USO AVANZADO DEL SHELL.

Ingrese a un shell de línea de comandos de su equipo .

Ejecutando más de un comando a la vez
En multitud de situaciones no vemos en la necesidad de ejecutar más de un comando en una sola línea. Para ejecutar más de un comando a la vez se utilizan grupos de comandos donde separa cada comando con el signo **;**.

Ejemplo: **comando1; comando2; ...; comandoN *<ENTER>***

A esto se le llama un grupo de comandos.

Ingrese los comandos **`cd`** y a continuación <ENTER>.
Ingrese los comandos **`pwd`** y a continuación <ENTER>.
Ingrese los comandos **`cd ..`** y a continuación <ENTER>.
Ingrese los comandos **`pwd`** y a continuación <ENTER>.
Ingrese los comandos **`ls`** y a continuación <ENTER>.
¿Qué información mostró?
Ahora ingrese **`cd; pwd; cd ..; pwd; ls`** y a continuación <ENTER>. ¿Qué haría este comando?¿Qué información mostró?¿Nota como opera el **;**?

Subshells
Un subshell es un proceso hijo del shell actual que ejecuta una o más tareas en un nuevo shell y una vez que termina de ejecutarse regresa el control al shell padre. Los comandos se ejecutan en un nuevo entorno diferente del actual, es decir, no afectan al entorno de ejecución desde el que fueron llamados. Para ejecutar comandos en un subshell se lo encierra entre paréntesis.

Ejemplo: **(comando) *<ENTER>***

Además podemos ejecutar grupos de comandos en un *subshell*.

Ejemplo : **(comando1; comando2; ...; comandoN) *<ENTER>***

Ingrese **`cd; pwd; cd ..; pwd; ls`** y a continuación <ENTER>. ¿Qué información mostró? ¿En qué directorio está parado luego de la ejecución del grupo de comandos?
Regrese a su directorio home. Ahora ingrese **`(cd; pwd; cd ..; pwd; ls)`** y a continuación <ENTER>. ¿Qué información mostró?¿En qué directorio está parado luego de la ejecución del grupo de comandos? ¿Nota la diferencia?

Los *subshells* tienen su propio entorno de ejecución (el entorno inicial es igual al entorno actual). Cuando se ejecuta el comando cd en el subshell, éste cambia el entorno del subshell. Sin embargo no afecta al entorno del shell padre, por esto es que el directorio actual antes y después de ejecutar el subshell es el mismo.

Redirección
Los sistemas operativos **nix* facilitan mucho a los programas el uso del terminal. Cuando un programa escribe algo en la pantalla, está usando algo llamado salida estándar. Salida estándar, en inglés standard output o **stdou**t, es la manera que tiene el programa de mostrarle cosas al usuario. El nombre por el que se indica un programa es entrada estándar (**stdin**). Es posible que un programa se comunique con el usuario sin usar la entrada o salida estándar, pero la mayoría de los comandos usan stdin y stdout.
Por ejemplo, el comando ls imprime una lista de los directorios en la salida estándar, que está normalmente "conectada" al terminal. Un comando interactivo, como el shell, bash, lee los comandos de la entrada estándar.
Un programa también puede escribir en el error estándar, ya que es muy fácil hacer que la salida estándar apunte a cualquier lugar aparte del terminal. El error estándar (**stderr**) está casi siempre conectado al terminal para que alguna persona pueda leer el mensaje.

Mucha de la utilidad de los shell viene de la posibilidad de redirigir la entrada y salida desde y hacia archivos. La mayoría de los shells están diseñados para procesar datos usando flujos (streams). Un flujo es nada menos que un flujo de datos que circula como una entrada o salida.

Por ejemplo, el flujo de entrada de un comando suele provenir del teclado y el flujo de salida usualmente va hacia la pantalla.
Los flujos pueden interconectarse fácilmente o enlazarse de diferentes maneras. Esto se denomina **redirección**.

STDOUT
La mayoría de los programas escriben su salida en la stdout. En muchas ocasiones es extremadamente útil enviar la salida de un comando a otro lugar que no sea la pantalla (stdout). Para redireccionar la salida de un comando se utiliza el operador **>**.

Sintaxis: **comando > DESTINO_DE_LA_SALIDA**

Algo de mucha utilidad es almacenar la salida de un comando en un archivo. Para esto debemos redirigir la salida del comando desde la stdout hacia un archivo.

Ubíquese en su directorio home.
Ingrese el comando **`ls -la /etc`** y a continuación <ENTER>. ¿Por donde salió el listado?
Ahora ingrese el comando **`ls -la /etc > listado`** y a continuación <ENTER>. ¿Qué haría este comando? ¿Vio algo por la pantalla?¿Dónde está el listado? Vea el contenido del archivo con nombre "**`listado`**".
Ahora ingrese el comando **`ls -la /home > listado`** y a continuación <ENTER>. Vea el contenido del archivo listado. ¿Qué ocurrió con el listado de /etc?

Si deseamos que la salida de un comando se agregue al final de un archivo (append) debemos usar el operador **>>**.

Sintaxis: **comando >> DESTINO_DE_LA_SALIDA**

Ingrese el comando **`ls -la /home > listado2`** y a continuación <ENTER>. Vea el contenido del archivo listado2.
Ahora ingrese el comando **`ls -la / >> listado2`** y a continuación <ENTER>. Vea el contenido del archivo listado. ¿Nota la diferencia?

STDIN
La mayoría de los programas leen su entrada de la stdin. En muchas ocasiones es extremadamente útil leer la entrada de un comando desde otro lugar que no sea el teclado (stdin). Para redireccionar la entrada de un comando se utiliza el operador **<**.

Sintaxis: **comando < ORIGEN_DE_LA_ENTRADA**

Ubíquese en su directorio home.
Cree usando un editor de texto un archivo y escriba en este unas cuantas lineas de texto (una palabra en ingles por cada linea) y guárdelo como archivo.txt.

Ingrese el comando **`ispell < archivo.txt`** y a continuación <ENTER>. >. ¿Qué haría este comando? ¿Nota lo que ocurrió?

STDERR
La mayoría de los programas escriben los errores en la stderr. En muchas ocasiones es útil enviar los errores de un comando a otro lugar que no sea la pantalla (stderr). Para redireccionar los errores de un comando se utiliza el operador **2>**.

Sintaxis: **comando 2> DESTINO_DE_LOS_ERRORES**

Algo de mucha utilidad es almacenar los errores de un comando en un archivo. Para esto debemos redirigir los errores del comando desde la stderr hacia un archivo.

Ubíquese en su directorio home. **(Asegúrese de NO estar logueado como root!!!!)**
Ingrese el comando **`ls -R /proc > salida`** y a continuación <ENTER>. ¿Por donde salió el listado? Y esos errores?
Ahora ingrese el comando **`ls -R /proc 2> errores`** y a continuación <ENTER>. ¿Qué haría este comando? ¿Vio algo por la pantalla? Vea el contenido del archivo errores.

En ocasiones no nos interesa ver los errores que genera un comando (porque nos molesta en la pantalla), por esto, quiero descartar los mensajes de error. Para esto debemos redirigir los errores del comando desde la stderr hacia el dispositivo nulo (/dev/null).

Ubíquese en su directorio home.
Ingrese el comando **`ls -R /home`** y a continuación <ENTER>. ¿Por donde salió el listado?¿Aparecieron errores?
Ahora ingrese el comando **`ls -R /home 2> /dev/null`** y a continuación <ENTER>. ¿Vio algo por la pantalla?¿Dónde están los errores?

La redirección de errores es independiente de la salida estándar, con lo cual se puede redireccionar la salida y los errores al mismo tiempo.

Ubíquese en su directorio home.
Ingrese el comando **`ls -R /var/man`** y a continuación <ENTER>. ¿Por donde salió el listado?¿Aparecieron errores?
Ahora ingrese el comando **`ls -R /var/man > salida 2> errores`** y a continuación <ENTER>. ¿Vio algo por la pantalla? ¿Dónde está la salida?¿Dónde están los errores?

Cañerías (Pipes)
Muchos comandos *nix producen gran cantidad de información. Por ejemplo, es normal que un comando como "**`ls /usr/bin`**" produzca más salida que la que se puede ver en pantalla. Para que se pueda ver toda la información de un comando como "**`ls /usr/bin`**", es necesario usar otro comando llamado more.

Sin embargo, eso no ayuda al problema de que "**`ls /usr/bin`**" muestre más información de la que se pueda ver. "**`more < ls /usr/bin`**" no funciona, ¡La redirección de entrada sólo funciona con ficheros, no comandos!

Se podría hacer esto:

```
ls /usr/bin > temp-ls
more temp-ls
rm temp-ls
```

Pero tenemos una forma más limpia de hacerlo.

Sintaxis: **comando1 | comando2**

Se puede usar el comando **`ls /usr/bin | more`**. El carácter **|** es una cañería. Como una cañería de agua, una cañería Unix controla el flujo. En vez de agua, se controla el flujo de información.

Las cañerías permiten usar la salida de un comando como entrada de otro. Se pueden conectar cuantos comandos necesitemos.

Un ejemplo extremo sería: **`ls /bin | grep sh | sort -r | more`**

Ubíquese en su directorio home.
Ingrese el comando **`ls -l /dev`** y a continuación <ENTER
Ahora ingrese el comando **`ls -l /dev | less`** y a continuación <ENTER>. ¿Nota la diferencia?
Ahora ingrese el comando **`who | wc -l`** y a continuación <ENTER>. ¿Qué haría este comando?