Vamos a seguir con los malditos comandos de Windows:D que son un montón y parece que no terminan más.

Estuve debatiendo conmigo mismo el hecho de qué sacar y qué dejar en esta clase pero sigo sin poder llegar al objetivo claro. Voy a intentar de no hacerlo MEGA extenso y a dejar más para otra clase en otro momento.

Let's start :)

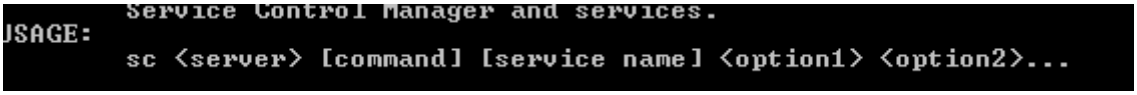
No recuerdo si anteriormente lo habíamos utilizado. Pero si lo hicimos, veámos este comando a profundidad. ***Ping*** es uno de los comandos más simples y útiles. La idea de éste es saber si un equipo, sin importar donde esté, está vivo **con conexión**. Por ejemplo si usamos el comando ***ping www.google.com*** también lograremos verlo.

```
C:\Windows\system32>ping www.google.com

Pinging www.google.com [173.194.42.113] with 32 bytes of data:
Reply from 173.194.42.113: bytes=32 time=55ms TTL=127
Reply from 173.194.42.113: bytes=32 time=13ms TTL=127
Reply from 173.194.42.113: bytes=32 time=14ms TTL=127
Reply from 173.194.42.113: bytes=32 time=21ms TTL=127

Ping statistics for 173.194.42.113:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
Approximate round trip times in milli-seconds:
    Minimum = 13ms, Maximum = 55ms, Average = 25ms
```

Analizemos cada parte. Al principio te dirá de dónde recibió la respuesta de cada paquete enviado. Cada línea es el reporte de un paquete enviado que puede ser **correcto** como aquí, o puede ser **incorrecto**. Luego tenemos que nos confirma la **cantidad** de **bytes** por paquete, que han vuelto en el tiempo que aparece allí. Otros de los componentes es **TTL** o **Time To Live** que representa la cantidad de veces que llego a un **nodo** (switch, router, y cada host intermedio) hasta el host donde fue objetivo del ping. También cabe añadir que el ping **no viajará** por **siempre**. Al llegar a 0, deja de salir.

Lo interesante es que según el **TTL** se puede identificar el **S.O.** al que le hicimos el ping. Si el valor es 128 o cercano (porque responde con un TTL = **128** pero se resta 1 por cada nodo intermedio), es la respuesta típica de un host **Windows** y un host **Linux** usa un TTL = **64**. Ya veremos un poco más a profundidad con los scanners de puertos.

El próximo comando es ***sc*** que es parecido al controlador de los **servicios** de Windows -en realidad se comunica con él-. Está hecho completo para la **administración** y **configuración** de servicios de Windows. Veamos como podemos utilizarlo:).

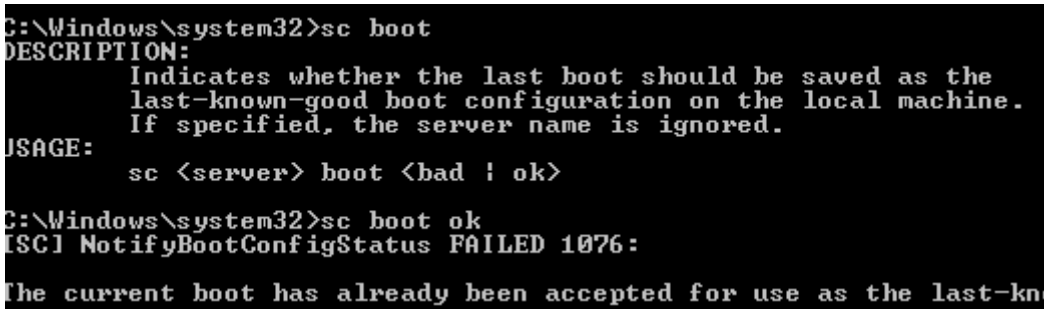


Sc direccionserver(si es que es uno externo, sino nada) **comando servicio opciones**

Los comandos pueden ser tantos como uno se imagina, ya que tiene que ser lo más flexible posible.

Sc boot

Indica cómo debe guardarse el último booteo del sistema (si bien o mal).



sc config

Éste configura los servicios para cambiarle el path, la manera de inicio o el tipo que es. En este caso, puse el servicio con inicio automático.

sc create

La idea es crear un servicio. Los parámetros serían **sc create nombredelservicio binPath="ejecutar el exe y sus parámetros" DisplayName="nombre corto para identificacion del servicio".** Miremos un ejemplo:

sc delete

Al igual que el comando create pero para borrarlo, claro que con menos parámetros.

```
C:\Windows\system32>sc delete FOCASERVICE
[SC] DeleteService SUCCESS
```

sc enumdepend

Enumera los servicios que dependen del servicio en el parámetro.

```
C:\Windows\system32>sc enumdepend tapisrv
[SC] EnumDependentServices: entriesread = 4

SERVICE_NAME: SharedAccess
DISPLAY_NAME: Internet Connection Sharing (ICS)
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 1  STOPPED
        WIN32_EXIT_CODE    : 1077  (0x435)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0

SERVICE_NAME: RemoteAccess
DISPLAY_NAME: Routing and Remote Access
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 1  STOPPED
        WIN32_EXIT_CODE    : 1077  (0x435)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
```

sc failure

La idea de ésta es programar una acción por si un servicio falla.

```
C:\Windows\system32>sc failure
DESCRIPTION:
        Changes the actions upon failure
USAGE:
        sc <server> failure [service name] <option1> <option2>...

OPTIONS:
        reset=   <Length of period of no failures (in seconds)
                  after which to reset the failure count to 0 (may be INFINITE)

                  (Must be used in conjunction with actions= )
        reboot=  <Message broadcast before rebooting on failure>
        command= <Command line to be run on failure>
        actions= <Failure actions and their delay time (in milliseconds),
                  separated by / (forward slash) -- e.g., run/5000/reboot/800
                  Valid actions are <run|restart|reboot> >
                  (Must be used in conjunction with the reset= option)
```

sc getdisplayname

Nos muestra cierta información del proceso.

```
C:\Windows\system32>sc getdisplayname netbios
[SC] GetServiceDisplayName SUCCESS
Name = NetBIOS Interface

C:\Windows\system32>sc getdisplayname tapisrv
[SC] GetServiceDisplayName SUCCESS
Name = Telephony
```

sc lock

Este interesante comando va a trabar el Administrador de Control de servicios. Es decir que ningun servicio se puede iniciar o parar.

sc pause

No muy distinto a lo que veníamos viendo y ya se les va a hacer inductivo. Es para pausar un servicio. Allí en la línea del STATE podemos ver el estado PAUSED.

```
C:\Windows\system32>sc pause tapisrv

SERVICE_NAME: tapisrv
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 7  PAUSED
                                (STOPPABLE, PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

sc continue

Lo mismo que el anterior comando pero para continuarlo luego de su pausa.

```
:\Windows\system32>sc continue tapisrv

ERVICE_NAME: tapisrv
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 4  RUNNING
                                (STOPPABLE, PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
```

sc qc

Consulta la información de un servicio.

```
C:\Windows\system32>sc qc netbios
[SC] QueryServiceConfig SUCCESS

SERVICE_NAME: netbios
        TYPE               : 2  FILE_SYSTEM_DRIVER
        START_TYPE         : 1   SYSTEM_START
        ERROR_CONTROL      : 1   NORMAL
        BINARY_PATH_NAME   : system32\DRIVERS\netbios.sys
        LOAD_ORDER_GROUP   : NetBIOSGroup
        TAG                : 2
        DISPLAY_NAME       : NetBIOS Interface
        DEPENDENCIES       :
        SERVICE_START_NAME :
```

sc query y sc queryex

Éstos comandos son para consultar informacion de los servicios o alguno si es especificado. El queryex lo hace de manera más extendida.

```
SERVICE_NAME: TrkWks
DISPLAY_NAME: Distributed Link Tracking Client
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 4  RUNNING
                             (STOPPABLE, NOT_PAUSABLE, ACCEPTS_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x0
        PID                : 864
        FLAGS              :

SERVICE_NAME: UxSms
DISPLAY_NAME: Desktop Window Manager Session Manager
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 4  RUNNING
```

sc start

Obviamente iniciamos un servicio.

```
C:\Windows\system32>sc start tapisrv

SERVICE_NAME: tapisrv
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 2  START_PENDING
                             (NOT_STOPPABLE, NOT_PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0x7d0
        PID                : 1236
        FLAGS              :
```

sc stop

También deducible, nos detiene un servicio.

```
C:\Windows\system32>sc stop tapisrv

SERVICE_NAME: tapisrv
        TYPE               : 20  WIN32_SHARE_PROCESS
        STATE              : 3  STOP_PENDING
                             (STOPPABLE, PAUSABLE, IGNORES_SHUTDOWN)
        WIN32_EXIT_CODE    : 0  (0x0)
        SERVICE_EXIT_CODE  : 0  (0x0)
        CHECKPOINT         : 0x0
        WAIT_HINT          : 0xfa0
```

Otro de los comandos es ***tasklist*** que, como dice su alias, se usa para **listar** los **procesos** con su **PID**.

```
C:\Windows\system32>tasklist

Image Name                     PID Session Name        Session#    Mem Usage
========================= ======== ================ =========== ============
System Idle Process              0 Services                   0         12 K
System                           4 Services                   0        504 K
smss.exe                       264 Services                   0        640 K
csrss.exe                      340 Services                   0      2,536 K
wininit.exe                    388 Services                   0      2,848 K
csrss.exe                      400 Console                    1      3,548 K
winlogon.exe                   440 Console                    1      3,704 K
services.exe                   484 Services                   0      4,876 K
lsass.exe                      492 Services                   0      5,640 K
lsm.exe                        500 Services                   0      2,532 K
```

Recuerden que el **netstat** también lo hacía aunque de manera distinta. Lo bueno de poder hacer algo de maneras **distintas** es que siempre vas a poder encontrar una manera de hacer lo que quieres.

Y podemos **matar** a alguno de los procesos que vemos allí con su **PID** y el comando ***taskkill***. Supongamos que el **smss.exe** nos miró mal y lo queremos borrar del mapa con un trueno de fondo y risas oscuras. Bueno el comando nos deja lo primero servido.

```
C:\Windows\system32>taskkill /pid 264
ERROR: The process with PID 264 could not be terminated.
Reason: This is critical system process. Taskkill cannot end this process.
```

Como el proceso es **crítico** para el sistema, **no** nos deja matarlo, pero ahí ven la manera de uso:).

Otra de las cosas que podemos hacer es crear una tarea para que Windows la ejecute en el momento específico que nosotros le digamos. Ésto lo hacemos con ***schtask.***

```
C:\Windows\system32>schtasks /CREATE /TR "echo hola" /SC MINUTE /MO 1 /TN hola
SUCCESS: The scheduled task "hola" has successfully been created.
```

Esos parámetros son más o menos los que tiene que tener. El **/CREATE** corresponde para **crear** la taréa, el **/TR** es lo **que** queremos que se ejecute -que puede ser un comando, un ejecutable, un batch, etc-, **/SC** es la **unidad** de tiempo (MINUTE, HOURLY, DAILY) que va a ser acompañado con **/MO** y su número **entero** que dirá cada cuánto se va a ejecutar. Para terminar, **/TN** va a decir el nombre con el que **identificaremos** el proceso.
Cabe decir que lo que hice es de ejemplo. Este proceso ni se va a ver. A parte tiene muchas más opciones que ahora no es necesario verlas. Pueden profundizar ustedes buscando y llamando a la ayuda.

Vamos a dejar de lado estas cosas raras para ver un poco el manejo de archivos.
Los comandos de la consola nos dejan hacer tantas cosas como queramos. Por ejemplo, si queremos **cambiar** los **atributos** de un archivo o directorio vamos a usar ***attrib***. El modo de uso es muy simple. Con "+" **agregamos** el **atributo**, y con "-" lo **quitamos**.

```
C:\Windows\system32>attrib +R +H c:/Users/w7/hola.bat

C:\Windows\system32>_
```

Aquí, por ejemplo, use **R** para darle el atributo de **solo lectura**, y **H** para hacerlo **archivo oculto**.

Ahora supongamos que necesito **encontrar** las **líneas** del **archivo** hola.bat que contengan el string "hola". ***Find*** nos dará la facilidad. Tiene varios comandos para sólo contar la cantidad de líneas que tuvieron; no especificar si es mayúscula o minúscula; o mostrar las líneas que no contienen ese string (con /V, como lo muestro en el ejemplo).

```
C:\Windows\system32>find "hola" c:/Users/w7/hola.bat

---------- C:/USERS/W7/HOLA.BAT
echo hola

C:\Windows\system32>find /V "hola" c:/Users/w7/hola.bat

---------- C:/USERS/W7/HOLA.BAT
PAUSE<NUL
```

Igualmente es otro comando para que vayan intentando practicar y profundizar.
Con ***findstr*** podemos **buscar strings** de distintas maneras. Como ejemplo, yo utilize el **punto** (.) que simboliza un caracter cualquiera, y un **asterisco** (*) que simboliza que el caracter anterior se repetirá cero o más veces. Entonces buscará por cualquier linea que contenga un string "h(cualquiercosaaquí)a". Ya sabemos cual es la línea que mostrará.

```
C:\Users\w7>findstr "h.*a" hola.bat
echo hola
```

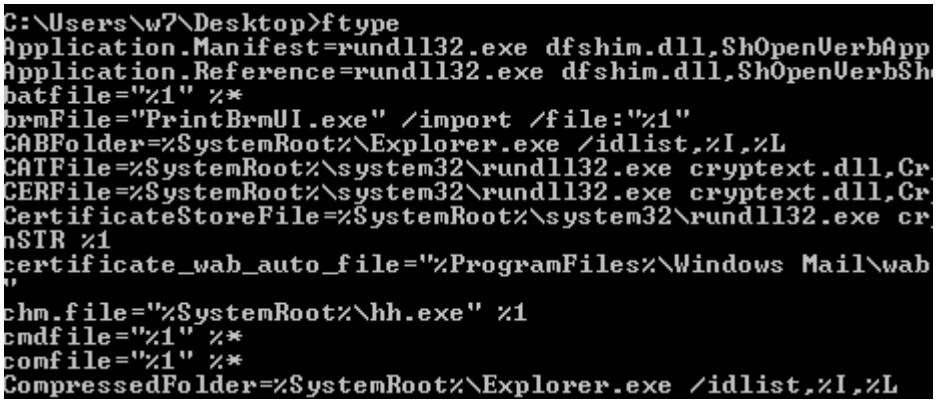
El comando ***move*** no necesita presentación, **mueve** algo a otro directorio. Fácil de entender :).
Ejemplo (recuerden que yo le había dado atributo de sólo lectura. Así no me dejará moverlo):

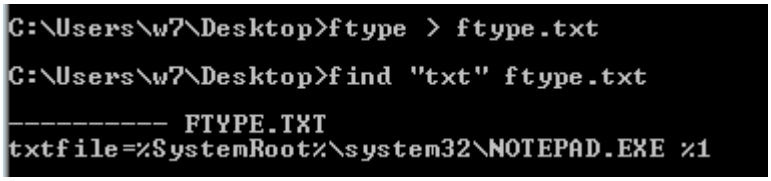
```
C:\Users\w7>attrib -R -H c:/Users/w7/hola.bat

C:\Users\w7>move hola.bat C:/Users/w7/Desktop/hola.bat
        1 file(s) moved.
```

¿Recuerdan cómo listábamos con el comando dir? Bueno, con ***tree*** podemos listar los archivos de manera pseudográfica con una representación de **archivos padre/hijo**. Si no ponemos el parámetro de cuál directorio debe listar, lo hace donde estamos parados.

```
C:\Users\w7>tree
Folder PATH listing
Volume serial number is 00000200 3403:DB3F
C:.
├───.android
├───Contacts
├───Desktop
│   ├───AnubisV1.3
│   │   └───AnubisV1.3
│   │       ├───nmap
│   │       │   ├───nselib
│   │       │   │   └───data
│   │       │   └───scripts
│   │       ├───wfuzz
│   │       │   └───wordlists
│   │       │       ├───Injections
│   │       │       ├───others
│   │       │       ├───stress
│   │       │       └───vulns
│   │       └───whois
│   ├───archivo
```

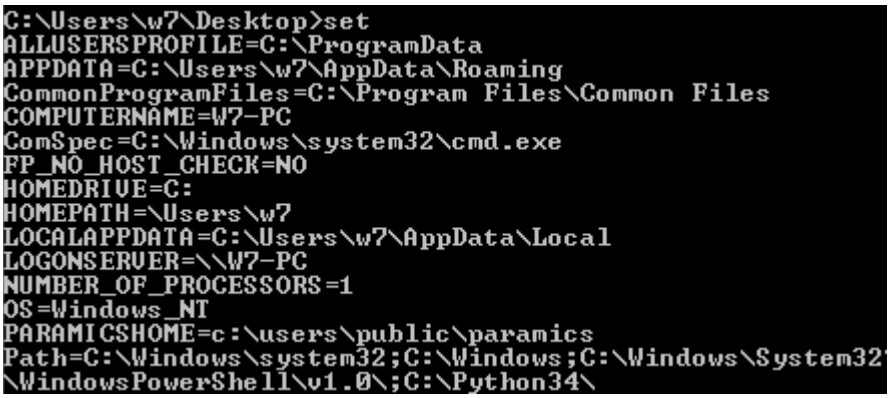
Uno de los comandos que a mi me interesan bastante es ***ftype***. No parece ser demasiado importante cuando lo nombramos, pero lo que hace es **determinar** cómo **abrir** cada **extensión** de archivo. Por ejemplo, si tenemos un archivo con extensión .txt por defecto se abre con el **notepad** y el **%1** significa que aquí va el parámetro del archivo que queremos abrir. Miren como logré ubicar la línea que quería usando dos simples comandos. El primero con ">" mandamos la **salida** del comando a un archivo (txt para poder leerlo) y con el segundo buscamos el string dentro. Lo interesante de esto es que si **cambiamos** el **comando** que hará lo que debería hacer cada vez que un usuario quiera abrir el archivo, podemos allí inyectar cositas nuestras;). Usen la inventiva.





El comando ***path*** servirá para imprimir en pantalla los **path** de los programas que usamos de comandos. Es decir, en estos **directorios**, se encontrarán los exe de "dir", "runas", y todo lo que estuvimos viendo o incluso mucho más, porque si queremos usar algún comando rápido, agregamos el directorio al path de Windows.

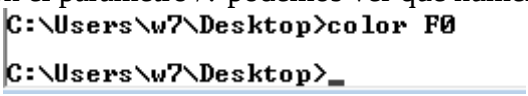
El sistema también tiene guardadas unas **variables** por defecto que se usan para poder **guíar** a ciertos softs. No sé si alguna vez se encontraron con cosas como **%Program Files%** o **%Username%,** pero éstos son formas de poder escribir un path sin saber realmente dónde está ubicado o cuál es el nombre exacto. Con ***set*** nos **muestra** a que cosa corresponde cada **variable**. Se me ocurre que podemos modificar varias entradas para proteger algún malware. Por ejemplo, el ComSpec podría ir a un cmd programado por alguien más, y que los programas estén limitados o que no muestre todo lo que el usuario quiere; o incluso negarle el acceso. Es algo simple pero puede llegar a ser molesto.



Ahora voy a darles algunos comandos variados no demasiado importantes.

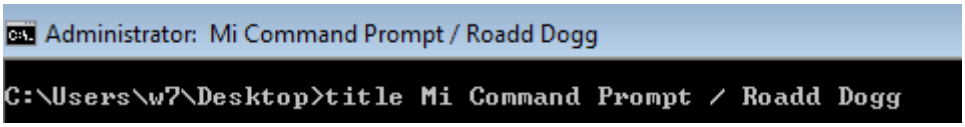
Vamos a ver algo totalmente inútil en cuanto a productividad:D. Con el comando ***color*** vamos a poder cambiar el **color** de la **consola** tanto del **background** como de las **letras**. Lo usamos con un número de dos cifras hexadecimal como parámetro. El primer número corresponde al background y

el segundo al de las letras. Con el parámetro /? podemos ver que número corresponde a cada color.



Voy a seguir usando el color por defecto por simples cuestiones de atención de ustedes. Si lo usamos sin parámetros vamos a volver al color normal.

Otro comando sin ninguna relevancia es ***title*** que cambia el **título** de la ventana del símbolo del sistema activo. Un pequeño ejemplo:



A parte, si queremos salir de la consola lo hacemos con ***exit***, y si queremos **apagar** el **equipo** lo hacemos con ***shutdown*** (si usamos las opciones veremos muchas formas de personalizar ésto).

Terminemos acá. Lamento la tardanza de las clases. Estoy intentando meterle pero no estoy en una época muy tranquila. Ya saben que siempre pueden hacer consultas y además está bueno que no los destruya en la mismas fechas que se dan los exámenes de las universidades.
Espero sacar la próxima clase en poco tiempo:)

Pueden seguirme en Twitter: @RoaddHDC

Cualquier cosa pueden mandarme mail a: r0add@hotmail.com

Para donaciones, pueden hacerlo en bitcoin en la dirección siguiente:
1HqpPJbbWJ9H2hAZTmpXnVuoLKkP7RFSvw

Roadd.
