

# Unidad 8: Instalación de Windows

Sistemas Informáticos  
Desarrollo de Aplicaciones Multiplataformas

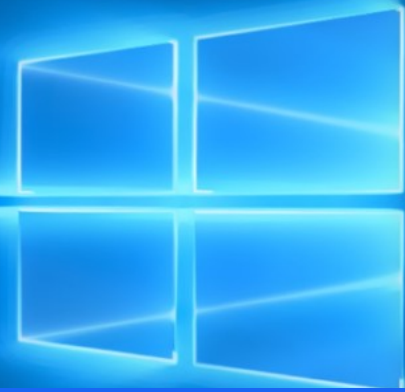




# Índice general

<b>1</b>	<b>Introducción a Windows</b>	<b>5</b>
1.1	Windows VS Linux. ¿En qué se diferencian?	5
1.2	Historia de Windows	6
1.2.1	MS-DOS	6
1.2.2	Windows 1.0	6
1.2.3	Windows 95	7
1.2.4	Windows 98	7
1.2.5	Windows 2000	7
1.2.6	Windows XP	7
1.2.7	Windows Vista	7
1.2.8	Windows 7	8
1.2.9	Windows 8	8
1.2.10	Windows 10	8
1.3	Instalación	9
1.3.1	Proceso de instalación	9
1.4	Interfaz gráfica	11
1.5	Navegación	12
1.6	Menú de Inicio	12
1.7	Monitor de recursos	14
1.7.1	Monitor de recursos	14
1.7.2	Monitor de rendimiento	16
1.8	Rutas	18
<b>2</b>	<b>Símbolo de Sistema</b>	<b>19</b>
2.1	Introducción	19

<b>2.2</b>	<b>Gestión de ficheros</b>	<b>19</b>
2.2.1	Crear archivos	19
2.2.2	Borrar archivos	20
2.2.3	Ver contenido de archivos	20
2.2.4	Copiar y mover	21
2.2.5	Renombrar	23
2.2.6	Atributos de ficheros	23
<b>2.3</b>	<b>Gestión de directorios</b>	<b>24</b>
2.3.1	Movernos entre directorios	24
2.3.2	Crear carpetas	24
2.3.3	Borrar carpetas	25
2.3.4	Listar directorios	25
2.3.5	Otros comandos útiles	27
<b>2.4</b>	<b>Redireccionamiento y filtros</b>	<b>27</b>
2.4.1	Redireccionamiento	27
2.4.2	Filtros	28
<b>2.5</b>	<b>Procesamiento por lotes</b>	<b>28</b>
<b>3</b>	<b>Windows PowerShell</b>	<b>33</b>
<b>3.1</b>	<b>Introducción</b>	<b>33</b>
<b>3.2</b>	<b>Sintaxis</b>	<b>33</b>
3.2.1	Verbos	33
3.2.2	Nombres	34
3.2.3	Buscando comandos	34
3.2.4	Parámetros	34
3.2.5	Confirm, WhatIf y Force	35
<b>3.3</b>	<b>Ayuda</b>	<b>36</b>
3.3.1	Ayuda actualizable	36
3.3.2	Comando Get-Help	36
3.3.3	Parámetros	37
3.3.4	Save-Help	38
3.3.5	Update-Help	38
<b>3.4</b>	<b>CMDLET básicos</b>	<b>38</b>
3.4.1	Crear archivos y directorios	38
3.4.2	Borrar archivos y directorios	40
3.4.3	Ver contenido de archivos	40
3.4.4	Copiar y mover	41
3.4.5	Renombrar	41
3.4.6	Otros comandos útiles	42
3.4.7	Atributos	44
<b>3.5</b>	<b>Expresiones regulares y patrones</b>	<b>44</b>
3.5.1	Patrones	44
3.5.2	Expresiones Regulares	45
<b>3.6</b>	<b>Pipeline</b>	<b>47</b>
3.6.1	Miembros	48
3.6.2	Filtros	49



# 1. Introducción a Windows

*Windows 10 es la última versión del sistema operativo de Microsoft. En esta unidad veremos en qué consiste este S.O. y sus diferencias con otros como Linux. Veremos todas las características necesarias tanto hardware como de otros componentes del sistema que sean necesarias para instalarlo. Windows 10 combina las características de sus dos versiones anteriores para que sea más fácil de usar tanto en ordenadores de sobremesa como en portátiles y dispositivos móviles. A continuación, veremos los fundamentos de Windows 10.*

## 1.1 Windows VS Linux. ¿En qué se diferencian?

Tanto Windows como Linux son dos sistemas operativos que se usan tanto a nivel empresarial en estaciones de trabajo o servidores, como a nivel doméstico. Sin embargo difieren en una serie de características. La principal diferencia entre ambos es la **licencia propietaria de Windows** y la **licencia de software libre de Linux**.

Otra de las principales diferencias es el sistema de fichero de Windows. En linux todo eran ficheros (dispositivos, directorios, entrada/salida, los propios ficheros...) ordenados en una **estructura en árbol**, donde el directorio raíz puede considerarse el inicio del sistema de archivos, y en él se ramifican otros subdirectorios. La raíz se indica con una barra inclinada '/'. En cambio, en Windows, los archivos se almacenan en carpetas en **diferentes unidades** de datos como C: D: E:

- Al ser linux un sistema operativo de código abierto, permite que los usuarios puedan cambiar el código fuente según las necesidades. Esto ha provocado que surjan distintas distribuciones de linux (distros) orientadas cada una a distintas necesidades y mantenidas por empresas o por una comunidad. En cambio, el sistema operativo Windows se trata de un **S.O. comercial**, por lo que el código fuente es cerrado y el usuario no puede acceder al código.
- Windows utiliza diferentes dispositivos como C, D, E para guardar los ficheros y directorios, en cambio, en linux se utiliza un **árbol de directorios**.
- En Windows se consideran las impresoras, discos HDD o SSD, unidades de CD-ROM como **dispositivos**, sin embargo en linux los periféricos son tratados como ficheros.
- Los **tipos de usuarios** también son diferentes entre ambos sistemas operativos, en Windows tenemos tres tipos: Administrador, Estándar e Invitado, pero en Linux tenemos la cuenta de usuario, el administrador o root y cuentas de servicios, creadas principalmente por los paquetes del sistema.

- En Windows no puede haber dos ficheros con el mismo nombre en la misma carpeta, aunque esté escrito de forma diferente entre mayúsculas y minúsculas. Linux por otra partes, es sensible a las mayúsculas y minúsculas y por lo tanto sí permite tener dos fichero como “ejemplo” y “Ejemplo” en el mismo directorio.
- El kernel de Windows es híbrido, es decir, tiene dos capas, una de usuario y otra del sistema, donde se ejecutan los componentes principales del sistema. En cambio Linux utiliza un kernel monolítico, donde todo se ejecuta en una misma capa (aunque internamente esté dividida en subcapas).

Windows también se trata de uno de los sistemas operativos más utilizados tanto a nivel doméstico como empresarial, con una cuota de mercado a nivel mundial en el primer semestre de 2020 del **87.54 %** según statista<sup>1</sup>. Por esta razón desarrollar software y aprender a administrar estos sistemas operativos, tiene una gran compatibilidad con dispositivos en todo el mundo. También como ventajas con respecto a otros sistemas operativos, tiene integración directa con los servicios y herramientas de Microsoft, esta es una de las principales razones por las que se utiliza en las empresas en todo el mundo, como **Microsoft Azure** o **Office 365**. Cabe destacar también, que en los últimos años Microsoft está incluyendo en su sistema operativo el kernel completo de Linux mediante el **WSL (Windows Subsystem for Linux)**, permitiendo utilizar linux desde la propia terminal de Windows y poder correr ejecutables de Linux nativamente en Windows 10 y Windows Server 2019.

## 1.2 Historia de Windows

### Microsoft Windows

family tree

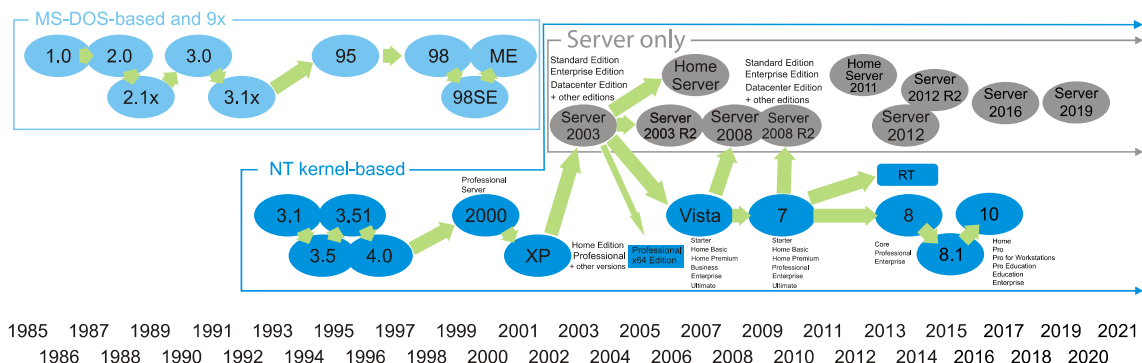


Figura 1.1: Cronología de los S.O. Windows<sup>2</sup>

### 1.2.1 MS-DOS

El sistema operativo original creado por Microsoft para IBM, MS-DOS, era el sistema operativo estándar para los ordenadores personales de IBM. Se trataba de un S.O. muy simple, cuyas versiones posteriores se fueron satisficando a medida que incorporaban características de los sistemas operativos para miniordenadores.

### 1.2.2 Windows 1.0

Presentado en 1985, Microsoft Windows 1.0 recibió su nombre debido a las cajas informáticas o “ventanas” que representaban un aspecto fundamental del sistema operativo. En lugar de teclear

<sup>1</sup><https://es.statista.com/estadisticas/576870/cuota-de-mercado-mundial-de-los-sistemas-operativos/>

<sup>2</sup>[https://upload.wikimedia.org/wikipedia/commons/0/0e/Windows\\_Family\\_Tree.svg](https://upload.wikimedia.org/wikipedia/commons/0/0e/Windows_Family_Tree.svg)

los comandos de MS-DOS, Windows 1.0 permitía a los usuarios apuntar y hacer clic para acceder a las ventanas.

### 1.2.3 Windows 95

Windows 95 fue lanzado, como su nombre indica, en 1995, siendo una importante actualización porque se introdujeron mejoras significativas entre las que se pueden mencionar los profundos **cambios en la interfaz gráfica** y el pasar de una arquitectura **multitarea cooperativa de 16 bits** a usar una arquitectura **multitarea apropiativa de 32 bits**.

### 1.2.4 Windows 98

Lanzado el 25 de junio de 1998, se sigue trantando de un sistema operativo **híbrido de 16 y 32 bits**, pero con soporte mejorado para **FAT32**. Tenía un arranque basado en MS-DOS. Se le añades características prácticas a la interfaz de usuario como poder minimizar las ventas haciendo clic en su botón de la barra de tareas o abrir inicio con un click y los botones de navegación (atrás/adelante) en el explorador de Windows. También desde el punto de vista del hardware se añade **soporte** para **DVD**, **puerto AGP** para tarjetas gráficas, Unidades **IDE** y **SCSI** y admitía muchos medios de red, incluido **Ethernet**.

### 1.2.5 Windows 2000

Todos los sistemas operativos anteriores se encontraban basados en MS-DOS, sin embargo Microsoft también desarrolló desde 1993 otra línea de S.O. orientado a la empresa y la oficina llamado **Windows NT**. Windows 2000 se trataba de la 5 versión de esta línea de sistemas operativos, llamado originalmente como Windows NT 5.0. Lanzado en el 2000, fue la **última versión** de esta línea de Windows destinada **únicamente a estaciones de trabajo y servidores**. En 1999, Microsoft empezó a trabajar en los proyectos **Neptune** y **Odyssey**. El primero iba a ser la primera versión NT para consumidor, mientras que el segundo iba a ser el sucesor de Windows 2000. Sin embargo, en enero de 2000, Microsoft decidió cancelar los dos proyectos para fusionarlos en uno solo, **Whistler**, que acabaría siendo **Windows XP** en 2001. Desde entonces, Windows NT se distribuye tanto para consumidores como para empresas.

### 1.2.6 Windows XP

Como se ha mencionado anteriormente, se trata del **sucesor de Windows 2000**, perteneciente a la línea de Windows NT, pero a diferencia de las anteriores versiones, estaba orientado tanto a nivel empresarial como doméstico. Windows XP introdujo una serie de categorías nuevas, como pueden ser:

- Un ambiente gráfico más agradable.
- Secuencias más rápidas de inicio y de hibernacion.
- Capacidad de desconectar un dispositivo externo sin necesidad de reiniciar el sistema.
- Uso de **varias cuentas de usuario** permitiendo que un usuario guarde el estado actual y las aplicaciones que tiene abiertas, haciendo que si otro usuario inicia sesión no se pierda esa información.
- **Escritorio remoto** (permitiendo abrir una sesión a través de la red).

Como curiosidad, se trata de la primera versión de Windows que utiliza la **activación del producto** para reducir la piratería del software.

### 1.2.7 Windows Vista

Tras el lanzamiento de Windows XP, Microsoft empezó el desarrollo del proyecto **Longhorn**, un proyecto muy ambicioso ideado originalmente como una **actualización menor de Windows XP**,

pero nunca llegó a ver la luz en su versión final. Tuvieron que abandonar muchas de las innovadoras ideas que pretendían llevar a cabo, convirtiéndolo en **Windows Vista**. Para muchos considerado el peor sistema operativo de Windows, principalmente por sus **requisitos hardware**, que hacían que la novedosa interfaz Windows Aero hiciese que los **tiempos de respuesta** perjudicasen la experiencia de usuario en equipos modestos. Pero también consta de una serie de **avances positivos**, como por ejemplo la introducción de **DirectX 10**, unas APIs que permitían mejorar tanto en el rendimiento como la calidad de los juegos.

### 1.2.8 Windows 7

A diferencia del salto de Windows Vista en cuanto arquitectura y características con respecto a Windows XP, Windows 7 fue inicialmente una **actualización incremental** y focalizada de Windows Vista, permitiendo mantener un cierto grado de **compatibilidad** con aplicaciones y hardware en los que ya era compatible su antecesor. En el desarrollo de Windows 7, se dio importancia a la **mejora de la interfaz gráfica** para volverla más **accesible** e incluir nuevas características que permitan realizar las tareas de forma **fácil y rápida**, la mismo tiempo que hacerlo más rápido, ligero y estable. Añade una serie de características nuevas como el soporte para sistemas con **múltiples tarjetas gráficas de distintos proveedores**, soporte para **discos duros virtuales**, **rendimiento** mejorado en procesadores **multinúcleo**, mejor **rendimiento de arranque** y mejoras en el núcleo.

### 1.2.9 Windows 8

La siguiente versión de Windows (Windows 8), añade soporte de **microprocesadores ARM**, además de los tradicionales x86 de intel y AMD. Por esta razón y por añadir compatibilidad con un gran número de dispositivos, incluido aquellos con pantallas táctiles, se cambió la interfaz gráfica de Windows **eliminando el menú de inicio**, existente desde Windows 95 como un estándar a la hora de presentar aplicaciones en interfaces gráficas, recibió muchas críticas al respecto hasta que el 2 de abril de 2014 Microsoft reconoció el error y anunció la implementación del menú de inicio en su siguiente versión.

El 18 de octubre de 2013, Microsoft lanzó una actualización gratuita al sistema: Windows 8.1. Mientras que el 29 de julio de 2015, presentó su sucesor, Windows 10, orientado a integrar de una mejor forma el sistema operativo en todos los dispositivos, desde ordenadores, tabletas y hasta teléfonos inteligentes, destacando el regreso de uno de sus elementos más característicos, el ausente Menú Inicio.

### 1.2.10 Windows 10

La **última versión** de Windows hasta el momento, perteneciente a la **línea de Windows NT**. Ha traído grandes avances y cambios hasta el momento, como por ejemplo la inclusión del **kernel completo de linux** de forma nativa, permitiendo que en un mismo S.O. se puedan ejecutar aplicaciones tanto de linux como de Windows.

Para la versión empresarial, **Windows 10 Enterprise**, se ofrece **características de seguridad adicionales**, como pueden ser establecer normativas para el cifrado de datos automático y bloquear selectivamente las solicitudes de acceso a los datos cifrados. Windows 10 también ofrece **Device Guard**, una característica que permite a los administradores reforzar la seguridad mediante el bloqueo de la ejecución de software que no está firmado digitalmente por un proveedor de confianza o Microsoft.

Se introduce también con esta versión el navegador **Edge** basado en chromium sustituyendo a Internet Explorer como el navegador predeterminado en Windows.



## 1.3 Instalación

Si ya disponemos de una versión anterior de Windows en nuestro sistema podemos actualizarlo directamente a Windows 10. Normalmente cuando actualizamos el sistema, todos los documentos y aplicaciones permanecen intactas, pero se debe de realizar una copia de seguridad previa de todo el sistema. En el caso de que quisiéramos realizar una instalación limpia de Windows, se borrará todo el sistema, incluido los documentos y las aplicaciones.

Antes de instalar el S.O. debemos de asegurarnos de que el ordenador dispone de todos los requisitos necesarios para su actualización a Windows 10, sobre todo en dispositivos antiguos, estos son:

- Un procesador (CPU) con una frecuencia de reloj de al menos 1GHz.
- De 1/2 GB de memoria RAM, aunque es recomendable usar 4GB.
- Al menos 16/32GB de espacio libre en el disco duro.

### 1.3.1 Proceso de instalación

En esta sección vamos a ver el proceso completo de instalación de Windows 10 en una máquina virtual, para ello vamos a utilizar virtualbox. De forma que las pruebas que realicemos no afectarán directamente al ordenador sin necesidad de crear una partición exclusiva para Windows.

Al ejecutar VirtualBox pinchamos en “Nueva” y escribimos el nombre con el que queramos identificar esta instancia, el tipo del sistema que queremos instalar y la versión, en nuestro caso seleccionaremos Windows 10.

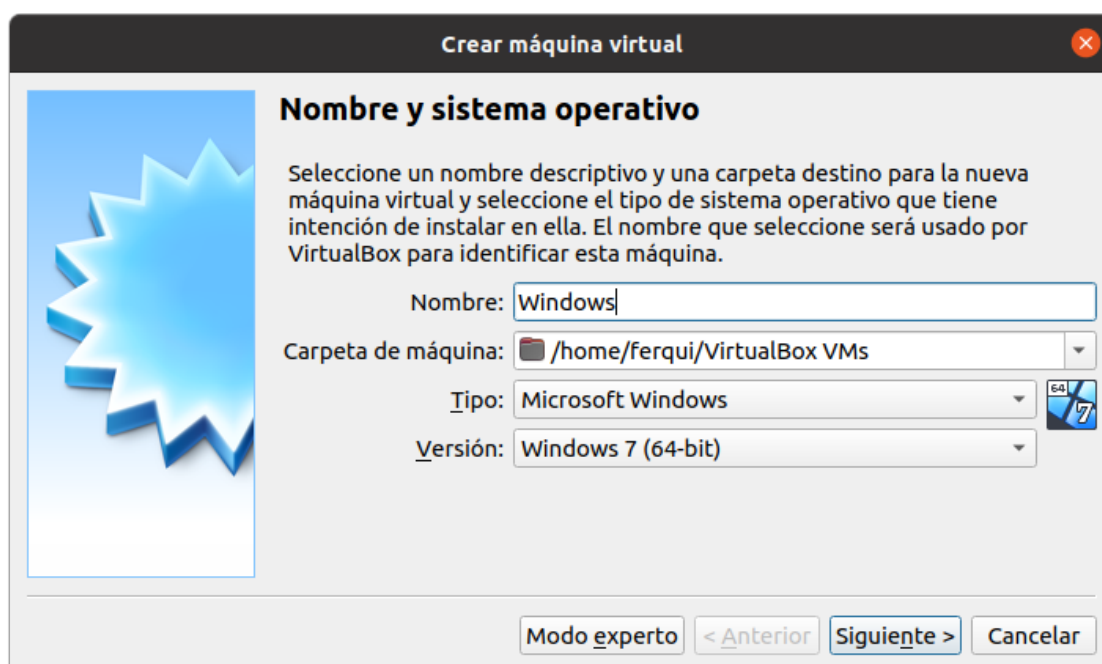


Figura 1.2: Selección de S.O. en VirtualBox

Después nos aparecerá una ventana en la que elijiremos la cantidad de memoria RAM necesaria para nuestro sistema (fig. 1.3). El mínimo recomendado es de 2GB

A partir de ahí, irán apareciendo distintas ventanas para la elección del disco duro virtual del sistema. Eligiremos las siguientes opciones:

- Crear un disco duro virtual ahora
- VDI (VirtualBox Disk Image)
- Reservado dinámicamente

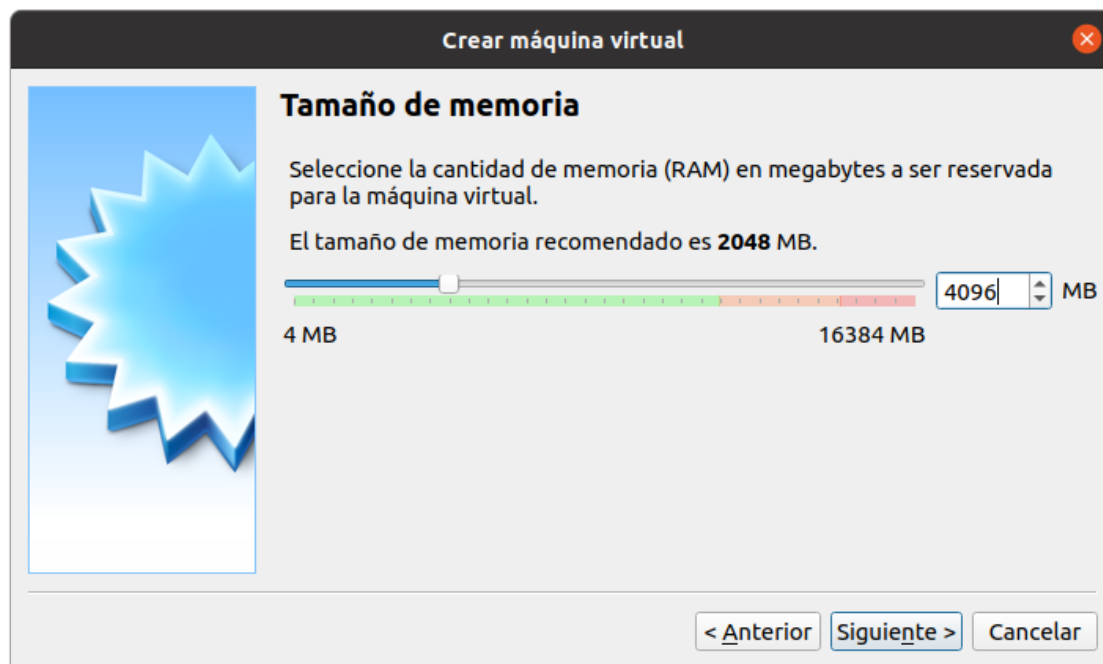


Figura 1.3: Selección de memoria

- Elegir el tamaño del disco duro

Una vez configurada la máquina virtual, procedemos a la instalación del sistema operativo. Para ello necesitamos la imagen ISO del sistema operativo, que se puede descargar directamente desde la página del proveedor, en nuestro caso, Microsoft <sup>3</sup>.

Con nuestra máquina virtual creada, la seleccionamos y pinchamos en Iniciar y en la nueva ventana seleccionamos la imagen ISO que nos hemos descargado. Una vez seleccionada, nuestra máquina virtual se reiniciará y mostrará la imagen principal del proceso de instalación de Windows 10 (fig. 1.4) donde seleccionaremos el idioma y la distribución del teclado.

Una vez seleccionado el idioma nos aparecerá una ventana donde pulsaremos intalar y nos pedirán la clave de producto. En nuestro caso como lo usaremos para realizar pruebas, seleccionaremos “No tengo clave de producto”.

una vez esto, seleccionaremos la versión que queramos instalar, Windows 10 Pro en nuestro caso y aceptamos los términos de licencia.

Como queremos realizar una instalación limpia de Windows, seleccionaremos la opción “Personalizada” (fig. 1.6) donde eligiremos las particiones que queramos usar. Esto nos vale también cuando queramos realizar un dual boot entre Windows y Linux por ejemplo.

A partir de aquí podemos personalizar las distintas particiones en las que vamos a dividir el disco para instalar Windows (fig. 1.7). En nuestro caso tenemos un único disco duro de 50GB y lo instalaremos ahí, pero es posible dividirlo en distintas particiones para realizar un dual boot con linux por ejemplo, o podemos tener también varios discos duros instalados en el sistema.

A partir de este momento comenzará el proceso de instalación copiando los archivos de Windows e instalando sus características (fig. ??).

En las siguientes ventanas irán apareciendo una serie de configuraciones de Windows, incluida la distribución del teclado y la creación de la cuenta. Seleccionaremos las siguientes opciones:

- Distribución del teclado: Español.
- Omitir.

<sup>3</sup><https://www.microsoft.com/es-es/software-download/windows10ISO>

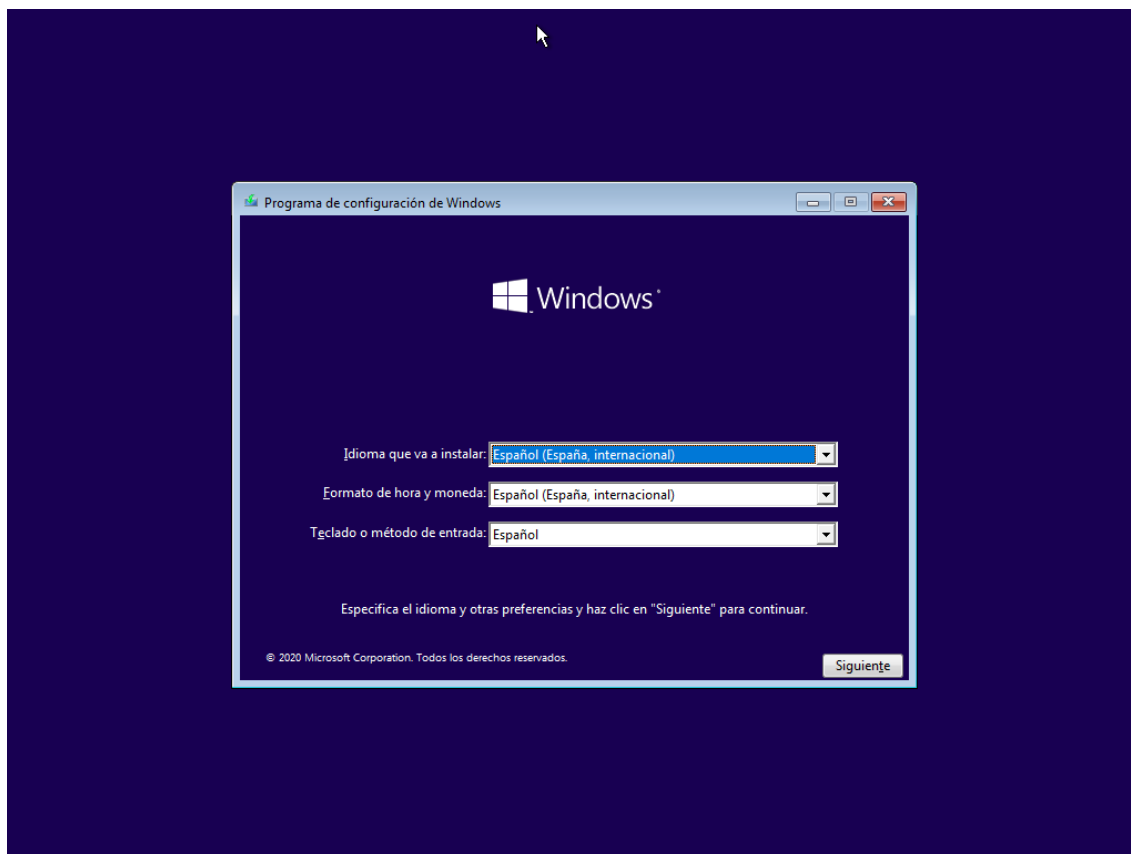


Figura 1.4: Selección de idioma en Windows

- Configuración para uso personal.
- Cuenta sin conexión.
- Nombre de usuario: usuario
- Contraseña: password
- Rellenar tres preguntas de seguridad.

Y finalmente ya tenemos instalado Windows 10 en nuestro equipo.

## 1.4 Interfaz gráfica

Una de las partes más importantes del escritorio es la barra de tareas. Por defecto, se encuentra en la parte inferior de la pantalla y da acceso al menú de inicio, a varios iconos de aplicaciones y al área de notificaciones.

En Windows 10, si una aplicación está activa o abierta, verás una línea verde debajo de su icono. Al hacer clic en el icono, aparecerá la ventana de la aplicación.

Todas las ventanas abiertas tienen tres botones en la esquina superior derecha. Sirven para minimizar, maximizar o cerrar la ventana. Estas se pueden desplazar o cambiar de tamaño a voluntad. Para mover una ventana, basta con hacer clic en su barra de título en la parte superior de la ventana y arrastrarla. Para cambiar el tamaño de una ventana, mueva el ratón a cualquier esquina hasta que vea una flecha de doble cara.

El fondo del escritorio se trata simplemente de una imagen que aparece en el fondo de tu pantalla. La mayoría de los ordenadores vienen con un fondo preseleccionado, pero se puede cambiar por cualquier imagen.

Para cambiar el fondo, siga estos pasos:

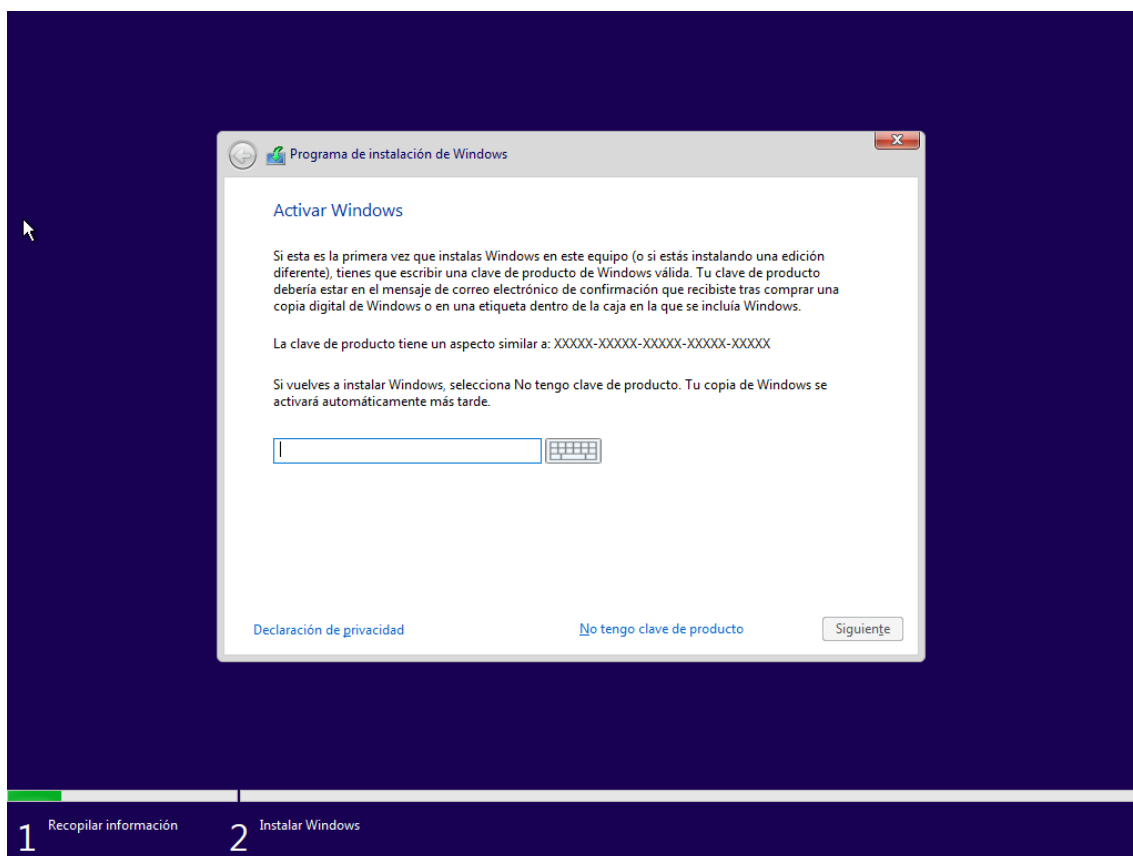


Figura 1.5: Selección de clave de producto de Windows

1. Haz clic con el botón derecho del ratón en el fondo y elige "Personalizar".
2. En la ventana de personalización, elija entre una serie de imágenes preseleccionadas o busque las tuyas propias.
3. Después de elegir una imagen, el Fondo cambiará automáticamente.

## 1.5 Navegación

Para navegar por tu Windows, sólo tienes que escribir lo que buscas en la búsqueda de la barra de tareas. Puede ser el nombre de un documento o aplicación, o simplemente cualquier información que esté buscando.

Si busca un documento concreto, otra alternativa es utilizar el Explorador de archivos haciendo clic en el icono de la carpeta de la barra de tareas. En la ventana del Explorador de Archivos, puedes navegar por todas tus carpetas y documentos.

Otra característica interesante que tiene Windows 10 son los escritorios virtuales. Esto nos permite tener varias pantallas de escritorio en las que se puede mantener organizadas todas las ventanas abiertas. Para crear un nuevo escritorio virtual, pulsamos sobre el botón de “Vista de tareas” que se encuentra en la barra de tareas, desde ahí pulsamos en la esquina inferior derecha en crear un nuevo escritorio. De esta forma también podemos ir alternando entre ellos.

## 1.6 Menú de Inicio

El Menú de Inicio es el punto principal por el que accedemos a todas las aplicaciones de Windows, podemos abrirlo tanto pulsando el botón del logo de Windows en la esquina inferior

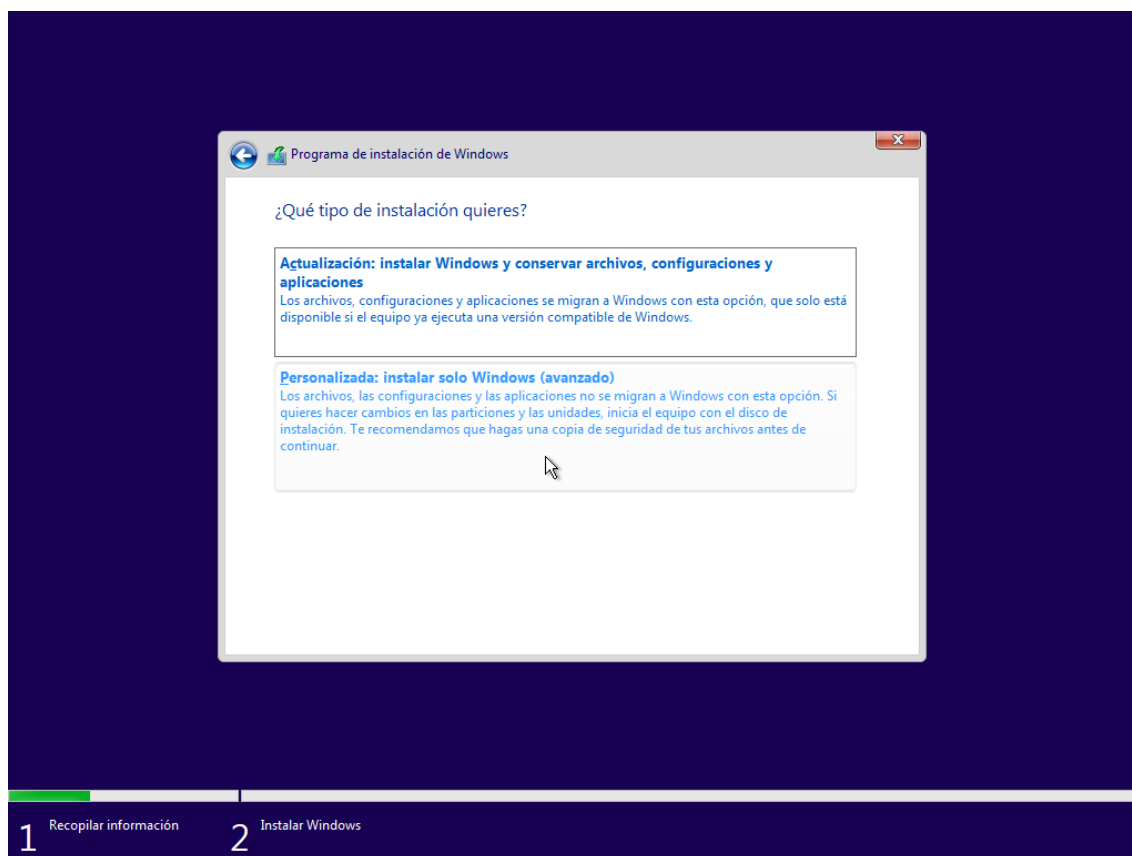


Figura 1.6: Selección del método de instalación

izquierda o desde el teclado pulsando la tecla de Windows. El “Start Menu” tiene dos paneles, la izquierda es como el menú tradicional, donde tenemos las opciones para cambiar los usuarios, acceder a las aplicaciones que más se usan, abrir el explorador de archivos, abrir los ajustes del sistema, apagar, reiniciar o suspender el equipo y ver todas las aplicaciones instaladas en el sistema.

Desde el menú también tenemos una barra de búsqueda que nos permite buscar cualquier fichero o aplicación que se encuentra en el equipo, así como en internet. En el panel de la derecha tenemos una serie de cuadrículas parecida a Windows 8, que se pueden personalizar.

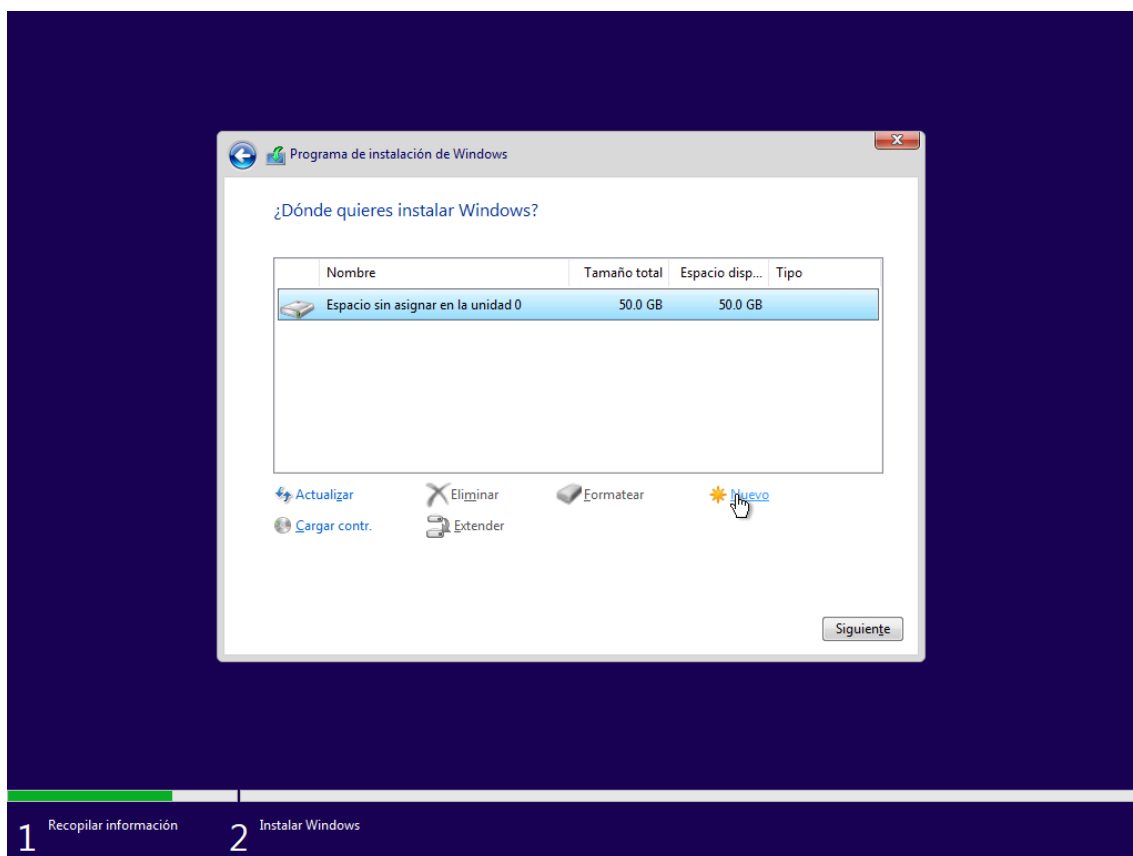


Figura 1.7: Partición de disco.

## 1.7 Monitor de recursos

Cuando estamos administrando un sistema Windows queremos ver el rendimiento que tienen los software que se están ejecutando sobre el sistema, de modo que no se ralentice. Para ello tenemos que ver los recursos de los que dispone el sistema, monitorizarlos y supervisar los programas que se están ejecutando para evitar una sobrecarga del equipo. Para este cometido, los sistemas Windows 10 disponen de dos herramientas:

- Monitor de recursos
- Monitor de rendimiento

### 1.7.1 Monitor de recursos

El monitor de recursos nos permite ver cuántos recursos del sistema **consumen los programas o servicios** que se están ejecutando. Para abrirlo simplemente tenemos que escribir “Monitor de recursos” en el buscador de Windows, como se muestra en la imagen 1.8.

Una vez abierto nos aparece una ventana (fig. 1.9) que tiene información general sobre el uso de CPU, disco, red y la memoria RAM. A la derecha tenemos un panel que nos muestran unas gráficas sobre las estadísticas de uso de estos cuatro componentes en el último minuto.

También tenemos una serie de pestañas en la parte superior para ver información más detallada de estos componentes. Si nos vamos a la pestaña de CPU podemos encontrarnos con unos gráficos en la parte derecha, pero esta vez nos indican las estadísticas de uso de los núcleos de los que dispone el procesador. En esta ventana tenemos distintos subapartados:

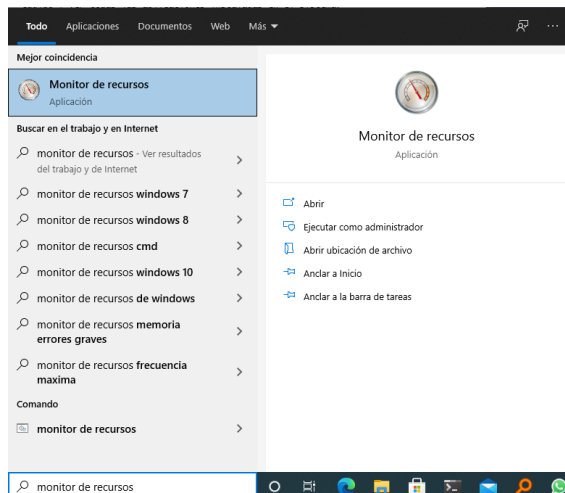


Figura 1.8: Se muestra cómo se abre el monitor de recursos en Windows 10.

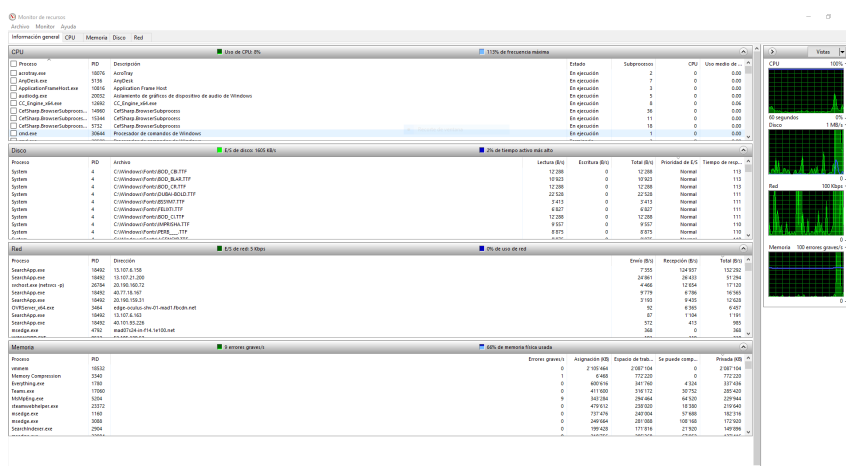


Figura 1.9: Pantalla principal del monitor de recursos.

## CPU

Aquí tenemos los procesos que se encuentran activos, junto con su PID, una descripción, el estado en el que se encuentra, el número de subprocesos que dependen de él, el uso de la CPU y su uso medio. Si seleccionamos cualquiera de esos procesos, en los subapartados inferiores nos filtran por él, indicándonos los servicios que se encuentran en uso, los identificadores y los módulos asociados a ese proceso. Esto puede ser de gran utilidad para identificar aplicaciones que dependen de otra principal, los servicios que tienen asociados y qué ejecutables y librerías (DLL) está utilizando. Si hubiera algún problema en el PC, este apartado nos ayuda a identificarlo y poder solventarlo.

## Memoria

En el caso de la pestaña de memoria, las gráficas nos muestra el uso de la memoria, la carga de asignación y si ocurren errores. En los paneles de la izquierda tenemos los distintos procesos en ejecución junto a la cantidad de memoria asignada a ellos. En el panel inferior tenemos un gráfico que nos indica el uso actual de la memoria RAM. En este apartado podemos identificar aquellos procesos que tienen un gran consumo de memoria y que pueden producir una disminución del tiempo de respuesta del sistema.

## Disco

En el caso de la pestaña de disco, las gráficas muestran el uso medio del disco duro y la cola de todos los discos instalados en el sistema (ya sean HDD o SSD). En los paneles de la izquierda tenemos los procesos activos como en las pestañas anteriores, pero con las estadísticas de lectura y escritura en disco. En el panel inferior tenemos la actividad de los discos de forma individual (tanto físicos como lógicos). Como en los casos anteriores, también es posible filtrar por procesos y podemos ver cuánta actividad de disco están consumiendo procesos concretos en el sistema. Esto puede ser de gran utilidad si vemos que el rendimiento de un SSD puede bajar, identificando el programa concreto que lo provoca.

## Red

Por último tenemos la pestaña de red que nos indica la actividad actual de red del sistema. En las gráficas de la derecha tenemos en este caso el uso de red (y de los distintos adaptadores de red) y el número de conexiones TCP que se crean en el sistema. Este apartado nos permite ver los recursos de red que están consumiendo y podemos filtrar, como en todos los casos anteriores por procesos, pudiendo ver la IP de los servidores a los que nos estamos conectando y los puertos a los que lo hacemos.

### 1.7.2 Monitor de rendimiento

El **monitor de rendimiento** no hay que confundirlo con el **monitor de recursos** visto anteriormente, esta nos permite realizar un seguimiento exhaustivo sobre el rendimiento de cada una de las partes del sistema. Para abrirlo es suficiente con buscarlo en la barra de búsqueda el menú principal de windows, como se muestra en la figura 1.10

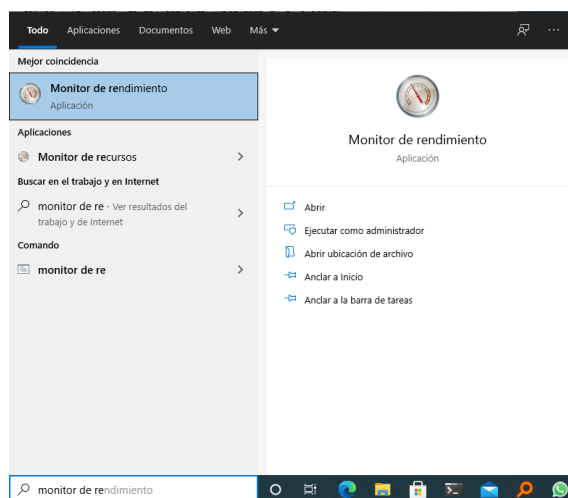


Figura 1.10: Se muestra cómo se abre el monitor de rendimiento de Windows

Una vez abierta podemos abrir el "Monitor de rendimiento" que se encuentra en el lado izquierdo de la ventana (fig. 1.11). Esto hace que nos muestre una gráfica en tiempo real sobre el porcentaje del tiempo de uso del procesador. En la parte inferior de la gráfica, también podemos ver la última medición, el promedio, el mínimo, el máximo y la ventana de tiempo de la gráfica.

Si suponemos que tenemos un problema de conexión y queremos ver qué es lo que está pasando con la tarjeta de red, pulsáramos el botón + verde ("Agregar") que se encuentra encima de la gráfica buscaríamos "Adaptador de red" y pulsando en cualquiera de los parámetros de debajo, nos saldrán las instancias del objeto seleccionado, donde deberemos escoger nuestra tarjeta de red (fig. 1.12). Una vez seleccionado pulsamos el botón "Agregar" y aceptar.



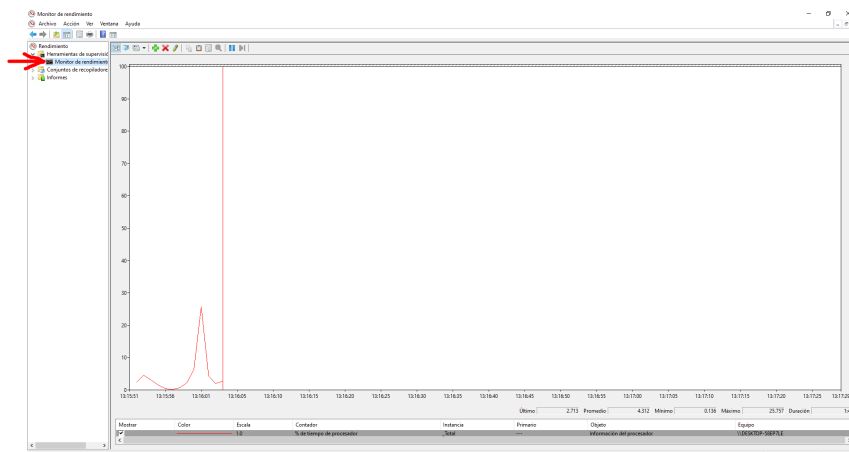


Figura 1.11: Pantalla principal del monitor de rendimiento

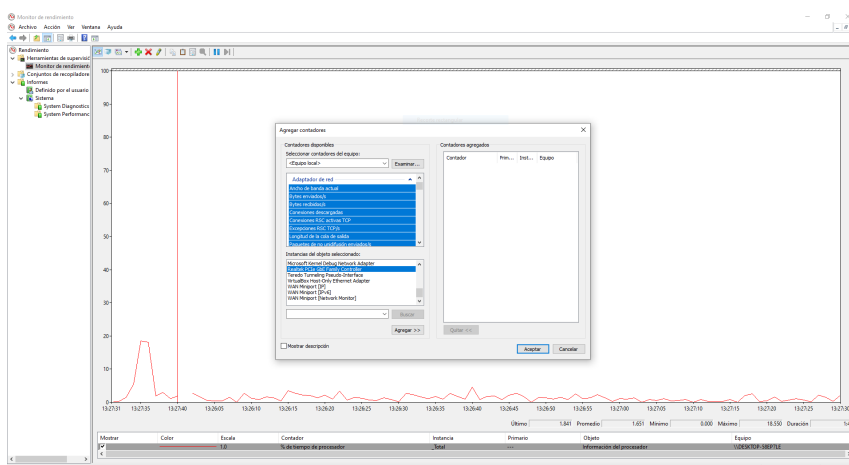


Figura 1.12: Seleccionamos el controlador de red en el monitor de rendimiento

Si seleccionamos el ancho de banda actual, el número de paquetes de salida con errores y el número de paquetes recibidos con errores, en el caso de que hubiera algún error en la red aparecería en la gráfica en tiempo real. Lo mismo que acabamos de hacer con la red es posible hacerlo con la memoria RAM, el disco duro, el procesador, puertos USB, la GPU, etc. Por lo tanto se trata de una herramienta muy potente que nos puede ayudar a identificar problemas de forma rápida en el equipo. Pero no se limita simplemente a una gráfica, también podemos personalizar los datos para mostrarlos como un histograma o como un informe de datos.

También es posible personalizar la gráfica modificando los colores de cada uno de los parámetros que hemos añadido, la forma de la línea e incluso su escala con respecto a otros datos.

Estas dos herramientas, el **monitor de recursos** y el **monitor de rendimiento** se tratan de dos herramientas muy poderosas e imprescindibles a la hora de realizar diagnósticos del PC. Cuando ocurre algún fallo o el rendimiento decae y no se sabe la razón, podemos buscar componente a componente diagnosticando dónde se encuentra el problema, podemos cruzar los datos entre ambos monitores y ver el comportamiento del sistema en momentos concretos, identificando el problema y finalmente poder solucionarlo.

## 1.8 Rutas

Cuando queremos acceder a un directorio o a un fichero para modificarlo, eliminarlo o crearlo tenemos que saber su ubicación o su ruta. Esta puede ser de dos tipos absoluta o relativa. La ruta absoluta en Windows está dividida en 2 partes:

- Una letra de volumen o unidad seguida del separador de volumen (:).
- Un nombre de directorio. El carácter separador de directorios separa los subdirectorios dentro de la jerarquía de directorios anidado
- Un nombre de archivo opcional. El carácter separador del directorio separa la ruta del archivo y el nombre del archivo.

Una ruta relativa hace referencia a una ubicación que es relativa a un directorio actual. Las rutas relativas utilizan dos símbolos especiales, un punto (.) y dos puntos seguidos (..), lo que significa el directorio actual y el directorio padre. Los dos puntos seguidos se utilizan para subir en la jerarquía. Un único punto representa el directorio actual.

Por ejemplo, si tenemos el siguiente árbol de directorios:

```
C:.\n|  fich4 . txt\n|\n+---dirB\n|  |  fich1 . txt\n|  |  fich2 . txt\n|  |\n|  \---dirD\n|          fich5 . txt\n|\n\---dirC\n        fich3 . txt
```

y nos encontramos en dirD, su ruta absoluta sería C:\dirD. Si desde este directorio queremos acceder al fichero fich3.txt, su ruta relativa sería: ../../dirC/fich3.txt



## 2. Símbolo de Sistema

### 2.1 Introducción

Los scripts por lotes se almacenan en simples archivos de texto que contienen líneas con comandos que se ejecutan en secuencia, uno tras otro. Estos archivos tienen la extensión especial BAT o CMD. Los archivos de este tipo son reconocidos y ejecutados a través de una interfaz (a veces llamada shell) proporcionada por un archivo del sistema llamado intérprete de comandos. En los sistemas Windows, este intérprete se conoce como cmd.exe.

Normalmente, para crear un archivo por lotes, se utiliza el bloc de notas. Esta es la herramienta más sencilla para la creación de archivos por lotes. Lo siguiente es el entorno de ejecución de los scripts por lotes. En los sistemas Windows, esto se hace a través del símbolo del sistema o cmd.exe. Todos los archivos por lotes se ejecutan en este entorno.

Para ejecutar archivos por lotes desde el símbolo del sistema, debe ir a la ubicación donde se almacena el archivo por lotes o, alternativamente, puede introducir la ubicación del archivo en la variable de entorno path.

### 2.2 Gestión de ficheros

En Windows nos encontramos con dos tipos de ficheros, estos pueden ser ficheros ejecutables, como por ejemplo aquellos con extensión .exe o .bat y aquellos que no son ejecutables y contienen otro tipo de información.

#### 2.2.1 Crear archivos

El símbolo del sistema ofrece dos comandos diferentes para crear un nuevo archivo en la línea de comandos. El primer comando es “echo” y el segundo es “fsutil”. Ambos tienen sus casos de uso.

Por ejemplo, si quieres crear un archivo con algún contenido en él, se utiliza “echo”. Esto es muy útil cuando quieres guardar la salida de un comando en un archivo. Por otro lado, para crear un archivo en blanco o un archivo con un tamaño específico, usa el comando “fsutil”.

## ECHO

El comando **echo** escribe texto con formato en la salida estándar. Esta salida por defecto es la terminal, pero también se puede redirigir para crear un archivo de texto. Para ello, es suficiente con poner la siguiente instrucción `echo mensaje > fichero.txt`. De esta forma crearemos un fichero de texto con nombre “fichero.txt” con el contenido “mensaje”. En el caso de que queramos incluir texto con espacios podemos entrecomillarlo, como se muestra en el siguiente ejemplo:

### ■ Example 2.1 ECHO

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\SI>echo "Hola Mundo" > fichero.txt
```

## FSUTIL

El comando **FSUTIL** puede hacer una gran variedad de cosas, incluyendo la creación de nuevos archivos desde la línea de comandos. La ventaja de **FSUTIL** es que le permite crear archivos desde la línea de comandos sin ningún contenido en ellos o archivos con un tamaño específico inicializado con ceros. La sintaxis para crear un nuevo fichero con **FSUTIL** es la siguiente: `fsutil file createnew <filename> <length>` donde indicamos primero el nombre del fichero (<filename>) y el tamaño del mismo (<length>) en bytes.

En el siguiente ejemplo, crearemos un fichero vacío llamado `fichA.txt`

### ■ Example 2.2 FSUTIL

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.


C:\SI>fsutil file createnew fichA.txt 0
```

## 2.2.2 Borrar archivos

### DEL

Si queremos borrar ficheros utilizaremos el comando **DEL**. Este comando tiene una serie de opciones, entre ellas podemos encontrarnos con:

- /P Pide confirmación antes de eliminar un fichero.
- /S Elimina los archivos de forma recursiva de todos los subdirectorios.
- /F Fuerza la eliminación de un archivo.

 Nota: Este comando solo borra ficheros, no directorios.

En el siguiente ejemplo borramos el fichero “hola.txt” con confirmación.

### ■ Example 2.3 DEL

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\ferqui>del /p hola.txt
C:\Users\ferqui\hola.txt, Eliminar (S/N)? S

C:\Users\ferqui>
```

## 2.2.3 Ver contenido de archivos

### TYPE

Para ver un archivo completo sin modificarlo en la terminal, usaremos el comando **TYPE**. Tiene la siguiente sintaxis:

type [`<drive>`][`<path>`]`<filename>`

Donde simplemente le indicamos la dirección del archivo que queremos ver. Si `<filename>` contiene espacios, debe encerrarlo entre comillas (por ejemplo, “nombre de archivo que contiene espacios.txt”). También puede agregar varios nombres de archivo agregando espacios entre ellos.

### MORE

El comando **MORE** realiza una paginación sobre los datos de entrada, es decir, nos muestra a trozos la información en vez de mostrarla toda a la vez. Pulsando una tecla podemos avanzar en el texto que le pasemos de entrada. Podemos hacer uso de redirección y tuberías para usarlo, como se muestra en su sintaxis:

```
<command> | more [/c] [/p] [/s] [/t<n>] [+<n>]
more [[/c] [/p] [/s] [/t<n>] [+<n>]] < [<drive>:][<path>]<filename>
more [/c] [/p] [/s] [/t<n>] [+<n>] [<files>]
```

Donde las opciones son las siguientes:

- **/s** Muestra varias líneas en blanco como una sola línea en blanco.
- **+n** Empieza mostrando el primer archivo en la línea n

A parte también tiene otros subcomandos que podemos ir usando que son:

- **P n** Mostrar las siguientes n líneas
- **S n** Saltarse las siguientes n líneas
- **F** Mostrar el siguiente archivo
- **Q** Salir
- **=** Mostrar el número de línea
- **?** Mostrar la línea de ayuda
- **<espacio>** Mostrar la siguiente página
- **<entrar>** Mostrar la siguiente línea

## 2.2.4 Copiar y mover

### COPY

Para copiar ficheros de una localización a otra utilizaremos el comando **COPY**. Si en vez de indicarle un fichero le indicamos una carpeta, se copiarán esta y todos los ficheros que se encuentren dentro, pero las carpetas anidadas no se copiarán.

```
COPY [/D] [/V] [/N] [/Y | /-Y] [/Z] [/L] [/A | /B] origen [/A | /B]
      [+ origen [/A | /B] [+ ...]] [destino [/A | /B]]
```

Donde los parámetros son los siguientes:

- **/V** Comprueba que los nuevos archivos están escritos correctamente.
- **/Y** Suprime el mensaje para confirmar que desea sobrescribir un archivo de destino existente.
- **/-Y** Le pide que confirme si desea sobrescribir un archivo de destino existente.
- **/A** Indica un archivo de texto ASCII.
- **/B** Indica un archivo binario.
- **origen** Especifica la ubicación desde la que desea copiar un archivo o un conjunto de archivos. El origen puede constar de una letra de unidad y dos puntos, un nombre de directorio, un nombre de archivo o una combinación de estos.
- **destino** Especifica la ubicación en la que desea copiar un archivo o un conjunto de archivos. El destino puede constar de una letra de unidad y dos puntos, un nombre de directorio, un nombre de archivo o una combinación de estos.

Si en el origen especificamos un conjunto de ficheros y en el destino un nombre de un único fichero, se fusionarán todos en el fichero destino.

### ■ Example 2.4 COPY

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\ferqui>copy holamundo.txt C:\Users\ferqui\dirA
        1 archivo(s) copiado(s).

C:\Users\ferqui>
```

## XCOPY

El comando **XCOPY** copia archivos y árboles de directorios.

```
XCOPY source [destination] [/A | /M] [/D[:date]] [/P] [/S [/E]] [/V] [/W]
[/C] [/I] [/Q] [/F] [/L] [/G] [/H] [/R] [/T] [/U]
[/K] [/N] [/O] [/X] [/Y] [/-Y] [/Z] [/B] [/J]
[/EXCLUDE:file1[+file2][+file3]...] [/COMPRESS]
```

- **source** Especifica los archivos que se van a copiar.
- **destination** Especifica la ubicación o el nombre de los archivos nuevos.
- **/EXCLUDE:file1[+file2][+file3]...** Especifica una lista de archivos que contienen cadenas. Cada cadena debe estar en líneas distintas en los archivos. Cuando cualquiera de esas cadenas coincida con cualquier parte de la ruta de acceso del archivo que se va a copiar, el archivo no se copiará. Por ejemplo, si se especifica una cadena como \obj\ o .obj, no se tendrán en cuenta todos los archivos del directorio obj o todos los archivos con la extensión .obj.
- **/P** Pide confirmación antes de crear cada archivo de destino.
- **/S** Copia directorios y subdirectorios excepto los que están vacíos.
- **/E** Copia directorios y subdirectorios, incluyendo los que están vacíos.
- **/H** Copia también los archivos de sistema y los ocultos.
- **/R** Sobrescribe archivos de solo lectura.
- **/T** Crea una estructura de directorios, pero no copia archivos. No incluye directorios o subdirectorios vacíos. **/T /E** incluye directorios y subdirectorios vacíos.
- **/U** Copia solo los archivos que ya existen en el destino.
- **/N** Realiza la copia mediante los nombres cortos generados.
- **/Y** Suprime la solicitud para confirmar se quiere sobrescribir un archivo de destino existente.
- **/-Y** Realiza la solicitud para confirmar si se quiere sobrescribir un archivo de destino existente.

El comando **xcopy** no se comporta de igual forma que **copy** cuando indicamos varios ficheros de origen y uno de destino. No fusiona los ficheros de entrada.

En el siguiente ejemplo se está copiando todos los archivos y subdirectorios (incluidos los subdirectorios vacíos) de la unidad A a la unidad B. Con la opción **/H** también incluimos los archivos ocultos y del sistema.

### ■ Example 2.5 XCOPY

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\SI>xcopy a: b: /s /e /h
```

## MOVE

El comando **MOVE** es diferente a los dos anteriores, en vez de copiar los archivos y directorios en otro lugar, los mueve. La sintaxis es la siguiente:

```
move [{/y|-y}] [<source>] [<target>]
```

Donde **source** indica los ficheros de origen, **target** la ruta de acceso y el nombre al que se van a trasladar los archivos. **/Y** Suprime la solicitud de confirmar si quiere sobrescribir un archivo de destino ya existente. **/-Y** Aparecerá la solicitud para confirmar si desea sobrescribir un archivo de destino ya existente.

En el siguiente ejemplo migramos todos los ficheros con extensión **xls** del directorio `\data` al directorio `\reports`

#### ■ Example 2.6 MOVE

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\SI>move \data\*.xls \reports\
```

### 2.2.5 Renombrar

Para cambiar el nombre a un fichero o directorio podemos usar los comandos **REN** o **RENAME** indistintamente. La sintaxis que usan es la siguiente:

```
ren [<drive>:][<path>]<filename1> <filename2>
rename [<drive>:][<path>]<filename1> <filename2>
```

Donde les indicamos la ruta del fichero o directorio y el nuevo nombre que le queremos dar.

### 2.2.6 Atributos de ficheros

Los atributos de archivo son un tipo de metadatos que describen y pueden modificar el comportamiento de los archivos y/o directorios en un sistema de archivos. Los atributos de archivo típicos pueden, por ejemplo, indicar o especificar si un archivo es visible, modificable, comprimido o cifrado. La disponibilidad de la mayoría de los atributos de archivo depende del soporte del sistema de archivos subyacente (como FAT, NTFS, ext4) donde los datos de los atributos deben ser almacenados junto con otras estructuras de control. Los atributos se consideran distintos de otros metadatos, como las fechas y horas, las extensiones de los nombres de los archivos o los permisos del sistema de archivos. Además de los archivos, las carpetas, los volúmenes y otros objetos del sistema de archivos pueden tener atributos.

Tradicionalmente, en DOS y Microsoft Windows, los archivos y carpetas aceptaban cuatro atributos que serán los que vamos a tratar en esta unidad:

- **Sólo lectura:** Permite leer un archivo, pero no se puede escribir en él ni modificarlo.
- **Almacenamiento:** Indica a la copia de seguridad de Windows que haga una copia de seguridad del archivo.
- **Sistema:** Archivo de sistema.
- **Oculto:** El archivo no se muestra cuando se hace un listado.

A medida que salían nuevas versiones de Windows, Microsoft ha ido añadiendo al inventario de atributos disponibles en el sistema de archivos NTFS, incluyendo, entre otros, los siguientes:

- **Comprimido:** Cuando se establece, Windows comprime el archivo al almacenarlo.
- **Cifrado:** Cuando se establece, Windows encripta el archivo de alojamiento durante el almacenamiento para evitar el acceso no autorizado.
- **No indexado por contenido:** Cuando se establece, el Servicio de Indexación o la Búsqueda de Windows no incluyen el archivo en su operación de indexación.

Finalmente, otros atributos que nos podemos encontrar son los siguientes:

- **Directorio:** Indica que se trata de un subdirectorio, que contiene entradas de archivos y directorios propios.
- **Punto de Reparación:** El archivo o directorio tiene un punto de reparación asociado, o es un enlace simbólico.
- **No indexado:** El archivo no está indexado en el dispositivo.
- **Fuera de línea:** Los datos del archivo se mueven físicamente al almacenamiento remoto.
- **Disperso:** El archivo es un archivo disperso, es decir, su contenido está parcialmente vacío y no es contiguo.
- **Temporal:** El archivo se utiliza para el almacenamiento temporal.

**ATTRIB**

El comando **ATTRIB** muestra, establece o quita atributos asignados a archivos o directorios. Si se usa sin parámetros, attrib muestra los atributos de todos los archivos del directorio actual.

La sintaxis es la siguiente;

```
attrib [{+|-}r] [{+|-}a] [{+|-}s] [{+|-}h] [{+|-}i] [<drive>:][<path>][<filename>] [/s [/d [/l]]
```

Donde tenemos los siguientes parámetros:

- **{+|-}r** Establece ( + ) o borra ( - ) el atributo de archivo de solo lectura.
- **{+|-}a** Establece ( + ) o borra ( - ) el atributo de archivo de almacenamiento. Este conjunto de atributos marca los archivos que han cambiado desde la última vez que se realizó una copia de seguridad. Tenga en cuenta que el comando xcopy usa atributos de archivo.
- **{+|-}s** Establece ( + ) o borra ( - ) el atributo de archivo del sistema. Si un archivo utiliza este conjunto de atributos, debe borrar el atributo antes de poder cambiar cualquier otro atributo del archivo.
- **{+|-}h** Establece ( + ) o borra ( - ) el atributo de archivo oculto. Si un archivo utiliza este conjunto de atributos, debe borrar el atributo antes de poder cambiar cualquier otro atributo del archivo.
- **{+|-}i** Establece ( + ) o borra ( - ) el atributo de archivo no indexado de contenido.
- **[<drive>:][<path>][<filename>]** Especifica la ubicación y el nombre del directorio, el archivo o el grupo de archivos para los que desea mostrar o cambiar atributos.
- **/S** Procesa archivos que coinciden en la carpeta actual y todas las subcarpetas.
- **/D** También procesa carpetas.
- **/L** Aplica el atributo al vínculo simbólico, en lugar del destino del vínculo simbólico.

## 2.3 Gestión de directorios

### 2.3.1 Movernos entre directorios

#### CD

Este comando sirve para movernos entre directorios o para mostrar cuál es el directorio actual.

#### ■ Example 2.7 CD

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\ferqui>cd ..\..

C:\>cd
C:\

C:\>cd \Program Files

C:\Program Files>
```

### 2.3.2 Crear carpetas

#### MD

El comando **MD** crea un directorio o subdirectorio. Las extensiones de comandos, que están habilitadas de forma predeterminada, permiten usar un único comando MD para crear directorios intermedios en una ruta de acceso especificada. La sintaxis es la siguiente:

```
md [<drive>:][<path>
```

Donde **<drive>** especifica la unidad en la que desea crear el nuevo directorio y **<path>** es un parámetro obligatorio que indica el nombre y la ubicación del nuevo directorio.

Por ejemplo para crear un directorio denominado “dirA” en el directorio actual, escribimos:



```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\SI>md dirA
```

Para crear el árbol de directorios dirA\dirB\dirC en el directorio raíz, escribimos:

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\SI>md \dirA\dirB\dirC
```

### 2.3.3 Borrar carpetas

#### RMDIR

Para borrar un directorio utilizamos el comando **RMDIR**, sin embargo tiene una serie de limitaciones:

No se puede eliminar un directorio que contenga archivos, incluidos los archivos ocultos o del sistema. Si se intenta hacerlo, aparece un mensaje de error. Con el comando `dir /a` podemos mostrar todos los archivos (incluidos los archivos ocultos y del sistema). Luego, con el comando `attrib` con `-h` para quitar atributos de archivo ocultos, `-s` para quitar atributos de archivo de sistema o `-h-s` para quitar los atributos de archivo ocultos y del sistema. Una vez quitados los atributos de archivo y ocultos, se pueden eliminar los archivos.

No se puede usar el comando `rmdir` para eliminar el directorio actual. Si se intenta eliminar el directorio actual, aparece un mensaje de error. En ese caso hay que cambiar a un directorio **superior** diferente.

La sintaxis es la siguiente:

```
rmdir [<drive>:]<path> [/S [/Q]]
```

Donde `<path>` indica la ruta del directorio que queremos eliminar, la opción `/S` quita todos los directorios y archivos del directorio además del mismo directorio. Por último la opción `/Q` es el modo silencioso, no pide confirmación para quitar un árbol de directorio con `/S`

### 2.3.4 Listar directorios

#### DIR

Para listar el contenido de un directorio utilizamos el comando **DIR**. Este comando tiene una serie de opciones, entre ellas podemos encontrarnos con:

- `/A` Muestra los archivos con un determinado atributo, algunos son:
  - D Directorios
  - H Archivos ocultos
  - S Archivos del sistema
  - R Archivos de solo lectura
  - - Prefijo de exclusión
- `/B` Utiliza el formato simple.
- `/S` Muestra los archivos del directorio especificado y todos sus subdirectorios.
- `/O` Muestra los archivos en un determinado orden según la opción que le pasemos como parámetros:
  - N Por nombre
  - S Por tamaño
  - E Por extensión
  - D Por fecha y hora
  - G Agrupa primero los directorios
  - - Prefijo para invertir el orden.

En el siguiente ejemplo listamos el directorio dirA

### ■ Example 2.8 DIR

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\ferqui>dir dirA
El volumen de la unidad C no tiene etiqueta.
El número de serie del volumen es: 64CD-BCD7

Directorio de C:\Users\ferqui\dirA

27/03/2021  22:01    <DIR>          .
27/03/2021  22:01    <DIR>          ..
27/03/2021  22:01                7 fileA
27/03/2021  22:03            11 fileB
                2 archivos                18 bytes
                2 dirs  113'739'362'304 bytes libres

C:\Users\ferqui>
```

### Tree

Muestra gráficamente la estructura de directorios de una ruta o del disco de una unidad. La estructura mostrada por este comando depende de los parámetros que se especifiquen en la línea de comandos. Si no se especifica una unidad o una ruta, este comando muestra la estructura de árbol empezando por el directorio actual de la unidad actual.

TREE [unidad :][ ruta ] [/F] [/A]

Donde los parámetros indican:

- /F Muestra los archivos de cada carpeta.
- /A Usa ASCII en lugar de caracteres extendidos.

A continuación podemos ver dos ejemplos:

### ■ Example 2.9 Comando TREE sin indicar la ruta

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\SI>tree /A
Listado de rutas de carpetas
El número de serie del volumen es 64CD-BCD7
C:.
+---dirA
|   \---dirC
\---dirB

C:\SI>
```

### ■ Example 2.10 Comando TREE indicando la ruta y el parámetro /F

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\SI>tree dirA /A /F
Listado de rutas de carpetas
El número de serie del volumen es 0000002D 64CD:BCD7
C:\SI\DIRA
|   fich1.txt
|   fich2.txt
|
\---dirC
     fich3.txt

C:\SI>
```

### 2.3.5 Otros comandos útiles

#### VER

El comando **VER** muestra la versión de MS-DOS que se está utilizando

#### ■ Example 2.11 VER

```
Microsoft Windows [Versión 10.0.19042.867]
(c) 2020 Microsoft Corporation. Todos los derechos reservados.

C:\Users\ferqui>ver

Microsoft Windows [Versión 10.0.19042.867]
C:\Users\ferqui>
```

#### CLS

Si queremos limpiar la pantalla utilizaremos el comando **CLS**

## 2.4 Redireccionamiento y filtros

En el Símbolo del Sistema, además de contar con los comandos vistos para la gestión de ficheros y directorios, existen también otros mecanismos que permiten enviar el resultado de la ejecución de un comando a otro dispositivo o asociar varios comandos de forma sucesiva y realizar tareas más complejas, como la búsqueda de un texto dentro de un árbol de directorios. Para realizar estas labores contamos con el redireccionamiento y los filtros

### 2.4.1 Redireccionamiento

En algunas ocasiones puede ser interesante e incluso necesario cambiar la entrada y salida por defecto. Para ello necesitamos **redireccionar** a otro dispositivo de entrada o salida. Para realizar esto, tenemos los siguientes operadores:

#### Operador <

Este operador redirecciona la entrada por defecto. Consiste en cambiar a otro dispositivo o fichero de entrada estándar.

Si nosotros grabamos en un fichero “fecha.txt” una fecha (15/12/97) y queremos hacer que el sistema tomase siempre la misma fecha, tenemos que redireccionar el fichero a la orden DATE. De esta forma estamos consiguiendo que la orden tome como entrada los datos del fichero en el que hemos grabado la fecha. De esta forma la orden quedaría como `C:\DATE<fecha.txt`

En el símbolo del sistema tenemos una serie de comandos que están diseñados para ser usados con el redireccionamiento de entrada. Estos son:

- SORT: Permite ordenar los datos de entrada.
- MORE: Permite visualizar el contenido de los datos de entrada, fraccionándolo en ventanas.
- FIND: Busca una cadena de caracteres dentro de los datos de entrada.

#### Redireccionamiento de salida >

Esta redirección es utilizada para dos operaciones: enviar la salida de un programa a la impresora o enviar la salida de un programa a un fichero.

Si lo que queremos es redireccionar la salida de un programa u orden concreta a un dispositivo diferente a la salida estándar, bastará con poner la orden concreta y, a continuación el símbolo > y el fichero o dispositivo al que se quiera redireccionar. Por ejemplo:

```
C:\DIR>FILE.TXT
```

### Adición >>

Este operador envía también la salida de un comando a un fichero, pero a diferencia del redireccionamiento de salida, la información original no se destruye, si no que se conserva y se le añade la nueva información al final del fichero. En el caso de que el fichero no existiera, se crearía.

## 2.4.2 Filtros

Con el redireccionamiento podemos desviar o cambiar la salida estándar por un dispositivo o un fichero, pero no podemos desviar la información a un programa para que la tome como datos de entrada. Con los filtros se puede transformar la salida de un programa en entrada para otro. En este sentido tenemos tres comandos en CMD a los cuales se les envía información tras ejecutar otra orden y modificar la salida. Estos son:

### More

El comando More ya lo hemos visto anteriormente, recibe como entrada un fichero de texto, devolviendo como salida el mismo fichero, pero en este caso, paginado. Sin embargo, podemos utilizarlo con otras sintaxis para paginar la salida de otros comandos. La nueva sintaxis que vamos a ver son las siguientes:

#### ■ Example 2.12 MORE

```
C:\command | more
```

O también

```
C:\more < fichero.txt
```

### Sort

El filtro **SORT** recibe de entrada una serie de datos, ya sea de un fichero o procedente de otro comando. Devuelve esos datos ordenados por el primer carácter de cada fila. La ordenación se realiza en función del código ASCII del primer carácter de cada línea. Tiene dos parámetros que son:

- /R Ordena de forma inversa
- /+n Indica el carácter a partir del cual se va a comparar cada fila.

### Find

Recibe como entrada un conjunto de caracteres organizados por filas y devuelve las filas que contengan una cadena de texto determinada. Los parámetros que tienen los los siguientes:

- /V Muestra todas las líneas que no tengan la cadena especificada.
- /C Muestra cuántas líneas tienen la cadena especificada.
- /N Muestra el número de las líneas que tienen la cadena especificada
- /I Omite mayúsculas y minúsculas.

## 2.5 Procesamiento por lotes

Un archivo por lotes es un archivo de script en DOS, OS/2 y Microsoft Windows. Consiste en una serie de comandos a ejecutar por el intérprete de la línea de comandos, almacenados en un archivo de texto plano. Un archivo por lotes puede contener cualquier comando que el intérprete acepte de forma interactiva y utilizar construcciones que permitan la ramificación condicional y el bucle dentro del archivo por lotes, como las etiquetas IF, FOR y GOTO. El término “lote” proviene del procesamiento por lotes, que significa “ejecución no interactiva”.

Un ejemplo de fichero por lotes puede ser el siguiente:

#### ■ Example 2.13 Ejemplo script batch

```
C:\>copy con script.bat
REM Esto es un comentario
echo Mi primer script batch
pause
^Z
```

Si ejecutamos el script nos muestra por pantalla “Esto es un comentario” y nos pide que pulsemos un botón para continuar

```
C:\>script.bat
C:\>REM Esto es un comentario
C:\>echo Mi primer script batch
Mi primer script batch
C:\>pause
Presione una tecla para continuar...
```

Como podeis ver, al ejecutar el script no solamente se muestra lo que nosotros queremos que se muestre con la instrucción echo, si no también cada una de las instrucciones que se están ejecutando. Para omitir esto que solamente se muestre lo que nosotros queramos que se imprima por pantalla, tenemos que desactivar el modo “echo” con la instrucción @echo off al principio del programa. Obteniendo como resultado la siguiente salida:

```
C:\>script.bat
Mi primer script batch
Presione una tecla para continuar...
```

Todos los ficheros por lotes tiene que tener una serie de características, estas son:

1. Tienen que tener la extensión BAT
2. Se tratan de ficheros ASCII, por lo que puede ser creado y editado con copy o cualquier editor de texto.
3. Para ejecutar el script solamente hace falta teclear su nombre en el símbolo del sistema.
4. Podemos detener en cualquier momento el procesamiento del fichero pulsando la combinación de teclas Ctrl+C o Ctrl+Pausa

## Órdenes

Todos los comandos que hemos visto anteriormente se pueden utilizar dentro de un fichero por lotes, pero existen una serie de órdenes que están especialmente diseñadas para este tipo de ficheros. Estas son:

- CALL: Llama a un fichero por lotes desde otro
- SET: Asigna un valor a una variable
- REM: Introduce un comentario
- PAUSE: Pausa la ejecución del archivo por lotes
- GOTO: Desvía el desarrollo del programa hacia una etiqueta
- IF: Desvía condicionalmente el flujo del programa.
- FOR: Repite una misma operación en un conjunto de ficheros.
- SHIFT: Desplaza el valor de los parámetros.

También tenemos una serie de símbolos especiales que necesitamos para acceder a variables. Estos símbolos son:

Símbolo	Descripción
%etiqueta	Nombre de una etiqueta
%número	Parámetro del fichero por lotes
%variable%	Variable del entorno
% %variable	Variable de la orden FOR

## REM

La orden REM nos permite introducir comentarios internos en el código para que sea más legible o para introducir alguna aclaración.

## Gestión de parámetros

Los parámetros son informaciones adicionales colocadas detrás del nombre de una orden o comando. Al igual que ocurre con los comandos que hemos visto para gestión de ficheros y directorios, los ficheros por lotes también admiten tener parámetros de entrada. Para mostrar eso, vamos a hacer un fichero por lotes que borre los ficheros introducidos como parámetros.

### ■ Example 2.14 Script batch de ejemplo

```
@echo off
rem Borra los ficheros que se le indica por parámetro al script.
del %1
del %2
```

Para ejecutar el comando basta con poner el nombre del fichero por lotes seguido de los parámetros.

```
C:\>borra.bat file1.txt file2.txt
```

Cada parámetro debe de estar separado por un espacio en blanco. Para acceder a estas variables dentro del script utilizamos el símbolo visto anteriormente, es decir, el símbolo de porcentaje seguido de un número del 1 al 9, (%1,%2...%9)

## Pause

La orden **pause** detiene temporalmente el desarrollo de un programa y pide su continuación cuando presionamos una tecla, mostrando el siguiente mensaje *Presione cualquier tecla para continuar*.

Para dar más información al usuario sobre la detención del programa, podemos insertar un mensaje con el motivo mediante el comando echo. Si no deseamos que se muestre el mensaje de pausa, también podemos redireccionarlo al dispositivo **NUL**

```
echo Pulse enter para continuar
pause > nul
```

## GOTO

Normalmente un fichero por lotes se ejecuta de forma secuencial. Sin embargo, mediante **GOTO** podemos desviar el flujo de ejecución del programa hasta una etiqueta. Esta debe de ir especificada con dos puntos (:). Por ejemplo

### ■ Example 2.15 Ejemplo de GOTO

```
inicio:
@echo off
ver
goto final
vol
:final
```

La salida producida será:

```
Microsoft Windows [Versión 10.0.19042.867]
```

## IF

La orden **IF** desvía condicionalmente el flujo de ejecución del fichero por lotes. Puede tener hasta 4 sintaxis diferentes:

- IF [NOT] EXIST archivo comando
- IF [NOT] cadena1==cadena2 comando
- IF [NOT] ERRORLEVEL número comando

- **IF [NOT] DEFINED** variable comando

En caso del **IF [NOT] EXIST** nos indica si un fichero existe o no, por ejemplo, si nuestro programa tiene que borrar un fichero que recibe por parámetros, antes deberíamos de comprobar que exista para evitar mensajes de error.

En el segundo caso, estamos comparando dos variables, por ejemplo, podemos usarlo para comprobar si uno de los parámetros introducidos está vacío.

Cada orden externa genera un código de salida que indica si la acción se realizó correctamente. Generalmente el código 0 indica que no hubo ningún problema y un código superior hace referencia a distintos errores. En algunos casos es necesario saber que no hubo ningún error y para ello usamos la orden **IF ERRORLEVEL**. En este caso, la orden se ejecutará si el código de salida es igual o superior al indicado detrás de **ERRORLEVEL**

- **Example 2.16** Ejemplo de **IF ERRORLEVEL**

```
xcopy a:\ b:\
if errorlevel 1 goto error
if errorlevel 0 echo Se ha realizado la copia correctamente.
goto final

:error
echo Se ha producido un error durante la copia

:final
```

## CHOICE

Con **CHOICE** podemos hacer que el usuario pueda escoger entre una o varias opciones, dependiendo de la sintaxis que usemos:

**CHOICE [mensaje] [/C:opciones]**

**/C:opciones** Especifica las distintas opciones posibles. Si el usuario pulsa la primera de ellas, choice devolverá un código de salida 1; si pulsa la segunda opción, choice devuelve 2, y así sucesivamente. Si no se especifica nada, se asumen las opción S y N

- **Example 2.17** Ejemplo **CHOICE**: En este ejemplo se pregunta al usuario si quiere continuar, en ese caso el comando choice devolverá un 1 en errorlevel, si indica N, entonces el valor de errorlevel será 2 y terminará el programa.

```
choice Quieres continuar?
if errorlevel 2 goto final
...
:final
```

## SET

También es posible inicializar variables dentro de nuestro script con el comando **set** siguiendo la siguiente sintaxis:

**set [/A] variable=valor**

Donde **variable** es el nombre de la variable, **valor** es el valor con el que inicializamos la variable. El modificador **/A** indica que el valor es numérico.

En cualquier lenguaje de programación, existe la opción de marcar las variables con algún tipo de ámbito. Normalmente, las variables que tienen un ámbito global pueden ser accedidas en cualquier lugar de un programa, mientras que las variables de ámbito local tienen un límite definido en el que pueden ser accedidas.

Los scripts del Símbolo del Sistema también tienen una definición para las variables de ámbito local y global. Por defecto, las variables son globales para toda la sesión de comandos. Para hacer variables locales al script, hay que llamar al comando **SETLOCAL**. Después de llamar a **SETLOCAL**, cualquier asignación de variables se revierte al llamar a **ENDLOCAL**, llamar a

**EXIT**, o cuando la ejecución alcanza el final del archivo (EOF) en su script. El siguiente ejemplo muestra la diferencia cuando se establecen variables locales y globales en el script.

■ **Example 2.18** Variables globales y locales

```
@echo off
set globalvar = 5
SETLOCAL
set var = 13145
set /A var = %var% + 5
echo %var%
ENDLOCAL
echo %globalvar%
```

En este ejemplo, la variable `var` se destruye inmediatamente después del comando `ENDLOCAL`, sin embargo la variable `globalvar` permanece incluso después de terminar la ejecución del script.

## FOR

La orden “FOR” ofrece capacidades de bucle para los archivos por lotes. A continuación se muestra la construcción común de la sentencia “for” para trabajar con una lista de valores.

```
FOR %%variable IN lista DO codigo
```

Esta orden repite el código *codigo* para cada valor que se encuentra en *lista*. *Lista* se trata de un conjunto de nombres de ficheros. En ella, se pueden establecer varios nombres separados por espacios y también utilizar comodines.

El código for consiste en tres partes

1. **Declaración de variables.** Este paso se ejecuta sólo una vez para todo el bucle y se utiliza para declarar las variables que se utilizarán dentro del bucle. En Batch Script, la declaración de variables se realiza con el `%%` al principio del nombre de la variable.
2. **Lista.** La lista de valores para los que se debe ejecutar la sentencia “for”.
3. El **código** que se ejecutará en cada iteración del bucle.

Por ejemplo, en la instrucción `lfor %%i in (file1.txt file2.txt *.dat) do type %%i` se va tomando cada uno de los valores del conjunto y se los envía a la orden `type`. En este ejemplo, se visualizarán los ficheros `file1.txt`, `file2.txt` y todos los que tengan la extensión `dat`.

## SHIFT

La orden **SHIFT** mueve el valor de cada parámetro a la variable anterior. Por ejemplo, si existen 3 parámetros (`%1`, `%2`, `%3`) y se utiliza la orden `Shift`, el valor de `%1` lo tomará `%0`, el valor de `%2` lo tomará `%1` y el valor de `%3` lo tomará `%2`. Esto es especialmente útil cuando tenemos más de 9 parámetros.

## CALL

Mediante la orden **CALL** podemos llamar a otro fichero por lotes desde el interior del que se está ejecutando. la sintaxis es la siguiente: `CALL fichero [parámetros]`.





## 3. Windows PowerShell

*PowerShell es una mezcla entre un lenguaje de comandos y un lenguaje de programación funcional orientado a objetos. Está basado en .NET, dando una gran flexibilidad que no se encontraba disponible con otros como CMD.*

### 3.1 Introducción

Windows PowerShell es un “Shell” de línea de comandos y un lenguaje de script diseñado para la administración de los sistemas Windows. El análogo en Linux puede ser ZSH o Bash. PowerShell tiene la peculiaridad de que se encuentra construido por encima de .NET, por lo que consta de una serie de funciones y módulos que permiten controlar y automatizar la administración de los sistemas Windows y las aplicaciones de Windows Servers.

En Windows PowerShell los comandos son llamados cmdlets y permiten acceder a los datos como registros o certificados de forma parecida al acceso a sistema de ficheros.

Para la programación en Windows PowerShell podemos utilizar el bloc de notas pero Windows trae el sistema integrado **ISE** (Integrated Scripting Environment). El **ISE** nos permite ejecutar comandos, escribir, testear y depurar scripts en una misma interfaz gráfica.

### 3.2 Sintaxis

En PowerShell los comandos están formados por parejas de verbos y sustantivos separados por un guión. Esto puede ser útil para buscar comandos y que no sea necesario memorizarlos todos.

#### 3.2.1 Verbos

La lista de verbos que puede tener un comando está mantenida por Microsoft y pueden ser: Add, Get, Set y New, entre otros. Para poder ver la lista completa de verbos se puede utilizar el comando **Get-Verb**. Nótese que es la composición de “Coger”-“Verbo”. Cada verbo pertenece a un grupo. Las acciones complementarias, como el cifrado y el descifrado, tienden a utilizar verbos del mismo grupo; por ejemplo, el verbo **Protect** puede utilizarse para cifrar algo y el verbo **Unprotect** puede utilizarse para descifrar algo. Es posible utilizar verbos distintos a los de la lista aprobada. Sin embargo, si un comando que utiliza un verbo no aprobado forma parte de un módulo, se mostrará una advertencia cada vez que se importe el módulo.

### 3.2.2 Nombres

Un sustantivo proporciona una descripción muy breve del objeto sobre el que el comando espera actuar. La parte del sustantivo puede ser una sola palabra, como es el caso de **Get-Process**, **New-Item**, o **Get-Help**, o más de una palabra, como se ve en **Get-ChildItem**, **Invoke-WebRequest**, o **Send-MailMessage**.

### 3.2.3 Buscando comandos

El uso del sistema verbo-sustantivo para nombrar a los comandos puede facilitar mucho la búsqueda de estos (sin recurrir a internet). Por ejemplo, si queremos ver las reglas del firewall y ya conocemos el módulo NetSecurity que está disponible en Windows PowerShell, podemos ejecutar el siguiente comando, que muestra los comandos Get de ese módulo:

```
PS> Get-Command Get-*Firewall* -Module NetSecurity
```

CommandType	Name	Version	Source
Function	Get-NetFirewallAddressFilter	2.0.0.0	NetSecurity
Function	Get-NetFirewallApplicationFilter	2.0.0.0	NetSecurity
Function	Get-NetFirewallInterfaceFilter	2.0.0.0	NetSecurity
Function	Get-NetFirewallInterfaceTypeFilter	2.0.0.0	NetSecurity
Function	Get-NetFirewallPortFilter	2.0.0.0	NetSecurity
Function	Get-NetFirewallProfile	2.0.0.0	NetSecurity
Function	Get-NetFirewallRule	2.0.0.0	NetSecurity
Function	Get-NetFirewallSecurityFilter	2.0.0.0	NetSecurity
Function	Get-NetFirewallServiceFilter	2.0.0.0	NetSecurity
Function	Get-NetFirewallSetting	2.0.0.0	NetSecurity

De la lista anterior, podemos ver que **Get-NetFirewallRule** coincide con el requisito (ver una lista de reglas de firewall).

```
Get-Command Get-*Firewall*
```

Una vez que hemos encontrado un comando que puede sernos útil, utilizaremos **Get-Help** para obtener la ayuda y ver si finalmente es el buscado.

### 3.2.4 Parámetros

Los comandos en PowerShell pueden recibir multitud de parámetros que pueden ser opcionales u obligatorios. En el caso de los parámetros opcionales, se encuentran entre corchetes, como se muestra a continuación:

```
SYNTAX
Get-Process [-ComputerName <String[]>] ...
```

En este caso, si el valor de un parámetro se tiene que especificar, su nombre también se debe de especificar, como se muestra a continuación:

```
Get-Process -ComputerName somecomputer
```

También nos podemos encontrar con parámetros **opcionales posicionales**, estos son los que se encuentran como primer parámetro del comando:

```
SYNTAX
Get-Process [[-Name] <String[]>] ...
```

En este caso, Podemos omitir el nombre del parámetro al usarlo, como se muestra a continuación:

```
Get-Process -Name powershell
Get-Process powershell
```

En el caso de los parámetros **obligatorios**, siempre se deben de definir y están indicados en la ayuda de la siguiente forma:

```
SYNTAX
Get-ADUser -Filter <string> ...
```

En este caso el parámetro **Filter** se debe de escribir y se le debe de dar un valor como en el siguiente ejemplo donde se le aplica un filtro a un comando (esto lo veremos más adelante):

```
Get-ADUser -Filter 'nombre_de_usuario -eq "Nombre"'
```

También nos podemos encontrar con que los primeros parámetros sean obligatorios, por lo que serán obligatorios posicionales y no hace falta indicar su nombre al pasarles un valor. Se indican en la ayuda con su nombre entre corchetes:

```
SYNTAX
Get-ADUser [-Identity] <ADUser> ...
```

Puede haber más de un parámetro obligatorio posicional como se indica a continuación:

```
SYNTAX
Add-Member [-NotePropertyName] <String> [-NotePropertyValue] <Object> ...
```

En este caso, se le puede llamar de distintas formas:

```
Add-Member -NotePropertyName Vombre -NotePropertyValue "valor"
Add-Member -NotePropertyValue "value" -NotePropertyName Nombre
Add-Member Nombre -NotePropertyValue "valor"
Add-Member Nombre "valor"
```

### 3.2.5 Confirm, WhatIf y Force

Los parámetros **Confirm**, **WhatIf** y **Force** se utilizan con comandos que realizan cambios (en archivos, variables, datos, etc.). Estos parámetros se utilizan a menudo con comandos que utilizan los verbos Set o Remove, pero los parámetros no están limitados a verbos específicos. En el caso del parámetro **Confirm** hace que muestre por pantalla una confirmación de que queremos realizar la acción. Por ejemplo, en el siguiente código hemos creado un fichero de texto llamado HolaMundo.txt y luego lo hemos eliminado con el parámetro Confirm, esto hace que se nos pida una confirmación por pantalla si realmente lo queremos eliminar.

```
PS> Set-Location $env:TEMP
PS> New-Item HolaMundo.txt -Force
PS> Remove-Item .\HolaMundo.txt -Confirm

Confirmar
Est á seguro de que desea realizar esta acción?
Se está realizando la operación "Quitar archivo" en el destino "C:\Users\ferqui\AppData
[S] Sí [O] Sí a todo [N] No [T] No a todo [U] Suspende [?] Ayuda (el valor pre
```

El parámetro **WhatIf** simplemente muestra la acción que se hubiera realizado si se hubiera ejecutado el comando, sustituyendo la confirmación por un aviso. Si lo aplicamos en el ejemplo anterior tendríamos lo siguiente:

```
PS> Set-Location $env:TEMP
PS> New-Item HolaMundo.txt -Force
PS> Remove-Item .\HolaMundo.txt -WhatIf

Whatif: Se está realizando la operación "Quitar archivo" en el destino "C:\Users\fer
```

Por último, el parámetro **Force** lo que hace es forzar la ejecución del comando. Por ejemplo, en el caso del comando **New-Item** sobrescribiría cualquier fichero existente con el mismo nombre del que queremos crear. En el caso de **Remove-Item**, si queremos borrar un fichero con los atributos oculto o de sistema, nos produciría un error. Sin embargo, con el parámetro **Force** podemos eliminarlo sin problemas. Este parámetro por tanto hay que tener cuidado al usarlo porque podemos borrar ficheros que no debemos de forma accidental.

### 3.3 Ayuda

Al igual que en los sistemas Linux tenemos el comando `man`, con Powershell tenemos varios comandos para obtener ayuda, actualizarla y guardarla. Se trata de un sistema extensible, donde los autores pueden escribir la ayuda de su propio contenido de una manera fácil. Para ello tenemos los siguientes comandos:

- `Get-Help`
- `Save-Help`
- `Update-Help`

#### 3.3.1 Ayuda actualizable

La ayuda actualizable les da a los autores la opción de guardar la versión más reciente de la ayuda fuera del PowerShell, por ejemplo en servidores web. Los ordenadores que no tiene conexión a internet o que se encuentran detrás de un servidor proxy restrictivo, puede que no se pueda descargar directamente

#### 3.3.2 Comando `Get-Help`

El comando `Get-Help` sin argumentos nos mostrará una ayuda sobre el sistema de ayudas.

```
PS C:\Users\ferqui>> Get-Help

TEMA
    Sistema de ayuda de Windows PowerShell

DESCRIPCION BREVE
    Muestra ayuda acerca de los cmdlets y los conceptos de Windows PowerShell.

DESCRIPCION LARGA
    En la ayuda de Windows PowerShell se describen los cmdlets, las funciones,
    los scripts y los módulos de Windows PowerShell y se explican conceptos,
    incluidos los elementos del lenguaje Windows PowerShell.

    Windows PowerShell no incluye archivos de ayuda, pero puede leer los
    temas de ayuda en línea o usar el cmdlet Update-Help para descargar archivos de
    en el equipo y, a continuación, usar el cmdlet Get-Help para mostrar los temas
    de ayuda en la línea de comandos.

    También puede usar el cmdlet Update-Help para descargar archivos de ayuda
    actualizados a medida que se publiquen, para que el contenido de ayuda local nun

    Sin archivos de ayuda, Get-Help obtiene ayuda generada automáticamente para los
    las funciones y los scripts.
```

Alternativamente, en Windows PowerShell se puede hacer para que el comando `Get-Help` cree una ventana emergente mediante la opción `-ShowWindow`

```
Get-Help default -ShowWindow
```

El contenido de la ayuda disponible se puede listar utilizando cualquiera de los dos comandos siguientes:

```
Get-Help *
Get-Help -Category All
```

Si queremos ver la ayuda de un comando específico se lo pasamos como parámetro a `Get-Help`

```
Get-Help <comando>
```

Si un documento de ayuda incluye un enlace a una versión en línea, se puede abrir en un navegador utilizando la opción `-Online`:

```
Get-Help Get-Command -Online
```

El contenido de la ayuda se divide en varias secciones visibles:

- Nombre
- Sinopsis
- Sintaxis
- Descripción
- Enlaces relacionados
- Observaciones.

### Sintaxis

La sección de sintaxis enumera cada una de las posibles combinaciones de parámetros que aceptará un comando; cada una de ellas se conoce como conjunto de parámetros. Un comando que tiene más de un conjunto de parámetros se muestra de la siguiente manera:

```
SYNTAX
Get-Process [[-Name] <String[]>] [-ComputerName <String[]>]
[-FileVersionInfo] [-Module] [<CommonParameters>]

Get-Process [-ComputerName [<String[]>]] [-FileVersionInfo]
[-Module] -InputObject <Process[]> [<CommonParameters>]
```

Los elementos de sintaxis escritos entre corchetes son opcionales; por ejemplo, la ayuda de sintaxis para **Get-Process** muestra que todos sus parámetros son opcionales, como se muestra en el siguiente código:

```
SYNTAX
Get-Process [[-Name] <String[]>] [-ComputerName <String[]>] [-FileVersionInfo] [
```

Get-Process puede ejecutarse sin ningún parámetro, o puede ejecutarse sólo con un valor y sin nombre de parámetro, o puede incluir el nombre del parámetro, así como el valor. Cada uno de los siguientes ejemplos es un uso válido de Get-Process:

```
Get-Process
Get-Process powershell
Get-Process -Name powershell
```

### Ejemplos

La sección de ejemplos de la ayuda suele ser muy valiosa. En algunos casos, un comando es lo suficientemente complejo como para requerir un ejemplo detallado que acompañe a las descripciones de los parámetros; en otros, el comando es simple, y un buen ejemplo puede servir en lugar de leer la documentación de ayuda. Para ver ejemplos de un comando, se puede utilizar la opción **-Examples** del comando **Get-Help**:

```
Get-Help Get-Process -Examples
```

#### 3.3.3 Parámetros

También es posible ver la ayuda de determinados parámetros de un comando, por ejemplo:

```
Get-Help Get-Command -Parameter <Parametro>
```

En esta línea de Código, queremos ver la ayuda del parámetro **<Parametro>** del comando Get-Command

#### Detailed y Full

El parámetro **Detailed** pide a **Get-Help** que devuelva el mayor contenido de ayuda. Esto añade información sobre cada parámetro y el conjunto de ejemplos a nombre, sinopsis, sintaxis y descripción. Usando la opción **Full** añade más detalles técnicos (en comparación con el uso del parámetro Detailed). Las entradas, salidas, notas y enlaces relacionados se añaden a los que se ven usando **Detailed**.

### 3.3.4 Save-Help

El comando **Save-Help** se puede utilizar con módulos que soporten la ayuda actualizable. Guarda el contenido de la ayuda en una carpeta. Por ejemplo, la siguiente instrucción guardaría la ayuda de DnsClient en C:\PSHelp

```
Save-Help -DestinationPath C:\PSHelp -Module DnsClient
```

Esto creará un fichero XML y CAB que contiene toda la ayuda relacionada con el módulo especificado. El contenido de la ayuda guardada puede copiarse a otro ordenador e importarse mediante Update-Help. Esta técnica es muy útil para los ordenadores que no tienen acceso a Internet, ya que permite disponer del contenido de la ayuda.

### 3.3.5 Update-Help

El comando **Update-Help** puede hacer dos tareas:

- Actualizar la ayuda a través de internet.
- Importar la ayuda anteriormente guardada, como se ha visto antes.

Para actualizar la ayuda a través de internet se ejecuta el comando **Update-Help** sin ningún parámetro:

```
Update-Help
```

Al actualizar la información de ayuda desde Internet, por defecto, Update-Help no descargará el contenido de la ayuda más de una vez cada 24 horas. Si quisiéramos forzar la actualización debemos añadir la opción **Force** al comando. Si en vez de actualizar a través de internet quisiéramos actualizar desde un fichero guardado como se hizo anteriormente, lo indicamos con el parámetro **-SourcePath**

```
Update-Help -SourcePath C:\temp
```

En este caso por ejemplo, se actualizaría la ayuda que se encuentra en la carpeta C:\temp

## 3.4 CMDLET básicos

Al estar construido sobre el **Framework .NET** PowerShell consta de una gran cantidad de comandos para la administración de sistemas, sin embargo, vamos a centrarnos algunos de los más comunes y básicos, así como en su sintaxis.

Un **cmdlet** o “Command let” es un comando ligero utilizado en el entorno de Windows **PowerShell**. El entorno de ejecución de Windows PowerShell invoca estos cmdlets en el símbolo del sistema. Puede crearlos e invocarlos mediante programación a través de las API de Windows PowerShell.

Los **cmdlet** se diferencian de los comandos de otros lenguajes de scripting como Batch o Bash en los siguientes aspectos:

- Los **cmdlet** son objetos de clases **.NET**, no solamente ejecutables.
- El análisis sintáctico, la gestión de errores y el formato de salida no son gestionados por los cmdlets. Lo hace el tiempo de ejecución de Windows PowerShell.
- El proceso de los **cmdlets** funciona con objetos, no con flujos de texto, y los objetos pueden pasarse como salida para canalizarlos usando tuberías (de una forma parecida a bash).
- Los **cmdlets** están basados en registros, ya que procesan un solo objeto a la vez.

### 3.4.1 Crear archivos y directorios

#### New-item

Con el comando **New-Item** podemos crear distintos objetos en PowerShell, entre ellos ficheros y directorios. New-Item también puede establecer el valor de los elementos que crea. Por ejemplo,

cuando crea un nuevo archivo, `New-Item` puede añadir contenido inicial al archivo. Puede tener dos sintaxis diferentes, dependiendo de si se utiliza el parámetro `-Name` o no:

#### `New-Item`

```
[ -Path ] <String []>
[ -ItemType <String >]
[ -Value <Object >]
[ -Force ]
[ -Credential <PSCredential >]
[ -WhatIf ]
[ -Confirm ]
[ <CommonParameters >]
```

#### `New-Item`

```
[ [ -Path ] <String [] >]
-Name <String >
[ -ItemType <String >]
[ -Value <Object >]
[ -Force ]
[ -Credential <PSCredential >]
[ -WhatIf ]
[ -Confirm ]
[ <CommonParameters >]
```

En el caso de que estemos creando un directorio o un fichero y no le pasemos el comando `-Name`, se creará con el nombre que le hayamos puesto en el `Path`. A continuación veremos unos cuantos ejemplos de su uso.

#### ■ **Example 3.1** Creación de un fichero de texto con una cadena de texto inicial.

```
New-Item -Path . -Name "fichA.txt" -ItemType "file" -Value "Esto es
un fichero de texto."
```

Como se puede ver en el primer ejemplo, se ha creado un fichero de texto llamado “fishA.txt” en el directorio actual con un contenido inicial.

#### ■ **Example 3.2** Creación de un directorio.

```
New-Item -Path "c:\\" -Name "logs" -ItemType "directory"
New-Item -ItemType "directory" -Path "c:\SI\scripts"
```

En el segundo ejemplo hemos creado un directorio en la raíz del dispositivo C que se llama “logs” y otro directorio llamado scripts en la ruta “C:\SI”.

### 3.4.2 Borrar archivos y directorios

Para borrar elementos utilizamos el cmdlet **Delete-Item**. No solamente puede borrar ficheros y directorios, si no que también se puede usar para eliminar claves de registro, alias, variables y funciones. Puede tener también dos sintaxis diferentes, como le pasaba a **New-Item** dependiendo de los parámetros que se usen:

#### Remove-Item

```
[ -Path ] <String []>
[ -Filter <String >]
[ -Include <String []>]
[ -Exclude <String []>]
[ -Recurse ]
[ -Force ]
[ -Credential <PSCredential >]
[ -WhatIf ]
[ -Confirm ]
[ -Stream <String []>]
[ <CommonParameters >]
```

#### Remove-Item

```
-LiteralPath <String []>
[ -Filter <String >]
[ -Include <String []>]
[ -Exclude <String []>]
[ -Recurse ]
[ -Force ]
[ -Credential <PSCredential >]
[ -WhatIf ]
[ -Confirm ]
[ -Stream <String []>]
[ <CommonParameters >]
```

■ **Example 3.3** Este ejemplo elimina todos los archivos que tienen nombres que incluyen un punto (.) de la carpeta C:\test. Debido a que el comando especifica un punto, el comando no elimina las carpetas o los archivos que no tienen una extensión de nombre de archivo.

```
Remove-Item C:\Test\*.*
```

■ **Example 3.4** Este ejemplo elimina de la carpeta actual todos los archivos que tengan una extensión de archivo .doc y un nombre que no incluya \*1\*. Utiliza el carácter comodín (\*) para especificar el contenido de la carpeta actual. Utiliza los parámetros Incluir y Excluir para especificar los archivos a eliminar.

```
Remove-Item * -Include *.doc -Exclude *1*
```

### 3.4.3 Ver contenido de archivos

Para ver el contenido de los archivos utilizamos el cmdlet **Get-Content** pasando como parámetro el fichero que queremos ver. También es posible utilizar comodines para leer múltiples ficheros a la vez, en ese caso se concatenarán el contenido de los ficheros. Como se puede ver en los siguientes ejemplos.

■ **Example 3.5** En este primer ejemplo imprimimos el contenido del fichero “fich2.txt”

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

PS C:\SI\dirA> Get-Content .\fich2.txt
"Hello world"
```

■ **Example 3.6** En este ejemplo imprimimos el contenido de los ficheros “fich2.txt” y “fich3.txt”

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

PS C:\SI\dirA> Get-Content .\fich[23].txt
"Hello world"
Hola Mundo
```



### 3.4.4 Copiar y mover

Para copiar y mover tanto ficheros como directorios utilizamos los cmdlets **Copy-Item** y **Move-Item**

#### Copy-Item

Para copiar un directorio le pasamos la ruta del directorio a copiar y la ruta de destino donde se va a copiar la carpeta. A continuación mostramos unos ejemplos:

■ **Example 3.7** Este ejemplo copia el archivo 050621.log.txt al directorio C:\backup. El archivo original no se elimina.

```
Copy-Item "C:\logs\050621.log.txt" -Destination "C:\backup"
```

■ **Example 3.8** Este ejemplo copia el contenido del directorio C:\logs en el directorio C:\backup existente. El directorio logs no se copia. Si el directorio logs contiene archivos en subdirectorios, esos subdirectorios se copian con sus árboles de archivos intactos. Por defecto, el parámetro Container está establecido en True, lo que preserva la estructura de directorios.

```
Copy-Item -Path "C:\logs\*" -Destination "C:\backup" -Recurse
```

■ **Example 3.9** Este ejemplo copia el fichero fich1.txt del directorio \SI al directorio \backup. Como parte de la operación de copia, el comando cambia el nombre del elemento de fich1.txt a fich2.txt.

```
Copy-Item "C:\SI\fich1.txt" -Destination "C:\backup\fich2.txt"
```

#### Move-Item

El cmdlet **Move-Item** mueve un elemento, incluyendo sus propiedades, contenidos y elementos hijos, de una ubicación a otra. Por ejemplo, puede mover un archivo o subdirectorio de un directorio a otro. Cuando mueve un elemento, éste se añade a la nueva ubicación y se elimina de su ubicación original. A continuación se muestran varios ejemplos

■ **Example 3.10** Este primer ejemplo mueve el archivo test.txt de la unidad C: al directorio E:\temp y lo renombra de test.txt a tst.txt.

```
Move-Item -Path C:\test.txt -Destination E:\temp\tst.txt
```

■ **Example 3.11** En este ejemplo se mueve el directorio C:\temp y su contenido al directorio C:\Logs. El directorio “temp”, y todos sus subdirectorios y archivos, aparecen entonces en el directorio “Logs”.

```
Move-Item -Path C:\temp -Destination C:\Logs
```

### 3.4.5 Renombrar

Para renombrar ficheros y directorios usaremos el cmdlet **Rename-Item**, aunque ya hemos visto cómo se puede hacer con otros medios, como **Move-Item**. Sin embargo, no es posible mover el objeto a otro lugar. El comando tiene dos parámetros Path y NewName. En el primer parámetro le especificamos el directorio o fichero que queremos renombrar, en el segundo indicamos el nuevo nombre.



Si al parámetro **NewName** le pasamos una dirección producirá un error, solo le podemos poner el nuevo nombre del archivo.

■ **Example 3.12** En este ejemplo cambiamos el nombre del archivo fich1.txt a fich2.txt.

```
Rename-Item -Path "C:\fich1.txt" -NewName "fich2.txt"
```

■ **Example 3.13** También es posible renombrar múltiples ficheros a la vez haciendo uso de tuberías y comodines, como se muestra en este ejemplo.

```
PS C:\SI\dirA> Get-ChildItem *.txt

Directorio: C:\SI\dirA

Mode                LastWriteTime         Length Name
----                -
-a-----         31/03/2021         17:04         231 fich1.txt
-a-----         30/03/2021         21:03          16 fich2.txt
-a-----         01/04/2021         16:12          10 fich3.txt

PS C:\SI\dirA> Get-ChildItem *.txt | Rename-Item -NewName { $_.Name
    -replace '.txt','.log' }
PS C:\SI\dirA> Get-ChildItem *.log

Directorio: C:\SI\dirA

Mode                LastWriteTime         Length Name
----                -
-a-----         31/03/2021         17:04         231 fich1.log
-a-----         30/03/2021         21:03          16 fich2.log
-a-----         01/04/2021         16:12          10 fich3.log

PS C:\SI\dirA>
```

### 3.4.6 Otros comandos útiles

A parte de los cmdlet mencionados anteriormente existen otros muy útiles para movernos entre los directorios

#### Get-ChildItem

Para listar directorios utilizamos el comando **Get-ChildItem**

#### ■ Example 3.14

```
PS C:\SI> Get-ChildItem

Directorio: C:\SI

Mode                LastWriteTime         Length Name
----                -
d-----         01/04/2021         17:05         dirA
d-----         30/03/2021         21:03         dirB
```

```
PS C:\SI> Get-ChildItem dirA
```

```
Directorio: C:\SI\dirA
```

Mode		LastWriteTime		Length	Name
----		-----		-----	----
d-----	31/03/2021	17:26			dirA
d-----	30/03/2021	21:04			dirC
-a----	31/03/2021	17:04		231	fich1.log
-a----	30/03/2021	21:03		16	fich2.log
-a----	01/04/2021	16:12		10	fich3.log

```
PS C:\SI>
```

### Get-Location

El comando **Get-Location** nos permite ver cuál es el directorio actual, al igual que lo haría **PWD** en Linux.

#### ■ Example 3.15

```
PS C:\SI> Get-Location
```

```
Path
```

```
----
```

```
C:\SI
```

```
PS C:\SI>
```

### Set-Location

El comando **Set-Location** nos permite cambiar de directorio al igual que lo haría **CD** en Linux.

#### ■ Example 3.16

```
PS C:\SI> Get-ChildItem
```

```
Directorio: C:\SI
```

Mode		LastWriteTime		Length	Name
----		-----		-----	----
d-----	01/04/2021	17:05			dirA
d-----	30/03/2021	21:03			dirB

```
PS C:\SI> Set-Location dirA
```

```
PS C:\SI\dirA> Get-Location
```

```
Path
```

```
----
```

```
C:\SI\dirA
```

```
PS C:\SI\dirA>
```

### Test-Path

El comando **Test-Path** nos permite comprobar la existencia de ficheros y directorios, pasándole como parámetro la ruta del mismo.

#### ■ Example 3.17 Ejemplo de Test-Path

```
Test-Path .\dir\
```

### 3.4.7 Atributos

Los comandos **Get-ItemProperty** y **Set-ItemProperty** permiten modificar propiedades individuales.

Si recordamos, los cmdlets trabajan con objetos en vez de texto plano, por lo tanto tienen una serie de atributos y métodos. el cmdlet **Get-ItemProperty** nos devuelve las propiedades de un objeto, que pueden ser ficheros o directorios, con un atributo llamado “Attributes” que podemos modificar añadiendo o quitando atributos a los ficheros como **ReadOnly**, **Archive**, **Hidden**, **Normal**.

#### ■ Example 3.18 Asignamos los atributos **ReadOnly** y **Hidden** a un fichero.

```
(Get-Item 'somefile.txt').Attributes = 'ReadOnly, Hidden'
```

#### ■ Example 3.19 Quitamos el atributo **ReadOnly** a un fichero

```
(Get-Item 'somefile.txt').Attributes -= 'ReadOnly'
```

#### ■ Example 3.20 Añadimos el atributo **ReadOnly** a un fichero

```
(Get-Item 'somefile.txt').Attributes += 'ReadOnly'
```

## 3.5 Expresiones regulares y patrones

### 3.5.1 Patrones

Los caracteres comodín representan uno o varios caracteres. Puede utilizarlos para crear patrones de palabras en los comandos. Las expresiones comodín se utilizan con el operador **-like** o con cualquier parámetro que acepte comodines. Por ejemplo, para buscar todos los archivos en el directorio **C:\docs** con una extensión de nombre de archivo **.ppt**, escriba:

```
Get-ChildItem C:\docs\*.ppt
```

### CMD

Si recordamos en CMD también disponíamos de una serie de caracteres comodín. Estos eran:

- \* Coincide con cualquier secuencia de caracteres (0 o más)
- ? Coincide con un único carácter (o ninguno en casos excepcionales)

Tenemos unas series de reglas para estos caracteres comodín que son:

- \* Generalmente coincide con 0 o más caracteres cualesquiera, con una excepción (ver la siguiente regla). El comodín es libre de coincidir con tantos o tan pocos caracteres como sea necesario para que el resto de la máscara coincida.
- \*. Al final de la máscara coincide con 0 o más caracteres excepto punto. En realidad, la regla se aplica con cualquier número de caracteres punto y espacio entre el \* y el punto terminal.
- ? Coincide con 0 o un carácter, excepto con punto. La única vez que coincide con 0 caracteres es cuando coincide con el **final del nombre**, o **la posición antes de un punto**. El signo de interrogación también puede utilizarse más de una vez para que coincida con más de un carácter.

Por ejemplo, si tenemos el fichero **BAR.TXT**, los siguientes patrones coincidirían:

- ?AR.TXT

- BAR.\*
- ??R.TXT
- B?R.???
- BA?.TXT
- BA??.TXT

Sin embargo, los siguientes patrones fallarían:

- ??AR.TXT
- ?BAR.TXT
- B??AR.TXT

### PowerShell

A diferencia de la lógica difusa de MS-DOS y el shell CMD, los comodines de PowerShell son consistentes en su significado, por lo que \*.\* coincidirá con cualquier carácter seguido de un punto (.) seguido de cualquier carácter. En otras palabras, \*.\* sólo devolverá los archivos que tengan una extensión, no los directorios. Los comodines también funcionarán dentro de comillas simples y dobles, para evitar la expansión de comodines se utiliza el parámetro -LiteralPath cuando esté disponible.

En PowerShell tenemos además otra serie de caracteres comodín, en total tenemos los siguientes cuatro:

- El comodín \* coincidirá con cero o más caracteres
- El comodín ? coincidirá con un solo carácter

m-n Coincide con un rango de caracteres de m a n, por lo que [f-m]ake coincidirá con fake/jake/-make]

abc Coincide con un conjunto de caracteres a,b,c..., por lo que [fm]ake coincidirá con fake/make

Para realizar una comparación de cadenas, utilice los operadores -Like o -notlike, estos operadores de comparación admiten los 4 comodines anteriores. Por ejemplo:

```
PS C:\Users\ferqui\prueba> 'Esto es una prueba' -like '*prueba*'
True
```



Para expresiones más complejas, utilizamos el operador -match con una **expresión regular**.

### 3.5.2 Expresiones Regulares

Las expresiones regulares (regex) se utilizan para realizar búsquedas avanzadas en un texto. Las expresiones regulares suelen diferir ligeramente entre los diferentes lenguajes de programación, plataformas y herramientas. Dado que PowerShell está construido sobre .NET, PowerShell utiliza expresiones regulares de estilo .NET. A menudo hay varias maneras diferentes de lograr un objetivo cuando se utilizan expresiones regulares.

#### Carácter literal

El mejor lugar para empezar es con las expresiones más simples, es decir, las que no contienen caracteres especiales. Estas expresiones contienen lo que se conoce como caracteres literales. Un carácter literal puede ser cualquier cosa excepto [ $\backslash$ \$.?\*( ) .| Los caracteres especiales deben escaparse utilizando  $\backslash$  para evitar errores.

#### Cualquier carácter (.)

El punto coincide con cualquier carácter simple, excepto los caracteres de fin de línea. La siguiente sentencia devolverá True:

```
'abcdef' -match '.....'
```

Como la expresión anterior coincide con seis caracteres cualesquiera en una cadena, también devolverá True cuando se proporcione una cadena más larga. No hay límites implícitos en la longitud de una cadena, sólo en el número de caracteres coincidentes:

```
'abcdefghijkl' -match '.....'
```

### Repetición \*, + y {}

El carácter \* se puede utilizar para repetir el carácter anterior cero o más veces. Considere el siguiente ejemplo:

```
'aaabc' -match 'a*'# Devuelve true, coincide con 'aaa'
```

Sin embargo, cero o más significa que el carácter en cuestión no tiene que estar presente en absoluto:

```
'bcd' -match 'a*' # Devuelve true, coincide con vacío
```

Si un carácter debe estar presente en una cadena, + es más apropiado:

```
'aaabc' -match 'a+'# Devuelve true
'bcd' -match 'a+' # Devuelve false
```

La combinación de \* o + con . produce dos expresiones muy sencillas: .\* y +. Estas expresiones pueden utilizarse de la siguiente manera:

```
'Anything' -match '.*' # 0 or more. Returns true
'' -match '.*' # 0 or more. Returns true
'Anything' -match '.*+' # 1 or more. Returns true
```

También es posible indicar un número fijo de repeticiones o incluso un rango usando los caracteres {}. Tenemos tres opciones:

- {n}: Repite el carácter anterior exactamente n veces.
- {m,n}: Repite el carácter anterior entre m y n veces.
- {n,}: Repite el carácter anterior al menos n veces.

### Carácter \

En este contexto, \ es un carácter de escape, pero quizás sea más exacto decir que \ cambia el comportamiento del carácter que le sigue. Por ejemplo, encontrar una cadena que contenga el carácter normalmente reservado, \*, puede lograrse utilizando \, de la siguiente manera:

```
'1 * 3' -match '\*'
```

### Carácter opcional

El carácter de interrogación (?) puede utilizarse para que el carácter precedente sea opcional. Por ejemplo, puede ser necesario buscar la forma singular o plural de una determinada palabra:

```
'Me voy a comprar una casa' -match 'casas?'
```

### Otros caracteres

Las expresiones regulares admiten otros caracteres que no se imprimen por pantalla por ejemplo:

- Tabulación (\t)
- Nueva línea (\n)
- Retorno de carro (\r)

### Anchor

Los anchors no coincide con un carácter; en cambio, coincide con lo que viene antes (o después) de un carácter. Tenemos tres:

- Principio de una cadena: ^

- Final de una cadena: \$
- Límite de la palabra: \b

Los anchors son útiles cuando un carácter, cadena o palabra puede aparecer en otra parte de una cadena y la posición es crítica.

Por ejemplo, podría haber una necesidad de obtener valores de la variable de entorno PATH que comienza con una letra de unidad específica. En este caso se utilizaría por ejemplo la expresión regular '^C' para mostrar todos los valores del PATH que empiezan en el disco C.

### Caracteres ( )

Se utilizan para hacer coincidir un único carácter con un conjunto de posibles caracteres. Se denota mediante corchetes ([ ]). Dentro de los propios corchetes, tenemos otra serie de caracteres especiales. Estos son:

- -: Usado para definir un rango.
- \: Carácter de escape.
- ^: Niega los caracteres.

El rango de los valores dentro de los corchetes, puede ser de cualquier carácter ascii.

### Carácter |

El carácter | en una expresión regular se utiliza para combinar varias expresiones regulares posibles. Un ejemplo sencillo es hacer coincidir una lista de palabras:

```
'one', 'two', 'three' | Where-Object { $_ -match 'one|three' }
```

El carácter de alternancia tiene la menor precedencia; en la expresión anterior, cada valor se comprueba primero con la expresión a la izquierda de la tubería y luego con la expresión a la derecha de la tubería.

El objetivo de la siguiente expresión es extraer cadenas que sólo contengan las palabras uno o tres. Añadir el inicio y el final de las cadenas garantiza la existencia de un límite. Sin embargo, debido a que la izquierda y la derecha se tratan como expresiones separadas, el resultado podría no ser el esperado al utilizar la siguiente expresión:

```
PS> 'one', 'one hundred', 'three', 'eighty three' | Where-Object { $_ -match '^one|three$' }
one
one hundred
three
eighty three
```

Una posible solución sería agrupando con paréntesis

```
'one', 'one hundred', 'three', 'eighty three' | Where-Object { $_ -match '^(one|three)$' }
```

## 3.6 Pipeline

Todo lo que hacemos en PowerShell gira en torno a trabajar con objetos. Los objetos, en PowerShell, pueden tener propiedades o métodos (o ambos). Es difícil describir un objeto sin recurrir a esto; un objeto es una representación de una cosa o elemento de datos. Podríamos utilizar una analogía para intentar dar sentido al término.

Un libro es un objeto y tiene propiedades que describen características físicas, como el número de páginas, el peso o el tamaño. Tiene también metadatos (información sobre los datos) que describen el autor, la editorial, el índice de contenidos, etc.

El libro también puede tener métodos. Un método afecta al cambio de estado de un objeto. Por ejemplo, puede haber métodos para abrir o cerrar el libro o métodos para saltar a diferentes capítulos. Un método también puede convertir un objeto en un formato diferente. Por ejemplo,

podría haber un método para copiar una página, o incluso métodos destructivos como uno para dividir el libro.

PowerShell tiene una variedad de comandos que nos permiten trabajar con conjuntos (o colecciones) de objetos en una tubería. De forma que la salida de un comando sea la entrada de otro comando, sobrescribiendo de esta forma la salida y entrada estándar (stdin, stdout).

Los lenguajes como Batch (en Windows) o Bash (normalmente en Linux o Unix) utilizan un pipeline para pasar texto entre comandos. Es el siguiente comando el que debe averiguar qué significa el texto. PowerShell, por otro lado, envía objetos de un comando a otro.

El símbolo de tubería (|) se utiliza para enviar la salida estándar entre comandos.

En el siguiente ejemplo, la salida de `Get-Process` se envía al comando `Where-Object`, que aplica un filtro. El filtro restringe la lista de procesos a aquellos que están usando más de 50MB de memoria:

■ **Example 3.21** Ejemplo pipeline

```
Get-Process | Where-Object WorkingSet -gt 50MB
```

### 3.6.1 Miembros

Como hemos visto anteriormente, PowerShell es un híbrido entre varios paradigmas de programación, entre ellos la programación orientada a objetos. Cada uno de ellos tiene una serie de propiedades y métodos conocidas colectivamente de miembros. Estos miembros se utilizan para interactuar con un objeto. Algunos de los miembros más utilizados son: `NoteProperty`, `ScriptProperty`, `ScriptMethod` y `Event`.

#### Get-Member

El comando `Get-Member` se utiliza para ver los diferentes miembros de un objeto. Por ejemplo, se puede utilizar para listar todos los miembros de un objeto de proceso

■ **Example 3.22** Ejemplo Get-Member

```
Get-Process -Id $PID | Get-Member
```

#### Accediendo a propiedades

Se puede acceder a las propiedades de un objeto en PowerShell escribiendo el nombre de la propiedad después de un punto. Por ejemplo, se puede acceder a la propiedad `Name` del proceso actual de PowerShell utilizando el siguiente código:

■ **Example 3.23** Ejemplo Get-Member

```
$process = Get-Process -Id $PID  
$process.Name
```

PowerShell también nos permite acceder a estas propiedades encerrando un comando entre paréntesis:

■ **Example 3.24** Ejemplo Get-Member

```
(Get-Process -Id $PID).Name
```

Las propiedades de un objeto son objetos en sí mismos. Por ejemplo, la propiedad `StartTime` de un proceso es un objeto `DateTime`. Podemos acceder a la propiedad `DayOfWeek` utilizando el siguiente código:

■ **Example 3.25** Ejemplo Get-Member

```
$process = Get-Process -Id $PID  
$process.StartTime.DayOfWeek
```



La asignación de la variable nos la podemos saltar si se utilizan paréntesis:

#### ■ Example 3.26 Ejemplo Get-Member

```
(Get-Process -Id $PID).StartTime.DayOfWeek
```

Si el nombre de una propiedad tiene un espacio, se puede acceder a ella utilizando diferentes estilos de notación. Por ejemplo, se puede acceder a una propiedad llamada 'Algún nombre' citando el nombre o encerrando el nombre entre llaves:

#### ■ Example 3.27 Ejemplo Get-Member

```
$objeto."Algún Nombre"
$objeto.'Algún Nombre'
$objeto.{Algún Nombre}
```

### 3.6.2 Filtros

Enumerar, o listar, los objetos de una colección en PowerShell no necesita un comando especializado. Por ejemplo, si los resultados de Get-PSDrive se asignaron a una variable, enumerar el contenido de la variable es tan sencillo como escribir el nombre de la variable y pulsar Intro:

```
PS C:\Users\ferqui> Get-PSDrive
```

Name	Used (GB)	Free (GB)	Provider	Root	CurrentLocation
Alias			Alias		
C	781.17	99.72	FileSystem	C:\	Users\ferqui
Cert			Certificate	\	
D	1265.86	449.14	FileSystem	D:\	
E	402.14	5186.76	FileSystem	E:\	
G	786.16	94.73	FileSystem	G:\	

#### ForEach-Object

ForEach-Object nos permite realiza una acción sobre una colección de elementos como salida de un comando. Por ejemplo, el siguiente comando trabaja en cada uno de los resultados de Get-Process sucesivamente

```
Get-Process | ForEach-Object {
    Write-Host $_.Name -ForegroundColor Green
}
```

En el ejemplo anterior, se utiliza una variable especial, \$\_, para acceder a cada uno de los objetos de entrada. En el ejemplo anterior, se utiliza para acceder a cada uno de los objetos devueltos por el comando Get-Process.

ForEach-Object también se puede utilizar para obtener una sola propiedad, o ejecutar un solo método en cada uno de los objetos.

Por ejemplo, ForEach-Object puede utilizarse para devolver sólo la propiedad Path cuando se utiliza Get-Process:

```
Get-Process | ForEach-Object Path
```

O bien, se puede utilizar ForEach-Object para ejecutar el método ToString en un conjunto de fechas:

```
PS> (Get-Date '01/01/2019'), (Get-Date '01/01/2020') | ForEach-Object ToString('yyyy
20190101
20200101
```

También puede ser de utilidad realizar un código antes de ejecutar las iteraciones y otro código al final de las mismas. Por ejemplo para sumar el tamaño de los ficheros de un directorio. Para estos casos, el cmdlet tiene tres parámetros:

- Begin
- Process
- End

En este ejemplo se inicializa una variable *suma* a 0 en el bloque **Begin**, luego se suman todos los valores que recorre el cmdlet en el bloque **Process**. Finalmente imprime por pantalla el valor de la suma

```
1,1,2,3,2,4,8,5 | ForEach-Object -Begin {$suma=0} -Process {$suma
+= $_} -End {Write-Host $suma -ForegroundColor red -
BackgroundColor gray}
```



El cmdlet Write-Host nos permite imprimir por pantalla variables, tiene una serie de parámetros como -ForegroundColor y -BackgroundColor que permiten personalizar la salida.

### Where-Object

El filtrado de la salida de los comandos puede realizarse utilizando Where-Object. Por ejemplo, podríamos filtrar los procesos que se iniciaron después de las 17 horas de hoy:

```
Get-Process | Where-Object StartTime -gt (Get-Date 17:00:00)
```

En este ejemplo hemos seleccionado los objetos de Get-Process cuya fecha de inicio (StartTime) sea mayor que (-gt) las 5 de la tarde del día actual (Get-Date 17:00:00). Este cmdlet también acepta el uso del parámetro FilterScript para describir filtros más complejos donde se usan más de un término.

```
Get-Service | Where-Object { $_.StartType -eq 'Manual' -and $_.
Status -eq 'Running' }
```

Estas condiciones se evalúan de izquierda a derecha, por lo que se pueden utilizar por ejemplo para verificar que un fichero existe antes de realizar otra comprobación sobre él.

```
'C:', 'D:' | Where-Object {
(Test-Path "$_\temp\file.txt") -and
(Get-Item "$_\temp\file.txt").LastWriteTime -lt (Get-Date).
AddDays (-90)
}
```

### Select-Object

El comando Select-Object se utiliza con mayor frecuencia para limitar las propiedades devueltas por un comando. El comando es extremadamente versátil ya que permite hacer lo siguiente:

- Limitar las propiedades devueltas por un comando por nombre

```
Get-Process | Select-Object -Property Name, Id
```

- Limitar las propiedades devueltas por un comando utilizando comodines

```
Get-Process | Select-Object -Property Name, *Memory
```

- Enumerar todo menos algunas propiedades

```
Get-Process | Select-Object -Property * -Exclude *Memory*
```

- Obtener los primeros objetos

```
Get-ChildItem C:\ -Recurse | Select-Object -First 2
```

- Obtener los últimos objetos

```
Get-Process | Select-Object -Property Name, *Memory
```

- Saltar los primeros elementos

```
Get-ChildItem C:\ | Select-Object -Skip 4 -First 1
```

- Saltar los últimos elementos

```
Get-ChildItem C:\ | Select-Object -Skip 2 -Last 1
```

Select-Object creará nuevos objetos copiando los valores de las propiedades seleccionadas de los objetos de entrada. Por lo tanto, también podemos crear nuestras propias propiedades en los nuevos objetos, siguiendo la siguiente sintaxis:

```
Select-Object -Property ..., { Código }
```

Donde código indica que podemos escribir nuestro propio código al igual que hacíamos con el cmdlet ForEach-Object. En el siguiente ejemplo se muestra el tamaño de los ficheros en KB.

```
Get-ChildItem -File | Select-Object -Property Name, {$_ .Length / 1024}
```

En algunos casos, puede ser útil que Select-Object nos devuelva un valor en vez de un Objeto con ese valor, para eso tenemos el parámetro -ExpandProperty. Por ejemplo:

```
PS>Get-ChildItem -File | Select-Object Name
Name
----
fichero.txt
```

En este código, select-object nos devuelve un objeto nuevo con un parámetro Name. Si nosotros queremos que directamente nos devuelva el valor del parámetro, podemos realizar lo siguiente:

```
PS>Get-ChildItem -File | Select-Object -ExpandProperty Name
fichero.txt
```

### Sort-Object

El comando Ordenar-Objeto permite ordenar los objetos según una o varias propiedades. Por defecto, Sort-Object ordenará los números en orden ascendente:

```
PS> 5, 4, 3, 2, 1 | Sort-Object
1
2
3
4
5
```

Cuando se trata de objetos complejos, se puede utilizar Sort-Object para ordenar en base a una propiedad con nombre. Por ejemplo, los procesos pueden ser ordenados en base a la propiedad Id:

```
Get-Process | Sort-Object -Property Id
```

Los objetos pueden ser ordenados en base a múltiples propiedades; por ejemplo, una lista de archivos puede ser ordenada en base a LastWriteTime y luego en base a Name:

```
Get-ChildItem C:\Windows\System32 | Sort-Object LastWriteTime, Name
```

En el ejemplo anterior, los elementos se ordenan primero en función de LastWriteTime. Los elementos que tienen el mismo valor para LastWriteTime se ordenan entonces en función de Name.

## Measure-Object

Measure-Object admite una serie de operaciones matemáticas sencillas, como contar el número de objetos, calcular una media, calcular una suma, etc. También permite contar caracteres, palabras o líneas en campos de texto. Cuando se utiliza sin ningún parámetro, Measure-Object devolverá un valor para Count, que es el número de elementos pasados mediante la tubería, por ejemplo:

```
PS> 1, 5, 9, 79 | Measure-Object
```

```
Count      : 4
Average    :
Sum        :
Maximum    :
Minimum    :
Property   :
```

Cada una de las propiedades restantes está vacía, a menos que se solicite mediante sus respectivos parámetros. El valor de la propiedad se rellena cuando se pide a Measure-Object que trabaje contra una propiedad concreta (en lugar de un conjunto de números), por ejemplo:

```
PS> Get-Process | Measure-Object WorkingSet -Average
```

```
Count      : 135
Average    : 39449395.2
Sum        :
Maximum    :
Minimum    :
Property   : WorkingSet
```

Cuando se trabaja con texto, Measure-Object puede contar caracteres, palabras o líneas. Por ejemplo, puede utilizarse para contar el número de líneas, palabras y caracteres de un archivo de texto:

```
PS> Get-Content C:\Windows\WindowsUpdate.log | Measure-Object -Line -Word -Character
```

Lines	Words	Characters	Property
3	32	268	