

Linux

Para acceder a un sistema Unix necesitamos tener una cuenta.

Una cuenta de Unix incluye:

- Nombre de usuario y contraseña.
- IdUsuario y idGrupo.
- Directorio de inicio.
- Shell.

Archivo:

- Un archivo es una unidad básica de almacenamiento (generalmente almacenamiento en un disco).
- Cada archivo tiene un nombre.
- Los nombres de archivos Unix pueden contener cualquier carácter (aunque algunos dificultan el acceso al archivo).
- Los nombres de archivos Unix pueden ser muy largos.
- Cada archivo puede contener algunos datos sin procesar.
- Unix no impone ninguna estructura a los archivos: los archivos pueden contener cualquier secuencia de bytes.

Directorio:

- Un directorio de inicio es un lugar en el sistema de archivos donde se almacenan los archivos de la cuenta.
- Un directorio es como una carpeta de Windows, es un tipo especial de archivo: Unix utiliza un directorio para almacenar información sobre otros archivos.
- Un directorio es un tipo especial de archivo: Unix utiliza un directorio para almacenar información sobre otros archivos.
- A menudo pensamos en un directorio como un contenedor que contiene otros archivos (o directorios).
- Cada archivo en el mismo directorio debe tener un nombre único.
- Los archivos que están en diferentes directorios pueden tener el mismo nombre.

Shell:

- Un shell, es un interprete de comandos (interfaz) que convierte texto que uno escribe (en la línea de comando) en acciones:
 - Ejecutar un programa.
 - Editar una línea de comando.
 - Poder establecer fuentes alternativas de entrada y destino de salida para los programas.
- Es un intermediario que se encarga de traducir los comandos del usuario, a instrucciones que solo el núcleo o kernel del sistema operativo entiende.
- Cuando inicia sesión en un sistema Unix, el programa con el que interactúa inicialmente es su shell.
- El shell establece 3 canales de E / S:
 - Entrada estándar.
 - Salida estándar.

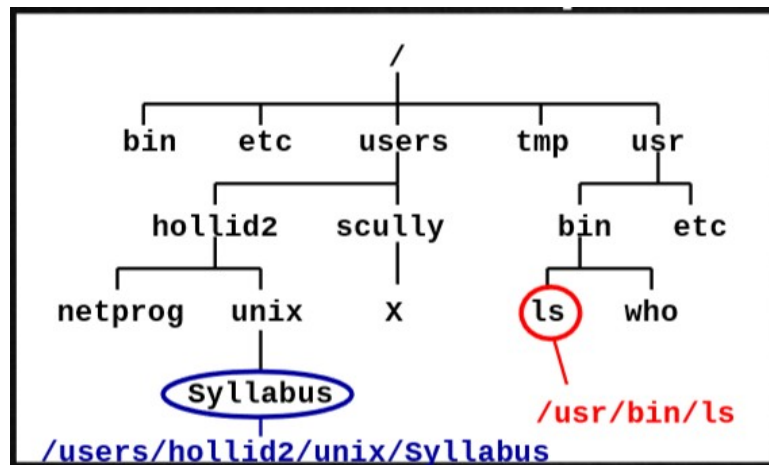
- Error estándar.
- La mayoría de los comandos (programas) de Unix:
 - Leen algo de la entrada estándar.
 - Envían algo a la salida estándar (generalmente depende de cuál sea la entrada).
 - Envían mensajes de error al error estándar.
- Cuando un shell ejecuta un programa para usted:
 - La entrada estándar es su teclado.
 - La salida estándar es su pantalla / ventana.
 - El error estándar es su pantalla / ventana.
- Cada vez que le da al shell una línea de comando, hace lo siguiente:
 - Comprueba si el comando es un shell incorporado.
 - Si no, intenta encontrar un programa cuyo nombre (el nombre del archivo) sea el mismo que el comando.
- La variable PATH le dice al shell dónde buscar programas (comandos no integrados).
- El shell te permite administrar trabajos:
 - Colocar trabajos en segundo plano.
 - Mover un trabajo al primer plano.
 - Suspende un trabajo.
 - Matar un trabajo.
- Shells populares:
 - **sh** Bourne Shell.
 - **ksh** Korn Shell.
 - **cs**h C Shell.
 - **bash** Bourne-Again Shell.
 - **dash** Debian Almquist Shell.

Sistema de archivos Unix:

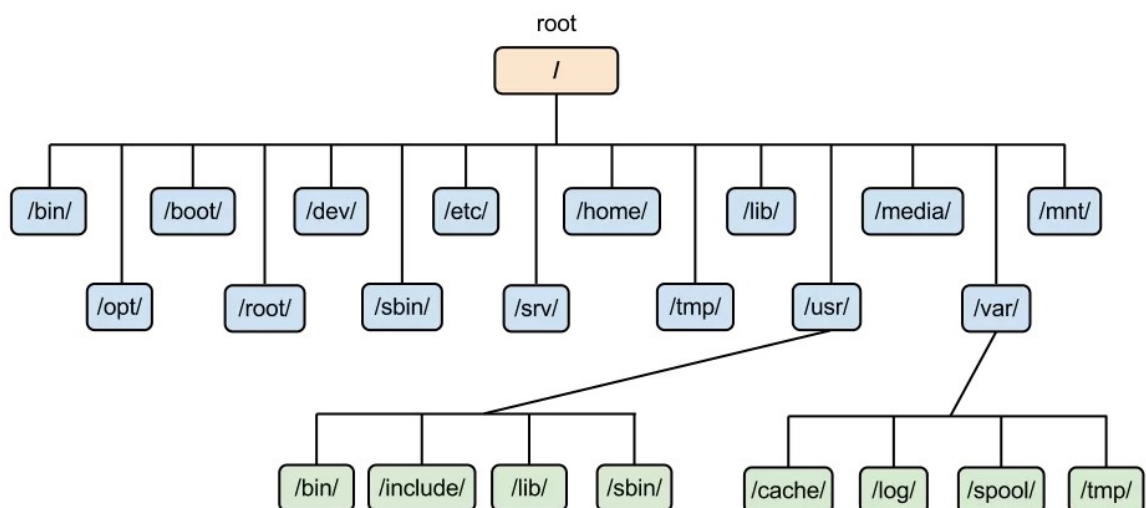
- El sistema de archivos es un sistema jerárquico de organización de archivos y directorios.
- El nivel superior en la jerarquía se llama "raíz" y contiene todos los archivos y directorios.
- El nombre del directorio raíz es "/".

Nombres de ruta:

- El nombre de ruta de un archivo incluye el nombre del archivo y la lista de nombres de directorio hasta la raíz.
- El nombre de ruta de cada archivo en un sistema de archivos Unix es único.
- Para crear un nombre de ruta, se comienza en la raíz (para comenzar con "/"), luego se sigue la ruta hacia abajo en la jerarquía (incluyendo cada nombre de directorio) y se termina con el nombre de archivo.
- Entre cada nombre de directorio hay que poner "/".



Algunos directorios Linux:



/bin – Binarios

- Contiene los binarios ejecutables.
- Los comandos esenciales como ls o cp generalmente están en esta carpeta.
- Pueden ser usados por todos los usuarios del sistema.

/sbin – Binarios del sistema

- Igual que /bin, tiene binarios que son comunes al sistema Linux.
- Generalmente son binarios que pertenecen a tareas administrativas.
- No todos los usuarios tienen acceso a los binarios de esta carpeta.

/etc – Archivos de configuración

- Contiene archivos de configuración requerido por los programas.
- También tiene archivos para iniciar o detener programas individuales.

/dev – Archivos de dispositivos

- Contiene los archivos de los dispositivos instalados.
- También incluye terminales, USB o cualquier dispositivo que esté en el sistema.

/proc – Información de procesos

- Contiene información sobre los procesos de Linux.
- Contiene un pseudo sistema de archivos, que consiste en /proc/{id_proceso}, el cual contiene información sobre ese proceso en particular.

/var – Archivos varios

- Contiene información variable.
- Información que se espera que crezca con el tiempo, es encontrado aquí.
- Archivos de logs (/var/log), paquetes y base de datos (/var/lib), email (/var/mail), archivos temporales (/var/tmp).

/tmp – Archivos temporales

- Lugar donde se encuentran archivos temporales creados por programas y los usuarios.
- Se eliminan en cada reinicio del sistema.

/usr – Rutinas especiales de Linux

- Aunque el nombre puede dar a confusión, no tiene nada que ver con archivos de usuarios.
- La jerarquía parece como la de root (/).
- Contiene binarios, archivos, documentación y recursos de segundo nivel.
- /usr/bin contiene también binarios, pero son menos usados que los de /bin, por ejemplo: awk, less, scp.
- /usr/sbin también contiene binarios del sistema que son menos utilizados, cron, sshd, useradd.
- /usr/local contiene programas que se instalan directamente desde el código fuente.

/home – Directorio de inicio

- Es el directorio de inicio de cada usuario.
- Este directorio contiene una carpeta por cada usuario en el sistema que puede iniciar sesión.
- Solo los usuarios que pertenecen a sus propias carpetas tienen permisos de escritura, lectura y ejecución (además de root).

/boot – Archivos de carga

- Contiene los archivos de inicio del kernel.
- Por ejemplo, grub está localizado en este directorio.
- No encontrarás información de cómo Linux bootea en este directorio.

/lib – Bibliotecas del sistema

- Contiene las bibliotecas que son necesarias para los programas de /bin /sbin.
- Generalmente son archivos que contienen ld* o li.so.* en su nombre.
- Por ejemplo: ld-2.11.1.so, libncurses.so.5.7.

/opt – Archivos binarios opcionales

- Contiene software de terceros.
- Muchos sistemas no utilizan este directorio.

/mnt – Directorio de carga

- Ideado para que los administradores monten otros sistemas de archivos, como USB, otros Discos Duros.

/media – Dispositivos removibles

- Sirve para montar dispositivos removibles como CD-ROM, Floppy Disk, cintas, etc.
- Actualmente muy en desuso, se suele utilizar /mnt.

Caracteres especiales:

- Cuando escribimos una línea de comando, el shell trata algunos caracteres como especiales.
- Estos caracteres especiales hacen fácil la especificación de nombres de archivo.
- El shell procesa lo que le damos, usando los caracteres especiales para reemplazar nuestra línea de comando por una que incluye un montón de nombres de archivos.
- ***** devuelve todo. Si proporciona el shell * por sí mismo (como argumento de línea de comando), el shell eliminará el * y lo reemplazará con todos los nombres de archivo en el directorio actual.
- **"a*b"** devuelve todos los archivos en el directorio actual que comienzan con a y terminan con b.
- **?** coincide con cualquier carácter.

`ls Test?.doc`

- **[abc...]** coincide con cualquiera de los caracteres incluidos.

`ls T[eE][sS][tT].doc`

- **[a-z]** coincide con cualquier carácter en un rango.

`ls [a-zA-Z]*`

- **[!abc...]** coincide con cualquier carácter (excepto los mencionados).

`ls [!0-9]*`

- Algunos caracteres especiales no se ignoran, incluso dentro de comillas dobles:

- **\$** obtiene el valor de una variable.
 - **"** el carácter de la cita en sí.
 - **** slash siempre es algo especial (\n).
 - Puede usar \\$ para decir \$ o \" para decir ".
- ```
echo "This is a quote \" "
```
- Puede usar comillas simples como comillas dobles.
- ```
> echo 'This is a quote "'  
This is a quote "  
> echo 'This is a backslash \'  
This is a backslash \
```
- Si rodea una cadena con comillas inversas, la cadena se reemplaza con el resultado de ejecutar el comando en comillas inversas.

Comandos:

- **ls:** Sirve para listar los archivos y carpetas que hay dentro del directorio el el que estés.
 - **ls [opciones] [nombres]:**
 - Los corchetes alrededor de las opciones y los nombres en la forma general del comando ls significa que algo es opcional.
 - Algunos comandos tienen parámetros requeridos.
 - Puede darle al comando ls muchos nombres (archivos o directorios):
`ls /usr /etc`
`ls -l /usr/bin /tmp /etc`
 - **ls:** lista los archivos del directorio donde estamos parados.
 - **ls /:** lista los archivos del directorio raíz.
 - **ls .:** lista los archivos del directorio actual.
 - **ls ..:** lista los archivos del directorio padre.
 - **ls /usr:** lista los archivos del directorio /usr.
 - Podemos modificar el formato de salida del programa ls con una opción de línea de comando.
 - El comando ls admite un montón de opciones.
 - Para usar una opción de línea de comando, hay que preceder la letra de opción con un menor:
 - **ls -a:** Nos muestra los archivos y directorios dentro del directorio actual, incluyendo los archivos y directorios ocultos.
 - **ls -t:** Ordena los archivos por fecha de modificación.
 - **ls -X:** Ordena los archivos por extensión.
 - **ls -l:** Muestra toda la información: usuario, grupo, permisos, tamaño, fecha y hora de creación.
 - **ls -lh:** Muestra la misma información que ls -l pero con las unidades de tamaño en KB, MB, etc.
 - **ls -R:** Muestra el contenido de todos los subdirectorios de forma recursiva.
 - **ls -S:** Ordena los resultados por tamaño de archivo.
 - Se puede usar 2 o más opciones al mismo tiempo de esta manera:
`ls -al`
- **who:** Nos muestra quien está conectando en el sistema.
- **date:** Muestra la hora y fecha actual.
- **pwd:** Nos muestra el nombre del directorio de trabajo actual.
- **cd:** El comando cd puede cambiar el directorio de trabajo actual.
`cd [directoryname]`
 - Sin ningún parámetro, el comando cd cambia el directorio actual a su directorio de inicio.
 - También puede dar a cd un nombre de ruta relativo o absoluto:
`cd /usr`
`cd ..`
- **df:** muestra qué disco contiene un directorio.

- **cp:** El comando cp copia archivos:

cp [opciones] source dest

- source es el nombre del archivo que desea copiar.
- dest es el nombre del nuevo archivo.
- source y dest pueden ser relativos o absolutos.
- si dest es un directorio, cp pondrá una copia de source en el directorio.
- Si se especifica más de dos nombres, cp supone que se está utilizando esta forma:

cp [opciones] source... destdir

- En este caso, cp copiará múltiples archivos a destdir.
- source ... significa al menos un nombre (podría ser más de uno).

- **rm:** El comando rm elimina archivos:

rm [opciones] nombres...

- rm significa "eliminar".
- Se puede eliminar muchos archivos a la vez:

rm foo /tmp/blah /users/clinton/intern

- **mv:** Mueve archivos o directorios de un lugar a otro:

mv source1,source2 dest

- También cambia el nombre de los archivos (por ejemplo, cambia el nombre del archivo.txt a test.txt):

mv file.txt test.txt

Redirección:

- Para indicarle al shell que almacene la salida de su programa en un archivo, siga la línea de comando para el programa con el carácter ">" seguido del nombre del archivo:

ls > lsout

(el comando anterior creará un archivo llamado lsout y colocará la salida del comando ls en el archivo)

- Para indicarle al shell que obtenga una entrada estándar de un archivo, use el carácter "<":

sort < nums

El comando anterior ordenaría las líneas en el archivo nums y enviaría el resultado a stdout.

- Se pueden hacer ambos al mismo tiempo:

sort < nums > sortednums

tr a-z A-Z < letter > rudeletter

- El comando **ls > foo** creará un nuevo archivo llamado foo (eliminando cualquier archivo existente llamado foo).
- Si usa **>>** la salida se agregará a foo:

ls /etc >> foo

ls /usr >> foo

- Hay 3 descriptores de archivo: stdin, stdout y stderr (std = estándar). Básicamente puedes:
 - Redirigir stdout a un archivo.
 - Redirigir stdout a un stderr.
 - Redirigir stderr a un stdout.
 - Redirigir stderr y stdout a un archivo.
 - Redirigir stderr y stdout a stdout.
 - Redirigir stderr y stdout a stderr.

Pipes:

- Una tubería (pipe) es un soporte para un flujo de datos.
- Se puede usar una tubería para mantener la salida de un programa y alimentarla a la entrada de otro.
- 2 comandos se separan con el carácter "|".

`ls | sort`

`ls | sort > sortedls`

- Se puede usar pipes con una serie de comandos de Unix para hacer algo nuevo.
- **tee**: Permite redirigir la entrada a la entrada estándar y también a uno o más archivos. Muy útil con pipes.

`find /home -name '*.txt' | tee search.output | grep "poems"`

`cut -f2 students.csv | tee students.unordered | sort -n > students.ordered`

Variables:

- Puede usar variables como en cualquier lenguaje de programación.
- No hay tipos de datos. Una variable en bash puede contener un número, un carácter, una cadena de caracteres:

`#put "something" into myvar`

`myvar="something"`

`#put 4 into MY_VAR2`

`MY_VAR2=4`

`#Concat myvar,MY_VAR2, add "extra" and save it as concat_var`

`concat_var="$myvar $MY_VAR2 extra"`

`# Prints something 4 extra`

`echo $concat_var`

- Prefije el nombre de una variable de shell con **"\$"** (obtiene el valor de una variable).
- Puede cambiar el valor de una variable de shell con un comando de asignación (este es un comando incorporado de shell).

`HOME=/etc`

`PATH=/usr/bin:/usr/etc:/sbin`

`NEWVAR="blah blah blah"`

- El comando set sin parámetros imprimirá una lista de todas las variables de shell.

- Ejemplo:

```
1 #!/bin/sh
2 STR="Hello World"
3 echo $STR
```

- Variables locales:

```
1 #!/bin/bash
2 HELLO=Hello
3 function hello {
4     local HELLO=World
5     echo $HELLO
6 }
7 echo $HELLO
8 hello
9 echo $HELLO
```

Atributos de archivo:

Cada archivo (incluidos los directorios) tiene algunos atributos:

- Tiempos de acceso:
 - cuándo se creó el archivo.
 - cuándo se modificó por última vez.
 - cuándo se leyó por última vez.
- Tamaño.
- Propietarios (usuario y grupo).
- Permisos.

Propietarios de archivos:

- Cada archivo es propiedad de un usuario.
- Puede encontrar el nombre de usuario del propietario del archivo usando "ls -l".
- Cada archivo también es propiedad de un grupo Unix.
- ls -l también muestra el grupo que posee el archivo.

Permisos de archivo:

Cada archivo tiene un conjunto de permisos que controlan quién puede meterse con el archivo.

Hay tres tipos de permisos:

- read abreviado "r".
- write abreviado "w".
- execute abreviado "x".

Hay permisos separados para el propietario del archivo, el propietario del grupo y todos los demás.



- Archivos:
 - r – permiso para leer.
 - w – permiso para escribir.
 - x – permiso para ejecutar.
- Directorios:
 - r – permite ver los nombres del archivo.
 - w – permite agregar y eliminar archivos.
 - X – permite ingresar al directorio.

Cambiar permisos:

El comando `chmod` cambia los permisos asociados con un archivo o directorio.

Hay varias formas de `chmod`, esta es la más simple:

chmod mode file

"mode" tiene la siguiente forma:

`[ugoa][+ -=][rwx]`

- u = usuario + añadir permiso
- g = grupo - remover permiso
- o = otros = establecer permiso
- a = todos

Ejemplos:

- `ls -al foo`
`rw-rw-r--x 1 hollind grads ...`
- `chmod o-x foo`
`ls -al foo`
`rw-rw-r-- 1 hollind grads`
- `chmod u-r .`
`ls -al foo`
`ls: .: Permission denied`

Otros comandos de sistemas de archivos y archivos:

- **mkdir:** El comando `mkdir` sirve para crear un nuevo directorio. Hay que tener en cuenta que lo crea por defecto en el directorio en el que nos encontramos, si se quisiera crearlo en otro directorio se debería incluir la ruta.
- **du:** tamaño del directorio.
- **rmdir:** eliminar directorio.
- **touch:** Con este comando podemos crear ficheros (vacíos).
- **cat:** `cat` sirve para ver el contenido de un archivo sin editarlo. Simplemente nos muestra su contenido sin posibilidad de cambiarlo. También se usa para concatenar archivos.

Texto:

- Hay muchos comandos de Unix que manejan datos de texto:
 - Operaciones en archivos de texto.
 - Operaciones en una secuencia de entrada.
- Operaciones:
 - Búsqueda.
 - Procesamiento (manipulaciones).
- **grep:** grep es un comando que utiliza expresiones regulares para realizar consultas. Ésto es realmente útil cuando te encuentras con una gran variedad de archivos y sólo desea visualizar aquellos que tengan un patrón en particular: una extensión en concreto, que comiencen sólo por una letra que nosotros queremos, etc...

grep [opciones] regexp [archivos]

- regexp es una "expresión regular" que describe algún patrón.
- los archivos pueden ser uno o más archivos (si no hay ninguno, grep lee de la entrada estándar).
- El siguiente comando buscará en los archivos a, b y c la cadena "foo". grep imprimirá cualquier línea de texto que encuentre (que contenga "foo").
grep foo a b c
- Sin ningún archivo especificado, grep leerá de la entrada estándar:
grep foo
- Opciones de grep:
 - **-c** imprime solo un recuento de líneas coincidentes.
 - **-h** no imprime nombres de archivo.
 - **-l** imprime el nombre del archivo pero no la línea que coincide.
 - **-n** imprime el número de líneas.
 - **-v** imprime todas las líneas que no coinciden.
- **egrep:** Esta variante es una versión más completa que el grep original, que mantiene las funcionalidades originales y además le añade unas nuevas. Este comando es muy útil para realizar consultas avanzadas que grep por sí sólo no es capaz de realizar. egrep acepta algunos símbolos que grep de por sí no acepta.
- **fgrep:** es una variante más veloz que grep pero que cómo contrapartida sólo puede buscar cadenas fijas (sin expresiones regulares, sólo la cadena). Cuando sólo se usa una cadena cómo patrón de búsqueda no se nota mucha diferencia, en cambio si se usa un gran número de patrones la diferencia es muy significativa.
- **strings:** El comando strings busca cadenas de texto en cualquier tipo de archivo (incluidos archivos de datos binarios y programas ejecutables) e imprime cada cadena encontrada. strings se utiliza generalmente para buscar texto en un archivo binario.
- **find:** find busca en el sistema de archivos, archivos cuyo nombre coincida con un patrón.

Conceptos comunes:

Estos comandos a menudo se usan como filtros, leen de entrada estándar y envían la salida a salida estándar.

- **head:** muestra solo la "cabeza" (comienzo) de un archivo.

head [opciones] [archivos]

- Por defecto head muestra las primeras 10 líneas.
- Este comando asume que el archivo es un archivo de texto.

- **tail:** muestra solo la "cola" (final) de un archivo.

tail [opciones] [archivos]

- Por defecto tail muestra las últimas 10 líneas.
- Este comando asume que el archivo es un archivo de texto.
- Algunas opciones:
 - **-r:** muestra las líneas en orden inverso. (no está en todas las versiones).
 - **-f:** no sale hasta el final del archivo.

- **cut:** selecciona (e imprime) columnas o campos de líneas de texto.

cut opciones [archivos]

- Se debe especificar una opción.
- **-c lista** corta las posiciones de caracteres definidas en la lista.
- **La lista puede ser un:**

- número (especifica un solo carácter posición).

cut -c1 imprime el primer carácter. (en cada línea)

- rango (especifica una secuencia de posiciones).

cut -c1-10 imprime los primeros 10 caracteres

- lista separada por comas (especifica múltiples posiciones o rangos).

cut -c1,10 imprime el primero y el décimo carácter

cut -c5-10,15,20- imprime el 5,6,7,8,9,10,15,20,21,... carácter de cada línea.

- **-f lista** corta los campos identificados en la lista. Un campo es una secuencia de texto que termina en algún carácter separador (delimitador). Puede especificar el separador con la opción -d.

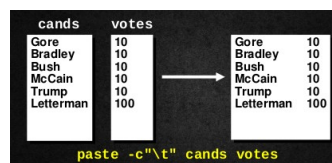
-dc donde c es el delimitador (El delimitador predeterminado es un enter)

- **cut -d: -f1** imprime todo antes del primer ":" (en cada línea).
- **cut -d" " -f1** imprime todo antes del primer espacio (en cada línea).
- **cut -f1** imprime todo antes del primer enter.
- **cut -d: -f2,3** imprime la 2da y 3er columna delimitada por :
- **cut -d" " -f2** imprime la segunda columna usando el espacio como delimitador.

- **paste:** paste pone líneas de uno o más archivos juntos en columnas e imprime el resultado.

paste [opciones] archivos

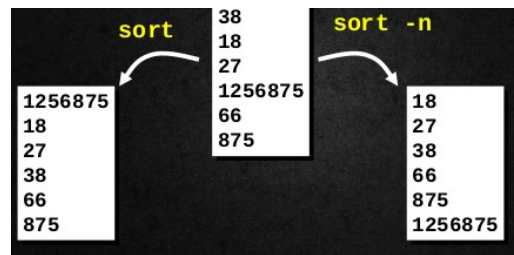
- La salida combinada tiene las columnas separadas por enters.



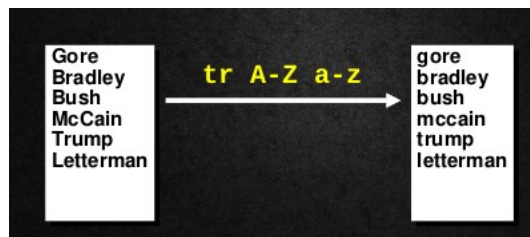
cands	votes
Gore	10
Bradley	10
Bush	10
McCain	10
Trump	10
Letterman	100

paste -c"\t" cands votes

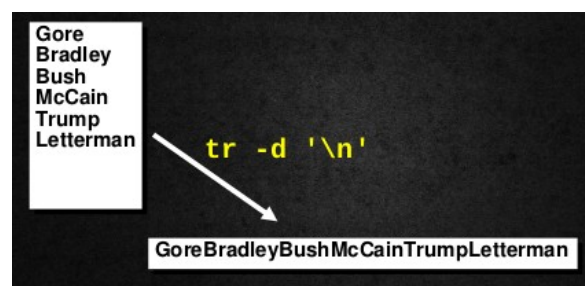
- -dc separa las columnas de salida con el carácter c.
- **sort:** sort reordena las líneas en un archivo (o archivos) e imprime el resultado.
sort [opciones] [archivos]
 - De forma predeterminada, la ordenación usa un orden alfabético.



- **uniq:** uniq elimina las líneas adyacentes duplicadas de un archivo.
 - uniq se usa típicamente en un archivo ordenado (que obliga a que las líneas duplicadas sean adyacentes).
 - uniq también puede reducir múltiples líneas en blanco a una sola línea en blanco.
- **tr:**
 - tr es la abreviatura de traducir.
 - tr traduce entre dos conjuntos de caracteres.
 - reemplaza todas las ocurrencias del primer carácter en el set 1 con el primer carácter en el set 2, el segundo carácter en el set 1 con el segundo carácter en el set 2, ...



- La opción -d significa "eliminar caracteres que se encuentran en la cadena1".



Scripts:

- Son archivos de texto, generalmente identificados con la extensión ".sh".
- Por lo general, se agrega un comentario especial para indicar el shell utilizado para ejecutar el script. Por ejemplo:

`#!/bin/bash`

- Funciones:
 - Una función es un sub-script. Recibe argumentos y para devolver valores debe imprimir a la salida estándar.
 - ¡ADVERTENCIA! La palabra reservada "retorno" devuelve un valor entero que indica el código de error, no se utiliza para devolver el valor de la función.
 - Las funciones pueden tener sus propias variables, si la localidad necesita ser aplicada, la palabra "local" puede usarse para indicar variables locales.
- Condiciones y pruebas:
 - Primero, necesitamos conocer las diferentes formas de expresar las condiciones:
 - Corchetes.

```
1 #!/bin/bash
2 x=3
3 if [ $x = 3 ]; then
4     echo "equals!"
5 fi
```

- Es lo mismo que: `if test $ x = 3; then` . Es por eso que debes usar espacios antes de los corchetes.
 - Puede invertir una condición usando ! (después del primer paréntesis).
 - -a se usa para hacer condiciones de and y -o para hacer condiciones de or.
 - Con corchetes puede probar:
 - números usando igual (-eq), mayor que (-gt), mayor o igual que (-ge), y así (-lt, -le, -ne).
- `if [$x -eq 3]; then`
 - strings usando ==, !=, < y > (< y > deben escaparse usando \, porque también se usan para la redirección).
 - Archivos:
 - Existe: `-a myfile.tar.gz`, verdadero si existe el archivo (o directorio).
 - Dir: `-d mydir`, verdadero si existe y es dir.
 - Tamaño: `-s myfile`, el archivo existe y su tamaño es > 0.
 - Pruebas de strings:
 - No vacío: `-n "$ myvar"`
 - Vacío: `-z "$ myvar"`
 - Corchetes dobles: `if [[condición]]`; Puede probar las mismas cosas que usando corchetes individuales y:
 - Usar expresiones regulares simples para comparar cadenas.
- `If [["$var1" == *[aA]le*]];`
 - Omitir comillas para strings.
 - `If [[$var1 == algo]];`
 - Al probar archivos, no utiliza globbing. Ejemplo: compruebe si existe el archivo `"* .txt"`.

`if [[-a *.txt]]; then`

- Usar &&, || para encadenar condiciones (también -a y -o).
- Doble paréntesis: Verifica las condiciones numéricas usando <, >, <=, >=, !=, == && y ||.

if ((\$num >= 5)); then

- Condiciones con else:

```
1 #!/bin/bash
2 T1="foo"
3 T2="bar"
4 if [ "$T1" = "$T2" ]; then
5     echo expression evaluated as true
6 else
7     echo expression evaluated as false
8 fi
```

- ForEach Loop:

```
1 #!/bin/bash
2 for i in $( ls ); do
3     echo item: $i
4 done
```

- For:

```
1 #!/bin/bash
2 for (( i=0 ; i <=10 ; i++ )); do
3     echo "Repeat me!"
4 done
```

- While:

```
1 #!/bin/bash
2 COUNTER=0
3 while [ $COUNTER -lt 10 ]; do
4     echo The counter is $COUNTER
5     let COUNTER=COUNTER+1
6 done
```

- Until:

```
1 #!/bin/bash
2 COUNTER=20
3 until [ $COUNTER -lt 10 ]; do
4     echo COUNTER $COUNTER
5     let COUNTER-=1
6 done
```

- Funciones sin parámetros:

```
1 #!/bin/bash
2 function quit {
3     exit
4 }
5 function hello {
6     echo Hello!
7 }
8 hello
9 quit
10 echo foo
```

- Parámetros: Un parámetro es una entidad que almacena valores y es referenciado por un:
 - Nombre de variables (por ejemplo, echo \$MYVAR).

- Argumentos numéricos (del 1 al 9, accesibles con \$1 a \$9).
- Símbolo especial ?, *, \$, #, @, _, 0, !, - (por ejemplo, \$\$ devuelve el PID del proceso actual o \$0 devuelve el nombre del shell o el script del shell actual).
- Valor de acceso: \$PARAMETER o \${PARAMETER}. Acceda al valor del PARÁMETRO (\$VAR, \$4, \$@, otro ejemplo: \${WORD}s, agrega una "s" después del valor de WORD).
- Indirección: \${!PARAMETER}, accede al valor del parámetro contenido en PARAMETER.
- Expansión de nombre de variable: \${! PREFIX*} o \${! PREFIX@} busca variables que comienzan con PREFIX
- Asignación de expresiones:

- **let:**

```
let i++ or let i=i+1
let i=4+3
let a=4+3; echo $a → prints 7
let a=2**3 → a=8
```

- i=\$((i+1))
- En muchos shells, -x (para xtrace) imprime la ejecución del comando a un error estándar que permite al usuario depurar scripts.

```
# bash -x <(echo "echo Hello World")
+ echo Hello World
Hello World
```

Mismo ejemplo (usando pipe, en lugar del descriptor de archivo):

```
# echo "echo Hello World" | bash -x
+ echo Hello World
Hello World
```