

Clase 2

- esta clase contiene
- > rutas (relativa y absoluta)
 - > comandos para manipular archivos y directorios
 - > Sistemas de archivos Unix (root, wr, ...)
 - > usuarios y grupos (permisos)
 - > apropos (buscar comando con keyword)
 - > man (documentación de comando)

ls	list
mkdir	make dir
cd	change directory
rm	borra archivos o directorios
mv	move (o renombra)
ln	crea enlace a archivo
touch	crea o actualiza archivo

Pwd first working directory

Unix y Redes

LFIS-228

Rutas

Dos tipos de rutas:

1. Absoluta
2. Relativa

Rutas

1. Absoluta:

/directorioA/directorioB/etc

El primer carácter es un "slash",
marca la raíz del sistema de
archivos

El último elemento, puede ser un
archivo o un directorio

- Si es archivo, se dice que es la ruta al archivo (path to file)
- Si es un directorio, se dice que "la ruta" (path)
- Si se trata de la ruta, se puede hacer explícita con un "/" al final

Rutas

Absoluta

/directorioA/directorioB/etc/

El primer carácter es un "slash",
marca la raíz del sistema de
archivos

Sólo válido para un
directorio.

El último elemento, puede ser un
archivo o un directorio

- Si es archivo, se dice que es la ruta al archivo (path to file)
- Si es un directorio, se dice que "la ruta" (path)
- Si se trata de la ruta, se puede hacer explícita con un "/" al final

Rutas

2. Relativa

directorio1/directorio2/etc

- No empieza con un slash
- El punto de inicio es el directorio “actual”
- El directorio actual se conoce con el comando “pwd”

Rutas

Hay dos directorios especiales:

1. "`.`"
2. "`..`"

Rutas

“.”: Cada directorio contiene una ruta relativa hacia sí misma.

“cd .” te lleva al mismo directorio

“cd ../../../../../../” qué hará?

Rutas

1. "..": Cada directorio tiene una ruta relativa hacia su padre

"cd .." te lleva al directorio anterior, "el padre"

El padre de la raíz, es la raíz!

Comandos para manipular archivos y directorios

- ls → lista archivos
- mkdir → crea un directorio
- cd → cambia de directorio
- rm → borra archivo (o directorio)
- mv → mueve (o renombra)
- ln → crea enlace a archivo
- touch → crea o actualiza un archivo

ls - list directory contents

ls [OPTION]... [FILE]...

ls

ls -a -> muestra "todos" los archivos

ls -l -> muestra en formato lista

ls -lh -> formato legible para el humano

ls -d -> lista el directorio sin entrar

ls -t -> ordena por fecha

Hay muchas más opciones: "man ls"

ls - list directory contents

```
[root@kosmos ~]# ls -l
```

permisos		tamaño	
	total 56996		
-rw----- 1 root root	378963 Nov 30 2017 anaconda-ks.cfg		
-rwxr-xr-x 1 root root	413 Nov 30 2017 apagar_hosts.sh		
lrwxrwxrwx 1 root root	10 May 30 09:09 data -> /data/root		
drwxr-xr-x 2 root root	4096 Aug 10 09:52 Desktop		
-rwxr-xr-x 1 root root	344 Jun 18 10:08 encender_hosts.sh		
-rw-r--r-- 1 root root	155 Nov 30 2017 findlastaccessed.sh		
-rw-r--r-- 1 root root	176 Nov 30 2017 findlastfile.sh		
...			

mkdir - make directories

`mkdir [OPTION]... DIRECTORY...`

`mkdir -p`

La opción “-p” sirve para crear múltiples
subdirectorios

cp - copy files and directories

cp [OPTION]... [-T] SOURCE DEST

Te permite tratar a "DEST" siempre como un archivo, aunque sea directorio.

cp [OPTION]... SOURCE... DIRECTORY

Forma "canónica"

cp [OPTION]... -t DIRECTORY SOURCE...

Te permite poner el destino al principio del comando, siendo los últimos argumentos los archivos "fuente"

cp -r

(para copiar directorios, y contenido de forma recursiva)

cp -i

Interactivo, preguntará antes de sobreescibir

cp -f

Forzado, no hará preguntas, intentará sobreescibir

cp – comportamiento por defecto (archivo)

cp archivo /ruta/x

si **x** no existe, **archivo** se copia con el nombre **x** en el directorio **/ruta**

si **x** existe y es **archivo**, se sobreescribe **x**

si **x** existe y es directorio, se copia **archivo** dentro de **x**

/ruta/x/archivo

cp – comportamiento por defecto (directorio)

`cp -r directorio /ruta/x`

si **x** no existe, directorio se copia con el nombre **x** en el directorio /ruta

si **x** existe y es archivo, cp falla

si **x** existe y es directorio, se copia directorio dentro de **x**

`/ruta/x/directorio`

mv - move (rename) files

mv [OPTION]... [-T] SOURCE DEST

Te permite tratar a "DEST" siempre como un archivo, aunque sea directorio.

mv [OPTION]... SOURCE... DIRECTORY

Forma "canónica"

mv [OPTION]... -t DIRECTORY SOURCE...

Te permite poner el destino al principio del comando, siendo los últimos argumentos los archivos "fuente"

mv -i

Interactivo, preguntará antes de sobreescribir

mv -f

Forzado, no hará preguntas, intentará sobreescribir

mv – MOVER/RENOMBRAR

`mv origen destino`

Significa que "origen" desaparecerá
"destino" tendrá el contenido de "origen"

mv – comportamiento por defecto

mv archivo /ruta/x

si **x** no existe, archivo se mueve con el nombre **x** en el directorio /ruta

si **x** existe y es **archivo**, se sobreescribe **x**

si **x** existe y es directorio, se mueve **archivo** dentro de **x**

/ruta/x/archivo

El sistema de archivos

El sistema de archivos

/ directorio raíz	
/bin	Contiene programas ejecutables básicos para el sistema.
/boot	Contiene los ficheros necesarios para el arranque del sistema.
/dev	Contiene los ficheros correspondientes a los dispositivos: sonido, impresora, disco duro, lector de cd/dvd, video, etc.
/etc	Contiene ficheros y directorios de configuración.
/home	Contiene los directorios de trabajo de los usuarios. Cada usuario tiene su propio directorio en el sistema dentro de /home.
/lib	Contiene las librerías compartidas y los módulos del kernel.
/media	Dentro de este directorio se montan los dispositivos como el CD-ROM, memorias USB, discos duros portátiles, etc.
/opt	Directorio reservado para instalar aplicaciones.
/sbin	Contiene los ficheros binarios ejecutables del sistema operativo.
/srv	Contiene datos de los servicios proporcionado por el sistema.
/tmp	Directorio de archivos temporales.
/usr	Aquí se encuentran la mayoría de los archivos del sistema, aplicaciones, librerías, manuales, juegos... Es un espacio compartido por todos los usuarios.
/var	Contiene archivos administrativos y datos que cambian con frecuencia: registro de errores, bases de datos, colas de impresión, etc.
/root	Directorio de trabajo del administrador del sistema (usuario root).
/proc	Aquí se almacenan datos del kernel e información sobre procesos.

- Estos directorios son “escribibles” solamente por el usuario “root”

Usuarios y grupos

Pero antes...

Una pequeña observación sobre los "permisos" de un archivo:

- Existen 3 permisos principales para un archivo
 - Lectura → R *Read*
 - Escritura → W *write*
 - Ejecución → X *execute*

Pero antes...

Y cada permiso se puede aplicar a tres
ENTIDADES:

- Usuario (U)
- Grupo (G)
- Otros (O)

Pero antes...

El comando "ls -l" nos entrega esta info:

```
drwxr-xr-x    1 root root 2592 Aug 13 15:06 bin
```

Los permisos en verde: Usuario (dueño)

Los permisos en amarillo: Grupo

Los permisos en rojo: Otros usuarios

***LOS DIRECTORIOS DEBEN SER EJECUTABLES PARA PODER
ACCEDER A ELLOS

Pero antes...

El comando "getfacl" también nos muestra:

```
$ getfacl .
# file: .
# owner: edgar
# group: edgar
user::rwx
group::r-x
other::r-x
```

chmod

chmod - change file mode bits

chmod [OPTION]... MODE[,MODE]...
FILE...

chmod

El "MODE" en el comando "chmod" puede ser de dos tipos. Ahora veremos el modo de la forma:

[ugo][+][-][rwx]

Ej:

chmod u+x nombre_archivo

chmod g-x nombre_archivo

chmod -R o-w nomre_archivo

secreto (a todo, los directorios, y archivos, dentro (incluyendo))

Los usuarios

Los sistemas operativos tipo UNIX son
"multiusuarios"

Múltiples usuarios pueden hacer uso del
sistema en forma simultánea

Un usuario "inicia sesión" con el sistema

Usuarios

Cada usuario dentro del sistema tiene un “código identificador de usuario” (UID), el cual es un número

También cada usuario está identificado con un nombre (por favor, sólo minúsculas)

Usuarios

```
Ubuntu 16.04 Xenial Xerus localhost tty1
localhost login: usuario
Password:
Welcome!
[usuario@localhost ~] $
```

Usuarios

Existen dos tipos de usuario fundamentales

Usuario "estándar"

Super-usuario (root)

Usuarios

El usuario estándar es un usuario limitado:

Sólo tiene permisos de escribir dentro de su directorio personal

Puede ejecutar programas, menos los ubicados en /sbin (o /usr/sbin)

Otras restricciones impuestas por el administrador de sistema

Usuarios

O mejor dicho aún, el usuario estándar está restringido por las reglas impuestas por el administrador del sistema

Usuarios

El super usuario (root)

Tiene UID 0 (cero)

Es el amo y señor del sistema

Puede modificar todo, y destruir todo (y sin querer)

Usuarios

Observando el árbol de directorios (ls -l /):

```
drwxr-xr-x    1 root root 2592 Aug 13 15:06 bin  
drwxr-xr-x    5 root root 3072 Jul 23 10:40 boot  
drwxr-xr-x    1 root root     0 Aug 21  2017 cdrom  
drwxr-xr-x   21 root root 4560 Sep 11 13:27 dev  
drwxr-xr-x    1 root root 5214 Sep 10 09:13 etc  
drwxr-xr-x.  10 root root 4096 Aug  9 16:47 home  
...
```

Todo pertenece a "root"!!

Comandos que afectan a los usuarios

Sudo

su	change user ID or become superuser
id	print real and effective user and group IDs
useradd	create a new user or update default new user information
usermod	modify a user account
userdel	delete a user account and related files
sudo	execute a command as another user

passwd

{ crear contraseña para user actual

man

{ ~~se~~ documentación sobre un comando

apropos; buscar palabra entre comandos

Comando que afectan a los usuarios

```
$ id  
uid=1000(edgar) gid=1000(edgar)  
groups=1000(edgar),4(adm),24(cdrom),27(sudo),30(dip),46  
(plugdev),113(lpadmin),130(sambashare),133(vboxusers),1  
39(libvirtd)  
$
```

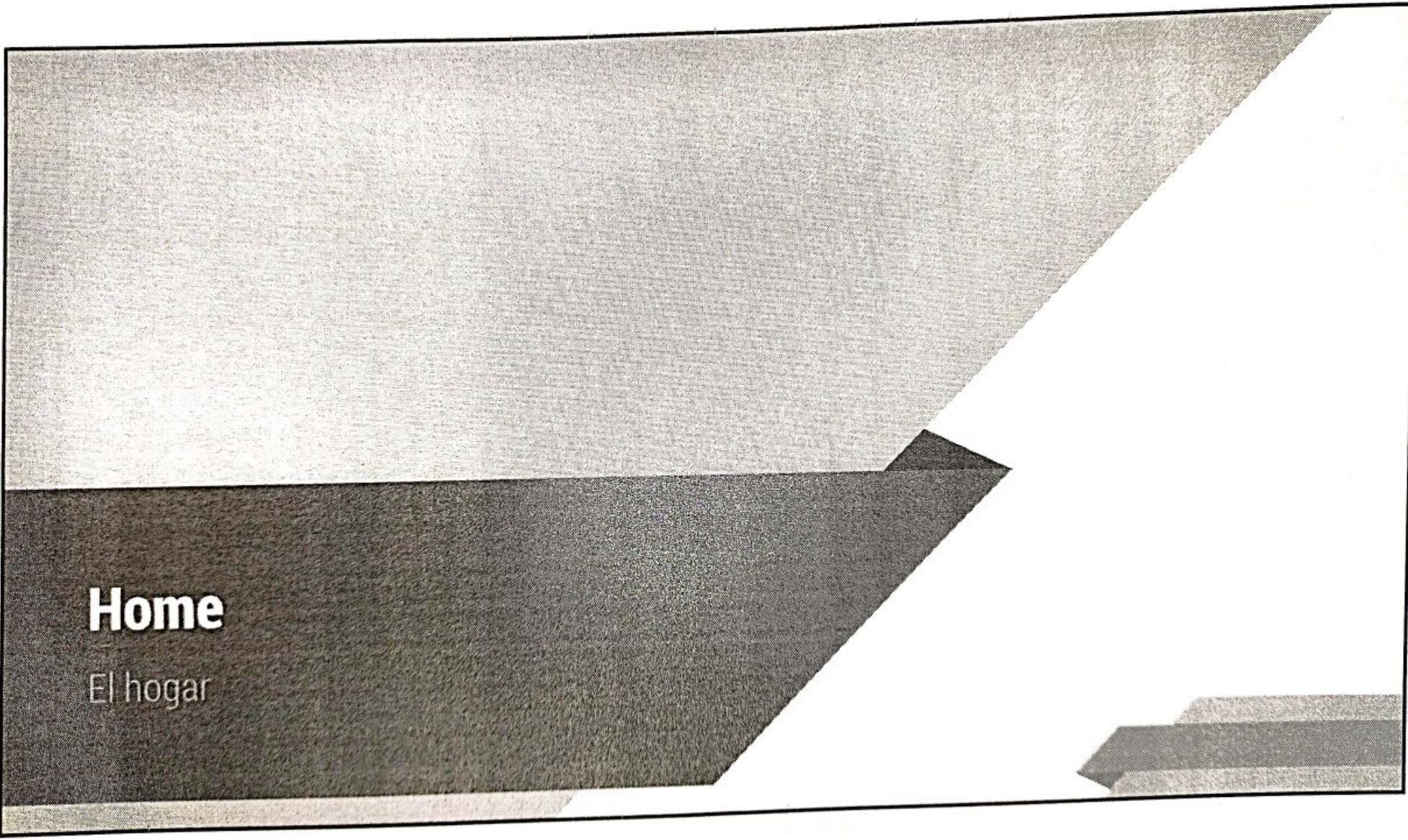
Grupos

Los grupos en UNIX permiten agrupar usuarios

De esta forma, se pueden compartir recursos
entre los miembros de un mismo grupo

```
drwxr-xr-x    1 root root 2592 Aug 13 15:06 bin
```

Un usuario puede pertenecer a múltiples grupos



Home

El hogar

Home

¿Dónde empiezo?

```
[usuario@localhost] $ pwd  
/home/usuario
```

El directorio "/home", por defecto, es el que contiene los directorios de inicio para cada usuario.

Por defecto, el directorio de un usuario tiene el mismo nombre que el nombre de usuario

Home

```
Ubuntu 16.04 Xenial Xerus localhost tty1
```

```
localhost login: usuario
```

```
Password:
```

```
Welcome!
```

```
[usuario@localhost ~] $ pwd
```

```
/home/usuario
```

```
[usuario@localhost ~] $ cd /usr
```

```
[usuario@localhost /usr] $ ls ~
```

- La tilde “~” es un atajo para el directorio home
- También lo suele ser “\$HOME”
- En Linux con teclado español, AltGr + ñ

Home

```
$ ls -a ~  
.. .bash_history .bash_logout .bash_profile .bashrc .emacs .gnome2  
.kshrc .mozilla .ncbirc .ssh .t_coffee  
$
```

Home

.bashrc	->	comandos a ser ejecutados al ejecutar bash
.bash_profile	->	comandos a ser ejecutados al hacer login
.bash_logout	->	comandos a ser ejecutados al cerrar la sesión
.bash_history	->	historial de comandos ejecutados por el usuario

Los demás archivos escondidos en el Home suelen ser configuraciones para distintos programas. Lo mismo con los directorios escondidos

Estos archivos son básicos al momento de crear un usuario cuyo shell será bash

Los "templates" se encuentran en /etc/skel

Comando

which ~~COMMAND~~

dira donde se encuentra el comando cuando lo ejecutes

— ejecutar bash y source

usage (bash archivo.sh
el archivo.sh (if executable))

se inicia nueva instancia de BASH

el cual ejecuta comandos dentro del
script y return \$?

al interprete original

source (source archivo.sh)
• archivo.sh

analogo a escribir los comando en
el interprete •
no hay output

> Variables internas

- \$ - ultimo argumento del ultimo comando usado
- \$? salida del ultimo comando
- \$! n° de proceso del shell
- \$! ~ de todos del ultimo comando en el background
- \$0 nombre del comando
- \$n Argumento en linea de comando (\$1, \$2, ...)
- \$*, \$@ Todo lo siguiente en la linea de comando (\$1 \$2 \$3 ...)
- \$@ si.
- \$PATH indica los directorios de un Programa (comandos son programas)

Clase 3

esta clase contiene

- > comodines (`?`, `[1-9]`, `[a-z]`, `[!-@]`, ...)
- para patrones que no son distintos `{...}` or `{[1-3], [5-6]} = 1,2,3,5,6`
- > atajo a home (~)
- > bash (comando shell) el bash (tambien es un comando)
- > variables entorno
 - ↳ `VARNAME = "hola"`
 - ↳ `echo VARNAME`
- > export `VAR=...` > crea una variable para todo el entorno (seguira a pesar de iniciar un nuevo bash)
- > variables internas

Comodines

wildcards

- También llamado "globbing" o "glob"
- Es una forma de englobar, o sea, nombrar muchos archivos usando caracteres especiales

comodines

```
~$ ls -a
.           .cache       .ssh          ejercicio      test2
..          .init.sh.swp .sudo_as_admin_successful ejercicio1.py testing
.bash_history .lessht     .viminfo      ejercicio2.py
.bash_logout   .profile    __pycache__  init.sh
.bashrc        .python_history a
```

- Y si quisiéramos mostrar los archivos que empiezan con "ejercicio"?
- O los que terminan con "o"?

comodines

■ Comodín "?"

- Coincide con cualquier carácter

■ Comodín "*"

- Coincide con cualquier cadena de caracteres (incluso vacíos)

Comodines

```
~/test$ ls -1
bucarest
cafest
diabetest
endulzantest
fast
fest
ketchup
last
pest
pruebasuperada
rest
super8
superman
test
```

```
~/test$ ls -1 ?est
fest
pest
rest
test
~/test$ ls -1 ??st
fast
fest
last
pest
rest
test
```

```
~/test$ ls -1  
bucarest  
cafest  
diabetest  
endulzantest  
fast  
fest  
ketchup  
last  
pest  
pruebasuperada  
rest  
super8  
superman  
test
```

```
~/test$ ls -1 *st  
bucarest  
cafest  
diabetest  
endulzantest  
fast  
fest  
last  
pest  
rest  
test
```

```
~/test$ ls -1
bucarest
cafest
diabetest
endulzantest
fast
fest
ketchup
last
pest
pruebasuperada
rest
super8
superman
test
```

```
~/test$ ls -1 *up*
ketchup
pruebasuperada
super8
superman

~/test$ ls *up
ketchup

~/test$ ls -1 f*
fast
fest
```

- Y queda el comodín [...], llamado "clase de caracteres"
- El comodín "?" coincide con cualquier carácter, pero... ¿si quisiéramos seleccionar qué caracteres coincidir?

```
~/test$ ls -1 ?est  
fest  
pest  
rest  
test
```

```
~/test$ ls -1 [pr]est  
pest  
rest  
~/test$ ls -1 [bucar]est  
rest  
~/test$ ls -1 *rest  
bucarest  
rest
```

Lo que esta entre [] representa a un solo caracter

- Los corchetes también aceptan "rangos"
 - [a-z] = [abcdefghijklmnopqrstuvwxyz]
 - [A-Z] = [ABCDEFGHIJKLMNOPQRSTUVWXYZ]
 - [0-9] = [0123456789]
- El carácter "-" separa el rango, pero debe estar entre dos caracteres.
- Si se busca coincidir "-", debe ponerse al principio o al final de la expresión en corchetes.

■ Podemos combinar los rangos:

▷ $[a-zA-Z0-9]$ = (algo muy largo...)

■ ¿Alguna inquietud? ¿Duda?

▷ $[?*-]$

▷ $[\backslash]$

▷ ¿Rangos?

▷ man glob

con $\{,\}$ podemos

poner dos patrones

en Hunter

$\{A, B\}$ primero - A 1º en pos B

$\{[0,1], [3,4]\} = 0,1, 3,4$

Comodines

■ Combinados!

```
~/test$ ls -1 [fkl]*  
fast  
fest  
ketchup  
last
```

```
~/test$ ls -1  
bucarest  
cafest  
diabetest  
endulzantest  
fast  
fest  
ketchup  
last  
pest  
pruebasuperada  
rest  
super8  
superman  
test
```

```
~/test$ ls -1 *[rf]est  
bucarest  
cafest  
fest  
rest
```

```
~/test$ ls -1 *[an8]  
pruebasuperada  
super8  
superman
```

```
~/test$ ls -1 *[rf]?st  
bucarest  
cafest  
fast  
fest  
rest
```

Home

El hogar

“ ■ OJO:

- Cuando digo “el directorio home”, no me refiero a /home, me refiero al directorio de inicio del usuario (ej., /home/usuario)

¿Dónde empiezo?

- [usuario@localhost] \$ pwd
 - ▷ /home/usuario
- El directorio "/home", por defecto, es el que contiene los directorios de inicio para cada usuario.
- Por defecto, el directorio de un usuario tiene el mismo nombre que el nombre de usuario

Ubuntu 16.04 Xenial Xerus localhost tty1

localhost login: usuario

Password:

Welcome!

[usuario@localhost ~] \$ pwd

/home/usuario

[usuario@localhost ~] \$ cd /usr

[usuario@localhost /usr] \$ ls ~

- La tilde “~” es un atajo para el directorio home
- También lo suele ser “\$HOME”
- En Linux con teclado español, AltGr + ñ

```
$ ls -a ~  
.. .bash_history .bash_logout .bash_profile .bashrc .emacs .gnome2  
.kshrc .mozilla .ncbirc .ssh .t_coffee  
$
```

- .bashrc → comandos a ser ejecutados al ejecutar bash
 - .bash_profile → comandos a ser ejecutados al hacer login
 - .bash_logout → comandos a ser ejecutados al cerrar la sesión
 - .bash_history → historial de comandos ejecutados por el usuario

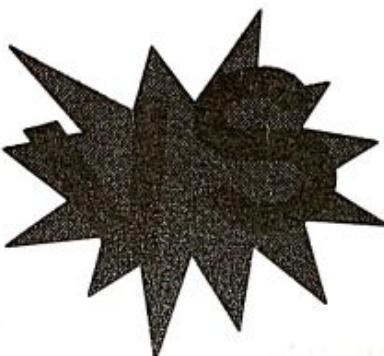
Los demás archivos escondidos en el Home suelen ser configuraciones para distintos programas. Lo mismo con los directorios escondidos

Estos archivos son básicos al momento de crear un usuario cuyo shell será bash.

Los "templates" se encuentran en /etc/skel

.bashrc

- **Se ejecuta cada vez que se ejecuta bash**
- Modifica variables de entornos
- Establece alias y funciones
- Edición más frecuente



.bash_profile

- **Se ejecuta una vez al iniciar sesión**
- Inicializa variable de entornos (de sistema)
- También ejecuta a .bashrc
- Edición menos frecuente

Variables de entorno

- Son espacios en memoria donde se guardan valores, referenciados con un nombre
 - \$ VARNAME="valor"
 - \$ echo \$VARNAME
- Definidas de esta forma, el "alcance" o "visibilidad" de la variable queda dentro del intérprete actual.
- Se referencia anteponiendo \$ al nombre de la variable

Variables de entorno

```
$ VAR="test"  
$ echo $VAR  
Test  
$ bash      #ejecutando nuevo intérprete  
$ echo $VAR
```

Variables de entorno

- Si queremos definir una variable para que quede disponible a todo el entorno, usamos **"export"**

Variables de entorno

```
$ export VAR="test"  
$ echo $VAR  
Test  
$ bash      #ejecutando nuevo intérprete  
$ echo $VAR  
Test  
$
```

Variables de entorno

- Esto permite que todas las aplicaciones que se ejecuten dentro de la terminal “que exporte” la variable, tengan acceso a ella

Otras variables útiles:

\$_	último argumento del último comando ejecutado
\$?	valor de salida del último comando
\$\$	Número de proceso del shell
\$!	Número de proceso del último comando en el background
\$0	El nombre del comando
\$n	Argumentos en la línea de comandos (\$1, \$2, ...\$n)
\$*, \$@	Todos los argumentos en la línea de comandos (\$1 \$2 \$3 ...)

■ Pregunta:

- ¿Cómo sabe "bash" dónde está "ls" cuando lo ejecutamos?
- ¿Por qué no podemos ejecutar un programa sin anteponer su directorio (relativo o absoluto)?

Variables de entorno

```
$ echo $PATH  
/usr/local/sbin:/usr/local/bin:/usr/sbin:/u  
sr/bin:/sbin:/bin:/usr/games:/usr/local/gam  
es  
$
```

- Por defecto, el sistema consultará la variable de entorno "PATH" para buscar donde se encuentra un ejecutable
- Los directorios en PATH se separan con ":"
- La variable PATH es de "SISTEMA"

Variables de entorno

■ Comando “which COMANDO”

- Consultará dónde en el “PATH” se encuentra el COMANDO
- `which cp`

Bash_profile, bashrc

■ Qué ocurre al abrir “bash”

- Dos formas de abrir bash: login e interactivo

Bash_profile, bashrc

■ Login: BASH leerá los siguientes archivos si es que existen:

1. /etc/profile ejecutará los comandos en el archivo
2. ~/.bash_profile ejecutará el contenido si encuentra el archivo
3. ~/.bash_login ejecutará el contenido si no encuentra bash_profile
4. ~/.profile ejecutará el contenido si no encuentra bash_login

"esta información se encuentra en "man bash", sección "INVOCATION"

Bash_profile, bashrc

■ BASH interactivo: por ejemplo, cuando ejecutamos "bash"

1. /etc/bash.bashrc
2. ~/.bashrc

"esta información se encuentra en "man bash", sección "INVOCATION"

Bash_profile, bashrc

■ Inspeccionemos:

1. /etc/profile
2. ~/.bash_profile
3. ~/.bashrc

Estos scripts no son ejecutados, si no que son sourced

Source vs ejecución

- Los scripts de bash pueden ser ejecutados, o pueden ser leídos e incorporados directamente en el intérprete actual (sourced)

Source vs ejecución

■ Cuando se ejecuta:

- se inicia una nueva instancia de BASH
- el cual ejecuta los comandos dentro del script
- Y cuando termina, "retorna" un valor al intérprete original (variable \$?)

■ Cuando es source:

- No se inicia nueva instancia de BASH
- Es análogo a escribir los comandos directamente en el interprete
- No hay valor de salida

Source vs ejecución

■ Ejecución:

- ▷ bash archivo.sh
- ▷ ./archivo.sh (si es que es ejecutable)
- ▷ archivo.sh (si es que está en el PATH)

■ Source:

- ▷ . archivo.sh
- ▷ source archivo.sh

CREA UN DIRECTORIO EN SU HOME)

■ Script "init.sh":

- Que cree un directorio si es que no existe
- Que agregue ese directorio al PATH (ojo: agregue)
- Que copie el archivo "\$HOME/ejercicio" dentro del directorio

■ Script "ejercicio"

- Que reconozca dos argumentos
 - Que cree archivos cuyos nombres son los argumentos, y sus contenidos es el argumento correspondiente
- Lograr que el script "ejercicio" se ejecute escribiendo "ejercicio" sin ./
- Esto último debe ocurrir cuando se inicie una terminal, automáticamente.
- COMENTE el código

Clase 4

23-08-2019

Redirección de Salidas

comando > Archivo

- = | > standart out
- | > standard out
- 2> error out
- &> std. error a la vez

> si el archivo existe se sobrescribe

||> si el archivo existe, se agrega al final

comando > archivo 2> & 1

- start a archivo
- * stderr a donde esté stdout previamente

También se puede sacar cosas de los archivos

comando < archivo

comando >> archivo

Redirigiendo

util para

cat <archivo leer el archivo

grep "patron" <archivo busca patron en archivo

read VAR <archivo lee VAR en archivo

> guardar out de un comando
en una variable

Salida de un comando a la entrada de otro



comando 1 | comando 2 | comando 3

se suelen usar filtros en la pipe

head - primera linea del file

tail - ultima linea del file

sort - ordena el archivo

sed - edita el archivo

grep - filtra por patron

uniq - omite lineas repetidas

shuf - shuffle the file

sed (editar de texto)

sed s/buscw/reemplazo/gi

s es el punto el comando reemplaza

Redirección de salidas

comando > Archivo

= | > Standard out
| > Standard err
2> Error out
| &> Standard errors
a la vez

> si el archivo existe se sobrescribe

>> si el archivo existe, se agrega al final

comando > archivo

stdout a archivo

y stderr a donde este stdout previous

También se puede sacar cosas de los archivos

comando < archivo

comando >> archivo

util para

cat <archivo leer el archivo

grep "patron" <archivo buscar patron
en archivo

read VAR <archivo lee VAR en archivo

> guardar Out de un comando
en una variable

VAR = \$(comando)

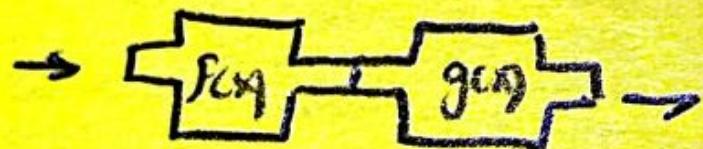
> grep 'patron' archivo
patrones (grep -e Patron1 -e Patron2

'word' → show lines that contain
'word'

'^word' → show lines that begin w/ 'word'

'word\$' → show lines that ends w/ 'word'

Salida de un comando a la entrada
de otro



comando 1 | comando 2 | comando 3

se suelen usar filtros en la pipe

head

- primera linea d file

tail

- ultima linea d file

sort - ordena el archivo

sed - edita el archivo

grep - filtra por patrones

uniq - omite lineas repetidas

shuf - shuffle the file

sed (edición de texto)

sed \$/busc/reemplazo/gi

s. es Paro el comando reemplaza

g global, reemplaza toda las coincidencias
en la linea

ignora mayus y minus

imprime líneas 75, 77 y 78

sed -n -e '75,77p' file

↑ {
parte}

arts Permite imprimir la linea
que d permite no
de imprimir otras lineas mantener

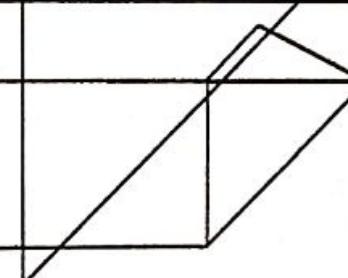
**Redirigiendo
streams**

streams

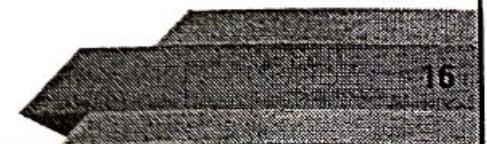
- Cuando se ejecuta un programa, éste genera un resultado, el cual es mostrado en la pantalla
- Al resultado, o lo que se muestra, se le llama "salida"
- La salida es "texto"



streams



- El texto generado por un programa puede viajar a distintas salidas
 - La pantalla
 - Un archivo
 - Otro programa



streams

- Por defecto, la salida de los programas están en un STREAM llamado "STDOUT" y "STDERR"
 - ▷ STDOUT, la salida estándar
 - ▷ STDERR, la salida estándar para errores
- Y, por defecto, estos streams están "conectados" a la terminal

streams

- También está STDIN, que es la "entrada estándar"
- Por defecto, está conectada al teclado

en resumen.

teclado → STDIN → terminal → STDOUT
STDERR

streams

~\$ ls -l /dev/std* #editado

/dev/stderr -> /proc/self/fd/2

/dev/stdin -> /proc/self/fd/0

/dev/stdout -> /proc/self/fd/1

↳ que man
wants me

■ "fd" = descriptor de archivo (file descriptor)

Cuando abres un archivo en Unix

se le asigna un nro (n: 0, 1, 2 por que

se usan)



streams

```
~$ ls -l /proc/self/fd  
0 -> /dev/tty2  
1 -> /dev/tty2  
2 -> /dev/tty2
```

- "fd" = descriptor de archivo (file descriptor)

streams

■ Redirección de la salida estándar STDOUT:

redirigir >

#si el archivo existe se sobreescribe

si no existe
lo crea

redirigir

>>

#si el archivo existe se agrega al final

streams

■ Redirección de la salida estándar STDERR:

- ▷ 2> # si el archivo existe se sobreescribe
- ▷ 2>> # si el archivo existe se agrega al final

streams

- Redirección de la salida estándar STDOUT:
 - ▷ 1> # si el archivo existe se sobreescribe
 - ▷ 1>> # si el archivo existe se agrega al final
- Sólo para STDOUT, el 1 es implícito, se puede omitir.

streams

■ Redirección de la salida estándar STDOUT y STDERR simultáneamente:

- ▷ &> # si el archivo existe se sobreescribe
- ▷ &>> # si el archivo existe se agrega al final



streams

- COMANDO > nombre_archivo
 - COMANDO >> nombre_archivo
 - COMANDO 2> nombre_archivo
 - COMANDO > nombre_archivo 2>&1 *Más simple*
 - COMANDO &> nombre_archivo
- conecta la salida STDOUT y STDERR
al archivo*

streams

■ Redirección de STDERR a STDOUT:

▷ 2>&1

■ La forma general es

▷ num_descriptor_origen>&num_descriptor_destino

streams

- Otra forma de redirigir STDERR y STDOUT a un archivo:

► COMANDO 1>archivo 2>&1

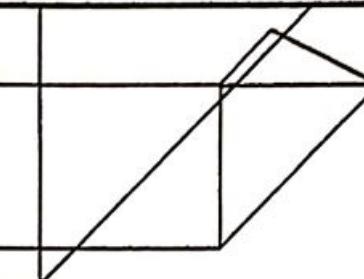
- Primero redirigimos STDOUT a un archivo

- Luego redirigimos STDERR a STDOUT

- Nótese el orden

streams

- Esto no funcionará:
 - COMANDO 2>&1 > archivo
- ¿Qué ocurrirá? (STDERR saldrá por STDOUT)
- Cuando se redirigió STDERR a STDOUT,
STDOUT aún no estaba redirigido al archivo

streams

■ Esto no funcionará:

► COMANDO 2>&1 > archivo

■ Será como decir:

► STDERR irá a donde está STDOUT ahora

► STDOUT irá a archivo



streams

■ En cambio:

► COMANDO > archivo 2>&1

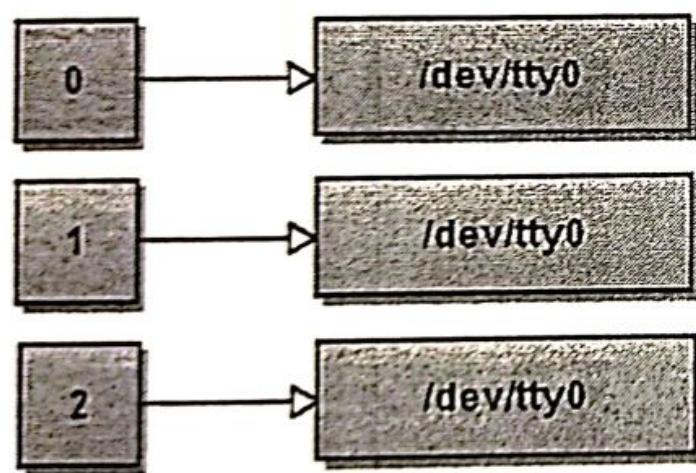
■ Será como decir:

► STDOUT irá a "archivo"

► STDERR irá a donde está STDOUT ahora

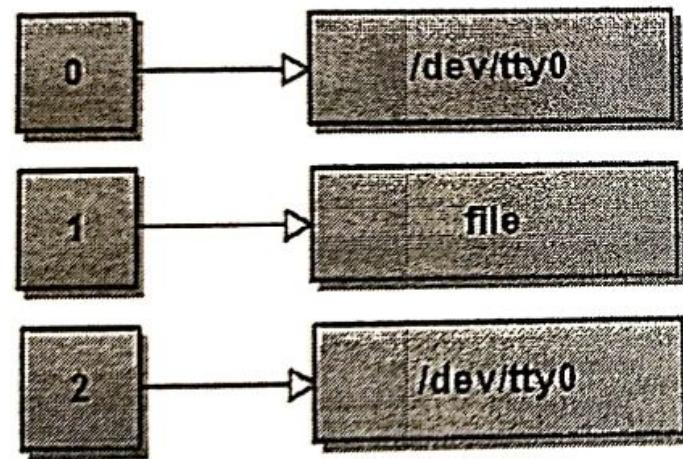


streams



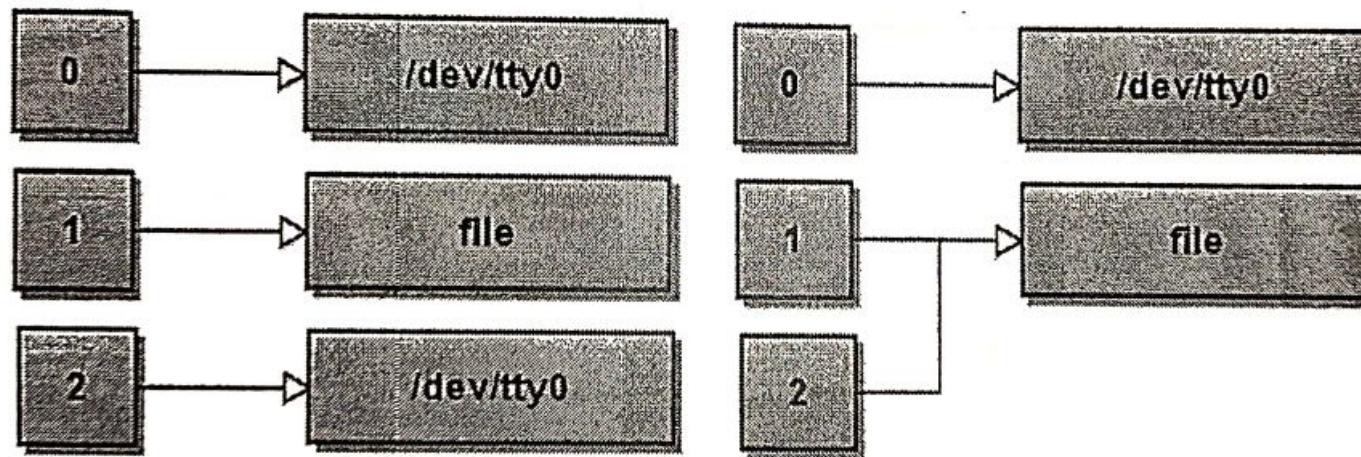
■ Todo normal

streams



■ COMANDO > file

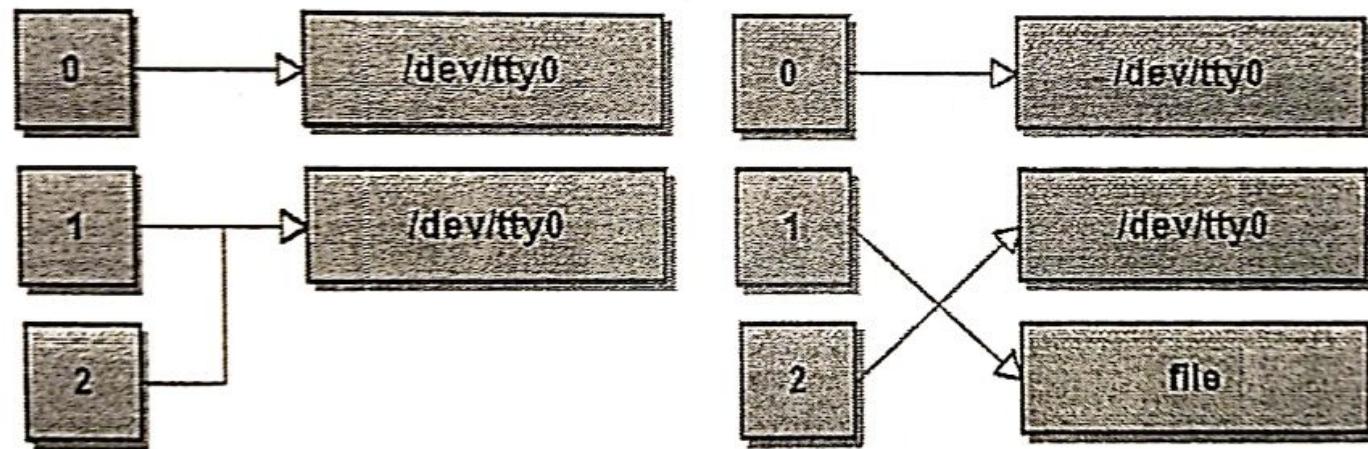
streams



■ COMANDO `&> file`

■ COMANDO `> file 2>&1`

streams



■ COMANDO `2>&1 > file`

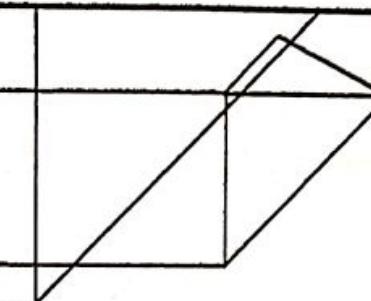
streams

- Hasta ahora, se ha redirigido las salidas
STDOUT y STDERR
- Pero también podríamos necesitar redirigir "la
entrada"

streams

- El stream STDIN, conectado por defecto al TECLADO
- Todo lo que escribimos está siendo copiado al stream STDIN (descriptor 0)

streams



■ Redirigiendo STDIN:

- ▷ COMANDO < archivo
- ▷ COMANDO 0< archivo

streams

■ Ejemplo:

- cat < archivo
- grep "patron" < archivo
- read VAR < archivo

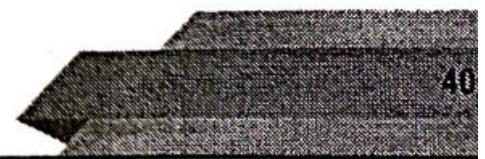
PIPES

- "Pipes" (tubos) es la manera de redirigir la salida de un comando a la entrada de otro comando
- el símbolo es |
- De esta forma, se encadenan comandos para que trabajen en conjunto



PIPES

■ COMAND01 | COMAND02 | COMAND03



PIPES

- Por ejemplo, los "filtros" trabajan de esta forma
- Los filtros son programas que modifican el contenido del Stream

PIPES

■ Filtros:

- ▷ head – Muestra las primeras líneas de un archivo
- ▷ tail – Muestra las últimas líneas de un archivo
- ▷ sort – ordena el archivo
- ▷ sed – edita el archivo
- ▷ grep – filtra por patrones
- ▷ shuf – desordena archivos
- ▷ uniq: omite líneas repetidas

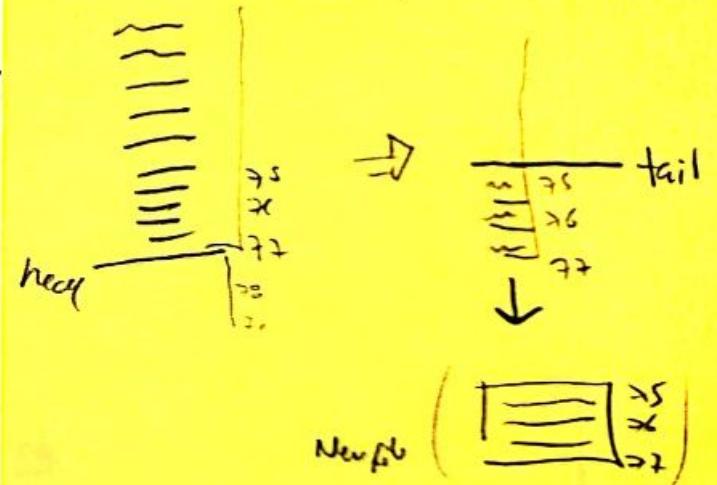
PIPES

■ Ejemplo:

- ▷ head -10 archivo # muestra las primeras 10 líneas del archivo
- ▷ tail -10 # muestra las últimas líneas del archivo
- ▷ quiero mirar las líneas 75, 76, 77 de un archivo...
- ▷ y luego escribirlas a otro archivo

Quiero mirar la linea 75, 76, 77

head -77 FILE | tail -3 > Newfile



PIPES

■ Con "sed", podemos reemplazar texto

- ▷ sed 's/buscar_esto/reemplazo/'
- ▷ sed 's/buscar_esto/reemplazo/g'
- ▷ ^ = principio de la línea
- ▷ \$ = final de la línea

Ejercicio

- sustitución de comando:
 - VAR = \$(COMANDO)
 - La variable VAR guardará la salida del comando
 - También se puede ejecutar el comando dentro de un String

grep

- grep es un filtro que nos muestra las líneas que coinciden con un patrón:
 - grep 'patron' archivo
- El patrón más simple es una palabra

grep

■ Patrones:

- 'palabra' -> muestra líneas que contienen 'palabra'
- '^palabra' -> muestra líneas que empiezan con 'palabra'
- 'palabra\$' -> muestra líneas que terminan con 'palabra'

grep

- Y podemos usar el comodín "clases de carácter"
 - ▷ [abcd], [a-zA-Z0-9], etc.

grep

- grep -e PATRON1 -e PATRON2 ...
- grep -i: ignora mayúscula, minúscula
- grep -n: nos muestra la línea de coincidencia
- grep -v: nos muestra todo, menos las líneas con coincidencias
- grep -c: nos muestra el número de líneas con coincidencias

grep**■ grep -Anum -Bnum -Cnum**

- ▷ -A num, muestra coincidencia y num líneas después
- ▷ -B num, lo mismo, pero num líneas antes
- ▷ -C num, lo mismo, num líneas antes y después

más sed

■ Otras formas de usar 'sed':

- ▷ `sed 'script' [nombre_archivo]`
- ▷ `sed [-i[backup]] -e 'script' [nombre_archivo]`
- ▷ `sed -e 'script' -f 'archivo con script'`

otras "recetas" con "sed"

■ reemplazar texto: comando 's'

- ▷ 's/buscar/reemplazo/gi'
- ▷ g = global, reemplaza todos en la línea
- ▷ i = ignorar mayúscula/minúscula
- ▷ se puede omitir cualquiera de las dos opciones

otras 'recetas' con 'sed'

■ rango de líneas:

- ▷ ya vimos que con tail, head, y pipes podemos mirar ciertas líneas de un archivo grande
- ▷ **head -77 archivo | tail -3**
 - ▷ nos muestra las líneas 75,76,77 del archivo



otras 'recetas' con 'sed'

- con "sed" podemos especificar el rango de líneas a las que queremos aplicar comandos
- `sed -n -e '75,77p' archivo`
 - el comando 'p' es para 'imprimir la línea'
 - antes de 'p', podemos especificar el rango de líneas
 - -n evita que se impriman las demás líneas



otras 'recetas' con 'sed'

- el rango de líneas lo podemos también usar con el comando 's' para reemplazar texto en las líneas especificadas
- ¿Y si se quiere reemplazar en sólo una línea?
 - (en lugar de rango, especifique el número)

otras 'recetas' con 'sed'

- También está el comando 'd', para no mostrar la línea. Podemos especificar las líneas también
- `sed -e '75,77d' archivo`

56

56

```
sed -e 's/",",;"/g' poblacion.csv | sed -e 's/,/;/g' | sed -e 's/"//g' >
```

nueva_poblacion.csv