Prácticas de Infraestructura Informática

Bloque 3: Infraestructuras de scritpting y de gestión en plataformas Windows

Sesión 1 - Fundamentos de PowerShell (I)

Objetivos

Conocer los fundamentos de PowerShell, el lenguaje de scripting de Microsoft que permite automatizar las tareas administrativas en los sistemas Windows.

Desarrollo

- ### Introducción al Scripting

Lenguajes de scripting

Son lenguajes orientados a crear programas que no se compilan, sino que son interpretados por un entorno de ejecución. Los programas creados usando estos lenguajes de denominan scripts. Estos se almacenan en ficheros de texto, no existiendo versiones compiladas de los mismos.

Comparación Programación tradicional / scripting

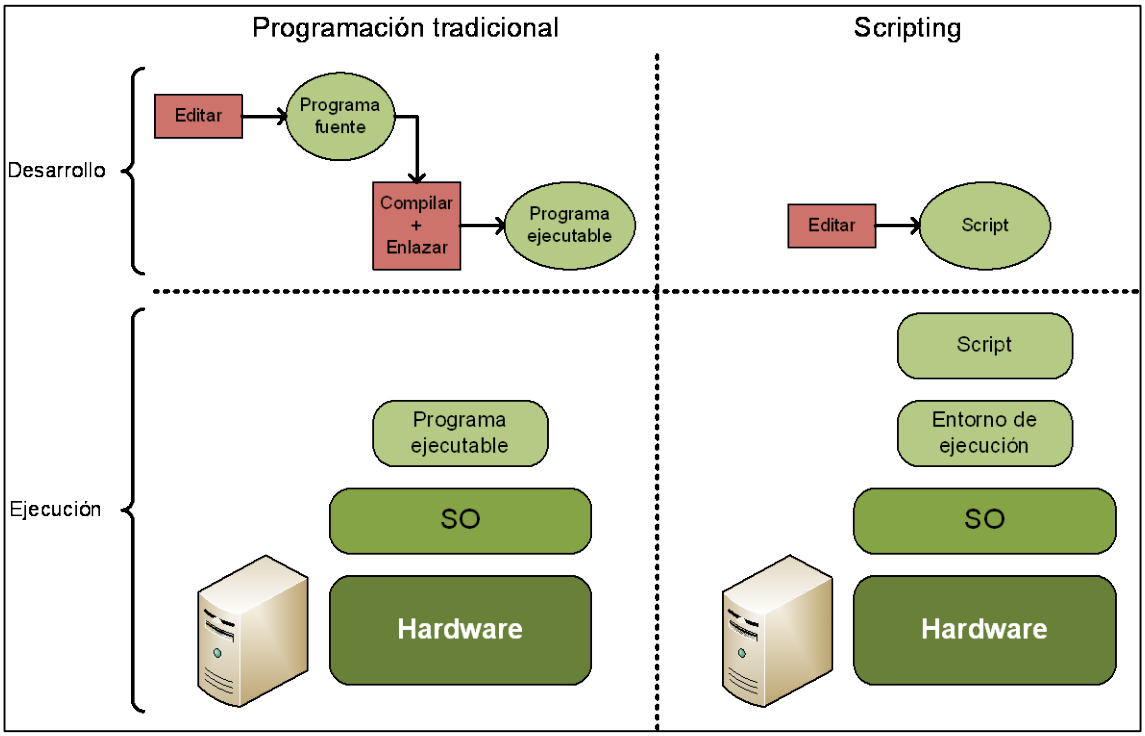

Figura 1. Programación tradicional vs scripting

Tipos de lenguajes de scripting

Lenguajes específicos de aplicación

Son lenguajes que proporcionan funcionalidades de programación para una aplicación específica. Un ejemplo típico es *Matlab*. Esta aplicación cuenta con su entorno de programación para construir simulaciones. El entorno de ejecución de los scripts de *Matlab* es la propia aplicación *Matlab*. Otro ejemplo es *Javascript*. Este lenguaje se utiliza en la programación de páginas web, con objeto de

mejorar muy significativamente las capacidades de interacción con el usuario en dichas páginas. La aplicación y, consecuentemente, el entorno en el que se ejecutan estos scripts es el navegador web.

Lenguajes de propósito general

Son lenguajes, que al igual que los leguajes de programación tradicionales, están orientados a resolver cualquier tipo de necesidad de procesamiento de información. Ejemplos de estos lenguajes son *Python*, *Perl* y *Tcl*. Estos lenguajes de scripting de propósito general son también llamados muy frecuentemente leguajes dinámicos. Cada uno de estos lenguajes cuenta con su correspondiente entorno para la ejecución de los scripts.

Lenguajes de scripting para entornos de comandos

Los entornos de comandos son programas interactivos que permiten al usuario ejecutar comandos para la gestión del sistema. Estos entornos proporcionan habitualmente lenguajes de scripting para ejecutar conjuntos de comandos de forma programada.

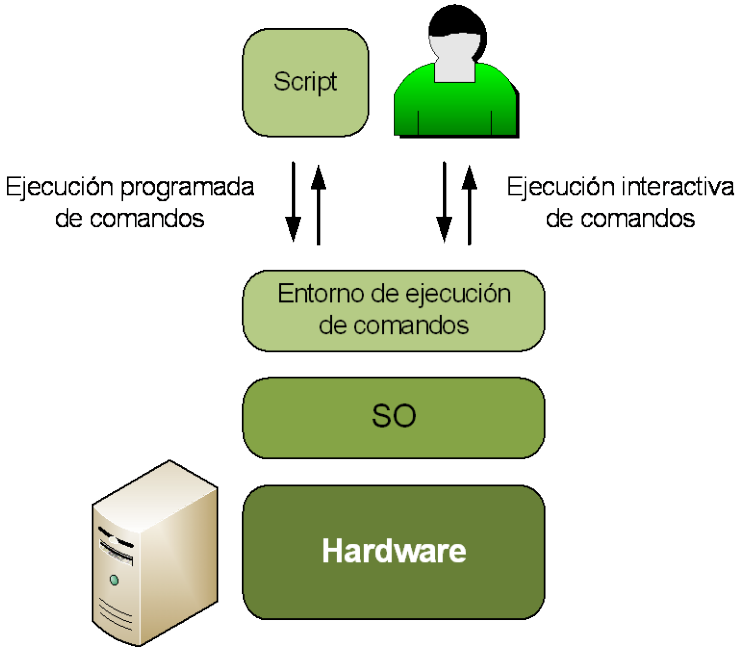

Figura 2. Scripting en los entornos de ejecución de comandos

En la plataforma Windows hay dos entornos de comandos: la consola CMD y el entorno PowerShell.

Ventajas de Powershell frente a CMD

- Nomenclatura de comandos más consistente
- Mayor número de comandos y posibilidad de creación de comandos nuevos
- Los comandos trabajan con objetos en vez de cadenas de texto
- Lenguaje de scripting mucho más potente

Razones para el uso del scripting en la administración de sistemas

La razón fundamental es la automatización de tareas repetitivas. Debe tenerse en cuenta que en las grandes organizaciones el número de equipos puede ser muy elevado. Una tarea de configuración que requiera 10 o 15 pulsaciones de ratón no presenta ningún problema en uno o en unos pocos equipos. Sin embargo, si esa misma tarea debe realizarse en cientos de equipos la situación se hace mucho más complicada. Los scripts automatizan este tipo de tareas, bien sea mediante la distribución de un script a todos los equipos a configurar, o bien utilizando scripts que llevan a cabo las operaciones a realizar mediante conexiones remotas establecidas con los equipos a configurar.

• Actuaciones previas

Las pruebas a realizar en esta práctica se llevarán a cabo en un equipo cliente. En concreto, utilizarás la máquina PLX-C-51, usando el usuario local *Alumno*.

- Arranca la máquina PLX-C-51, e inicia sesión con el usuario local *Alumno*.

  Recuerda que, en la configuración habitual de la plataforma de prácticas, *Red virtual interna* está desconectada de Internet para evitar que los clientes Win11 cojan actualizaciones. Sin embargo, en esta práctica, es necesario conectar PLX-C-51 a Internet, ya que se requiere descargar los ficheros de ayuda de PowerShell desde un servicio de Microsoft.
- Para conectar *Red virtual interna* a Internet, en el sistema anfitrión, *Herramientas administrativas -> Enrutamiento y acceso remoto ->* botón derecho sobre *Servidor (local) -> Todas las tareas -> Iniciar*.
- En PLX-C-51, comprueba que tienes acceso a la red externa. Para ello, haz ping al DNS de Uniovi (156.35.14.2).
- En PLX-C-51, abre el navegador y trata de acceder a alguna página de Internet. Comprobarás que no es posible. ¿Qué está ocurriendo? El sistema tiene salida a la red externa, pero no se pude navegar. La explicación está en el DNS. Recuerda que el DNS de las MV conectadas a *Red virtual interna* es el controlador de dominio, que, en este momento, está apagado.
- Arranca el controlador de dominio. No es necesario que inicies sesión, pero sí debes esperar a que se muestre la pantalla de bloqueo para asegurar que todos los servicios del controlador, incluido el DNS, estén operativos.
- En PLX-C-51, comprueba que puedes navegar.

Descarga de los ficheros de ayuda de PowerShell

- Más adelante, en el punto *Introducción a PowerShell*, se describen las herramientas para el manejo de *PowerShell*, entre las que se encuentra la consola de *PowerShell*. En este punto, simplemente vas a abrir esta consola, para proceder a descargar los ficheros de ayuda.
- Utilizando la herramienta *Buscar*, busca *Windows PowerShell*. Entonces se muestra el enlace a *Windows PowerShell*. Botón derecho sobre *Windows PowerShell -> Ejecutar como administrador*. En la venta *Control de cuentas de usuario*, pulsa Sí. Entonces, se abre la consola de *PowerShell*.
- Ejecuta el comando *Update-Help*. El sistema comienza el proceso de descarga de los ficheros de ayuda de PowerShell. Es posible que al final del proceso se muestre algún error. Puede deberse a algún módulo de ayuda que no se descarga correctamente. Ignora el error, ya que no es relevante para la información de ayuda requerida en estas prácticas.
- Cierra la consola de PowerShell.

Desconexión de "Red virtual interna" de la red externa

Una vez descargados los ficheros de ayuda, no se requiere que las MV sigan conectadas a la red externa, por lo que se procede a su desconexión, siempre con el objetivo de evitar actualizaciones.

- En el sistema anfitrión, abre *Enrutamiento y acceso remoto ->* botón derecho sobre *Servidor (local) -> Todas las tareas -> Detener*.
- Apaga el controlador de dominio, ya que no se necesita a partir de este momento.

• Introducción a PowerShell

Herramientas de PowerShell

Para trabajar con PowerShell, la plataforma Windows proporciona cuatro herramientas. Un enlace a cada una de ellas está disponible en *Herramientas de Windows*.

- Herramienta *Buscar -> Herramientas de Windows*. Entonces, observa los enlaces proporcionados a las 4 herramientas disponibles, que son los siguientes:

- Windows PowerShell
- Windows PowerShell (x86)
- Windows PowerShell ISE
- Windows PowerShell ISE (x86)

Windows PowerShell es la consola estándar de PowerShell, implementada mediante software de 64 bits, que es el estándar en las plataformas Windows modernas.

Windows PowerShell ISE es un entorno gráfico de programación en PowerShell. Es de gran utilidad cuando se programan scripts.

Windows PowerShell (x86) y Windows PowerShell ISE (x86) son las versiones de 32 bits de las herramientas de PowerShell. Solo son necesarias para gestionar mediante PowerShell plataformas Windows de 32 bits de forma remota. Estas herramientas no son necesarias en el contexto de estas prácticas, ya que no se utilizan plataformas Windows de 32 bits.

Comandos de PowerShell: *CmdLets*

Los comandos de PowerShell se denominan CmdLets (pronunciado "comandlets") ¿Cómo se forman los nombres de estos comandos? Los CmdLets utilizan un esquema de nombrado *Verbo-Nombre*. El verbo expresa una acción y el nombre un objeto sobre el que se aplica la acción. Este esquema de nombrado se muestra significativamente eficaz con relación al esfuerzo requerido por los usuarios para memorizar los comandos. Así, por ejemplo, en el caso de los comandos Get-Process, Stop-Process, Get-Service y Stop-Service, dos verbos (Get y Stop) aplicados sobre dos tipos de objetos (Process y Service) dan lugar a cuato CmdLets diferentes.

Primeros ejemplos de CmdLets y ejecución de comandos con "derechos elevados"

- A modo de ejemplo, probarás tres CmdLets que trabajan sobre servicios de Windows.
- Abre la consola *Servicios* (accesible en *Herramientas de Windows*), para ver los servicios disponibles.
- Selecciona el servicio *Cola de impresión*. Según se vio en prácticas anteriores, este servicio es el encargado de enviar los trabajos de impresión generados en los diferentes programas a las impresoras (físicas o virtuales) disponibles en el sistema.
- Abre el servicio para ver su nombre. ¿Cómo se llama? *Spooler* ¿En qué estado se encuentra este servicio? *En ejecución*.
- Utilizarás ahora la consola de PowerShell para obtener información del servicio, pararlo y volver a ponerlo *En ejecución*.
- Abre la consola de *PowerShell*. Para ello, herramienta *Buscar -> Windows PowerShell*.
- En la consola de PowerShell, para obtener información del servicio, ejecuta el comando siguiente:

  ```
  > Get-Service spooler
  ```

  El comando muestra el estado del servicio (*running*), su nombre (*spooler*) y su nombre para mostrar *(Cola de impresión).*

- Intenta parar el servicio ejecutando el siguiente comando:

  ```
  > Stop-Service spooler
  ```

- Se produce el error siguiente: stop-service : No se puede detener el servicio 'Cola de impresión (spooler)' debido al error siguiente: No se puede abrir el servicio spooler en el equipo '.'. En muchas ocasiones, la información de error proporcionada por PowerShell no es muy precisa y hay que interpretar lo que puede estar ocurriendo. En este caso, se trata de un problema de privilegios. La idea es que para parar un servicio es necesario disponer de privilegios de *Administrador*. A continuación, se analiza este asunto.
- El usuario que ha iniciado sesión es *Alumno*. Abre *Administración de equipos -> Usuarios locales y grupos -> Usuarios*. Selecciona el usuario *Alumno* y en la ficha *Miembro de*,

comprueba que pertenece al grupo *Administradores*. Entonces, ¿por qué no tiene privilegios de administrador? La respuesta es que salvo el usuario *Administrador* (que en los clientes utilizados en estas prácticas está desactivado, y que es equivalente al usuario *root* de Linux), el resto de usuarios de la máquina, pertenezcan o no al grupo *Administradores*, inician sesión con derechos de *usuario estándar*, y no de *Administrador*. Esto es una medida de seguridad. Entonces, cuando el usuario que tiene iniciada sesión trata de hacer una operación que requiere derechos de *Administrador*, la operación se deniega. Sin embargo, si el usuario que tiene iniciada sesión pertenece al grupo *Administradores* (como es el caso del usuario *Alumno*), éste tiene la posibilidad de ejecutar la operación con "derechos elevados", es decir, como si fuera el *Administrador* (*root*) de la máquina, evitando así que se le deniegue realizar la operación. En el caso de que se trate de comandos ejecutados desde una consola (ya sea PowerShell o CMD), la solución para ejecutar los comandos con privilegios de *Administrador* es abrir la consola con "derechos elevados".

- Para abrir la consola de PowerShell con "derechos elevados", herramienta *Buscar -> Windows PowerShell ->* botón derecho sobre *Windows PowerShell -> Ejecutar como administrador*. Se abre la ventana *Control de cuentas de usuario*, donde debes indicar Sí. Entonces se abre la consola con los mismos derechos que si la hubiera abierto el usuario *Administrador* (*root*) de la máquina. De esta forma, no habrá ninguna limitación en los comandos que se pueden ejecutar. Observa que la consola se posiciona en la carpeta C:\Windows\system32, en vez de en la carpeta del usuario.
- De nuevo, ejecuta el comando siguiente:

  ```
  > Stop-Service spooler
  ```
- En esta ocasión, el comando debe ejecutarse sin error.
- Vuelve a ejecutar el comando *Get-Service spooler* para comprobar que el servicio se ha parado.
- Vuelve a la consola *Servicios* y comprueba el nuevo estado del servicio (posiblemente habrá que actualizar la visualización de la consola).
- Arranca de nuevo el servicio.

  ```
  > Start-Service spooler
  ```
- Vuelve a ejecutar el comando *Get-Service spooler* para comprobar que el servicio se ha iniciado.
- Vuelve a la consola *Servicios* y comprueba el nuevo estado del servicio.

Pedir ayuda a un CmdLet

Para esto se pasa el parámetro '?' al CmdLet. Los parámetros se pasan siempre precedidos del carácter '-'.

```
> Get-Service -?
```

- Una información muy importante proporcionada por la ayuda es la sintaxis del CmdLet. Se trata del nombre del comando junto con sus parámetros. Observa que el nombre de los parámetros se precede siempre de un '-'. Los parámetros se nombran utilizando nombres largos para que tengan un mayor significado.

Parámetro determinante de los objetos sobre los que actúa un CmdLet

En muchos CmdLets que trabajan sobre objetos del sistema, hay un parámetro que adquiere una importancia especial, por ser el elemento que determina los objetos sobre los que opera el comando. Habitualmente, pueden existir varias alternativas para este parámetro, siendo dichas alternativas excluyentes.

Por ejemplo, en el caso de los CmdLets que operan sobre el objeto servicio, hay tres parámetros alternativos para especificar los servicios sobre los que opera el comando. Estos parámetros son -DisplayName, -InputObject y -Name. Estos parámetros son excluyentes, ya que solo puede utilizarse uno de ellos para indicar los servicios sobre los que opera el comando.

En la ayuda de los CmdLets, en el apartado SINTAXIS, se muestran tantas alternativas de la sintaxis del comando como parámetros diferentes existan para indicar los objetos sobre los que opera el comando. Vas analizar este asunto en la ayuda de *Get-Service*.

- Por tu atención en el apartado SINTAXIS de la ayuda de *Get-Service*. Observa que se muestran tres alternativas para la sintaxis de este comando. Lo que diferencia a las distintas alternativas es el parámetro que determina cómo obtener los servicios sobre los que opera el comando. A continuación, se van a analizar las tres alternativas.
- En la primea alternativa se utiliza el parámetro -DisplayName. Este parámetro es lo que se conoce como el "Nombre para mostrar" del servicio.
- Abre la consola *Servicios*. Busca el servicio *Cola de impresión*. Ábrelo. En la parte superior, puedes observar que el "Nombre para mostrar" de este servicio es "Cola de impresión".
- Entonces, para indicar al comando Get-Service que se desea el servicio "Cola de impresión", se puede utilizar el parámetro -DisplayName de la siguiente forma:

  ```
  > Get-Service –DisplayName "Cola de impresión"
  ```
- Ejecuta el comando anterior, observando que su comportamiento es el esperado. (NOTA: cuando el nombre para mostrar del servicio contiene espacios, las comillas son imprescindibles).
- Vuelve a solicitar la ayuda del comando *Ger-Service*.
- En esta ocasión, por tu atención en la tercera alternativa de la sintaxis de este comando. En esta alternativa se utiliza el parámetro -Name. Este parámetro hace referencia al "Nombre del servicio".
- En la consola *Servicios*, comprueba que en el caso del servicio *Cola de impresión* el nombre del servicio es *Spooler*.
- Entonces, para indicar al comando *Get-Service* que se desea actuar sobre el servicio "Cola de impresión", se puede utilizar el parámetro -Name de la siguiente forma:

  ```
  > Get-Service –Name spooler
  ```
- Ejecuta el comando anterior, observando que su comportamiento es el esperado.
- En la ayuda, observa que el parámetro [-Name] se muestra entre corchetes. Esto significa que el nombre del parámetro es opcional, o sea, que puede indicarse o no en el comando. Para comprobar esto, ejecuta el comando según se indica a continuación:

  ```
  > Get-Service spooler
  ```
- Finalmente, solicita de nuevo la ayuda de *Get-Service*. Observa que hay una alternativa más para indicar el servicio o servicios sobre los que actúa el comando. Se trata del parámetro -InputObject. Cuando se utiliza esta alternativa, hay que pasar al parámetro un objeto de tipo servicio. Esto es menos común y no se verá en estas prácticas.

Carácter comodín ('*')

El carácter '*' es un sustituto de cualquier grupo de caracteres.

- Así, por ejemplo, para obtener todos los servicios cuyo "Nombre para mostrar" *(DisplayName)* comience por "Servicio", ejecuta el comando siguiente:

  ```
  > Get-Service –DisplayName Servicio*
  ```

  (1) EJERCICIO. ¿Qué comando debes ejecutar para listar todos los servicios que incluyan la palabra "redes" en su *DisplayName*? Indícalo a continuación:

- Ejecuta el comando anterior para ver la salida que genera.

Obtener información sobre los comandos disponibles en PowerShell

Para ello se utiliza el comando *Get-Command.*

- Solicita al comando *Get-Command* su ayuda:

  ```
  > Get-Command -?
  ```

- Observa el parámetro -CommandType. Este parámetro indica el tipo de comando. En PowerShell hay varios tipos de comandos, tal y como puedes observar en la ayuda (*Alias*, *Function*, *Filter*, *Cmdlet*, etc.). De momento, solo estás trabajando con comandos del tipo *CmdLet*.

  (2) EJERCICIO. Ejecuta un comando que liste todos los comandos de tipo CmdLet. Indica el comando a continuación.

  (3) EJERCICIO. Ejecuta un comando que liste todos los comandos de tipo CmdLet que contengan la palabra *Process*. Indica el comando a continuación.

  Debe mostrar 11 comandos, entre los que se encuentran Debug-Process, Get-Process, Start-Process, Stop- Process y Wait-Process.

- Observa los parámetros -Noun y -Verb. -Noun permite obtener todos los comandos que se aplican a un determinado objeto. De la misma forma, -Verb sirve para ver todos los comandos que utilizan un determinado verbo.

  (4) EJERCICIO. Utiliza el parámetro -Verb para listar los comandos que utilizan el verbo Get. Indica el comando a continuación.

Obtener ayuda detallada

Para esto se utiliza el comando Get-Help

- Obtén ayuda sobre el propio comando:

  ```
  > Get-Help -?
  ```

  Observa que se presentan varias alternativas de sintaxis para el comando *Get-Help*. Éstas vienen determinadas por un parámetro, que en caso de utilizarse indica el grado de detalle deseado de la ayuda. Así, puedes ver los parámetros -Detailed (para obtener ayuda detallada), -Examples (para ver ejemplos de uso del comando), y -Full (para ver la ayuda completa). Por otra parte, el parámetro -Name, que es opcional, se utiliza para indicar el nombre del comando del que se desea obtener ayuda.

  (5) EJERCICIO. Utiliza el comando *Get-Help* para analizar el funcionamiento del comando *Write-Host*. Después, ejecuta este comando de modo que escriba en la consola la cadena de caracteres `"Prácticas de PowerShell"`, utilizando fuente de color rojo y fondo de color amarillo. Indica el comando ejecutado a continuación.

  (6) EJERCICIO. Utilizando el parámetro *–Separator*, escribe un comando *Write-Host* que imprima en la consola las cadenas de caracteres "Pedro", "María" y "Juan" separadas mediante el carácter '/'. Las cadenas deben pasarse al comando separadas por comas.

Aliasing

Se trata de una técnica que permite generar nombres alternativos para comandos o funciones. Los elementos generados mediante esta técnica se denominan *Alias*.
Hay dos razones básicas para el *aliasing*:

- Asignar a comandos o funciones de PowerShell nombre de comandos provenientes de otros entornos, como el CMD o los entornos de UNIX.
- Generar nombres más breves para los comandos de PowerShell (aunque mucho menos legibles).

Los Alias pueden crearse a voluntad. No obstante, ya hay muchos creados por defecto.

(7) EJERCICIO. Utilizando previamente la ayuda de Get-Command, ejecuta este comando de modo que muestre una lista de todos los Alias disponibles en la consola de PowerShell. Indica a continuación el comando que has ejecutado.

- Un ejemplo de alias, definido para los usuarios de la consola CMD, es *dir*. En el listado obtenido mediante el comando anterior observarás que este alias se corresponde con el CmdLet *Get-ChildItem*.
- Ejecuta ambos comandos en la consola de PowerShell y comprueba que el resultado que se obtiene es el mismo.

(8) EJERCICIO. Se desea utilizar el comando *Get-Alias* para conocer el alias correspondiente al comando *Get-Command*, si existiera. Utiliza la ayuda para ver cómo debes ejecutar el comando *Get-Alias*. Escribe a continuación el comando requerido.

El resultado obtenido es "gcm". Este es un ejemplo de alias definido con el objetivo de la brevedad. La 'g' es una abreviatura de Get, y 'cm' de Command.

(9) EJERCICIO. Ejecuta un comando *gcm* para obtener un listado de todos los comandos que actúen sobre el nombre *Host*. Indica a continuación el comando que has ejecutado.

Variables

Una variable es una unidad de memoria en la que se almacenan valores. En PowerShell las variables se representan mediante una cadena de texto precedida del símbolo $, como por ejemplo $a, o $mi_variable.

En PowerShell, una forma habitual de crear variables es mediante expresiones de asignación, las cuales tienen la siguiente estructura:

```
<VARIABLE><Operador de asignación><VALOR>
```

Hay varios tipos de operadores de asignación, siendo el más común el '='. Este operador asigna el valor a la variable.

- Ejecutar la siguiente expresión de asignación:

  ```
  > $a = "Hola"
  ```

  La expresión anterior crea la variable $a y le asigna la cadena de caracteres "Hola". Una vez creada una variable, ésta puede utilizarse en cualquier comando o expresión.

- Ejecuta el comando siguiente para escribir en la consola el contenido de $a:

  ```
  > Write-Host $a
  ```

  Salvo que se indique expresamente, las variables no son tipificadas. Esto quiere decir que pueden contener cualquier tipo de dato.

- Ejecuta los comandos siguientes:

  ```
  > $a = 5
  > Write-Host $a
  ```

  La variable $a, que anteriormente contenía la cadena "Hola", ha pasado a contener el entero 5.

Arrays

Los *arrays* son variables que contienen múltiples elementos.

- Ejecuta el siguiente comando:

```
> $b = 5, 6, 7
```

Esto crea la variable "b". Esta variable es de tipo *array* porque se le asignan varios elementos: tres enteros en este ejemplo.

Para acceder a cada elemento simple del *array* se indica el índice del elemento entre corchetes. El índice del primer elemento es el cero.

- Para imprimir en la consola el elemento de índice 1 del *array* b, ejecuta el siguiente comando:

  ```
  > Write-Host $b[1]
  ```

  Los *arrays* pueden contener colecciones de objetos de diferentes tipos.
- Ejecuta el siguiente comando:

  ```
  > $c = 5, 7.5, "nueve"
  ```

  Este comando crea un *array* que contiene tres elementos de tipos diferentes: un entero, un doble y una cadena.
- ¿Qué salida provocará en la consola el comando **`Write-Host $c[2][2]`**?
- Piensa la respuesta y después ejecuta el comando para comprobarla.

(10) EJERCICIO. Ejecuta una línea de comandos que solicite al usuario introducir una cadena de caracteres por consola y que la imprima en color rojo. La línea de comandos debe generar el mensaje `Introduzca una cadena: ` , con objeto de guiar al usuario en la acción a realizar.

- Para hacer este ejercicio necesitas utilizar un comando aún no visto. Se trata de *Read-Host*. Consulta la ayuda para aprender a utilizarlo.
- Este ejercicio requiere ejecutar dos comandos. Para encadenar comandos en una misma línea se utiliza el carácter ';'.

• Objetos en PowerShell

Modelo de funcionamiento orientado a objetos

La mayoría de los entornos de scripting orientados a la gestión de sistemas están diseñado para manejar cadenas de caracteres. Reciben cadenas de entrada, las interpretan siguiendo la sintaxis del lenguaje que corresponda y generan cadenas de salida.

Respecto a esto, PowerShell introduce una gran novedad, ya que se trata de un entorno orientado a objetos. Así, los comandos de PowerShell reciben objetos de entrada, los procesan y generan objetos de salida. Los objetos manejados son muy variados, desde aquellos que contienen datos simples (cadenas, enteros, reales de precisión doble, etc.), hasta los manejados por el sistema operativo, como los objetos de tipo proceso y servicio, entre otros. Este esquema de funcionamiento, con orientación a objetos, proporciona a PowerShell una gran potencia y flexibilidad para interaccionar con el sistema operativo.

El comando Get-Member

Este comando proporciona las propiedades y métodos (miembros) del objeto que recibe como parámetro o a través de una canalización (el concepto de canalización se verá más adelante).

- Obtener ayuda del comando Get-Member:

  ```
  > Get-Member -?
  ```
- Observa que el parámetro mediante el que se le pasa el objeto de entrada se denomina *InputObject*.
- Crea una variable con un objeto de tipo cadena de caracteres:

  ```
  > $a="Hola"
  ```
- Obtén las propiedades y métodos de $a:

```
> Get-Member –InputObject $a
```

En el listado proporcionado por el comando anterior, se muestra en primer lugar el tipo de objeto recibido por Get-Member, en este caso *System.String*. La nomenclatura de objetos utilizada por PowerShell es idéntica a la utilizada por el sistema operativo. A continuación del tipo, se muestran las propiedades y métodos del objeto.

Propiedades y métodos de los objetos

Debe recordarse que los objetos están formados por propiedades y métodos. Las propiedades representan características del objeto y se almacenan como valores de un determinado tipo. Los métodos representan operaciones a realizar con el objeto.

Una vez que un objeto es almacenado en una variable, se puede acceder a sus propiedades y ejecutar sus métodos. Para ello se utiliza el operador punto (.).

- Por ejemplo, para imprimir en la consola la longitud de la cadena almacenada en $a, se utiliza la propiedad *Length* del objeto tipo *System.String*. Para ello, ejecuta el comando siguiente:

  ```
  > Write-Host $a.Length
  ```

- Cuando se trata de enviar a la consola el contenido de una variable, no es necesario utilizar el comando Write-Host. Se puede introducir directamente la variable en la línea de comandos y pulsar <ENTER>.

  ```
  > $a.Length
  ```

  De aquí en adelante, se mostrarán las variables de esta forma.

- Para convertir la cadena a mayúsculas, se utiliza el método *ToUpper()*, que se ejecuta de la siguiente forma:

  ```
  > $a.ToUpper()
  ```

  El resultado se muestra en la siguiente línea.

  (11) EJERCICIO. Ejecuta una línea de comandos que solicite al usuario introducir una cadena de caracteres por consola y que, a continuación, reemplace todos los caracteres ‘a’ por el carácter ‘%’. Para ello, será necesario definir una variable de tipo cadena, utilizar un método del objeto tipo *System.String* y encadenar dos comandos mediante el operador ‘;’. Escribe la línea de comandos ejecutada a continuación.

  (12) EJERCICIO. Define una variable que se llame $Pi e inicialízala con el valor 3.1416. Después, utilizando el comando *Get-Member*, determina el tipo de objeto asignado a esta variable. Indica el tipo a continuación.

Objetos del Sistema Operativo

Un gran número de comandos de PowerShell devuelven objetos del sistema operativo. Por ejemplo, el comando *Get-Process* devuelve objetos de tipo proceso (o, más precisamente, de tipo *System.Diagnostics.Process* en la nomenclatura utilizada por el sistema operativo). Una vez que se dispone de un objeto de este tipo referenciado por una variable, se puede utilizar las propiedades y métodos del objeto para gestionarlo.

- Abre la herramienta *Administrador de tareas*. Si es necesario, utiliza el botón *Más detalles* para presentar la información del sistema de forma completa.
- Selecciona la pestaña *Detalles*. Esta pestaña muestra los procesos en ejecución en el sistema. Si fuera necesario, pulsa sobre el campo *Nombre* de la cabecera de la tabla de modo que todos los procesos queden ordenados por nombre de forma descendente. A la derecha de la columna *Nombre*, observa la columna *PID*, en la que se indica el identificador numérico (PID) de cada proceso.

- Abre el *Bloc de notas*. Esta herramienta se implementa mediante el proceso *Notepad.exe*. Comprueba que el *Administrador de tareas* muestra este proceso.
- En la consola de PowerShell, ejecuta el siguiente comando:

  ```
  > $a = Get-Process notepad
  ```
- Observa ahora la estructura del objeto almacenado en $a. Para ello, ejecuta el comando siguiente:

  ```
  > Get-Member -InputObject $a
  ```

  (13) EJERCICIO. Tras analizar la información proporcionada por *Get-Member*, ejecuta un comando de PowerShell que muestre el PID del proceso *Notepad*. Comprueba que el valor proporcionado por PowerShell coincide con el indicado por el *Administrador de tareas*. Escribe el comando ejecutado a continuación.

  (14) EJERCICIO. Ejecuta un comando de PowerShell que muestre la fecha y hora en la que ha comenzado a ejecutarse el proceso *Notepad*. Comprueba que el valor proporcionado por PowerShell es coherente con el momento aproximado de arranque del proceso. Escribe el comando a continuación.

  (15) EJERCICIO. Ejecuta el método *Kill()* para eliminar el proceso de ejecución. Comprueba que *Notepad* se cierra y que el proceso es eliminado de la lista de procesos mostrados por el *Administrador de tareas*. Escribe el comando ejecutado a continuación.

• Canalizaciones *(pipelining)*

Se trata de una técnica que permite encadenar comandos, de modo que los objetos de salida que genera un comando son pasados como objetos de entada al siguiente comando en la cadena. El encadenamiento se realiza mediante el operador "|". Esta técnica permite llevar a cabo operaciones muy potentes con unas líneas de comandos muy compactas. A continuación, se verán algunas áreas funcionales en las que la canalización es de gran utilidad.

Formateo de salida

Para formatear la salida se utilizan los comandos *Format-Wide*, *Format-List* y *Format-Table*. Estos comandos reciben un objeto o un *array* de objetos y los presentan en la consola de formas diferentes. A continuación, se van a utilizar estos comandos para formatear la salida generada por *Get-Process*.

- Vas a observar la salida generada por *Get-Process* sin aplicar ningún formateo adicional. Para ello, ejecuta el domando siguiente:

  ```
  > Get-Process
  ```

  Cada comando está diseñado para generar su salida en consola de una manera determinada. Por defecto, *Get-Process* genera una salida en forma de tabla.
- Obtén ayuda completa del comando *Format-List*. Para ello, ejecutar el comando siguiente:

  ```
  > Get-Help -Full Format-list
  ```

  El parámetro clave de este comando es *–Property*, que se utiliza para especificar las propiedades de los objetos que se van a mostrar. Si no se especifican propiedades, *Format-list* muestra unas propiedades por defecto que dependen del tipo de objeto de entrada que recibe. El comando *Format-List* muestra los objetos y las propiedades de éstos en formato de lista.
- A continuación, se ejecuta un comando en el que se canaliza *(pipeline)* la salida de *Get-Process* hacia el comando *Format-List*. No se indica ninguna propiedad en *Format-List*, por lo que este comando mostrará las propiedades por defecto que correspondan al objeto proceso.

```
> Get-Process | Format-List
```

Observa que se muestran 5 propiedades de los objetos proceso.

(16) EJERCICIO. Hasta ahora, se ha utilizado *Get-Member* pasándole los objetos de entrada mediante el parámetro *–InputObject*. *Get-Member* también puede recibir los objetos de entrada a través de una canalización. Así, para obtener todas las propiedades y métodos de un objeto proceso, ejecuta el comando *Get-Process* pasándole el nombre de un proceso cualquiera y canaliza su salida hacia *Get-Member*. Escribe el comando a continuación.

(17) EJERCICIO. Ejecuta una línea de comandos que muestre los procesos en ejecución en formato de lista, indicando de cada uno de ellos las siguientes propiedades: Nombre del proceso, fecha y hora en la que ha comenzado a ejecutarse el proceso y ruta en la que se encuentra el fichero ejecutable del proceso. Escribe la línea de comandos ejecutada a continuación.

(18) EJERCICIO. Repite el ejercicio anterior, pero mostrando los procesos (con las tres propiedades indicadas) en formato de tabla. Escribe la línea de comandos ejecutada a continuación.

Canalizaciones múltiples

En los ejemplos vistos hasta ahora se ha utilizado una sola canalización, de modo que solo un comando canaliza su salida hacia otro comando. Sin embargo, no hay límite teórico al número de canalizaciones a realizar. A continuación, se muestra un ejemplo en el que se utiliza una canalización doble.

- Solicita ayuda sobre el comando *Out-Host*:

  ```
  > Get-Help Out-Host
  ```

  Se trata de un comando que genera texto de salida correspondiente a los objetos de entrada que recibe. Dicho texto es enviado a la consola. El comando proporciona el parámetro *-Paging*, que permite partir en páginas la salida realizada en la consola.
- Como ejemplo de canalización múltiple, ejecuta el comando siguiente:

  ```
  > Get-Process | Format-List | Out-Host -Paging
  ```

El comando Sort-Object

Este comando es utilizado de forma habitual en canalizaciones.

- Obtén ayuda del comando *Sort-Object* y después realiza los siguientes ejercicios.

  (19) EJERCICIO. Ejecuta una línea de comandos que muestre todos los procesos del sistema cuyo nombre empiece por ‘s’ y ordenados de forma descendente por su ‘Id’. Escribe el comando a continuación.

  (20) EJERCICIO. El comando *Get-Alias* proporciona los Alias definidos en la consola de PowerShell. Por defecto, este comando ordena su información de salida por el nombre de los Alias. Ejecuta una línea de comandos que muestre los alias definidos en la consola de PowerShell ordenados por su definición, en vez de por su nombre. Escribe la línea de comando ejecutada a continuación.

- **Unidades de PowerShell *(PowerShell drives)***

  Una unidad de PowerShell es un almacén orientado a contener un determinado tipo de información. Las unidades más comunes son las asignadas a los volúmenes del sistema, que contienen sistemas de ficheros. Sin embargo, PowerShell puede asignar unidades a almacenes que contienen otros tipos de información, como por ejemplo, el registro (que contiene la información de configuración del sistema organizada en claves), el almacén de certificados, el almacén de las variables de entorno, el almacén de los alias, etc.

  PowerShell define un conjunto de comandos para manejar unidades. Así, las unidades proporcionan un marco común para gestionar diferentes almacenes de información de una forma estandarizada.

  Cada tipo de unidad es gestionada por un componente software conocido como proveedor. Las unidades asignadas a los volúmenes del sistema son gestionadas por el proveedor *Filesystem*, el registro de Windows por el proveedor *Registry*, etc.

  - Para ver las unidades definidas por defecto ejecutar el siguiente comando:

    ```
    > Get-PSDrive
    ```

    En el caso de estas prácticas se trabajará fundamentalmente con las unidades correspondientes a los volúmenes del sistema. Todas estas unidades son gestionadas por el proveedor *Filesystem*.

    (21) EJERCICIO. Obtén ayuda del comando *Get-PSDrive*. Después, ejecuta este comando de modo que muestre solo las unidades gestionadas por el proveedor de tipo *Filesystem*. Escribe el comando ejecutado a continuación.

  Gestión de la posición actual

  En cada momento, la consola de PowerShell está ubicada en una posición, a la que se hace referencia como posición actual. Ésta viene determinada por una unidad y, en el caso de algunas unidades, por una ubicación dentro de la propia unidad.

  La posición actual es indicada en el símbolo del sistema de la consola de PowerShell. Por defecto, cuando se abre la consola de PowerShell, ésta se posiciona en la unidad C:. Como esta unidad es de tipo *Filesystem*, la posición actual requiere también una ubicación dentro de la unidad; esta ubicación es, por defecto, el directorio del usuario (*\Users\Alumno*).

  Hay un conjunto de comandos de PowerShell que están orientados a trabajar con el contenido de las unidades. Cuando estos comandos se utilizan sin indicar una posición concreta, trabajan siempre sobre la posición actual.

  - Para obtener la posición actual, ejecuta el siguiente comando:

    ```
    > Get-Location
    ```
  - Para cambiar a otra posición se utiliza el comando Set-Location, indicando la ruta de la nueva posición. Por ejemplo, para ubicar la consola en el directorio \Windows, ejecutar el siguiente comando:

    ```
    > Set-Location \Windows
    ```
  - Ejecuta de nuevo *Get-Location* para ver la nueva posición actual.
  - Vuelve a dejar la consola en la posición de partida, es decir, en *\Users\Alumno*.

  Trabajar con los elementos de una unidad

  PowerShell utiliza la nomenclatura *Item* para hacer referencia a los elementos ubicados dentro de una unidad. Los *ítems* representan elementos diferentes en función del tipo de unidad manejada. Por ejemplo, si la unidad es un volumen (*filesystem*), los ítems son ficheros o carpetas, pero si la unidad es de tipo alias, los ítems son los alias definidos en el sistema.

  - Para obtener los ítems disponibles en una determinada ubicación se utiliza el comando *Get-ChildItem*. Cuando este comando se ejecuta sin parámetros, muestra todos los ítems de la ubicación actual. Ejecuta el comando según se indica a continuación:

```
> Get-ChildItem
```

(22) EJERCICIO. Obtén ayuda del comando *Get-ChildItem*. Determina el parámetro necesario para indicar la ubicación de trabajo. Después, ejecuta este comando de modo que obtenga todos los archivos que se encuentren en la ubicación *\Windows\Sytem32* que sean de tipo ejecutable, es decir, que tengan la extensión .exe. Además, se debe conseguir que la salida se muestre página a página. Indica el comando requerido a continuación.

(23) EJERCICIO. Busca los alias definidos para el comando Get-ChildItem. Para ello, obtén ayuda del comando *Get-Alias* y fíjate en el parámetro *–Definition*. Indica el comando requerido a continuación.

El alias *dir* es útil para los usuarios familiarizados con la consola CMD, ya que el comando *Get-ChildItem* aplicado a un volumen funciona igual que el comando *dir* en la consola CMD. De forma similar, el alias *ls* es para los usuarios familiarizados con Unix/Linux.

- Ejecuta los alias *dir* y *ls* y observa que se obtienen los ficheros y carpetas del directorio actual.

A continuación, se realizarán algunas operaciones con *ítems*. Para ello, primero se creará un directorio en el que se llevarán a cabo las operaciones.

- Ubica la consola en el directorio raíz del volumen C. Para ello, ejecuta el comando siguiente:

```
> Set-Location \
```

- A continuación, crearás un directorio para hacer las pruebas. Para ver cómo crear el directorio, se puede buscar los comandos que operan sobre el nombre *ítem*. Para ello, ejecuta

```
> Get-Command -Noun Item
```

Uno de los comandos es *New-Item*. Este comando es el que debe utilizarse para crear un nuevo Ítem, y consecuentemente, un nuevo directorio.

(24) EJERCICIO. Solicita ayuda sobre el comando *New-Item*. Después, utilizando este comando crea un directorio llamado *Temp* en el directorio raíz del volumen C. A continuación, ejecuta otro comando que posicione la consola en el directorio que acabas de crear. Finalmente, ejecuta otro comando que cree en *Temp* un fichero de texto llamado *prueba.txt* que contenga el texto `"Esto es una prueba"`. NOTA: No utilizar el parámetro *–Path* para que las operaciones realizadas por *New-Item* tengan lugar sobre el directorio (posición) actual. Indica los comandos requeridos a continuación.

- Comprueba con el explorador de archivos que se ha creado *prueba.txt* con el contenido especificado.

(25) EJERCICIO. Utilizando el comando *Remove-Item*, elimina el fichero *prueba.txt* y el directorio *Temp*. Indica los comandos requeridos a continuación.

Cambiar la posición actual a otra unidad

- Para cambiar la posición actual a otra unidad se utiliza el comando *Set-Location* usando la sintaxis siguiente:

```
> Set-Location <NOMBRE_UNIDAD>:
```

- Cambia a la unidad alias utilizando el siguiente comando:

  ```
  > Set-Location Alias:
  ```
- Observa el cambio ocurrido en el símbolo del sistema.
- Utiliza el comando *Get-ChildItem* para ver los elementos de la unidad *Alias*.
- Cambia de nuevo la posición actual al volumen C mediante el siguiente comando:

  ```
  > Set-Location C:
  ```