

# UEFI

## Introducción

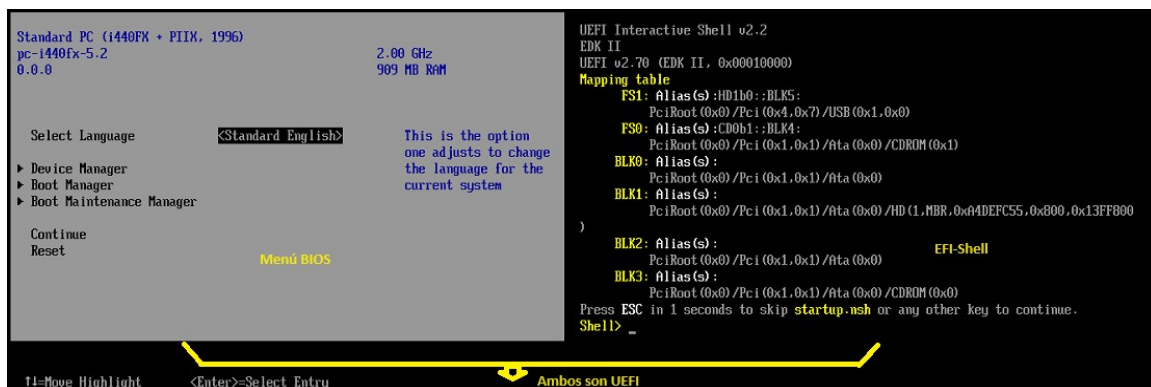
la Unified Extensible Firmware Interface (UEFI), es un especificacion que define la arquitectura del firmware de la plataforma usado para inicializar el hardware de la computadora y es la interface para la interacción con el sistema operativo.

UEFI es independiente de la plataforma y el leguaje de programación, pero usualmente C es usado por referencia para la implementacion de TianoCore EDKII.

Contrario a sus predecesor el BIOS el cual es un estandar *de facto* originalmente creado por IBM como software propietario, UEFI es un estandar abierto mantenido por la consorcio industrial.

La propuesta UEFI no es solo reemplazar al BIOS, sino tambien proveer un entorno que permita diagnosticar y resolver problemas relacionados con el arranque del OS o funcionamiento de hardware, por lo que provee un ambiente EFI-Shell, el cual cuenta con un rico conjunto de comandos que extienden y mejoran la capacidades de la interfaz.

**Nota:** aunque usamos la plabara **BIOS** no nos referimos al PC BIOS, sino a la parte de menus grafica de EDKII a la cual estamos acostumbrados desde las PC compatibles, la cual forma parte de EFI, pero guarda la apariencia para generar un ambiente homoganeo, asi pues tendremos los menus "BIOS" y la interface EFI-Shell



## EFI-Shell

La interface no es muy diferente a una Ventana de Comandos en Windows o a la terminal Bash de Linux, compuesta de un conjunto de comandos, los cuales pueden agruparse en Scripts para realizar tareas mas complejas, se presentan los comandos mas comunes para introducir al uso de la plataforma, conforme se avance en los temas se iran agregando mas comandos, el proposito no es cubrir todo los comandos referentes al EFI-Shell, es mas una guía rapida para que el usuario comience a usar la interface.

Se muestra a continuacion la ventana de la interface:

```

UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
FS1: Alias(s) :HD1b0::BLK5:
    PciRoot (0x0) /Pci (0x4,0x7) /USB (0x1,0x0)
FS0: Alias(s) :CD0b1::BLK4:
    PciRoot (0x0) /Pci (0x1,0x1) /Ata (0x0) /CDROM (0x1)
BLK0: Alias(s) :
    PciRoot (0x0) /Pci (0x1,0x1) /Ata (0x0)
BLK1: Alias(s) :
    PciRoot (0x0) /Pci (0x1,0x1) /Ata (0x0) /HD (1,MBR,0xA4DEFC55,0x800,0x13FF800
)
BLK2: Alias(s) :
    PciRoot (0x0) /Pci (0x1,0x1) /Ata (0x0)
BLK3: Alias(s) :
    PciRoot (0x0) /Pci (0x1,0x1) /Ata (0x0) /CDROM (0x0)
Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
Shell> _

```

Examinando la imagen de arriba hacia abajo tenemos primero el titulo que nos indica cual es la version y modelo de UEFI que estamos manejando (en este caso es EDKII).

La tabla de Mapeo, indica cuales dispositivos estan instalados en nuestro sistema, tenemos que FS0 FS1 se refieren a dispositivos Flash, mientras que BLK1 BLK2 y BLK3 son dispositivos de Bloque de puertos PCI, dependiendo de nuestro hardware, podriamos tener listados otros tipos de dispositivos.

```

FS1: Alias(s) :HD1b0::BLK5:
    PciRoot (0x0) /Pci (0x4,0x7) /USB (0x1,0x0)

```

Podemos observar que algunos dispositivos cuentan con un *Alias(s)* el cual indica que como es que el BIOS lo tratara, podemos ver que FS1 es una unidad Flash extraible, pero que es tratado como si fuera un disco duro HD1B0 el cual es a su vez un dispositivo de bloque BLK5.

```

BLK3: Alias(s) :
    PciRoot (0x0) /Pci (0x1,0x1) /Ata (0x0) /CDROM (0x0)

```

El dispositivo de BLK3 es tratado como un CDROM instalado en un puerto PCI, observamos que la direccion de lbase del arbol de dispositivos PCI es 0x0 el dispositivo esta listado como el primero *Pci* (0x1, 0x1) es tratado como un dispositivo tipo ATA, ya que fisicamente es una unidad CDROM, por lo que se le asigna un controlador de dispositivo acorde.

```

BLK1: Alias(s) :
    PciRoot (0x0) /Pci (0x1,0x1) /Ata (0x0) /HD (1,MBR,0xA4DEFC55,0x800,0x13FF800

```

Mencion especial recibe el dispositivo BLK1, el cual es un HD que posee un sistema Operativo, ya que tiene un Master Boot Record (MRB), y nos indica la direccion en hexadecimal que el BIOS le ha asignado.

Para mayor informacion sobre la asignacion de las direcciones en memoria consulte:



White Paper  
Jenny M. Pelner  
James A. Pelner  
Firmware Architects  
Intel Corporation

## Minimal Intel Architecture Boot Loader

Bare Bones Functionality  
Required for Booting an  
Intel Architecture Platform

En la seccion de Mapa de la memoria, asi como la bibliografia de referencia del documento.

```
Press ESC in 1 seconds to skip startup.nsh or any other key to continue.  
Shell> _
```

Por ultimo tenemos la linea con la palabra **startup.nsh**, este es un comportamiento por defecto del Shell EFI, al iniciar buscara automaticamente ese archivo, el cual podria contener un script de recuperacion o de diagnostico provisto por el fabricante del equipo de computo o tareas de rutina que nosotros hallamos escrito y que deseamos se ejecuten al inicio, en caso de no encontrarse ese archivo o de haber presionado **ESC** se nos mostrara ahora si el Prompt de EFI-Shell.

En ese momento estamos listos para trabajar con nuestra Comand Line Interface (CLI), en caso de que nuestro sistema no cuente habilitado el EFI-Shell tendremos que instalarlo, consulte el Anexo A para mas informacion, mostraremos algunos comandos usuales a continuacion.

### Comandos mas comunes de EFI-Shell

#### *help*

```
Shell> help  
alias      - Displays, creates, or deletes UEFI Shell aliases.  
attrib     - Displays or modifies the attributes of files or directories.  
bcfg       - Manages the boot and driver options that are stored in NVRAM.  
cd         - Displays or changes the current directory.  
cls        - Clears the console output and optionally changes the background  
and foreground color.
```

Nos muestra todos los comandos disponibles en nuestro Shell, la informacion puede ocupar varias pantallas por lo que para poder verla usaremos las teclas de *AvPag Repag* para movernos en el listado. es posible usar configuraciones *more less* para mostrar en partes la informacion, justo como se haria en Ventana de Comandos o Bash.

Si desesamos información de un comando en especificao escribimos *help **commando***, como se muestra en la imagen:

```

Shell> help alias
Displays, creates, or deletes UEFI Shell aliases.

ALIAS [-d|-v] [alias-name] [command-name]

-d          - Deletes an alias. Command-name must not be specified.
-v          - Makes the alias volatile.
alias-name  - Specifies an alias name.
command-name - Specifies an original command's name or path.

NOTES:
1. This command displays, creates, or deletes aliases in the UEFI Shell environment.
2. An alias provides a new name for an existing UEFI Shell command or UEFI application. Once the alias is created, it can be used to run the command or launch the UEFI application.
3. There are some aliases that are predefined in the UEFI Shell environment. These aliases provide the MS-DOS and UNIX equivalent names for the file manipulation commands.
4. Aliases will be retained even after exiting the shell unless the -v option is specified. If -v is specified then the alias will not be valid after leaving the shell.

EXAMPLES:
* To display all aliases in the UEFI Shell environment:

```

Al igual que en otras CLI, los comandos usaran parametros en su sintaxis, donde [] significa parametros que pueden estar presentes o no, asi como los parametros dentro de <> son obligatorios, *help* tambien mostrara para cada comando la sintaxis la descripcion de los parametros Notas y ejemplos de uso asi como informacion adicional, se recomienda familiarizarse con la interpretacion de la linea de sintaxis:

```
ALIAS [-d|-v] [alias-name] [command-name]
```

En este caso el comando ALIAS tiene tres parametros los cuales son opcionales y en el primer parametro [-d | -v] indica que se puede elegir entre dos parametros para esta opcion (-d o -v), asi el simbolo | se interpreta como el condicional logico OR.

Se recomienda usar el comando *help* para encontrar mas informacion sobre los comandos, asi como acudir a la bibliografia provista al final de este documento.

#

Este simbolo sirve para marcar un comentario, lo usamos para indicar al CLI que esas lineas no contienen ordenes ejecutables.

**cls**

Este comando borra la pantalla.

```
UEFI Interactive Shell v2.2
EDK II
UEFI v2.70 (EDK II, 0x00010000)
Mapping table
FS1: Alias(s) :HD1b0:;BLK5:
    PciRoot(0x0)/Pci(0x4,0x7)/USB(0x1,0x0)
FS0: Alias(s) :CD0b1:;BLK4:
    PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)/CDROM(0x1)
BLK0: Alias(s) :
    PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
BLK1: Alias(s) :
    PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)/HD(1,MBR,0xA4DEFC55,0x800,0x13FF800
)
BLK2: Alias(s) :
    PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
BLK3: Alias(s) :
    PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)/CDROM(0x0)
Press ESC in 1 seconds to skip startup.nsh or any other key to continue.
Shell> cls_
```

Podemos ver a la izquierda una pantalla con bastante informacion, tras la ejecucion del comando a la derecha observamos que se ha limpiado la pantalla y nos ha ubicado el Prompt en la primera linea, es posible añadir un color tras limpiar la pantalla la sintaxis es `cls <coloir>`

Donde color puede ser: 1 Blue 2 Green 3 Cyan 4 Red 5 Magneta 6 Yellow

```
Shell> _
```

## echo

sirve para enviar un mensaje a la pantalla de EFI, es el comando mas usado para escribir en la pantalla.

```
Shell> echo "Hola mundo"
Hola mundo
Shell> echo HOLA MUNDO
HOLA MUNDO
Shell> #sin comillas
Shell> _
```

Se recomienda envolver el mensaje entre comillas para poder reconocerlo facilmente.

## reset

En tres variantes este comando reinicia el sistema (<vacio> -c -e) se puede agregar un comentario para indicar el motivo del reinicio, sobre todo cuando se trabaja con servidores, por ejemplo para una tarea de mantenimiento como intentar inciar tras haber actualizado el Firmware.

```

Shell> help reset
Resets the system.

RESET [-w [string]]
RESET [-s [string]]
RESET [-c [string]]

-s      - Performs a shutdown.
-w      - Performs a warm boot.
-c      - Performs a cold boot.
string  - Describes a reason for the reset.

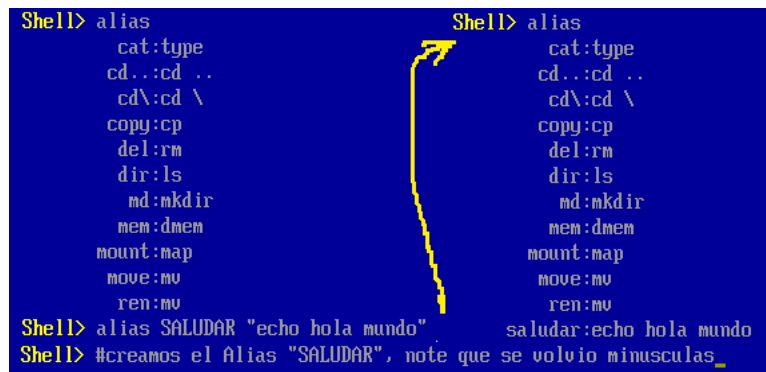
NOTES:
1. This command resets the system.
2. The default is to perform a cold reset unless the -w parameter is
   specified.
3. If a reset string is specified, it is passed into the Reset()
   function, and the system records the reason for the system reset.

```

La opción con el parámetro -s (shutdown) apaga el sistema.

## alias

Si nos referimos a la ayuda del comando *help alias*, observamos que realiza varias tareas, usarlo sin parámetros muestra una lista de los alias en el sistema, donde un alias es un apodo a una línea de comando que puede contener varias instrucciones y parámetros, los alias ayudan a evitar escribir demasiado, ya que es posible ejecutar la acción si necesidad de escribir toda la línea en el CLI cada que se necesite realizar la acción.



```

Shell> alias
cat:type
cd...:cd ..
cd\:cd \
copy:cp
del:rm
dir:ls
md:mkdir
mem:dmem
mount:map
move:mv
ren:mv

Shell> alias
cat:type
cd...:cd ..
cd\:cd \
copy:cp
del:rm
dir:ls
md:mkdir
mem:dmem
mount:map
move:mv
ren:mv
saludar:echo hola mundo
Shell> #creamos el Alias "SALUDAR", note que se volvio minusculas_

```

Si usamos la palabra saludar, aparecerá el mensaje "hola mundo", los alias son **persistentes** es decir, que cuando el sistema se reinicia, estos permanecen en memoria, si deseamos que el valor se borre cada que se reinicia el sistema, debemos agregar el parámetro -v, si deseamos borrar un Alias, utilizamos el parámetro -d seguido del nombre del alias que deseamos borrar (en este caso "saludar").

```

Shell> #usamos el alias "saludar"
Shell> saludar
hola mundo
Shell> #borramos el alias saludar
Shell> alias -d saludar
Shell> #verificamos que se haya borrado
Shell> alias
cat:type
cd...cd ..
cd\:cd \
copy:cp
del:rm
dir:ls
md:mkdir
mem:dmem
mount:map
move:mv
ren:mv
Shell> #se borro el alias "saludar" _

```

Un ejercicio seria crear un alias reiniciar el sistema y verificar que el alias continua existiendo.

### set

Nos ayuda sin parametros a visualizar el conjunto de variables de ambiente que tiene nuestro sistema, a diferencia de los *alias*, que son valores constantes, las variables de ambiente si pueden modificarse, usualmente sirven como paso de parametros entre programas.

```

Shell> set
path = FS0:\efi\tools\;FS0:\efi\boot\;FS0:\;FS1:\efi\tools\;FS1:\efi\boot\;FS1:\
nonesting = False
lasterror = 0x0
profiles = ;Driver1;Install1;Debug1;network1;network2;
uefishellsupport = 3
uefishellversion = 2.2
uefiversion = 2.70
homefilesystem = FS1:
Shell> _

```

por ejemplo la variable *path* muestra las posibles direcciones donde hay archivos que pueden ser accedados mediante EFI-Shell, variables con informacion del sistema, etc.

tambien es posible crear nuestras variables para almacenar informacion que necesitemos.

```

Shell> set file image.boot
Shell> set
path = FS0:\efi\tools\;FS0:\efi\boot\;FS0:\;FS1:\efi\tools\;FS1:\efi\boot\;FS1:\
nonesting = False
lasterror = 0x0
profiles = ;Driver1;Install1;Debug1;network1;network2;
uefishellsupport = 3
uefishellversion = 2.2
uefiversion = 2.70
homefilesystem = FS1:
file = image.boot
Shell> _

```

Usualmente las variables son persistentes, es decir que sobreviviran a un reinicio del sistema, por lo que si deseamos que nuestras variables sean limpiadas cada que reiniciamos el sistema, debemos usar el parametro -v,

para borrar una variable de ambiente usamos el parametro -d (igual que con los alias).

## map

Sin argumentos lista los dispositivos instalados en nuestro sistema que fueron detectados durante el arranque del mismo.

```
Shell> map
Mapping table
FS1: Alias(s) :HD1b0:;BLK5:
      PciRoot(0x0)/Pci(0x4,0x7)/USB(0x1,0x0)
FS0: Alias(s) :CD0b1:;BLK4:
      PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)/CDROM(0x1)
BLK0: Alias(s) :
      PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
BLK1: Alias(s) :
      PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)/HD(1,MBR,0xA4DEFC55,0x800,0x13FF800)
)
BLK2: Alias(s) :
      PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
BLK3: Alias(s) :
      PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)/CDROM(0x0)
Shell> _
```

Esto significa que si nosotros colocamos un nuevo dispositivo como una USB esta no sera visible por mas que ejecutemos el comando, para forzar a una busqueda y actualizacion de la lista de dispositivos, usamos el parametro -r (refresh).

```
Shell> map -r
```

## FS0: (accediendo a un dispositivo)

Tecleando el nombre del dispositivo en el CLI, ingresaremos al dispositivo, lo cual nos permitira ver la informacion contenida en ese medio de almacenamiento.

```
Shell> FS0:
FS0:\> #hemmos cambiado a la unidad FS0
FS0:\> FS1:
FS1:\> #ahora estamos en la unidad FS1_
```

En conjunto con el comando *map* podemos movernos entre dispositivos, la unica manera de regresar al prompt *shell>* es saliendo y volviendo a ingresar al CLI. En la practica como veremos esto rara vez es una limitacion.

## ls

Lista los archivos y directorios almacenados en nuestro dispositivo.

```
FS0:\> ls
Directory of: FS0:\
05/07/2023  20:44 <DIR>          2,048  EFI
05/07/2023  20:44 <DIR>          2,048  boot
           0 File(s)              0 bytes
           2 Dir(s)
FS0:\> _
```

Aunque se muestra una descripcion del tipo de archivo en la tercera columna de izquierda a derecha, los archivos vendran en colores, por lo que facilita su identificacion. Tambien podemos mostrar el contenido de un directorio usando *ls [directorio]*.



```

FS0:\> ls -r EFI
Directory of: FS0:\EFI\
05/07/2023  20:44 <DIR>          2,048 .
05/07/2023  20:44 <DIR>          0 ..
05/07/2023  20:44 <DIR>          2,048 BOOT
0 File(s)          0 bytes
3 Dir(s)

Directory of: FS0:\EFI\BOOT\
05/07/2023  20:44 <DIR>          2,048 .
05/07/2023  20:44 <DIR>          2,048 ..
04/21/2023  14:30          3,440,640 BOOTia32.efi
03/09/2023  02:58           948,768 BOOTx64.efi
04/21/2023  14:30          3,855,808 grubx64.efi
01/30/2023  20:11           849,616 mmx64.efi
4 File(s)        9,094,832 bytes
2 Dir(s)

FS0:\> _

```

Como muestra la figura anterior, usando el parametro `-r` podemos realizar una busqueda recursiva, la cual mostrara el contenido de los directorios que hay dentro de un directorio, use `help ls` para mayor informacion.

## cd

Es una abreviacion de Change Directory, nos permite cambiarnos de de directorio y navegar en el sistema de archivos de nuestros dispositivos.

```

Directory of: FS0:\
05/07/2023  20:44 <DIR>          2,048 EFI
05/07/2023  20:44 <DIR>          2,048 boot
0 File(s)          0 bytes
2 Dir(s)

FS0:\> cd EFI ← 1
FS0:\EFI\> ls
Directory of: FS0:\EFI\
05/07/2023  20:44 <DIR>          2,048 .
05/07/2023  20:44 <DIR>          0 ..
05/07/2023  20:44 <DIR>          2,048 BOOT
0 File(s)          0 bytes
3 Dir(s)

FS0:\EFI\> cd Boot ← 2
FS0:\EFI\Boot\> ls
Directory of: FS0:\EFI\Boot\
05/07/2023  20:44 <DIR>          2,048 .
05/07/2023  20:44 <DIR>          2,048 ..
04/21/2023  14:30          3,440,640 BOOTia32.efi
03/09/2023  02:58           948,768 BOOTx64.efi
04/21/2023  14:30          3,855,808 grubx64.efi
01/30/2023  20:11           849,616 mmx64.efi
4 File(s)        9,094,832 bytes
2 Dir(s)

FS0:\EFI\Boot\> _

```

Observe como nos movemos entre los directorios y mediante el comando `ls` nos enteramos del contenido dentro de cada directorio. para regresar al directorio anterior usamos `cd ..`, para el compendio de opciones disponibles consulte el comando `help cd`.

## cp

Este comando nos permite realizar copias de los archivos, se muestra un ejemplo de uso:

```

12/03/2023  22:12                86 script.nsh
12/04/2023  03:12 <DIR>          8,192 Pepermint
12/04/2023  03:17 <DIR>          8,192 Poppy
12/04/2023  03:20 <DIR>          8,192 Bodhi
12/05/2023  00:00 <DIR>          8,192 ArchLinux
1 File(s)                86 bytes
7 Dir(s)
FS1:\> #copiaremos el archivo script.nsh como script2.nsh
FS1:\> cp script.nsh script2.nsh
Copying FS1:\script.nsh -> FS1:\script2.nsh
- (1)
FS1:\> ls
Directory of: FS1:\
11/29/2023  20:07 <DIR>          8,192 efi
12/02/2023  23:38 <DIR>          8,192 DebianOS
12/02/2023  22:02 <DIR>          8,192 AntiXLinux
12/03/2023  22:12                86 script.nsh
12/04/2023  03:12 <DIR>          8,192 Pepermint
12/04/2023  03:17 <DIR>          8,192 Poppy
12/04/2023  03:20 <DIR>          8,192 Bodhi
12/05/2023  00:00 <DIR>          8,192 ArchLinux
12/10/2023  05:56                86 script2.nsh
2 File(s)                172 bytes
7 Dir(s)
FS1:\> _

```

Observamos que en (1) escribimos el comando de la manera mas simple, el archivo destino y fuente estaran en el mismo directorio, en (2) podemos ver como se ha creado el archivo copia con el nombre que le designamos antes.

## rm

El comando "remove" elimina los archivos:

```

12/03/2023  22:12                86 script.nsh
12/04/2023  03:12 <DIR>          8,192 Pepermint
12/04/2023  03:17 <DIR>          8,192 Poppy
12/04/2023  03:20 <DIR>          8,192 Bodhi
12/05/2023  00:00 <DIR>          8,192 ArchLinux
12/10/2023  05:56                86 script2.nsh
2 File(s)                172 bytes
7 Dir(s)
FS1:\> #ahora eliminaremos el archivo con el nombre script.nsh
FS1:\> rm script.nsh
Deleting 'FS1:\script.nsh'
Delete successful.
FS1:\> ls
Directory of: FS1:\
11/29/2023  20:07 <DIR>          8,192 efi
12/02/2023  23:38 <DIR>          8,192 DebianOS
12/02/2023  22:02 <DIR>          8,192 AntiXLinux
12/04/2023  03:12 <DIR>          8,192 Pepermint
12/04/2023  03:17 <DIR>          8,192 Poppy
12/04/2023  03:20 <DIR>          8,192 Bodhi
12/05/2023  00:00 <DIR>          8,192 ArchLinux
12/10/2023  05:56                86 script2.nsh
1 File(s)                86 bytes
7 Dir(s)
FS1:\> _

```

Podemos ver que en (A) el archivo existe pero en (B) ejecutamos el comando y en (C) observamos que fue eliminado el archivo y ya no esta en la lista.

## mv

Este comando es similar a `cp` solo que el archivo original es eliminado y se conserva solamente la copia, usualmente se utiliza para renombrar archivos,

```
12/02/2023 22:02 <DIR>      8.192 0utDL.t0ux
12/04/2023 03:12 <DIR>      8.192 Peperwinl
12/04/2023 03:17 <DIR>      8.192 Peppy
12/04/2023 03:20 <DIR>      8.192 Bodhi
12/05/2023 00:00 <DIR>      8.192 0rchM.t0ux
12/10/2023 05:56           86 script2.nsh (A)
    1 File(s)          86 bytes
    7 Dir(s)
FS1:\> Musaremos mv para renombrar script2.nsh a script.nsh
FS1:\> mv script2.nsh script.nsh (B)
Moving FS1:\script2.nsh -> FS1:\script.nsh
- [ok]
FS1:\> ls
Directory of: FS1:\
11/29/2023 20:07 <DIR>      8.192 afi
12/02/2023 23:38 <DIR>      8.192 0ebian0S
12/02/2023 22:02 <DIR>      8.192 0utDL.t0ux
12/04/2023 03:12 <DIR>      8.192 Peperwinl
12/04/2023 03:17 <DIR>      8.192 Peppy
12/04/2023 03:20 <DIR>      8.192 Bodhi
12/05/2023 00:00 <DIR>      8.192 0rchM.t0ux
12/10/2023 05:56           86 script.nsh (C)
    1 File(s)          86 bytes
    7 Dir(s)
FS1:\> _
```

Como podemos apreciar en (A) existe el archivo "script2.nsh" y en (B) lo renombramos a "script.nsh", el cual se muestra en (C).

*Nota:* este comando tambien puede eliminar directorios, pero como medida de seguridad se preguntara si se desea eliminar los archivos que pudiera contener el directorio a remover.

## bcfg

Es una abreviacion de "Boot Configuration", permite administrar el orden y los dispositivos de almacenamiento donde el sistema puede inciar.

Usando *bcfg boot dump -b* nos mostrara la lista de los dispositivos desde donde puede inicializarse el sistema:

```
FS1:\> bcfg boot dump -b_
Option: 00. Variable: Boot0004
  Desc - UEFI QEMU HARDDISK QM00001
  DevPath - PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
  Optional- Y
Option: 01. Variable: Boot0001
  Desc - UEFI Blackpcs 92070261AA136006734
  DevPath - PciRoot(0x0)/Pci(0x4,0x7)/USB(0x1,0x0)
  Optional- Y
Option: 02. Variable: Boot0002
  Desc - UEFI QEMU DVD-ROM QM00002
  DevPath - PciRoot(0x0)/Pci(0x1,0x1)/Ata(0x0)
  Optional- Y
Option: 03. Variable: Boot0000
  Desc - UiApp
  DevPath - Fv(7CB8BDC9-F8EB-4F34-AAEA-3EE4AF6516A1)/FvFile(462CAA21-7614-4503-836E-8AB6F4662331)
  Optional- N
Option: 04. Variable: Boot0003
  Desc - EFI Internal Shell
  DevPath - Fv(7CB8BDC9-F8EB-4F34-AAEA-3EE4AF6516A1)/FvFile(7C04A583-9E3E-4F1C-AD65-E05268D0B4D1)
  Optional- N
Option: 05. Variable: Boot0005
  Desc - UEFI PXEv4 (MAC:525400CA49D1)
Press ENTER to continue or 'Q' break: _
  Optional- Y
Option: 03. Variable: Boot0000
  Desc - UiApp
  DevPath - Fv(7CB8BDC9-F8EB-4F34-AAEA-3EE4AF6516A1)/FvFile(462CAA21-7614-4503-836E-8AB6F4662331)
  Optional- N
Option: 04. Variable: Boot0003
  Desc - EFI Internal Shell
  DevPath - Fv(7CB8BDC9-F8EB-4F34-AAEA-3EE4AF6516A1)/FvFile(7C04A583-9E3E-4F1C-AD65-E05268D0B4D1)
  Optional- N
Option: 05. Variable: Boot0005
  Desc - UEFI PXEv4 (MAC:525400CA49D1)
Press ENTER to continue or 'Q' break:
  DevPath - PciRoot(0x0)/Pci(0x3,0x0)/MAC(525400CA49D1,0x1)
  Optional- Y
Option: 06. Variable: Boot0006
  Desc - UEFI PXEv4 (MAC:525400CA49D1) 2
  DevPath - PciRoot(0x0)/Pci(0x3,0x0)/MAC(525400CA49D1,0x1)/IPv4(0.0.0.0)
  Optional- Y
Option: 07. Variable: Boot0007
  Desc - UEFI HTTPv4 (MAC:525400CA49D1)
  DevPath - PciRoot(0x0)/Pci(0x3,0x0)/MAC(525400CA49D1,0x1)/IPv4(0.0.0.0)/Uri()
  Optional- Y
FS1:\> _
```

Como podemos ver este sistema cuenta con 8 opciones de arranque, (contamos desde el 0), nos centraremos en las primeras dos opciones 00 y 01:

```
Option: 00. Variable: Boot0004
Desc      - UEFI QEMU HARDDISK QM00001
DevPath   - PciRoot (0x0) /Pci (0x1,0x1) /Ata (0x0)
Optional- Y
Option: 01. Variable: Boot0001
Desc      - UEFI Blackpcs 92070261AA136006734
DevPath   - PciRoot (0x0) /Pci (0x4,0x7) /USB (0x1,0x0)
Optional- Y
```

Usando el comando `bcfg boot mv 01 00` intercambiaremos de lugar el dispositivo 01 hacia la posicion 00, asi sera la primera opcion para inicializar el sistema:

```
FS1:\> bcfg boot mv 01 00_
```

```
Option: 00. Variable: Boot0001
Desc      - UEFI Blackpcs 92070261AA136006734
DevPath   - PciRoot (0x0) /Pci (0x4,0x7) /USB (0x1,0x0)
Optional- Y
Option: 01. Variable: Boot0004
Desc      - UEFI QEMU HARDDISK QM00001
DevPath   - PciRoot (0x0) /Pci (0x1,0x1) /Ata (0x0)
Optional- Y
```

Podemos observar que los dispositivos han intercambiado posiciones.

## EFI Shell Resumen

Podemos apreciar que la aplicacion no es muy diferente de cualquier otro CLI tipico, la orientacion de este programa es no solo sustituir al BIOS si no facilitar el diagnostico del sistema y asistir en la solucion de problemas de arranque del Sistema Operativo, contiene un listado de comandos no muy extenso, dichos comandos son faciles de usar (si se tiene experiencia con otros CLI), algunos requieren una sintaxis extensa para realizar una operacion, el tecleado puede volverse tedioso con el tiempo si la tarea ha de repetirse extensivamente, contamos con alias y variables de entorno que nos facilitan reproducir el efecto, esto funciona bien para una linea de un comando, pero muchas tareas de mantenimiento requieren la escritura de varios comandos diferentes con sintaxis muy variadas, la edicion de scripts nos permite resolver este problema ademas de proveer una forma de guardar nuestras soluciones de mantenimiento en una unidad extraible (como una USB) y asi usar nuestras soluciones en varios sistemas fisicos, el siguiente tema discutira el proceso de edicion y ejecucion de scripts EFI.

## Shell Scripting en EFI

Como hemos observado con forme usemos los comandos de EFI-Shell, llegara un punto en que necesitaremos repetir varias veces una lista de secuencia de instrucciones, por lo que se necesita una manera eficiente de tratar el problema de la repetibilidad, lo cual es muy usual en tareas de mantenimiento.

Usaremos la aplicacion de Edicion que provee el ambiente EFI, este comando requiere que sea llamado desde un dispositivo fisico, por lo que tendremos que estar en alguno de los dispositivos FSX, con eso tendremos a la vez los archivos disponibles en un medio extraible para ser usados en algun otro equipo de computo.

Comenzaremos creando un directorio de trabajo donde colocaremos todos nuestros scripts, usaremos el comando **mkdir** para crear dicha area de trabajo:

```

FS1:\> mkdir scripts
FS1:\> ls
Directory of: FS1:\
11/29/2023 20:07 <DIR>      8,192  efi
12/02/2023 23:38 <DIR>      8,192  biosd03
12/02/2023 22:02 <DIR>      8,192  boot08.100x
12/04/2023 03:12 <DIR>      8,192  Popcornial
12/04/2023 03:17 <DIR>      8,192  Pappu
12/04/2023 03:28 <DIR>      8,192  budhi
12/05/2023 00:08 <DIR>      8,192  ArchLinux
12/10/2023 05:56 (A) → 86  script.nsh
12/10/2023 07:02          9,524  EFI_Commands
12/10/2023 07:03          8,586  befy_Lista
12/10/2023 19:32 <DIR> (B) → 8,192  scripts
          3 File(s)      18,196 bytes
          8 Dir(s)
FS1:\> _

```

En (A) este es un archivo de EFI-Script, observe la extension *.nsh*, en contraste (B) es un directorio *scripts* es donde guardaremos todos nuestros scripts para tener un mejor control, usando el comando **cd** nos movemos dentro de nuestro area de trabajo.

```

FS1:\> cd scripts
FS1:\scripts> ls
Directory of: FS1:\scripts\
12/10/2023 19:32 <DIR>      8,192  _
12/10/2023 19:32 <DIR>      0
          0 File(s)          0 bytes
          2 Dir(s)
FS1:\scripts> _

```

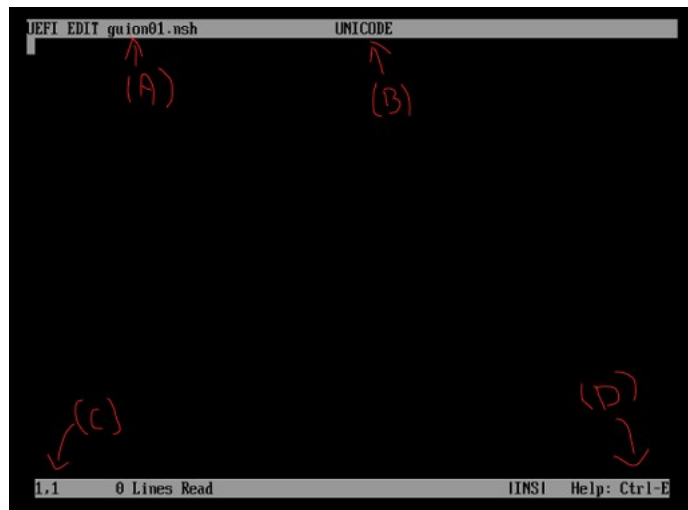
Invocamos al editor de scripts (y texto simple) para nuestro primer script al que llamaremos *guion01.nsh* (los script tendran extension *nsh*, mientras que los ejecutables sera *efi*).

```

FS1:\scripts> edit guion01.nsh_

```

Nos enviaran al area de edicion:



Describimos brevemente el area de edicion (A) el nombre del archivo que estamos editando, (B) el juego de caracteres usado (consulte *help* para mayor informacion), (C) El indicador en pares *LINEA-COLUMNA*, (D) La combinaci3n de teclas para entrar al men3 del editor que nos brinda mas informacion sobre el uso del programa.

Escribimos lo siguiente:

```
UEFI EDIT guion01.nsh      UNICODE      Modified
echo "hola mundo"
_

2.1      IINSI      Help: Ctrl-E
```

Presionamos *ctrl + s* para guardar los cambios del archivo (puede que necesite presionar la tecla de flecha hacia abajo para que se muestre el menu de guardado en la parte inferior derecha del area de edicion) y *ctrl + q* para salir del editor, en el CLI (Comand Line Interface), usamos *ls* para verificar la creación del archivo y *cat* para la verificar el contenido del mismo:

```
FS1:\scripts> ls
Directory of: FS1:\scripts\
12/10/2023  19:32 <DIR>          8,192
12/10/2023  19:32 <DIR>           0
12/10/2023  19:48          44  guion01.nsh
1 File(s)          44 bytes
2 Dir(s)
FS1:\scripts> cat guion01.nsh
echo "hola mundo"
FS1:\scripts> _
```

Podemos ejecutar el script tanto si usamos el prompt de la direccion absoluta (en caso de que el script no este en el directorio actual), o simplemente con su nombre (si se encuentra en el directorio actual como en este caso)

```
FS1:\scripts> .\guion01.nsh
FS1:\scripts> echo "hola mundo"
hola mundo
FS1:\scripts> #ejecucion del script sin el prompt de direccion (path)
FS1:\scripts> guion01.nsh
FS1:\scripts> echo "hola mundo"
hola mundo
FS1:\scripts> _
```

Observamos que nos muestra el envio de la instruccion *echo "hola mundo"* en la linea del CLI, y despues nos muestra la ejecucion de la instruccion, si deseamos que no se muestren las instrucciones y solo ver el resultado, editamos el archivo y agregamos la siguiente instruccion *@echo -off*

```
UEFI EDIT guion01.nsh
@echo -off
echo "hola mundo"
```

Guardamos los cambios, salimos y ejecutamos nuevamente el script:

```
FS1:\scripts> #ejecutamos el script
FS1:\scripts> guion01.nsh
hola mundo
FS1:\scripts> #observamos que no se muestra la linea de la instruccion
FS1:\scripts> _
```

El simbolo *@* escrito antes de cada instruccion evita que se muestre en el CLI, pero para evitar escribir en todas las

líneas el símbolo, usamos el parámetro `-off` de la instrucción `echo`, para indicar que no se mostraran las líneas de instrucción en el CLI (recuerde que los Scripts son interpretados, es decir ejecutados línea por línea a la vez).

### Variables (de ambiente)

Podemos acceder a las variables de ambiente del CLI desde el editor:

```
FS1:\scripts> set
path = FS0:\efi\tools\FS0:\efi\boot\FS0:\FS1:\efi\tools\FS1:\efi\boot\F
S1:\
nonesting = False
lasterror = 0x0
profiles = :Driver1:Install1:Debug1:network1:network2:
uefihellsupport = 3
uefihellversion = 2.2
uefiversion = 2.70
homefilesystem = FS1:
cwd = FS1:\scripts
FS1:\scripts> _
```

Por ejemplo si deseamos mostrar el valor de la versión actual de nuestro EFI-Shell (llamada Shell de ahora en adelante), modificamos nuestro script como sigue:

```
UEFI EDIT guion01.nsh          UNICODE
echo -off
#vamos a mostrar el numero de version
#de nuestro Shell
echo "UEFI Version    = %uefiversion%"
```

El nombre de la variable va rodeada por el símbolo %, **%uefiversion%**, en este caso, guardamos el archivo, salimos y ejecutamos el script:

```
FS1:\scripts> guion01.nsh
UEFI Version    = 2.70
FS1:\scripts> _
```

Observamos que la información se muestra en pantalla tal como se esperaba, podemos también crear nuestras propias variables, las cuales pueden ser o no volátiles con el parámetro `-v`.

```
UEFI EDIT guion01.nsh          UNICODE
echo -off

#Creamos nuestras variables de ambiente propias
#Por convencion las variables se escriben en Mayusculas
#para ayudar a distinguirlas

set -v NOMBRE_ARCHIVO guion01.nsh

#usando el parametro -v la variable no es persistente

echo "Nombre del Script    = %NOMBRE_ARCHIVO%"

_
```

Guardamos, Salimos y Ejecutamos el script:

```
FS1:\scripts> guion01.nsh
Nombre del Script    = guion01.nsh
FS1:\scripts> _
```

**Nota:** La opción autocompletar está disponible en el CLI escriba las primeras letras del comando o archivo que dese referirse y use TAB.



```
FS1:\scripts> set
path = FS0:\efi\tools\;FS0:\efi\boot\;FS0:\;FS1:\efi\tools\;FS1:\efi\boot\;FS1:\
S1:\
nonesting = False
lasterror = 0x0
profiles = :Driver1:Install1:Debug1:network1:network2:
uefishellsupport = 3
uefishellversion = 2.2
uefi version = 2.70
homefilesystem = FS1:
cwd = FS1:\scripts
NOMBRE_ARCHIVO = guion01.nsh
FS1:\scripts> _
```

Al ser variable de ambiente, podemos apreciar que se creo en nuestro CLI, la variable NOMBRE\_ARCHIVO, usar la opcion -d en lugar de -v removera la variable de ambiente, en general un script es un compendio de instrucciones del CLI, asi que todo lo que se puede hacer en el Shell se puede "copiar y pegar" en un Script para ejecutarse como una lista de instrucciones, esa es la esencia general de los Scripts.

### **Parametros de linea**

Cuando invocamos un comando de linea, podemos incluir valores al script, estos se guardan en unas variables especiales numeradas simplemente del 1 al 9, para acceder a su valor guardado usamos el simbolo de % y el numero de la variable, no necesitan ir entre % como las variables de ambiente.

```
FS1:\scripts> cat guion01.nsh
@echo -off

#escribimos el valor de la primera variable de parametro
echo "Parametro 1: %1"
#el segundo parametro
echo "Parametro 2: %2"

FS1:\scripts> guion01.nsh 123 456
Parametro 1: 123
Parametro 2: 456
FS1:\scripts> #podemos ver que los valores de los parametros se muestran_
```

Nuestro script *guion01.nsh* recibio como primer parametro "123" y como segundo "456", los cuales son mostrados en la salida.

```
FS1:\scripts> guion01.nsh "hola" 456
Parametro 1: hola
Parametro 2: 456
FS1:\scripts> #los valores no estan limitados a numeros
FS1:\scripts> _
```

Los Parametros pueden ser tambien cadenas de caracteres, no se limita solo a numeros.

### **Instrucciones de seleccion**

#### **Operadores relacionaes**

El siguiente codigo resume la sentencia de seleccion sencilla y los operadores relacionales basicos, igual, menor y mayor que.

```

FS1:\scripts> cat guion02.nsh
@echo -off

#el simbolo == indica una comparacion por igualdad
if %1 == 5 then
    echo "Parametro 1 es igual a 5"
endif

#no existe simbolo para menor que se utiliza en su
#lugar lt : less than
if %1 lt 5 then
    echo "%1 es menor que 5"
endif

#Tampoco hay un simbolo para mayor que se usa
#en su lugar gt : greater than
if %1 gt 5 then
    echo "%1 es mayor que 5"
endif

FS1:\scripts> _

```

Al ejecutar el script con valores que cumplan las condiciones observamos:

```

FS1:\scripts> guion02.nsh 5
Parametro 1 es igual a 5
FS1:\scripts> #operador igual
FS1:\scripts> guion02.nsh 4
4 es menor que 5
FS1:\scripts> #operador menor que lt
FS1:\scripts> guion02.nsh 6
6 es mayor que 5
FS1:\scripts> #operador mayor que gt_

```

En resumen las palabras reservadas para las comparaciones son: == : igual que, lt : menor que, gt : mayor que, respectivamente.

### ***Operadores logicos***

El siguiente código muestra como realizar la operacion inversa de un operador relacional:

```

FS1:\scripts> cat guion03.nsh
@echo -off

#agregar la palabra not invierte el operando
#en este caso es no igual a

if not %1 == 5 then
    echo "%1 no es igual a 5"
endif
FS1:\scripts> #probamos cumpliendo la condicion
FS1:\scripts> guion03.nsh 6
6 no es igual a 5
FS1:\scripts> #ahora elegimos 5 para verificar el caso contrario
FS1:\scripts> guion03.nsh 5
FS1:\scripts> #no hubo salida, significa que no se cumpio la condicion_

```

Para la negacion tenemos que usar la palabra *not* antes del operador como muestra el código anterior, si necesitamos que se cumplan ciertas condiciones usamos la concatenacion de operadores relacionales con operadores logicos:

```

FS1:\scripts> cat guion04.nsh
@echo -off

#se incluyen tambien los operadores logicos

if %1 gt 2 and %1 lt 7 then
    echo "el valor esta comprendido entre 2 y 7"
    echo "sin incluir los valores extremos (2,7)"
endif

FS1:\scripts> guion04.nsh 5
el valor esta comprendido entre 2 y 7
sin incluir los valores extremos (2,7)
FS1:\scripts> guion04.nsh 2
FS1:\scripts> guion04.nsh 7
FS1:\scripts> #no hay mensaje porque estos valores
FS1:\scripts> #estan excluidos
FS1:\scripts> _

```

El código muestra el operador *and* queremos incluir los valores de los extremos usamos el operador *or* y anidamos sentencias *if*:

```

FS1:\scripts> cat guion05.nsh
@echo -off

#se incluyen tambien los operadores logicos
if %1 gt 2 or %1 == 2 then
    if %1 lt 7 or %1 == 7 then
        echo "el valor esta entre 2 y 7"
    endif
endif

FS1:\scripts> #probamos con el valor medio 5
FS1:\scripts> guion05.nsh 5
el valor esta entre 2 y 7
FS1:\scripts> #probamos con el valor menor 2
FS1:\scripts> guion05.nsh 2
el valor esta entre 2 y 7
FS1:\scripts> #probamos con el valor mayor 7
FS1:\scripts> guion05.nsh 7
el valor esta entre 2 y 7
FS1:\scripts> #probamos con el valor 1 fuera de rango
FS1:\scripts> guion05.nsh 1
FS1:\scripts> #no se escribio nada como se esperaba_

```

Dependiendo de la version EFI que estemos manejando sera la cantidad de comandos y operadores con los que contaremos, tambien hay que tener en cuenta que el CLI no fue diseñado para ser un lenguaje propiamente de programacion, por lo que carecera de muchas de las características de otros Shell o lenguajes interpretados.

***if <Condiciones> then***

***[sentencias]***

***else***

***[sentencias]***

***endif***

Como en otros lenguajes interpretados, tambien contamos con la variante *si.. entonces.. sino..* en el CLI, asi modificando un poco el programa anterior:

```

FS1:\scripts\> cat guion06.nsh
@echo -off

#se incluyen tambien los operadores logicos
if %1 == 2 or %1 == 7 then
    echo "el valor es igual a 2 o 7"
else
    if %1 gt 2 and %1 lt 7 then
        echo "el valor esta entre 2 y 7"
    else
        echo "el valor esta fuera del rango"
    endif
endif

FS1:\scripts\> guion06.nsh 5
el valor esta entre 2 y 7
FS1:\scripts\> guion06.nsh 2
el valor es igual a 2 o 7
FS1:\scripts\> guion06.nsh 7
el valor es igual a 2 o 7
FS1:\scripts\> guion06.nsh 1
el valor esta fuera del rango
FS1:\scripts\> guion06.nsh 8
el valor esta fuera del rango
FS1:\scripts\>

```

Nos permite enriquecer la capacidad de decision que tenemos para crear nuestros scripts.

#### ***Verificando al existencia de un directorio o archivo***

Una operacion muy importante es poder averiguar si un archivo existe o no, así podemos evitar perdida de información.

```

FS1:\scripts\> cat verificar_archivo.nsh
@echo -off

#le pasamos el nombre del archivo o directoiro
#al script en la primer variable de parametro
if exist %1 then
    echo "%1 si existe"
else
    echo "%1 no existe"
endif

FS1:\scripts\> verificar_archivo.nsh guion01.nsh
guion01.nsh si existe
FS1:\scripts\> verificar_archivo.nsh guion07.nsh
guion07.nsh no existe
FS1:\scripts\> _

```

La instrucción ***exist*** nos permite verificar si un archivo esta presente o no, retorna un valor TRUE o FALSE que puede ser aprovechado por una instruccion de selección.

#### ***exit***

Este comando nos permite terminar la ejecucion de un script de manera adecuada, debemos de asgurar regresar un valor de retorno, el cual sirve para indicar que el programa termino adecuadamente.

```

FS1:\scripts\> cat salir.nsh
@echo -off
#verificamos si coloco algun parametro
if %1 == "" then
    echo "Usage: salir.nsh <name>"
    exit /b 1 #valor de retorno ejecucion incorrecta
endif
echo "Bienvenido, %1"
exit /b 0 #un valor de retorn ejecucion correcta
FS1:\scripts\> salir.nsh
Usage: salir.nsh <name>
FS1:\scripts\> #el valor de retorno esta guardado en
FS1:\scripts\> echo %lasterror%
0x1
FS1:\scripts\> salir.nsh Rodrigo
Bienvenido, Rodrigo
FS1:\scripts\> echo %lasterror%
0x0
FS1:\scripts\> #ese resultado sera util mas adelante_

```

El resultado de la terminacion del script se guarda en la variable de entorno **lasterror**, este resultado binario sera util mas adelante, considere por el momento que tambien es posible llamar scripts dentro de scripts.

### Revisar el estado de ejecucion de un programa

Verificaremos si la ejecucion de un programa fue correcta, mostrando solamente su valor de retorno, el cual sera 0x0 si fue exitoso o diferente de este valor si fue un fracaso.

```

FS1:\> cat \scripts\revisarExito.nsh
@echo -off
#tomaremos el primer parametro y lo usaremos
#como ruta para de acceso para el comando
#CD
cd %1
echo "Estado de ejecucion del comando: %lasterror%"
#si el comando es ejecuta con exito regresara 0x0
#en caso de erro un valor diferente cualquiera
FS1:\> #probamos con un parametro que sera exitoso
FS1:\> \scripts\revisarExito.nsh scripts
Estado de ejecucion del comando: 0x0
FS1:\scripts\> #ahora con un valor que fallara
FS1:\scripts\> \scripts\revisarExito.nsh scripts
cd: 'FS1:\scripts\scripts' is not a directory.
Estado de ejecucion del comando: 0xE
FS1:\scripts\> #El codigo de error es diferente de 0
FS1:\scripts\> #Se deja al lector la deducccion del fallo
FS1:\scripts\> #Pista: observe el PATH del prompt_

```

En el programa anterior se utilizo el mismo parametro pero el programa fallo, esto es porque la instruccion **cd** intento acceder a un directorio inexistente, como indica la salida del codigo 0xE.

Revisar el código de ejecucion contenido en la varialbe de ambiente **lasterror** asi como los mensajes de error arrojados por los comandos son tareas de utilidad en la creación de scripts.

### Lazos for.. in .. endfor

Si deseamos realizar un proceso para una cantidad de elementos conocida, usamos el lazo FOR

```

FS1:\scripts\> cat loop1.nsh
@echo -off

for %a in 11 22 33 44 55 66 77 88 99 Esta es una "linea de" Texto
echo "%a"
endfor
FS1:\scripts\> loop1.nsh
* 11
* 22
* 33
* 44
* 55
* 66
* 77
* 88
* 99
* Esta
* es
* una
* linea de
* Texto
FS1:\scripts\> _

```

Podemos apreciar como la cadena de texto fue manejada como si cada palabra fuera un elemento sobre el cual "iterar". Una característica que salta a la vista es la manera en la que funciona la instrucción *for*, la razón por la que itera sobre una lista de elementos es para facilitar el acceso a las variables de parámetros.

Si deseamos iterar no en una lista de elementos sino en un rango, usamos la notación de índices (inicio final), como se muestra a continuación.

```

FS1:\scripts\> cat loop2.nsh
@echo -off
for %a run (1 10)
echo "%a"
endfor

FS1:\scripts\> loop2.nsh
FS1:\scripts\> @echo -off
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
FS1:\scripts\> _

```

No solo podemos avanzar de un elemento a un elemento tenemos la opción de indicar el paso entre cada elemento:

```

FS1:\scripts\> cat loop3.nsh
@echo -off
#usaremos una seleccion para mostrar pares o impares
#el parametro 1 = impar 2 = par
if %1 == 1 then
    for %a run (1 10 2)
        echo "%a"
    endfor
else
    if %1 == 2 then
        for %a run (2 10 2)
            echo "%a"
        endfor
    else
        echo "Seleccion no disponible "
    endif
endif
FS1:\scripts\> _

```

Como podemos apreciar estamos incluyendo lo anteriormente aprendido anidando bloques *for* dentro de bloques *if*, si agregamos como parametro 1 se imprimiran los impares, 2 los pares y cualquier otra seleccion indicara "no disponible".

```

FS1:\scripts\> loop3.nsh 1
* 1
* 3
* 5
* 7
* 9
FS1:\scripts\> loop3.nsh 2
* 2
* 4
* 6
* 8
* 10
FS1:\scripts\> loop3.nsh 3
Seleccion no disponible
FS1:\scripts\> _

```

En el ejemplo anterior una mejora seria agregar un código de Terminacion de Ejecucion y la apropiada salida del programa así como indicar el uso del script, lo cual seria mas cercano al funcionamiento esperado de un comando de un CLI.

### ***Salto en el código y desplazamiento de parametros***

Una particularidad del CLI es que solo contamos con 9 variables de parametros, por lo que si deseamos ingresar mas de esta cantidad, hay que usar la instruccion *shift* y ser creativos con las instrucciones que nos ofrecen por defecto, *goto* es una instruccion que debe manejarse con cuidado.

```

FS1:\scripts> cat saltoDesplazamiento.nsh
#una particularidad es que no podemos ingresar mas
#de 9 parametros, es decir %1 %2 %3 %4 %5 %6 %7 %8 %9
#corresponden a las variables que podemos acceder
#la instruccion shift nos ayuda a manejar mas de 10
#parametros
#%0 es reservado al nombre del comando
@echo -off
#si el primer argumento no esta vacio se muestra en
#pantalla, la sintaxis :PRINT es una etiqueta para goto
:PRINT
if not %1 == "" then
    echo "%1"
endif
#Si el segundo argumento esta disponible
#es decir %2 tiene un dato, lo desplazamos
#a la ubicacion del primer parametro con
#la instruccion shift
if not %2 == "" then
    shift #el contenido de %2 se mueve a %1
    goto PRINT
endif
#un efecto colateral de usar SHIFT es que los
#argumentos se recorren una ubicacion TODOS
FS1:\scripts> _

```

Debemos estar preparados que pese a todas nuestras mejores intenciones, muchos de los comandos y técnicas de programación podrían no funcionar como esperamos, esto puede ser debido las limitantes de la versión de EFI que estemos manejando.

```

FS1:\scripts> saltoDesplazamiento.nsh 1 2 3 4 5 6 7 8 9 10
* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
FS1:\scripts> #recorrimos uno a uno los parametros hasta llegar al 10
FS1:\scripts> #pero no se ha mostrado el parametro, debido a las limitaciones
FS1:\scripts> #de la version de EFI y memoria impuestas en la MU usada_

```

Esto cubre la parte introductoria a la creación de Scripts en EFI-Shell

### Conclusion de la creación de Scripts:

Tenemos que tomar en cuenta que EFI-Shell no está diseñado como un lenguaje de programación o de scripts general, es una herramienta para ayudar a realizar tareas de mantenimiento y recuperación, por lo que no tendrá todas las prestaciones esperadas, podemos notar que no tenemos acceso a instrucciones para interactuar con el usuario y obtener información de él, la única manera de que el usuario ingrese datos es mediante los parámetros.

## Opciones Miscelaneas del EFI-SHELL

### Redireccion de la salida:

En ocasiones será necesario recuperar información presentada en la pantalla del CLI, para su posterior análisis,



usando el comando > (se enviara la salida de la izquierda hacia la derecha). Si el archivo destino no existe se creara.

```
FS1:\> ls
Directory of: FS1:\
11/29/2023  20:07 <DIR>          8,192  efi
12/02/2023  23:38 <DIR>          8,192  DebianOS
12/02/2023  22:02 <DIR>          8,192  AntiXLinux
12/04/2023  03:12 <DIR>          8,192  Pepermint
12/04/2023  03:17 <DIR>          8,192  Puppy
12/04/2023  03:20 <DIR>          8,192  Bodhi
12/05/2023  00:08 <DIR>          8,192  ArchLinux
12/10/2023  05:56                86  script.nsh
12/10/2023  07:02            9,524  EFI_Commands
12/10/2023  07:03            8,586  bcfg_Lista
12/12/2023  22:02 <DIR>          8,192  scripts
          3 File(s)      18,196 bytes
          8 Dir(s)
FS1:\> ls > listaArchivos.txt
FS1:\> _
```

Usando el comando EDIT para verificar el contenido del archivo listaArchivos.txt:

```
UEFI EDIT listaArchivos.txt  UNICODE
Directory of: FS1:\
11/29/2023  20:07 <DIR>          8,192  efi
12/02/2023  23:38 <DIR>          8,192  DebianOS
12/02/2023  22:02 <DIR>          8,192  AntiXLinux
12/04/2023  03:12 <DIR>          8,192  Pepermint
12/04/2023  03:17 <DIR>          8,192  Puppy
12/04/2023  03:20 <DIR>          8,192  Bodhi
12/05/2023  00:08 <DIR>          8,192  ArchLinux
12/10/2023  05:56                86  script.nsh
12/10/2023  07:02            9,524  EFI_Commands
12/10/2023  07:03            8,586  bcfg_Lista
12/12/2023  22:02 <DIR>          8,192  scripts
12/12/2023  23:06                2  listaArchivos.txt
          4 File(s)      18,198 bytes
          8 Dir(s)

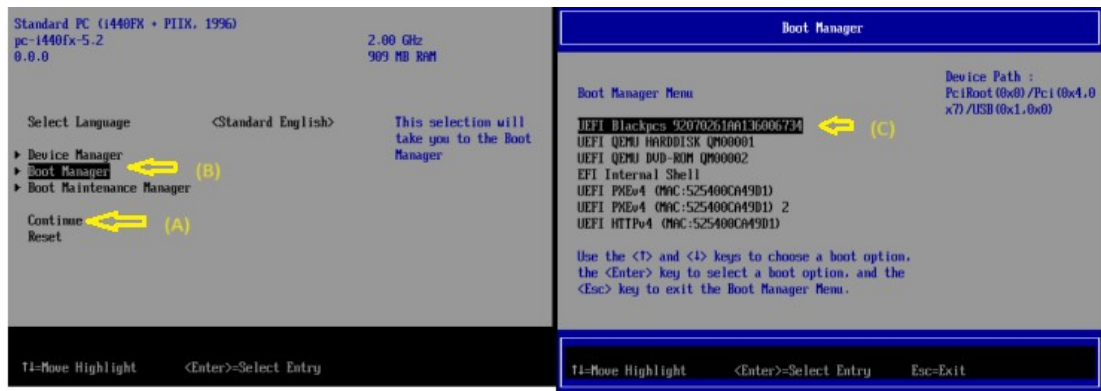
1.1      16 Lines Read      IINSI  Help: Ctrl-E
```

Podemos constar que el contenido es el mismo que se mostro anteriormente con *ls*, solo que ahora esta incluido el archivo *listaArchivos.txt* que fue creado mediante la redireccion de la salida. Si usamos el simbolo > nuevamente en el mismo archivo de salida, la información que contenia se reescribira y se perdera sera suplantada por la nueva informacion, si desea conservar la informacion y solo agregar la nueva se debera usar el simbolo >> el cual apendizara la nueva información al final del archivo, evitando la perdida de información.

## Anexo A

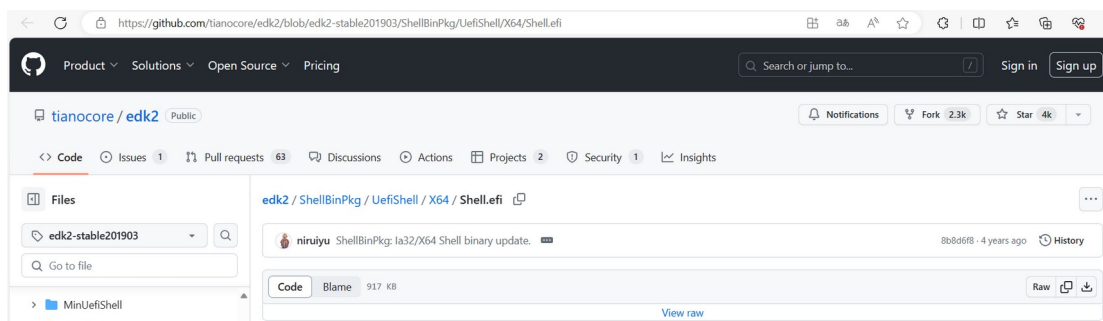
## Crear una USB con EFI-Shell

Configuración de UEFI en sistemas que no cuentan con EFI-Shell instalado como parte de su sistema de arranque, tendremos que ingresar al menú del BIOS (una parte de UEFI, con apariencia antigua si se usa EDK2).



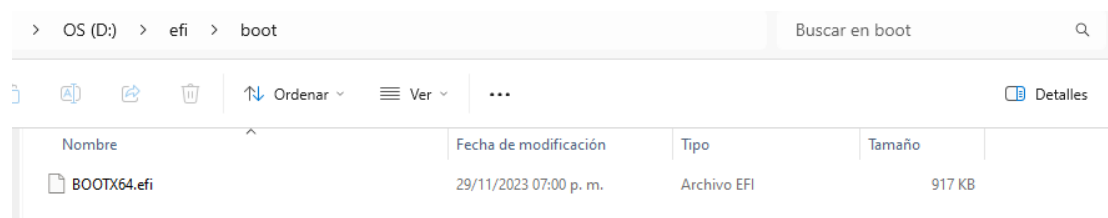
Si escogemos la opción (A) el sistema inicializará siguiendo el orden normal de dispositivos, podemos escoger colocar una USB con los archivos de EFI, así cada vez que iniciemos si esta instalada esta memoria el sistema entrará directo a EFI-Shell, en (B) podremos iniciar nuestro sistema desde cualquier dispositivo válido conectado, como se muestra en (C) la USB contiene nuestro ambiente EFI-Shell que necesitamos, algunos sistemas de cómputo tendrán un menú diferente al mostrado aquí, será cuestión de navegar en las opciones o consultar el manual del fabricante de su tarjeta madre, y seleccionar la opción Iniciar desde una USB, puede que tenga que desactivar opciones de seguridad como **inicio seguro (safe boot)**.

La USB deberá estar formateada en sistema FAT32 y contendrá un directorio /efi/boot el cual tendrá el archivo BOOTX64.efi (o X32 según el caso). Descarga el ejecutable de la página del proyecto EDK2

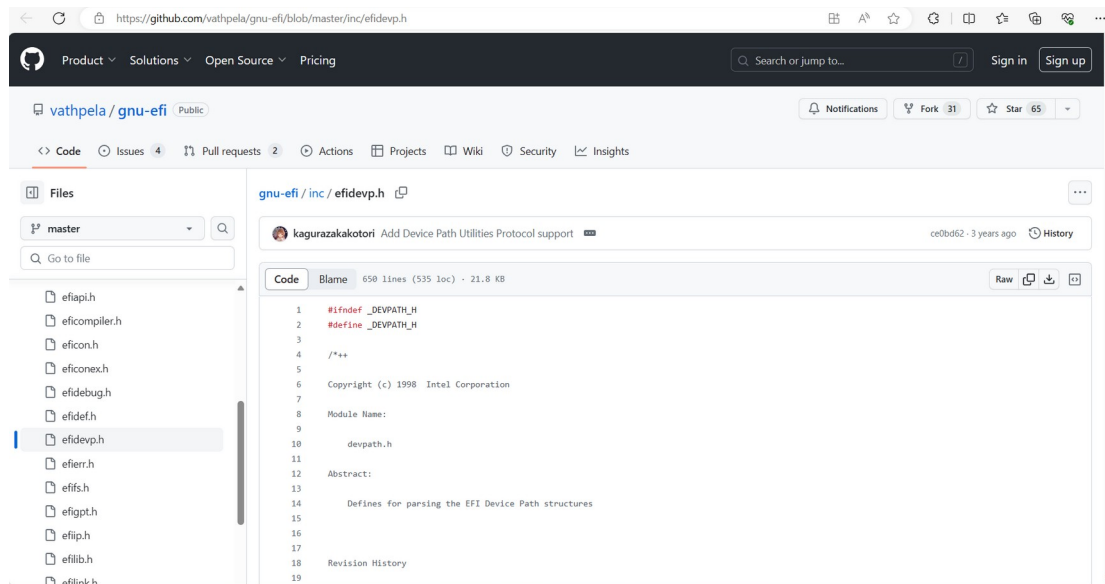


<https://github.com/tianocore/edk2/blob/edk2-stable201903/ShellBinPkg/UefiShell/X64/Shell.efi>

Guardado en la USB dentro de la ubicación marcada:



Con esto ya tenemos lista nuestra USB con EFI-Shell. También puede usar GNU EFI



<https://github.com/vathpela/gnu-efi/blob/master/inc/efidevp.h>

La diferencia es que el usuario debera compilar el codigo fuente y crear su ejecutable BOOTX64.efi por su cuenta.

**Nota:** Modificar o alterar el sistema de arranque de EFI de un sistema de computo puede dañarlo, se da por sentado que el usuario sabe lo que esta haciendo.

## Bibliografia:

*Configuracion de UEFI en sistemas*

*Intel-Basic Instructions for Using the Extensible Firmware Interface (EFI)*

*Intel-Shell Command Reference Manual*

*Intel-Minimal Intel Architecture Boot Loader*

<https://www.tianocore.org/>

*EDKII User Manual*