UD 05_02 – GNU/Linux. El shell Bash. Redirecciones, tuberías y filtros.

1 EL SHELL BASH

En este apartado veremos algunas características de un shell concreto, el shell bash, que permiten configurarlo y hacer que sea más versátil.

1.1 Comando echo

El comando `echo` muestra en su salida todo lo que se le pase como argumento. Utilizaremos en muchas ocasiones este comando al trabajar con el shell, por ejemplo, para mostrar el valor de una variable.

Ejemplo:

```
$ echo Hola mundo
Hola mundo
```

1.2 Archivos de inicialización y finalización del shell bash

Por defecto, la distribución que utilicemos configura bash para que podamos empezar a utilizarlo inmediatamente. Pero si queremos cambiar o añadir otros parámetros de configuración, es bueno saber dónde tenemos que hacerlo.

Primero se ejecuta un fichero de configuración general para todos los usuarios del sistema (si existe) cuando inician sesión, `/etc/profile`.

Existen tres ficheros en el directorio de un usuario que tienen un significado especial para el shell Bash. Estos ficheros permiten al usuario configurar el entorno de su cuenta automáticamente cuando entra en el sistema, cuando arranca un subshell o ejecutar comandos cuando sale del sistema.

Los nombres de estos ficheros son `.bash_profile`, `.bashrc` y `.bash_logout`.

Para todos los usuarios:

/etc/profile

Para un usuario específico:

~/.bash_profile

~/.bash_login

~/.profile

El orden de carga de los scripts de inicio es el indicado, es decir primero se ejecuta: /etc/profile y luego al hacer login, por este orden: ~/.bash_profile, ~/.bash_login, ~/.profile

`.bash_profile` es el más importante de los tres. Es leído y los comandos incluidos en él ejecutados cada vez que el usuario entra en el sistema. Cualquier cambio hecho en este fichero no tendrá efecto hasta que salgamos y entremos en el sistema de nuevo.

Bash permite dos sinónimos para este fichero, `.bash_login` (derivado del C shell) y `.profile` (derivado del Bourne y Korn shell). Si `.bash_profile` no existe, el sistema buscara primero `.bash_login` y luego `.profile`. Solamente uno de estos ficheros es leído y en este mismo orden, en el caso que existan simultáneamente.

`.bashrc` es leído cuando el usuario arranca un shell interactivo, como por ejemplo escribiendo bash en la línea de comandos. Esto nos permite ejecutar diferentes comandos para la entrada al sistema o para la ejecución de un subshell. Desde aquí, por ejemplo se puede llamar al fichero `.alias` donde se almacenen los `alias` definidos por el usuario.

`.bash_logout` es el fichero leído por Bash, cuando salimos del sistema. Podemos definir, por ejemplo, que se borren los ficheros temporales creados en nuestra última sesión o registrar el tiempo que hemos estado utilizando el sistema. Si `.bash_logout` no existe, ningún comando será ejecutado a nuestra salida.

Archivo	Labor desempeñada
/etc/profile	Se ejecuta automáticamente en la conexión.
~/.bash_profile, ~/.bash_login, ~/.profile	Se ejecuta automáticamente en la conexión.
~/.bashrc	Se ejecuta automáticamente en shell interactivo.
~/.bash_logout	Se ejecuta automáticamente en la desconexión.

Resumen de archivos de arranque del shell bash.

1.3 Comando alias/unalias

El comando alias permite asignarles otros nombres a los comandos. De esta forma se pueden abreviar o llamarlos de forma más mnemotécnica.

Sintaxis: `alias [nombre[=valor]...]`

Sin argumentos muestra todos los alias definidos por el usuario actual. Para deshabilitar un alias se emplea el comando unalias.

Los alias no son heredados por los diferentes subshells por lo que si queremos que haya una herencia se deben incluir en el fichero `~/.bashrc` para un usuario concreto y si queremos hacerlo globalmente para todos los usuarios, en `/etc/bash.bashrc` (en otras distribuciones este archivo podría llamarse `/etc/bashrc`).

Ejemplos:

```
$ alias l='ls -l --color'
$ alias l # muestra qué comando ejecuta el alias definido
```

Si deseamos eliminar un comando previamente definido con alias, se utiliza el comando `unalias`.

Ejemplos:

```
$ unalias l
```

El comando anterior es equivalente a hacer

```
$ alias l=
```

1.4 Metacaracteres

Se denomina metacarácter al carácter que tiene un significado especial en el programa informático en el que se use. A diferencia del resto de caracteres, no tiene un significado literal. Comúnmente se llama metacarácter al carácter específico que actúa como **comodín**.

Los metacaracteres del shell son la interrogación ?, los corchetes [] y el asterisco.

?	Sustituye a cualquier carácter, pero sólo uno, excepto el primer punto (no nos sirve para los archivos que empiezan por “.”, es decir, los ocultos).
*	Sustituye a 0 o más caracteres, excepto un primer punto.
[] (opcionalmente junto con - y !)	Define una clase de caracteres. Dentro de esta clase se puede utilizar: • un guión, -, entre dos caracteres ASCII para poner de relieve todos los caracteres en ese rango inclusive • un signo de exclamación, !, como primer carácter para negar la clase definida.

Los caracteres para la generación de nombres de archivo son expandidos por el shell antes de ejecutar la orden. Por eso la orden `ls` tiene el mismo efecto que `echo *`.

En caso de que utilicemos como intérprete de órdenes a bash, podemos considerar adicionalmente los siguientes metacaracteres:

~	La virgulilla al comienzo de una palabra se expande con el nombre de su directorio de trabajo (directorio `home`).
~luis	Representa el directorio de trabajo (directorio `home`) del usuario luis.
~+	Representa el directorio de trabajo actual (`pwd`).
~-	Representa el último directorio de trabajo anterior al actual (`oldpwd`).

Ejemplos:

```
$ ls -d ????
Dani Mail Xini chan grub i386 lost mbox spro

$ echo ????   #si en el directorio actual hay ficheros cuyo nombre tenga
                4 caracteres exactamente, los listará por pantalla
Dani Mail Xini chan grub i386 lost mbox spro

$ ls -d [a-n]???
chan grub i386 lost mbox

$ ls -d /bin/[!a-n]*
/bin/ping /bin/stty /bin/tcsh /bin/view /bin/sort /bin/sync
/bin/true /bin/zcat
```

Se dispone también de los siguientes **comodines extendidos**:

?(patrón)	Debe coincidir **cero o una ocurrencia** de un determinado patrón
*(patrón)	Debe coincidir **cero o más ocurrencias** de un determinado patrón
+(patrón)	Debe coincidir **una o más ocurrencias** de un determinado patrón
@(patrón)	Debe coincidir con **exactamente una ocurrencia** de un determinado patrón
!(patrón)	Debe coincidir con **cualquier cosa, excepto con patrón**

Ejemplos:

```
$ shopt -s extglob # activa (set) los comodines extendidos si es necesario (pueden estar activados por defecto)

$ ls
file filename filenamename fileutils

$ ls file?(name)
file filename

$ ls file*(name)
file filename filenamename

$ ls file+(name)
filename filenamename

$ ls file@(name)
filename

$ ls file!(name) # divertido esse
file filenamename fileutils

$ ls file+(name|utils)
filename filenamename fileutils

$ ls file@(name|utils) # "lembra" um {name,utils}
filename fileutils
```

Más información:

http://mural.uv.es/oshuso/832_metacaracteres_en_linux.html

https://www.linuxjournal.com/content/bash-extended-globbing

1.5 Entrecomillado y caracteres especiales

Hay muchos caracteres en GNU/Linux que tienen significados especiales. Por ejemplo, el carácter $ se puede usar literalmente o como sustituto de las variables del shell. Puesto que no es suficiente con el contexto para determinar el significado de un carácter, es necesario tener un mecanismo que evite el significado especial y lo obligue a ser tratado simplemente como un símbolo. A este mecanismo se le denomina **entrecomillado**.

El intérprete de órdenes reconoce como caracteres especiales los siguientes:

$	Usado para la sustitución de variables.
?, [,],*	Usados para la generación de nombres de archivo.
<,>,2>,>>,2>>	Usados para la redirección de E/S.
Espacio en blanco	Usado como delimitador de argumentos.
|	Usado para interconectar procesos.
&,;,(,)	Usados para manejar procesos.
{,}	Otros propósitos.

Para tratar como caracteres de forma literal tenemos tres formas:

- Utilizar *backslash* (\).
- Utilizar comillas simples (').
- Utilizar comillas dobles (“).

→ El *backslash* elimina el significado especial de cualquier carácter especial que le siga.

→ La comilla simple hace que los caracteres especiales pierdan su significado especial (excepto la posible comilla simple).

→ Dentro de las comillas dobles, la mayoría de los caracteres especiales pierden su significado especial. Las excepciones son el símbolo $ (cuando se usa para la sustitución de variables), las comillas dobles, el *backslash* y el acentro grave (`). Se puede usar el *backslash* dentro de las comillas dobles para evitar el significado.

Ejemplos:

```
$ echo $TERM \$TERM '$TERM' "$TERM"
xterm-color $TERM $TERM xterm-color
```

1.6 Sustitución de órdenes

La sustitución de órdenes es otra característica práctica del shell. Nos permite captar la salida de una orden y asignarla a una variable, o bien usar esa salida como un argumento de otra orden. Puesto que la mayoría de las órdenes de GNU/Linux generan salida estándar, la sustitución de órdenes puede ser muy útil. Encerrando la orden entre comillas invertidas (`), conocidas también como acentos graves, captamos la salida de la orden y la asignamos a la variable del shell.

Otra forma de sustitución de órdenes es mediante $(). Dentro de los paréntesis pondremos la orden a ejecutar.

Ejemplos:

```
$ fecha=`date`
$ echo $fecha
vie may 1 15:44 CEST 2009

$ echo Hoy es `date`
Hoy es vie may 1 15:44 CEST 2009

$ fecha=$(date)
$ echo $fecha
vie may 1 15:44 CEST 2009

$ echo Hoy es $(date)
Hoy es vie may 1 15:44 CEST 2009
```

1.7 Modos de invocar una orden

El intérprete de órdenes es capaz de reconocer distintos modos de invocar una o varias órdenes. Sabemos que para ejecutar un programa simplemente tenemos que invocarlo por su nombre, pero el shell nos va a ofrecer distintas posibilidades de ejecutarlo. Éstas se indican a continuación:

`ord &`	Ejecuta la orden en segundo plano. De este modo, mientras se ejecuta la orden ord, el shell nos devuelve el control, y podremos ejecutar otros programas.
`ord1 ; ord2`	Permite ejecutar varias órdenes invocadas desde una única línea. Las distintas órdenes deben ir separadas por un punto y coma “;”.
`(ord1 ; ord2)`	Ejecuta ambas órdenes formando un único grupo.
`ord1 | ord2`	Las órdenes se van a comunicar mediante una tubería (lo veremos en el próximo tema).
`ord1 `ord2``	Sustitución de órdenes. La salida de ord2 se utiliza como argumento de ord1.
`ord1 && ord2`	Ejecuta ord1, y si finaliza con éxito, ejecuta ord2. Operación AND.
`ord1 || ord2`	Ejecuta la segunda orden ord2, sólo si la primera ord1 falla. Operación OR.

Ejemplos:

```
$ date ; sleep 10 ; date
vie may 1 15:44 CEST 2009
vie may 1 15:54 CEST 2009

$ cp && date
cp: faltan argumentos (ficheros)
Pruebe 'cp –help' para más información.

$ cp || date
cp: faltan argumentos (ficheros)
Pruebe 'cp –help' para más información.
vie may 1 15:44 CEST 2009
```

1.8 Variables

1.8.1 Definición de variables

Las variables son contenedores de información que pueden ser utilizadas por el proceso actual o cualquier subproceso que generemos.

Para definir una variable es tan simple como escribir en el intérprete de órdenes

```
VARIABLE=valor
```

Las variables en GNU/Linux diferencian mayúsculas de minúsculas. Por lo tanto, la variable que acabamos de definir es diferente de la variable `variable` o `Variable`.

Para extraer el contenido de una variable hemos de poner un **$ delante de su nombre**. Así, en el ejemplo anterior escribiríamos `$VARIABLE`.

Ejemplos:

```
$ VARIABLE=5
$ echo $VARIABLE
5

$ fecha=date
$ $fecha
vie may 1 15:44 CEST 2009
```

Por defecto, cuando definimos una variable pertenece al proceso que lo ha generado. Cualquier subproceso que generemos no tendrá definida esa variable. Si queremos que los subprocesos que generemos hereden alguna variable hemos de exportarla mediante el comando `export`.

```
export VARIABLE
```

Al mismo tiempo que declaramos una variable podemos exportarla y asignarle un valor

```
export VARIABLE=5
```

A partir de este momento, cualquier subproceso que generemos heredará la variable y su valor.

Ejemplos:

```
$ VARIABLE=5
$ echo $VARIABLE
5

$ bash
$ echo $VARIABLE

$ export VARIABLE=5
$ echo $VARIABLE
5

$ bash
$ echo $VARIABLE
5

$ export VARIABLE=5
$ echo $VARIABLE
5

$ bash
$ echo $VARIABLE
5

$ VARIABLE=27
$ echo $VARIABLE
27

$ exit
$ echo $VARIABLE
5
```

1.8.2 Comando set

El comando `set` sirve para gestionar las variables y funciones del shell. Si se ejecuta set directamente sin parámetros, nos devuelve los nombres y valores de varias funciones del shell.

```
# set
```

En bash no es necesario asignar variables mediante `set variable=valor`, sino que se puede usar directamente `variable=valor`. Sin embargo, nos podemos encontrar trabajando con otros shells, como csh en los que es obligatorio usar el set para asignar valores a las variables.

1.8.3 Variables de entorno

Una variable de entorno contiene datos utilizados por una o más aplicaciones. El valor de una variable de entorno puede ser, por ejemplo, la ruta de los ficheros ejecutables en el sistema de archivos o la configuración del sistema. Las variables de entorno proporcionan una forma simple de compartir configuraciones entre diferentes aplicaciones y procesos en Linux.

Variable	Descripción
HOME	Directorio personal.
HOSTNAME	Nombre de la máquina.
MAIL	Archivo de correo.
PATH	Lista de directorios donde buscar los programas.
PS1	Prompt.
SHELL	Intérprete de comandos por defecto.
TERM	Tipo de terminal.
USER	Nombre del usuario.

Podemos ver, crear y modificar variables de entorno. Por defecto, cuando creamos, modificamos o borramos variables de entorno del sistema, al reiniciar se pierden los cambios efectuados.

1.8.3.1 Administración de variables de entorno

Valor de una variable de entorno

El comando `printenv` devuelve el valor de una variable de entorno específica:

```
[root@LINUX1 ~]# printenv USER

root
```

también podemos hacer uso del comando `echo`, pero tendremos que anteponer el signo $ al nombre de la variable:

```
[root@LINUX1 ~]# echo $USER

root
```

Visualizar todas las variables de entorno

Para visualizar las variables de entorno definidas, podemos ejecutar los comandos `printenv` o `env`.

Crear variables de entorno

Para crear una variable de entorno, utilizaremos el comando `export`:

```
[root@LINUX1 ~]# export VARIABLEENTORNO=SYSADMIT
```

(No devuelve nada por pantalla)

Con la siguiente línea se devuelve la variable de entorno creada:

```
[root@LINUX1 ~]# printenv | grep VARIABLEENTORNO
VARIABLEENTORNO=SYSADMIT
```

(El comando grep busca patrones en ficheros y por defecto devuelve todas las líneas que contienen un determinado patrón. Es un comando muy utilizado, lo veremos con detalle más adelante)

Eliminar variables de entorno

Si quisiéramos eliminar la variable de entorno, bastaría con ejecutar:

```
[root@LINUX1 ~]# export VARIABLEENTORNO=
```

Resultado:

```
[root@LINUX1 ~]# echo $VARIABLEENTORNO
```

Configurar una variable de entorno permanente

Tendremos que establece la variable de entorno al inicio, editando alguno de los scripts de inicio que se ejecutan de forma predeterminada. Para ello, disponemos de los siguientes scripts que se cargan al iniciar (los vimos con más detalle en un apartado anterior):

Para todos los usuarios:

/etc/profile

Para un usuario específico:

~/.bash_profile

~/.bash_login

~/.profile

El orden de carga de los scripts de inicio es el indicado, es decir primero se ejecuta: /etc/profile y luego al hacer login, por este orden: ~/.bash_profile, ~/.bash_login, ~/.profile

Podemos ver las variables de entorno permanentes que se inicializan en `/etc/profile`, ejecutando:

```
cat /etc/profile|grep export
```

El fichero **preferible** para ubicar las variables de entorno, si existe en nuestro sistema, es **`/etc/environment`**.

El formato del interior de `/etc/environment` para definir las variables de entorno permanentes es el siguiente:

```
VARIABLEENTORNO=SYSADMIT
```

Debemos tener en cuenta que las reglas aquí configuradas se ejecutarán **para todos los usuarios**.

Ejemplo:

```
[root@LINUX1 ~]# ls -lF /etc/environment

-rw-r--r--. 1 root root 0 Oct 29  2014 /etc/environment
```

En este ejemplo, vemos que existe el fichero `/etc/environment` y ocupa 0 bytes, pero al existir, significa que el sistema lo procesará al inicio, por lo tanto, si lo editamos y situamos las variables de entorno allí, serán procesadas.

Esta es la forma más limpia de configurar nuestras variables de entorno permanentes a nivel de equipo, ya que la definición de las mismas quedará separada del resto de configuración definida en los scripts de inicio.

1.9 Comando history

El comando history nos permite controlar el histórico de comandos que se van almacenando en el shell. El histórico se almacena por defecto en el fichero ~/.bash_history lo que nos indica que existe un histórico diferenciado para cada usuario. Si ejecutamos el comando `history` nos devolverá la relación de comandos para ese usuario.

Algunas opciones:

-c	→Borra la lista histórica.
-w [nombrearchivo]	→Escribe la lista histórica actual en el archivo histórico (escribiendo sobre el contenido actual del archivo); si se especifica 'nombrearchivo' se utiliza como archivo histórico.

Los ejemplos siguientes muestran el uso de la orden `history` con y sin argumentos. La primera orden muestra el listado histórico completo, incluyendo la orden actual. La segunda orden muestra las cuatro últimas órdenes del listado histórico. La tercera orden borra el listado y la última se utiliza para confirmar que realmente se ha borrado el listado histórico.

Ejemplos

```
$ history
...
242 ls
243 man history
244 more ~/.bash_history
245 ps
246 history

$ history 4
244 more ~/.bash_history
245 ps
246 history
247 history 4

$ history -c
$ history
1 history
```

Indicador de suceso	Significado	Ejemplo	
!N	El evento es la orden de la línea N del listado.	`!10`	La línea es la orden 10 del listado histórico.
!-N	El evento es la orden de la línea situada N líneas antes de la actual.	`!-6`	La orden de seis líneas más atrás
!!	El evento es la orden anterior	`!!`	La última orden que hemos ejecutado
!cad	El evento es la orden más reciente que empieza por “cad”	`!grep`	La orden más reciente que empiece por la cadena “grep”
!?cad[?]	El evento es la orden más reciente que contien “cad”; si después de “cad” hay un fin de línea, sobra el ? del final	`!?cut?`	La línea más reciente que contenga la cadena “cut”

Ejemplos:

```
$ history
...
70 ps
80 echo Esto es una prueba
81 cd ~
82 cd /etc
83 more passwd
84 ls -a
85 history

$ !?etc?
cd /etc

$ ?~?
cd ~

$ 84
ls -a
.bashrc bin

$ !mo
more passwd
```

Las palabras de la orden seleccionada se numeran de tal modo que la primera (el nombre de la orden) tiene el número 0, y las demás palabras crecen consecutivamente respecto a la anterior. De este modo, la palabra que sigue al nombre de la orden tiene el número 1.

Se puede emplear un indicador de palabra para identificar la palabra o palabras deseadas dentro de la orden o evento seleccionado. El signo dos puntos (:) se inserta entre el indicador de suceso y el indicador de palabra. En la siguiente tabla encontramos dos indicadores de palabra, aunque hay más.

Indicador de suceso	Significado
N	La N-ésima palabra.
N1-N2	El intervalo de palabras desde el número N1 al N2

Ejemplos:

```
$ history
...
70 cd /usr/include
71 ps
72 echo Esto es una prueba
73 ls -li stdio.h
74 cd ~/cursos/2017/DAMW
75 ls -al
77 history

$ chown alumno:ciclos !74:1
chown alumno:ciclos ~/cursos/2017/DAMW

$ echo !72:1-3
echo Esto es una
Esto es una
```

1.10 EJERCICIOS

A. Crea una variable con nombre **MIVARIABLE** con el valor 5. Muestra el contenido de esta variable. ¿Que se mostraría si escribiéramos **"$MIVARIABLE"**? ¿Y '**$MIVARIABLE**'?

B. Crea un nuevo comando denominado **midir** que muestre un listado largo del directorio en el que te encuentras. Haz que el nuevo comando que acabas de crear sea accesible a tu usuario cada vez que inicie sesión. ¿Y si quisiéramos que estuviera accesible a todos los usuarios de nuestro sistema? (En este último caso, prueba a añadir el alias al final del fichero /etc/bash.bashrc . Para probar el alias con otro usuario sin tener que cerrar sesión, puedes ejecutar `su nombre_usuario` en el shell que estás utilizando. Para cerrar la sesión de ese usuario, escribe `exit`)

C. ¿Qué comodines utilizarías en cada uno de estos casos?:

1	Ficheros de 3 caracteres cuya primera letra sea mayúscula	[A-Z]??
2	Ficheros cuyo quinto carácter sea un número	
3	Ficheros que comiencen por *g* mayúscula o minúscula	
4	Ficheros de música MP3, cuya extensión puede contener caracteres mayúscula o minúscula	
5	Ficheros cuya extensión de 3 dígitos sea un número	
6	Ficheros que contengan una *r* mayúscula o minúscula	
7	Ficheros que contengan al menos un número	
8	Ficheros que finalicen por los siguientes caracteres y solo estos caracteres: j, m, R, p, A	
9	Ficheros de longitud 5 caracteres	
10	Ficheros de 5 caracteres formados por números	
11	Ficheros que no empiecen por un número	
12	Ficheros que no contengan números en su interior en el segundo dígito	
13	Ficheros que no tengan extensión mp3.	

D. Ejecuta un comando que muestre por pantalla el siguiente mensaje:

Hoy es el <día_del_año_eliminando_los_ceros_de_la_izquierda> día del año.

Por ejemplo: Hoy es el 5 día del año

Haz el ejercicio de dos formas diferentes.

E. En una sola instrucción verifica si el directorio **prueba** (cd prueba) existe o no y en caso de que no exista, créalo.

F. En una sola instrucción verifica si el directorio **prueba** (ls prueba) existe o no y en caso de que exista, bórralo.

G. Crea una variable denominada **CURSO** con el valor **2018-2019**. ¿Cómo harías para que esa variable estuviera accesible para cualquier proceso hijo que creáramos?

H. ¿Cómo se muestran el historial de órdenes que has ejecutado con tu usuario? Ejecuta el quinto comando anterior al actual. Ejecuta el último comando almacenado en el historial que empieza por *cd*.

2 REDIRECCIONES

En Linux, los comandos aceptan texto como entrada, realizan algún tipo de operación y producen texto como salida.

Para tener más posibilidades al aplicar los comandos de filtros que encontramos en el apartado 4, sirve de ayuda entender algunas formas de redirección que se pueden utilizar con muchos comandos.

2.1 Salida estándar

Cuando un comando se ejecuta sin errores, la salida que se produce se conoce como salida estándar, también stdout o STDOUT. Por defecto, esta salida se enviará a la terminal donde se ejecutó el comando.

Es posible redireccionar la salida estándar de un comando a un fichero en lugar de a la terminal. Para redireccionar a la salida estándar se utiliza el operador >. Si el fichero no existe lo crea y si ya existiera borra todo su contenido y lo sustituye por el nuevo.

Si lo que queremos es añadir nuevo contenido al fichero en vez de eliminar el existente utilizamos el redirector >>.

Ejemplos:

```
$ ls > fichero
$ ls >> fichero
```

Existe un fichero especial en el sistema que funciona como un sumidero. Si redireccionamos a /dev/null el contenido se pierde por completo.

Ejemplo:

```
$ ls > /dev/null
```

Si queremos redireccionar la ejecución de más de un comando a la salida estándar:

```
$ (comando1 ; comando2 ; … ; comandoN) > fichero_salida
```

2.2 Salida de error estándar

Por defecto, cuando un programa produce un error, se producirá una salida conocida como error estándar, llamada stderr o STDERR. Al igual que la salida estándar, la salida de error estándar normalmente se envía a la misma terminal donde el comando se estaba ejecutando. Por este motivo los mensajes se mezclan con los de la salida sin error del programa.

Para realizar la redirección de la salida de error, podemos utilizar cualquiera de las dos formas siguientes:

- **2>** si el fichero no existe lo crea y si existe, borra su contenido. En ambos casos se escribe el contenido de la salida de error al fichero.
- **2>>** se añade al final del archivo el contenido de la salida estándar de error. Si no existiera el archivo, se comporta de manera semejante a 2>.

Ejemplo:

```
$ ls fichero_no_existente 2> errores
```

2.3 Entrada estándar

La entrada estándar, llamada también stdin o STDIN, normalmente viene del teclado y será la entrada proporcionada por el usuario que escribe el comando. Muchos comandos pueden leer la entrada desde ficheros, pero algunos solo aceptan que el usuario la introduzca a través del teclado.

Ejemplo:

Un ejemplo en que el uso de la redirección de la entrada estándar es deseable es con el comando tr (veremos este comando con más detalle en el apartado de filtros). Este comando, entre otras acciones, cambia los caracteres en minúscula a mayúsculas. Si ejecutas el comando, escribes algún texto (`hola`) y pulsas intro, verás la traducción.

```
$ tr 'a-z' 'A-Z'

hola
HOLA
```

Este comando no para de leer de la entrada estándar a menos que se le envíe un carácter de "fin de transmisión", por ejemplo ctrl+d.

El comando tr no aceptará un fichero como argumento, pero se puede realizar la traducción del contenido de un fichero utilizando la redirección de entrada. Para utilizar la redirección de entrada, escribe el comando con sus opciones y argumento y a continuación el signo < y la ruta del fichero que se utilizará como entrada.

Por ejemplo, si el contenido de animales.txt es:

```
gato
perro
gorrión
```

Prueba lo que sucederá si ejecutas el comando:

```
$ tr 'a-z' 'A-Z' < animales.txt
```

2.4 Combinación

Si quisiéramos redireccionar la salida estándar y la salida de error a un mismo fichero, la lógica nos diría que deberíamos hacer algo parecido a lo siguiente:

```
$ comando > f1 2> f1
```

Esto provoca que el 2> f1 se pierda.

Si queremos redireccionar la salida estándar y la salida de error a la vez a un mismo fichero hemos de utilizar alguna de las siguientes formas:

```
$ comando > salida 2>&1
$ comando >> salida 2>&1
```

Si lo que queremos es redireccionar la salida estándar a un fichero y la de error a otro:

```
$ comando > salida_ok 2> salida_error
```

EJERCICIO

Puedes probar todo esto con el comando ls. Por ejemplo, en tu directorio home, crea un fichero con el nombre que quieras, por ejemplo “prueba” y utiliza también otro nombre de fichero que no exista: `ls -l prueba noExiste`

Ahora prueba a redireccionar la salida estándar y la de error y observa qué se escribe en cada fichero.

3 TUBERÍAS

Las tuberías (pipes) son un poderoso mecanismo del shell en Linux. Las tuberías permiten tomar la salida de un comando y pasársela como entrada a otro comando.

Muchos de los comandos mencionados anteriormente, que reciben como argumento un fichero, en caso de omitirse este, utilizan su entrada estándar. Esta entrada puede provenir a su vez de la salida de otros comandos. Gracias a esto las tuberías permiten realizar filtrados de los más diversos tipos

Las tuberías pueden estar formadas por un número "ilimitado"' de comandos. El carácter que se emplea para separar un comando de otro mediante una tubería es |. (Alt Gr + 1 o Alt + 124)

Ejemplo

```
$ ls -l /dev | less     #   Toma la salida de ls y la envia como
                            entrada a la orden less, con lo que se
                            consigue un listado paginado.
$ ls | sort -r # ordena de forma inversa el contenido de un directorio
$ ls | sort -r | more # las órdenes se pueden encadenar
```

Es importante tener en cuenta la diferencia entre tubería y redirección. Las tuberías separan comandos, es decir, a izquierda y derecha de una tubería hay comandos de Linux. La redirección direcciona un comando a un fichero o directorio, es decir, bien a la izquierda o bien a la derecha de una redirección no existirá un comando de Linux, sino un fichero o directorio.

Estos ejemplos por lo tanto estarían mal y son erróneos:

```
ls -lia | fichero.txt
echo hola Mundo | carta.txt
cat /etc/passwd > less
```

Tomemos por ejemplo el comando `sort`, que permite ordenar un flujo de texto. Si queremos ordenar el contenido del fichero carta.txt lo podemos hacer así:

```
sort carta.txt
```

pero también sería válido hacerlo así:

```
cat carta.txt | sort
```

lo que estaría mal sería lo siguiente:

`cat carta.txt > sort` (crearía un fichero con nombre sort y el contenido de carta.txt)

4 FILTROS

Los filtros leen texto por la entrada estándar, lo procesan y sacan la información por la salida estándar. Se pueden encadenar varios filtros.

Los filtros hacen tareas muy simples. La combinación de varios filtros permite realizar tareas muy complejas.

Los filtros permiten el uso de argumentos para modificar su comportamiento. La sintaxis suele ser la misma en la mayoría de los casos:

```
filtro [opciones] [archivos]
```

4.1 Comando cat

El comando `cat` concatena (conCATenate) ficheros y los imprime en la salida estándar. Si no se le pasa ningún argumento lee de la entrada estándar. Existe también `zcat` que hace lo mismo, pero con ficheros comprimidos.

Si solo se da el origen a `cat`, utiliza como salida la pantalla. Es decir, `cat hola` muestra por pantalla el fichero hola. Si solo se da la salida a cat, (`cat > fichero`) utiliza como entrada el teclado.

Ejemplos:

```
# cat /etc/passwd /etc/shadow
$ cat > fichero
$ cat < origen > destino
```

`cat` además permite crear archivos

Ejemplo:

```
$ cat > fichero
texto
texto
…...
<ctrl+D> - Equivale a mandar un EOF
```

Si queremos mostrar el contenido de un archivo podemos hacer un

```
$ cat < fichero
```

Podemos concatenar archivos

```
$ cat fichero1 fichero2 … ficheroN > nuevo_fichero_destino
```

Se pueden utilizar varias opciones:

-n	muestra el número de línea
-s	elimina los grupos de líneas vacías consecutivas, dejando sólo una
-E	pone un $ al final de cada línea

En general, cat se utiliza para fusionar varios archivos:

```
$ cat fichero1 fichero2 fichero3 | sort > fichero_fusion
```

4.2 Comandos more y less

Los comandos `more` y `less` paginan (dividen en páginas) uno o varios ficheros y los muestran en la terminal (pantalla). De no indicarles un fichero, paginan la entrada estándar (que se manda mediante una tubería).

Se diferencian en las facilidades que brindan. Por ejemplo, `more` es más restrictivo en cuanto al movimiento dentro del texto, mientras que `less` no limita este aspecto pues acepta el empleo de todas las teclas de movimiento tradicionales. Cuando se alcanza el final del último fichero a paginar, more termina automáticamente, no así `less`. También `more` muestra sucesivamente el porcentaje del fichero visto hasta el momento. Tanto `less` como `more` proveen una serie de comandos para moverse con facilidad dentro del texto paginado.

Ejemplos:

```
$ less /etc/passwd
$ more /etc/passwd
$ cat fichero | less
```

Algunas teclas que podemos usar mientras usamos estos programas son:

h o ?	muestra la ayuda de todas las opciones de teclado.
q	permite interrumpir el proceso y salir.
/p	realiza búsquedas del patrón p dentro del texto. Para repetir la búsqueda del mismo patrón sólo es necesario escribir /.
[n]b	en more permite regresar n páginas (por defecto n es 1).
[n]f	en more se adelantan n páginas y en less, n líneas.

El `man`, para dar formato a su salida, utiliza por defecto el paginador `less`. Existen además los comandos `zless` y `zmore` que permiten paginar con `less`

y `more` respectivamente, a los ficheros comprimidos sin necesidad de descomprimirlos previamente.

Además, con more podemos empezar a visualizar un archivo a partir de la línea que le indiquemos mediante `more +N`

```
$ more +5 fichero
```

4.3 Comandos tail y head

Los comandos tail y head muestran respectivamente el final y el comienzo (10 líneas por defecto) de uno o varios ficheros. De no especificarse al menos un fichero toman la entrada estándar.

Sintaxis:

```
tail [opciones] [ficheros]
head [opciones] [ficheros]
```

Algunas opciones:

-f	para el caso de **tail** se ejecuta de forma sostenida o sea se continúa visualizando el final del fichero hasta que se interrumpa el proceso (Ctrl-c).
-q	no coloca los encabezamiento con el nombre de los ficheros cuando se indican varios (quiet).
-<n>	imprime las n últimas (primeras) líneas en lugar de las diez establecidas por defecto.
+<n>	con **tail** se imprimen las últimas líneas del fichero empezando por la línea n, contada desde el principio

Ejemplos:

```
# tail -f /var/log/messages
# tail -20 /var/log/secure
# head -50 /var/spool/mail/pepe
# head -2 -q /etc/*.conf
```

Si queremos escoger un trozo de texto, se puede utilizar la combinación de ambas:

Ejemplo:

Imprimir las líneas de la 7 a la 18

```
$ cat fichero | tail +7 | head -12
```

Puesto que 18 -7 = 11 → 11+1 =12

4.4 Comando uniq

El comando `uniq` elimina las líneas repetidas de un fichero ordenados (lo más útil en este comando), imprimiéndolo por la salida estándar o en otro fichero argumento. De no especificarse un fichero toma la entrada estándar.

Sintaxis: `uniq [opciones] [fichero]`

Algunas opciones:

-c	utiliza como prefijo en cada línea su número de ocurrencias.
-d	sólo imprime las líneas repetidas
-u	sólo imprime las líneas no repetidas

Ejemplo:

```
$ uniq -d lista.txt
```

4.5 Comando wc

El comando `wc` imprime el número de líneas, palabras y bytes de uno o varios ficheros. Si son varios ficheros hace también un resumen de los totales. De no especificarse un fichero toma la entrada estándar.

Sintaxis: `wc [opciones] [ficheros]`

Algunas opciones:

-l	sólo cuenta líneas.
-c	sólo cuenta bytes.
-w	sólo cuenta palabras.

Ejemplos:

```
$ wc -l /etc/passwd
$ wc -w /doc/dicciorario.txt
```

4.6 Comando nl

Numera las líneas, pero sólo aquellas que contienen texto, pero no las que contengan retornos de carro. Por lo tanto, `nl` es distinto a utilizar `cat -n` puesto que este último numeraría todas las líneas.

Ejemplo:

```
$ nl /etc/passwd
$ cat fichero | nl
```

4.7 Comando sort

El comando sort ordena las líneas de un fichero mostrándolas por la salida estándar. De no especificarse un fichero toma la entrada estándar.

Sintaxis: `sort [opciones] [fichero]`

Algunas opciones:

-r	ordena al revés.
-f	trata las mayúsculas y minúsculas por igual.
-g	ordena de forma numérica, de modo que no es necesario que los números se rellenen con ceros por la izquierda.

Ejemplo:

```
$ sort -f /etc/passwd
```

EJERCICIO:

Crea un archivo llamado lista-desordenada con el editor nano, por ejemplo, y escribe dentro 5 nombres desordenados (uno por línea). Comprueba que con la orden

```
sort lista-desordenada > lista-ordenada
```

creamos un fichero llamado lista-ordenada que contiene la lista ordenada.

4.8 Comando cut

El comando `cut` nos permite cortar una línea de texto, para obtener un subconjunto en lugar de la línea completa. Podemos cortar por número de caracteres, por campos de una tabla, etc.

Normalmente los campos están separados por espacios o tabulaciones, aunque se puede utilizar cualquier separador. Por defecto, si no se dice nada, el separador es la tabulación.

Sintaxis: `cut -clista [ficheros]`

`cut -flista [-dcar] [-s] [ficheros]`

Con **-c** se buscan caracteres.
Con **-f** se buscan campos de una tabla.
Los campos se empiezan a contar desde 1.

Algunas opciones:

Para caracteres

-cN-M	corta desde el carácter número N hasta el carácter número M.
-cN-	corta desde el carácter número N hasta el final
-c-N	corta desde el principio hasta el carácter número N
-cN,M	corta el carácter número N y el carácter número M

Para campos

-d":"	utiliza como delimitador de campos los dos puntos.
-d"-"	utiliza como delimitador de campos el guion.
-s	no mostrar las líneas que no tengan el carácter delimitador.
-f1	nos muestra sólo el primer campo
-f1,3,5	nos muestra sólo los campos 1, 3 y 5.
-f1-7	nos muestra sólo los campos del 1 al 7.

Ejemplos:

```
$ cut -c 3-9 /etc/passwd
$ cut -d: -f4 /etc/passwd
```

4.9 Comando paste

Concatena horizontalmente (línea a línea) los archivos que se le pasan como parámetro.

Sintaxis: `paste [-dlista] [ficheros]`

Si no se utiliza el parámetro -d, se utiliza como carácter separador el tabulador. Después de -d, se espera una lista de caracteres separadores (pueden ser todos iguales o no).

Ejemplo

```
$ paste -d":-=" fichero1 fichero2 fichero3 fichero4 fichero5
```

Como caracteres separadores entre los contenidos de los ficheros del ejemplo anterior se utilizaría:

- : entre el fichero1 y fichero2
- - entre el fichero2 y el fichero3
- = entre el fichero 3 y el fichero4
- como hay más ficheros y no quedan caracteres separadores, se utiliza el último, en este caso el carácter =.

4.10 Comando grep

El comando grep (Globally Regular Expressions Pattern) busca patrones en ficheros. Por defecto devuelve todas las líneas que contienen un patrón (cadena de texto) determinado en uno o varios ficheros. Utilizando las opciones se puede variar mucho este comportamiento. Si no se le pasa ningún fichero como argumento hace la búsqueda en su entrada estándar.

```
Sintaxis: grep [opciones] <patrón> [ficheros]
```

Algunas opciones:

-c	devuelve sólo la cantidad de líneas que contienen al patrón.
-i	ignora las diferencias entre mayúsculas y minúsculas.
-H	imprime además de las líneas, el nombre del fichero donde se encontró el patrón. Es así por defecto cuando se hace la búsqueda en más de un fichero.
-l	cuando son múltiples ficheros sólo muestra los nombres de aquellos donde se encontró el patrón y no las líneas correspondientes.
-v	devuelve las líneas que no contienen el patrón.
-n	imprime el número de cada línea que contiene al patrón.

Ejemplos:

```
$ grep linux /usr/share/doc
$ grep root  /etc/passwd
# grep -n error /var/log/messages
$ grep -i pepe /etc/passwd
$ grep -c root /etc/group
$ ls -lia | grep "carta roja"
```

Expresiones regulares

^	principio de línea.
$	final de línea.
.	equivale a ? del shell. Representa uno y sólo un carácter. Ejemplo: a...b serían palabras que contengan al menos 5 caracteres que empiecen por a y terminen por b.
[abc]	representa un sólo carácter como en el shell.
[^abc]	Cualquier carácter que no sea el especificado.
[A-Z]	rango.
r*	0 o más ocurrencias de r. Por ejemplo cor*al haría referencia a las palabras coal, coral, corral, corrral, etc. Siempre cogerá la posibilidad más larga de las posibles.
\<cadena	palabras que comienzan por cadena

cadena\>	palabras que finalizan en cadena.
r\{n\}	El elemento precedente concuerda exactamente n veces.
r\{n,\}	El elemento precedente concuerda n o más veces.

Extendidas (con el parámetro -E)

cad1|cad2	cualquiera de las dos cadenas
r+	1 o más ocurrencias de r
()	Agrupa expresiones

Ejemplos:

```
$ grep "^rosa" poemas # poemas que empiecen por rosa
$ grep "rosa$" poemas # poemas que finalicen por rosa
$ grep "^rosa$" # líneas que contengan SÓLO esa palabra y ninguna otra
$ grep '^[A-H]' alumnos # líneas que empiezan por una letra mayúscula.
$ grep 'u\{2\}' /etc/passwd # líneas que contienen dos caracteres u seguidos
```

4.11 Comando tr

El comando `tr` (translator) lee los datos de la entrada estándar, los procesa y deposita los resultados en la salida estándar. Es el único filtro de los vistos que no puede leer los datos de un fichero.

Sintaxis: `tr [-dsc] cadena1 cadena2`

Hace una traducción de carácter a carácter de la cadena1 en la cadena2. Pero, ¿qué ocurre si cadena1 tiene una longitud diferente de cadena2.

- Si cadena1 > cadena2, el último carácter de cadena2 se repite
- Si cadena1 < cadena2, el resto de caracteres de cadena2 que no tiene pareja con cadena1 se desprecian.

Ejemplos:

Para probar los siguientes ejemplos, crea un fichero que contenga el contenido que ves tras hacer el cat siguiente.

```
$ cat fichero
Este es un archivo de texto
QUE CONTIENE LETRAS MAYUSCULAS Y minusculas

$ tr A-Z a-z < fichero
este es un archivo de texto
que contiene letras mayusculas y minusculas
```

```
$ tr a-z A-Z < fichero
ESTE ES UN ARCHIVO DE TEXTO
QUE CONTIENE LETRAS MAYUSCULAS Y MINUSCULAS

$ tr A-Z x < fichero
xste es un archivo de texto
xxx xxxxxxxx xxxxxx xxxxxxxxxx x minusculas
```

Opciones:

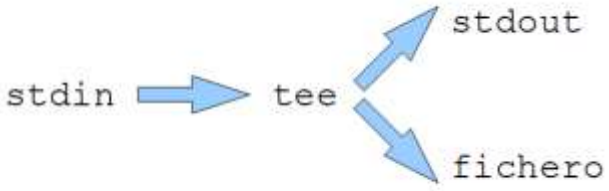
-d	elimina los caracteres de un fichero
-s	elimina caracteres repetidos
-c	complemento de un patrón de caracteres

Ejemplos:

```
$ tr -d A-Z < fichero
ste es un archivo de texto
     minúsculas

$ cat fichero1
Aqqquiiii       hhaaayyyyyyy       ccccccaaaaarrraacccttteeeerreeesss
rrreeeepppeettidos

$ tr -s a-z < fichero1
Aqui hay caracteres repetidos

$ tr -c a-z " " < fichero
 ste es un archivo de texto                                    minusculas

$ tr -c A-Z " " < fichero
E                          QUE CONTIENE LETRAS MAYUSCULAS Y

$ls -l | tr -s " "| cut -d" " -f3,4
```

4.12 Comando tee

Se utiliza para bifurcar una tubería. Lee por la entrada estándar y escribe el resultado en la salida estándar y en la tubería. Para terminar el tee, pulsa CTRL+C.



Sintaxis: `tee [-a] ficheros`

Opción

-a añade los datos en el archivo en vez de sobrescribirlo.

Ejemplo:

```
$ ls -l | tee dirlist | wc -l
```

En este ejemplo, la entrada del comando `tee` es el resultado del comando `ls -l`. Este listado de ficheros se escribirá en el fichero dirlist (prueba a hacer `cat dirlist`), pero no aparecerá por pantalla (si quitamos la segunda tubería y el comando `wc -l`, sí que aparecerá). El comando `wc -l` contará el número de líneas que le pasa el comando `tee` a través de la tubería.

4.13 Comando sed (ampliación)

sed es un editor con muchas opciones posibles, que permite realizar transformaciones en un fichero de entrada (o en texto introducido a través de la entrada estándar).

Su sintaxis es:

```
sed comandos_sed archivo
```

Puedes encontrar más información y ejemplos aquí: https://www.americati.com/doc/sed/sed.html

5 ANEXO I: VARIABLES DE ENTORNO

A continuación, se recoge una lista de variables reservadas por el intérprete de comandos. Todas ellas tienen un significado especial para el mismo, algunas de ellas sólo se pueden leer, a otras se le asignan ciertos valores automáticamente y algunas pierden su significado si le cambiamos los valores que tienen por defecto.

Variable	Explicación
CDPATH	Una lista de directorios separados por el signo ':' usada como ruta de acceso por el comando cd
HOME	El directorio principal de usuario
IFS	Una lista de caracteres para separar campos; usado cuando el intérprete de comandos separa palabras como parte de una expansión.
MAIL	Si este parámetro tiene un fichero definido y la variable MAILPATH no está definida, bash informa al usuario de la llegada de correo al fichero especificado.
MAILPATH	Una lista de ficheros separada por comas, en los cuales el intérprete de comandos comprueba periódicamente de la llegada de correo.
OPTARG	El valor del último argumento procesado por getopts.
OPTIND	El índice del último argumento procesado por getopts
PATH	Una lista de directorios, separados por comas, en los cuales el intérprete de comandos busca por comandos
PS1	Prompt principal. El valor por defecto es “'\s-\v\$ '
PS2	El prompt secundario. El valor por defecto es '> '
auto_resume	Esta variable controla como el intérprete de comandos interacciona con el control de usuario y trabajos/procesos
BASH	La ruta de acceso completa usada para ejecutar la instancia actual de bash
BASH_ENV	Si esta variable está definida cuando bash es llamado para ejecutar un script, su valor es expandido y usado como el nombre del fichero leído antes de ejecutar el script.
BASH_VERSION	El número de versión de bash usada
BASH_VERSINFO	Una matriz de solo lectura con información sobre la versión de bash usada.

Variable	Explicación
COLUMNS	Usada por 'select' para determinar el ancho de la terminal cuando imprime listas de menús.
COMP_CWORD	Un índice en ${COMP_WORDS} de la palabra conteniendo la posición del puntero actual
COMP_LINE	La línea de comando actual
COMP_POINT	El índice de la posición relativa del puntero actual con respecto al comienzo del comando actual
COMP_WORDS	Una matriz con las palabras individuales en la línea de comando actual
COMPREPLY	Una matriz de donde bash lee las palabras posibles generadas por una función del intérprete de comandos usada por la utilidad de generación de términos posibles.
DIRSTACK	Una matriz que contiene los contenidos actuales del stack de directorios
EUID	El identificador numérico de usuario del usuario actual
FCEDIT	El editor usado por defecto por la opción -e del comando 'fc'
FIGNORE	Una lista separada por comas de sufijos a ignorar cuando se efectúa la generación de posibles nombres de ficheros.
FUNCNAME	El nombre de la función que se está ejecutando actual
GLOBIGNORE	Una lista separada por comas de los patrones que definen el conjunto de nombres de ficheros a ignorar cuando se efectúa la generación de posibles nombres
GROUPS	Una matriz que contiene la lista de los grupos a que pertenece el usuario actual.
HISTCMD	El índice del comando actual en la historia de comandos
HISTCONTROL	Define si un comando es añadido a la historia de comandos
HISTFILE	El nombre del fichero en el cual se graba la historia de comandos de comandos. El valor por defecto es ~/.bash_history.
HISTFILESIZE	El número máximo de líneas contenidas en la historia de comandos, por defecto 500
HISTIGNORE	Una lista separada por comas de los patrones usados para definir que comandos deben de grabarse en la historia de comandos
HISTSIZE	El máximo número de comandos a recordar en la historia de comandos, por defecto 500

Variable	Explicación
HOSTFILE	Contiene el nombre de un fichero en el mismo formato que /etc/hosts que debería de usarse cuando el intérprete de comandos necesita completar un nombre de máquina (hostname)
HOSTNAME	El nombre de máquina actual
HOSTTYPE	Cadena describiendo la máquina que está ejecutando Bash
IGNOREEOF	Controla la acción a tomar cuando el intérprete de comandos recibe un carácter EOF
INPUTRC	Nombre del fichero de inicialización de 'Readline', sobrescribiendo el valor por defecto /etc/inputrc.
LINES	Usada para determinar la anchura de la columna usada para imprimir listas
MACHTYPE	Cadena describiendo el tipo de sistema que está ejecutando Bash
MAILCHECK	Frecuencia de comprobación (en segundos) del correo electrónico en el fichero definido en las variables MAILPATH o MAIL
OLDPWD	Directorio previo definido por el comando 'cd'
OSTYPE	Cadena describiendo el sistema operativo que está ejecutando Bash
PPID	El número de proceso del proceso padre del intérprete de comandos
PS3	El valor de esta variable se usa como 'prompt'
PWD	Directorio actual definido por el comando 'cd'.
RANDOM	Cuando se llama esta variable un numero entero entre 0 32767 es generado.
SECONDS	Numero de segundos desde que Bash fue arrancado.
SHELL	Intérprete de órdenes que se está ejecutando
SHELLOPTS	Lista con opciones de Bash activadas.
UID	El valor numérico real del usuario actual.

6 BIBLIOGRAFÍA

- Introducción a Linux: https://es.wikibooks.org/wiki/Introducci%C3%B3n_a_Linux
- Comandos: https://www.guia-ubuntu.com/index.php?title=Comandos
- El shell: https://www.guia-ubuntu.com/index.php/Terminal
- Expresiones regulares: https://enavas.blogspot.com.es/2008/03/linux-expresiones-regulares.html