



**HFC**

# INTRODUCCIÓN A UNIX

DANNA MÁRQUEZ  
FERNANDO ROMERO





# DIRECTORIOS PRINCIPALES

```
[ookami@lothric] [/dev/pts/0]
[~]> ls / | nl
 1 bin
 2 boot
 3 dev
 4 etc
 5 home
 6 lib
 7 lib64
 8 lost+found
 9 mnt
10 opt
11 proc
12 root
13 run
14 sbin
15 srv
16 sys
17 tmp
18 usr
19 var
```

Generalmente, las distribuciones de *Linux* mantienen la misma estructura de directorios en la carpeta raíz ( / ) con el objetivo de organizar los archivos y generalizar el propósito de los archivos en estos.



Directorio  
raíz de la  
jerarquía  
del sistema  
de ficheros

/

bin/	<i>BINARIOS DE ORDENES DE USUARIO ESENCIALES</i>
boot/	<i>ARCHIVOS ESTATICOS DEL CARGADOR DE AUTOARRANQUE</i>
dev/	<i>ARCHIVOS DE DISPOSITIVOS</i>
etc/	<i>CONFIGURACION DEL SISTEMA DEL HOST</i>
home/	<i>DIRECTORIOS CASA DE LOS USUARIOS</i>
lib/	<i>BIBLIOTECAS COMPARTIDAS Y MODULOS DEL NUCLEO (KERNEL)</i>
media/	<i>PUNTO DE MONTAJE PARA MEDIOS DESMONTABLES</i>
mnt/	<i>PUNTO DE MONTAJE PARA SISTEMAS DE ARCHIVOS TEMPORALMENTE MONTADOS</i>
opt/	<i>PAQUETES DE APLICACIONES DE SOFTWARE COMPLEMENTARIAS</i>
sbin/	<i>BINARIOS DEL SISTEMA</i>
srv/	<i>DATOS PARA SERVICIOS PROPORCIONADOS POR EL SISTEMA</i>
tmp/	<i>FICHEROS TEMPORALES</i>
usr/	<i>UTILIDADES Y APLICACIONES DE USUARIO(S)</i>
var/	<i>ARCHIVOS VARIABLES</i>
root/	<i>DIRECTORIOS CASA DEL USUARIO ADMINISTRADOR</i>
proc/	<i>ESTADO DE LOS PROCESOS E INFORMACION DEL NUCLEO</i>

# Estándar de la jerarquía de ficheros

home/user1

home/user2

usr/bin/

usr/include/

usr/lib/

usr/local/bin/

usr/local/

usr/sbin/

usr/local/games/

usr/share/







# DIRECTORIOS PRINCIPALES

## /etc

Este directorio está pensado para almacenar **archivos de configuración** que se consideran **estrictamente necesarios** para el funcionamiento del sistema.

## /tmp

Este directorio esta pensado para almacenar archivos temporales, que son removidos cada tanto, por ejemplo, al reiniciar la computadora.

## /lib, /lib32, /lib64

Aqui se encuentran las **bibliotecas** de código necesarias para el correcto funcionamiento del sistema.

## /boot

En esta carpeta se representan los contenidos de la partición de arranque, encargados de iniciar el sistema operativo.



# DIRECTORIOS PRINCIPALES

## /proc

Este **no** es un directorio físico o real en el disco duro, si no una especie de **interfaz** que nos ofrece el sistema operativo para obtener información acerca de los **procesos** presentes.

## /var

Idealmente, almacena archivos con información variable, como lo son archivos de *logs*, bases de datos, caché y más archivos temporales.

## /dev

Este directorio almacena todas las **representaciones** de los “**dispositivos**” con los que la computadora interactúa. Desde discos duros, entradas, interfaces de red, etc.

## /mnt

Esta carpeta es un espacio para **montaje**, de modo que podamos interactuar con otros sistemas de archivos como si estuvieran en nuestro propio entorno.





# DIRECTORIOS PRINCIPALES

## /home

Este directorio almacena las carpetas personales de los usuarios, aquellas que funcionan como su entorno de trabajo.

## /root

Esta carpeta funciona como la carpeta de trabajo del usuario **root**.

Podría contener información sensible del sistema y los recursos privilegiados que este posea.

## /usr

En esta carpeta se almacena información no tan vital del sistema pero necesaria.

Por ejemplo, particularmente en **/usr/share**, se instalan todos los archivos que formen parte de los **paquetes** que instalamos por el repositorio



# ¿QUÉ ES UN BINARIO?

## ¿Porqué se ven tan raros?

Estamos acostumbrados a visualizar caracteres legibles como letras, números, símbolos, etc.

Si revisamos el código *ASCII*, estos son incluso menos que la mitad de todos los posibles valores de un *byte* ( $< 128$ ).

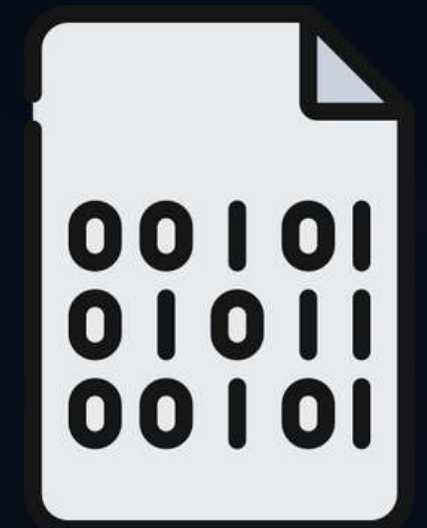
Un binario no tiene esta restricción pues necesita toda la capacidad que pueda para expresar las instrucciones al procesador.

Un binario es un archivo **ejecutable** por nuestra computadora, pues está compuesto por **código máquina o binario**.

En *Linux* estos archivos son conocidos como **ELF executable** y ofrecen una manera ligera y fácil de ejecutar programas.

## ¿Porqué nos interesan?

En realidad, los comandos que ejecutamos en la terminal son archivos binarios ejecutables que están en alguna carpeta de nuestra computadora.







# DIRECTORIOS PRINCIPALES

## /bin

En esta carpeta se encuentran binarios completamente necesarios para la interacción del sistema con cualquier usuario.

## /usr/bin

En este directorio se almacenan binarios **no esenciales** para el funcionamiento o interacción con el sistema, típicamente al instalar algún paquete del repositorio.

## /sbin

En esta carpeta se encuentran binarios enfocados a la **administración del sistema**, típicamente enfocada a usuarios privilegiados.

## /usr/local/bin

Esta carpeta se nos ofrece como una alternativa para guardar binarios en el sistema que poseamos por cualquier alternativa distinta a métodos más “oficiales”.



# PREGUNTAS



## ¿Qué son?

Las **banderas** son “opciones” en los comandos que modifican el comportamiento de este, sin perder totalmente de vista el objetivo original del comando.

Estas banderas se indican con uno o 2 guiones al inicio, usualmente de la forma **-f** o como **--flag**.


De esta manera, cuando se identifican los guiones el comando trata ese argumento como una **bandera**. Además, pueden o no recibir un argumento propio

# BANDERAS

## Ejemplos

- ls -l
- ls -a
- rm -r
- zip -r
- ncat -lnvp 80
- tar --help

--delete vs -D  
--version vs -v



Precisamente tanto en el manual (**man**) como con la bandera **--help** o **-h** se nos despliega información acerca de las distintas banderas que el comando soporta, las más utilizadas, sus funciones, parámetros y demás.



# OPERADORES ESPECIALES



- Se usa para ejecutar un comando anterior de manera específica.
  - !ls: Ejecuta el comando ls más reciente en el historial.
  - !n: Ejecuta el comando

## !!

- Ejecuta el último comando utilizado en la terminal.
  - sudo !! repite el último comando con permisos de superusuario.

## ;

- Permite ejecutar varios comandos secuencialmente, independientemente de si el anterior tuvo éxito o no.
  - echo "Inicio"; mkdir nueva\_carpeta; cd nueva\_carpeta; echo "Listo"

## &&

- Ejecuta el siguiente comando solo si el comando anterior tuvo éxito (código de salida 0).

## ||

- Ejecuta el siguiente comando solo si el anterior falló (código de salida distinto de 0).
  - mkdir datos || echo "La carpeta ya existe"

## \$()

- Se usa para ejecutar un comando dentro de otro y capturar su salida.
  - echo "Hoy es \$(date)": Sustituye \$(date) con la salida del comando date.

## ``` (comillas invertidas)

- Similar a \$(), se usa para ejecutar comandos y capturar su salida.
  - echo "Hoy es `date`": Sustituye `date` con la salida del comando date.



# ENTRADA ESTÁNDAR, SALIDA ESTÁNDAR Y SALIDA DE ERRORES



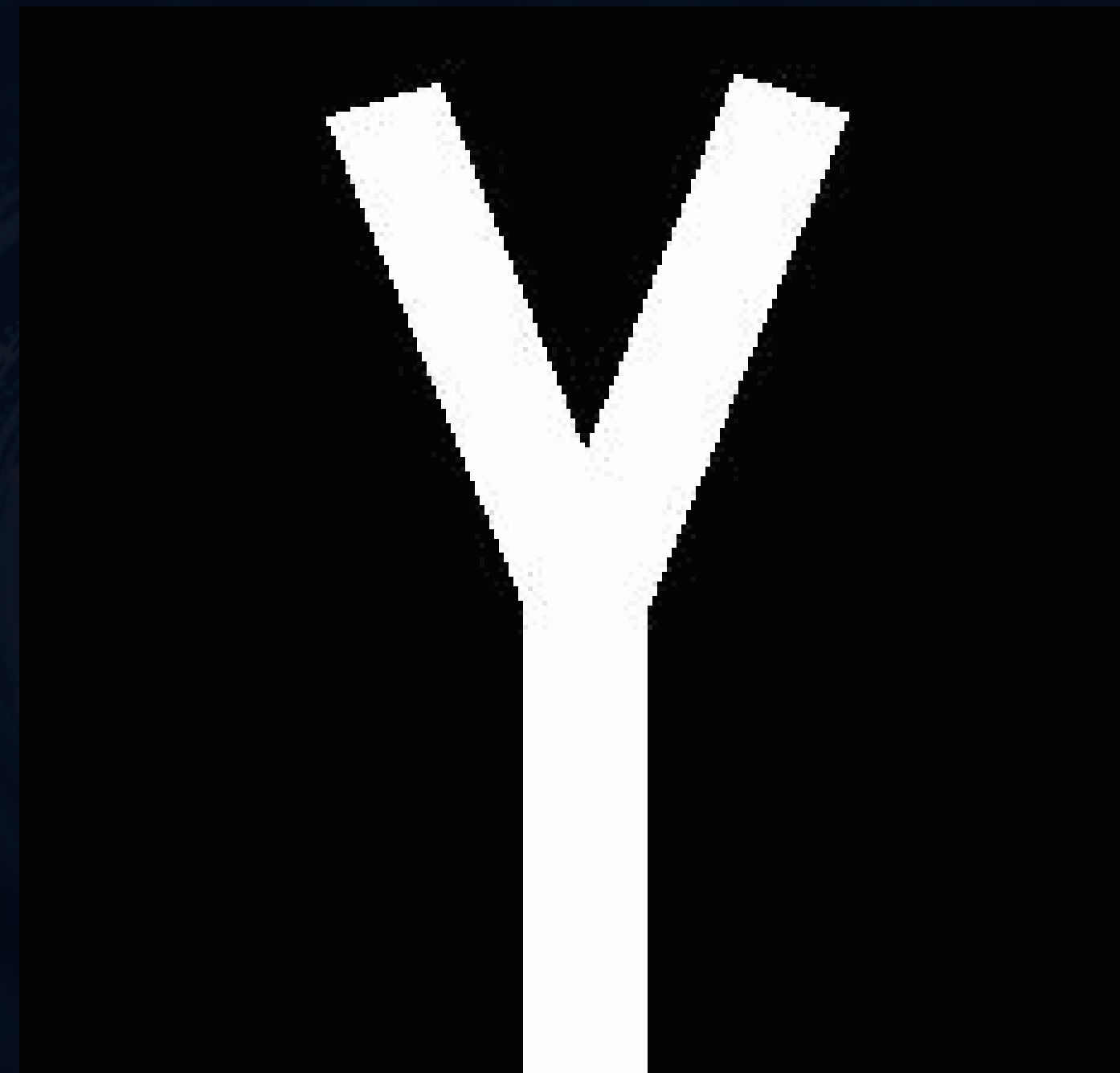
La **entrada estándar** se refiere a la manera convencional que tenemos de interactuar con la terminal. Donde simplemente escribimos en la **línea de comandos**.

La **salida estándar**, de igual manera, es la misma ventana de la terminal donde imprime los resultados a nuestras instrucciones

La **salida de error**, aunque visualmente idéntica pues también se imprime en la ventana, se refiere al canal donde se envían todos aquellos mensajes de error, idealmente.

Salida estándar

Salida de error



Pantalla



# REDIRECCIÓN DE SALIDA Y ENTRADA →

**>**: Redirige la **salida estándar** de un comando a un archivo y si ya existe lo **sobreescribe**. No desvía errores

**>>**: Lo mismo que el anterior pero no sobreescribe, lo añade.

**2>**: Redirige errores del comando (salida de error) y los guarda en un archivo

**2>>**: Lo mismo pero no sobreescribe, añade.

**<**: Redirige el **contenido** de un archivo a la **entrada estándar**, como si lo hubieramos tecleado. (No es totalmente compatible con todos los comandos)

**<< EOF**: Permite escribir **interactivamente** varias líneas redirigiéndolas a entrada estándar hasta encontrar un delimitador (en este ejemplo EOF).

**<<<**: Redirige una cadena a la **entrada estándar**.

**| (Pipe)**: Encadena la **salida estándar** de un comando a la **entrada estándar** de otro comando.

**tee**: Básicamente redirige la salida estándar tanto a la pantalla como a un archivo.

**n>&m**: Redirige **n** (ya sea 1 o 2 para salida estándar o de error) a m.

**&>**: Redirige ambas salidas al mismo archivo



# MÁS COMANDOS =)

- **gzip/bzip** Comprimen archivos, generando .bz2 y .gz respectivamente.
- **tar** Crea y extrae archivos .tar
- **xxd**: Convierte archivos a formato hexadecimal y viceversa.
  - **Ejemplo:**
  - xxd archivo Muestra el contenido del archivo en formato hexadecimal.
  - xxd -r archivo.hex: Convierte un archivo hexadecimal de vuelta al formato original.
- **base64 / base32**: Codifican y decodifican datos en formato Base64 o Base32.
  - **Ejemplo:**
  - base64 -d archivo
- **locate**: Busca archivos rápidamente utilizando una base de datos preconstruida.
  - **Ejemplo:**
  - locate archivo: Encuentra archivos con "archivo" en el nombre.
- **find**: Busca archivos en tiempo real según criterios específicos.
  - **Ejemplo:**
  - find /ruta -name "archivo": Busca archivos con nombre "archivo" en la ruta especificada.
  - find . -size +10M: Encuentra archivos mayores a 10 MB en el directorio actual





# MÁS COMANDOS =)



- **nl:** Numera las líneas de un archivo.
- **head:** Muestra las primeras líneas de un archivo.
- **tail:** Muestra las últimas líneas de un archivo.
  - `tail -f` permite ver archivos en tiempo real, útil para logs.
- **strings:** extrae y muestra solo las cadenas de texto legibles de archivos binarios.
- **grep:** Busca texto dentro de archivos.
- **cut:** extrae columnas o partes específicas de un archivo o entrada de texto.
- **tr:** reemplaza, elimina o transforma caracteres de una entrada de texto.
- **sed:** permite buscar, reemplazar y manipular texto de manera avanzada



# EJERCICIO

