

## 0.- Introducción a Windows

En las siguientes semanas nos centraremos en administrar sistemas operativos Windows. Veremos un Sistema Windows para equipos clientes y un Sistema Windows para equipos servidores. Lo primero que debes hacer es acceder a las páginas siguientes de Microsoft y leer qué nos ofrecen estos dos sistemas operativos :

- [Introducción y Características de Windows 11](#)
- [Introducción y Características de Windows Server 2022](#)

Vamos rápidamente a empezar a trabajar con Windows (empezaremos con Windows 11 y, más adelante nos pasaremos a Windows Server 2022, aunque, en general son prácticamente iguales, salvo que en este último podremos instalar servicios de servidor, como Directorio Activo, DHCP, DNS, IIS, etc). Así, lo primero, será tener una máquina Windows 11 (la versión Profesional) con la que poder trabajar, párate a instalarla y ver la problemática con el **chip TPM** que requiere este sistema operativo que tenga tu equipo.

Windows es un sistema operativo muy orientado a entorno gráfico, donde la línea de comandos **cmd** fue olvidada durante muchos años. Hace unos 15 años, Microsoft abordó mejorar la línea de comandos de Windows, creando para ello una nueva interfaz de consola llamada **PowerShell**. Esta consola está cada vez más optimizada y, por lo tanto, es ampliamente utilizada. Como futuros administradores de sistemas, nuestra obligación es saber administrar el equipo desde línea de comandos para así poder automatizar las tareas y evitar, en lo posible, la intervención “manual” para llevarlas a cabo.



## 1.- Introducción a PowerShell

Windows PowerShell es una interfaz de consola (*Command-Line Interface*, CLI) con posibilidad de escritura y unión de comandos por medio de instrucciones (*scripts*). Es un CLI mucho más rico que DOS. Esta interfaz de consola está diseñada para su uso por parte de administradores de sistemas, con el propósito de automatizar tareas o realizarlas de forma más controlada. Originalmente denominada como **MONAD** en 2003, su nombre oficial cambió al actual cuando fue lanzada al público el 25 de abril de 2006.

### 1.1.- Versiones de Powershell

- **PowerShell 1.0** : PowerShell 1.0 fue liberado en 2006 para el Windows XP SP2, Windows Server 2003 y Windows Vista. Es un componente opcional para Windows Server 2008.
- **PowerShell 2.0** : PowerShell 2.0 está integrado con Windows 7 y Windows Server 2008 R2 y es liberado para Windows XP con Service Pack 3, Windows Server 2003 con Service Pack 2 y Windows Vista con Service Pack 1.
  - PowerShell V2 incluye cambios del lenguaje de *scripting* y la API del equipo, añadiendo unos 240 *cmdlets*.
  - **Windows PowerShell ISE v2.0**, es un entorno de desarrollo para *scripts* PowerShell, está integrado en el Windows 7 y en el Windows Server 2008 R2 y es liberado para Windows XP con Service Pack 3, Windows Server 2003 con Service Pack 2 y Windows Vista con Service Pack 1.
- **PowerShell 3.0** : PowerShell 3.0 está integrado en el Windows 8 y en el Windows Server 2012. Microsoft también hizo PowerShell 3.0 posible para Windows 7 con Service Pack 1, para Windows Server 2008 con Service Pack 1 y para Windows Server 2008 R2 con Service Pack 1.
  - PowerShell 3.0 es una parte de un paquete mucho más grande, *Windows Management Framework* 3.0 (WMF3), que también contiene el servicio [WinRM](#).
- **PowerShell 4.0** : PowerShell 4.0 está integrado con Windows 8.1 y con Windows Server 2012 R2. Microsoft también hizo posible PowerShell 4.0 para Windows 7 SP1, Windows Server 2008 R2 SP1 y Windows Server 2012.
- **PowerShell 5.0** : La revisión previa del PowerShell 5.0 estuvo disponible con *Windows Management Framework* 5.0 (WMF5) en Abril de 2014.
  - Una de las novedades que introduce es [OneGet PowerShell cmdlets](#) para soportar el manejo de paquetes del repositorio [Chocolatey](#).
  - En Agosto de 2015 Microsoft se libera la última actualización de PowerShell 5.0.
  - Windows 10 ya lo trae integrado.
- **PowerShell 5.1** : Aparece con la Versión 1607 de Windows 10 (*Anniversary Update*) y en Windows 2016.
  - Windows 11 y Windows Server 2022 ya lo traen integrado.
- **PowerShell Core 6.0, 6.1 y 6.2** : Anunciada en Agosto de 2016.
  - PowerShell Core es un PowerShell más pequeño que el PowerShell convencional y puede instalarse en sistemas operativos: MacOS y Linux, también en Windows.
  - Con PowerShell Core, podremos administrar y ejecutar los mismos scripts sobre los tres tipos de sistemas operativos: Windows MacOS y Linux como si fueran uno solo.
- **PowerShell 7** : La primera versión es del 2019.
  - Es el producto de reemplazo para los productos PowerShell Core 6.x y para Windows PowerShell 5.1, que es la última versión de Windows PowerShell soportada y que llevan preinstalada el Windows 2011 y el Windows Server 2022.
  - Para que PowerShell 7 sea un reemplazo viable para Windows PowerShell 5.1 debe tener casi paridad con Windows PowerShell en términos de compatibilidad con los módulos que se envían con Windows.
  - Construido sobre .NET Core 3.1 (LTS).
  - Reemplazará en el futuro al PowerShell Core 6.x y a Windows PowerShell 5.1.

## 1.2.- La sintaxis de Windows PowerShell

El lenguaje PowerShell está formado por comandos que se denominan cmdlets. Se trata de comandos descritos con la forma de un verbo y, a continuación, un sustantivo, separados por un guión. Para hacerse una idea, algunos cmdlets pueden ser los siguientes:

- **New-Item** : Crear (*New*) un nuevo objeto (*Item*).
- **Get-Service** : Obtener (*Get*) información vinculada a los servicios (*Service*) de Windows.
- **Set-ExecutionPolicy** : Definir (*Set*) la política de ejecución (*ExecutionPolicy*) de los *scripts* en PowerShell.

## 1.3.- Ejecutar Windows PowerShell

Un modo rápido de ejecutar PowerShell es pulsar la **Teca de Windows + R** para que se ejecute "Ejecutar", escribir **PowerShell** y pulsar **Enter**.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

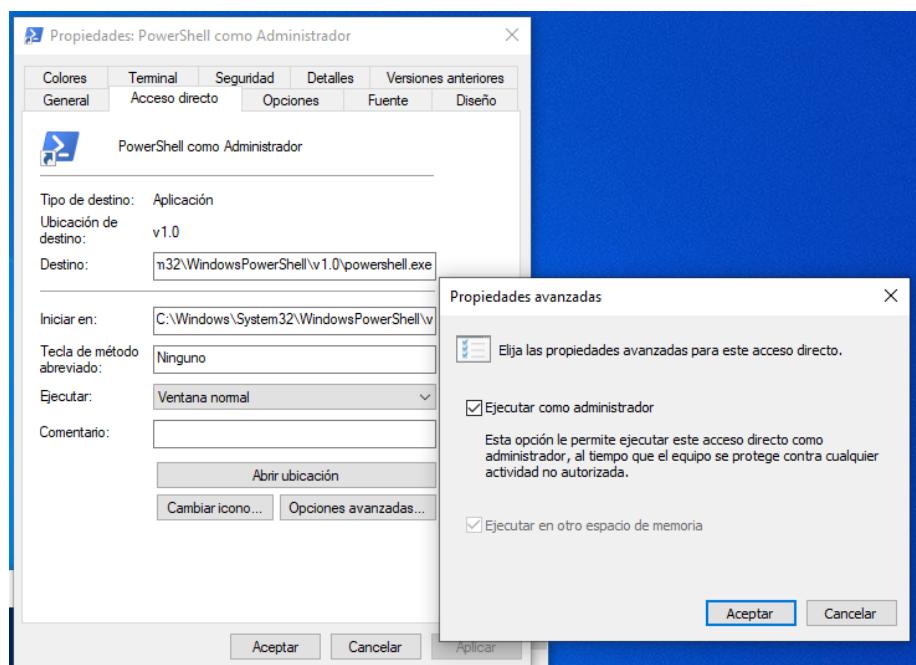
Prueba la nueva tecnología PowerShell multiplataforma https://aka.ms/powershell

PS C:\Users\vieites>
```

Al ejecutarlo así, en un sistema Windows Cliente, lo haremos sin privilegios, por lo que NO será posible realizar tareas de administración, como crear particiones, formatearlas, instalar software, etc.

**Para ejecutar Windows PowerShell como Administrador**, lo podemos realizar de varios modos:

- **Crear un Acceso directo especial:** Se crea un acceso directo en el escritorio. Donde pone "Escriba la localización del elemento" escribimos simplemente "PowerShell" y pulsamos en "Siguiente". Luego le damos un nombre para este acceso directo como, por ejemplo: "PowerShell como Administrador". Luego, una vez creado el acceso directo, hacemos *click* con el botón derecho sobre él y, en la ficha "Acceso directo" pulsamos en el botón "Opciones avanzadas...", nos aparece una ventana en la que tendremos que seleccionar la casilla de verificación "Ejecutar como administrador". Luego pulsamos en aceptar y ya tendremos preparado el acceso directo para ejecutar en cualquier momento PowerShell con privilegios de Administrador, siempre que nuestro usuario pueda realizarlo o tengamos acceso a un usuario/contraseña que tenga esos privilegios.



- Ejecutar Powershell como administrador desde línea comandos:  
Para realizar esta tarea se utiliza el cmdlet [Start-Process](#).  
→ Para Abrir PowerShell como Administrador utilizamos el comando:

```
PS> Start-Process Powershell -Verb RunAs
```

→ Para ejecutar un *script* de PowerShell como Administrador:

```
PS> Start-Process Powershell -Verb RunAs -ArgumentList "-file c:\scripts\cambionome.ps1"
```

**Para conocer la versión de PowerShell** instalada en nuestro equipo utilizamos el cmdlet [Get-Host](#).

- En la siguiente imagen vemos que se trata de la versión 2.0, era un equipo con Windows Server 2008.

```
Windows PowerShell
Copyright © 2009 Microsoft Corporation. Reservados todos los derechos.

PS C:\Users\vieites> get-host

Name          : ConsoleHost
Version       : 2.0
InstanceId   : 840ab2f7-d8bb-4409-bf6b-8f8b440606c1
UI            : System.Management.Automation.Internal.Host.InternalHostUserInterface
CurrentCulture : es-ES
CurrentUICulture : es-ES
PrivateData   : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
IsRunspacePushed : False
Runspace      : System.Management.Automation.Runspaces.LocalRunspace
```

- En la siguiente imagen vemos que se trata de la versión 5.1, era un equipo con Windows 10 Versión 21H1, tal y como se puede ver al ejecutar el comando [winver](#).

```
Windows PowerShell
PS C:\Users\vieites> Get-Host

Name          : ConsoleHost
Version       : 5.1.19041.1237
InstanceId   : 5b6e67b5-0763-4fc7-b094-7f411a453
UI            : System.Management.Automation.InternalHostUserInterface
CurrentCulture : gl-ES
CurrentUICulture : es-ES
PrivateData   : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
DebuggerEnabled : True
IsRunspacePushed : False
Runspace      : System.Management.Automation.Runspaces.LocalRunspace
```



Otros modos de conocer la versión de PowerShell instalada en nuestra versión de Windows es accediendo a las variables globales:

```
PS> $PSVersionTable
PS> $Host
```

#### 1.4.- Descubrir los cmdlets existentes

Existen muchísimos cmdlets y, en cada versión de Windows PowerShell se incorporan nuevos, igual que se incorporan nuevos al instalar aplicaciones y servicios a nuestro equipo.

Para hacerse una idea de los cmdlets existentes en nuestro puesto de trabajo, utilizaremos el comando:

```
PS> Get-Command
```

Y, para conocer, exactamente el número exacto:

```
PS> (Get-Command).Count
```

```
PS C:\Users\vieites> $PSVersionTable
Name          Value
----          -----
PSVersion     5.1.19041.1320
PSEdition    Desktop
PSCompatibleVersions {1.0, 2.0, 3.0, 4.0...}
BuildVersion  10.0.19041.1320
CLRVersion   4.0.30319.42000
WSManStackVersion 3.0
PSRemotingProtocolVersion 2.3
SerializationVersion 1.1.0.1

PS C:\Users\vieites> $Host
Name          : ConsoleHost
Version       : 5.1.19041.1320
InstanceId   : 4c1b8bfe-e421-4b8e-b3d2-827978eee8a1
UI            : System.Management.Automation.InternalHostUserInterface
CurrentCulture : es-ES
CurrentUICulture : es-ES
PrivateData   : Microsoft.PowerShell.ConsoleHost+ConsoleColorProxy
DebuggerEnabled : True
IsRunspacePushed : False
Runspace      : System.Management.Automation.Runspaces.LocalRunspace
```

Los comandos **NO** son sensibles a mayúsculas/minúsculas.

## 1.5.- Configurar la ejecución de *scripts* de PowerShell

Para configurar en Windows la ejecución de *scripts* PowerShell, antes de nada debemos saber que:

- Los *scripts* de PowerShell son archivos de texto guardados con extensión **.PS1**.
- Para crearlos podemos emplear el propio **NotePad** u otro editor de texto cualquiera o, mejor, el software **PowerShell ISE** que se encuentra ya instalado en los sistemas operativos Windows actuales. También existen herramientas de terceros como **PowerGUI** y, como no, podemos utilizar **Visual Studio Code** con la extensión adecuada.
- Por defecto, para mejorar la seguridad, estos archivos los abre el **NotePad** por lo que, si hacemos doble *click* en uno, o hacemos que se ejecute automáticamente al iniciar el equipo, lo único que ocurre es que se "abren" sin más (se ve su contenido). Si nos interesa, podemos configurar el Sistema Operativo Windows para que los archivos con extensión .ps1 se ejecuten todos por defecto con el programa **Powershell.exe**.
- Además, muchos de los *scripts* que vamos a programar, para que hagan la tarea para lo que están programados, tendremos que ejecutarlos con permisos de **Administrador**. Para que esta tarea no pida confirmación, hay que bajar el nivel de "Notificación acerca de cambios en el equipo", o **UAC (Control de Cuentas de Usuario)**.
- Por último, hay que cambiar la preferencia del usuario para la Directiva de Ejecución de Windows PowerShell, ejecutando el comando **Set-ExecutionPolicy**.
  - Para ver la Preferencia actual para la directiva de ejecución de Windows PowerShell (Fijarse lo que implica la preferencia configurada) :

```
PS> Get-ExecutionPolicy
      → Para cambiar la Preferencia a RemoteSigned:
PS> Set-ExecutionPolicy RemoteSigned
      • También se puede ajustar cómo el PowerShell se comporta cuando se produce un error cuando se ejecuta código o scripts. Para ver el modo actual emplearemos la variable ErrorActionPreference:
PS> $ErrorActionPreference
Continue
      • El modo por defecto vemos que es Continue que muestra los errores y continúa con la ejecución. Si queremos que no muestre los errores y que siga con la ejecución tendríamos que configurarlo en SilentlyContinue.
PS> $ErrorActionPreference = "SilentlyContinue"
```

## 1.6.- Mini-Introducción a la consola de PowerShell

Realizar las siguientes tareas para una primera aproximación al manejo de la consola PowerShell (a partir de ahora PS === PowerShell):

- Ejecutar PS como un usuario normal
- Ejecutar PS como administrador:

```
PS> Start-Process Powershell -Verb RunAs
      • Ejecutar Powershell ISE para la creación de scripts
      • Comprobar la versión de PS que estamos utilizando
```

```
PS> Get-Host
      • Configurar el equipo para ejecutar scripts de PS realizados por nosotros y bajados firmados:
```

```
PS> Get-ExecutionPolicy
      - Restricted           - AllSigned
      - RemoteSigned         - Unrestricted
PS> Set-ExecutionPolicy RemoteSigned
      • Recordar el formato que tienen los nuevos cmdlets de PS.
      • Ejemplo típico con un cmdlet como Get-Date:
```

```
#Actualizar la ayuda
PS> Update-Help
#Pedir Fecha y hora actual
PS> Get-Date
#Pedir Ayuda del comando Get-Date → Analizar parámetros '-full' y '-online'
PS> Get-Help Get-Date
```

```

#Pedir en formato lista todos los datos devueltos por el comando Get-Date
PS> Get-Date | fl *
#Pedir en formato tabla solo los datos day,hour,minute,second
PS> Get-Date | ft day,hour,minute,second
#Guardar en la variable $fecha la salida del comando Get-Date
PS> $fecha = Get-Date
#Ver el tipo de variable que es $fecha.
PS> $fecha.GetType()
#Centrarse en el valor de 'Name'
PS> $fecha.GetType().Name
#Ver los datos guardados en la variable $fecha
PS> $fecha
#Verlos en formato lista
PS> $fecha | fl
#Descubrir todas las Propiedades y Métodos de ese 'objeto' $fecha
PS> $fecha | Get-Member
#Centrarnos sólo en las Propiedades
PS> $fecha | Get-Member -MemberType Property
#Descubrir de todas las propiedades el valor de la propiedad 'Date'
PS> $fecha.Date
#Centrarnos sólo en los Métodos
PS> $fecha | Get-Member -MemberType Method
#Crear una fecha nueva que sean 10 días después de la variable $fecha
PS> $fechaNueva = $fecha.AddDays(10)
#Ver que, efectivamente, es la misma pero 10 días después
PS> $fechaNueva.Date
#Ver que volvemos a recuperar esos 10 días...
PS> ($fechaNueva - $fecha).TotalDays
    • Ver de qué comandos disponemos para trabajar:
#Ver todos los cmdlets instalados en nuestro equipo
PS> Get-Command
#Para ver todos esos comandos página a página
PS> Get-Command | Out-Host -Paging
#Aunque también funciona
PS> Get-Command | more
#Número de cmdlets existentes
PS> (Get-Command).Count
#Cmdlets en cuyo nombre aparece la palabra 'date'
PS> Get-Command -Noun "*date*"
#Todos los cmdlets que tienen de acción 'get'
PS> Get-Command -Verb 'get' | more
#Todos los Alias de cmdlets configurados hasta este momento
PS> Get-Alias | more
#Alias del comando 'rm'
PS> Get-alias -Name rm
#Todos los alias en cuyo nombre existe una 'm'
PS> get-alias | where {$_.Name -Like "*m*"}
#El alias del comando utilizado antes 'where'
PS> Get-Alias -Name where
    • Instalar nuevos módulos de PowerShell :
#Instalar módulo PSWindowsUpdate (como Administrador)
PS> Install-Module -Name PSWindowsUpdate
#Cargar un módulo ya instalado
PS> Import-Module PSWindowsUpdate
#Descubrir los comandos instalados con ese nuevo módulo
PS> Get-Command -Module PSWindowsUpdate
#Listar la lista de todas las actualizaciones de Windows
PS> Get-WindowsUpdate
#Ver si alguna de estas actualizaciones necesita reinicio del sistema
PS> Get-WURebootStatus
#Descargar e instalar todas las actualizaciones de Windows Update
PS> Install-WindowsUpdate -AcceptAll -AutoReboot
#Ver el historial de actualizaciones de Windows
PS> Get-WUHistory
#Desinstalar una actualización de Windows
PS> Remove-WindowsUpdate -KBArticleID KB2267602

```

```
#Instalar software en Windows
→ Acceder a la web de Chocolatey e instalarlo
#Ayuda
$ choco /?
#Buscar el paquete a instalar en la web y de ahí el comando
$ choco install firefox
    • Información del sistema :
#Ver contenido unidad env: (Environment)
PS> Get-ChildItem env:
#Descubrir el nombre del equipo
PS> $env:computername
#Descubrir nombre del usuario que ejecuta el terminal
PS> $env:username
#Descubrir todas las clases WMI
PS> Get-WmiObject -List
#Sólo las que tienen en su nombre la palabra "computer"
PS> Get-WmiObject -List | where { $_.Name -like "*computer*" }
#Ver la información devuelta por la API WMI "Win32_ComputerSystem
PS> Get-WmiObject -Class Win32_ComputerSystem
    • Otros comandos:
#Apagar el equipo
PS> Stop-Computer
#Reiniciar el equipo
PS> Restart-computer
    • Historial de comandos:
#Ver el historial de comandos
PS> Get-History
## → Ctrl+R y buscar coincidencia de comando
## → Ctrl+R y Ctrl+S para moverse
#Borrar el Historial de comandos
PS> Clear-History
```

## 2.- Las unidades de Windows PowerShell

Una unidad Windows PS es la ubicación de algún almacén de datos, como por ejemplo una unidad de sistema de archivos. Pero, los programadores de Windows PS crean más unidades para poder acceder de un modo amigable a todas esas 'unidades'.

```
#Conocer las unidades de Windows PS accesibles
PS> Get-PSDrive
Name      Used (GB)     Free (GB)      Provider      Root
----      -----      -----      -----
Alias
C          14,75        184,58       FileSystem    C:\\
Cert
D
Env
Function
HKCU
HKLM
Variable
WSMan
```

Name	Used (GB)	Free (GB)	Provider	Root
Alias			Alias	
C	14,75	184,58	FileSystem	C:\
Cert			Certificate	\
D			FileSystem	D:\
Env			Environment	
Function			Function	
HKCU			Registry	HKEY_CURRENT_USER
HKLM			Registry	HKEY_LOCAL_MACHINE
Variable			Variable	
WSMan			WSMan	

Veamos una explicación de estas unidades atendiendo a la columna **Provider** :

- **Alias** : son alias que se implementan para invocar cmdlets. Por ejemplo, `del` invoca realmente el cmdlet `Remove-Item`. Se pueden agregar nuevos alias .
- **FileSystem** : son unidades que contienen un sistema de archivos. Estas unidades pueden ser locales o estar en red .
- **Certificate** : es el almacén de certificados del equipo. Estos certificados también están accesibles desde el componente Certificados de la consola MMC (*Microsoft Management Console*) .
- **Environment** : contiene la lista de variables de entorno. Así, la variable de entorno `$env:COMPUTERNAME` nos devuelve el nombre del equipo.

- **Function** : contiene el conjunto de funciones disponibles en PS. Estas funciones contienen código PS que se ejecuta cuando se las invoca. Por ejemplo, cuando se introduce el comando `D:` en PS, se ejecuta en realidad el comando `Set-Location D:` .
- **Registry** : designa el registro de Windows. Como se sabe, el registro está también accesible a través de una interfaz gráfica abriendo el ejecutable `regedit.exe` .
- **Variable** : contiene el conjunto de variables durante el tiempo de vida del proceso PS. Algunas variables se construyen previamente, como `$PSVersionTable` y `$Host` , también se pueden crear otras nuevas.
- **WSMan** : Contiene parámetros `WS-Management` . Estos parámetros se utilizan principalmente cuando se trabaja con la funcionalidad **PS Remoting**.

## 2.1.- Los sistemas de archivos

Los sistemas de archivos permiten almacenar archivos y, por tanto, datos. Estos archivos pueden estar organizados en una o varias carpetas, de manera jerárquica. Veamos algunos cmdlets que nos permiten trabajar con dichos 'sistemas de archivos' :

- **Get-Location** : devuelve el nombre del directorio de trabajo actual, se trata del comando Linux `pwd` .

```
PS C:\Users\Administrador> Get-Location
Path
-----
C:\Users\Administrador
PS C:\Users\Administrador> pwd
Path
-----
C:\Users\Administrador
PS> Get-Alias pwd
 CommandType      Name
-----          -----
 Alias           pwd -> Get-Location
```

- **Set-Location** : se utiliza para cambiar de directorio, se trata del comando `cd` .

```
PS C:\Users\Administrador> cd 'C:\Program Files\' 
PS C:\Program Files> Set-Location ~
PS C:\Users\Administrador> Get-Alias cd
 CommandType      Name
-----          -----
 Alias           cd -> Set-Location
```

- **Get-ChildItem** : permite enumerar el contenido de una carpeta, e indica también los atributos de cada uno de sus elementos (archivos y carpetas). Se trata del equivalente al comando `dir` o al comando `ls` .

→ Mostrar el contenido del directorio de trabajo

```
PS C:\Program Files> Get-ChildItem
 Directorio: C:\Program Files
 Mode                LastWriteTime        Length Name
 ----              -----          ----- 
 d----    08/05/2021     10:34                Common Files
 d----    23/05/2023     12:33                Internet Explorer
 d----    08/05/2021     10:20                ModifiableWindowsApps
 d----    21/12/2022     21:18                Oracle
 d----    21/12/2022     17:00                Windows Defender
 d----    23/05/2023     12:33                Windows Defender Advanced Threat Protection
 d----    23/05/2023     12:33                Windows Mail
 d----    05/11/2022      8:43                 Windows Media Player
 d----    21/12/2022     16:01                Windows NT
 d----    05/11/2022      8:43                Windows Photo Viewer
 d----    08/05/2021     10:34                WindowsPowerShell
```

→ Mostrar el contenido del directorio de trabajo, incluidos los archivos ocultos, filtrados por la extensión de archivo `*.dat`

```
PS C:\Users\Administrador> Get-ChildItem -Path . -Hidden -Filter *.dat
 Directorio: C:\Users\Administrador
 Mode                LastWriteTime        Length Name
 ----              -----          ----- 
 -a-h--   13/01/2024     16:19       786432 NTUSER.DAT
```

- **Get-Item** : permite recuperar el objeto indicando. Este cmdlet es útil en combinación del carácter *pipe* ( | ), que permite redirigir a otro cmdlet el flujo de información del objeto recuperado.

→ Recuperar el conjunto de archivos con extensión \*.txt presentes en el directorio

**C:\Temp\**

```
PS> Get-Item -Path C:\Temp\*.txt
```

→ Copiar el conjunto de archivos con extensión \*.txt presentes en el directorio

**C:\Temp\** al directorio **C:\Temp2\**

```
PS> Get-Item -Path C:\Temp\*.* | Copy-Item -Destination C:\Temp2
```

→ Recuperar las propiedades pertenecientes a un archivo imagen **foto.jpg**

```
PS> Get-Item .\foto.jpg | Format-List *
```

→ Conocer el tamaño en MegaBytes de un archivo imagen **foto.jpg**

```
PS> (Get-Item .\foto.jpg).Length / 1MB
```

- **Get-Content** : permite recuperar el contenido de un objeto, incluso aunque sea un archivo binario.

→ Recuperar el contenido del archivo **hosts**

```
PS> Get-Content C:\Windows\System32\drivers\etc\hosts
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97      rhino.acme.com          # source server
#      38.25.63.10      x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#      127.0.0.1      localhost
#      ::1            localhost
```

→ Recuperar las líneas del archivo **hosts** que tienen la palabra 'localhost'

```
PS> Get-Content C:\Windows\System32\drivers\etc\hosts | 
Where-Object { $_ -match "localhost" }
# localhost name resolution is handled within DNS itself.
#      127.0.0.1      localhost
#      ::1            localhost
```

- **Set-Content** : de manera inversa a **Get-Content** permite escribir o reemplazar datos en un objeto.

→ Como ejemplo simple, vamos a escribir un texto en un archivo

```
PS> Set-Content -Path .\saludo.txt -Value "Hola caracola"
```

```
PS> Get-Content .\saludo.txt
```

```
Hola caracola
```

- **Get-ItemProperty** : obtiene las propiedades de un elemento especificado.

→ Recuperar las propiedades del archivo antes creado **saludo.txt**

```
PS> Get-ItemProperty .\saludo.txt | fl *
```

```
PSPath          : Microsoft.PowerShell.Core\FileSystem::C:\Users\Administrador\Documents\saludo.txt
PSParentPath    : Microsoft.PowerShell.Core\FileSystem::C:\Users\Administrador\Documents
PSChildName    : saludo.txt
PSDrive         : C
PSProvider      : Microsoft.PowerShell.Core\FileSystem
Mode           : -a---
```

```

VersionInfo      : File: C:\Users\Administrador\Documents\saludo.txt
                   InternalName:
                   OriginalFilename:
                   FileVersion:
                   FileDescription:
                   Product:
                   ProductVersion:
                   Debug:    False
                   Patched:  False
                   PreRelease:  False
                   PrivateBuild:  False
                   SpecialBuild:  False
                   Language:

BaseName        : saludo
Target          : {}
LinkType        :
Name            : saludo.txt
Length          : 15
DirectoryName   : C:\Users\Administrador\Documents
Directory       : C:\Users\Administrador\Documents
IsReadOnly      : False
Exists          : True
FullName        : C:\Users\Administrador\Documents\saludo.txt
Extension       : .txt
CreationTime    : 14/01/2024 16:41:24
CreationTimeUtc : 14/01/2024 15:41:24
LastAccessTime  : 14/01/2024 16:41:24
LastAccessTimeUtc : 14/01/2024 15:41:24
LastWriteTime   : 14/01/2024 16:41:24
LastWriteTimeUtc : 14/01/2024 15:41:24
Attributes      : Archive

```

- **New-Item** : permite crear archivos y también carpetas.

→ Crear un archivo vacío

```

PS> New-Item -Path ./cosa.txt -ItemType File
    Directorio: C:\Users\Administrador\Documents
    Mode           LastWriteTime      Length Name
    ----          -----          ----- -----
-a---- 14/01/2024 16:32             0 cosa.txt
PS> Get-Content .\cosa.txt
#No sale nada, pues el archivo está vacío

```

→ Crear una nueva carpeta

```

PS> New-Item ./backups -ItemType Directory
    Directorio: C:\Users\Administrador\Documents
    Mode           LastWriteTime      Length Name
    ----          -----          ----- -----
d---- 14/01/2024 16:59             backups
PS> Get-ChildItem
    Directorio: C:\Users\Administrador\Documents
    Mode           LastWriteTime      Length Name
    ----          -----          ----- -----
d---- 14/01/2024 16:59             backups
-a---- 14/01/2024 16:32             0 cosa.txt
-a---- 14/01/2024 16:41             15 saludo.txt

```

- **Remove-Item** : elimina archivos o carpetas.

→ Eliminar todos los archivos con extensión txt que existen en el archivo de trabajo

```

PS> Remove-Item -Path \*.txt
PS> Get-ChildItem
    Directorio: C:\Users\Administrador\Documents
    Mode           LastWriteTime      Length Name
    ----          -----          ----- -----
d---- 14/01/2024 16:59             backups

```

→ Eliminar un directorio y todo su árbol (subcarpetas y archivos) basta con indicar la carpeta que se ha de eliminar

```

PS> Remove-Item -Path .\backups
PS> Get-ChildItem
#No sale nada, pues el directorio de trabajo está vacío ahora

```

→ Eliminar todo el árbol (subcarpetas y archivos), sin eliminar la carpeta base

```
PS> Remove-Item -Path .\backups\*
```

- **Copy-Item** : permite copiar uno o varios objetos.

→ Copiar un archivo

```
#Creamos un directorio de trabajo
PS> New-Item ./archivos1 -ItemType Directory
    Directorio: C:\Users\Administrador\Documents
      Mode           LastWriteTime         Length Name
      ----           -----              ----- 
d----          14/01/2024       17:23            archivos1
#Copiamos el archivo host al directorio de trabajo
PS> Copy-Item -Path C:\Windows\System32\drivers\etc\hosts ` 
-Destination .\archivos1\
#Creamos también un archivo en el directorio de trabajo
PS> Set-Content -Path .\archivos1\saludo.txt -Value "Hola caracola"
#Comprobamos el contenido del directorio de trabajo
PS> Get-ChildItem -Path .\archivos1\
    Directorio: C:\Users\Administrador\Documents\archivos1
      Mode           LastWriteTime         Length Name
      ----           -----              ----- 
-a---          08/05/2021       10:18            824 hosts
-a---          14/01/2024       17:31            15 saludo.txt
```

→ Copia de un árbol de directorios completo, incluyendo la carpeta raíz

```
#Comprobamos el contenido del directorio de trabajo 'archivos1'
PS> Get-ChildItem -Path .\archivos1\
    Directorio: C:\Users\Administrador\Documents\archivos1
      Mode           LastWriteTime         Length Name
      ----           -----              ----- 
-a---          08/05/2021       10:18            824 hosts
-a---          14/01/2024       17:31            15 saludo.txt
#Copiamos 'archivos1' y todo su contenido en 'archivos2'
PS> Copy-Item -Path .\archivos1\ -Destination .\archivos2 -Recurse
#Comprobamos que el directorio 'archivos2' está creado
PS> Get-ChildItem
    Directorio: C:\Users\Administrador\Documents
      Mode           LastWriteTime         Length Name
      ----           -----              ----- 
d----          14/01/2024       17:31            archivos1
d----          14/01/2024       17:53            archivos2
#Comprobamos que todos los archivos existentes en 'archivos1' se copiaron
PS> Get-ChildItem -Path .\archivos2\
    Directorio: C:\Users\Administrador\Documents\archivos2
      Mode           LastWriteTime         Length Name
      ----           -----              ----- 
-a---          08/05/2021       10:18            824 hosts
-a---          14/01/2024       17:31            15 saludo.txt
```

- **Rename-Item** : permite renombrar un objeto.

→ Renombramos el archivo **hosts** del directorio **.\archivos1** a **hosts\_backup**

```
PS> Rename-Item -Path .\archivos1\hosts -NewName hosts_backup
PS> Get-ChildItem -Path .\archivos1\
    Directorio: C:\Users\Administrador\Documents\archivos1
      Mode           LastWriteTime         Length Name
      ----           -----              ----- 
-a---          08/05/2021       10:18            824 hosts_backup
-a---          14/01/2024       17:31            15 saludo.txt
```

- **Move-Item** : permite mover un archivo o una carpeta a otra ubicación.

→ Mover el archivo **.\archivos1\hosts\_backup** a **.\archivos2**

```
PS> Move-Item -Path .\archivos1\hosts_backup .\archivos2\
[ Si en la carpeta destino existen archivos con el mismo nombre que los que se van a mover, y se desea reemplazarlos, se debe especificar el parámetro -Force ]
```

- **Invoke-Item** : permite abrir un archivo y realizar su acción por defecto (ejecutando la aplicación asociada a la extensión del archivo). Si se trata de un archivo ejecutable, se ejecuta. Para un archivo **.html**, se abre el navegador web asociado por defecto y se muestra el documento.

### 3.- Administración de Discos - Teoría

#### 3.1.- MBR vs GPT

Es posible instalar Windows en un disco manteniendo una de las dos estructuras siguientes.

##### 3.1.1.- MBR (MBR – BIOS)

Tipos de particiones en discos MBR (Master Boot Record) :

**Particiones Primarias** : 4 como máximo si no está creada la partición extendida, en el caso de que se cree ésta solo se pueden tener 3 primarias como máximo (primarias+extendida <= 4).

- Son las particiones en las que se instalan los Sistemas Operativos.

- **Partición Activa** : Partición primaria elegida para arrancar el equipo.

**Partición Extendida** : 1 como máximo. En ella no se pueden guardar archivos directamente, hay que dividirla en Particiones Lógicas (como mínimo una que ocupe todo el espacio).

- **Particiones Lógicas** : Todas las que se quieran. Son las particiones en las que se divide la Partición Extendida.

##### 3.1.2.- GPT (GPT – UEFI)

GPT (GUID Partition Table) está disponible en los ordenadores UEFI. Esta tabla de particiones GPT resuelve las restricciones vinculadas a los discos MBR.

\_Número máximo de 128 particiones, todas "primarias" de hasta 18 EiB.

\_Discos de capacidad superior a 2TiB (tamaño máximo del disco 2 ZiB).

\_Ayuda a proteger el proceso previo al inicio del sistema operativo frente a ataques de *bootkit*.

\_Se crea en cada disco arrancable una partición del sistema ESP (*Extensible Firmware Interface System Partition*) y, en el caso de Windows, también otra llamada MSR (*Microsoft Reserved Partition*) para su buen funcionamiento.

Se podrá convertir un disco MBR en uno GPT utilizando el comando **MBR2GPT.exe** . Este comando tiene dos parámetros interesantes :

→ **/validate** – dpara validar si es factible convertir un disco específico

→ **/convert** – para convertir el disco en cuestión

### 3.2.- Sistemas RAID

#### 3.2.1.- Introducción a los Sistemas RAID

**RAID (Redundant Array Of Independent/Inexpensive Disks)** es un término inglés que hace referencia a un conjunto de discos redundantes independientes/baratos.

Estos dispositivos se utilizan para:

- Aumentar la integridad de los datos en los discos.
- Mejorar la tolerancia a fallos y errores.
- Mejorar el rendimiento.

Oficialmente los sistemas RAID se implementan en 7 configuraciones o niveles: desde el RAID0 al RAID6.

También existen combinaciones de niveles de RAID, las combinaciones más comunes son: RAID10, RAID01, RAID50 y RAID60.

A nivel comercial podemos encontrar los siguientes: RAID0, RAID1, RAID1E, RAID3, RAID5, RAID5EE, RAID6, RAID10, RAID50 y RAID60 .

También existen muchos sistemas RAID propios de cada fabricante (RAID Z de Sun).

Antes de guardar la información, en algunos sistemas RAID, ésta se divide en porciones de un tamaño fijo y, normalmente, configurable denominado **banda (stripe)**. El tamaño habitual de la banda será de 64KB o 128KB.

Por último, indicar que los sistemas RAID son normalmente implementados con discos de la misma capacidad, pero no es obligatorio que así sea.

### 3.2.2.- Discos Básicos – Discos Dinámicos

#### 3.2.2.1.- Discos Básicos

El disco se puede dividir en Particiones MBR o GPT.

#### 3.2.2.2.- Discos Dinámicos

Ahora el disco se divide en Volúmenes.

- **Volumen Simple** : Uno o varios trozos de un solo disco duro.
- **Volumen Distribuido** : Varios trozos de varios discos duros:  $\geq 2$  y  $< 32$  Discos.  
Se llenan secuencialmente.
- **Volumen Seccionado** : Varios trozos iguales de varios discos duros:  $\geq 2$  y  $< 32$  Discos.  
Los datos se dividen en Bandas de 64 KB (RAID 0).  
No se puede hacer con la del Sistema Operativo.  
No es extensible.
- **Volumen Reflejado** : Espejos de dos trozos iguales de dos discos.
- **Mirroring** o RAID 1 (Se puede hacer con la del Sistema Operativo - Reflejar).  
No es extensible.
- **Volumen RAID 5** (Solo Windows Server):  $\geq 3$  Discos y  $< 32$  Discos.  
No se puede hacer con la del Sistema Operativo.  
No es extensible.

### 3.2.3.- Espacios de Almacenamiento

A partir de un *Pool* de Discos se crearán uno o más Discos Virtuales. Los tipos posibles de Discos Virtuales son:

- **Simple** : Se guardan los datos como en un RAID0.
- **Mirror** : Se crea algo parecido a un RAID1, o un RAID1E si los discos son impares. También permite la realización de dos copias (eficiencia de almacenamiento del 50%) o tres copias (eficiencia de almacenamiento del 33%). Se necesitan dos discos para prevenir la avería de un disco y, como mínimo, tres discos para prevenir la avería simultánea de dos discos.
- **Parity** : Permite la paridad simple, creándose algo parecido a un RAID5, o paridad doble, creando algo parecido a un RAID6. Se necesitan, como mínimo, tres discos para crear un Disco Virtual de Paridad Simple y asegurar la avería de un disco y, como mínimo, cuatro discos para asegurar la avería simultánea de dos discos.

Una vez seleccionado el modo de almacenamiento, tendremos que seleccionar y tipo de **provisionamiento**:

- **Thin** : El disco virtual va cogiendo espacio del *Pool* a medida que es necesario, hasta llegar a ese tamaño máximo definido.
- **Fixed** : Se reserva para el disco virtual ese espacio prefijado, aunque luego no se emplee (como un particionamiento típico).

## 3.3.- Sistemas de Archivos en Windows

### 3.3.1.- FAT (*File Allocation Table* - Táboa de Asignación de Ficheiros)

- Cada entrada de la tabla corresponde a un cluster, en cada entrada se indica: Si el cluster está vacío; si está defectuoso; cuál es el cluster siguiente que tiene parte del archivo; si es el último cluster que tiene información del archivo.
- El tamaño de la FAT depende del sistema de archivos empleado:
  - FAT12 implica que da dirección 2<sup>12</sup> clusters
  - FAT16 implica que da dirección 2<sup>16</sup> clusters
  - FAT32 implica que da dirección 2<sup>28</sup> clusters
- En teoría, FAT32 soporta particiones más grandes de 2 TB. Sin embargo, debido a una limitación en el BOOT, que utiliza una variable de 32 bits para numerar los sectores físicos presentes en el disco duro, el límite real de una partición en FAT32 es de 2 TB.

- Las utilidades incluidas en Windows permiten crear y formatear unidades FAT32 de tamaño máximo 32 GB. Sin embargo, pueden acceder en modo lectura y escritura a volúmenes FAT32 de hasta 2 TB creados usando otras utilidades.
- Tamaño máximo de un archivo:
  - Partición FAT 12 : 32 MB
  - Partición FAT 16 : 2 GB
  - Partición FAT 32 : 4 GB

### 3.3.2.- NTFS

- Hay tres versiones de NTFS:
  - v1.2 ou v.4.0 en NT 3.51 e NT 4
  - v3.0 ou v5.0 en Windows 2000
  - v3.1 ou v5.1 en Windows XP y Windows 2003 Server (y superiores)
- Utiliza 64 bits para direccionar bloques, por lo que el tamaño máximo de la unidad será de 16 HexaBytes.
- No se recomienda utilizar NTFS en un volumen de menos de 400 MB, debido a la sobrecarga de espacio que implica NTFS.
- Implementa ACLs: Listas de Control de Acceso a archivos y directorios.
- *Journaling*: Registro de transacciones.
- Nombres de archivos de hasta 256 caracteres
- Compresión individual: compact.exe
- Cifrado de archivos (EFS): cipher.exe
- Soporta Cuotas de disco a partir de la v5.0 (sobre él se implementan [FSRM](#)).

### 3.3.3.- exFAT (*Extended File Allocation Table*)

- Sistema de archivos especialmente adaptado para memorias flash presentado con Windows Embedded CE 6.0. exFAT se utiliza cuando se decide no emplear el sistema de archivos NTFS debido a la sobrecarga de las estructuras de datos.
  - Límite teórico para el tamaño de un archivo de  $2^{64}$  bytes (16 Exabytes) mejorando el límite de FAT32 de 4GB.
  - Tamaño de *cluster* de hasta  $2^{255}$  bytes (el implementado por defecto es de 32MB).
  - Mejoras en el rendimiento de la asignación de espacio libre gracias a la introducción de un *free space bitmap*.
  - Soporte para más de 1000 ficheros en un solo directorio.
  - Soporte para ACLs, pero en Windows no se implementan.
  - No implementa *journaling*.

### 3.3.4.- ReFS (*Resilient File System*)

- Aparece con el SO Windows Server 2012 R2.
- Este sistema de archivos viene a ofrecer prácticamente un tamaño ilimitado en los archivos y en los directorios (superior a petabytes) – “Resilient : Resistente – Elástico”.
- También incrementa la resistencia a la aparición de errores y elimina la necesidad de emplear herramientas de búsqueda de errores (chkdsk.exe).
- Por otro lado, ReFS no soporta muchas de las características NTFS como: compresión de archivos, EFS o Cuotas (ni FSRM).
- Por último, comentar que ReFS no podrá ser leído por sistemas operativos “antiguos” como Windows Server 2012 y Windows 8.

## 3.4.- Resumen de cmdlets de gestión de discos y sistemas RAID

### 3.4.1.- Gestión de discos

- **Get-Disk** : devuelve la lista de discos visibles por el sistema operativo.
- **Get-PhysicalDisk** : devuelve una lista de los discos físicos únicamente.
- **Get-Partition** : devuelve una lista de todas las particiones visibles.
- **Get-Volume** : devuelve una lista de todos los volúmenes.

- **Resize-Partition**: permite redimensionar el tamaño de una partición.
- **Get-PartitionSupportedSize**: recupera los tamaños mínimo y máximo soportados por la partición.
- **New-Partition**: crea una nueva partición.
- **Format-Volume**: permite dar formato a una unidad.
- **Add-PartitionAccessPath**: agrega una letra de unidad a una partición.
- **Remove-PartitionAccessPath**: elimina la letra de unidad de una partición.
- **Initialize-Disk**: permite inicializar un disco.
- **Set-Disk**: permite realizar acciones sobre el disco (cambiar la tabla de particiones, configurar un disco fuera de conexión o de solo lectura).
- **Clear-Disk**: elimina la configuración del disco (iy los datos!).
- **Optimize-Volume**: permite optimizar el rendimiento de las unidades (desfragmentación de los datos, TRIM).
- **Repair-Volume**: permite buscar y corregir los errores del sistema de archivos en las unidades.
- **Remove-Partition**: elimina una partición.
- **New-VHD**: crea un nuevo disco virtual.
- **Mount-VHD**: monta un disco duro virtual.
- **Optimize-VHD**: compacta los datos de un disco duro virtual.
- **Convert-VHD**: convierte un disco duro virtual de tamaño fijo a extensión dinámica.
- **Resize-VHD**: permite redimensionar el tamaño máximo de un disco duro virtual.
- **Cleanmgr.exe**: Herramienta para liberar espacio en disco (papelera, temporales...).

### 3.4.2.- Storage Spaces

- **Get-StorageSubSystem**: devuelve los objetos StorageSubsystem.
- **New-StoragePool**: crea un nuevo storage pool utilizando discos físicos.
- **Get-StoragePool**: devuelve un storage pool específico, o un conjunto de ellos.
- **Remove-StoragePool**: elimina un storage pool y sus VirtualDisk asociados.
- **Set-StoragePool**: modifica las propiedades de un storage pool específico.
- **Add-PhysicalDisk**: añade un physical disk a un storage pool específico.
- **New-Volume**: crea un volumen con un sistema de archivos específico.
- **Remove-PhysicalDisk**: elimina un physical disk de un storage pool específico.
- **New-VirtualDisk**: crea un nuevo virtual disk en un storage pool específico.
- **Get-VirtualDisk**: devuelve una lista de los VirtualDisk existentes.

## 4.- Ejemplos prácticos Administración de Discos

**4.1.-** Agrega un disco duro de 200GB a un sistema Windows y partícnalo en modo gráfico del siguiente modo utilizando el Administrador de Discos de Windows:

- Partición Primaria de 100GB, letra V, etiqueta “DATOS” y formateada en NTFS.
- Partición Extendida que ocupe el resto del disco.
- Partición Lógica de 50GB, letra Z, etiqueta “Almacen” y formateada en NTFS.
- Partición Lógica que ocupe el resto del disco, etiqueta “Intercambio”, montada en el directorio C:\Cosas y formateada en exFAT.

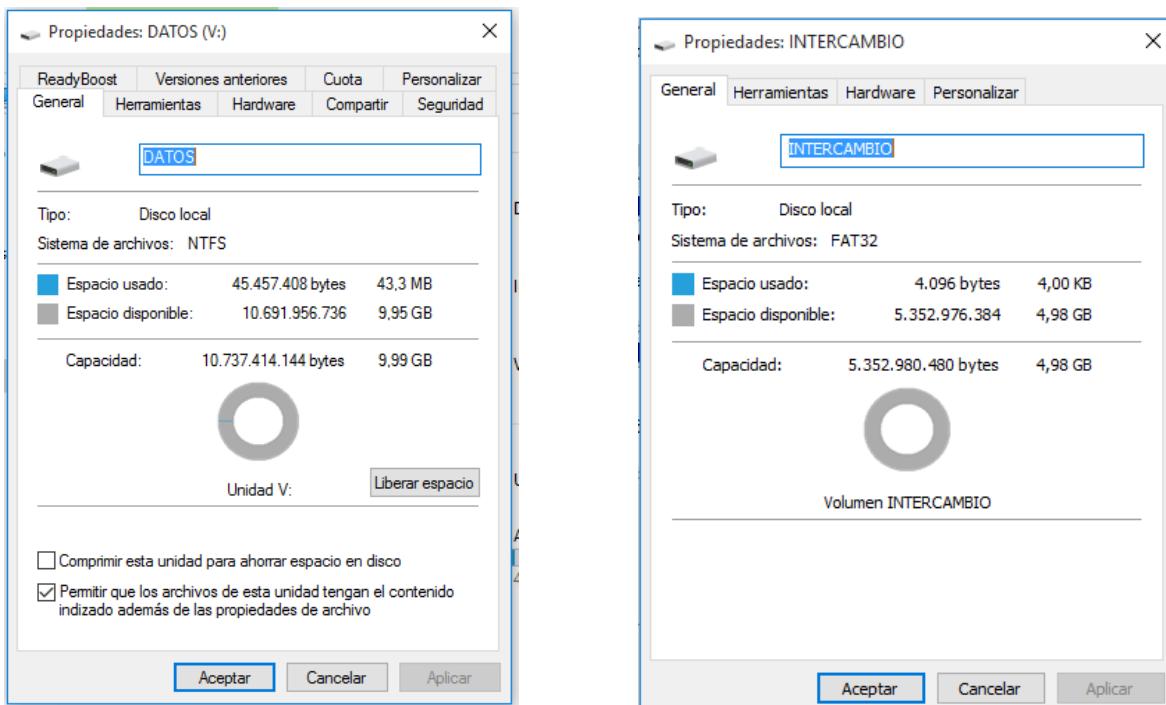
 Disco 1 Básico 20,00 GB En pantalla	DATOS (V:) 10,00 GB NTFS Correcto (Partición primaria)	Almacen (Z:) 5,00 GB NTFS Correcto (Unidad lógica)	INTERCAMBIO 5,00 GB FAT32 Correcto (Unidad lógica)
--	--	--	--

Gráficamente NO hay forma de dejar justo así el particionamiento... Tendremos que hacer algo equivalente sin llegar a realizar la Extendida con las lógicas (sólo primarias):

- Partición Primaria de 100GB, letra V, etiqueta “DATOS” y formateada en NTFS.
- Partición Extendida que ocupe el resto del disco.
- Partición Lógica Primaria de 50GB, letra Z, etiqueta “Almacen” y formateada en NTFS.

- Partición Lógica Primaria que ocupe el resto del disco, etiqueta "Intercambio", montada en el directorio C:\Cosas y formateada en exFAT.

<b>Disco 0</b> Básico 120,00 GB En pantalla	<b>Reservado para el sistema</b> 500 MB NTFS Correcto (Sistema, Activo, Partición primaria)	(C) 119,51 GB NTFS Correcto (Arranque, Archivo de paginación, Volcado, Partición primaria)
<b>Disco 1</b> Básico 20,00 GB En pantalla	<b>DATOS (V:)</b> 10,00 GB NTFS Correcto (Partición primaria)	<b>Almacen (Z:)</b> 5,00 GB NTFS Correcto (Partición primaria)



**4.1.A.-** Realiza exactamente la misma tarea del ejercicio anterior pero utilizando la herramienta Diskpart.

Creamos el archivo "particionar.bat":

```
@ECHO OFF
REM Creamos el directorio donde irá montada la partición lógica
MKDIR C:\COSAS
REM Ejecutamos diskpart pasándole como parámetro el archivo de respuestas
DISKPART /s C:\Scripts\particiones.txt
```

Creamos el archivo "particiones.txt":

```
REM Seleccionamos el disco a particionar
SELECT DISK 1
REM Borramos todo el disco
CLEAN
REM Creamos una partición primaria de 10GB
CREATE PARTITION PRIMARY SIZE=10240
REM Le asignamos la letra V
ASSIGN LETTER=V
REM La formateamos en NTFS y con etiqueta DATOS
FORMAT FS=NTFS LABEL="DATOS" QUICK
REM Creamos partición extendida con el resto del disco
CREATE PARTITION EXTENDED
REM Creamos partición lógica de 5GB
CREATE PARTITION LOGICAL SIZE=5120
REM Le asignamos la letra Z
ASSIGN LETTER=Z
```

```

REM La formateamos en NTFS y con etiqueta Almacen
FORMAT FS=NTFS LABEL="Almacen" QUICK
REM Creamos partición lógica con el resto de la extendida
CREATE PARTITION LOGICAL
REM La montamos en C:\Cosas
ASSIGN MOUNT=C:\Cosas
REM La formateamos en FAT32 y con etiqueta "Intercambio"
FORMAT FS=FAT32 LABEL="Intercambio" QUICK
REM Salimos
EXIT

```

#### 4.1.B- Realiza exactamente la misma tarea del ejercicio anterior pero utilizando cmdlets :

#Miramos el número identificativo del disco de trabajo

#Los discos se empiezan a numerar en el 0...

PS> Get-Disk

##### Contenido del script:

```

#Disco de trabajo
$numDisko = 1
#Borramos contenido del disco
Clear-Disk -Number $numDisko -RemoveData -Confirm:$false 2>$null
#Inicializamos el disco
Initialize-Disk -Number $numDisko -PartitionStyle MBR
#Partición primaria 100GB - V: - DATOS - NTFS
New-Partition -DiskNumber $numDisko -DriveLetter V -Size 100GB
Format-Volume -DriveLetter V -FileSystem NTFS -NewFileSystemLabel "DATOS"
#Partición extendida que ocupe resto del disco
New-Partition -DiskNumber $numDisko -MbrType Extended -UseMaximumSize
##Partición lógica 50GB - Z: - ALMACEN - NTFS
New-Partition -DiskNumber $numDisko -DriveLetter Z -Size 50GB
Format-Volume -DriveLetter Z -FileSystem NTFS -NewFileSystemLabel "ALMACEN"
##Partición lógica RestoExtendida - C:\Cosas - INTERCAMBIO - exFAT
####Creamos el directorio C:\Cosas si este NO existe
$dCosas = "C:\Cosas"
if (!(Test-Path $dCosas)) {
    New-Item $dCosas -ItemType Directory
}
else {
    Write-Host "$dCosas ya existe"
}
#La partición creada es la 3 del disco 1
#Lo comprobamos con "Get-Partition -DiskNumber $numDisko"
New-Partition -DiskNumber $numDisko -UseMaximumSize
Get-Partition -DiskNumber $numDisko -PartitionNumber 3 | `
Format-Volume -FileSystem exFAT -NewFileSystemLabel "INTERCAMBIO"
Add-PartitionAccessPath -DiskNumber $numDisko -PartitionNumber 3 -AccessPath $dCosas

```

#### 4.2.- Pequeños ejercicios del uso de cmdlets para la administración de discos:

→ Recuperar los discos conectados al puesto de trabajo

PS> Get-Disk | Format-List

→ Mostrar el conjunto de particiones presentes en el puesto de trabajo

PS> Get-Partition | fl

→ Redimensionar la partición número 3 del disco 0 - se redimensiona a 100GB

PS> Resize-Partition -DiskNumber 0 -PartitionNumber 3 -Size 100GB

→ Recuperar los valores de los tamaños mínimo y máximo soportados por una partición

PS> Get-PartitionsupportedSize -DiskNumber 0 -PartitionNumber 3

→ Redimensionar una partición a su tamaño máximo

PS> \$size = Get-PartitionsupportedSize -DiskNumber 0 -PartitionNumber 3

PS> Resize-Partition -DiskNumber 0 -PartitionNumber 3 -Size \$size.SizeMax

→ Crear una partición con el mayor tamaño posible (se le asigna la siguiente letra libre)

PS> New-Partition -DiskNumber 0 -UseMaximumSize

→ Crear una partición con un tamaño y una letra de unidad concretos  
 PS> New-Partition -DiskNumber 0 -Size 100GB -DriveLetter F

→ Formatear una partición (si no se indica, el FS por defecto será NTFS)  
 PS> \$partition = Get-Partition -DiskNumber 0 -PartitionNumber 4  
 PS> Format-Volume -Partition \$partition -Confirm:\$false |  
 Format-List DriveLetter,DriveType,FileSystem,FileSystemLabel,  
 HealthStatus,OperationalStatus,Path,Size,SizeRemaining

→ Crear y formatear una partición en una sola línea de comando  
 PS> New-Partition -DiskNumber 0 -UseMaximumSize -DriveLetter E |  
 Format-Volume -NewFileSystemLabel "DATOS" -Confirm:\$false |  
 Format-List DriveLetter,DriveType,FileSystem,FileSystemLabel,  
 HealthStatus,OperationalStatus,Path,Size,SizeRemaining

→ Asignar una letra de unidad automáticamente  
 PS> Add-PartitionAccessPath -DiskNumber 0 -PartitionNumber 4 -AssignDriveLetter

→ Asignar una letra de unidad específica a una partición  
 PS> Add-PartitionAccessPath -DiskNumber 0 -PartitionNumber 4 -AccessPath E:\

→ Eliminar la letra de unidad de una partición  
 PS> Remove-PartitionAccessPath -DiskNumber 0 -PartitionNumber 4 -AccessPath [E:\](#)

→ Inicializar un disco (por defecto, -PartitionStyle es GPT)  
 PS> Initialize-Disk -Number 1  
 PS> Get-Disk -Number 1 | Format-List DiskNumber,PartitionStyle,  
 ProvisioningType,OperationalStatus,HealthStatus,BusType,FirmwareVersion,  
 FriendlyName,Guid,Manufaturer,Model,Number,NumberOfPartition,SerialNumber,Size

→ Ver si es necesario inicializar un disco, esto se verá si la propiedad -PartitionStyle es RAW  
 PS> Get-Disk | Where-Object { \$\_.PartitionStyle -eq "RAW" } | Select-Object  
 Number,Model,Size,PartitionStyle

→ Inicializar todos los discos no iniciados en una única línea de comando  
 PS> Get-Disk | Where-Object { \$\_.PartitionStyle -eq "RAW" } | Initialize-Disk

→ Cambiar la tabla de particiones del disco a MBR  
 PS> Set-Disk -Number 1 -PartitionStyle MBR

→ Al revés, convertir un disco de MBR a GPT (Esta operación NO puede realizarse sobre un  
 disco de sistema)  
 PS> Set-Disk -Number 1 -PartitionStyle GPT

→ Dejar un disco fuera de conexión  
 PS> Set-Disk -Number 1 -IsOffline:\$true

→ Pasar el disco de "fuera de conexión" a "en línea"  
 PS> Set-Disk -Number 1 -IsOffline:\$false

→ Limpiar un disco que posee particiones y datos  
 PS> Clear-Disk -Number 1 -RemoveData -Confirm:\$false

→ Optimizar un volumen determinado, por ejemplo, realizando una desfragmentación (El  
 cmdlet, dependiendo del tipo de volumen y del tipo de disco en el que está dicho volumen,  
 sabe qué función realizar sin indicarle nada)

[ Get-Partition para descubrir las particiones existentes ]

```
PS> Optimize-Volume -DriveLetter V
```

→ Escanear y reparar una partición (chkdsk.exe), desmontando primeramente el volumen  
 PS> Repair-Volume -DriveLetter V -OfflineScanAndFix

→ Escanear y reportar únicamente los errores (NO reparar)  
 PS> Repair-Volume -DriveLetter V -Scan

→ Eliminar una partición

[ Recordar que si queremos **simular** el resultado de algún comando crítico como este, esos cmdlets suelen tener el parámetro **-WhatIf** ]

```
PS> Remove-Partition -DriveLetter Z -Confirm:$false
```

**4.3.-** Para realizar el siguiente ejercicio prácticos, tendrás que añadir varios discos a una máquina virtual en concreto. Puedes utilizar el siguiente *script* (revisa su código aunque no entiendas mucho ahora cómo funciona).

```
#Script que agrega discos duros a VirtualBox
#Se utiliza la herramienta VboxManage.exe que puede ejecutar cualquier usuario del sistema
```

```
#Configuramos la salida por consola UTF8 y recuerda guardar el archivo como UTF8 sin BOM
[Console]::OutputEncoding = [System.Text.Encoding]::UTF8

#Buscamos la máquina a la que queremos agregar los discos
$maquinas = C:\'Program Files'\Oracle\VirtualBox\VBoxManage.exe list vms
#$maquinas
# La salida de este comando tiene la forma siguiente :
##"Windows 2019 08-01-2020" {16718d20-ac52-494f-83da-c579e389bd76}
##"Windows 10 08-01-2020" {e880ecac-9f61-451b-a987-d7b81b3f4dba}
#Creamos una lista de máquinas:
$listamaquinas = @()
$i = 1
Write-Host "Las máquinas existentes en tu equipo son : "
foreach ($maquina in $maquinas) {
    $m = $maquina.split()[0]
    $listamaquinas += $m
    Write-Host " t$i.-$m"
    $i += 1
}
$numMaquina = Read-Host "`nIndica en qué máquina quieras añadir los discos (Introduce número) "
$maquinaTrabajo = $listamaquinas[$numMaquina-1]
if ( $maquinaTrabajo -eq $null )
{
    Write-Host "Esa máquina NO existe"
    Exit
}
#$maquinaTrabajo
#$maquinaTrabajo = "Windows 2019 08-01-2020"

#Directorio archivos máquina de trabajo
$directorioMaquina = $(C:\'Program Files'\Oracle\VirtualBox\VBoxManage.exe ` 
showvminfo $maquinaTrabajo) | Select-String -Pattern "^Config file" | Out-String
$directorioMaquina = $($directorioMaquina.split('`')[ -2..-1] -join ":").Trim()
$directorioMaquina = $($directorioMaquina.split('`') | Select-Object -SkipLast 1) -join "\"
#$directorioMaquina
#$directorioMaquina = "D:\MaquinasVirtuales\vieites\Windows 2019 08-01-2020"

#Número de discos a crear
[int]$numDiscos = Read-Host "`nNº de discos que quieres crear"
#$numDiscos
#$numDiscos = 6
```

```

#Tamaño discos en GB
[int]$tamDiscos = Read-Host "`nTamaño de los discos a crear en GB"
$tamDiscosMiB = 1024 * $tamDiscos
#$tamDiscosMiB

#Tipo de controladora, a crear si fuese necesario
$tipoControladora = 'SAS'
Write-Host "`nCrearemos $numDiscos discos de $tamDiscos GiB conectados por $tipoControladora"
Write-Host "en la máquina $maquinaTrabajo."

#Agregamos la controladora SAS a la máquina
C:\'Program Files'\Oracle\VirtualBox\VBoxManage.exe ` 
storagectl $maquinaTrabajo --name $tipoControladora --add $tipoControladora
#Configuramos la controladora SAS para añadirle el número de discos nuevos deseados
C:\'Program Files'\Oracle\VirtualBox\VBoxManage.exe ` 
storagectl $maquinaTrabajo --name $tipoControladora --portcount $numDiscos
#Creamos un array para los discos
$numCadaDisco = (1..$numDiscos)

#Creamos los discos
foreach ($n in $numCadaDisco) {
    #Generamos nombre del disco del modo: $directorioMaquina\HD1-250GB.vdi
    $nombreDisco = $directorioMaquina + '\HD' + $n + '-' + [string]$tamDiscos + 'GB.vdi'
    #Agregamos disco
    C:\'Program Files'\Oracle\VirtualBox\VBoxManage ` 
        createmedium disk --filename $nombreDisco --size $tamDiscosMiB --format VDI
    #Conectamos disco a la controladora
    C:\'Program Files'\Oracle\VirtualBox\VBoxManage ` 
        storageattach $maquinaTrabajo --storagectl $tipoControladora ` 
        --port $n --device 0 --type hdd --medium $nombreDisco
}

```

**4.4.-** Crear un servidor de archivos empleando la tecnología de Windows de “Espacios de almacenamiento” (**Storage Spaces**). Se debe crear un sistema con 6 discos duros, cinco de ellos formarán un sistema con paridad y el sexto disco quedará en [Hot-Spare](#) para una posible avería.

→ Una vez creado el *Pool* de Discos, se crearán dos Discos Virtuales y, en cada uno de ellos un volumen que ocupe todo el espacio disponible del Disco, formateados en NTFS y con el nombre identificativo correspondiente:

Disco Virtual	Tamaño	Tipo	Letra	Volumen	FS
RAPID	200GB	RAID0	W:	BORRASE	NTFS
SECURE	Resto (+/- 1,8TB)	RAID5	V:	DATOS	NTFS

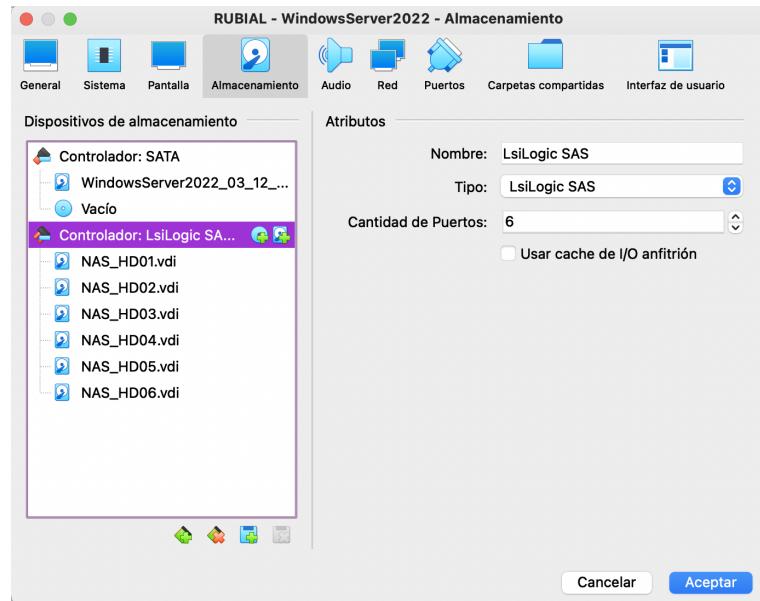
→ Forzar la avería de un disco y comprobar que el que estaba en *Hot-Spare* sustituye automáticamente al averiado.

→ Luego, una vez que el sistema ya tiene restaurada la paridad, forzar la avería de otro disco. Comprobar que el volumen sigue operativo y que el usuario puede acceder a los datos.

### A.- Se añaden 6 discos de 500GB

Se apaga el equipo y, accediendo a **Almacenamiento** de la máquina virtual, se Añade un nuevo controlador de almacenamiento, en este caso optamos por añadir el controlador **SAS**.

Una vez tenemos ese nuevo controlador añadimos a él 6 nuevos discos de 500GB con tamaño dinámico.



### B.- Se crea un *pool* de 5 discos y 1 en *Hot Spare*.

Para ello accedemos al Panel del **Administrador del Servidor** donde vemos la herramienta **Servicios de archivos y de almacenamiento**

Seleccionamos **Grupos de almacenamiento** y vemos que aparece una página con tres apartados: Grupos de almacenamiento, Discos físicos y Discos Virtuales.

Vemos que, de momento, solo existe el Grupo de almacenamiento **Primordial** donde se agrupan los discos disponibles para agregar a un *pool*.

Como es de esperar, en la sección de **Discos Físicos** vemos los 6 discos SAS de 500GB añadidos al Servidor **rubial**.

**GRUPOS DE SERVIDORES Y ROLES**  
Roles: 1 | Grupos de servidores: 1 | Servidores en total: 1

Servicios de archivos y de almacenamiento	
<span style="color: green;">●</span> Estado	1
<span style="color: green;">●</span> Eventos	
<span style="color: green;">●</span> Rendimiento	
<span style="color: green;">●</span> Resultados de BPA	

Servicio	
<span style="color: green;">●</span> Estado	
<span style="color: green;">●</span> Evento	
<span style="color: red;">●</span> Servicio	2
<span style="color: red;">●</span> Rendir	
<span style="color: red;">●</span> Resultado	

The screenshot displays the Windows Storage Management console interface. On the left navigation bar, 'Grupos de almacenamiento' is selected. The main area has two tabs: 'GRUPOS DE ALMACENAMIENTO' and 'DISCOS FÍSICOS'. The 'GRUPOS DE ALMACENAMIENTO' tab shows a single storage pool named 'Windows Storage (1)' with one primary disk (Primordial). The 'DISCOS FÍSICOS' tab lists six physical disks, all labeled 'VBOX HARDDISK (rubial)' with 500 GB capacity, connected via SAS and integrated controllers. The 'DISCOS VIRTUALES' tab indicates 'No hay datos relacionados...'.

Para crear el *pool* de los 6 discos, 5 de ellos para datos y 1 de reserva activa (*hot spare*) se podría utilizar perfectamente esta herramienta gráfica pero, tal y como se hizo en ejercicios de clase, realizaremos un pequeño *script* de Powershell. El *script* tendrá el siguiente contenido:

```
#Script Powershell que crea un Pool de 6 discos SAS de 500GB uno de ellos como hot spare#
$storagePool = "poolRubial"
#Discos que podemos agregar al Pool
$discos = Get-PhysicalDisk -CanPool $true
#Necesitamos conocer la propiedad UniqueID de cada disco
#Recogemos el elemento StorageSubSystem para crear sobre él un nuevo StoragePool
$SSS = Get-StorageSubSystem
#Creamos un nuevo Storage Pool con los discos 0 a 4
New-StoragePool -FriendlyName $storagePool -StorageSubSystemUniqueId $SSS.UniqueId ` 
-PhysicalDisks ($discos | where {$_.DeviceId -in 0..4})
#El disco 5 lo añadimos como Hot Spare
$disco5 = $discos | where {$_.DeviceId -eq 5}
Add-PhysicalDisk -StoragePoolFriendlyName $storagePool -PhysicalDisks $disco5 -Usage HotSpa
#####Ya está el Pool de almacenamiento creado#####

```

Antes de ejecutar el *script* analiza la salida del comando `Get-PhysicalDisk -CanPool $true` y comprueba el **Número** que identifica los discos pues no siempre va de 0 a 5, hay veces que va de 1 a 6 y, si es así, tendrías que adaptar el *script* anterior.

Una vez ejecutado el *script*, la Herramienta gráfica debería tener el siguiente aspecto.

The screenshot shows two main windows from the Windows Storage Management snap-in:

- GRUPOS DE ALMACENAMIENTO**: Shows one group named "poolRubial" which is a "Grupo de almacenamiento" (Storage space) with a capacity of 2,93 TB.
- DISCOS VIRTUALES**: Shows a message stating "No existen discos virtuales relacionados." and "Para crear un disco virtual, inicie el Asistente para nuevo disco virtual."
- DISCOS FÍSICOS**: Shows six physical disks (VBOX HARDDISK) each with 500 GB capacity, connected via Integrated bus and Device 22. They are listed under the poolRubial storage space.

Eso quiere decir que se deberían de tener  $5 \times 500\text{GB} \leq 2500\text{GB}$  para crear los discos virtuales. Pero en la imagen anterior vemos que la Herramienta gráfica nos indica que el Espacio disponible es de 2,93TB, dato erróneo pues está sumando también el disco configurado como Reserva activa. Luego, con los Discos Virtuales creados comprobaremos que los 500GB de ese disco siempre permanecerán ahí, que no permite el sistema utilizarlos ahora para crear ningún Disco Virtual tal y como era de esperar.

C.- Creamos los dos Discos Virtuales que necesitamos:

**Rapid** : Tipo Simple (RAID 0) de 200GB, por lo que quedarán aún 2300GB para utilizar en otros discos.

**Secure** : Tipo Parity de Paridad Simple (RAID 5) que, con esos 2300GB harán un disco virtual de aproximadamente de 1,8TB (supongo para este cálculo aproximado que el software de Storage Spaces dividirá ese espacio de 2300GB entre los 5 discos y luego, uno de esos espacios efectivos lo utilizará para paridad – 460GB – quedando así unos 1840GB útiles).

Para crear estos dos Discos Virtuales también se emplearán los cmdlets de PowerShell pues será más inmediato que la Herramienta gráfica. El *script* que los creará tendrá esta codificación:

```
#Script Powershell que crea dos Discos Virtuales sobre el poolRubial#
$storagePool = "poolRubial"
#Simple (RAID0) de 200GB - Rapid - BORRASE - W:
New-VirtualDisk -StoragePoolFriendlyName $storagePool -ResiliencySettingName Simple ` 
-Size 200GB -ProvisioningType Fixed -FriendlyName "Rapid"
#Parity (RAID5) del resto del Pool - Secure - DATOS - V:
New-VirtualDisk -StoragePoolFriendlyName $storagePool -ResiliencySettingName Parity ` 
-UseMaximumSize -ProvisioningType Fixed -FriendlyName "Secure"
##### Ya están los Virtual Disks creados#####
```

Ahora la Herramienta gráfica de Grupos de Almacenamiento tiene la siguiente apariencia:

The screenshot shows three main windows from the Storage Management interface:

- GRUPOS DE ALMACENAMIENTO**: Shows one group named "poolRubial" which is a storage pool for the "rubial" volume.
- DISCOS VIRTUALES**: Shows two virtual disks: "Rapid" (Simple, Fijo, 200 GB, 200 GB) and "Secure" (Parity, Fijo, 1,79 TB, 1,79 TB).
- DISCOS FÍSICOS**: Shows two physical disks: "VBOX HARDDISK (rubial)" (500 GB, SAS, Automático) and "VBOX HARDDISK (rubial)" (500 GB, SAS, Automático).

En ella podemos ver los dos Discos Virtuales creados y comprobar ahora si todo cuadra con nuestros cálculos teóricos. Recuerda que se comenzó con 6 discos de 500GB, estos suman 3000GB de almacenamiento bruto. Uno de esos discos quedó como Reserva Activa que podemos ver en la imagen anterior como los 507GB del Espacio disponible aún del **poolRubial**. Así que nos quedarían ahora unos 2500GB. De ellos, 200GB pertenecen al Disco Virtual **Rapid** que, como es un RAID0 no desperdicia nada en seguridad. Ahora se tendrán unos 2300GB disponibles para crear un RAID5 de 5 discos, por lo que es fácil calcular la capacidad útil del Disco Virtual **Secure** creado haciendo la operación  $(2300/5)*4 = 1840\text{GB}$ , ya que el espacio equivalente a un disco se pierde para el almacenamiento de la paridad. Se ve que este cálculo teórico es muy parecido a los 1,79TB que indican de capacidad efectiva.

Se pueden observar los Discos Virtuales creados en la herramienta de Administración de discos de la consola Administración de equipos.

Ahora queda inicializar los discos, crear las particiones, instalar el sistema de archivos y asignar las letras de acceso y un nombre a cada volumen.

The screenshot shows the Disk Management console with the following details:

- Administración del equipo (local)** tree view includes: Herramientas del sistema, Visor de eventos, Carpetas compartidas, Usuarios y grupos locales, Rendimiento, Administrador de dispositivos, Almacenamiento, Copias de seguridad de Win, Administración de discos, and Servicios y Aplicaciones.
- Volumen** table:
 

Volumen	Distribución	Tipo	Sistema de archivos	Estado	Capacidad	Espacio disponible	% disponible
(C)	Simple	Básico	NTFS	Correcto (Arranque, Archivo de paginación, Volcado, Partición primaria)	119,34 GB	108,53 GB	91 %
(Disk 6 Partición 3)	Simple	Básico		Correcto (Partición de recuperación)	568 MB	568 MB	100 %
- Disco 6** table:
 

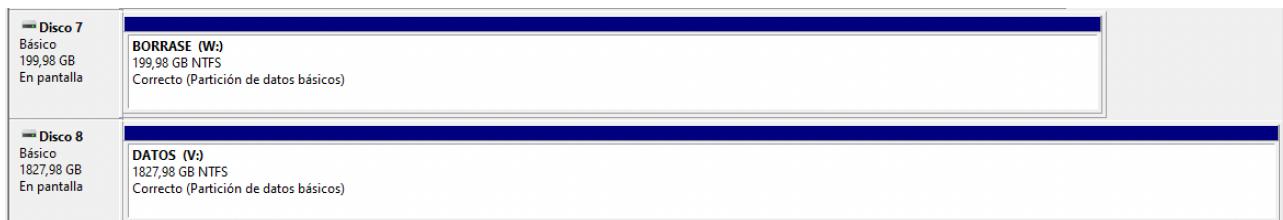
Disco 6	Reservado para el sistema	(C)	568 MB
Básico 120,00 GB En pantalla	100 MB NTFS Correcto (Sistema, Activo, Par)	119,34 GB NTFS Correcto (Arranque, Archivo de paginación, Volcado, Partición primaria)	Correcto (Partición de recuperación)
- Disco 7** table:
 

Disco 7	200,00 GB
Desconocido 200,00 GB Desactivada	No asignado
- Disco 8** table:
 

Disco 8	1828,00 GB
Desconocido 1828,00 GB Desactivada	No asignado

D.- En cada uno de esos discos virtuales creamos una partición que lo ocupe totalmente, le asignamos una letra de acceso a la unidad y le instalamos el sistema de archivos NTFS:

```
RAPID      -      W:      -      BORRASE      -      NTFS
SECURE     -      V:      -      DATOS       -      NTFS
#Creamos los volúmenes NTFS correspondientes, que ocupen todo el disco
#Rapid - W: - BORRASE
Initialize-Disk -FriendlyName "Rapid" -PassThru | 
New-Partition -DriveLetter W -UseMaximumSize
Format-Volume -DriveLetter W -NewFileSystemLabel BORRASE -FileSystem NTFS
-Force -Confirm:$false
#Secure - V: - DATOS
Initialize-Disk -FriendlyName "Secure" -PassThru | 
New-Partition -DriveLetter V -UseMaximumSize
Format-Volume -DriveLetter V -NewFileSystemLabel DATOS -FileSystem NTFS
-Force -Confirm:$false
```



E.- Forzar la avería de un disco y comprobar que el que estaba en *hot-spare* substituye automáticamente al averiado.

```
# Con el siguiente comando comprobamos que el sistema cambia
##el disco averiado por lo marcado como hot-spare:
PS> Get-PhysicalDisk
# Miramos también el estado del Storage Pool:
PS> Get-StoragePool | fl
    # Vemos que:
    #HealthStatus           : Warning
    #OperationalStatus       : Degraded
# Para que ya no se encuentre "Degradado" tenemos que
# sacar el disco averiado
# de la lista de los discos del pool:
PS> $pool = Get-StoragePool | Where-Object { $_.OperationalStatus -eq "Degraded" }
PS> $failedDisk = Get-PhysicalDisk | Where-Object -property HealthStatus -ne Healthy
PS> $failedDisk | Set-PhysicalDisk -Usage retired
PS> $pool | Get-VirtualDisk | Repair-VirtualDisk -AsJob
PS> Remove-PhysicalDisk -StoragePool $pool -PhysicalDisks $failedDisk
#Ahora ya todo bien:
PS> Get-StoragePool
    # Vemos que:
    #HealthStatus           : Healthy
```

F.- Luego, una vez que el sistema ya tiene restaurada la paridad, forzar la avería de otro disco. Comprobar que el volumen sigue operativo y que el usuario puede acceder a los datos.

```
#Vemos el estado actual del Pool y
##de los discos una vez eliminado uno de ellos:
PS> Get-StoragePool
    OperationalStatus: Degraded
    HealthStatus: Warning
```

G.- Agregar un disco nuevo y restaurar, nuevamente, el sistema con paridad.

```
# Añadimos al equipo un nuevo disco. El tamaño no importa,
# siempre y cuando sea mayor que el disco más pequeño del pool
PS> Get-PhysicalDisk
# También se puede ver el disco que falta
# Hay que retirar el disco que falta y añadir ese nuevo al pool:
PS> $NewDisk = Get-PhysicalDisk -CanPool $true
#Seleccionamos el pool degradado
PS> $pool = Get-StoragePool | Where-Object { $_.OperationalStatus -eq "Degraded" }
#Agregamos el $NewDisk al pool como Autoselect
PS> Add-PhysicalDisk -StoragePool $pool -PhysicalDisks $NewDisk -Usage AutoSelect
# Reparamos:
PS> $pool | Get-VirtualDisk | Where-Object -Property HealthStatus -ne Healthy | ` 
Repair-VirtualDisk -AsJob
# Comenzamos con la retirada del disco averiado.
# Lo primero es hacer que ese disco tenga su propiedad
# "Usage" como "Retired"
PS> Get-PhysicalDisk | Where-Object -Property healthstatus -ne healthy
PS> $failedDisk = Get-PhysicalDisk | Where-Object -Property healthstatus -ne healthy
PS> $failedDisk | Set-PhysicalDisk -Usage retired
PS> $failedDisk = Get-PhysicalDisk | Where-Object -Property healthstatus -ne healthy
#Y lo retiramos
PS> $pool | Get-VirtualDisk | Repair-VirtualDisk -AsJob
PS> Remove-PhysicalDisk -StoragePool $pool -PhysicalDisks $failedDisk
# Y comprobamos:
PS> Get-StoragePool
OperationalStatus: OK
HealthStatus: Healthy
```

## 5.- Configuración de la red y del nombre del equipo

Algo fundamental en un equipo es la configuración de su nombre, que debe ser único en la red en la que se encuentra y, como no, su configuración TCP/IP, que también tenemos que conseguir que sea única en nuestra red. Para ello PowerShell nos proporciona una serie de comandos que debemos conocer.

### 5.1.- Comandos cmdlets para la configuración de la red y del nombre del equipo

- **Get-NetAdapter**: devuelve el nombre de las NIC existentes, MAC, velocidad...
- **Rename-NetAdapter** : renombra un adaptador de red
- **Enable-NetAdapter** : activa un adaptador de red
- **Disable-NetAdapter** : desactiva un adaptador de red
- **Restart-NetAdapter** : reinicia un adaptador de red *disable && enable*
- **Test-Connection** : comando ping (envía mensajes ICMP)
- **Get-NetIPInterface** : recupera la lista de interfaces IP configuradas
- **Get-NetIPAddress** : recupera la dirección IP de una interfaz
- **Get-NetIPConfiguration** : recupera la configuración IP
- **Get-NetRoute** : muestra la tabla de enrutamiento
- **New-NetRoute** : crea una ruta nueva en la tabla de enrutamiento
- **New-NetIPAddress** : configura una nueva dirección IP en una interfaz
- **Set-NetIPAddress** : modifica la dirección IP
- **Remove-NetIPAddress** : elimina la dirección IP de una interfaz
- **Remove-NetRoute** : elimina una entrada de la tabla de enrutamiento
- **Set-NetIPInterface** : modifica uno o varios parámetros de una interfaz
- **Get-DnsClientServerAddress** : recupera las direcciones IP de los servidores DNS
- **Set-DnsClientServerAddress** : define las direcciones de los servidores DNS
- **Set-DnsClientGlobalSetting** : define las configuraciones globales DNS
- **Resolve-DnsName** : resuelve los nombres DNS o IP
- **Get-DnsClientCache** : recupera la caché DNS del puesto de trabajo
- **Clear-DnsClientCache** : vacía la caché DNS
- **Register-DnsClient** : renueva la inscripción del cliente DNS

- **Rename-Computer** : cambia el nombre de equipo
  - **Add-VpnConnection** : agrega una nueva conexión VPN
  - **Get-VpnConnection** : recupera la información de conexión de una VPN
  - **Set-VpnConnection** : modifica los parámetros de una conexión VPN
  - **Remove-VpnConnection** : elimina una conexión VPN

## **5.2.- Configura el equipo tal y como se indica a continuación**

- **Configuración IP** : IP fija y libre de la red del aula
    - **IP** : 10.22.100.xx/16
    - **PE** : 10.22.0.254
    - **DNS** : 10.0.4.1
    - **Sufijo DNS** : ies.local
  - **Nombre del equipo** : wclient1

Creamos un *script* que realice estos cambios :

```

#Creamos un Script que realice estos cambios :
#Buscamos el nombre del adaptador a configurar
#Get-NetAdapter -Physical                                         #Una vez que lo tenemos claro...
$nombre = "wclient1"
$nic = "Ethernet"
$ip = "10.22.100.xx"
$ms = 16
$pe = "10.22.0.254"
$dnsServers = @("10.0.4.1")
$dnsSuffix = @("ies.local")

# Deshabilitamos el DHCP si fuese necesario
Set-NetIPInterface -InterfaceAlias $nic -DHCP Disabled
# Eliminamos la configuración IP del equipo si ésta ya existe
Remove-NetIPAddress -InterfaceAlias $nic -IncludeAllCompartments -Confirm:$false 2> $null
# Eliminamos la configuración de la Puerta de Enlace si ésta ya existe
Remove-NetRoute -InterfaceAlias $nic -Confirm:$false 2> $null
# Configuramos la IP estática, la máscara de subred y la puerta de enlace
New-NetIPAddress -InterfaceAlias $nic -AddressFamily IPv4 -IPAddress $ip ` 
-PrefixLength $ms -Type Unicast -DefaultGateway $pe
# Configuramos los servidores DNS
Set-DnsClientServerAddress -InterfaceAlias $nic -ServerAddresses $dnsServers
# Anexar sufijo DNS
Set-DnsClientGlobalSetting -SuffixSearchList $dnsSuffix
#Habilitar pings
New-NetFirewallRule -name 'ICMPv4' -DisplayName 'ICMPv4' ` 
-Description 'Allow ICMPv4' -Profile Any -Direction Inbound -Action Allow ` 
-Protocol ICMPv4 -IcmpType 8 -Program Any -LocalAddress Any -RemoteAddress Any

#Para cambiar el nombre al equipo
Rename-Computer -NewName $nombre -Restart -Force

```

Y comprobamos la nueva configuración:

### 5.3.- Pequeños ejercicios del uso de cmdlets para la administración de redes

→ Realizar un **ping** a una máquina de la red en la que te encuentres. Envía un único mensaje **ICMP**.

```
PS> Test-Connection -ComputerName 192.168.1.1 -Count 1
```

→ Guarda en un archivo las propiedades de los equipos que contestan a un mensaje **ICMP** de los existentes en tu subred.

```
PS> $subred = '192.168.1.'
PS> $equipo = @(1..254)
PS> $salida = foreach ($i in $equipo) ` 
{ $(Test-Connection -ComputerName $($subred+$i) -Count 1 2> $null) | fl }
PS> $salida | Out-File .\equipos.txt -Force
```

→ Obtener los distintos adaptadores de red de un puesto de trabajo.

```
PS> Get-NetAdapter -Physical | fl
```

→ Mostrar el conjunto de interfaces **IP** del puesto de trabajo ( propiedades interesantes: **InterfaceAlias** , **AddressFamily** , **Dhcp** , **ConnectionState** ).

```
PS> Get-NetIPInterface | fl InterfaceAlias, AddressFamily, Dhcp, ConnectionState
```

→ Descubrir la **IP** de la tarjeta de red Ethernet de tu equipo.

```
PS> $(Get-NetIPAddress -InterfaceAlias 'Ethernet' -AddressFamily IPv4).IPAddress
```

→ Recuperar todas las **IPs IPv4** configuradas en tu equipo.

```
PS> $ IPs = $(Get-NetIPAddress -AddressFamily IPv4)
PS> foreach ($IP in $ IPs) { $IP.IPAddress }
```

Luego, solamente las configuradas manualmente.

```
PS> Get-NetIPAddress -AddressFamily IPv4 ` 
-PrefixOrigin Manual -SuffixOrigin Manual
```

→ Descubre la puerta de enlace predeterminada de la *interface* 'Ethernet'

```
PS> Get-NetRoute -InterfaceAlias 'Ethernet' -DestinationPrefix 0.0.0.0/0
```

→ Descubrir los servidores DNS de la interface 'Wi-fi'

```
PS> Get-DnsClientServerAddress -InterfaceAlias 'Wi-fi'
```

→ Resolver el nombre **DNS** '[www.google.es](http://www.google.es)':

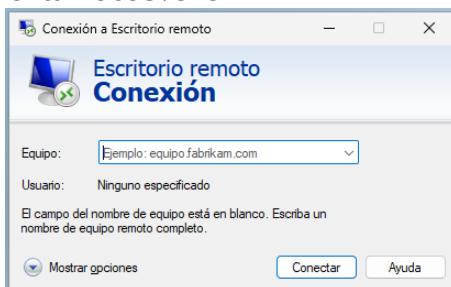
```
PS> Resolve-DnsName -Name 'www.google.es'
```

→ Recuperar y borrar la caché **DNS** del equipo de trabajo.

```
PS> Get-DnsClientCache
PS> Clear-DnsClientCache
```

#### 5.4.- Conexión remota por SSH

Para poder administrar un equipo Windows, lo normal será acceder a él por **Escritorio Remoto** empleando la herramienta **mstsc.exe**.



Otro modo de administrar un equipo Windows remotamente será instalando en el equipo a conectarse el **Servidor SSH**. En la siguiente imagen se puede ver cómo instalar y activar el servicio **sshd**.

```
PS> Add-WindowsCapability -Online -Name openssh.server
PS> Set-Service sshd -StartupType Automatic
PS> Start-Service sshd
PS> New-NetFirewallRule -Name sshd -DisplayName 'Alloww SSHD' `
-Enabled True -Direction Inbound -Protocol TCP -Action Allow -LocalPort 22
```

Luego, para conectarse a ese equipo, es necesario emplear un **cliente ssh** cualquiera. Suponiendo realizado un redireccionamiento de puertos a 127.0.0.1:2222, utilizaríamos el comando siguiente :

```
$ ssh -p 2222 jefe@127.0.0.1
```

En el equipo cliente la conexión es al **cmd**, por lo que tendremos que ejecutar el Powershell :  
C:\Users\jefe>powershell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.
```

Instale la versión más reciente de PowerShell para obtener nuevas características y mejoras. <https://aka.ms/PSWindows>

```
PS C:\Users\Administrador>
```

Instalamos el editor **nano** para Windows :

```
PS> Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1'))
PS> choco install nano -y
```

Ya podemos utilizarlo exactamente igual que en Linux.