# 1. Introducción Linux

Linux es una versión gratuita o libre de UNIX. Como tal es un sistema operativo multitarea y multiusuario.

Tiene un entorno gráfico (que estáis viendo) como otros sistemas operativos, pero la manera más potente de utilizarlo es empleando la "TERMINAL"de comandos donde se introducen breves líneas con órdenes sencillas que realizan las tareas usuales y muchas más.

# 2. Comandos de Linux

## 2.1. Comandos básicos

Los comandos son esencialmente los mismos que en cualquier sistema UNIX. En la tabla 2.1 tenemos una lista de los comandos más frecuentes usados para el manejo de archivos; en la tabla 2.2 recogemos algunos de los comandos para el control de procesos. Siempre es posible obtener ayuda sobre la sintaxis de uno de estos comandos haciendo `man comando`.

| **Comando** | **Sintaxis** | **Descripción** |
|---|---|---|
| cat | cat *[fich1 . . . fichN]* | Muestra un fichero o ficheros. |
| cd | cd [*dir*] | Cambia de directorio. Si no se especifica ninguno, irá al directorio particular del usuario. |
| chmod | chmod *permisos arch* | Cambia los permisos de un archivo. |
| cp | cp *fich1. . . fichN dir* | Copia un fichero o ficheros a un directorio. |
| diff | diff *fich1 fich2* | Busca diferencias entre ficheros |
| gedit | gedit *fich* | Edita un archivo. |
| find | find *dir* -name *fich* | Busca el archivo *fich* por debajo del directorio *dir* |
| gcc | gcc -lm *fich.c* | compilador de C y C++. |
| gdb | gdb *fich.exe* | *debugger* de C y C++. |
| gfortran | gfortran *fich.f* | compilador de Fortan. |
| grep | grep *cadena fich(s)* | Muestra líneas de que contengan una cadena de caracteres. |
| head | head *fich* | Muestra las primeras líneas de una archivo. |
| less | less *fich(s)* | Visualiza página a página el contenido de un fichero. Funcionan los comandos del editor *vi*. |
| ln | ln -s *fich1 fich2* | Crea un *link (enlace)* entre dos archivos. |
| ls | ls | Lista el contenido del directorio actual o del especificado. |
| lpr | lpr -P*cola fich* | Imprime una archivo en una cola de impresión. |
| mkdir | mkdir *dir* | Crea un directorio. |
| more | more *fich(s)* | Visualiza página a página el contenido de un fichero. |
| mv | mv *fich1. . . fichN dir* | Mueve un fichero o ficheros a un directorio. |
| | mv *fich1 fich2* | Renombra un fichero. |
| pwd | pwd | Muestra la ruta del directorio actual. |
| rm | rm *fich* | Borra un fichero. |
| | rm -r *dir* | Borra un directorio con todo su contenido. |
| tail | tail *fich* | Muestra las últimas líneas de un archivo. |
| tar | tar *opciones fich.tar fich(s)* | Crea o manipula un archivo que es una colección de otros archivos. |
| wc | wc *fich* | Cuenta el número de bytes, palabras o líneas de un archivo. |
| xmgrace | xmgrace | Abre un programa de dibujo gráfico height |

Cuadro 2.1: Comandos Linux más frecuentes para manejo de archivos

| Comando | Sintáxis | Descripción |
|---|---|---|
| bg | bg %*n* | Envía al background un proceso suspendido. |
| fg | fg %*n* | Recupera del background un proceso. |
| finger | finger *usuario* | Muestra información sobre un usuario. |
| jobs | jobs | Muestra los procesos activos del usuario. |
| & | *comando* & | Ejecuta un comando en background. |
| kill | kill %*n* | Detiene un proceso. |
| Control-z | Control-z | Suspende un proceso activo. |
| Control-c | Control-c | Detiene un proceso activo. |
| ps | ps *opciones* | Muestra los procesos activos. |
| telnet | telnet *hostname* | Establece una conexión con otra máquina. |
| top | top | Muestra información sobre el estado de la máquina. |
| at | at *opciones* TIME | Manipula un proceso para ejecución posterior. |
| exit | exit | Sale de la sesión. |
| who | who | Muestra los usuarios de la máquina. |

Cuadro 2.2: Comandos Linux más frecuentes para control de procesos

## 2.2. Comandos en background

Linux, como cualquier sistema Unix, puede ejecutar varias tareas al mismo tiempo. En sistemas monoprocesador, se asigna un determinado tiempo a cada tarea de manera que, al usuario, le parece que se ejecutan al mismo tiempo.

Para ejecutar un programa en background, basta con poner el signo ampersand (&) al término de la línea de comandos (ver epígrafe 2.3.1). Por ejemplo, si quisiéramos copiar el directorio */usr/src/linux* al directorio */tmp*:

```
#cp -r /usr/src/linux /tmp &
```

Si ahora ejecutamos el comando

```
#jobs
```

veremos que el proceso lanzado al background está corriendo. Cuando ha terminado la ejecución del programa (tarda bastante, podeis ir haciendo otra cosa), el sistema nos lo dice mediante un mensaje:

```
#
   [Done] cp -r /usr/src/linux /tmp
#
```

Si hubiésemos ejecutado el programa y no hubiésemos puesto el ampersand, podríamos pasarlo a background de la siguiente manera:

1. Suspendemos la ejecución del programa, pulsando `Ctrl-z`.
2. Ejecutamos la siguiente orden: `bg %1`

## 2.3. Shell *bash*

Linux dispone de diversos intérpretes de comandos, como pueden ser: *bash*, *csh*, *tcsh*, *sh*,... En Unix, al intérprete de comandos se le denomina *shell*.

El objetivo de cualquier intérprete de comandos es ejecutar los programas que el usuario teclea en el prompt del mismo. El prompt es una indicación que muestra el intérprete para decirnos que espera una orden nuestra. Cuando el usuario escribe una orden, el intérprete ejecuta dicha orden.

### 2.3.1. Sintaxis de los comandos

Los comandos tienen la siguiente sintaxis:

```
# programa arg1 arg2 ... argn
```

Vemos que, en la "línea de comandos", introducimos el *programa* seguido de uno o varios *argumentos*. Así, el intérprete ejecutaría el *programa* con las opciones que le hayamos escrito.

Una característica interesante de *bash* es que, mediante el tabulador, intenta completar los nombres de los ficheros o comandos que empezamos a escribir. Basta con introducir las primeras dos o tres letras del fichero o comando y darle al tabulador para que el *shell* complete el nombre o nos ofrezca varias alternativas.

Cuando queremos que el comando sea de varias líneas, separaremos cada línea con el carácter *barra invertida (\)*. Además, cuando queremos ejecutar varios comandos en la mísma línea, los separaremos con *punto y coma (;)*. Por ejemplo,

```
# ls -l ; ps aux
```

En los comandos, también podemos utilizar los comodines:

- El *asterisco (∗)* es equivalente a uno o más caracteres en el nombre de un archivo.
- El *signo de interrogación (?)* es equivalente a un único carácter.
- Un *conjunto de caracteres entre corchetes* es equivalente a cualquier carácter del conjunto.

### 2.3.2. Intérprete y compilador

Un lenguaje de programación interpretado va leyendo comandos que el usuario dicta y los va ejecutando uno a uno: ejemplos típicos son Python, Mathematica, Maple, Matlab, etc. El programa no sabe lo que viene a continuación. Suelen ser muy lentos salvo que se exporte la salida precompilada y se ejecute por separado. Son el equivalente a un traductor simultáneo entre dos idiomas humanos.

Los lenguajes compilados toman un fichero de texto de entrada y producen una salida çompilada"(equivalente al traductor humano que traduce un libro). Son mucho más rápidos para el cálculo numérico. Fortran y C/C++ son los más usados. En estas prácticas emplearemos Fortran.

Pasos a seguir:

- Escribir fichero con el programa, por ejemplo Miprograma.f
- Compilar `gfortran Miprograma.f` (produce el programa ejecutable)
- Ejecutar el programa `./a.out`
- Estudiar la salida de datos, por ejemplo con el programa de dibujo xmgrace.

### 2.3.3. Redireccionamiento de E/S

La mayoría de los programas que se han sido diseñados para trabajar conjuntamente (mediante su combinación se puede crear un comando más complejo), leen de la entrada estándar (teclado) y escriben en la salida estándar (pantalla).

Gracias a esta característica, podemos sustituir la entrada y salida estándar por, por ejemplo, un archivo que contenga las opciones a ejecutar y un archivo de salida, respectivamente. Ejemplos:

**Entrada:** Si queremos mandar un mail al usuario `user32` en la máquina `ftlab3` conteniendo el archivo `fich` hacemos

```
mail user32@ftlab2 < fich
```

**Salida:** Si queremos saber los archivos que empiezan por i o I y almacenarlo en un archivo: `ls [iI]* > listado.txt`

### 2.3.4. Historia de comandos

La shell *bash* va almacenando lo que llamamos "historia de comandos", es decir, todas las órdenes que hemos ido escribiendo en la "línea de comandos". De este modo, podemos volver a ejecutar una orden que ya habíamos escrito anteriormente sin tenerla que escribir de nuevo. Para recuperar órdenes anteriores usamos las teclas ↑ o ↓.

### 2.3.5. Archivos de bash

Estos archivos se muestran en la tabla 2.3 y son utilizados para especificar opciones dentro del *bash*.

| Archivo | Descripción |
|---|---|
| /bin/bash | Ejecutable *bash*. |
| /etc/profile | Archivo de inicialización utilizado por los shells. |
| ~/.bash_profile | Archivo de inicialización personal utilizado por los shells. |
| ~/.bashrc | Archivo de inicialización del shell. |
| ~/.inputrc | Archivo de inicialización individual. |

Cuadro 2.3: Archivos de *bash*

### 2.3.6. Programación shell

La programación *shell* en Unix/Linux es, en cierto sentido, equivalente a crear un archivo *.BAT* en DOS. La diferencia es que en Unix/Linux es mucho más potente.

Podemos crear *scripts*[1] que hagan una determinada tarea, utilizando toda la potencia del *shell* que utilicemos (*sh, bash, csh, tcsh,...*).

En *bash*, disponemos de diversas instrucciones útiles para la programación *shell* (ver tabla 2.4).

## 2.4. Permisos

Linux es un sistema multiusuario, por ello los archivos gozan de tres tipos de permisos. Permisos del propietario, del grupo y del resto. Así, un archivo puede ser accedido únicamente por el propietario, por el grupo al que pertenece o por cualquier usuario. Los permisos son de lectura (`r`), de escritura (`w`) y de ejecución (`x`). Para ver los permisos ejecutaríamos `#ls -l`, obteniendo por ejemplo

```
drwxr-xr-x   5 fc21   fc21        4096 Mar 24 09:29 Dir_1
-rw-rw-r--   1 fc21   fc21        324 Mar 24 10:53 file_1.dat
```

[1] En Unix/Linux, se llama así al equivalente a los archivos *.BAT* de DOS.

- `for name [ in word; ] do list ; done`
- `select name [ in word ; ] do list ; done`
- `case word in [ pattern [ | pattern ]\ldots ) list ;; ]\ldots esac`
- `if list then list [ elif list then list ]\ldots [ else list ] fi`
- `$while list do list done`
- `$until list do list done`
- `[ function ] name () { list; }`

Cuadro 2.4: Instrucciones *bash* para programación *shell*

En este caso `Dir_1` es un directorio con permiso de lectura, escritura y ejecución por el propietario `fc21` y permiso de lectura y ejecución para los miembros de su grupo `fc21` y para cualquier otro usuario. Por otra parte, `file_1.dat` es un archivo que pueden leer y escribir el propietario y los miembros de su grupo, mientras que el resto de usuarios sólo pueden leerlo.

Para modificarlos permisos ejecutamos `chmod args`. Para modificar el propietario y el grupo de un archivo se hace `chown user.group filename`.

## 2.5. X-Windows

Es el sistema de ventanas para Linux análogo al sistema operativo Windows95, etc.. Existen diversos gestores de ventanas: *fvwm, kde, gnome,...* Éste último es el que está instalado en los ordenadores del laboratorio. Para iniciar una sesión debeis teclear

```
#startx
```

Entre las aplicaciones que se pueden ejecutar en el entorno X destacamos:

- `xterm` o `gnome-terminal` que emulan terminales.
- `gnuplot` que permite dibujar gráficas en varios entornos.
- `gv` muestra archivos postscript *ps, eps*
- `emacs` en este modo el editor es más versátil.
- `xv` programa para ver imágenes en diferentes formatos: *tiff, pic, jpeg, ..*
- `xmaple` programa de cálculo algebraico y numérico, con capacidad gráfica. Se recomienda realizar el tutorial correspondiente.