

# LINUX

# Una breve introducción

# CONTENID OS



¿Qué es Linux?



Estructura de Linux.



Usuarios y acceso.



Directorios y ficheros.



Permisos, redireccionamientos y encauzamientos.



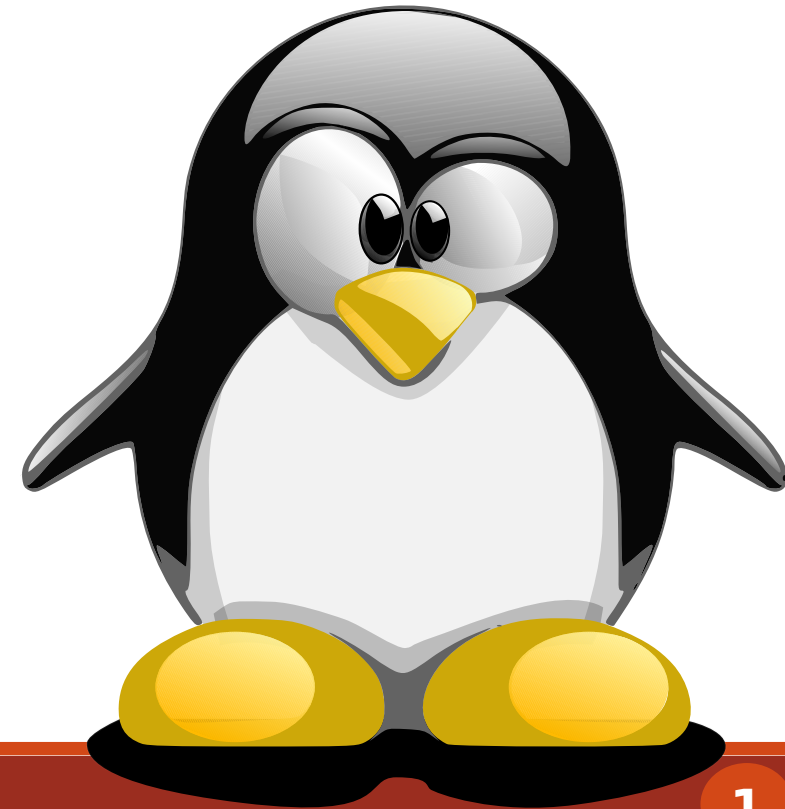
Procesos y variables de la shell.



Miscelánea

# 1. ¿Qué es Linux?

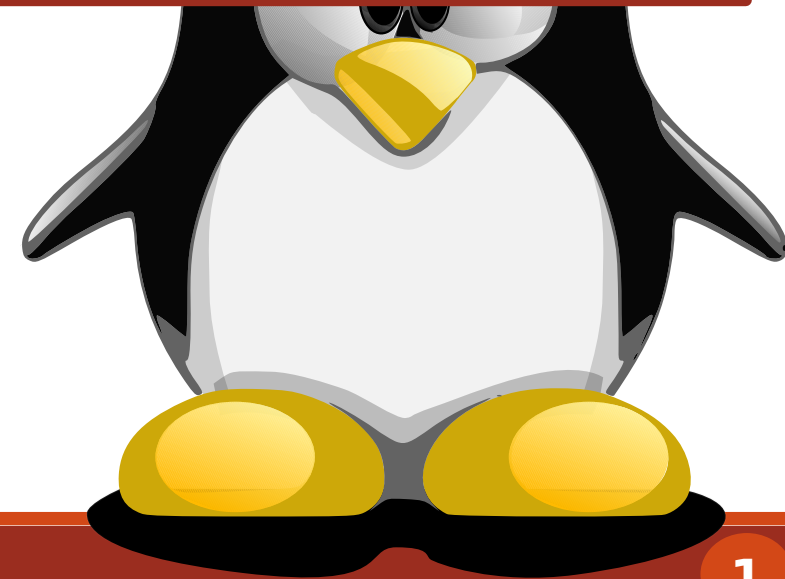
- ❑ Sistema Operativo (SO): software (programa) encargado de gestionar y usar el hardware.
- ❑ Algunas propiedades de Linux
  - Multitarea y multiplataforma
  - **Multiusuario**
  - Estable (meses e incluso años sin reiniciar)
  - Seguro
  - **Software libre:** ¡libre es mucho más que gratuito!



# 1. ¿Qué es Linux?

- ❑ Sistema Operativo (SO): software (programa) encargado de gestionar y usar el hardware.
- ❑ Algunas propiedades de Linux
  - Multitarea y multiplataforma
  - **Multiusuario**
  - Estable (meses e incluso años sin reiniciar)
  - Seguro
  - **Software libre:** ¡libre es mucho más que gratuito!

A junio de 2019, el 100% de los TOP 500 superordenadores corren en Linux!



# Software libre

- Movimiento iniciado en 1984 con el proyecto **GNU** (GNU is not Unix)
- **Postulados** (o libertades) del software libre:
  - Libertad de usar el programa, con cualquier propósito
  - Libertad de estudiar cómo funciona el programa y modificarlo, adaptándolo a tus necesidades.
  - Libertad de distribuir copias del programa, con lo cual puedes ayudar a otros.
  - Libertad de mejorar el programa y hacer públicas esas mejoras a los demás, beneficiando así a la comunidad

# 2. Estructura de Linux

## Estructura por capas:

### ❑ **Núcleo (Kernel):** interactúa directamente con el hardware

- ❖ Gestión de memoria
- ❖ Mantenimiento del sistema de archivos
- ❖ Asignación de recursos
- ❖ Control de accesos y permisos ...

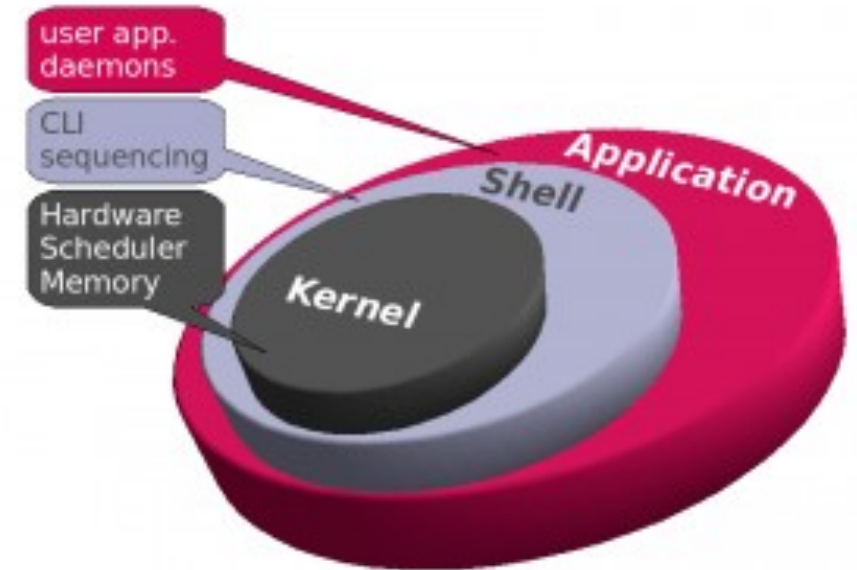
→ *Linux es el kernel*

### ❑ **Shell y servicios del sistema**

- ❖ Línea de comandos o **terminal** (shell). P. ej., bash, zsh, csh...
- ❖ Entorno gráfico

### ❑ **Aplicaciones**

- ❖ Procesadores de texto, paquetes matemáticos, entornos de programación, navegación ...
- ❖ Fácil instalación: `sudo apt-get install nombreprograma`



# 2. Estructura de Linux

**Distribución:** S.O. formado por

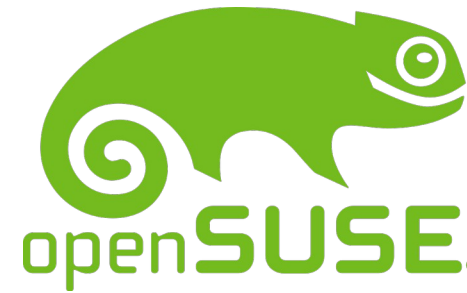
Kernel de Linux

+

Software adicional

- ❑ Entorno gráfico (KDE, Gnome, Cinnamon, Xfce...)
- ❑ Gestor de aplicaciones
- ❑ Aplicaciones

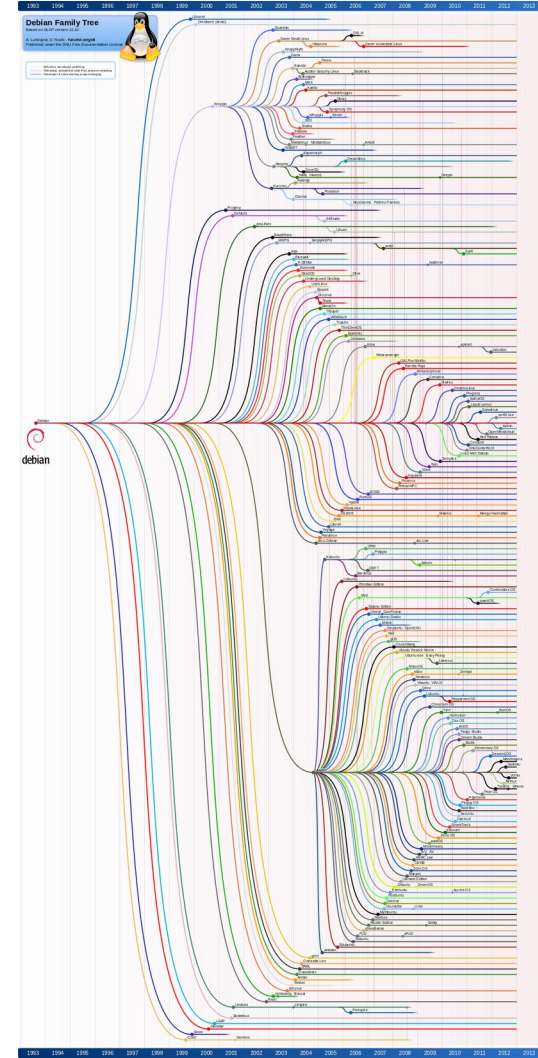
Hay ~**500 distribuciones** diferentes de Linux activas



# 2. Estructura de Linux

Evolución de las distribuciones de Linux:

[https://en.wikipedia.org/wiki/List\\_of\\_Linux\\_distributions](https://en.wikipedia.org/wiki/List_of_Linux_distributions)





# 3. Usuarios y acceso

## ❑ Superusuario (root):

- Tiene todos los privilegios: **Instalar** paquetes nuevos, modificar archivos del sistema, borrar cualquier archivo (incluso **destruyendo el sistema**)

## ❑ Usuario normal o final:

- Usuarios habituales del sistema, que utilizarán los recursos de éste. Cada usuario sólo podrá personalizar su entorno de trabajo.

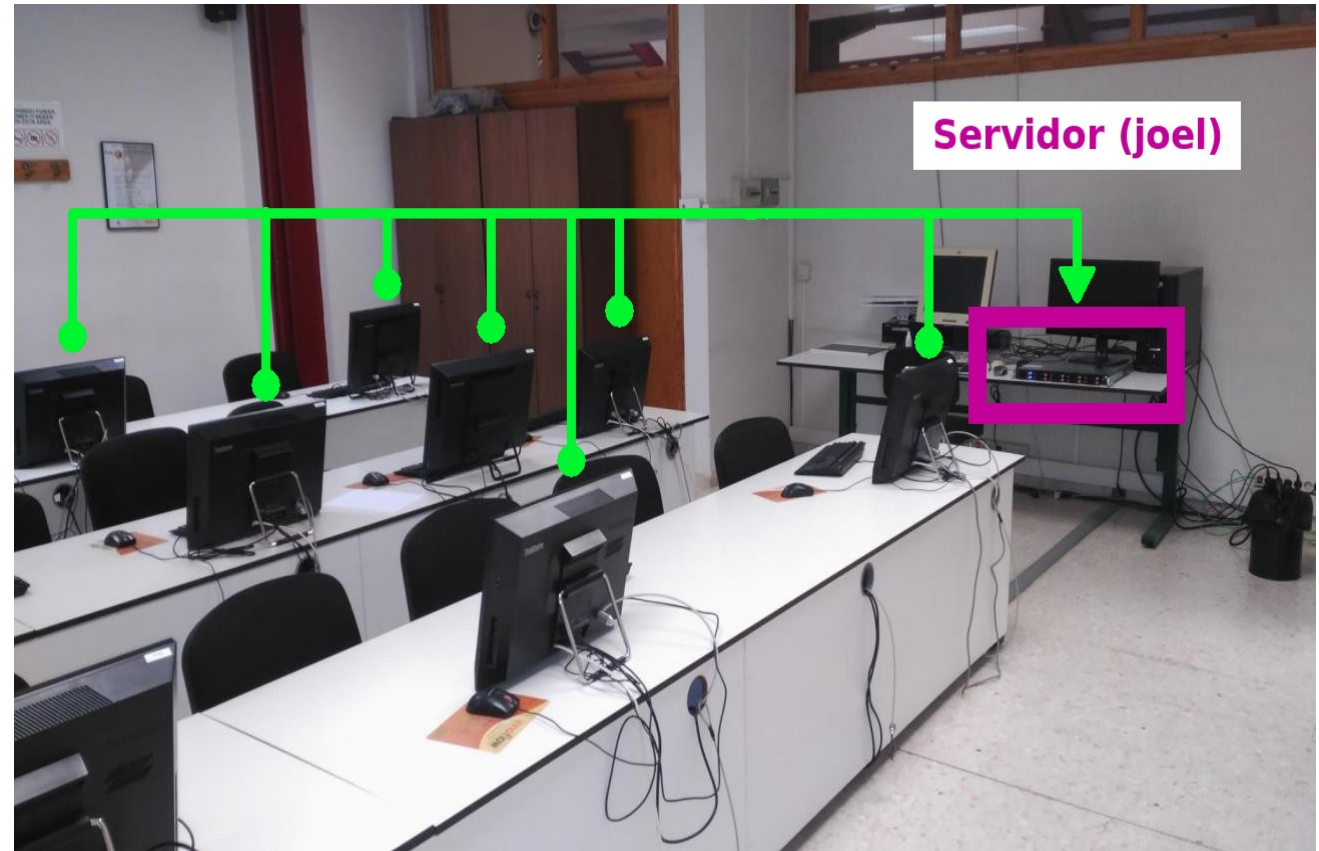
## ❑ Usuarios especiales del sistema:

- Incorporados por el propio sistema. No pueden iniciar sesión en el sistema, ni tener una shell donde trabajar. No tienen contraseña asignada. Ej.: bin, daemon, adm, lp, sync, shutdown, mail, operator, squid, apache, etc.

# 3. Usuarios y acceso

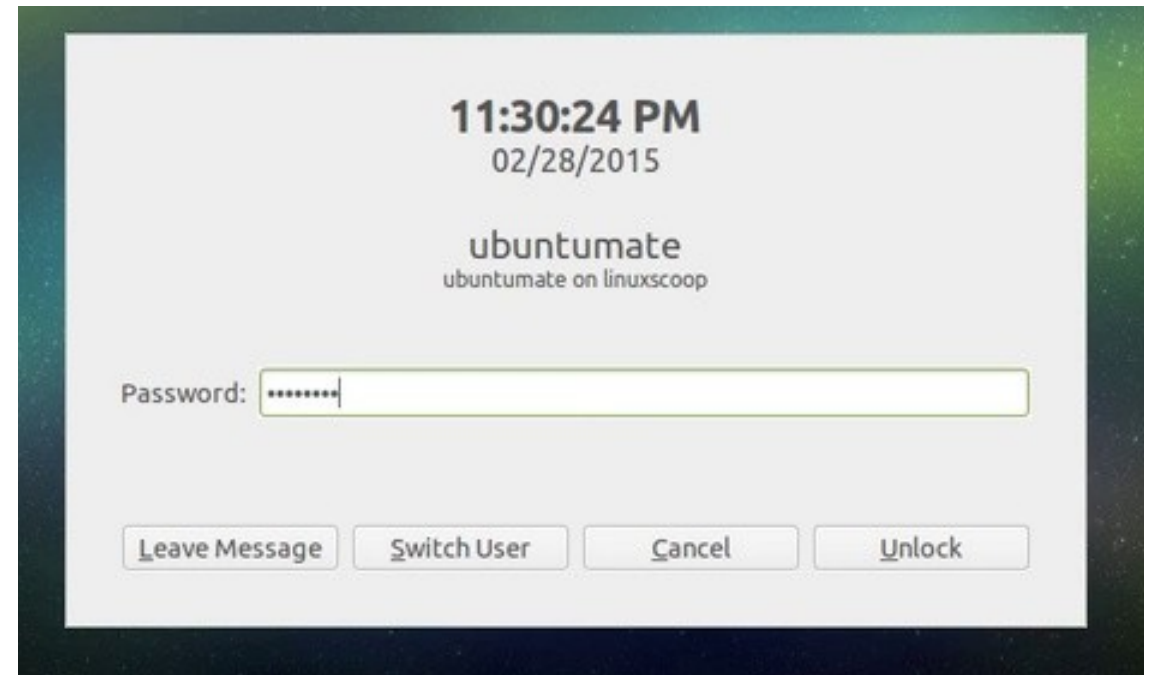
Los ordenadores del aula funcionan de forma remota.

Todo se hace en el servidor (joel), que se encuentra alojado en Proteus.



# 3. Usuarios y acceso

- ❑ **Acceded a vuestra cuenta del aula** (cphys-*apellido*)
- ❑ Vamos a familiarizarnos con la interfaz gráfica. **Buscad y abrid:**
  - Explorador de archivos
  - Navegador web
  - Editor de texto

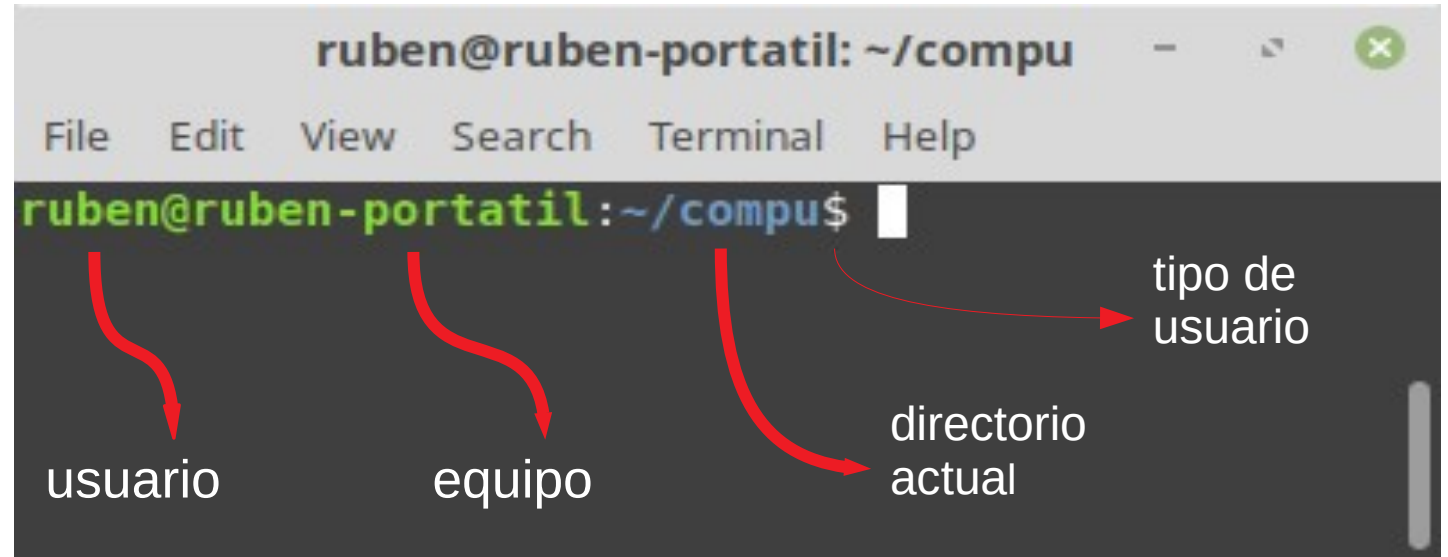


# 3. Usuarios y acceso

- ❑ **Abrid la terminal** (*shell*, línea de comandos)

Aparecerá el “**prompt**” con información de la sesión:

- Usuario
- Equipo
- Directorio
- Tipo usuario
  - \$: normal
  - #: root (!!!)



```
ruben@ruben-portatil: ~/compu
```

The screenshot shows a terminal window with a menu bar (File, Edit, View, Search, Terminal, Help) and a title bar (ruben@ruben-portatil: ~/compu). The main area displays the prompt `ruben@ruben-portatil: ~/compu$`. Red arrows point from labels to parts of the prompt: 'usuario' points to 'ruben', 'equipo' points to 'ruben-portatil', 'directorio actual' points to '~/compu', and 'tipo de usuario' points to '\$'.

# 3. Usuarios y acceso

## ❏ Comandos de interés

- ❖ **whoami**: nos dice con qué usuario estamos accediendo
- ❖ **hostname**: muestra el nombre del equipo al que estamos conectados
- ❖ **who**: muestra los usuarios conectados de forma interactiva al equipo

**Ejercicio:** comprobar usuarios con **whoami**, **who**, y máquina con **hostname**. ¿Se corresponde con lo se muestra en el “prompt”? Limpiar la terminal con **clear**.

- ❖ **ssh**: permite acceder a la terminal en una máquina remota de forma segura

Ej. `ssh nombre_usuario@nombre_equipo`

- ❖ **exit**: cierra la última sesión abierta en la terminal (alternativa “**Ctrl +D**”)

# 4. Directorios y ficheros

## ❑ Moviéndonos entre directorios

❖ **pwd** (*print working directory*): devuelve el nombre del directorio actual

❖ **cd** *name\_dir* (*change directory*): abre el directorio *name\_dir*

▪ Ruta **absoluta**. Ej. `cd /home/guillermobm/Escritorio`

Separador entre niveles (**OJO, EN WINDOWS** "ES" )

▪ Ruta **relativa** (al directorio actual). Ej. `cd Escritorio` (estando ya en vuestro usuario)

**Truco 1:** “~” = “/home/nombreusuario/” (directorio personal)

**Truco 2:** pulsando TAB se autocompleta el nombre

**Ejercicio:** entrad en vuestro escritorio usando la ruta relativa, luego comprobad la ruta absoluta con **pwd**. Volved a vuestro directorio personal usando la virgüilla.

Nombres especiales

- Directorio **actual**: .
- Directorio **“padre”**: ..
- Directorio **personal**: ~

# 4. Directorios y ficheros

- ❖ **ls**: muestra todos los ficheros y directorios contenidos en el directorio actual u otro (si se indica el path)
- ❖ **ls -l**: muestra información extendida sobre los archivos
- ❖ **ls -a**: muestra todos los ficheros, incluyendo los ocultos (comienzan por .)

## ❑ Creación y eliminación de directorios

- ❖ **mkdir** *name\_dir*: crea un directorio en la carpeta actual.
- ❖ **rmdir** *name\_dir*: elimina el directorio cuyo path es *name\_dir*. ¡Sólo si está vacío!

### Banderas (flags)

- ❑ Modifican el comportamiento usual del comando.
- ❑ Info. sobre banderas y funcionamiento de comandos:
  - **man** *comando* (info. extendida)
  - *comando* **--help** (info. más breve)

**Ejercicio 1:** cread un directorio en el escritorio y otro dentro del recién creado. Borrad ambos. ¿Afecta el orden de borrado?

**Ejercicio 2:** buscad la bandera que, aplicada a **ls**, muestra los resultados ordenados por orden de última modificación

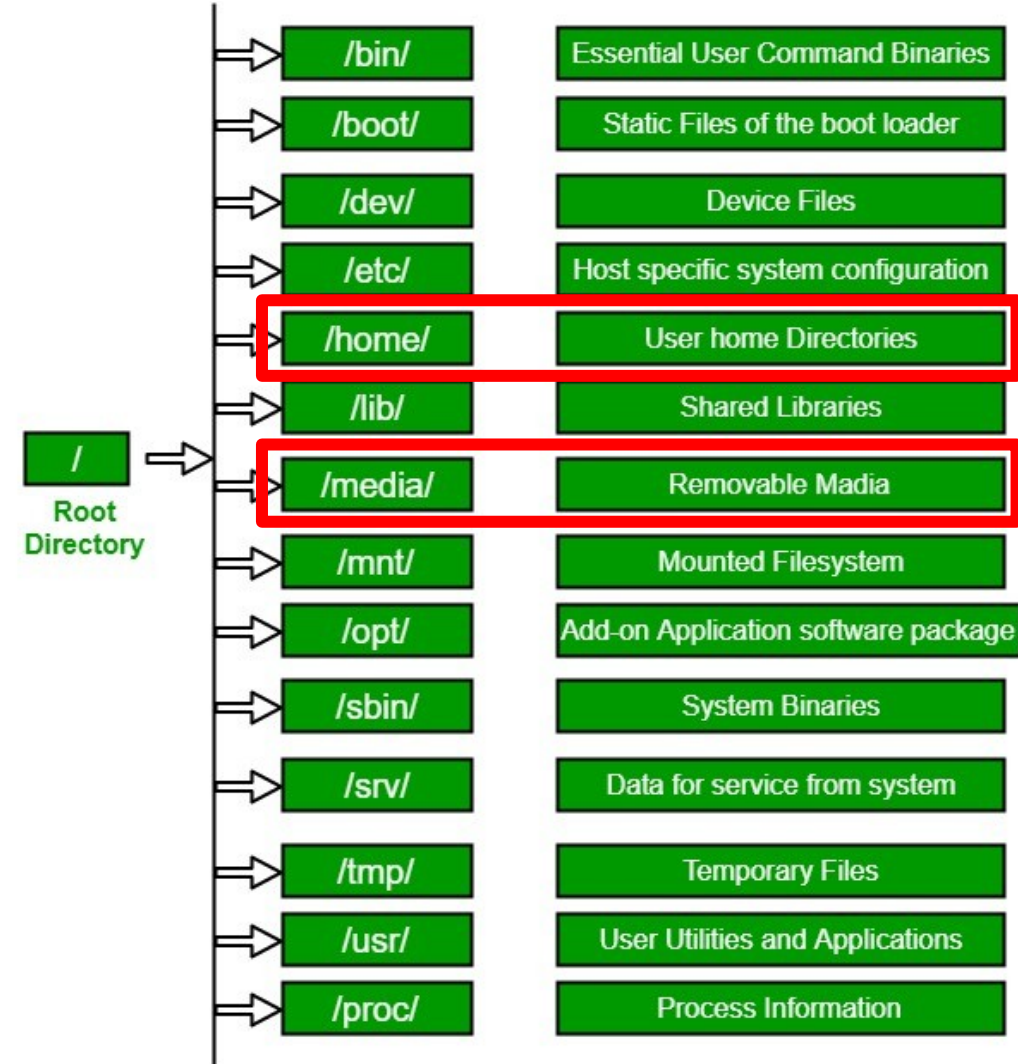
# 4. Directorios y ficheros

## Archivos

- Poseen un nombre único (máx. 255 caracteres, **4096 con la ruta!**)
- No necesitan incluir la extensión (Ej. “.txt”)
- Evitar caracteres especiales (Ej. \$, %, > ...)
- Tienen asociados un conjunto de permisos (lectura, escritura y ejecución)

## Directorios

- Poseen estructura jerárquica (en árbol).
- “ / ” denota el directorio raíz (inicio del árbol)





# 4. Directorios y ficheros

## ❑ Trabajando con ficheros

- ❖ **cp** *origen destino*: copia uno o más ficheros en el directorio *destino*. Si queremos copiar recursivamente un directorio y todo su contenido:
  - ❖ **cp -r** *dir\_origen dir\_destino*.
- ❖ **mv** *nombre\_antiguo nombre\_nuevo*: mueve ficheros o directorios, borrando los originales. Sirve también para renombrar ficheros, siempre que la carpeta de origen y destino coincidan.
- ❖ **rm** *nombre1 nombre 2 ...*: borra uno o varios ficheros. También puede borrarse un directorio (con TODO su contenido) mediante borrado recursivo:
  - ❖ **rm -r** *nombre\_directorio*

→ ¡Cuidado! Los archivos borrados con **rm** **no se pueden recuperar**.

Recordad que los nombres deben incluir el path absoluto o relativo a la carpeta en la que nos encontramos.



**Ejercicio:** copiad la carpeta *cosas\_linux* alojada en mi carpeta personal (rhurtado), con todo su contenido, a vuestro escritorio. Ya en vuestra copia, cambiad el nombre de la carpeta a *ejercicios\_linux* usando el comando **mv**.

# 4. Directorios y ficheros

## ❑ Visualización de archivos

- ❖ **cat** *nombre*: muestra el contenido de uno o varios ficheros por pantalla
- ❖ **more** *nombre*: igual, pero avanzando poco a poco. Para movernos podemos usar enter (avanza una línea), espacio (avanza una pantalla) o q para salir.
- ❖ **head** *-n nombre*: muestra las n primeras líneas de un fichero.
- ❖ **tail** *-n nombre*: muestras las n últimas líneas de un fichero.

Para ficheros muy largos:  
**Ctrl + S**: congela la salida  
**Ctrl + Q**: la restablece  
**Ctrl + C**: cancela la salida

**Ejercicio.** Usad el comando *more* para leer el fichero dentro de la carpeta que habéis copiado `data2/random_walk_018.out`. Leed los últimos valores del fichero `datos2/long_random_walk_000.out`

# 4. Directorios y ficheros

## ❑ Creación y edición de ficheros

- Editores en línea de comandos (terminal):
  - **vi/vim**: muy potente (poco intuitivo)
  - **pico**, **nano**: sencillos y prácticos
- Editores en entorno gráfico:
  - **emacs**: muy potente (también raro)
  - **gedit**, **xed**: sencillos y prácticos

**Ejercicio:** cread en la carpeta *listas* de mi directorio personal (mi usuario es *rhurtado*) un fichero de texto con título “<vuestro nombre>.txt” en el que indiquéis el curso en el que estáis, usando nano.

# 4. Directorios y ficheros

## ❏ Wildcards (comodines)

Son símbolos especiales que actúan como “comodines” en los nombres

- ❖ ? : equivale a un carácter cualquiera.
- ❖ \* : equivale a cualquier número (incluido cero) de caracteres cualesquiera.
- ❖ [abc3]: equivale a cualquiera de los caracteres entre corchetes (a, b, c, 3).
- ❖ [2-8]: equivale a cualquiera de los caracteres en el rango (2,3,4,5,6,7,8).
- ❖ [!abc3] ó [!2-8]: cualquier carácter excepto los especificados entre [...]

Más info:

<https://tldp.org/LDP/GNU-Linux-Tools-Summary/html/x11655.htm>

**Ejercicio 1:** copiad fuera de la carpeta *datos* los ficheros correspondientes a días comprendidos entre el 2 y el 9 de enero, mayo y septiembre para cualquier año. Borradlos después.

**Ejercicio 2:** igual pero sólo con los días 18 y 24. ¿Qué pasa?

# 4. Directorios y ficheros

## ❏ Wildcards (comodines)

Son símbolos especiales que actúan como “comodines” en los nombres

- ❖ ? : equivale a un carácter cualquiera.
- ❖ \* : equivale a cualquier número (incluido cero) de caracteres cualesquiera.
- ❖ [abc3]: equivale a cualquiera de los caracteres entre corchetes (a, b, c, 3).
- ❖ [2-8]: equivale a cualquiera de los caracteres en el rango (2,3,4,5,6,7,8).
- ❖ [!abc3] ó [!2-8]: cualquier carácter excepto los especificados entre [...]

Más info:

<https://tldp.org/LDP/GNU-Linux-Tools-Summary/html/x11655.htm>

Se limitan a  
**caracteres**

**Ejercicio 1:** copiad fuera de la carpeta *datos* los ficheros correspondientes a días comprendidos entre el 2 y el 9 de enero, mayo y septiembre para cualquier año. Borradlos después.

**Ejercicio 2:** igual pero sólo con los días 18 y 24. ¿Qué pasa?

# 4. Directorios y ficheros

## ❑ Wildcards “**extendidas**” (extended globbing)

Se activan con el comando: `shopt -s extglob` (debe ejecutarse cada vez que se abra la terminal)

- ❖ `?(text1|text2|text3)`: cero o una ocurrencia de alguno de text1/2/3.
- ❖ `*(text1|text2|text3)`: zero o más ocurrencias de text1/2/3.
- ❖ `+(text1|text2|text3)`: una o más ocurrencias de text1/2/3.
- ❖ `@(text1|text2|text3)`: una ocurrencia de text1/2/3.
- ❖ `!(text1|text2|text3)`: cualquier cosa excepto text1/2/3.

Más info:

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

## ❑ Rangos con varios dígitos

- ❖ `{8..13}`: Cualquier número en el rango (8, 9, 10, 11, 12, 13).

**Ejercicio 1:** repetid el ejercicio 2 de la diapositiva anterior.

**Ejercicio 2:** para activar el “extended globbing” de forma permanente, añadid “`shopt -s extglob`” a vuestro fichero `~/.bash_profile` (fichero oculto). Los comandos en este fichero se ejecutan automaticamente cada vez que abrís una terminal.

# 4. Directorios y ficheros

## ❏ Permisos

Los ficheros y directorios tienen 3 tipos de permiso:

– lectura (**r**) – escritura (**w**) – ejecución (**x**)

Los permisos pueden concederse al:

– propietario (**u**) – grupo (**g**) – todos (**o**)

❖ **ls -l**: muestra permisos de los ficheros.

❖ **chmod**: modifica los permisos de ficheros o directorios (“+” concede, “-” quita).

*Ej.:* `chmod ug+r-w file_name` → Concede al propietario y su grupo permiso de escritura y les quita el de escritura

```
rhurtado@joel:~$ ls -l
total 0
drwx----- 3 rhurtado users 23 ene 21 18:52 Arduino
drwx----- 3 rhurtado users 56 ene 21 18:49 Descargas
drwx----- 2 rhurtado users  6 feb 12 2019 Documentos
drwx----- 2 rhurtado users 93 ene 21 18:50 Escritorio
drwxrw---- 4 rhurtado users 60 mar  4 2019 fortran
drwxrw---- 3 rhurtado users 91 mar  4 2019 gnuplot
drwx----- 2 rhurtado users  6 feb 12 2019 Imágenes
drwxrwxrwx 2 rhurtado users  6 mar  1 21:35 listas
drwx-g--o- 2 rhurtado users  6 feb 12 2019 Música
drwx----- 2 rhurtado users  6 feb 12 2019 Plantillas
drwx----- 2 rhurtado users  6 feb 12 2019 Público
drwx----- 2 rhurtado users 63 mar  2 10:51 utilidades
drwx----- 2 rhurtado users  6 feb 12 2019 Vídeos
```

**Ejercicio:** cread un fichero y un directorio en vuestro escritorio. Dad y quitad permisos comprobando qué sucede en cada caso.

# 5. Redireccionamientos y encauzamientos

## ❑ Redireccionamientos

Podemos redirigir la entrada y salida de los comandos a ficheros:

- Redireccionamiento de entrada: `comando < fichero_entrada`
- Redireccionamiento de salida: `comando > fichero_salida`
- Combinación: `comando < fichero_entrada > fichero_salida`



¡Útil para conservar la salida de un programa!

**Ejercicio:** ejecutad el script `random_walk_generator.sh` (antes tendréis que darle permisos de ejecución) y guardad la salida en el fichero “aleatorio.dat”.



# 5. Redireccionamientos y encauzamientos

## ❏ Encauzamientos (*pipes*)

Permiten redireccionar la salida estándar de un programa a la entrada de otro programa

Sintaxis: comando\_1 | comando\_2

Un comando útil:

❖ **grep** *cadena file\_name*: selecciona las líneas que contienen cierta cadena de caracteres.

→ ¡Podemos encadenar tantos comandos como queramos!

Grep también admite comodines, que se conocen como *regular expressions* o **regex**.

→ Más info: <https://tldp.org/LDP/GNU-Linux-Tools-Summary/html/x11655.htm> (apartado “Regular Expressions”)

Ejemplos:

ls -l | grep cphys-\* (en /home/)

cat numbers.txt | grep 2

### Información útil sobre grep0

**grep -l** : muestra sólo los nombres de los archivos

**grep -i** : búsqueda insensible a mayúsculas.

**grep -r** : búsqueda en subdirectorios

**grep -v** : devuelve las líneas que NO contienen la cadena de caracteres

**pdgrep**: funciona igual que grep pero permite buscar en ficheros pdf.

# 5. Redireccionamientos y encauzamientos

**Ejercicio:** escribid los valores de los archivos `data/aviones_YYYY-MM-DD.out` asociados a los días 7, 10, 15 y 21 de los meses de enero, febrero y septiembre y a la hora 17:15. Usando wildcards (comodines), redireccionamientos y pipes (encauzamientos), puede hacerse en una sola línea.

# 5. Redireccionamientos y encauzamientos

## ❑ Bonus track: scripts

- Un script es un fichero ejecutable que contiene un conjunto de comandos.
- La extensión habitual para scripts de bash (terminal) es **.sh**
- Para poder usarlos hay que otorgarles permisos de ejecución:  
`chmod +x script_name.sh`
- Para ejecutarlos hay que dar la ruta absoluta: `./script_name.sh`

### Script generador.sh

```
for i in {1..50}
do
    echo $RANDOM
done
```

**Ejercicio 1:** cread el script generador.sh y otorgadle permisos de ejecución. Emplead el comando **grep** para encontrar todos los números que no contienen un “1”, ordenadlos por valor numérico y verted todo en un fichero de texto “ordered\_numbers.txt” (os hará falta el comando “sort”). **Pista:** todo puede realizarse en una única línea usando encauzamientos.

# 6. Procesos y variables de la Shell

- Cada programa que ejecuta el ordenador es un proceso.
- Un S.O. multitarea como Linux puede ejecutar varios procesos asignando pequeñas fracciones de tiempo a cada uno.
- Un *daemon* (demonio) es un proceso residente que está a la espera de realizar alguna función (Ej.: *lpd* es el daemon de impresión)

## □ Administración de procesos

- ❖ **ps aux**: muestra todos los procesos del sistema
- ❖ **top**: muestra de forma actualizada los procesos que más recursos consumen
- ❖ **kill -X PID** : envía una señal X (por lo general de terminación) al proceso con el PID especificado. Por defecto se aplica la señal 15 (*soft kill*). Un kill más agresivo sería **kill -9** (*hard kill*).

A cada proceso se le asigna un número PID (*process identification number*) que lo identifica

**Ejercicio:** Probad todos los comandos anteriores. En particular, puede ser útil el encauzamiento **ps aux | more** para mostrar los procesos uno por uno o **ps aux | grep nombre\_programa** para encontrar el PID de cierto programa.

# 6. Procesos y variables de la Shell

## ❑ Relación padre-hijo

- Un proceso puede “crear” otro proceso, convirtiéndose en padre del segundo.
- Cuando lanzamos un proceso desde la terminal, esta se bloquea hasta que el proceso hijo acaba.
- Si un proceso padre muere, también desaparecerán los procesos hijos (**Init** es el proceso padre de todos).

**Ejercicio:** lanzad xed desde la terminal e intentad utilizar esta última. ¿Qué ocurre si cerramos la terminal?

- Para evitar lo anterior:
  - **comando &**: lanza el proceso en segundo plano (*background*), de modo que la terminal no se queda colgada. Si el proceso ya ha sido lanzado, lo pausamos con **Ctrl + Z** y lo mandamos al background escribiendo **bg** en la terminal. Si queremos simplemente reactivarlo, escribimos **fg** (*foreground*).
  - **nohup comando**: emancipa el proceso hijo del proceso padre (es el abuelo el que hará las veces de padre).

**Ejercicio:** lanzad xed en segundo plano y emancipado de la terminal. Buscad su PID con **top** y matadlo con **kill -9 PID**.

# 6. Procesos y variables de la Shell

## ❑ Variables de la Shell

- La shell posee variables que pueden utilizarse para configuración personal de nuestro entorno, programación de scripts, transferencia de parámetros entre un proceso padre y otro hijo ...
  - ❖ **echo** *\$variable*: nos permite ver el valor actual de una variable.
  - ❖ **set** *variable=valor*: permite asignar un valor a una variable.
- Algunas variables de la Shell:
  - *\$HOME*: indica el directorio “home” del usuario.
  - *\$PATH*: directorios donde el sistema busca un comando.
  - *\$UID*: numero de identificación del usuario.

**Ejercicio:** ejecutad en la terminal el comando **echo** *\$USER*. ¿Qué comando de Linux que ya hemos visto es equivalente a esta orden?

# 7. Miscelánea

❖ **find** *directorio* *–opciones* *criterios*: permite encontrar archivos en el sistema.

▪ Ej.: `find . -name “*numbers*”`

❖ **du** *directorio*: desglosa el tamaño de cada subdirectorio en bytes. Al final indica el tamaño total.

▪ Ej.: `du ~/Escritorio`

❖ **wc** *nombre\_fichero*: cuenta caracteres (-c), líneas (-l) o palabras (-w) en un fichero.

▪ Ej.: `wc -l ordered_numbers.txt`

❖ **crontab** *-e*: permite editar tareas periódicas (crond es el *daemon* de control de tareas).

❖ **tar** *–opciones* *–lista\_ficheros*: permite empaquetar (-c) y extraer (-x) ficheros.

▪ Ej.: `tar -cf numeros.tar “ordered_numbers.txt”`  
“numbers.txt”



This Photo by Unknown Author is licensed under CC BY-SA-NC



# Linux Command Cheat Sheet

 [Share This Cheat Sheet](#)

## Basic commands

	Pipe (redirect) output
sudo [command]	run < command> in superuser mode
nohup [command]	run < command> immune to hangup signal
man [command]	display help pages of < command>
[command] &	run < command> and send task to background
>> [fileA]	append to fileA, preserving existing contents
> [fileA]	output to fileA, overwriting contents
echo -n	display a line of text
xargs	build command line from previous output
1>2&	Redirect stdout to stderr
fg %N	go to task N
jobs	list task
ctrl-z	suspend current task

## File permission

chmod -c -R	chmod file read, write and executable permission
touch -a -t	modify (or create) file timestamp
chown -c -R	change file ownership
chgrp -c -R	change file group permission
touch -a -t	modify (or create) file timestamp

## Network

netstat -r -v	print network information, routing and connections
telnet	user interface to the TELNET protocol
tcpdump	dump network traffic
ssh -i	openSSH client
ping -c	print routing packet trace to host network

## File management

find	search for a file
ls -a -C -h	list content of directory
rm -r -f	remove files and directory
locate -i	find file, using updatedb(8) database
cp -a -R -i	copy files or directory
du -s	disk usage
file -b -i	identify the file type
mv -f -i	move files or directory
grep, egrep, fgrep -i -v	print lines matching pattern

## File compression

tar xvfz	create or extract .tar or .tgz files
gzip, gunzip, zcat	create, extract or view .gz files
uuencode, uudecode	create or extract .Z files
zip, unzip -v	create or extract .ZIP files
rpm	create or extract .rpm files
bzip2, bunzip2	create or extract .bz2 files
rar	create or extract .rar files

## File Editor

ex	basic editor
vi	visual editor
nano	pico clone
view	view file only
emacs	extensible, customizable editor
sublime	yet another text editor
sed	stream editor
pico	simple editor

## Directory Utilities

mkdir	create a directory
rmdir	remove a directory

## File Utilities

tr -d	translate or delete character
uniq -c -u	report or omit repeated lines
split -l	split file into pieces
wc -w	print newline, word, and byte counts for each file
head -n	output the first part of files
cut -s	remove section from file
diff -q	file compare, line by line
join -i	join lines of two files on a common field
more, less	view file content, one page at a time
sort -n	sort lines in text file
comm -3	compare two sorted files, line by line
cat -s	concatenate files to the standard output
tail -f	output last part of the file

## Scripting

awk, gawk	pattern scanning
tsh	tiny shell
" "	anything within double quotes is unchanged except \ and \$
' '	anything within single quote is unchanged
python	"object-oriented programming language"
bash	GNU bourne-again SHell
ksh	korn shell
php	general-purpose scripting language
csch, tcsh	C shell
perl	Practical Extraction and Report Language
source [file]	load any functions file into the current shell, requires the file to be executable

## Memory & Processes

free -m	display free and used system memory
killall	stop all process by name
sensors	CPU temperature
top	display current processes, real time monitoring
kill -1 -9	send signal to process
service [start   stop   restart]	manage or run sysV init script
ps aux	display current processes, snapshot
dmesg -k	display system messages

## Disk Utilities

df -h, -i	File system usage
mkfs -t -V	create file system
resize2fs	update a filesystem, after lvextend*
fsck -A -N	file system check & repair
pvccreate	create physical volume
mount -a -t	mount a filesystem
fdisk -l	edit disk partition
lvcreate	create a logical volume
umount -f -v	umount a filesystem

## Misc Commands

pwd -P	print current working directory
bc	high precision calculator
expr	evaluate expression
cal	print calender
export	assign or remove environment variable
` [command]	backquote, execute command
date -d	print formatted date
\$(variable)	if set, access the variable

Sponsored by [loggly](#)

[Read the Blog Post »](#)  
[bit.ly/Linux-Commands](http://bit.ly/Linux-Commands)