



## Escuela Politécnica Superior de Elche Ingeniería de Telecomunicaciones

# Fundamentos de Sistemas Operativos Cuaderno de practicas

### Introducción al Shell.

Objetivos.....	1
Desarrollo .....	1
1. Conexión al servidor Linux .....	1
2. Salida del sistema .....	2
3. Cambio de contraseña.....	2
4. Teclas útiles .....	2
5. Uso de comandos básicos .....	3
6. Comandos relacionados con ficheros y directorios .....	3
7. Navegar por el árbol de directorios .....	4
8. Editor de textos.....	5
9. Uso de comandos básicos .....	5
10. Comandos relacionados con propiedad y protección .....	6
11. Enlaces.....	7
12. Redirección E/S .....	9
13. Filtros y tuberías .....	11
14. Gestión de procesos .....	12
15. Metacaracteres .....	15
16. Comillas.....	16
17. Terminadores de órdenes.....	18
18. Agrupación de órdenes: .....	18
19. Comando find y sus opciones.....	20
20. Comunicación entre usuarios.....	21

## Objetivos

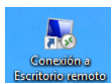
El objetivo de este cuaderno es familiarizarse con el Shell:

- Acceder al sistema y salir del mismo.
- Manejo básico de comandos Linux.

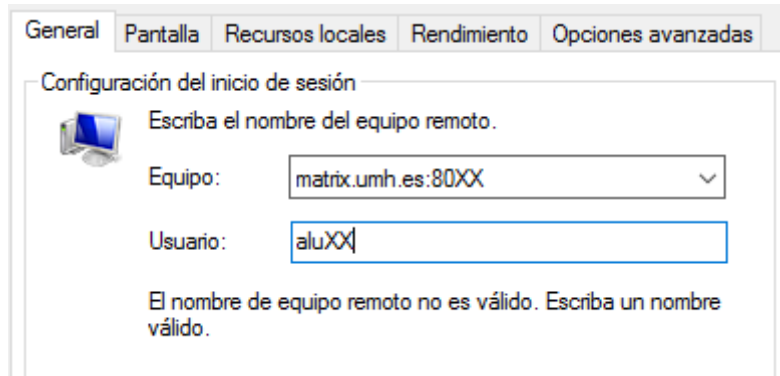
## Desarrollo

Realizaremos los siguientes pasos.

### 1. Conexión al servidor Linux



Utilizaremos la herramienta rdp para conexión a escritorio remoto que configuraremos con los siguientes parámetros como se muestra en la imagen.



Esto conectará con el servidor linux debian jessie que nos pedirá un password que es proporcionado a cada alumno por el profesor.

Podemos guardar la configuración de la conexión.

## 2. Salida del sistema

Para salir del sistema tenemos varias posibilidades:

- **logout**
- **exit** (equivalente a CTRL+D)

## 3. Cambio de contraseña

A todos los alumnos de la asignatura se les ha dotado de un login y un password.

Los alumnos podrán cambiar el password si lo desean.

Para ello utilizaremos el comando **passwd**.

Este comando nos pide nuestra contraseña actual y luego la nueva contraseña, la cual tendremos que escribir dos veces por seguridad, ya que, siempre que escribimos contraseñas no vemos lo que estamos escribiendo.

La nueva contraseña no debe ser muy simple. Al menos debe contener 6 u 8 caracteres, conteniendo algún carácter especial o algún número. Si olvidásemos nuestra contraseña, deberemos ponernos en contacto con el profesor para que la modifique.

NOTA: Es posible que si todos los alumnos se ponen a modificar su contraseña al mismo tiempo aparezca un error. Si sucede esta saturación, volver a intentarlo más tarde.

## 4. Teclas útiles

Combinación de teclas	Uso
CTRL+D	Señal de fin de texto o fin de sesión (igual que exit o logout)
CTRL+S	Realizar una pausa en la salida de la pantalla
CTRL+Q	Continuar la salida por pantalla pausada con CTRL+S
CTRL+C	Interrupción de la orden actual
CTRL+H	Borra el carácter de la izquierda (igual que Backspace)

Tecleamos hola y pulsamos return  
 Queremos con esto que el shell ejecute el comando o inicie la ejecución del programa  
 hola  
 ¿Que obtenemos? ¿Porque?

Cual es mi directorio de trabajo, ¿Dónde estoy?. Con **pwd** se obtiene.  
 /home/alu01 si soy el alumno 01

Con ls veo el contenido de mi directorio de trabajo, inicialmente vacio.

## 5. Uso de comandos básicos

A continuación vamos a ver algunos de los comandos más utilizados. En algunos de los comandos podremos especificar varias opciones, precediéndolas con un guión. Por ejemplo: `ls -l -F`. Esta orden también se podrá escribir de forma abreviada: `ls -lF`

**echo:** Muestra en la salida estándar la cadena. Sintaxis:

**echo [opciones] cadena**

La opción `-n` evita la nueva línea

**echo estoy en clase de practicas**

**echo -n estoy en clase de practicas**

Comando	Uso
<b>man</b> <i>comando</i>	Nos proporciona información del uso del comando. Para salir de las páginas de man pulsar la tecla 'Q'
<b>man</b> <i>ascii</i>	Documentación de la tabla ASCII. Util para saber que número tiene cada carácter o símbolo.
<b>date</b> / <b>cal</b>	Obtener la fecha y hora actual / Calendario mes actual
<b>w</b>	Nos informa de quien está conectado al sistema ahora y que está haciendo
<b>who</b>	Nos informa de quien está conectado al sistema
<b>who am i</b>	Nos informa de la identidad de nuestro terminal
<b>whoami</b>	Nos informa de quien somos nosotros (login)
<b>echo</b> <i>cadena1</i> <i>cadena2</i> ...	Muestra por pantalla las cadenas de texto especificadas
<b>clear</b>	Borrar la pantalla
<b>wc</b> [-l]	Contar líneas, palabras y letras de su entrada (comando). Para terminar la entrada hacerlo en una nueva línea pulsando CTRL+D

## 6. Comandos relacionados con ficheros y directorios

Comando	Uso
<b>pwd</b>	Nos muestra el directorio de trabajo actual
<b>cd</b>	Nos lleva de vuelta a nuestro directorio HOME
<b>cd ..</b> / <b>cd</b> <i>dir</i>	Nos lleva al directorio padre o al <i>dir</i> especificado
<b>mkdir</b>	Crear un directorio

- Crear un directorio llamado *shell* en nuestro home. Notar que debe de estar escrito en minúsculas.
- Crear un directorio llamado *trash* en nuestro home, esta será nuestra papelera de reciclaje. Notar que debe de estar escrito en minúsculas.
- Crear un directorio cualquiera dentro del directorio shell y borrarlo (ver siguiente tabla)

<b>rmdir</b>	Borrar un directorio (si está vacío de ficheros y directorios)
<b>ls</b>	Listar el contenido de un directorio: -a: (all) todos los archivos, incluso los ocultos (. , .. y otros) -l: (long) listado largo, mostrando detalles (fecha/hora, permisos, propietario, etc.). -F: listado para saber si son archivos o directorios. Además marca con un '*' los ficheros ejecutables -d: (directory) sólo directorios -R: (recursive) listado recursivo -i: presenta los i-nodes (números de archivo) de los archivos

## 7. Navegar por el árbol de directorios

Comienza siempre por el directorio raíz "/", del cual cuelgan directamente otros directorios, cada uno de los cuales tiene un uso distinto:

/bin: binarios o ejecutables (comandos del sistema)  
 /dev: controladores de dispositivo (ficheros especiales):  
     mouse, ttyS0 (cua: modem), lp0, fd, hda, sda, stderr, stdin, stdout,  
     tty1 (consolas virtuales), null (agujero negro) ...  
 /etc: ficheros de configuración del sistema (/etc/passwd, /etc/rc => guiones)  
 /sbin: almacenar programas esenciales del sistema (fdisk, mkfs, etc.)  
 /home: directorios personales de los usuarios (después del Login)  
 /lib: Librerías compartidas (ejecutables más pequeños y más espacio)  
 /proc: sistema de ficheros virtual (son procesos en memoria)  
 /tmp: para archivos temporales  
 /usr: programas de usuario y ficheros de configuración  
     /usr/X386: sistema X Window  
     /usr/bin: donde se instalan los programas  
     /usr/etc: ficheros de configuración (no esenciales para el sistema)  
     /usr/include: ficheros de cabecera para el compilador C  
     /usr/g++-include: ficheros de cabecera para el compilador C++  
     /usr/lib: librerías de las aplicaciones  
     /usr/local: programas del sistema (no esenciales)  
     /usr/man: manuales: páginas de ayuda de los comandos  
     /usr/src: código fuente de diversas utilidades  
 /var: directorios de tamaño variable (que cambian mucho de tamaño)  
     /var/adm: históricos del sistema

/var/spool: zona de spool, para archivos que van a ser enviados o se acaban de recibir (correo)

/mnt: donde normalmente montamos sistemas de archivos (mount).

## 8. Editor de textos


Utilizaremos el editor de textos Vim o nano desde consola o gedit u otro cualquiera desde el entorno gráfico.

Vim es el nombre del editor de textos. Ver documentación entregada del editor vim.

Crear un archivo de texto con una lista de nombres y apellidos separados por tabulador. Guardarlo con el nombre de fichero 'nombres'

## 9. Uso de comandos básicos

NOTA: en todos estos comandos podremos utilizar caracteres máscara o comodines:

Comodín	Significado
*	Representa cualquier cadena de caracteres, de cualquier longitud, incluyendo la cadena vacía.
?	Representa a cualquier carácter (uno).
cp	Copiar archivos. <ul style="list-style-type: none"> <li>Copiar el archivo <i>nombres</i> con otro nombre.</li> </ul>
mv	Renombrar y mover archivos. <ul style="list-style-type: none"> <li>Mover el archivo recién creado</li> </ul>
rm [-i] [-f] [-r]	Eliminar (remove) archivos. <p>Con -i nos pide confirmación previa.</p> <p>Con -f fuerza el borrado sin confirmación, incluso para los ficheros protegidos contra escritura.</p> <p>Con -r eliminamos un directorio, así como todos sus ficheros y subdirectorios.</p>
	
cat	Visualizar el contenido de un archivo <ul style="list-style-type: none"> <li>Copiar el archivo nombre con otro nombre.</li> <li>Editarlo con Vi para copiar su contenido y pegarlo varias veces, de forma que tengamos que realizar scroll para ver su contenido, es decir que no quepa todo en una pantalla.</li> </ul>
pg <i>file</i> pg -n <i>file</i> pg +n <i>file</i>	Visualizar el contenido de un archivo por páginas (pulsando [Intro]. Pulsando [q]+[Intro] terminamos. Cuando especificamos -n cambiamos el número de líneas por página. Cuando especificamos +n indicamos que la presentación del fichero comienza en el número de línea especificado. <ul style="list-style-type: none"> <li>Utilizar el comando pg con sus variantes con el archivo creado anteriormente.</li> </ul>
head <i>file</i> head -n <i>file</i> head +n <i>file</i>	Visualizar las primeras 10 líneas del fichero. Cuando especificamos -n cambiamos el número de líneas que vamos a mostrar. Cuando especificamos +n indicamos el número de línea desde el que comenzamos. <ul style="list-style-type: none"> <li>Utilizar el comando head con sus variantes con el archivo</li> </ul>

	creado anteriormente.
<b>tail</b> <i>file</i> <b>tail -n</b> <i>file</i> <b>tail +n</b> <i>file</i>	Visualizar las últimas 10 líneas del fichero. Cuando especificamos -n cambiamos el número de líneas que vamos a mostrar. Cuando especificamos +n indicamos el número de línea desde el que comenzamos. <ul style="list-style-type: none"> <li>Utilizar el comando tail con sus variantes con el archivo creado anteriormente.</li> </ul>
<b>more</b>	Visualizar el contenido de un archivo por líneas (pulsando [Intro]) o por páginas (pulsando [Espacio]). Pulsando [q] termina. <ul style="list-style-type: none"> <li>Utilizar el comando pg con sus variantes con el archivo creado anteriormente.</li> </ul>
<b>grep</b> [opciones] <expresion_a_buscar> <lista_archivos>	Buscar una cadena en los archivos especificados. Nos muestra las líneas en las que encuentra dicha cadena. -c (count) sólo cuenta las líneas, no las visualiza. -i ignora mayúsculas/minúsculas. -l lista los ficheros con alguna ocurrencia. -n visualiza el número de línea. -v: se muestran las líneas donde no esta <expresion_a_buscar> <ul style="list-style-type: none"> <li>Utilizar el comando grep con sus opciones con el archivo creado anteriormente</li> </ul>
<b>sort</b> [-opc] [+columna]	Ordena su entrada (teclado) por orden alfabético en su salida (pantalla). -b ignora espacios iniciales. -f ignora may/míns. -n, los números se ordenan por su valor. -r se invierte el orden (descendente).

Probar los comandos anteriores.

## 10. Comandos relacionados con propiedad y protección

Comando	Uso
<b>chown</b> <i>usuario fichero</i>	Cambia el propietario de un fichero o directorio
<b>chgrp</b> <i>grupo fichero</i>	Cambia el grupo al que pertenece un fichero
<b>chmod</b> <i>modo fichero</i> <b>chmod</b> {a,u,g,o}{+,-} {r,w,x} fichero (all,user,group,other)	Cambia los permisos de acceso de un fichero. Cuando hacemos ls -l podremos verlos en la primera columna. Son 10 letras o guiones que representan: <b>drwxrwxrwx</b> d: sólo aparece si es un directorio s: sólo aparece si es un socket rwx: permisos de lectura, escritura y ejecución del usuario rwx: permisos de lectura, escritura y ejecución del grupo rwx: permisos de lectura, escritura y ejecución del resto Cuando en cualquiera de ellos aparece un guión "-"

Si hacemos **ls -l** obtenemos un listado largo del contenido del directorio.

Si empezamos de izquierda a derecha a analizar el listado encontramos:

- Tipo de archivo (directorio [d], enlace [l], archivo normal [-], socket [s], etc...)
- -rw-r--r- Permisos del fichero
- Número de enlaces duros al fichero (hard links)
- Propietario del fichero
- Grupo del fichero
- Tamaño en bytes del fichero
- Fecha y hora de la última modificación
- ... y por fin el Nombre del fichero.

## Permisos de ficheros

Cada fichero tiene ciertos permisos asociados a él, que indican al sistema operativo quién puede acceder a ese fichero, cambiarlo o en el caso de ser un programa, ejecutarlo. Cada uno de esos permisos puede ser establecido por separado para el dueño del fichero, para el grupo al que pertenece el dueño del fichero y para el resto de usuarios del sistema.

El modo de cambiar los permisos de un fichero es

**chmod [a,u,g,o] [+,-] [r,w,x]**

**chmod o+w documento** da a todos los usuarios permisos para escribir, borrar, etc el documento.

Con **chmod o-rw documento** quitamos los permisos de lectura y escritura para el resto de los usuarios.

## 11.Enlaces

<b>ln</b> <i>f1 f2</i>	Crear un enlace duro
<b>ln -s</b> <i>f1 f2</i>	Crear un enlace simbólico o suave

Los sistemas de ficheros de tipo Unix permiten crear enlaces entre ficheros. Los enlaces pueden ser de dos tipos, duros y simbólicos (hard links, symbolic link también llamados hard link y soft link).

El primer caso consiste en asignar más de un nombre a los mismos datos en disco. Este nombre aparece como un fichero nuevo al hacer **ls**, e incluso puede residir en el mismo o en otro directorio, pero los datos son los mismos, simplemente el fichero con el nuevo nombre está apuntando a los mismos datos en disco que el fichero original. Por tanto no consume más espacio en disco, excepto el necesario en la tabla de i-nodes (un i-node es el identificador físico de los datos de un fichero en disco). Este tipo de enlace (hard link) solo es válido para ficheros que estén en el mismo sistema de ficheros (partición).

Los enlaces simbólicos son ficheros que apuntan a otro fichero o directorio, en el mismo **o en otro sistema de ficheros**. La diferencia, a parte de poder referenciar fuera del sistema de ficheros actual, es que éstos crean un pequeño fichero en disco que almacena la dirección del fichero apuntado. Por tanto utilizan espacio en disco. La dirección esta

formada por referencias al sistema de ficheros y a la ruta donde se encuentra el fichero apuntado. Los datos del fichero no se copian de nuevo al crear el link simbólico, solo se referencian.

## Hard links

Usamos el comando **ln** para crear hard links.

Crear un fichero llamado mesa con cierto contenido. Ahora si usamos **ls -li** vemos el número de inodo para nuestro fichero mesa.

Ahora vamos a crear un enlace llamado silla: **ln mesa silla** . Con **ls -li** vemos que los dos ficheros tienen el mismo inodo. Vemos que los dos ficheros tienen el mismo i-node, por lo que están apuntando al mismo bloque de datos (fichero) en disco.

Si borramos un fichero con **rm**, solamente se borrará el enlace al fichero.

**rm mesa** no borra los datos en disco (el fichero), solo elimina el fichero de la tabla de ficheros que apuntan a i-nodes.

Veamos esto con un poco más de calma. Cuando un fichero se crean, se almacenan sus datos en el disco y se le asigna un i-node, luego se registra el nombre del fichero en una tabla de ficheros. En esta tabla están (a parte de más cosas) el nombre del fichero y el puntero a sus datos.

Ahora cuando se crea un hard link, solo se da de alta otro fichero en dicha tabla apuntando al mismo i-node o bloque de datos, a los datos del fichero. Vemos ahora que si borramos uno de los dos ficheros con **rm**, sólo lo estamos eliminando de la tabla de ficheros, no estamos eliminando el fichero en si (los datos, vaya).

Será el último **rm** que se realice sobre el fichero el que elimine realmente los datos del disco. (y el nombre de la tabla, claro).

## Soft Links

El comando **ln -s mesa silla** crea un enlace simbólico silla que apunta a mesa. Si realizamos un **ls -li** vemos que el fichero silla es un enlace simbólico y además el primer carácter del grupo de permisos es 'l' que indica este hecho. Vemos que todos los permisos aparecen activados. Esto no significa que se pueda hacer todo por todos sobre el enlace simbólico, pues realmente los permisos que se tienen en cuenta son los del fichero apuntado.

Funcionalmente, los enlaces duros y simbólicos son similares, pero hay algunas diferencias. Por una parte, puede crear un enlace simbólico a un fichero que no está en el mismo dispositivo de almacenamiento, por ejemplo a un fichero en un cd-rom, cosa que no es cierta con un hard link. Los enlaces simbólicos son procesados por el núcleo del sistema operativo de forma diferente a los duros, lo cual es solo una diferencia técnica, pero a veces es importante. Los enlaces simbólicos son de ayuda puesto que identifican al fichero al que apunta; con enlaces duros no es tan fácil saber que fichero está enlazado al mismo inodo.



Los enlaces se usan en muchas partes del sistema operativo. Los enlaces simbólicos son especialmente importantes para las imágenes de las librerías compartidas en /lib.

## 12.Redirección E/S

En los sistemas UNIX tenemos 3 ficheros estándar:

fd	Fichero	Dispositivo	Descripción
0	/dev/stdin	Teclado	Entrada estándar
1	/dev/stdout	Pantalla	Salida estándar
2	/dev/stderr	Pantalla	Salida estándar para los errores

Podemos redirigir la E/S de un programa para que no utilice el dispositivo asociado por defecto. Por ejemplo, podemos utilizar un fichero como entrada o como salida del programa.

>	Redireccionamiento de la salida estándar
>>	Redirección de la salida añadiendo
<	Redirección de la entrada estándar
<<carácter	Asume entrada por teclado hasta introducir el carácter especificado.
2>	Redirección de la salida de error.
	Interconexión, a través de tubería o pipeline. La salida de la orden a la izquierda, es la entrada de la siguiente.
*	Sustituye a cualquier cadena
/?	Sustituye cualquier carácter
[cadena]	Sustituye la cadena indicada a través de intervalo ([1-3], 0 bien por un conjunto de caracteres ([123]).
;	Ejecución secuencial de órdenes.
&	Ejecución paralela u desatendida (background)
\	Anula interpretación de un carácter especial
`_...`	Fuerza la ejecución de ordenes de shell.
#	Introduce comentarios.

### Redirección de la entrada estándar:

Se utiliza para tomar un fichero existente como si fuera el teclado.

El nombre del fichero debe ir precedido del carácter "<".

- Crear, como se indica a continuación, un fichero de texto "nombres2" con una serie de nombres, uno en cada línea. Para ello podemos utilizar cualquier editor de textos (vi, joe, mcedit) o bien, con lo siguiente:

```
cp /dev/stdin nombres [Intro]
<Teclamos los nombres separados por [Intro]>
CTRL+D para finalizar.
```

Con esto hemos redireccionado la entrada estándar del comando cp al dispositivo estándar /dev/stdin (la consola, el teclado), sustituyendo la forma de operar por defecto del comando cp que coge como entrada el primer parámetro que se le pasa.

Ahora podemos ordenarlo. La siguiente orden nos muestra por pantalla el fichero ordenado alfabéticamente:

```
sort < nombres
```

Con esto hemos diseccionado la entrada estandar del comanto sort al fichero nombres, es decir, ahora coge los datos a ordenar desde el fichero nombres.

### Redirección de la salida estándar:

Si el argumento de una orden es un nombre de fichero precedido del carácter ">" o ">>", la salida de esa orden se dirige en el fichero especificado en lugar de a la pantalla. En el caso de ir precedido de ">" se trata de una salida destructiva, ya que si el fichero no existe lo crea, y si ya existe lo borra y lo vuelve a crear; en el caso de ">>" la salida es no destructiva, ya que si el fichero no existe lo crea igualmente, pero si ya existe, la salida de la orden se concatena al final del fichero.

Por ejemplo:

```
ls -l > listado      (y después...)
ls /usr/incluye/*.h >> listado
```

Otro ejemplo:

```
sort > nombresOrdenados
```

Escribimos por teclado una serie de nombres, pulsando [Intro] al final, y cuando terminemos pulsaremos CTRL+D e [Intro], y se nos genera un fichero "nombresOrdenados" con los nombres que hemos teclado ordenados alfabéticamente.

### Redirección de la salida de errores:

Para poder redirigir la salida de errores debemos poner su número de fd: 2>. Si no quisiéramos ver los mensajes de error en pantalla y tampoco redirigirlos a un fichero podemos utilizar el dispositivo null: orden 2> /dev/null.

Es posible redirigir la entrada y la salida simultáneamente. En este caso, debemos tener la precaución de no especificar el mismo nombre en ambos casos. Si esto ocurre, sólo se considera el fichero de entrada.

Por ejemplo:

```
sort < nombres > nombresOrdenados
```

- Ejecutar la orden: **lzcab**, ¿Qué pasa, que nos dice el shell?
- Ejecutar ahora: **lzcab 2> salida** ¿Qué pasa, que nos dice el shell, donde está la salida?
- Ejecutar **cat salida** ¿Está dando un error la orden cat? ¿Qué estamos haciendo realmente?

El número 2 hace referencia al error estándar que redireccionamos al fichero "salida".

Si comprobamos el contenido de “salida” veremos que contiene el mensaje de error

### 13.Filtros y tuberías

Los **filtros** son comandos que reciben una información, la procesan y generan una salida. Los filtros típicos son los comandos: sort, more, grep, etc. Su principal uso es con las tuberías, para realizar operaciones intermedias.

Las **tuberías** consisten en lo siguiente. La salida de una orden puede ser la entrada de otra orden. Cuando hacemos esto decimos que hemos hecho una tubería y utilizamos el carácter "|" (ALT+124).

Ejemplos:

```
ls /usr/include/*.h | more      (visualizamos por páginas)
ls /usr/include/*.h | sort -r   (ordenamos un listado en orden inverso)
who | wc -l                    (contar las líneas que tiene la salida del comando who)
```

Podemos "entubar" más de un comando:

```
ls | sort -r | head 10        (los primeros 10 de la lista)
```

#### Desdoblamiento de la salida: el comando tee

Hemos hablado de redirigir la salida a un fichero. Pero al hacer esto, no vemos por pantalla la salida. Si nos interesa ver la salida por pantalla además de almacenarla en un fichero, tenemos que utilizar el comando **tee** y una tubería.

```
comando | tee [-a] [-i] fichero
```

Con -a (append) añadimos al fichero, en vez de sobrescribirlo.

Con -i (ignore) ignoramos las señales de interrupción (CTRL+C).

Ejemplos:

```
sort nombres | tee nombresOrdenados
```

En este ejemplo, veremos la salida del comando sort por pantalla y se almacenará en un fichero "nombresOrdenados".

```
ls -l | tee listado
```

Aquí visualizamos por pantalla el listado y además, se almacena en el fichero "listado".

**cut:** Filtra un fichero de manera que solo se deja pasar una columna especificada en la línea de órdenes. Sintaxis:

```
cut [opciones] <lista_ficheros>
```

Las opciones de la orden se pueden obtener mediante **man cut**

Para ilustrar la orden utilizaremos la opción "c" que nos muestra el número de caracteres que se especifique.

Mostrar por pantalla el contenido del fichero usuarios y comparar con lo que se obtiene al ejecutar las siguientes órdenes:

```
cut -c1 usuarios
cut -c1-3 usuarios
cut -c1-6 usuarios
```

Ejecutar: **who** | **wc -l** ¿Qué hace?

Ejecutar:

**who** > **usuarios**

**grep fsoN < usuarios** (N es el numero de tu usuario)

¿En qué fichero busca la palabra sopN?

¿Qué fichero utiliza como salida?

Ejecutar: **grep sopN <usuarios |cut -c1-6** ¿Qué da como resultado?. Guarda el resultado en un fichero

Escribe la orden que localiza todas las líneas del fichero nombres que tengan la palabra JUAN y dime el número total de líneas

**grep sopN <usuarios | tee intermedio |cut -c1-6** Comprueba que contiene el fichero "intermedio"

**sort <usuarios | tee ordena | wc -l > numero\_lineas**

¿Qué contiene el fichero "ordena" y qué contiene el fichero "numero\_lineas"?

## 14.Gestión de procesos

---

### Conceptos de primer plano y segundo plano.

Un proceso puede estar en primer plano (foreground) o en segundo plano (background). Sólo puede haber un proceso en primer plano al mismo tiempo en un terminal o sesión, y es el que está interactuando con el usuario, recibiendo órdenes o entrada de datos desde el teclado y mostrando resultados o salida de datos por pantalla.

El proceso que está en segundo plano no recibe ninguna señal u orden desde el teclado. Algunos programas necesitan mucho tiempo para su ejecución y si no proporcionan salida alguna no tenemos porqué estar esperando a su finalización mirando la pantalla, a ver cuando acaba... En estos casos es mejor lanzarlos en segundo plano mediante el operador &, donde se ejecutarán en "*silencio*" y dejarán el terminal o sesión disponible para interactuar con el usuario.

Los procesos pueden ser suspendidos. Un proceso suspendido es aquel que no se está ejecutando actualmente, sino que está temporalmente parado. Después de suspender un proceso, puede indicar a la misma que continúe en primer plano o en segundo plano. Retomar una tarea suspendida no cambia en nada el estado de la misma, la tarea continuará ejecutándose justo donde lo dejó.

Hay que tener en cuenta que suspender un proceso no es lo mismo que interrumpirlo. Cuando se interrumpe un proceso (generalmente con la pulsación de CTRL+C), el proceso muere, termina, deja de estar en memoria y de utilizar recursos del ordenador. Una vez eliminado el proceso no puede continuar ejecutándose y deberá ser lanzado

otra vez para volver a realizar sus tareas, empezando otra vez por el principio, no por donde se interrumpió.

La diferencia entre un proceso suspendido y un proceso en segundo plano es que el suspendido no se está ejecutando (no utiliza la CPU, y puede estar en memoria principal o volcado a disco), mientras que el proceso en segundo plano se está ejecutando y utilizando memoria principal hasta que termine, incluso puede escribir en la pantalla de la consola actual.

Se puede dar el caso de que los programas capturen la interrupción provocada por la tecla CTRL+C y no respondan a la interrupción de manera que continúan ejecutando y no mueren. Siempre hay una solución, y es enviarle la señal de KILL para que muera, esta no la pueden evitar.

Comando	Uso
ps [-l] [-a] [-e] [-u <i>user</i> ]	Nos muestra los procesos en ejecución actualmente -l: listado largo -a: activados por todos los usuarios -e: visualiza todos los procesos -u <i>user</i> : visualiza todos los procesos del usuario <i>user</i>
nice	Nos permite especificar la prioridad de un proceso. Normalmente se utiliza para reducir la prioridad del proceso, en el caso de que no nos urjan los resultados.
kill [-9] <i>pid</i>	Nos sirve para detener la ejecución de un proceso. Para ello necesitaremos conocer su PID. En la misma orden podemos cancelar varios procesos. Al poner el -9 (señal KILL) nos aseguramos de que se va a realizar la cancelación.
CTRL+Z	Suspende (duerme) un proceso (no gastando tiempo de CPU)
sleep	Suspende la ejecución durante un cierto número de segundos
jobs	Nos lista los trabajos actuales en ejecución. Nos pone un & al lado para los procesos que estén en background. Podemos matar trabajos con kill % <i>n</i> (si ponemos el número de trabajo: 1,2,3, ...) o kill <i>pid</i> (si ponemos el número de pid: 5794, ...).
fg [% <i>n</i> ] ≡ % <i>n</i>	foreground: pasa una tarea a primer plano, o la activa si estaba suspendida. Podemos especificar un número de trabajo (job); si no lo hacemos se considera el último que se ha dormido.
bg [% <i>n</i> ]	background: pasa una tarea a segundo plano, del mismo modo que si hubiéramos puesto & en el momento de invocar la orden. El [% <i>n</i> ] es igual que en "fg".
nohup <i>comando</i> < <i>f1</i> > <i>f2</i> 2> <i>errores</i> &	Nos permite ejecutar un comando, el cual perdurará cuando cerremos la sesión. Las entradas y salidas de datos las realizará sobre ficheros: nohup.in y nohup.out (por defecto).
top	Muestra los procesos que más CPU están utilizando

El comando yes es un comando aparentemente inútil que envía una serie interminable de y-es a la salida estándar y hay que pulsar CTRL+C para que pare. Puede ser útil cuando en un programa se nos realiza una serie de preguntas y queremos responder a todas que si, direccionando su entrada estándar al comando yes, el lo hará por nosotros.

- Ejecutar el comando `yes` y parar su ejecución con `CTRL+C`  
(La tecla de interrupción se puede definir con el comando `stty`, y puede que en todos los sistemas no sea la misma)

Lo que podemos hacer es que no nos moleste la aparición de `y-es` en la pantalla, redirigiendo la salida del comando `yes` a un dispositivo especial, `/dev/null` con lo que ya no aparece nada por pantalla,

- Redirigir la salida, ¿ que ocurre entonces?

Pulsando la tecla interrupción podemos parar el proceso, ya que sigue ejecutandose en primer plano, aunque no veamos nada.

Supongamos que queremos dejar que el comando `yes` siga pero que nosotros podamos recuperar el prompt para seguir trabajando.

- Lanzar el comando `yes > /dev/null &` para dejarlo ejecutando en segundo plano.

¿Qué pasaría si no pusiesemos `> /dev/null`?

Ahora nos aparece un número entre corchetes, por ejemplo `[1]` seguido de un numero, por ejemplo `32832`. ¿Qué es eso?

`[1]` representa el número de tarea que el sistema operativo le da al proceso `yes` cuando lo ubica en segundo plano. El número que le sigue es el PID (identificador de proceso) que es el número de proceso que tiene `yes`, independientemente de en que plano se ejecute. Estos dos números se utilizan para referirse a la tarea o proceso.

Ahora tenemos el proceso `yes` corriendo en segundo plano, y enviando constantemente la letra `'y'` hacia el dispositivo `/dev/null`. Para chequear el estado del proceso, utilice el comando `jobs`.

- Utilizar `jobs` para ver las tareas.

También podemos usar el comando `ps`

- ¿Qué diferencia hay entre `jobs` y `ps`?

Para eliminar una tarea podemos hacerlo por dos vías. A través de su número de tarea y a través de su número de proceso y mediante el comando `kill`. (NOTA: `kill`, a pesar de su nombre no es una orden para matar procesos, sino para enviar señales a procesos, lo que pasa es que normalmente si un proceso recibe una señal que no espera, muere).

Así que con `kill %1` nos referimos al número de tarea `1` y con `kill 32832` le decimos que le envíe la señal por defecto al proceso con PID `32832`. La señal por defecto es `KILL`.

Una vez matada la tarea, la siguiente vez que ejecutemos `jobs` (pero solo esa vez, la siguiente) aparecerá un mensaje indicando que el proceso `[1]` ha terminado. Si volvemos a usar `jobs` ya no muestra nada.

Hay otra manera de poner una tarea en segundo plano. Se lanza el proceso, se para y queda parada en primer plano, pero nos devuelve el prompt, se pone entonces en segundo plano.

- Lanzar en primer plano, yes >/dev/null (SIN &) (no saldrá el prompt, recordais?)

Ahora en vez de interrumpir (CTRL+C) la vamos a suspender con CTRL+Z.

Nos indica el sistema que la tarea [1] ha sido stopped, parada.

Ahora el proceso (tarea) esta suspendido y no consume CPU, ni recursos. Podemos ver que es así con ps.

Para relanzar la tarea a primer plano y que siga ejecutando como si no hubiera pasado nada se utiliza fg. Nos mostrará el nombre del comando que pasa a foreground pero sigue sin salir nada por pantalla porque hemos redirigido a /dev/null.

Volvemos a parar el proceso con CTRL+Z.

Ahora hacemos bg y entonces la tarea sigue ejecutándose pero esta vez en segundo plano. Con jobs vemos la tarea.

¿Que pasa ahora si pulsamos CTRL+C o CTRL+Z? Que no nos hace caso pues la tarea está en segundo plano. La podemos pasar a primer plano con fg y luego pararla o interrumpirla.

Nota: los comandos fg y bg actúan sobre la última tarea parada (suspendida). Con fg%n pasamos a primer plano la tarea n.

```
yes > /dev/null&      // Lo ejecutamos en background
fg                    // Lo ponemos en foreground
CTRL+Z (suspender proceso) // Lo suspendemos (para recuperar el control)
bg                    // Lo colocamos de nuevo en background
```

Si nuestro terminal se queda "colgado" o hay un proceso en marcha que queremos cancelar y no se puede abortar con CTRL+C, debemos abortar el proceso del shell. Para ello, tenemos que:

1. iniciar otra sesión (ALT+Fn si estamos en local ó con otra ventana de Putty si estamos en remoto).
2. Visualizar los procesos de nuestro usuario: ps -u *usuario*.
3. Matar el proceso cuyo PID corresponde al shell que queremos abortar: kill -9 *pid*

## 15. Metacaracteres

Este conjunto de caracteres especiales que hemos estudiado:

< > | ; &

son caracteres interpretados por la shell no argumentos para programas que ejecuta el shell. Los caracteres que tienen una interpretación especial por la shell se conocen como **METACARACTERES**. Se incluyen en este grupo también el \*, \$, =, [, #.

# significa que lo que este a su derecha lo va a considerar como un comentario.

Ejecutar:

```
echo hola # y esto que sigue es un comentario
```

¿Qué mostrará echo \* ?

El shell sustituye el \* por cualquier cadena por tanto la orden **echo \*** mostrará todos los ficheros que tenga el directorio en el que estoy actualmente. Si ejecutamos **echo .\*** el SO nos muestra aquellos ficheros que comienzan por . seguidos de cualquier carácter.

Ya sabemos que los nombres de ficheros pueden tener cualquier combinación de letras.

¿Cómo listar solo aquellos ficheros que comiencen con la letra '\*'?

Hay que decirle a la shell que ese carácter no lo interprete sino que lo trate como un carácter únicamente. Para ello veamos el uso de las COMILLAS.

## 16.Comillas.

Hacer:

```
$ ls -l >*volcado
$ ls -l >***volcado
```

### Comillas simples ' '

Inhabilitan la interpretación de los metacaracteres. Por ejemplo, quiero listar todos los ficheros que comiencen con '\*', será

```
ls '***'
```

y si quiero los que empiecen con \*\*

```
ls '***'
```

Lo mismo ocurre para cualquier otro metacaracter. Quiero localizar los ficheros que tengan un signo de ?. Hazlo.

Por tanto, las comillas simples desactivan toda función de análisis de la shell sobre lo que hay dentro de las comillas y lo que hay dentro se traspasa a la shell como un único parámetro.

Otro modo de decirle a la shell que trate un metacaracter como carácter es mediante la barra invertida \. Ejemplo: **ls \\*\***

¿Y si quiero visualizar un fichero de nombre "\var" Prueba como lo podrías hacer.

### Comillas dobles " "

Consideran todos los caracteres entre dos dobles comillas como una cadena de caracteres. Todos salvo: \ " \$ `(comilla invertida) los cuales son interpretados por el shell.

Ejecutar:

```
echo "ahora muestro mi login -> $LOGNAME"
```

¿es equivalente a esto?

```
echo ahora muestro mi login -> $LOGNAME
```

¿que ha hecho esta última orden?



quiero que me aparezca mi login entre comillas

```
echo "ahora muestro mi login entre comillas \" $LOGNAME \""
```

o con una \

```
echo "ahora muestro mi login entre \\ $LOGNAME \\"
```

*Qué hace y por qué: echo "\$LOGNAME=\$LOGNAME"*

Y si tengo que enviar un mensaje en inglés

```
echo "I don't Know"
```

Prueba lo que ocurre con : `echo I don't Know`

**Comillas invertidas ``**

(Nota: las comillas invertidas están en la misma tecla que el caracter ^. Hay que poner un espacio en blanco para que aparezca)

El intérprete considera la cadena entre comillas invertidas como una orden a ejecutar y la sustituye por el resultado de la ejecución de dicha orden. Ejecutar: `echo `pwd``

Ejecutar:

```
echo hola este soy yo `whoami`
```

que no es lo mismo que:

```
echo hola esta soy yo whoami
```

Probemos ahora que realmente las "" no desactivan ``. Veamos

```
echo "hola esta soy yo `whoami`"
```

Combinemos algo de lo que hemos visto:

```
hora=`date|cut -c11-18`
```

```
echo $hora
```

Me defino la variable hora y cuando la muestro lo que devuelve es la ejecución de lo que contiene hora. Comprueballo

¿Qué hace? `cd=`pwd``

```
echo $cd
```

*Ejercicio:*

¿Cómo puedo poner estas órdenes en una sola línea sin utilizar el ;'?

```
echo "Hola, buenos dias, hoy es :"
```

```
date
```

La orden de shell "set" despliega los valores de todas las variables definidas por el usuario. Comprobarlo, ejecutar set

La orden de shell "alias" despliega los valores de todos los alias que se hayan definido.

La sintaxis de definición de un **alias** es la siguiente:

**alias <variable>=<expresion>** (atención a ambos lados del = no hay espacios)

la expresión debería ir entre comillas siempre que tenga espacios en blanco para que la shell lo reconozca como un único parámetro.

Por ejemplo, podemos crear un alias llamado `dir` que haga un listado largo de directorios de la siguiente manera:

```
alias dir='ls -laF'
```

Si ejecutamos `dir` ¿qué se obtiene?

## 17. Terminadores de órdenes

---

Una orden termina casi siempre con **nueva-línea** pero el **punto y coma** también es un terminador de órdenes:

```
date;who|wc
```

es equivalente a

```
date  
who|wc
```

Compruébalo: primero se ejecuta “`date`” y más tarde “`who|wc`”

Otro terminador de órdenes es **&**. En este caso **&** le indica a la shell que no espere a que se termine la orden. **&** suele emplearse para ejecutar una orden que tarda mucho tiempo y mientras tanto se quiere continuar tecleando órdenes interactivamente.

## 18. Agrupación de órdenes:

---

En el ejemplo anterior se ejecuta `date`, luego `who` y la salida de `who` es la que se cuenta mediante `wc`. Sin embargo si escribimos:

```
(date;who)|wc
```

lo que toma como entrada la orden `wc` será el resultado de `date` y `who`.

Comprobarlo y observar que en este caso el número de líneas que muestra `wc` es una superior que en el caso anterior.

Comprobémoslo de otro modo:

```
(date;who)|tee medio|wc
```

¿Qué contiene el fichero medio?

**Sleep:** Esta orden espera el número especificado de segundos antes de terminar.

Veámoslo:

```
sleep 5
```

Veamos ahora nuestro ejemplo:

```
(sleep 5;date)
```

Probar ahora:

```
(sleep 5;date)&date
```

Explicar que pasa:

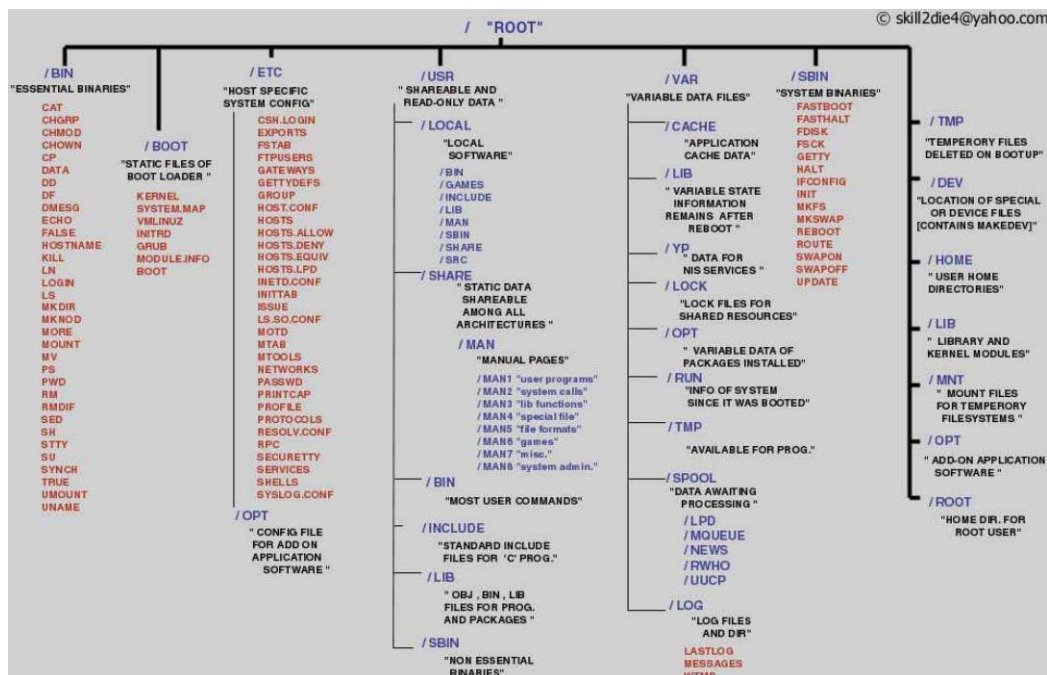
```
(ls;echo)&(who;echo)
```

Si quieres que te avise dentro de una hora para tomar café:

```
(sleep 3600;echo Nos toca cafe)&
```

## 19. Estructura de archivos en Linux

Una de las partes fundamentales del sistema operativo Unix son los archivos. Todo se hace a través de ellos. Los archivos se encuentran agrupados en como directorios. Estos directorios se encuentran organizados en una jerarquía de Árbol, donde la raíz está representada por el carácter `\`. Ver figura



A continuación se explica brevemente el contenido de cada directorio:

Directorio	Contenido
<code>/bin</code>	Contiene los ficheros ejecutables esenciales del sistema.
<code>/dev</code>	Alberga los controladores de dispositivo usados para acceder a elementos como discos duros, modems.....
<code>/etc</code>	Información y programas. Se encuentran archivos de configuración del sistema como "profiles", "hosts" o "bashrc"
<code>/sbin</code>	Ordenes ejecutables sólo por el administrador
<code>/home</code>	Directorio de usuarios
<code>/lib</code>	Librerías
<code>/proc</code>	Estructura virtual de ficheros. Es un directorio espejo de la memoria. En realidad los archivos ubicados allí no residen en el disco duro, con lo que no es posible modificarlos o borrarlos.
<code>/tmp</code>	Ficheros temporales
<code>/usr</code>	Archivos de configuración y programas usados por el sistema
<code>/var</code>	Históricos del sistema
<code>/boot</code>	Información necesaria para el sistema de arranque
<code>/cdrom</code>	Manejadores para el cd-rom
<code>/dev/console</code>	Sistema de consola
<code>/dev/ttyS</code>	Acceso a puertos
<code>/dev/cua</code>	Acceso a puertos
<code>/dev/hda</code>	Primer disco duro
<code>/dev/sda</code>	Primer disco duro SCSI
<code>/dev/lp0</code>	Primer puerto paralelo
<code>/dev/tty</code>	Consolas virtuales
<code>/dev/pty</code>	Seudoterminals
<code>/usr/x386</code>	Sistema X-windows
<code>/usr/bin</code>	Ficheros binarios o ejecutables comunes
<code>/usr/etc</code>	Información y programas

/usr/include	Ficheros para compilador C
/usr/man	Páginas man
/usr/src	Código fuente
/usr/src/linux	Código fuente del núcleo linux
/var/adm	Ficheros de administración
/var/spool	Ficheros de spool
/usr/x11/bin	Ejecutables X Window

## 20. Comando find y sus opciones

**find:** Pasa por los directorios especificados y genera una lista de archivos que cumplen los criterios que se han especificado. Se utiliza para buscar archivos normalmente. Sintaxis:

**find <lista\_directorios> <concordancia>.**

**<lista\_directorios>:** lista de directorios por los que se desea buscar separados por espacios

**<concordancia>:** se especifica de la siguiente manera:

**-name <fichero>:** nombre de fichero a buscar

**-user <id\_usuario>:** busca todos los ficheros cuyo propietario coincida con id\_usuario

HACER:

buscar desde el directorio "/" (raíz) el fichero cuyo nombre es passwd

*¡Ayuda, redireccionar a null la salida de error!*

Opciones: (ver manuales)

**! expr** Busca los que no cumplen la expresión.

**-name** nombre busca los que se llaman como se indica en nombre

Si se usan metacaracteres \*, &#92;, [ ] poner comillas dobles

<b>-print</b>	saca el resultado por la salida estándar. Se utiliza al final de expr.
<b>-type f</b>	solo busca ficheros regulares
<b>-type d</b>	solo busca directorios
<b>-type l</b>	solo busca link simbólicos (soft)
<b>-size 42c</b>	busca que el tamaño sea el número especificado en bytes.
<b>-size +42c</b>	Tamaño mayor que el especificado.
<b>-size -42c</b>	Tamaño inferior al especificado.
<b>-perm n<sup>o</sup>octal</b>	Busca los permisos concretos que se especifican.
<b>-perm permisos</b>	Busca que los usuarios que se indican tengan los permisos que se especifican. Con signo - busca que al menos tenga esos. Ejemplos: -perm -0105 -perm -g+rx
<b>-atime numero</b> <b>-ctime</b> <b>-mtime</b>	Da los ficheros o directorios que han sido accedidos, cambiados, o modificado en los múltiplos de 24 horas que se da.
<b>-links n</b>	Los que tienen el número de enlaces n (+ = mayor que )
<b>-user usuario</b>	Los que tienen como propietario el usuario especificado.
<b>-exec comando \;</b>	ejecuta el comando que se dice si la búsqueda tiene éxito. Si se indica \{ } despues del comando los argumentos serán el resultado del comando find. Debe acabar la línea en espacio\;
<b>-ok</b>	Pide confirmación antes de ejecutar.

Ejemplo:

**find \$HOME -type d -links 2 que no tengan subdirectorios**

**find \$HOME -type f -size 0 -exec rm -i {} \;** borra los ficheros de tamaño 0

## 21. Comunicación entre usuarios

Tenemos varias posibilidades: write, wall, talk y mail.

### Comando **write**

Nos permite enviar un mensaje al usuario especificado. La sintáxis es: `write Usuario`

Justo en el momento en que escribimos el mensaje y pulsamos [Intro], el otro usuario lo recibe, siempre y cuando esté también conectado.

Los mensajes que se reciben se entremezclan con los comandos que escribe el usuario remoto, lo cual puede ser molesto. Además si no deseamos que nos interrumpan otros usuarios con mensajes los podemos anular con: `mesg n`

A partir de este momento, si alguien intenta hacerle un write le aparecerá el siguiente mensaje:

```
write: warning: you have write permission turned off.
```

Si después queremos escribir un mensaje a alguien y recibir respuesta, tendremos que volver a activar los mensajes escribiendo: `mesg y`.

### Comando **wall**

Este comando es parecido al comando write, salvo que el mensaje lo reciben todos los usuarios que estén conectados, y que no tengan **mesg** fijado a "n".

Al escribir wall y pulsa [Intro], podremos escribir el mensaje (incluso en varias líneas), y en el momento en que pulsemos [CTRL+D] se enviará a todos los usuarios (broadcast).

```
$ wall
ATENCION: quedan 5 minutos para terminar! [CTRL+D]
```

IMPORTANTE: cuando se envía un mensaje con wall, aparece el nombre de usuario del remitente, es decir, no es posible enviar mensajes anónimos.

```
$ _
Broadcast Message from usuario1@maquina
(/dev/pts/0) at 17:17 ...

ATENCION: quedan 5 minutos para terminar!
$ _
```