dominio.
True
Presione Entrar para continuar...:
```

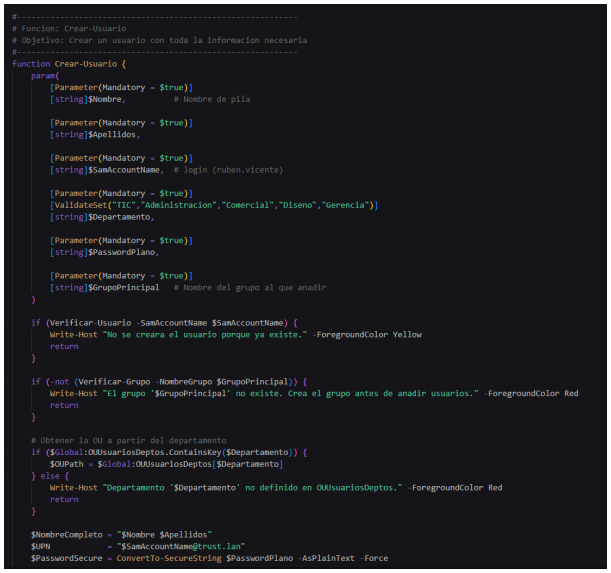
IV. Crear una función que, dados unos parámetros de entrada, cree un usuario. Los parámetros de entrada deben ser válidos y deben contener toda la información relativa a un usuario.

La función **Crear-Usuario** automatiza todo el proceso de dar de alta un nuevo usuario en el dominio *Trust.lan*. Recibe la información básica del usuario (nombre, apellidos, SamAccountName, departamento, contraseña inicial y grupo principal), comprobando que todos los datos necesarios estén completos.

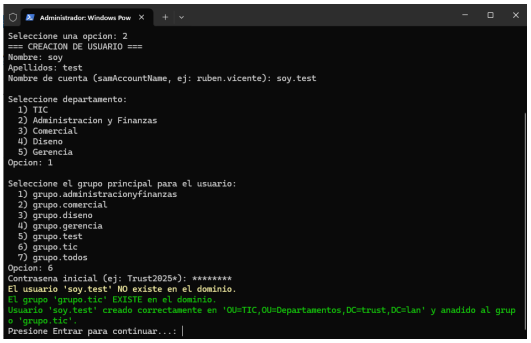
En primer lugar, verifica si el usuario ya existe mediante **Verificar-Usuario**, para evitar duplicados. También comprueba que el grupo principal indicado esté creado en el dominio usando **Verificar-Grupo**; si no existe, el proceso se detiene para evitar errores.

Después, el script elige automáticamente la **ruta LDAP (OU)** correcta según el departamento. Una vez validado todo, genera el nombre completo, el UPN y convierte la contraseña en un formato seguro.

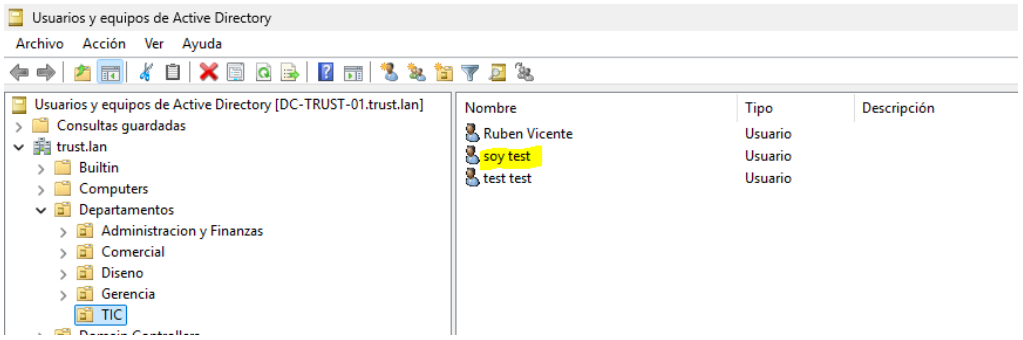
Finalmente, crea el usuario en Active Directory con **New-ADUser**, habilita la cuenta y obliga a cambiar la contraseña en el primer inicio de sesión. Para completar el proceso, añade al usuario al grupo principal con **Add-ADGroupMember**.



```
#------------------------------------------------------------
# Funcion: Crear-Usuario
# Objetivo: Crear un usuario con toda la informacion necesaria
#------------------------------------------------------------
function Crear-Usuario {
    param(
        [Parameter(Mandatory = $true)]
        [string]$Nombre,          # Nombre de pila

        [Parameter(Mandatory = $true)]
        [string]$Apellidos,

        [Parameter(Mandatory = $true)]
        [string]$SamAccountName,  # login (ruben.vicente)

        [Parameter(Mandatory = $true)]
        [ValidateSet("TIC","Administracion","Comercial","Diseno","Gerencia")]
        [string]$Departamento,

        [Parameter(Mandatory = $true)]
        [string]$PasswordPlano,

        [Parameter(Mandatory = $true)]
        [string]$GrupoPrincipal   # Nombre del grupo al que anadir
    )

    if (Verificar-Usuario -SamAccountName $SamAccountName) {
        Write-Host "No se creara el usuario porque ya existe." -ForegroundColor Yellow
        return
    }

    if (-not (Verificar-Grupo -NombreGrupo $GrupoPrincipal)) {
        Write-Host "El grupo '$GrupoPrincipal' no existe. Crea el grupo antes de anadir usuarios." -ForegroundColor Red
        return
    }

    # Obtener la OU a partir del departamento
    if ($Global:OUUsuariosDeptos.ContainsKey($Departamento)) {
        $OUPath = $Global:OUUsuariosDeptos[$Departamento]
    } else {
        Write-Host "Departamento '$Departamento' no definido en OUUsuariosDeptos." -ForegroundColor Red
        return
    }

    $NombreCompleto = "$Nombre $Apellidos"
    $UPN            = "$SamAccountName@trust.lan"
    $PasswordSecure = ConvertTo-SecureString $PasswordPlano -AsPlainText -Force
```



```
Administrador: Windows Pow
Seleccione una opcion: 2
=== CREACION DE USUARIO ===
Nombre: soy
Apellidos: test
Nombre de cuenta (samAccountName, ej: ruben.vicente): soy.test

Seleccione departamento:
  1) TIC
  2) Administracion y Finanzas
  3) Comercial
  4) Diseno
  5) Gerencia
Opcion: 1

Seleccione el grupo principal para el usuario:
  1) grupo.administracionyfinanzas
  2) grupo.comercial
  3) grupo.diseno
  4) grupo.gerencia
  5) grupo.test
  6) grupo.tic
  7) grupo.todos
Opcion: 6
Contrasena inicial (ej: Trust2025+): ********
El usuario 'soy.test' NO existe en el dominio.
El grupo 'grupo.tic' EXISTE en el dominio.
Usuario 'soy.test' creado correctamente en 'OU=TIC,OU=Departamentos,DC=trust,DC=lan' y anadido al grup
o 'grupo.tic'.
Presione Entrar para continuar...:
```



v. Crear una función que, dados unos parámetros de entrada, cree un grupo. Los parámetros de entrada deben ser válidos y deben contener toda la información relativa al grupo.

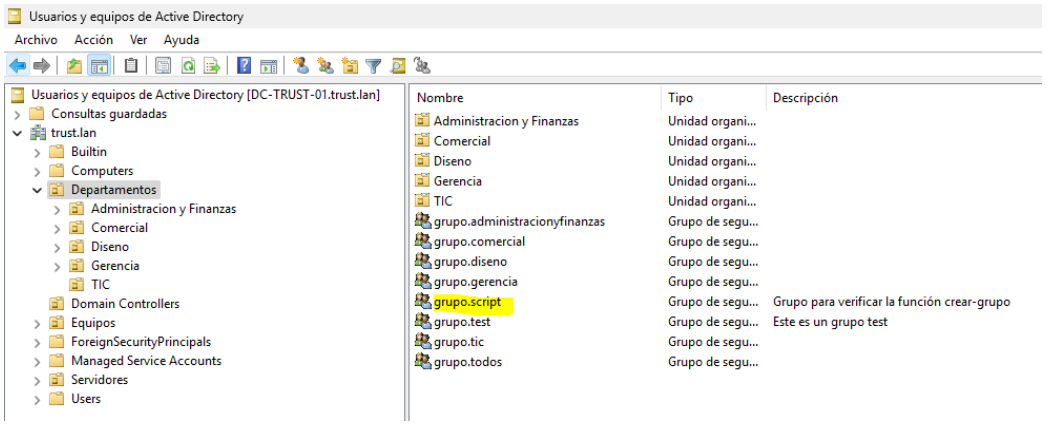
La función **Crear-Grupo** se encarga de automatizar la creación de grupos de seguridad en el dominio *Trust.lan*, asegurando que los parámetros introducidos sean válidos y que el grupo se cree correctamente dentro de la estructura del Active Directory.

Para ello, define varios parámetros obligatorios: el nombre del grupo, su alcance (Global, Universal o DomainLocal), su tipo (Security o Distribution) y una descripción. Gracias a **ValidateSet**, solo se pueden elegir valores correctos, evitando errores de escritura. Antes de crear el grupo, la función llama a **Verificar-Grupo** para comprobar si ya existe; si es así, muestra un aviso y detiene el proceso para evitar duplicados.

Si el grupo no existe, lo crea con **New-ADGroup**, colocándolo automáticamente en la OU definida en **$Global:OUGrupos**. Al finalizar, muestra un mensaje confirmando que el grupo se ha creado correctamente y en qué ubicación del Active Directory se encuentra.

```powershell
#------------------------------------------------------------
# Funcion: Crear-Grupo
# Objetivo: Crear un grupo de seguridad con parametros validos
#------------------------------------------------------------
function Crear-Grupo {
    param(
        [Parameter(Mandatory = $true)]
        [string]$NombreGrupo,

        [Parameter(Mandatory = $true)]
        [ValidateSet("Global","Universal","DomainLocal")]
        [string]$Alcance,

        [Parameter(Mandatory = $true)]
        [ValidateSet("Security","Distribution")]
        [string]$Tipo,

        [Parameter(Mandatory = $true)]
        [string]$Descripcion
    )

    if (Verificar-Grupo -NombreGrupo $NombreGrupo) {
        Write-Host "No se creara el grupo porque ya existe." -ForegroundColor Yellow
        return
    }

    # Ruta donde se crean los grupos
    $RutaOU = $Global:OUGrupos

    New-ADGroup `
        -Name $NombreGrupo `
        -GroupScope $Alcance `
        -GroupCategory $Tipo `
        -Path $RutaOU `
        -Description $Descripcion

    Write-Host "Grupo '$NombreGrupo' creado correctamente en '$RutaOU'." -ForegroundColor Green
}
```

```
Seleccione una opcion: 4
=== CREACION DE GRUPO ===
Nombre del grupo (ej: grupo.comercial): grupo.script
Alcance [Global/Universal/DomainLocal]: DomainLocal
Tipo [Security/Distribution]: Security
Descripcion del grupo: Grupo para verificar la función crear-grupo
El grupo 'grupo.script' NO existe en el dominio.
Grupo 'grupo.script' creado correctamente en 'OU=Departamentos,DC=trust,DC=lan'.
Presione Entrar para continuar...:
```



VI. El script tan solo puede ser ejecutado por el administrador.

Este apartado se encarga de impedir que el script se ejecute sin privilegios, asegurando así la seguridad y el buen funcionamiento del dominio. Para ello, se crea la función **Test-Administrador**, que comprueba si el usuario que está ejecutando el script pertenece al grupo **Administrators**. La función obtiene la identidad del usuario y verifica su rol usando clases de seguridad de .NET.

Si el usuario no tiene permisos administrativos, el script muestra un mensaje en rojo indicando que no cumple los requisitos y se detiene inmediatamente con **exit 1**.

```powershell
#------------------------------------------------------------
# Funcion: Test-Administrador
# Objetivo: Verificar que el script se ejecuta como admin
#------------------------------------------------------------
function Test-Administrador {
    $identity  = [Security.Principal.WindowsIdentity]::GetCurrent()
    $principal = New-Object Security.Principal.WindowsPrincipal($identity)

    if (-not $principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)) {
        Write-Host "Este script debe ejecutarse con privilegios de administrador." -ForegroundColor Red
        Write-Host "Cierra esta ventana y vuelve a abrir PowerShell como 'Ejecutar como administrador'."
        exit 1
    }
}
```

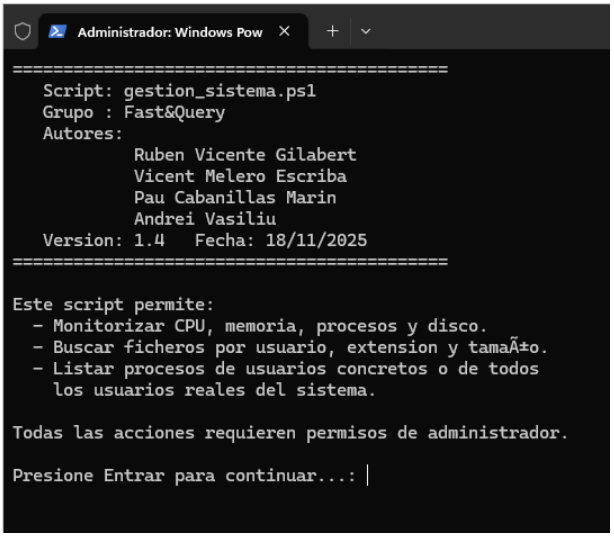
```
PS C:\Scripts> .\gestion_usuarios.ps1
Este script debe ejecutarse con privilegios de administrador.
Cierra esta ventana y vuelve a abrir PowerShell como 'Ejecutar como administrador'.
PS C:\Scripts>
```

3. Crear un script para gestionar diversos aspectos del sistema como son la gestión de procesos, el sistema de ficheros y monitorización del sistema. En este caso se pueden grabar 3 vídeos independientes y/o crear capturas de pantalla de su funcionamiento. Monitorización del sistema:

I. Crear una función que muestre la ayuda/información de uso del script. Mostrará información sobre funcionamiento y uso del script, autor, fecha, versión...

En este script se incluye la función **Mostrar-Ayuda**, que presenta los datos esenciales del script: nombre, autores, versión y fecha, junto con un resumen de sus funcionalidades principales. Ofrece al administrador una referencia rápida sobre qué tareas automatiza el script y cómo debe utilizarse.

```powershell
# Mostrar ayuda / informacion del script
function Mostrar-Ayuda {
    Clear-Host
    Write-Host "============================================"
    Write-Host "   Script: gestion_sistema.ps1"
    Write-Host "   Script: gestion_usuarios.ps1"
    Write-Host "   Grupo : Fast&Query"
    Write-Host "   Autores: "
    Write-Host "            Ruben Vicente Gilabert"
    Write-Host "            Vicent Melero Escriba"
    Write-Host "            Pau Cabanillas Marin"
    Write-Host "            Andrei Vasiliu"
    Write-Host "   Version: 1.4   Fecha: 18/11/2025"
    Write-Host "============================================"
    Write-Host ""
    Write-Host "Este script permite:"
    Write-Host "  - Monitorizar CPU, memoria, procesos y disco."
    Write-Host "  - Buscar ficheros por usuario, extension y tamaño."
    Write-Host "  - Listar procesos de usuarios concretos o de todos"
    Write-Host "    los usuarios reales del sistema."
    Write-Host ""
    Write-Host "Todas las acciones requieren permisos de administrador."
    Write-Host ""
    Pause
}
```

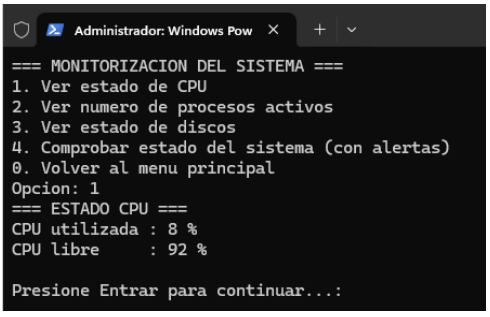


II. Crear una función que muestre sobre el estado del sistema, el porcentaje de cpu utilizada y libre.

Para cumplir con este requisito se creó la función **Obtener-EstadoCPU**, cuya tarea es consultar el estado actual del procesador. Para ello, usa la clase WMI **Win32_Processor** con **Get-CimInstance**, de donde obtiene el valor **LoadPercentage**, que indica el porcentaje de CPU en uso en ese momento.

Con ese dato, la función calcula también la CPU libre restando el porcentaje utilizado a 100%. Por último, muestra ambos valores por pantalla de forma clara y formateada, permitiendo que el administrador vea rápidamente la carga del procesador en tiempo real.

```powershell
#------------------------------------------------------------
# MONITORIZACION DEL SISTEMA
#------------------------------------------------------------

# Estado CPU (utilizada y libre)
function Obtener-EstadoCPU {
    $cpuUsed = (Get-CimInstance Win32_Processor).LoadPercentage
    $cpuFree = 100 - $cpuUsed


    Write-Host "=== ESTADO CPU ==="
    Write-Host ("CPU utilizada : {0} %" -f $cpuUsed)
    Write-Host ("CPU libre     : {0} %" -f $cpuFree)
    Write-Host ""
}
```



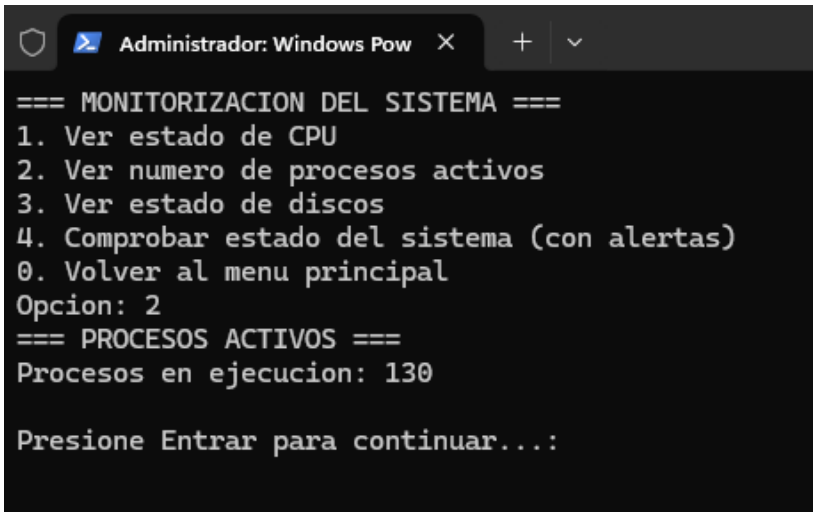
III. Crear una función que muestre sobre el estado del sistema, la cantidad de procesos activos en el sistema.

Para este apartado se creó la función **Obtener-NumeroProcesos**, cuyo objetivo es mostrar cuántos procesos están activos en el sistema en el momento de la consulta. La función utiliza **Get-Process**, que devuelve todos los procesos en ejecución, y obtiene su cantidad mediante la propiedad **.Count**.

Finalmente, muestra esta información por pantalla de forma clara, permitiendo al administrador conocer de un vistazo la carga general del sistema en cuanto a procesos activos. Estos datos pueden ayudar a detectar saturaciones, servicios bloqueados o comportamientos anómalos.

.

```powershell
# Cantidad de procesos activos
function Obtener-NumeroProcesos {
    $total = (Get-Process).Count
    Write-Host "=== PROCESOS ACTIVOS ==="
    Write-Host "Procesos en ejecucion: $total"
    Write-Host ""
}
```

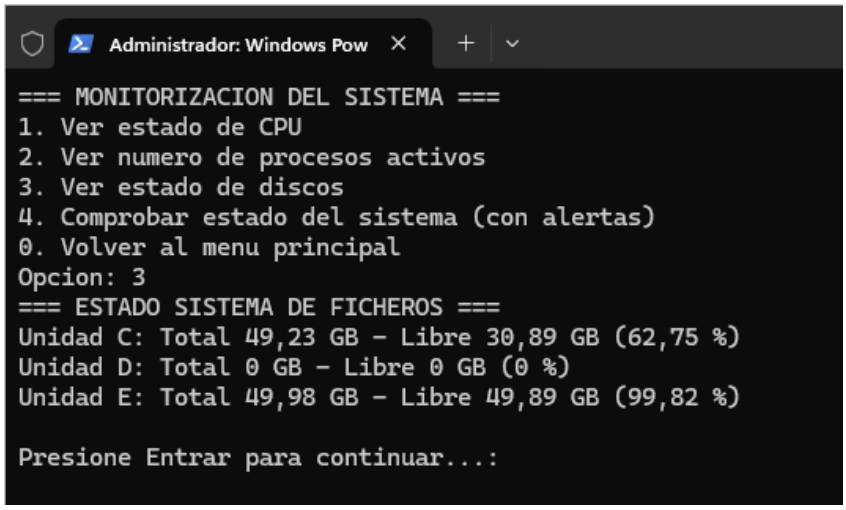


IV. Crear una función que muestre sobre el estado del sistema, el espacio total y libre del sistema de ficheros.

Para este apartado se implementó la función **Obtener-EstadoDiscos**, encargada de mostrar el estado de cada unidad del sistema de archivos. La función usa **Get-PSDrive** para obtener los discos gestionados por el proveedor *FileSystem* y calcula el espacio total y el espacio libre en gigabytes. Con esos datos también se obtiene el porcentaje de espacio disponible.

Toda la información se muestra en pantalla de forma clara y bien formateada, lo que permite ver rápidamente si algún disco está cerca de quedarse sin espacio.

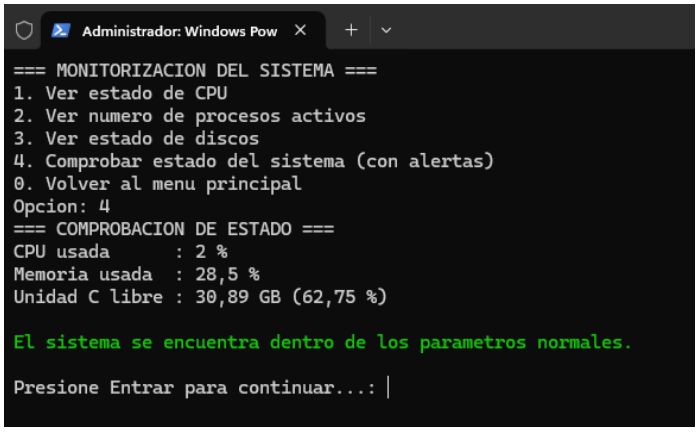
```powershell
# Espacio total y libre del sistema de ficheros (por unidad)
function Obtener-EstadoDiscos {
    Write-Host "=== ESTADO SISTEMA DE FICHEROS ==="
    Get-PSDrive -PSProvider FileSystem | ForEach-Object {
        $totalGB  = [math]::Round(($_.Used + $_.Free)/1GB,2)
        $freeGB   = [math]::Round($_.Free/1GB,2)
        $porcFree = if ($totalGB -ne 0) { [math]::Round(($freeGB/$totalGB)*100,2) } else { 0 }
        Write-Host ("Unidad {0}: Total {1} GB - Libre {2} GB ({3} %)" -f $_.Name,$totalGB,$freeGB,$porcFree)
    }
    Write-Host ""
}
```



v. Crear una función que compruebe el estado del sistema. La función debe registrar los sucesos en el fichero alerta.log y ha de mostrar un aviso. Los avisos/registros deben producirse cuando:

Para este apartado se creó la función **Comprobar-EstadoSistema**, cuyo objetivo es revisar los parámetros críticos del servidor —CPU, memoria y almacenamiento— y generar alertas tanto en pantalla como en el archivo **alerta.log** cuando alguno supera valores peligrosos. La función obtiene en tiempo real el porcentaje de memoria usada, el uso de CPU y el espacio libre de la unidad **C:**, aplicando los umbrales establecidos: más del **80 %** de memoria usada, más del **85 %** de CPU o menos del **15 %** de espacio libre.

Si alguno de estos valores supera el límite, la función genera un mensaje de alerta, lo muestra claramente al administrador y lo registra en el log usando **Escribir-Alerta**. Si todo está dentro de los rangos normales, informa de que el sistema funciona correctamente.



1. La cantidad de memoria usada supere el 80%.

```powershell
# Umbrales
if ($usedMemPerc -gt 80) {
    $msg = "ALERTA: Memoria usada $usedMemPerc % (>80%)"
    Write-Host "ADVERTENCIA: $msg" -ForegroundColor Yellow
    Escribir-Alerta $msg
}
```

2. El porcentaje de uso de la cpu supere el 85%.

```powershell
if ($cpuUsed -gt 85) {
    $msg = "ALERTA: CPU usada $cpuUsed % (>85%)"
    Write-Host "ADVERTENCIA: $msg" -ForegroundColor Yellow
    Escribir-Alerta $msg
}
```

3. El espacio libre sea inferior al 15%.

```powershell
if ($freePerc -lt 15) {
    $msg = "ALERTA: Espacio libre en C: $freePerc % (<15%)"
    Write-Host "ADVERTENCIA: $msg" -ForegroundColor Yellow
    Escribir-Alerta $msg
}
```

El sistema de ficheros:

VI. Crear una función que localice todos los ficheros de un usuario en una ruta especificada como parámetro de entrada y lo muestre por pantalla.

Para cumplir este requisito se creó la función **Buscar-FicherosUsuario**, que sirve para localizar todos los archivos que pertenecen a un usuario concreto dentro de una ruta indicada por el administrador. La función recibe dos parámetros obligatorios: el nombre del usuario (en formato *dominio\usuario* o solo el *SamAccountName*) y la ruta donde se quiere buscar.

Con esa información, recorre el directorio de forma recursiva usando **Get-ChildItem**, analizando solo archivos y excluyendo carpetas. Para comprobar si un archivo pertenece al usuario, obtiene su ACL y compara el **Owner** con el usuario indicado. Además, incluye control de errores por fichero para evitar que el script se detenga si algún archivo no tiene permisos o es inaccesible.

Todos los archivos encontrados se muestran en una tabla formateada con la ruta completa, el tamaño y el propietario real.

```powershell
# Localizar ficheros de un usuario en una ruta
function Buscar-FicherosUsuario {
    param(
        [Parameter(Mandatory = $true)]
        [string]$Usuario,   # dominio\usuario o solo usuario
        [Parameter(Mandatory = $true)]
        [string]$Ruta
    )

    Write-Host "Buscando ficheros propiedad de '$Usuario' en $Ruta..." -ForegroundColor Cyan
    Get-ChildItem -Path $Ruta -Recurse -ErrorAction SilentlyContinue |
        Where-Object {
            try {
                ($_.PSIsContainer -eq $false) -and
                ($_.GetAccessControl().Owner -like "*$Usuario")
            } catch {
                $false
            }
        } |
        Select-Object FullName,
                      Length,
                      @{Name="Owner";Expression={$_.GetAccessControl().Owner}} |
        Format-Table -AutoSize

    Write-Host ""
    Pause
}
```

```
Administrador: Windows Pow
=== SISTEMA DE FICHEROS ===
1. Buscar ficheros por propietario (usuario)
2. Buscar ficheros por extension y tamano minimo
0. Volver al menu principal
Opcion: 1
Introduce el usuario (ej: TRUST\ruben.vicente): ruben.vicente
Ruta base (ej: C:\Usuarios\): C:\Scripts
Buscando ficheros propiedad de 'ruben.vicente' en C:\Scripts...

FullName                                  Length Owner
--------                                  ------ -----
C:\Scripts\archivo propiedad de ruben.txt     46 TRUST\ruben.vicente



Presione Entrar para continuar...:
```

VII. Crear una función que localice todos los ficheros de un tipo/extensión (puede ser una o más) y un tamaño especificado como parámetro de entrada y lo muestre por pantalla.

La función **Buscar-FicherosTipo** permite localizar archivos según su extensión y, de forma opcional, aplicando también un tamaño mínimo. El administrador puede indicar una o varias extensiones (por ejemplo: *log*, *txt*, *docx*), y la función las normaliza automáticamente para evitar errores. Además, valida el parámetro de tamaño: si el valor es numérico, lo convierte a MB para usarlo como filtro; si está vacío o es incorrecto, la búsqueda se hace solo por extensión y se muestra un aviso.

Una vez validados los parámetros, la función recorre la ruta indicada de forma recursiva y selecciona los archivos que cumplen las condiciones. Finalmente, muestra los resultados en una tabla con la ruta completa y el tamaño en MB, lo que facilita identificar archivos relevantes, pesados o que requieren revisión.

```powershell
# Localizar ficheros por extension y tamaño minimo
function Buscar-FicherosTipo {
    param(
        [Parameter(Mandatory = $true)]
        [string[]]$Extensiones,   # ej: "log","txt","docx"

        [Parameter(Mandatory = $true)]
        [string]$TamanoMinMB,     # cadena tal cual viene de Read-Host (puede estar vacia)

        [Parameter(Mandatory = $true)]
        [string]$Ruta
    )

    # Normalizar extensiones: "txt" -> ".txt"
    $extsNormalizadas = $Extensiones | ForEach-Object {
        $e = $_.Trim()
        if ($e.StartsWith(".")) { $e.ToLower() } else { (".$e").ToLower() }
    }

    if (-not [string]::IsNullOrWhiteSpace($TamanoMinMB)) {
        # Intentamos convertir a entero
        $minInt = 0
        if ([int]::TryParse($TamanoMinMB, [ref]$minInt)) {
            $usarTamano = $true
            $minBytes   = $minInt * 1MB
        } else {
            Write-Host "Valor de tamaño minimo no valido. Se ignorara el filtro de tamano." -ForegroundColor Yellow
        }
    }
```

```
Administrador: Windows Pow
=== SISTEMA DE FICHEROS ===
1. Buscar ficheros por propietario (usuario)
2. Buscar ficheros por extension y tamano minimo
0. Volver al menu principal
Opcion: 2
Extensiones separadas por coma (ej: log,txt,docx): txt,ps1
Tamano minimo en MB: 0
Ruta base (ej: C:\): C:\Scripts
Buscando ficheros en C:\Scripts con extensiones .txt, .ps1 y tamano >= 0 MB...

FullName                                TamanoMB
--------                                --------
C:\Scripts\archivo propiedad de ruben.txt        0
C:\Scripts\gestion_sistema.ps1                0,02
C:\Scripts\gestion_usuarios.ps1               0,01
C:\Scripts\menu_principal.ps1                    0
C:\Scripts\tareas_programadas.ps1             0,01


Presione Entrar para continuar...:
```

Gestión de procesos:

VIII. Crear una función que muestre los procesos de dos usuarios en concreto del sistema.

La función **Mostrar-ProcesosDosUsuarios** permite consultar los procesos que están ejecutando dos usuarios concretos del sistema. Para ello, recibe como parámetros los nombres de usuario en formato *DOMINIO\usuario* y usa **Get-Process -IncludeUserName** para obtener solo los procesos cuyo propietario coincide con alguno de los dos usuarios. El filtrado se realiza comparando el valor **UserName** con ambos nombres proporcionados.

Después, la función muestra los procesos encontrados en una tabla ordenada con información como el usuario, el ID del proceso, el nombre de la aplicación, el tiempo de CPU utilizado y la memoria asignada.

```powershell
# Mostrar procesos de dos usuarios concretos
function Mostrar-ProcesosDosUsuarios {
    param(
        [Parameter(Mandatory = $true)]
        [string]$Usuario1,   # ej: "TRUST\ruben.vicente"
        [Parameter(Mandatory = $true)]
        [string]$Usuario2
    )

    Write-Host "=== PROCESOS DE $Usuario1 Y $Usuario2 ==="
    Get-Process -IncludeUserName -ErrorAction SilentlyContinue |
        Where-Object { $_.UserName -eq $Usuario1 -or $_.UserName -eq $Usuario2 } |
        Select-Object UserName, Id, ProcessName, CPU, PM |
        Format-Table -AutoSize

    Write-Host ""
    Pause
}
```

```
=== GESTION DE PROCESOS ===
1. Mostrar procesos de dos usuarios concretos
2. Mostrar procesos de todos los usuarios reales
0. Volver al menu principal
Opcion: 1
Usuario 1 (ej: TRUST\ruben.vicente): TRUST\ruben.vicente
Usuario 2 (ej: TRUST\benjamin.placeta): TRUST\Administrador
=== PROCESOS DE TRUST\ruben.vicente Y TRUST\Administrador ===

UserName               Id ProcessName                CPU        PM
--------               -- -----------                ---        --
TRUST\ruben.vicente  3588 AzureArcSysTray        0,109375   3801088
TRUST\ruben.vicente  5888 backgroundTaskHost       0,3125   5279744
TRUST\ruben.vicente  6440 backgroundTaskHost      0,15625   5603328
TRUST\ruben.vicente  1856 conhost                 0,390625   8441856
TRUST\ruben.vicente  6972 ctfmon                      0,25   5267456
TRUST\ruben.vicente  8136 dllhost                 0,15625   5632000
TRUST\ruben.vicente  1800 explorer                 7,5625  48427008
TRUST\ruben.vicente  5340 powershell              1,390625  64241664
TRUST\ruben.vicente  3700 RuntimeBroker            1,21875   9793536
TRUST\ruben.vicente  4228 RuntimeBroker            0,03125   1667072
TRUST\ruben.vicente  6104 RuntimeBroker           0,203125   5988352
TRUST\ruben.vicente  5116 SearchHost                5,1875  96124928
TRUST\ruben.vicente  6032 ServerManager            4,765625 111869952
```

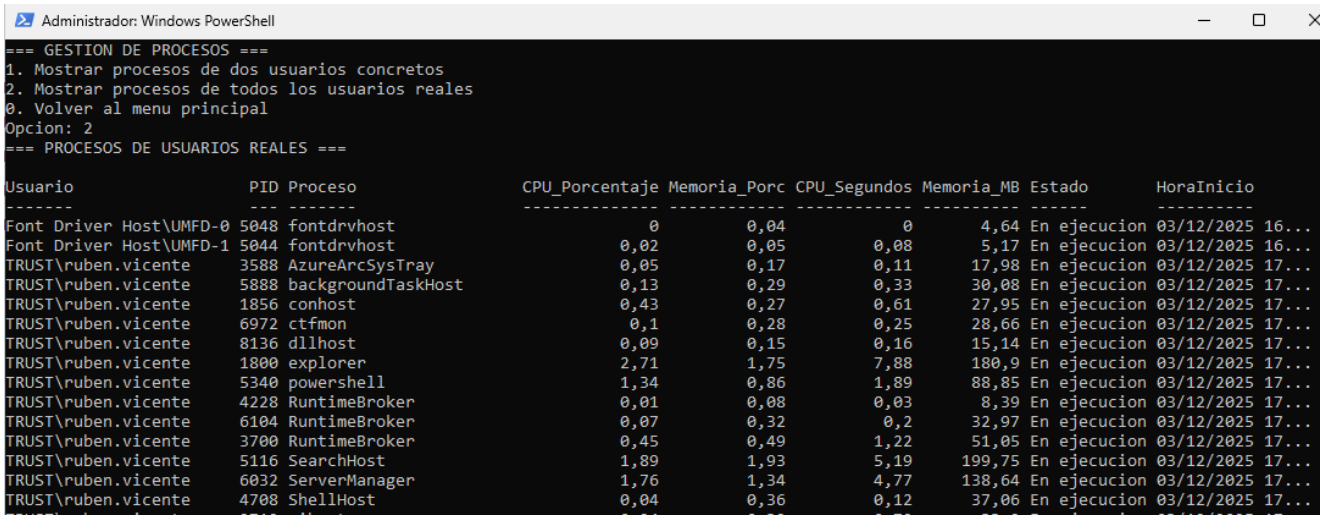
IX. Crear una función que muestre todos los procesos de los usuarios reales del sistema. Se ha de mostrar: el usuario, el identificador del proceso, el porcentaje de cpu utilizado, el porcentaje de memoria utilizado, estado, hora de inicio, tiempo de ejecución y que comando lo inició.

La función **Mostrar-ProcesosUsuariosReales** permite obtener un listado completo de todos los procesos ejecutados por usuarios reales del sistema, excluyendo automáticamente cuentas como **SYSTEM**, **SERVICIO LOCAL**, **SERVICIO DE RED** y otras cuentas internas. Para cada proceso, la función muestra información detallada: usuario propietario, ID del proceso, uso aproximado de CPU, porcentaje de memoria utilizada, estado, hora de inicio, tiempo total de ejecución y el comando o ejecutable que lo lanzó.

Para recopilar estos datos, combina **Get-Process -IncludeUserName** con consultas adicionales a WMI/CIM para calcular los porcentajes de memoria y CPU. También usa bloques **try/catch** para evitar fallos cuando algún proceso no permite obtener ciertos datos, asegurando que el script siga funcionando.

Por último, toda la información se presenta en una tabla ordenada, lo que ofrece al administrador una visión clara del uso de recursos por usuario y facilita tareas de auditoría, diagnóstico de problemas y monitorización del sistema.

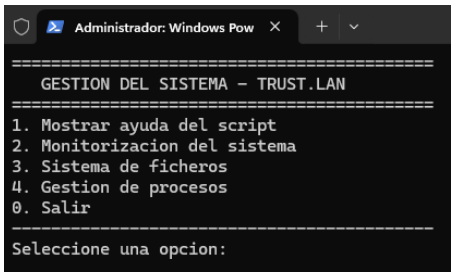
```powershell
# Mostrar todos los procesos de usuarios reales
function Mostrar-ProcesosUsuariosReales {

    Write-Host "=== PROCESOS DE USUARIOS REALES ==="

    $os = Get-CimInstance Win32_OperatingSystem
    $totalMemBytes = [double]$os.TotalVisibleMemorySize * 1KB

    Get-Process -IncludeUserName -ErrorAction SilentlyContinue |
        Where-Object {
            $_.UserName -and
            ($_.UserName -notlike "*SYSTEM*") -and
            ($_.UserName -notlike "*SERVICIO*") -and
            ($_.UserName -notlike "*SERVICE*")
        } |
        ForEach-Object {
            $proc    = $_
            $cpuSeg = $proc.CPU
            $mem     = $proc.WorkingSet64

            $inicio = $null
            $estado = "Desconocido"
            $comando = $null

            try { $inicio = $proc.StartTime } catch {}
            try { $comando = $proc.Path } catch {}

            if ($proc.Responding) { $estado = "En ejecucion" }

            # Porcentaje aproximado de memoria
            $memPerc = if ($totalMemBytes -gt 0) { [math]::Round(($mem / $totalMemBytes) * 100, 2) } else { 0 }

            # Porcentaje aproximado de CPU (tiempo de CPU / tiempo vivo)
            $cpuPerc = 0
            if ($inicio) {
                $segundosVida = (New-TimeSpan -Start $inicio -End (Get-Date)).TotalSeconds
                if ($segundosVida -gt 0) {
                    $cpuPerc = [math]::Round(($cpuSeg / $segundosVida) * 100, 2)
                }
            }
```



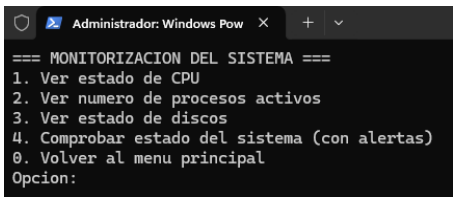
General:

x. Crear un menú mediante el cual el administrador del sistema pueda ejecutar cualquiera de las opciones definidas en el script.

Para facilitar el uso del script por parte del administrador, se incluye un **menú interactivo** que permite acceder de forma ordenada a todas las funciones de monitorización, sistema de ficheros y gestión de procesos. Este menú funciona con un bucle **do…while**, que mantiene la navegación activa hasta que el administrador decide salir, evitando así tener que ejecutar cada función manualmente desde la consola.



Cada submenú (monitorización, ficheros y procesos) muestra una lista numerada con las opciones disponibles, cada una vinculada directamente a las funciones creadas anteriormente. Gracias a la estructura **switch**, el administrador puede elegir una opción y ejecutar al instante la tarea correspondiente, como comprobar el estado del sistema, buscar archivos o listar procesos de usuarios.



```
#------------------------------------------------------------
# MENUS INTERACTIVOS
#------------------------------------------------------------

function Menu-Monitoreo {
    do {
        Clear-Host
        Write-Host "=== MONITORIZACION DEL SISTEMA ==="
        Write-Host "1. Ver estado de CPU"
        Write-Host "2. Ver numero de procesos activos"
        Write-Host "3. Ver estado de discos"
        Write-Host "4. Comprobar estado del sistema (con alertas)"
        Write-Host "0. Volver al menu principal"
        $op = Read-Host "Opcion"

        switch ($op) {
            "1" { Obtener-EstadoCPU; Pause }
            "2" { Obtener-NumeroProcesos; Pause }
            "3" { Obtener-EstadoDiscos; Pause }
            "4" { Comprobar-EstadoSistema }
            "0" { }
            default { Write-Host "Opcion no valida."; Pause }
        }
    } while ($op -ne "0")
}

function Menu-Ficheros {
    do {
        Clear-Host
        Write-Host "=== SISTEMA DE FICHEROS ==="
        Write-Host "1. Buscar ficheros por propietario (usuario)"
        Write-Host "2. Buscar ficheros por extension y tamano minimo"
        Write-Host "0. Volver al menu principal"
        $op = Read-Host "Opcion"

        switch ($op) {
            "1" {
                $usuario = Read-Host "Introduce el usuario (ej: TRUST\ruben.vicente)"
                $ruta    = Read-Host "Ruta base (ej: C:\Usuarios\)"
                Buscar-FicherosUsuario -Usuario $usuario -Ruta $ruta
            }
            "2" {
                $exts  = Read-Host "Extensiones separadas por coma (ej: log,txt,docx)"
                $minMB = Read-Host "Tamano minimo en MB"
                $ruta  = Read-Host "Ruta base (ej: C:\)"
                $listaExt = $exts.Split(",") | ForEach-Object { $_.Trim() }
```

XI. El script tan solo puede ser ejecutado por el administrador o un usuario con privilegios.

Para garantizar la seguridad del sistema y evitar que usuarios sin permisos realicen tareas críticas, el script incluye la función **Test-Administrador**, encargada de comprobar si la ejecución se está realizando con privilegios elevados. Esta función obtiene la identidad del usuario actual y verifica si pertenece al grupo de administradores de Windows.

Si el usuario no tiene los permisos necesarios, el script muestra un aviso indicando que la operación no es válida y se detiene de inmediato. De esta manera se evita que cuentas estándar puedan crear usuarios, modificar grupos, consultar procesos sensibles o generar tareas programadas.

```powershell
# Verificar que el script se ejecuta como administrador
function Test-Administrador {
    $identity  = [Security.Principal.WindowsIdentity]::GetCurrent()
    $principal = New-Object Security.Principal.WindowsPrincipal($identity)

    if (-not $principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)) {
        Write-Host "Este script debe ejecutarse con privilegios de administrador." -ForegroundColor Red
        Write-Host "Cierra esta ventana y vuelve a abrir PowerShell como 'Ejecutar como administrador'."
        exit 1
    }
}

Test-Administrador
```

```
PS C:\Scripts> .\gestion_sistema.ps1
Este script debe ejecutarse con privilegios de administrador.
Cierra esta ventana y vuelve a abrir PowerShell como 'Ejecutar como administrador'.
PS C:\Scripts>
```

4. **Crear un script para crear tareas programadas en el sistema. El script llamará a los scripts creados anteriormente pasándoles una serie de parámetros de entrada:**
    I. Crear un menú mediante el cual el administrador del sistema pueda seleccionar que tipo de tarea quiere automatizar. Según la opción seleccionada, se necesitará pasar una serie de parámetros u otros, estos se deben pedir al usuario. Las tareas pueden ser:

        1. Comprobar el estado del sistema con registro de sucesos.

Para automatizar la supervisión del servidor, se creó una **tarea programada** que ejecuta cada día la función **Comprobar-EstadoSistema** del script gestion_sistema.ps1. Esta función revisa los parámetros críticos del sistema (CPU, memoria y espacio en disco) y registra en el archivo alerta.log cualquier situación que supere los umbrales definidos.

Para ello, en el script tareas_programadas.ps1 se ha desarrollado la función **Crear-Tarea-EstadoSistema**, que pide al administrador el nombre de la tarea y la hora a la que se quiere ejecutar. Después, crea una acción de PowerShell que lanza el script de monitorización con privilegios elevados y deja todo registrado automáticamente. Al finalizar, se muestra un mensaje en pantalla indicando que la tarea se ha creado correctamente.

```powershell
#------------------------------------------------------------
# Crear tarea: Comprobar estado del sistema
#   Llama a gestion_sistema.ps1 -Operacion Estado
#------------------------------------------------------------
function Crear-Tarea-EstadoSistema {

    Write-Host "=== CREAR TAREA: COMPROBAR ESTADO DEL SISTEMA ==="

    $nombreTarea = Read-Host "Nombre de la tarea (ej: ComprobarEstadoDiario)"
    $hora        = Read-Host "Hora de ejecucion diaria (HH:MM, ej: 09:00)"

    # Argumentos: ejecutar el script gestion_sistema.ps1 con la operacion Estado
    $argumentos = "-NoProfile -ExecutionPolicy Bypass -File `"$Global:RutaGestionSistema`" -Operacion Estado"

    $accion  = New-ScheduledTaskAction -Execute $Global:PowerShellExe -Argument $argumentos
    $trigger = New-ScheduledTaskTrigger -Daily -At $hora

    Register-ScheduledTask `
        -TaskName $nombreTarea `
        -Action $accion `
        -Trigger $trigger `
        -Description "Comprueba el estado del sistema y registra alertas en alerta.log" `
        -User "SYSTEM" `
        -RunLevel Highest

    Write-Host "Tarea '$nombreTarea' creada correctamente." -ForegroundColor Green
    Pause
}
```

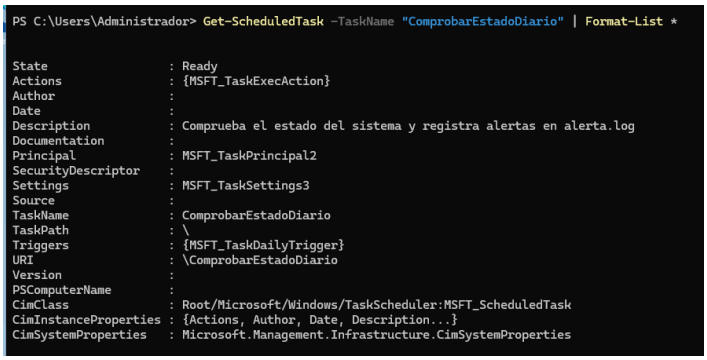
Tras la creación de la tarea, se utiliza el comando:

➢ Get-ScheduledTask -TaskName "ComprobarEstadoDiario" | Format-List *

Para mostrar todos los detalles de la tarea programada. En la salida se pueden observar elementos clave como:

- **Nombre de la tarea**: ComprobarEstadoDiario
- **Estado**: Ready (lista para ejecutarse)
- **Descripción**: “Comprueba el estado del sistema y registra alertas en alerta.log”
- **Trigger**: ejecución diaria a la hora especificada por el administrador
- **Acción**: lanzamiento del script **gestion_sistema.ps1**
- **RunLevel**: Highest (se ejecuta con privilegios de administrador)

Esta comprobación confirma que la automatización está correctamente configurada y lista para ejecutarse.



2. Buscar ficheros de uno o varios usuarios en su carpeta personal y de una extensión específica.

Para automatizar la búsqueda de ficheros por extensión, se creó la función **Crear-Tarea-BuscarFicheros** dentro del script tareas_programadas.ps1. Esta función permite al administrador programar una tarea que ejecuta automáticamente el módulo de búsqueda del script gestion_sistema.ps1, pasándole los parámetros necesarios (ruta base, extensiones y tamaño mínimo).

El sistema pide al administrador el nombre de la tarea, la hora de ejecución, el directorio donde buscar y los tipos de archivo. Con esos datos, crea una tarea programada que se ejecutará todos los días a la hora indicada, realizando la búsqueda de forma automática sin intervención manual.

El script registra la acción mediante un **ScheduledTaskAction** que ejecuta PowerShell con los parámetros necesarios, permitiendo que la búsqueda se realice de forma automática sin intervención del administrador.

```powershell
#------------------------------------------------------------
# Crear tarea: Buscar ficheros por extension
#   Llama a gestion_sistema.ps1 -Operacion BuscarFicheros
#   Pasando ruta, extensiones y tamaño minimo
#------------------------------------------------------------
function Crear-Tarea-BuscarFicheros {

    Write-Host "=== CREAR TAREA: BUSCAR FICHEROS POR EXTENSION ==="

    $nombreTarea = Read-Host "Nombre de la tarea (ej: BuscarLogsUsuarios)"
    $hora        = Read-Host "Hora de ejecucion diaria (HH:MM, ej: 23:00)"
    $rutaBase    = Read-Host "Ruta base donde buscar (ej: C:\Usuarios\)"
    $exts        = Read-Host "Extensiones separadas por coma (ej: log,txt)"
    $tamanoMin   = Read-Host "Tamano minimo en MB (dejar vacio para sin filtro)"

    # Construir argumentos para el script gestion_sistema.ps1
    # Ejemplo final:
    #   -File "C:\Scripts\gestion_sistema.ps1" -Operacion BuscarFicheros -Ruta "C:\Usuarios" -Ext "log,txt" -MinMB 10
    $argumentos = "-NoProfile -ExecutionPolicy Bypass -File `"$Global:RutaGestionSistema`" -Operacion BuscarFicheros -Ruta `"$rutaBase`" -Ext `"$exts`""

    if (-not [string]::IsNullOrWhiteSpace($tamanoMin)) {
        $argumentos += " -MinMB $tamanoMin"
    }

    $accion  = New-ScheduledTaskAction -Execute $Global:PowerShellExe -Argument $argumentos
    $trigger = New-ScheduledTaskTrigger -Daily -At $hora

    Register-ScheduledTask `
        -TaskName $nombreTarea `
        -Action $accion `
        -Trigger $trigger `
        -Description "Busca ficheros por extension y tamaño minimo en $rutaBase" `
        -User "SYSTEM" `
        -RunLevel Highest

    Write-Host "Tarea '$nombreTarea' creada correctamente." -ForegroundColor Green
    Pause
}
```

Tras crear la tarea, se verifica su correcta configuración utilizando el comando correspondiente.

```
PS C:\Users\Administrador> Get-ScheduledTask -TaskName "BuscarLogsUsuarios" | Format-List *


State                 : Ready
Actions               : {MSFT_TaskExecAction}
Author                :
Date                  :
Description           : Busca ficheros por extension y tamaÃ±o minimo en C:\Scripts
Documentation         :
Principal             : MSFT_TaskPrincipal2
SecurityDescriptor    :
Settings              : MSFT_TaskSettings3
Source                :
TaskName              : BuscarLogsUsuarios
TaskPath              : \
Triggers              : {MSFT_TaskDailyTrigger}
URI                   : \BuscarLogsUsuarios
Version               :
PSComputerName        :
CimClass              : Root/Microsoft/Windows/TaskScheduler:MSFT_ScheduledTask
CimInstanceProperties : {Actions, Author, Date, Description...}
CimSystemProperties   : Microsoft.Management.Infrastructure.CimSystemProperties
```

La salida confirma que la tarea está lista para ejecutarse y muestra información como:

- **TaskName:** BuscarLogsUsuarios
- **Estado:** Ready
- **Descripción:** “Busca ficheros por extensión y tamaño mínimo”
- **Trigger diario** configurado a la hora indicada
- **Acción** que ejecuta gestion_sistema.ps1 con los parámetros seleccionados
- **RunLevel Highest**, lo que garantiza que se ejecuta con privilegios administrativos

Gracias a esta configuración, la búsqueda periódica de archivos relevantes se realiza de forma automática, cumpliendo los requisitos del apartado y mejorando el mantenimiento general del sistema.

II. Crear una función que muestre la ayuda/información de uso del script. Mostrará información sobre funcionamiento y uso del script, autor, fecha, versión...

Dentro del script de tareas programadas se ha incluido la función **Mostrar-Ayuda**, cuyo objetivo es ofrecer al administrador una visión general del funcionamiento del módulo. Esta función muestra información básica como el nombre del script, el grupo de trabajo del proyecto, los autores, la versión y la fecha de creación.

Además, explica qué tareas pueden automatizarse desde este script —la comprobación del estado del sistema y la búsqueda automática de ficheros— e indica los requisitos necesarios para su uso, como ejecutar PowerShell con privilegios de administrador y mantener los scripts en el directorio C:\Scripts.

```powershell
#------------------------------------------------------------
# MENU PRINCIPAL
#------------------------------------------------------------
function Mostrar-Menu {
    do {
        Clear-Host
        Write-Host "========================================="
        Write-Host "   TAREAS PROGRAMADAS - TRUST.LAN"
        Write-Host "========================================="
        Write-Host "1. Mostrar ayuda del script"
        Write-Host "2. Crear tarea: Comprobar estado del sistema"
        Write-Host "3. Crear tarea: Buscar ficheros por extension"
        Write-Host "0. Salir"
        Write-Host "-----------------------------------------"
        $opcion = Read-Host "Seleccione una opcion"

        switch ($opcion) {
            "1" { Mostrar-Ayuda }
            "2" { Crear-Tarea-EstadoSistema }
            "3" { Crear-Tarea-BuscarFicheros }
            "0" { Write-Host "Saliendo..." }
            default { Write-Host "Opcion no valida."; Pause }
        }
    } while ($opcion -ne "0")
}

Mostrar-Menu
```

```
Administrador: Windows Pow
=========================================
   Script: tareas_programadas.ps1
   Grupo : Fast&Query
   Autores:
            Ruben Vicente Gilabert
            Vicent Melero Escriba
            Pau Cabanillas Marin
            Andrei Vasiliu
   Version: 1.1    Fecha: 18/11/2025
=========================================

Este script crea tareas programadas en el sistema
para automatizar tareas de mantenimiento:

  1) Comprobar el estado del sistema y registrar
     sucesos en alerta.log.
  2) Buscar ficheros de uno o varios usuarios en
     su carpeta personal y de una extension concreta.

Internamente utiliza el script gestion_sistema.ps1,
pasandole parametros de linea con la operacion a realizar.

Presione Entrar para continuar...:
```

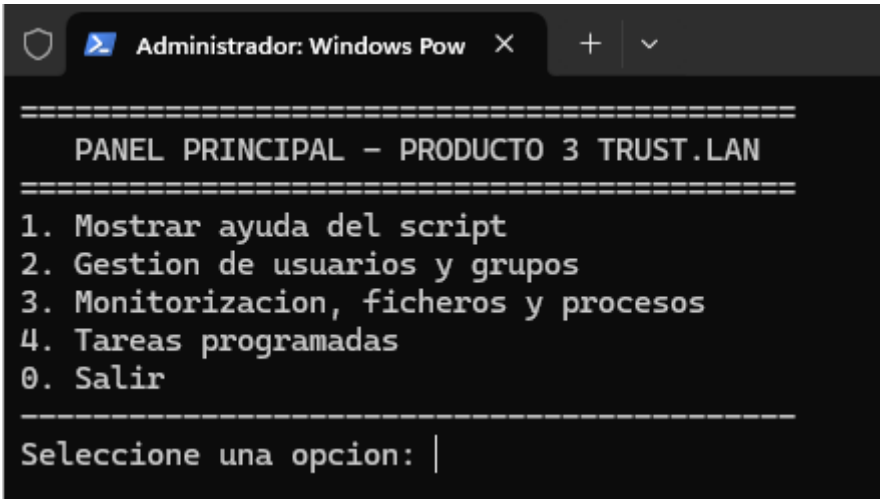
5. Crear un script que muestre un menú con opciones para hacer uso de todos los scripts anteriores.

Se ha creado un script principal, **menu_principal.ps1**, que funciona como punto de entrada al sistema de automatización desarrollado en el Producto 3.

Desde este menú, el administrador puede acceder a:

- **Gestión de usuarios y grupos** (gestion_usuarios.ps1)
- **Monitorización del sistema, gestión de ficheros y procesos** (gestion_sistema.ps1)
- **Creación de tareas programadas** (tareas_programadas.ps1)

De este modo, todo el sistema queda centralizado en un único menú, facilitando el uso y la navegación entre los distintos módulos.




```powershell
#------------------------------------------------------------
# Funciones para lanzar otros scripts
#------------------------------------------------------------

function Ejecutar-GestionUsuarios {
    if (Test-Path $Global:RutaGestionUsuarios) {
        Write-Host ">> Abriendo script de gestion de usuarios..." -ForegroundColor Cyan
        & $Global:RutaGestionUsuarios
    } else {
        Write-Host "No se ha encontrado $Global:RutaGestionUsuarios" -ForegroundColor Red
        Pause
    }
}

function Ejecutar-GestionSistema {
    if (Test-Path $Global:RutaGestionSistema) {
        Write-Host ">> Abriendo script de gestion del sistema..." -ForegroundColor Cyan
        & $Global:RutaGestionSistema
    } else {
        Write-Host "No se ha encontrado $Global:RutaGestionSistema" -ForegroundColor Red
        Pause
    }
}

function Ejecutar-TareasProgramadas {
    if (Test-Path $Global:RutaTareasProgramadas) {
        Write-Host ">> Abriendo script de tareas programadas..." -ForegroundColor Cyan
        & $Global:RutaTareasProgramadas
    } else {
        Write-Host "No se ha encontrado $Global:RutaTareasProgramadas" -ForegroundColor Red
        Pause
    }
}

#------------------------------------------------------------
# MENÚ PRINCIPAL
#------------------------------------------------------------
do {
    Clear-Host
    Write-Host "========================================="
    Write-Host "   PANEL PRINCIPAL - PRODUCTO 3 TRUST.LAN"
    Write-Host "========================================="
    Write-Host "1. Mostrar ayuda del script"
    Write-Host "2. Gestion de usuarios y grupos"
    Write-Host "3. Monitorizacion, ficheros y procesos"
    Write-Host "4. Tareas programadas"
    Write-Host "0. Salir"
```

Bibliografía

- Baiget Pellicer, G. (2019). *Administración avanzada, scripts, procesos, automatización* [recurso de aprendizaje]. Universitat Oberta de Catalunya (UOC). https://aula.uoc.edu/courses/67525/pages/recursos-de-aprendizaje-relacionados-con-el-producto-3

Anexo

gestion_usuarios.ps1

```
#############################################################
# Nombre del script   : gestion_usuarios.ps1
# Módulo / Producto   : FP.048 - Producto 3
# Empresa             : Trust SL - Administración de servidores
# Grupo               : Fast&Query
# Miembros            : Rubén Vicente Gilabert
#                       Vicent Melero Escriba
#                       Pau Cabanillas Marin
#                       Andrei Vasiliu
# Fecha               : 18/11/2025
# Version             : 1.4
#
# Descripcion:
#   Script para automatizar la gestion de usuarios y grupos
#   del dominio Trust.lan. Permite comprobar si existen y
#   crearlos de forma interactiva.
#
# Uso:
#   Ejecutar en PowerShell como Administrador del dominio.
#   Ejemplo:
#       .\gestion_usuarios.ps1
#
# Requisitos:
#   - Modulo ActiveDirectory instalado.
#   - Ejecucion con privilegios de administrador.
#############################################################

# Cargar modulo de Active Directory
Import-Module ActiveDirectory -ErrorAction Stop

#-------------------------------------------------------------
# Configuracion de rutas LDAP para evitar errores de escritura
#-------------------------------------------------------------
$Global:BaseDN    = "DC=trust,DC=lan"
```

```
# Donde se van a crear los grupos (estan dentro de 'Departamentos')
$Global:OUGrupos = "OU=Departamentos,$BaseDN"

# OUs de usuarios por departamento
$Global:OUUsuariosDeptos = @{
    "TIC"            = "OU=TIC,OU=Departamentos,$BaseDN"
    "Administracion" = "OU=Administracion y Finanzas,OU=Departamentos,$BaseDN"
    "Comercial"      = "OU=Comercial,OU=Departamentos,$BaseDN"
    "Diseno"         = "OU=Diseno,OU=Departamentos,$BaseDN"
    "Gerencia"       = "OU=Gerencia,OU=Departamentos,$BaseDN"
}

#------------------------------------------------------------
# Funcion: Test-Administrador
# Objetivo: Verificar que el script se ejecuta como admin
#------------------------------------------------------------
function Test-Administrador {
    $identity  = [Security.Principal.WindowsIdentity]::GetCurrent()
    $principal = New-Object Security.Principal.WindowsPrincipal($identity)

    if (-not
$principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)) {
        Write-Host "Este script debe ejecutarse con privilegios de administrador."
-ForegroundColor Red
        Write-Host "Cierra esta ventana y vuelve a abrir PowerShell como 'Ejecutar
como administrador'."
        exit 1
    }
}

# Ejecutar la comprobacion de administrador al inicio
Test-Administrador

#------------------------------------------------------------
# Funcion: Mostrar-Ayuda
# Objetivo: Mostrar informacion del script, autor, uso, etc.
#------------------------------------------------------------
function Mostrar-Ayuda {
    Clear-Host
    Write-Host "============================================"
```

```
    Write-Host "   Script: gestion_usuarios.ps1"
    Write-Host "   Grupo : Fast&Query"
    Write-Host "   Autores: "
    Write-Host "            Ruben Vicente Gilabert"
    Write-Host "            Vicent Melero Escriba"
    Write-Host "            Pau Cabanillas Marin"
    Write-Host "            Andrei Vasiliu"
    Write-Host "   Version: 1.4   Fecha: 18/11/2025"
    Write-Host "=========================================="
    Write-Host ""
    Write-Host "Descripcion:"
    Write-Host "  Automatiza la creacion y comprobacion de"
    Write-Host "  usuarios y grupos en el dominio Trust.lan."
    Write-Host ""
    Write-Host "Uso:"
    Write-Host "  Ejecutar el script y utilizar el menu para:"
    Write-Host "    1) Crear usuarios"
    Write-Host "    2) Verificar usuarios"
    Write-Host "    3) Crear grupos"
    Write-Host "    4) Verificar grupos"
    Write-Host ""
    Write-Host "Requisitos:"
    Write-Host "  - Ejecutar como administrador del dominio."
    Write-Host "  - Modulo ActiveDirectory instalado."
    Write-Host ""
    Pause
}

#------------------------------------------------------------
# Funcion: Verificar-Usuario
# Objetivo: Comprobar si un usuario existe en el dominio
#------------------------------------------------------------
function Verificar-Usuario {
    param(
        [Parameter(Mandatory = $true)]
        [string]$SamAccountName
    )

    $filtro  = "SamAccountName -eq '$SamAccountName'"
    $usuario = Get-ADUser -Filter $filtro -ErrorAction SilentlyContinue
```

```
    if ($null -ne $usuario) {
        Write-Host "El usuario '$SamAccountName' EXISTE en el dominio."
-ForegroundColor Green
        return $true
    } else {
        Write-Host "El usuario '$SamAccountName' NO existe en el dominio."
-ForegroundColor Yellow
        return $false
    }
}

#------------------------------------------------------------
# Funcion: Verificar-Grupo
# Objetivo: Comprobar si un grupo existe en el dominio
#------------------------------------------------------------
function Verificar-Grupo {
    param(
        [Parameter(Mandatory = $true)]
        [string]$NombreGrupo
    )

    $filtro = "Name -eq '$NombreGrupo'"
    $grupo  = Get-ADGroup -Filter $filtro -ErrorAction SilentlyContinue

    if ($null -ne $grupo) {
        Write-Host "El grupo '$NombreGrupo' EXISTE en el dominio."
-ForegroundColor Green
        return $true
    } else {
        Write-Host "El grupo '$NombreGrupo' NO existe en el dominio."
-ForegroundColor Yellow
        return $false
    }
}

#------------------------------------------------------------
# Funcion: Crear-Grupo
# Objetivo: Crear un grupo de seguridad con parametros validos
#------------------------------------------------------------
```

```
function Crear-Grupo {
    param(
        [Parameter(Mandatory = $true)]
        [string]$NombreGrupo,

        [Parameter(Mandatory = $true)]
        [ValidateSet("Global","Universal","DomainLocal")]
        [string]$Alcance,

        [Parameter(Mandatory = $true)]
        [ValidateSet("Security","Distribution")]
        [string]$Tipo,

        [Parameter(Mandatory = $true)]
        [string]$Descripcion
    )

    if (Verificar-Grupo -NombreGrupo $NombreGrupo) {
        Write-Host "No se creara el grupo porque ya existe." -ForegroundColor
Yellow
        return
    }

    # Ruta donde se crean los grupos
    $RutaOU = $Global:OUGrupos

    New-ADGroup `
        -Name $NombreGrupo `
        -GroupScope $Alcance `
        -GroupCategory $Tipo `
        -Path $RutaOU `
        -Description $Descripcion

    Write-Host "Grupo '$NombreGrupo' creado correctamente en '$RutaOU'."
-ForegroundColor Green
}

#-----------------------------------------------------------
# Funcion: Crear-Usuario
# Objetivo: Crear un usuario con toda la informacion necesaria
```

```
#-------------------------------------------------------------
function Crear-Usuario {
    param(
        [Parameter(Mandatory = $true)]
        [string]$Nombre,           # Nombre de pila

        [Parameter(Mandatory = $true)]
        [string]$Apellidos,

        [Parameter(Mandatory = $true)]
        [string]$SamAccountName,   # login (ruben.vicente)

        [Parameter(Mandatory = $true)]
        [ValidateSet("TIC","Administracion","Comercial","Diseno","Gerencia")]
        [string]$Departamento,

        [Parameter(Mandatory = $true)]
        [string]$PasswordPlano,

        [Parameter(Mandatory = $true)]
        [string]$GrupoPrincipal    # Nombre del grupo al que anadir
    )

    if (Verificar-Usuario -SamAccountName $SamAccountName) {
        Write-Host "No se creara el usuario porque ya existe." -ForegroundColor
Yellow
        return
    }

    if (-not (Verificar-Grupo -NombreGrupo $GrupoPrincipal)) {
        Write-Host "El grupo '$GrupoPrincipal' no existe. Crea el grupo antes de
anadir usuarios." -ForegroundColor Red
        return
    }

    # Obtener la OU a partir del departamento
    if ($Global:OUUsuariosDeptos.ContainsKey($Departamento)) {
        $OUPath = $Global:OUUsuariosDeptos[$Departamento]
    } else {
```

```
        Write-Host "Departamento '$Departamento' no definido en OUUsuariosDeptos."
-ForegroundColor Red
        return
    }

    $NombreCompleto = "$Nombre $Apellidos"
    $UPN            = "$SamAccountName@trust.lan"
    $PasswordSecure = ConvertTo-SecureString $PasswordPlano -AsPlainText -Force

    New-ADUser `
        -Name $NombreCompleto `
        -GivenName $Nombre `
        -Surname $Apellidos `
        -SamAccountName $SamAccountName `
        -UserPrincipalName $UPN `
        -Path $OUPath `
        -AccountPassword $PasswordSecure `
        -Enabled $true `
        -ChangePasswordAtLogon $true

    Add-ADGroupMember -Identity $GrupoPrincipal -Members $SamAccountName

    Write-Host "Usuario '$SamAccountName' creado correctamente en '$OUPath' y
anadido al grupo '$GrupoPrincipal'." -ForegroundColor Green
}

#------------------------------------------------------------
# Funciones INTERACTIVAS para pedir datos al usuario
#------------------------------------------------------------
function Crear-Grupo-Interactivo {
    Write-Host "=== CREACION DE GRUPO ==="
    $nombre      = Read-Host "Nombre del grupo (ej: grupo.comercial)"
    $alcance     = Read-Host "Alcance [Global/Universal/DomainLocal]"
    $tipo        = Read-Host "Tipo [Security/Distribution]"
    $descripcion = Read-Host "Descripcion del grupo"

    Crear-Grupo -NombreGrupo $nombre -Alcance $alcance -Tipo $tipo -Descripcion
$descripcion
    Pause
}
```

```
function Seleccionar-GrupoPrincipal {
    # Buscar todos los grupos que empiecen por 'grupo.'
    $grupos = Get-ADGroup -Filter 'Name -like "grupo.*"' | Sort-Object Name

    if (-not $grupos) {
        Write-Host "No se han encontrado grupos 'grupo.*' en el dominio."
-ForegroundColor Red
        return $null
    }

    Write-Host ""
    Write-Host "Seleccione el grupo principal para el usuario:"
    $index = 1
    foreach ($g in $grupos) {
        Write-Host ("  {0}) {1}" -f $index, $g.Name)
        $index++
    }

    $op = Read-Host "Opcion"
    if (-not ($op -as [int]) -or $op -lt 1 -or $op -gt $grupos.Count) {
        Write-Host "Opcion de grupo no valida." -ForegroundColor Red
        return $null
    }

    return $grupos[$op - 1].Name
}

function Crear-Usuario-Interactivo {
    Write-Host "=== CREACION DE USUARIO ==="
    $nombre     = Read-Host "Nombre"
    $apellidos  = Read-Host "Apellidos"
    $login      = Read-Host "Nombre de cuenta (samAccountName, ej: ruben.vicente)"

    Write-Host ""
    Write-Host "Seleccione departamento:"
    Write-Host "  1) TIC"
    Write-Host "  2) Administracion y Finanzas"
    Write-Host "  3) Comercial"
    Write-Host "  4) Diseno"
```

```
    Write-Host "  5) Gerencia"
    $depOpcion = Read-Host "Opcion"

    switch ($depOpcion) {
        "1" { $departamento = "TIC" }
        "2" { $departamento = "Administracion" }
        "3" { $departamento = "Comercial" }
        "4" { $departamento = "Diseno" }
        "5" { $departamento = "Gerencia" }
        default {
            Write-Host "Opcion de departamento no valida." -ForegroundColor Red
            Pause
            return
        }
    }

    $grupo = Seleccionar-GrupoPrincipal
    if (-not $grupo) {
        Pause
        return
    }

    $password   = Read-Host "Contrasena inicial (ej: Trust2025*)" -AsSecureString
    $passPlano  = [System.Net.NetworkCredential]::new("", $password).Password

    Crear-Usuario -Nombre $nombre -Apellidos $apellidos -SamAccountName $login
-Departamento $departamento -PasswordPlano $passPlano -GrupoPrincipal $grupo
    Pause
}

function Verificar-Usuario-Interactivo {
    $login = Read-Host "Introduce el samAccountName del usuario"
    Verificar-Usuario -SamAccountName $login
    Pause
}

function Verificar-Grupo-Interactivo {
    $grupo = Read-Host "Introduce el nombre del grupo"
    Verificar-Grupo -NombreGrupo $grupo
    Pause
```

```
}

#-------------------------------------------------------------
# MENU PRINCIPAL
#-------------------------------------------------------------
do {
    Clear-Host
    Write-Host "============================================"
    Write-Host "   GESTION DE USUARIOS Y GRUPOS - TRUST.LAN"
    Write-Host "============================================"
    Write-Host "1. Mostrar ayuda del script"
    Write-Host "2. Crear nuevo usuario"
    Write-Host "3. Verificar si un usuario existe"
    Write-Host "4. Crear nuevo grupo"
    Write-Host "5. Verificar si un grupo existe"
    Write-Host "0. Salir"
    Write-Host "--------------------------------------------"
    $opcion = Read-Host "Seleccione una opcion"
    $opcion = $opcion.Trim()

    switch ($opcion) {
        "1" { Mostrar-Ayuda }
        "2" { Crear-Usuario-Interactivo }
        "3" { Verificar-Usuario-Interactivo }
        "4" { Crear-Grupo-Interactivo }
        "5" { Verificar-Grupo-Interactivo }
        "0" {
            Write-Host "Saliendo..."
            Start-Sleep -Seconds 1
        }
        default {
            Write-Host "Opcion no valida."
            Pause
        }
    }

} until ($opcion -eq "0")
```

gestion_sistema.ps1

```
#############################################################
# Nombre del script  : gestion_sistema.ps1
# Módulo / Producto  : FP.048 - Producto 3
# Empresa            : Trust SL - Administración de servidores
# Grupo              : Fast&Query
# Miembros           : Rubén Vicente Gilabert
#                      Vicent Melero Escriba
#                      Pau Cabanillas Marin
#                      Andrei Vasiliu
# Fecha              : 18/11/2025
# Version            : 1.4
#
# Descripcion:
#   Script para gestionar la monitorizacion del sistema,
#   el sistema de ficheros y los procesos en servidores
#   del dominio Trust.lan.
#
# Uso:
#   Ejecutar en PowerShell como Administrador.
#   Ejemplo:
#       .\gestion_sistema.ps1
#
# Requisitos:
#   - PowerShell 5+.
#   - Ejecucion con privilegios de administrador.
#############################################################

#-------------------------------------------------------------
# FUNCIONES GENERALES
#-------------------------------------------------------------

# Verificar que el script se ejecuta como administrador
function Test-Administrador {
    $identity  = [Security.Principal.WindowsIdentity]::GetCurrent()
    $principal = New-Object Security.Principal.WindowsPrincipal($identity)
```

```
    if (-not
$principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)) {
        Write-Host "Este script debe ejecutarse con privilegios de administrador."
-ForegroundColor Red
        Write-Host "Cierra esta ventana y vuelve a abrir PowerShell como 'Ejecutar
como administrador'."
        exit 1
    }
}

Test-Administrador

# Ruta del fichero de alertas
$Global:FicheroAlerta = "C:\Scripts\logs\alerta.log"
$logDir = Split-Path $Global:FicheroAlerta -Parent

if (-not (Test-Path $logDir)) {
    New-Item -ItemType Directory -Path $logDir -Force | Out-Null
}
if (-not (Test-Path $Global:FicheroAlerta)) {
    New-Item -ItemType File -Path $Global:FicheroAlerta -Force | Out-Null
}

# Mostrar ayuda / informacion del script
function Mostrar-Ayuda {
    Clear-Host
    Write-Host "=========================================="
    Write-Host "   Script: gestion_sistema.ps1"
    Write-Host "   Grupo : Fast&Query"
    Write-Host "   Autores: "
    Write-Host "            Ruben Vicente Gilabert"
    Write-Host "            Vicent Melero Escriba"
    Write-Host "            Pau Cabanillas Marin"
    Write-Host "            Andrei Vasiliu"
    Write-Host "   Version: 1.4   Fecha: 18/11/2025"
    Write-Host "=========================================="
    Write-Host ""
    Write-Host "Este script permite:"
    Write-Host "  - Monitorizar CPU, memoria, procesos y disco."
    Write-Host "  - Buscar ficheros por usuario, extension y tamaño."
```

```
    Write-Host "  - Listar procesos de usuarios concretos o de todos"
    Write-Host "    los usuarios reales del sistema."
    Write-Host ""
    Write-Host "Todas las acciones requieren permisos de administrador."
    Write-Host ""
    Pause
}

# Registrar alerta en fichero
function Escribir-Alerta {
    param(
        [string]$Mensaje
    )
    $linea = "$(Get-Date -Format 'yyyy-MM-dd HH:mm:ss') - $Mensaje"
    Add-Content -Path $Global:FicheroAlerta -Value $linea
}

#------------------------------------------------------------
# MONITORIZACION DEL SISTEMA
#------------------------------------------------------------

# Estado CPU (utilizada y libre)
function Obtener-EstadoCPU {
    $cpuUsed = (Get-CimInstance Win32_Processor).LoadPercentage
    $cpuFree = 100 - $cpuUsed


    Write-Host "=== ESTADO CPU ==="
    Write-Host ("CPU utilizada : {0} %" -f $cpuUsed)
    Write-Host ("CPU libre     : {0} %" -f $cpuFree)
    Write-Host ""
}

# Cantidad de procesos activos
function Obtener-NumeroProcesos {
    $total = (Get-Process).Count
    Write-Host "=== PROCESOS ACTIVOS ==="
    Write-Host "Procesos en ejecucion: $total"
    Write-Host ""
}
```

```
# Espacio total y libre del sistema de ficheros (por unidad)
function Obtener-EstadoDiscos {
    Write-Host "=== ESTADO SISTEMA DE FICHEROS ==="
    Get-PSDrive -PSProvider FileSystem | ForEach-Object {
        $totalGB  = [math]::Round(($_.Used + $_.Free)/1GB,2)
        $freeGB   = [math]::Round($_.Free/1GB,2)
        $porcFree = if ($totalGB -ne 0) { [math]::Round(($freeGB/$totalGB)*100,2)
} else { 0 }
        Write-Host ("Unidad {0}: Total {1} GB - Libre {2} GB ({3} %)" -f
$_.Name,$totalGB,$freeGB,$porcFree)
    }
    Write-Host ""
}

# Comprobar estado del sistema con umbrales y registrar en alerta.log
function Comprobar-EstadoSistema {

    # --- CPU ---
    $cpuUsed = (Get-CimInstance Win32_Processor).LoadPercentage


    # --- Memoria ---
    $os = Get-CimInstance Win32_OperatingSystem
    $totalMemKB  = [double]$os.TotalVisibleMemorySize
    $freeMemKB   = [double]$os.FreePhysicalMemory
    $usedMemPerc = [math]::Round((1 - ($freeMemKB / $totalMemKB)) * 100,2)

    # --- Disco (unidad C:) ---
    $driveC    = Get-PSDrive -Name C
    $totalGB   = [math]::Round(($driveC.Used + $driveC.Free)/1GB,2)
    $freeGB    = [math]::Round($driveC.Free/1GB,2)
    $freePerc  = if ($totalGB -ne 0) { [math]::Round(($freeGB/$totalGB)*100,2) }
else { 0 }

    Write-Host "=== COMPROBACION DE ESTADO ==="
    Write-Host ("CPU usada      : {0} %" -f $cpuUsed)
    Write-Host ("Memoria usada  : {0} %" -f $usedMemPerc)
    Write-Host ("Unidad C libre : {0} GB ({1} %)" -f $freeGB, $freePerc)
    Write-Host ""
```

```
    # Umbrales
    if ($usedMemPerc -gt 80) {
        $msg = "ALERTA: Memoria usada $usedMemPerc % (>80%)"
        Write-Host "ADVERTENCIA: $msg" -ForegroundColor Yellow
        Escribir-Alerta $msg
    }

    if ($cpuUsed -gt 85) {
        $msg = "ALERTA: CPU usada $cpuUsed % (>85%)"
        Write-Host "ADVERTENCIA: $msg" -ForegroundColor Yellow
        Escribir-Alerta $msg
    }

    if ($freePerc -lt 15) {
        $msg = "ALERTA: Espacio libre en C: $freePerc % (<15%)"
        Write-Host "ADVERTENCIA: $msg" -ForegroundColor Yellow
        Escribir-Alerta $msg
    }

    if (($usedMemPerc -le 80) -and ($cpuUsed -le 85) -and ($freePerc -ge 15)) {
        Write-Host "El sistema se encuentra dentro de los parametros normales."
-ForegroundColor Green
    }

    Write-Host ""
    Pause
}

#-------------------------------------------------------------
# SISTEMA DE FICHEROS
#-------------------------------------------------------------

# Localizar ficheros de un usuario en una ruta
function Buscar-FicherosUsuario {
    param(
        [Parameter(Mandatory = $true)]
        [string]$Usuario,    # dominio\usuario o solo usuario
        [Parameter(Mandatory = $true)]
        [string]$Ruta
```

```
    )

    Write-Host "Buscando ficheros propiedad de '$Usuario' en $Ruta..."
-ForegroundColor Cyan
    Get-ChildItem -Path $Ruta -Recurse -ErrorAction SilentlyContinue |
        Where-Object {
            try {
                ($_.PSIsContainer -eq $false) -and
                ($_.GetAccessControl().Owner -like "*$Usuario")
            } catch {
                $false
            }
        } |
        Select-Object FullName,
                      Length,
                      @{Name="Owner";Expression={$_.GetAccessControl().Owner}} |
        Format-Table -AutoSize

    Write-Host ""
    Pause
}

# Localizar ficheros por extension y tamaño minimo
function Buscar-FicherosTipo {
    param(
        [Parameter(Mandatory = $true)]
        [string[]]$Extensiones,    # ej: "log","txt","docx"

        [Parameter(Mandatory = $true)]
        [string]$TamanoMinMB,      # cadena tal cual viene de Read-Host (puede
estar vacia)

        [Parameter(Mandatory = $true)]
        [string]$Ruta
    )

    # Normalizar extensiones: "txt" -> ".txt"
    $extsNormalizadas = $Extensiones | ForEach-Object {
        $e = $_.Trim()
        if ($e.StartsWith(".")) { $e.ToLower() } else { (".$e").ToLower() }
```

```
    }

    if (-not [string]::IsNullOrWhiteSpace($TamanoMinMB)) {
        # Intentamos convertir a entero
        $minInt = 0
        if ([int]::TryParse($TamanoMinMB, [ref]$minInt)) {
            $usarTamano = $true
            $minBytes   = $minInt * 1MB
        } else {
            Write-Host "Valor de tamaño minimo no valido. Se ignorara el filtro de
tamano." -ForegroundColor Yellow
        }
    }

    if ($usarTamano) {
        Write-Host "Buscando ficheros en $Ruta con extensiones $($extsNormalizadas
-join ', ') y tamano >= $TamanoMinMB MB..." -ForegroundColor Cyan
    } else {
        Write-Host "Buscando ficheros en $Ruta con extensiones $($extsNormalizadas
-join ', ') (sin filtro de tamano)..." -ForegroundColor Cyan
    }

    Get-ChildItem -Path $Ruta -Recurse -File -ErrorAction SilentlyContinue |
        Where-Object {
            $extOK = $extsNormalizadas -contains $_.Extension.ToLower()

            if ($usarTamano) {
                $sizeOK = $_.Length -ge $minBytes
                $extOK -and $sizeOK
            } else {
                $extOK
            }
        } |
        Select-Object FullName,

@{Name="TamanoMB";Expression={[math]::Round($_.Length/1MB,2)}} |
        Format-Table -AutoSize

    Write-Host ""
    Pause
```

```
}



#------------------------------------------------------------
# GESTION DE PROCESOS
#------------------------------------------------------------

# Mostrar procesos de dos usuarios concretos
function Mostrar-ProcesosDosUsuarios {
    param(
        [Parameter(Mandatory = $true)]
        [string]$Usuario1,    # ej: "TRUST\ruben.vicente"
        [Parameter(Mandatory = $true)]
        [string]$Usuario2
    )

    Write-Host "=== PROCESOS DE $Usuario1 Y $Usuario2 ==="
    Get-Process -IncludeUserName -ErrorAction SilentlyContinue |
        Where-Object { $_.UserName -eq $Usuario1 -or $_.UserName -eq $Usuario2 } |
        Select-Object UserName, Id, ProcessName, CPU, PM |
        Format-Table -AutoSize

    Write-Host ""
    Pause
}

# Mostrar todos los procesos de usuarios reales
function Mostrar-ProcesosUsuariosReales {

    Write-Host "=== PROCESOS DE USUARIOS REALES ==="

    $os = Get-CimInstance Win32_OperatingSystem
    $totalMemBytes = [double]$os.TotalVisibleMemorySize * 1KB

    Get-Process -IncludeUserName -ErrorAction SilentlyContinue |
        Where-Object {
            $_.UserName -and
            ($_.UserName -notlike "*SYSTEM*") -and
            ($_.UserName -notlike "*SERVICIO*") -and
```

```
            ($_.UserName -notlike "*SERVICE*")
        } |
        ForEach-Object {
            $proc   = $_
            $cpuSeg = $proc.CPU
            $mem    = $proc.WorkingSet64

            $inicio = $null
            $estado = "Desconocido"
            $comando = $null

            try { $inicio = $proc.StartTime } catch {}
            try { $comando = $proc.Path } catch {}

            if ($proc.Responding) { $estado = "En ejecucion" }

            # Porcentaje aproximado de memoria
            $memPerc = if ($totalMemBytes -gt 0) { [math]::Round(($mem /
$totalMemBytes) * 100, 2) } else { 0 }

            # Porcentaje aproximado de CPU (tiempo de CPU / tiempo vivo)
            $cpuPerc = 0
            if ($inicio) {
                $segundosVida = (New-TimeSpan -Start $inicio -End
(Get-Date)).TotalSeconds
                if ($segundosVida -gt 0) {
                    $cpuPerc = [math]::Round(($cpuSeg / $segundosVida) * 100, 2)
                }
            }

            [PSCustomObject]@{
                Usuario          = $proc.UserName
                PID              = $proc.Id
                Proceso          = $proc.ProcessName
                CPU_Porcentaje   = $cpuPerc
                Memoria_Porc     = $memPerc
                CPU_Segundos     = [math]::Round($cpuSeg,2)
                Memoria_MB       = [math]::Round($mem/1MB,2)
                Estado           = $estado
                HoraInicio       = $inicio
```

```
                TiempoEjecucion  = if ($inicio) { (New-TimeSpan -Start $inicio
-End (Get-Date)).ToString() } else { "N/D" }
                Comando          = $comando
            }
        } |
        Sort-Object Usuario, Proceso |
        Format-Table -AutoSize

    Write-Host ""
    Pause
}

#------------------------------------------------------------
# MENUS INTERACTIVOS
#------------------------------------------------------------

function Menu-Monitoreo {
    do {
        Clear-Host
        Write-Host "=== MONITORIZACION DEL SISTEMA ==="
        Write-Host "1. Ver estado de CPU"
        Write-Host "2. Ver numero de procesos activos"
        Write-Host "3. Ver estado de discos"
        Write-Host "4. Comprobar estado del sistema (con alertas)"
        Write-Host "0. Volver al menu principal"
        $op = Read-Host "Opcion"

        switch ($op) {
            "1" { Obtener-EstadoCPU; Pause }
            "2" { Obtener-NumeroProcesos; Pause }
            "3" { Obtener-EstadoDiscos; Pause }
            "4" { Comprobar-EstadoSistema }
            "0" { }
            default { Write-Host "Opcion no valida."; Pause }
        }
    } while ($op -ne "0")
}

function Menu-Ficheros {
    do {
```

```
        Clear-Host
        Write-Host "=== SISTEMA DE FICHEROS ==="
        Write-Host "1. Buscar ficheros por propietario (usuario)"
        Write-Host "2. Buscar ficheros por extension y tamano minimo"
        Write-Host "0. Volver al menu principal"
        $op = Read-Host "Opcion"

        switch ($op) {
            "1" {
                $usuario = Read-Host "Introduce el usuario (ej:
TRUST\ruben.vicente)"
                $ruta    = Read-Host "Ruta base (ej: C:\Usuarios\)"
                Buscar-FicherosUsuario -Usuario $usuario -Ruta $ruta
            }
            "2" {
                $exts  = Read-Host "Extensiones separadas por coma (ej:
log,txt,docx)"
                $minMB = Read-Host "Tamano minimo en MB"
                $ruta  = Read-Host "Ruta base (ej: C:\)"
                $listaExt = $exts.Split(",") | ForEach-Object { $_.Trim() }
                Buscar-FicherosTipo -Extensiones $listaExt -TamanoMinMB $minMB
-Ruta $ruta
            }
            "0" { }
            default { Write-Host "Opcion no valida."; Pause }
        }
    } while ($op -ne "0")
}


function Menu-Procesos {
    do {
        Clear-Host
        Write-Host "=== GESTION DE PROCESOS ==="
        Write-Host "1. Mostrar procesos de dos usuarios concretos"
        Write-Host "2. Mostrar procesos de todos los usuarios reales"
        Write-Host "0. Volver al menu principal"
        $op = Read-Host "Opcion"

        switch ($op) {
```

```
            "1" {
                $u1 = Read-Host "Usuario 1 (ej: TRUST\ruben.vicente)"
                $u2 = Read-Host "Usuario 2 (ej: TRUST\benjamin.placeta)"
                Mostrar-ProcesosDosUsuarios -Usuario1 $u1 -Usuario2 $u2
            }
            "2" {
                Mostrar-ProcesosUsuariosReales
            }
            "0" { }
            default { Write-Host "Opcion no valida."; Pause }
        }
    } while ($op -ne "0")
}

#------------------------------------------------------------
# MENU PRINCIPAL
#------------------------------------------------------------

do {
    Clear-Host
    Write-Host "============================================"
    Write-Host "   GESTION DEL SISTEMA - TRUST.LAN"
    Write-Host "============================================"
    Write-Host "1. Mostrar ayuda del script"
    Write-Host "2. Monitorizacion del sistema"
    Write-Host "3. Sistema de ficheros"
    Write-Host "4. Gestion de procesos"
    Write-Host "0. Salir"
    Write-Host "--------------------------------------------"
    $opcion = Read-Host "Seleccione una opcion"

    switch ($opcion) {
        "1" { Mostrar-Ayuda }
        "2" { Menu-Monitoreo }
        "3" { Menu-Ficheros }
        "4" { Menu-Procesos }
        "0" { Write-Host "Saliendo..." }
        default { Write-Host "Opcion no valida."; Pause }
    }
}
```

```
while ($opcion -ne "0")
```

tareas_programadas.ps1

```
#############################################################
# Nombre del script   : tareas_programadas.ps1
# Modulo / Producto   : FP.048 - Producto 3
# Empresa             : Trust SL - Administracion de servidores
# Grupo               : Fast&Query
# Miembros            : Ruben Vicente Gilabert
#                       Vicent Melero Escriba
#                       Pau Cabanillas Marin
#                       Andrei Vasiliu
# Fecha               : 18/11/2025
# Version             : 1.1
#
# Descripcion:
#    Script para crear tareas programadas en el servidor
#    que llamen a otros scripts de administracion:
#    - Comprobar el estado del sistema (alerta.log)
#    - Buscar ficheros de usuarios por extension
#
# Uso:
#    Ejecutar en PowerShell como Administrador.
#    Ejemplo:
#        .\tareas_programadas.ps1
#
# Requisitos:
#    - PowerShell 5+
#    - Ejecutar como administrador
#    - Scripts ubicados en C:\Scripts\
#############################################################

# Ruta al script de gestion del sistema
$Global:RutaGestionSistema = "C:\Scripts\gestion_sistema.ps1"
$Global:PowerShellExe      =
"$env:SystemRoot\System32\WindowsPowerShell\v1.0\powershell.exe"

#-------------------------------------------------------------
```

```
# Comprobar que se ejecuta como administrador
#-------------------------------------------------------------
function Test-Administrador {
    $identity  = [Security.Principal.WindowsIdentity]::GetCurrent()
    $principal = New-Object Security.Principal.WindowsPrincipal($identity)

    if (-not
$principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)) {
        Write-Host "Este script debe ejecutarse con privilegios de administrador."
-ForegroundColor Red
        exit 1
    }
}
Test-Administrador

#-------------------------------------------------------------
# Mostrar ayuda del script
#-------------------------------------------------------------
function Mostrar-Ayuda {
    Clear-Host
    Write-Host "============================================"
    Write-Host "   Script: tareas_programadas.ps1"
    Write-Host "   Grupo : Fast&Query"
    Write-Host "   Autores: "
    Write-Host "            Ruben Vicente Gilabert"
    Write-Host "            Vicent Melero Escriba"
    Write-Host "            Pau Cabanillas Marin"
    Write-Host "            Andrei Vasiliu"
    Write-Host "   Version: 1.1   Fecha: 18/11/2025"
    Write-Host "============================================"
    Write-Host ""
    Write-Host "Este script crea tareas programadas en el sistema"
    Write-Host "para automatizar tareas de mantenimiento:"
    Write-Host ""
    Write-Host "  1) Comprobar el estado del sistema y registrar"
    Write-Host "     sucesos en alerta.log."
    Write-Host "  2) Buscar ficheros de uno o varios usuarios en"
    Write-Host "     su carpeta personal y de una extension concreta."
    Write-Host ""
    Write-Host "Internamente utiliza el script gestion_sistema.ps1,"
```

```
    Write-Host "pasandole parametros de linea con la operacion a realizar."
    Write-Host ""
    Pause
}

#-----------------------------------------------------------
# Crear tarea: Comprobar estado del sistema
#   Llama a gestion_sistema.ps1 -Operacion Estado
#-----------------------------------------------------------
function Crear-Tarea-EstadoSistema {

    Write-Host "=== CREAR TAREA: COMPROBAR ESTADO DEL SISTEMA ==="

    $nombreTarea = Read-Host "Nombre de la tarea (ej: ComprobarEstadoDiario)"
    $hora        = Read-Host "Hora de ejecucion diaria (HH:MM, ej: 09:00)"

    # Argumentos: ejecutar el script gestion_sistema.ps1 con la operacion Estado
    $argumentos = "-NoProfile -ExecutionPolicy Bypass -File
`"$Global:RutaGestionSistema`" -Operacion Estado"

    $accion  = New-ScheduledTaskAction -Execute $Global:PowerShellExe -Argument
$argumentos
    $trigger = New-ScheduledTaskTrigger -Daily -At $hora

    Register-ScheduledTask `
        -TaskName $nombreTarea `
        -Action $accion `
        -Trigger $trigger `
        -Description "Comprueba el estado del sistema y registra alertas en
alerta.log" `
        -User "SYSTEM" `
        -RunLevel Highest

    Write-Host "Tarea '$nombreTarea' creada correctamente." -ForegroundColor Green
    Pause
}

#-----------------------------------------------------------
# Crear tarea: Buscar ficheros por extension
#   Llama a gestion_sistema.ps1 -Operacion BuscarFicheros
```

```
#   Pasando ruta, extensiones y tamaño minimo
#-------------------------------------------------------------
function Crear-Tarea-BuscarFicheros {

    Write-Host "=== CREAR TAREA: BUSCAR FICHEROS POR EXTENSION ==="

    $nombreTarea = Read-Host "Nombre de la tarea (ej: BuscarLogsUsuarios)"
    $hora        = Read-Host "Hora de ejecucion diaria (HH:MM, ej: 23:00)"
    $rutaBase    = Read-Host "Ruta base donde buscar (ej: C:\Usuarios\)"
    $exts        = Read-Host "Extensiones separadas por coma (ej: log,txt)"
    $tamanoMin   = Read-Host "Tamano minimo en MB (dejar vacio para sin filtro)"

    # Construir argumentos para el script gestion_sistema.ps1
    # Ejemplo final:
    #   -File "C:\Scripts\gestion_sistema.ps1" -Operacion BuscarFicheros -Ruta
"C:\Usuarios" -Ext "log,txt" -MinMB 10
    $argumentos = "-NoProfile -ExecutionPolicy Bypass -File
`"$Global:RutaGestionSistema`" -Operacion BuscarFicheros -Ruta `"$rutaBase`" -Ext
`"$exts`""

    if (-not [string]::IsNullOrWhiteSpace($tamanoMin)) {
        $argumentos += " -MinMB $tamanoMin"
    }

    $accion  = New-ScheduledTaskAction -Execute $Global:PowerShellExe -Argument
$argumentos
    $trigger = New-ScheduledTaskTrigger -Daily -At $hora

    Register-ScheduledTask `
        -TaskName $nombreTarea `
        -Action $accion `
        -Trigger $trigger `
        -Description "Busca ficheros por extension y tamaño minimo en $rutaBase" `
        -User "SYSTEM" `
        -RunLevel Highest

    Write-Host "Tarea '$nombreTarea' creada correctamente." -ForegroundColor Green
    Pause
}
```

```
#------------------------------------------------------------
# MENU PRINCIPAL
#------------------------------------------------------------
function Mostrar-Menu {
    do {
        Clear-Host
        Write-Host "============================================"
        Write-Host "   TAREAS PROGRAMADAS - TRUST.LAN"
        Write-Host "============================================"
        Write-Host "1. Mostrar ayuda del script"
        Write-Host "2. Crear tarea: Comprobar estado del sistema"
        Write-Host "3. Crear tarea: Buscar ficheros por extension"
        Write-Host "0. Salir"
        Write-Host "--------------------------------------------"
        $opcion = Read-Host "Seleccione una opcion"

        switch ($opcion) {
            "1" { Mostrar-Ayuda }
            "2" { Crear-Tarea-EstadoSistema }
            "3" { Crear-Tarea-BuscarFicheros }
            "0" { Write-Host "Saliendo..." }
            default { Write-Host "Opcion no valida."; Pause }
        }
    } while ($opcion -ne "0")
}

Mostrar-Menu
```

menu_principal.ps1

```
############################################################
# Nombre del script  : menu_principal.ps1
# Módulo / Producto  : FP.048 - Producto 3
# Empresa            : Trust SL - Administracion de servidores
# Grupo              : Fast&Query
# Miembros           : Ruben Vicente Gilabert
#                      Vicent Melero Escriba
#                      Pau Cabanillas Marin
#                      Andrei Vasiliu
```

```
# Fecha              : 18/11/2025
# Versión            : 1.1
#
# Descripción:
#   Script principal que muestra un menú para hacer uso de
#   todos los scripts desarrollados en el Producto 3:
#     - gestion_usuarios.ps1
#     - gestion_sistema.ps1
#     - tareas_programadas.ps1
#
# Uso:
#   Ejecutar en PowerShell como Administrador.
#   Ejemplo:
#       .\menu_principal.ps1
#
# Requisitos:
#   - PowerShell 5+
#   - Ejecutar como administrador
#   - Scripts ubicados en C:\Scripts\
############################################################

# Rutas de los otros scripts
$Global:RutaGestionUsuarios   = "C:\Scripts\gestion_usuarios.ps1"
$Global:RutaGestionSistema    = "C:\Scripts\gestion_sistema.ps1"
$Global:RutaTareasProgramadas = "C:\Scripts\tareas_programadas.ps1"

#-----------------------------------------------------------
# Comprobar que se ejecuta como administrador
#-----------------------------------------------------------
function Test-Administrador {
    $identity  = [Security.Principal.WindowsIdentity]::GetCurrent()
    $principal = New-Object Security.Principal.WindowsPrincipal($identity)

    if (-not
$principal.IsInRole([Security.Principal.WindowsBuiltInRole]::Administrator)) {
        Write-Host "Este script debe ejecutarse con privilegios de administrador."
-ForegroundColor Red
        exit 1
    }
}
```

```
Test-Administrador

#-------------------------------------------------------------
# Mostrar ayuda del script
#-------------------------------------------------------------
function Mostrar-Ayuda {
    Clear-Host
    Write-Host "============================================"
    Write-Host "   Script: menu_principal.ps1"
     Write-Host "   Grupo : Fast&Query"
    Write-Host "   Autores: "
    Write-Host "            Ruben Vicente Gilabert"
    Write-Host "            Vicent Melero Escriba"
    Write-Host "            Pau Cabanillas Marin"
    Write-Host "            Andrei Vasiliu"
    Write-Host "   Versión: 1.1   Fecha: 18/11/2025"
    Write-Host "============================================"
    Write-Host ""
    Write-Host "Este script actua como panel principal del"
    Write-Host "Producto 3. Desde aqui se puede acceder a:"
    Write-Host ""
    Write-Host "  1) Gestion de usuarios y grupos del dominio."
    Write-Host "  2) Monitorizacion, ficheros y procesos."
    Write-Host "  3) Creacion de tareas programadas."
    Write-Host ""
    Write-Host "Todos los scripts deben estar ubicados en:"
    Write-Host "  C:\Scripts\"
    Write-Host ""
    Pause
}

#-------------------------------------------------------------
# Funciones para lanzar otros scripts
#-------------------------------------------------------------

function Ejecutar-GestionUsuarios {
    if (Test-Path $Global:RutaGestionUsuarios) {
        Write-Host ">> Abriendo script de gestion de usuarios..." -ForegroundColor
Cyan
        & $Global:RutaGestionUsuarios
```

```
    } else {
        Write-Host "No se ha encontrado $Global:RutaGestionUsuarios"
-ForegroundColor Red
        Pause
    }
}

function Ejecutar-GestionSistema {
    if (Test-Path $Global:RutaGestionSistema) {
        Write-Host ">> Abriendo script de gestion del sistema..." -ForegroundColor
Cyan
        & $Global:RutaGestionSistema
    } else {
        Write-Host "No se ha encontrado $Global:RutaGestionSistema"
-ForegroundColor Red
        Pause
    }
}

function Ejecutar-TareasProgramadas {
    if (Test-Path $Global:RutaTareasProgramadas) {
        Write-Host ">> Abriendo script de tareas programadas..." -ForegroundColor
Cyan
        & $Global:RutaTareasProgramadas
    } else {
        Write-Host "No se ha encontrado $Global:RutaTareasProgramadas"
-ForegroundColor Red
        Pause
    }
}

#-------------------------------------------------------------
# MENÚ PRINCIPAL
#-------------------------------------------------------------
do {
    Clear-Host
    Write-Host "============================================"
    Write-Host "   PANEL PRINCIPAL - PRODUCTO 3 TRUST.LAN"
    Write-Host "============================================"
    Write-Host "1. Mostrar ayuda del script"
```

```
    Write-Host "2. Gestion de usuarios y grupos"
    Write-Host "3. Monitorizacion, ficheros y procesos"
    Write-Host "4. Tareas programadas"
    Write-Host "0. Salir"
    Write-Host "-----------------------------------------"
    $opcion = Read-Host "Seleccione una opcion"

    switch ($opcion) {
        "1" { Mostrar-Ayuda }
        "2" { Ejecutar-GestionUsuarios }
        "3" { Ejecutar-GestionSistema }
        "4" { Ejecutar-TareasProgramadas }
        "0" { Write-Host "Saliendo..." }
        default { Write-Host "Opcion no valida."; Pause }
    }
} while ($opcion -ne "0")
```