Curso de Linux

Antonio Espín Herranz

Contenidos

- Sistema de Archivos
  - Estructura del sistema
  - Navegación básica
  - Manipulación de archivos y directorios

- Comandos básicos:
  - Edición de texto
  - Visualización de archivos
  - Búsqueda de archivos
  - Permisos

- Redirección E/S y tuberías

- Creación de scripts
  - Creación y ejecución de scripts
  - Variables y estructuras de control
  - Expresiones y funciones

Introducción a Unix

- Sistema Operativo de la década de los 70.
- Nace en los laboratorios Bell (BTL) junto a GE y MIT.
- Desarrolladores Ken Thompson y Dennis Ritchie (inventor del lenguaje C).
- Crearon un sistema operativo llamado MULTICS (MULTiplexed Information and Computers Services - Servicio de Información Multiplexada y Cálculo).
- BTL se retira del Proyecto.
- K.T. y D.R. reescriben el S.O. → UNICS que después se transformó a UNIX.
- Entre 1972-73 se reescribe en lenguaje C.
- Hoy día es un sistema operativo versátil y flexible.

Características de Unix

- Sistema operativo abierto.
- Distingue entre mayúsculas y minúsculas.
- Se puede instalar en cualquier plataforma de Hardware.
- Unix está escrito en Lenguaje C, ventaja para los programadores para añadir nuevas características.
- Es un sistema multiusuario y multitarea.
- Comparte recursos en tiempo real.
- Entorno flexible, se utiliza en múltiples campos: negocios, ciencias, educación e industria.

¿Qué es Linux?

- **Linux** es un sistema operativo **distribuido gratuitamente** basado en el sistema operativo UNIX.

- Fue desarrollado originalmente por **Linus Torvalds**, quien empezó a trabajar sobre Linux en **1991** siendo **estudiante** de la Universidad de Helsinki en Finlandia.

- Contribuyeron miles de desarrolladores y se distribuyo por internet y **Linus Torvalds** desarrolló el **kernel** de Linux.

Características de Linux I

- **Multitarea**.
  - Puede ejecutar varios programas al mismo tiempo. Hasta 2^{22} PID (*Process ID*)

- **Multiusuario**.
  - Los usuarios se distribuyen en grupos de usuarios y es necesaria una autenticación.

- **Multiplataforma**.
  - Se ha portado a un gran número de plataformas hardware: Intel 32 y 64 bits. IBM, Sun Sparc, ARM (en particular bajo Android).

- **Sistema de archivos**.
  - Los de tipo Unix, CD-Dom, DVD-Rom, Bluy-Ray, VFAT, NTFS, etc...

Características de Linux II

- **Administración de la memoria**.
  - Reduce las operaciones de swap (de procesos completos). Pagina la memoria virtual para limitar el número de accesos a disco.

- **Redes**.
  - Linux posee una capa de red fiable y rápida.
  - Soporta gran cantidad de protocolos: TCP/IP v4 y v6, ...

- Adecuación a múltiples **estándares** como **POSIX**.

- Distingue entre MAYÚSCULAS / minúsculas

Distribuciones de Linux

- Una distribución de Linux necesita:
  - Un núcleo (**kernel**) a parte necesita un conjunto de herramientas estándar y programas libres básicos comunes a todos los Unix.
  - Herramientas GNU, por ejemplo, el **compilador** de C (GNU C/C++).
  - Un **conjunto de comandos accesibles desde la Shell** como grep, find, awk, etc.
  - Un **editor** de textos **vi**.
  - A parte, **herramientas de administración** como un instalador o el **gestor de paquetes** software.

Distribuciones de Linux

- Las **10 distribuciones principales**:
  - **Mandrake Linux**, desarrollado por MandrakeSoft.
  - **Red Hat Linux**, desarrollado por Red Hat
  - **Debian** GNU/Linux, desarrollado por Debian.
    - Ubuntu (se basa en Debian)
  - **SuSE Linux**, desarrollado por SuSe, Inc.
  - **Gentoo Linux**, desarrollado por Gentoo Technologies, Inc.
  - El Proyecto **Slackware Linux**, desarrollado por Slackware Linux, Inc.
  - **Lycoris** Desktop, desarrollado por Lycoris
  - **Beehive** Linux, desarrollado por el Equipo Beehive
  - Caldera **OpenLinux**, desarrollada por Caldera Internacional, Inc.
  - **Turbolinux**, desarrollado por Turbolinux, Inc.

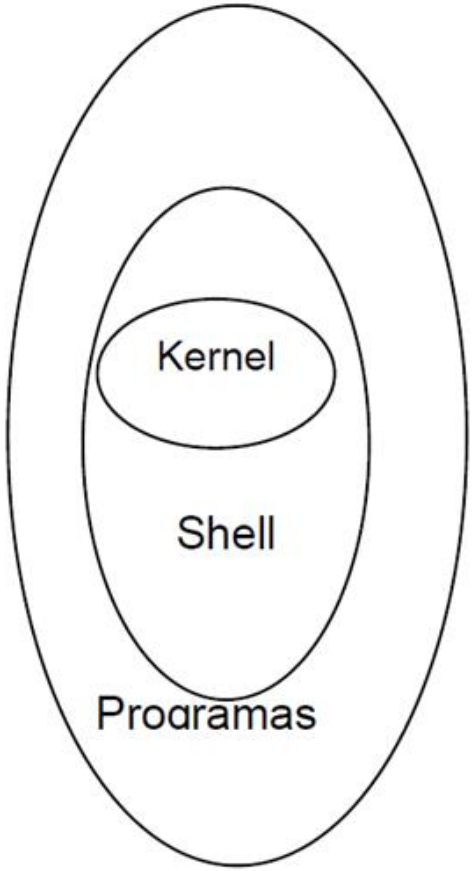
Organización de Linux

- Se encuentra organizado en 3 niveles:

  - **Kernel** (el núcleo).
    - Interactúa directamente con el HW de la máquina.
    - Verificar si el usuario es un usuario autorizado.
    - Control de los programas que se ejecutan.
    - Asignar espacio de almacenamiento a los archivos del sistema.
    - Ejecutar la Shell

  - **Shell** (interprete de comandos)
    - Actúa como un intérprete entre los programas de los usuarios y el kernel
      - La Shell en Linux es **bash** (Bourne Again Shell)
      - Otros tipos de Shell en Linux:
        - **chs** (C Shell)
        - **ksh** (Korn Shell)
        - **sh** (Shell)
        - **esch** (Enhanced C Shell)

  - **Herramientas** y **aplicaciones (Programas)**
    - Compiladores, aplicaciones de negocio, procesamiento de texto, etc.



Sistema de archivos

- Estructura del sistema
- Navegación básica
- Manipulación de archivos y directorios

Directorios principales de Linux

- El árbol de Linux. Los principales directorios que cuelgan de la **raíz /**
- Estos directorios están relacionados con programas ejecutables.
  - **/bin**
    - Contiene los ejecutables (binarios) básicos para el funcionamiento del sistema. Se encuentran comandos como date.
  - **/sbin**
    - Los super binarios. Programas fundamentales para la administración del sistema.
  - Los dos siguientes están relacionados con librerías compartidas que utilizan los programas binarios. De 32 y 64 bits
    - **/lib**
    - **/lib64**
  - Estas carpetas vienen a ser el equivalente a las carpetas de Windows y system32 en el S.O. de Microsoft Windows.

Directorios principales de Linux

- **/boot**
  - Contiene el núcleo de Linux. Contiene archivos que se ejecutan durante el arranque del sistema.

- **/dev**
  - Linux presenta los dispositivos conectados al sistema a modo de ficheros. Como son, por ejemplo: la terminal de texto (/dev/tty0) y el CD Rom (/dev/cdrom).

- **/home**
  - La carpeta home hace referencia a los archivos personales de los distintos usuarios. Se crearía una carpeta a partir de home por cada uno de los usuarios del sistema.
  - También se guardan ficheros de configuración propios del usuario. Por ejemplo, la configuración del editor vi estaría en el fichero: /home/user/.exrc (siendo user el nombre del usuario).
  - Viene a ser la carpeta users o documents and settings en Windows.

Directorios principales de Linux

- **/root**
  - Tiene la misma función que el anterior, pero está reservado al usuario root. No se encuentra en /home por razones de seguridad.

- **/tmp**
  - Es un directorio temporal. Todos los usuarios pueden escribir en esta carpeta pero es el administrador el que planifica la limpieza de esta carpeta.

- **/lost+found**
  - Este directorio se crea automáticamente al arrancar el sistema. Lo utiliza la herramienta fsck (File System Check) para guardar los archivos recuperados tras un incidente (por ejemplo: un corte de luz).

Directorios principales de Linux

- **/mnt**
  - Es un directorio vacío o que contiene una serie de directorios vacíos predefinidos. Se reserva para el montaje de sistemas de archivos a terceros.
  - En los sistemas más actuales se reserva para el montaje de USBs la carpeta /media.
  - Por ejemplo, para las carpetas compartidas (de una máquina virtual) .
- **/proc**
  - Es un sistema de archivos virtual que representa el estado del sistema en curso de ejecución. No ocupa en disco, si no en RAM.
  - Cada proceso que se inicia en el sistema. Existe un directorio que lo caracteriza en /proc. Los comandos como **ps** utilizan esta información.

Directorios principales de Linux

- **/sys**
  - Parecido a /proc. Su meta es representar los distintos periféricos e indicar sus características.

- **/usr**
  - Es el directorio más voluminoso al instalar el sistema, contiene todos los programas que no están en /bin y /sbin.
  - Dentro de este directorio podemos encontrar subdirectorios como: /usr/bin, /usr/sbin y /usr/lib para programas, para juegos, código fuente, etc.

Directorios principales de Linux

- **/var**
  - Otro directorio voluminoso. Reúne todos los archivos de datos “variables”: por ejemplo: las colas de espera de impresora, los buzones de usuarios y los registros del sistema.
    - Por ejemplo: /var/spool: Agrupa los archivos de espera (spool) de diferentes servicios de impresión.
- **/run**
  - Al igual que /proc y /sys. El directorio /run aparecido hace poco, en un sistema de archivos virtual que no ocupa lugar en disco.
  - Agrupa archivos de aplicaciones residentes en memoria como los archivos de bloqueo o que contengan el PID de los servicios ejecutados.

Directorios principales de Linux

- **/etc**
  - Etcétera, para guardar otros tipos de archivos que no tienen cabida en los otros directorios. Como pueden ser archivos de configuración y scripts de arranque del sistema.

Comunicación Linux Windows desde WSL

- Cuando estamos en la parte de Linux podemos entrar a la parte de Windows a través de la carpeta **/mnt** (se ven las unidades que tengamos dentro de Windows como puntos de montaje)

- Prueba:
  - **cd /mnt**
  - **ls** → debería de mostrar las unidades que tenemos en Windows
  - A partir de aquí vamos navegando por las unidades de Windows.

- Desde Windows, en el explorador tendremos una entrada para Linux. Podemos navegar por las carpetas de Linux,

Ruta absolutas y relativas

- Rutas **absolutas**: Una ruta absoluta se basa en la raíz de un árbol. Toda ruta absoluta empieza por **/**

- Todo fichero o directorio se puede referenciar mediante la ruta absoluta.

- Se puede aplicar a todos los comandos de Linux en los que se especifique un archivo o directorio.

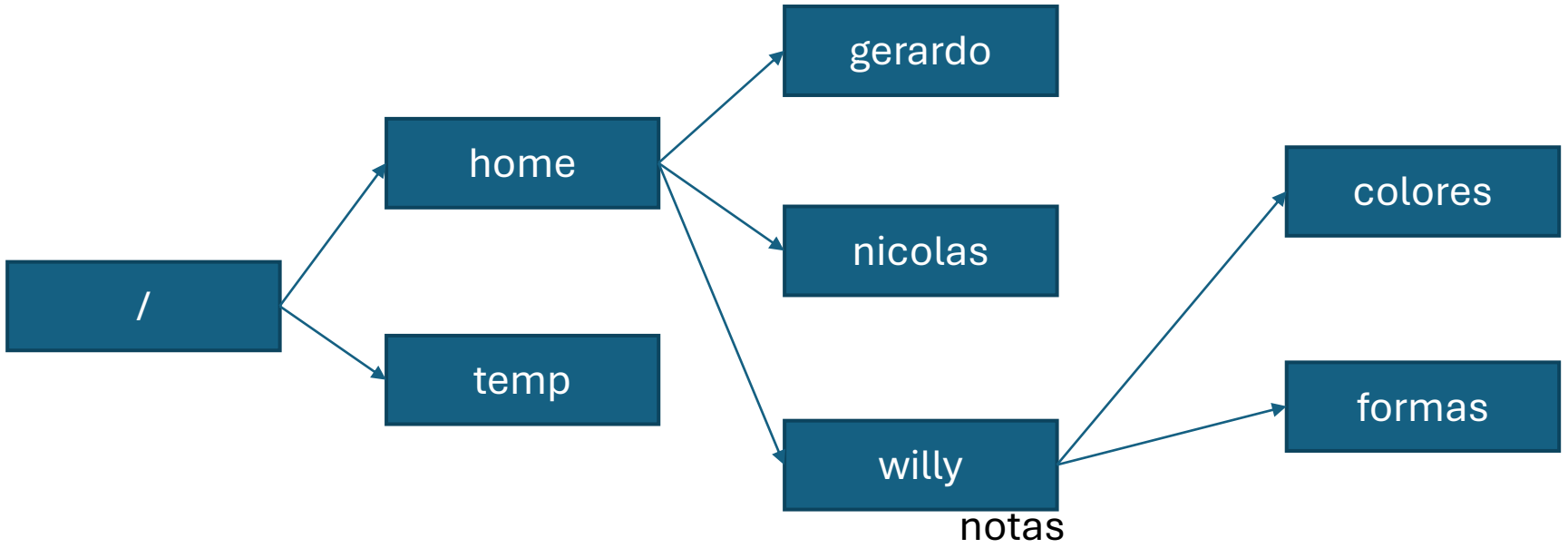
- Ejemplos:

  ls /home/willy/documentos
  cat /usr/admin/cuentas.dat

Rutas absolutas y relativas

- Las rutas relativas dependen del directorio actual en el que se encuentra el usuario.
- Cada directorio del sistema, contiene los archivos: `.` y `..`
- El archivo `.` referencia al directorio actual.
- El archivo `..` al directorio padre

Ejemplo de árbol



Rutas absolutas y relativas

Directorio actual	**Ruta relativa correspondiente**
home	willy/notas
/home/willy	./notas
/home/willy/colores	../notas
/	home/willy/notas
/home	../tmp/./../home/./willy/notas

Ejecutar programas

- Si en Linux desarrollamos programas en C o C++ una vez compilados con gcc o g++ podemos ejecutarlos con:

  ./nombrePrograma

- Los comandos de Linux se ejecutan con el nombre.
  - ls, cat, more, etc.

Operaciones con directorios

- **pwd**: (print working directory) El directorio actual.

- **ls**: listar archivos y directorios de un directorio.
  - Sirve la para listar los ficheros y directorios de un directorio.
  - Parámetros más habituales:
    - **ls -a** Muestra los directorios: actual y padre. Representados por . y ..
    - **ls -l** Muestra un listado en el formato largo, con información de permisos, número de enlaces asociados al archivo, usuario, grupo, tamaño, fecha y hora de la última modificación, además del nombre.
  - Los permisos:
    - d**rwx**rwx**rwx**, d: Directorio, r: read, w: write, x: ejecución
    - Los 3 primeros para el dueño del archivo, los 3 siguientes para el grupo y los 3 últimos para otros usuarios. Los permisos se pueden cambiar con **chmod**.

Operaciones con directorios

- Se pueden combinar los parámetros: **ls -la**

```
drwxr-xr-x  21 root root     4096 jul 11 18:27 lib
drwxr-xr-x   2 root root     4096 feb 10  2019 lib64
drwx------   2 root root    16384 jul 11 18:22 lost+found
drwxr-xr-x   4 root root     4096 sep  8 17:05 media
drwxr-xr-x   2 root root     4096 feb 10  2019 mnt
drwxr-xr-x   3 root root     4096 jul 11 18:31 opt
dr-xr-xr-x 199 root root        0 dic 25 09:56 proc
drwx------   3 root root     4096 dic 25 09:56 root
drwxr-xr-x  28 root root      940 dic 25 10:27 run
drwxr-xr-x   2 root root    12288 dic 25 10:25 sbin
drwxr-xr-x  13 root root     4096 jul 19 18:34 snap
drwxr-xr-x   2 root root     4096 feb 10  2019 srv
```

Opciones del comando ls (i)

- **ls -a**
  - Nos muestra los archivos y directorios dentro del directorio actual.
  - Incluyendo los archivos y directorios ocultos.
- **ls -t**
  - Ordena los archivos por fecha de modificación.
- **ls -r**
  - Ordenar de forma inversa.
- **ls -X**
  - Ordena los archivos por extensión.
- **ls -l**
  - Muestra toda la información: usuario, grupo, permisos, tamaño, fecha y hora de creación.
- **ls –d**
  - Muestra solo los directorios. En WSL sólo muestra el directorio actual.

Opciones del comando ls (ii)

- **ls -lh**
  - Muestra la misma información que ls -l pero con las unidades de tamaño en KB, MB, etc.
- **ls -R**
  - Muestra el contenido de todos los subdirectorios de forma recursiva.
- **ls -S**
  - Ordena los resultados por tamaño de archivo.
- **ls -i**
  - Muestra el número de i-nodo donde están los ficheros y directorios.
- **ls -1**
  - En una columna.
- **ls -m**
  - Es una sola fila separados por comas.

PRACTICAS LS

Operaciones con directorios

- **cd:** Cambiar de directorio.

cd ..	Sube al directorio padre.
cd directorio	Cambia al directorio. Tiene que ser visible desde donde estamos.
cd	Nos lleva al directorio home. Sin indicar parámetros.
.	Un punto representa el directorio actual.
..	Doble punto el directorio padre.

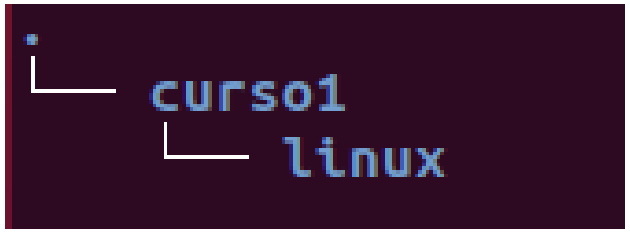
Utilizar **tabulador** para completar las rutas.

Por ejemplo, si en el directorio actual hay un directorio llamado: curso

cd c <<tab>>

cd $HOME	Nos lleva a la carpeta personal.

Operaciones con directorios



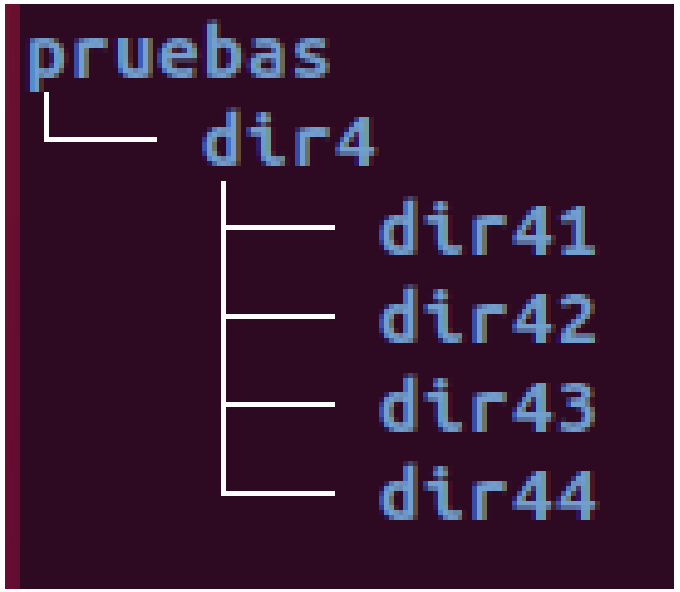
- **mkdir**
  - Para crear un directorio. **mkdir nuevo_directorio**
  - **Opciones:**
    - -p (parents)
    - -v (verbose), informa de cada directorio eliminado.

  - Creando paso a paso:
    - $ mkdir curso1
    - $ cd curso1
    - $ mkdir linux

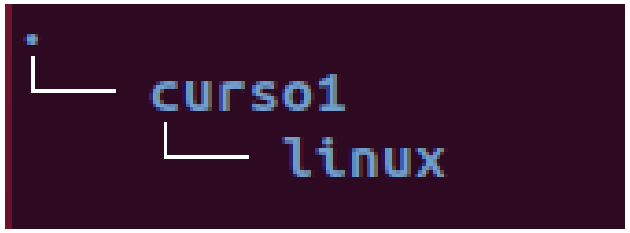
  - En un solo paso:
    - **$ mkdir -p curso1/linux**

Operaciones con directorios

- **mkdir**
  - Para crear varios directorios (hermanos).
  - Por ejemplo, a partir del directorio actual (**home**) queremos crear el siguiente árbol.
  - En este caso pruebas ya existe.
  - **$ mkdir -p pruebas/dir4/{dir41,dir42,dir43,dir44}**



Operaciones con directorios



- **rmdir**
  - Para eliminar directorios. **rmdir directorio**
  - **Opciones:**
    - -p (parents)
    - -v (verbose), informa de cada directorio eliminado.

  - Para eliminar los dos directorios paso a paso:
    - Si intentamos rmdir curso1 → Error no está vacío.
    - $ cd curso1
    - $ rmdir linux
    - $ cd ..
    - $ rmdir curso1

  - Para eliminar los dos directorios en un solo paso:

  - Situados por encima de “curso1”
  - $ **rmdir -p curso1/linux**

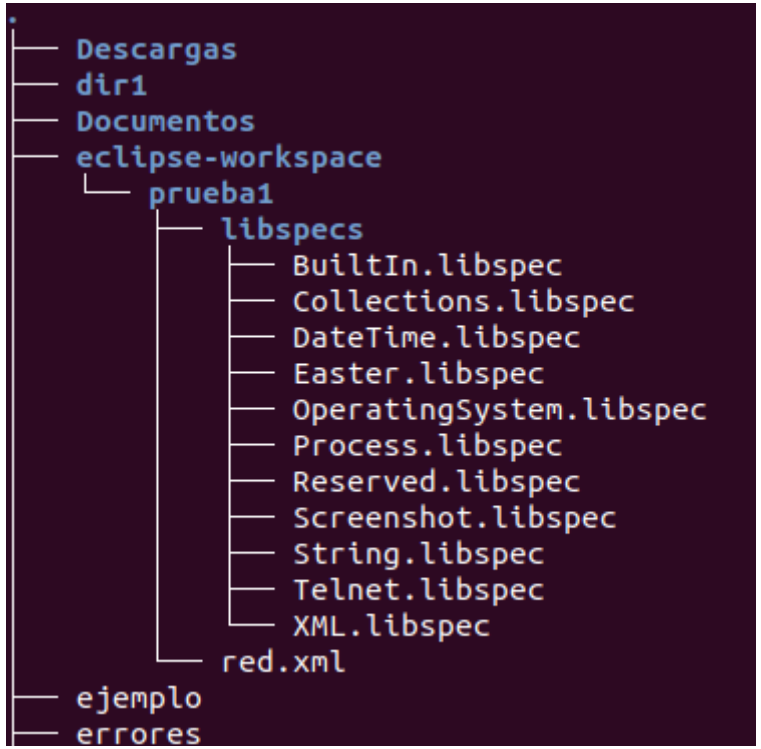
PRACTICAS DIRECTORIOS-1

Operaciones con directorios: tree

- El comando **tree** muestra una vista jerarquizada de los directorios que se encuentran por debajo del directorio indicado.

- Por ejemplo: **tree .**

- Si el comando no está instalado:
  - **sudo apt install tree**



Operaciones con directorios

- **du *(disk usage)***
  - Muestra los directorios y lo que ocupan en bloques a partir de un directorio. **du subdirectorio**

```
antonio@antonio-VirtualBox:~$ du *
4       Descargas
4       dir1
4       Documentos
356     eclipse-workspace/prueba1/libspecs
368     eclipse-workspace/prueba1
8       eclipse-workspace/.metadata/.mylyn/.taskListIndex
4       eclipse-workspace/.metadata/.mylyn/contexts
32      eclipse-workspace/.metadata/.mylyn
12      eclipse-workspace/.metadata/.plugins/org.eclipse.m2e.logback.configuration
4       eclipse-workspace/.metadata/.plugins/org.eclipse.debug.core
16      eclipse-workspace/.metadata/.plugins/org.eclipse.jdt.ui
4       eclipse-workspace/.metadata/.plugins/org.eclipse.mylyn.context.core/contexts
8       eclipse-workspace/.metadata/.plugins/org.eclipse.mylyn.context.core
60      eclipse-workspace/.metadata/.plugins/org.eclipse.core.runtime/.settings
64      eclipse-workspace/.metadata/.plugins/org.eclipse.core.runtime
8       eclipse-workspace/.metadata/.plugins/org.eclipse.oomph.setup
12      eclipse-workspace/.metadata/.plugins/org.eclipse.ui.workbench
4       eclipse-workspace/.metadata/.plugins/org.eclipse.tm.terminal.view.ui
```

OPCIONES:

-a	El recuento de todos los archivos & dirs.
-h	Formato legible humanos: kb, Gb, etc.
-c	Imprime al final el total.
-d n	Profundidad máxima. **du -d 2**

Ejemplo:

- Muy útil cuando queremos ver el total con un nivel y luego ordenar por los subdirectorios que más ocupan.
  - du . –h –d 1 | sort –k1 –rh

- Para directorios de Windows como la carpeta: AppData si tenemos instalado: **WSL**
  - Navegar primero a la carpeta **/mnt** y accedemos a las carpetas de Windows → c:\users**usuario**\AppData
  - Y lanzar el comando:
    - du . –h –d 1 | sort –k1 –rh

Comandos básicos

- Edición de texto
- Visualización de archivos
- Búsqueda de archivos
- Permisos

Los comandos Linux

- Existen una gran cantidad de comandos en Linux.
- Se suele trabajar desde una terminal.
  - **$ comando**

- Los comandos pueden tener o no parámetros, estos irán después del comando y **separados por espacios en blanco**.
  - **$ comando arg1 arg2**

- Si son opciones del comando irán precedidas por un **-**
- Estas opciones modificarán el comportamiento por defecto del comando.

Los comandos Linux II

- Todos los comandos tienen la opción **--help** y **--version**
- Por ejemplo, comando **ls** para listar ficheros y directorios:
  - **ls --help**
  - **ls --version**

- En un comando las opciones se suelen especificar en formato corto, pero también se pueden especificar en formato largo con el doble guion.
  - Los comandos con las opciones con **un guión** provienen de **Unix familia BSD**.
  - Los comandos con **doble guión** son **reescrituras del comando GNU**.

Los comandos Linux III

- En resumen:
  - Se escriben en minúsculas.
  - Los comandos pueden ser simples sin ningún parámetro y ninguna opción: **clear**
  - Sintaxis: **comando [-opciones] [argumentos]**
  - Pueden tener opciones (precedidas de un guión): **ls -l -a**
  - Las opciones si son varias se pueden fusionar con un solo guión: **ls -la**
  - Los argumentos van sin guiones. **ls -la /etc**
  - El parámetro es: /etc

  - Que ocurre si el parámetro empieza por un guión. Por ejemplo, visualizar un fichero que el nombre empieza por **-**
  - **cat -numeros** → en estos casos se utilizan **--** para avisar que lo que viene es una cadena (argumento) NO una opción del comando: **cat -- -numeros**

Prueba

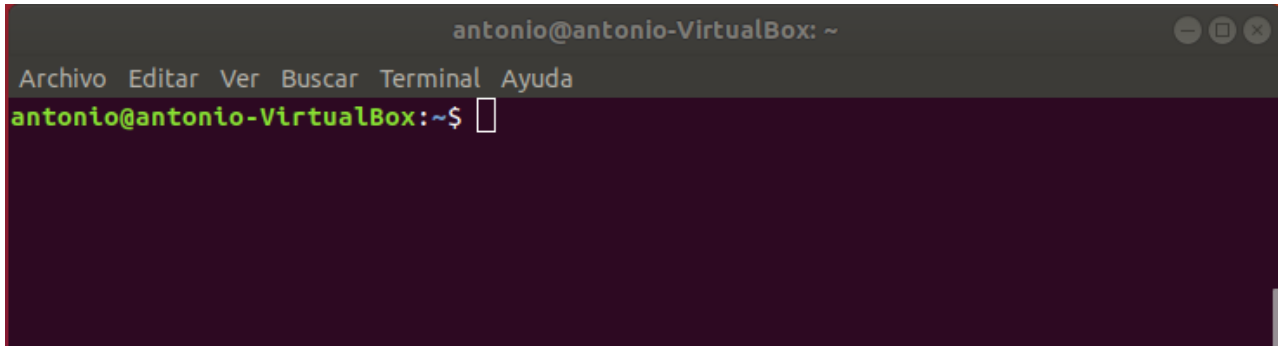
- Generar un fichero con echo. Escribir un mensaje con echo y redirigir a un fichero que empiece por – (guión medio)
  - echo mensaje > -numeros
- Tratar de visualizar el contenido del fichero con: cat –números
- Anular el efecto del guión medio con doble guión medio delante:
  - cat -- -números
- Sólo necesario cuando el fichero empiece por -

Nuevos comandos

- Es posible que algunos comandos no estén disponibles, pero, se pueden instalar:
  - Primero actualizar, si no hacemos este paso primero no nos deja instalar nada.
  - **sudo apt-get update**

  - Luego podemos probar a instalar **gedit**:
    - **sudo apt install gedit**

En la terminal



- Dentro de la terminal se interactúa con la Shell (el intérprete de comandos).
  - Las flechas repiten el último comando.
  - Se pueden completar los argumentos del comando (cuando son ficheros / dirs) con el tabulador.
  - **Control+c** corta la ejecución de un comando.
  - **Control+d** envía fin de fichero.
    - Probar comando **cat** sin parámetros → repite la entrada
    - O mejor **cat > nombreFichero** → redirige al fichero

- Los comandos se pueden ejecutar como super-usuario si no tenemos suficientes privilegios.
- **sudo** <<seguido del comando>> solicitará la contraseña.

El manual: comando man

- La documentación se instala dentro de Linux en formato electrónico y es posible consultarla con el comando **man**.

- Para ejecutar el comando: **man [comando]**
- **Si no se indica el comando, pregunta.**
- Incluso se puede consultar el propio comando man → **man man**
  - Nos muestra información sobre la utilización del manual.

- Para salir del comando man teclear (**q**)uit y la (**h**)elp

Descripción de la página del manual

- Principalmente tiene estas secciones:
  - **NOMBRE**
    - Nombre del comando.
  - **SIPNOSIS**
    - Sintaxis del comando, los [] indican que es opcional.
  - **DESCRIPCION**
    - Descripción detallada del comando. Contiene una explicación de las diferentes opciones.
  - **AUTHOR**
    - Autor del comando.
  - **VER TAMBIEN**
    - Otras páginas del manual.
- Pueden aparecer también Ejemplos, Bugs, etc.

Comando: man

- Se muestra por defecto con el programa **less**.

- Con el cual podemos desplazarnos:
  - **Desplazarnos** hacia delante y hacia atrás con las teclas de **AVPÁG** y **REPÁG**,
  - **Buscar** una palabra con el carácter “/” seguido de la palabra (“n” nos sirve para buscar las siguientes ocurrencias y “N” para las anteriores),

  - Cuando buscamos dentro del manual un término se recorren las secciones en este orden: 1,8,2,3,4,5,7,9

Comando man (Secciones)

Sección	Descripción
1	Comandos Generales
2	Llamadas al sistema
3	Biblioteca C de funciones
4	Ficheros especiales (normalmente dispositivos, que se pueden encontrar en /dev) y drivers
5	Formatos de fichero y convenciones
6	Juegos y salvapantallas
7	Miscelánea
8	Comandos de administración del sistema y Demonios

Prueba: man 1 intro

Comando man

- Con un número muestra información de una de las secciones.
  - **man 1 intro**
  - man 2 intro
  - …
  - man 8 intro

- Con la opción **-f** muestra información abreviada.
  - **man -f ls**

- Para buscar con una cadena de caracteres sin conocer el nombre del comando:
  - **man -k <cadena de caracteres>**

- La opción **-w** muestra en que página del manual se encuentra un comando.
  - **man -w ls**

Configuración man

- El fichero de configuración para el comando man en Ubuntu es: **/etc/manpath.config**

- El administrador del sistema lo puede editar:
  - **sudo gedit /etc/manpath.config**

- En otros sistemas de Linux puede ser: /etc/man.config.

- ***También disponemos del comando: info que presenta la información del manual de forma jerarquizada.***

- ***Los comandos de Linux también tienen la opción --help y es otra forma de buscar ayuda: ls --help***

Comandos básicos

- Identidad de usuarios:
  - **who**. Muestra los usuarios que hay conectados al sistema. Y a que, hora se inició la sesión y la consola virtual donde está conectado.
    - **who -q**. Muestra sólo el nombre y el número de usuarios. Solo en los sistemas Linux
  - **whoami**. El nombre del usuario que está conectado.
  - **finger**. Muestra información más precisa de la cuenta del usuario conectado.
  - Si no reconoce el comando, se puede instalar con: **sudo apt install finger**
    - En el sistema WSL no muestra a nadie como conectado o logeado.
- Cambio de contraseña:
  - **passwd**. Pide la contraseña actual y luego 2 veces la nueva

Comandos básicos

- Conteo:
  - **wc**. Abreviatura de Word count. Cuenta líneas, palabras y caracteres que hay en un archivo. **wc <nombre_fichero>**

- Visualización:
  - **clear**: limpiar la pantalla.
  - **echo**: visualizar mensajes. **echo “Hola que tal”**
  - **echo mensaje > fichero** → Para redirigir a un fichero

- Tiempo:
  - **date**: muestra la fecha actual
  - **cal**: muestra el calendario del mes y año actual.
    - Se puede indicar el mes y el año: **cal 5 2020**
    - Hay que instalar: **sudo apt install ncal**

Edición de texto

- Editores instalados por defecto:
  - **vi**
  - **nano**
  - **vim** (vi mejorado)

  - Se puede añadir **gedit** (como el block de notas de Windows, pero con soporte para la sintaxis de múltiples lenguajes)

  - Editores multiplataforma como **Visual Studio Code** con la posibilidad de instalar extensiones para distintos lenguajes de programación.

Editor VI

- Edición de archivos.
  - Modos de funcionamiento.
  - Desplazamientos (entre líneas y en la misma línea).
  - Comandos de inserción.
  - Comandos de edición y de corrección.
  - Copiar texto, anular y repetir.
  - Guardar y salir.
  - Buscar y reemplazar.
  - Comandos externos.
  - Resumen de comandos.
  - vimtutor.
  - Otros editores de texto.

Editor VI

- Vi → **Visual Editor**.
- El editor Vi se ha convertido en un **estándar** en los sistemas **UNIX**.
- Actualmente se utiliza más otra variante que es **Vim** (Vi mejorado).
- Sólo funciona con las teclas alfanuméricas básicas (pueden funcionar también las teclas de desplazamiento y calculadora)
- El comando es **vi [nombre de archivo]**
- Para **vim** puede ser necesario **instalar**:
  - **sudo apt install vim**

Inicio en Vi

- Aparece un texto de bienvenida que no afecta al contenido del fichero.
- El carácter **~** significa que no hay caracteres (está en el final del archivo).
- Hay que tener en cuenta que tiene **varios modos de funcionamiento**.
  - El pulsar la misma tecla según el modo que estemos, puede tener un significado distinto. Por ejemplo, la tecla ‘q’ en modo comando sale del editor.
- La **tecla de Esc** permite cambiar de modo (nos lleva al modo comando).

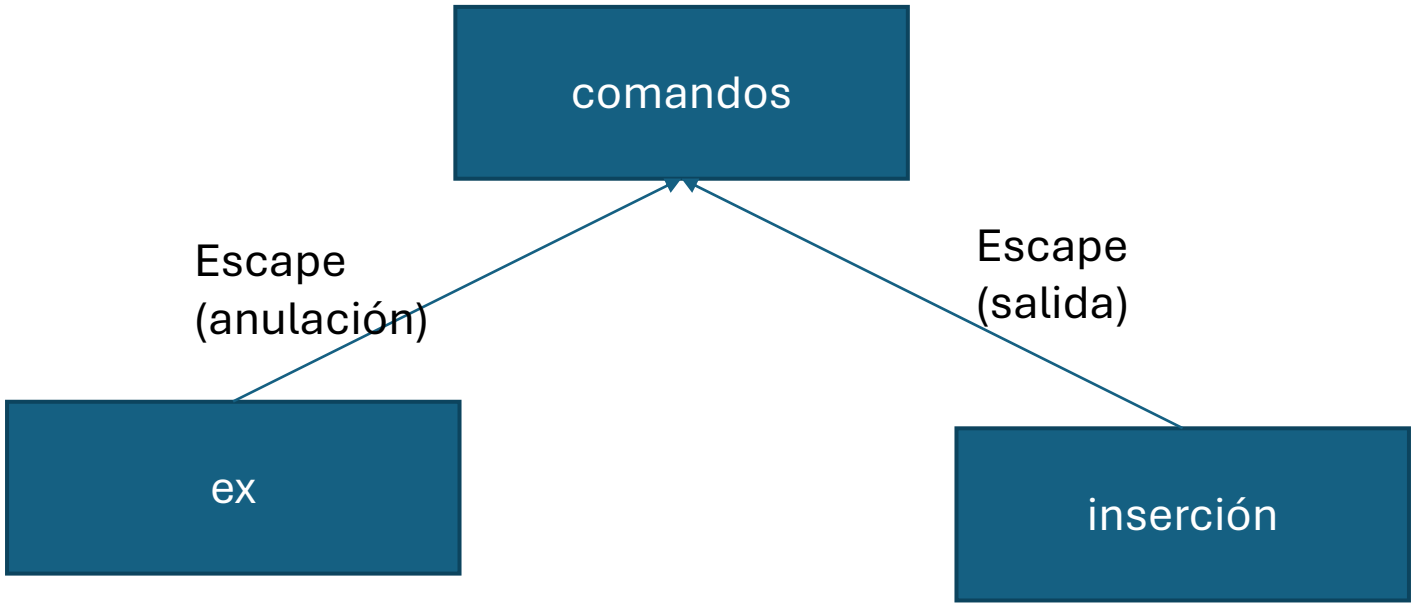
Modos de funcionamiento

- Modo comando:
  - Cada tecla o combinación de teclas se interpretan como un comando.

- Modo inserción / edición:
  - Cada tecla se insertará en el fichero que estamos editando.

- Modo “inferior de la pantalla”, “ex” o “global”:
  - Permite ordenar a vi la ejecución de comandos complejos que se teclean en la parte inferior de la pantalla.

Modos de funcionamiento



Desplazamientos (por líneas)

- Sin utilizar las teclas de desplazamiento:
  - h → a la izquierda
  - L (minúsculas) → a la derecha
  - j → abajo
  - k → arriba

Desplazamientos (dentro de la misma línea)

- Cuando estamos situados en una línea:
  - 0 Mueve el cursor al principio de la línea.
  - $ Mueve el cursor al final de la línea.

  - w Moverse por palabras. Como separador toma cualquier carácter
  - W También por palabras, pero el separador tiene que ser: espacio, tabulación y salto de línea.

  - b Primer carácter de la palabra anterior, teniendo en cuenta lo de ‘w’
  - B Primer carácter de la palabra anterior, teniendo en cuenta lo de ‘W’

  - e, E Iguales que la anterior, pero con el último carácter de la palabra.

Comandos de inserción

- Desde el modo comando:
- **i** insert Para insertar nuevos caracteres delante del cursor.
- **a** append Para añadir al final, permite escribir al final de la línea.
- **o** open Abre una nueva línea debajo del cursor. Y después pasa al modo de inserción.

- Estos mismos comandos en mayúsculas:
- **I** Mueve el cursor al principio de la línea y pasa al modo inserción.
- **A** Mueve el cursor al final de la línea y pasa al modo inserción.
- **O** Abre una nueva línea encima del cursor y pasa a modo inserción.

Comandos de edición y de corrección

- Con **VIM** podemos utilizar **teclas** de **retroceso** y **suprimir**.

- Y disponemos de los comandos:
  - **x** Borra el carácter situado bajo el cursor.
  - **dd** Borra la línea entera.
  - **d0** Borra desde el cursor hasta el inicio de la línea.
  - **d$** Borra desde el cursor hasta el final de la línea.
  - **dw** Borra desde el cursor hasta el inicio de la palabra siguiente.

  - Se pueden combinar con las anteriores: dW, de, dE ...

- Todos los comandos que empiezan por **d**, **permiten cortar**.
- Se puede pegar lo cortado con **p** (detrás del cursor), **P** (delante del cursor).

Copiar texto

- Los comandos que empiezan por y permiten copiar:
  - **yy** Copia toda la línea
  - **y0** Copia del cursor al principio de la línea
  - **y$** Del cursor al final.

- Para pegar con **p** o **P**

Anular y repetir

- El comando **u anula** el **comando anterior**.
- Vim permite la anulación sucesiva.
- **‘U’ anula de golpe todas** las modificaciones aportadas a la línea actual.
- El ‘**.**’ (**punto**) **repite** el comando en Vi excepto, desplazamientos y anulaciones.

Guardar y salir

- :w — Guarda en el archivo existente.
- :w archivo — Guarda en el archivo indicado.
- :w! — Fuerza la grabación, el fichero debe pertenecer al usuario.
- :q — Para salir, si hay cambios no sale (avisa).
- :q! — Fuerza la salida sin grabar.

- :wq — Con las 3 opciones guarda y sale.
- :x
- ZZ

Buscar

- Para **buscar** una cadena de caracteres.
- El carácter **/**
  - Busca desde el cursor **hasta el final** del archivo.
- El carácter **?**
  - Busca desde el cursor **hasta el principio** del archivo.

- El carácter y la palabra se escriben en la parte inferior de la pantalla del editor.
- Si el comando llega al final empieza por el principio.
- En las búsquedas se pueden utilizar expresiones regulares.

- Una vez se ha localiza la cadena buscada, “en modo comando” y con la tecla “**n**” se pueden buscar las siguientes ocurrencias. Con “**N**” a las anteriores.

Buscar y reemplazar

- Sintaxis general del comando:
  - :[rango]s/cadena_a_reemplazar/nueva_cadena/[g][i][I]

  - Con el rango se indican las líneas: n,m (primera y última línea)
  - La s es obligatoria (sustituir).
  - Para señalar la última línea se puede indicar con el $
  - Ejemplos:
  - 1,45 → Entre la 1 y la 45
  - 4,$ → De la 4 a la última.

- Si alguna de las dos cadenas presentan el carácter / hay que escaparlo con una \ para que no se interprete como el separador de ambas cadenas.

Buscar y reemplazar

- g Para que se sustituyan todas las instancias. Si no se indica sólo se cambiará la primera.

- i No se consideran mayúsculas y minúsculas.
  - Estas dos cadenas serían iguales Madrid ← → Madrid

- I Se tienen en cuenta las mayúsculas y las minúsculas. Este es el modo por defecto.

- Ejemplo: Sustituir en todo el fichero: hola por adiós (todas las ocurrencias)
- :1,$s/hola/adiós/g

Comandos externos

- Desde el mismo editor se pueden ejecutar comandos externos (como en la Shell) sin necesidad de salir del editor.
  - Esto nos permite escribir el programa, compilar, ejecutar sin necesidad de salir o hacerlo desde otra Shell.

- También se puede insertar el resultado del comando dentro del fichero que estamos editando.

- Por ejemplo, para insertar la fecha del sistema:
  - **:r!date**

Resumen de comandos

- **dd** borra una línea completa.
- **i** inserta texto antes del carácter sobre el que está el cursor.
- **a** inserta texto después del carácter sobre el que está el cursor.
- **I** inserta texto al comienzo de la línea en la que está el cursor.
- **A** inserta texto al final de la línea en la que está el cursor.
- **o** abre espacio para una nueva línea después de la línea en la que está el cursor y permite insertar texto en la nueva línea.
- **O** análogo al anterior, pero abre espacio en la línea anterior.
- **ESC** como hemos dicho antes, permite bandonar el modo de inclusión de texto para volver al modo de comandos; también se usa para cancelar comandos. (Usarlo en caso de duda).
- **Ctrl-F** avanzar una pagina hacia adelante.
- **Ctrl-B** avanzar una pagina hacia atrás.
- **Ctrl-L** refrescar la pantalla.

Resumen de comandos II

- **G** poner el cursor al final del fichero.
- **1G** poner el cursor al principio del fichero.
- **XG** poner el cursor en la línea X.
- **$** poner el cursor al final de la linea.
- **0 (cero)** poner el cursor al principio de la linea.
- **/ texto** busca el texto en el fichero y sitúa el cursor delante de la primera coincidencia que encuentra.
- **:set number** pone el número de línea de cada una.
- **:q** salir del editor si no ha habido cambios.
- **:q!** salir del editor sin guardar los cambios.
- **:w** guardar los cambios.
- **:wq** guardar los cambios y salir del editor.
- **ZZ** guardar los cambios y salir del editor.

vimtutor

- Guía del editor para repasar comandos.
- Seguir el tutorial.

PRACTICAS VIMTUTOR

Conceptos y características de archivos

- Linux admite nombres de fichero largos y se puede utilizar cualquier carácter excepto **/**.

- De todas maneras, no es recomendable usar los siguientes caracteres, por tener significado especial en Linux:
- **\ ^ ~ ! # ? & ( ) ´ " ` ; · $ = ¿ ¡ < > @ { } * + -**

- A la hora de diferenciar un fichero de otro, **Linux distingue mayúsculas y minúsculas**, por lo que los ficheros "texto1.txt" y "Texto1.txt" son ficheros distintos.

Conceptos y características de archivos

- El sistema de ficheros de Linux permite al usuario crear, borrar y acceder a los ficheros sin necesidad de saber el lugar exacto en el que se encuentran.

- En Linux **no existen unidades físicas**, sino ficheros que hacen referencia a ellas, integrados en la estructura de ficheros como cualquier otro.

- El sistema de ficheros de Linux consta de varias partes importantes:
  - **Superbloque**
  - **Tabla de inodos**
  - **Bloques de datos**

- En **Linux cada bloque es de 512 bytes** o de múltiplos de 512. Al igual que el cluster era la estrella del sistema de ficheros FAT, en **ext3** es el **bloque**.

Conceptos y características de archivos

- El **bloque de carga** o bloque cero de cada sistema está reservado para almacenar un programa que utiliza el sistema para gestionar el resto de las partes del sistema de ficheros.

- El **superbloque** o bloque uno contiene la información sobre el sistema de ficheros.

- La **tabla de inodos** es el equivalente a las entradas de la FAT. Por cada fichero, Linux tiene asociado un elemento en esta tabla que contiene un número. Este número identifica la ubicación del archivo dentro del área de datos.
  - Cada inodo almacena información de un archivo o un directorio.

Información que almacena un inodo I

- El **identificador de dispositivo** del dispositivo que alberga al sistema de archivos.
- El **número de inodo** que identifica al archivo dentro del sistema de archivos.
- La **longitud del archivo** en bytes.
- El **identificador de usuario del creador** o un propietario del archivo con derechos diferenciados.
- El **identificador de grupo** de un grupo de usuarios con derechos diferenciados.

Información que almacena un inodo II

- El **modo de acceso**: capacidad de leer, escribir, y ejecutar el archivo por parte del propietario, del grupo y de otros usuarios.

- Las **marcas de tiempo** con las fechas de **última modificación** (mtime), **acceso** (atime) **y de alteración** del propio inodo (ctime).

- El **número de enlaces** (hard links), esto es, el número de nombres (entradas de directorio) asociados con este inodo.
  - El número de enlaces se emplea por el sistema operativo para eliminar el archivo del sistema de ficheros, tanto el inodo como el contenido, cuando se han borrado todos los enlaces y el contador queda a cero

Fechas de un archivo

- Cualquier archivo tiene 3 marcas de tiempo:
  - **ctime (change time):** Se actualiza cuando se modifican algunos de los campos del i-nodo del fichero (permisos, dueño, grupo y enlaces duros), se mueve el fichero a otro directorio, se le cambia de nombre y también cuando se modifica. Puede consultarse ejecutando el comando ***ls –lc***

  - **mtime (modify time)**: Se actualiza cuando se modifica el fichero, es decir, cuando se escribe sobre los bloques de datos del fichero. La mayoría de las veces *ctime* y *mtime* coincidirán. Puede consultarse ejecutando el comando ***ls -l***.

  - **atime (access time):** Se actualiza cuando se abre el fichero y se leen todos o algunos de sus bloques de datos. Multitud de comandos modifican el *atime*: *cat*, *grep*, *head*, *tail*, etc. Puede consultarse ejecutando el comando ***ls -lu***.

Sistema de ficheros

- Linux soporta gran variedad de sistemas de ficheros, desde sistemas basados en discos, como pueden ser:
  - **ext2, ext3, ReiserFS, XFS, JFS, UFS, ISO9660, FAT, FAT32 o NTFS**, a sistemas de ficheros que sirven **para comunicar equipos en la red de diferentes sistemas operativos**, como **NFS** (utilizado para compartir recursos **entre equipos Linux**) o **SMB** (para compartir recursos entre máquinas **Linux y Windows**).

- El sistema **ext2** hasta hace poco era el estándar.
  - Fragmentación muy baja.
  - Compatible con ficheros grandes: 2 GB.
  - Nombres de ficheros largos: 255 chars.
  - Establece y actualización.

Sistema de ficheros II

- **ext3:** Es la versión mejorada de ext2, con previsión de pérdida de datos por fallos del disco o apagones.
- En contraprestación, **es totalmente imposible recuperar datos borrados**.
- Es compatible con el sistema de ficheros ext2. Actualmente es el más difundido dentro de la comunidad GNU/Linux y es considerado el estándar.
- Sus ventajas frente a ext2 son: Actualización. Debido a que los dos sistemas comparten el mismo formato, es posible llevar a cabo una actualización a ext3, incluso aunque el sistema ext2 esté montado.
- Fiabilidad y mantenimiento.

Tipos de archivos

- **Archivos ordinarios**: Información con la que trabaja cada usuario.
- **Enlaces físicos o duros** (**hardlinks**) es un segundo nombre de un archivo. Sirve para compartir el mismo archivo por dos usuarios. Cada actualización se muestra. Hacen referencia al **inodo** del archivo.
- **Enlaces simbólicos**: es un segundo nombre, pero no referencia al inodo sino al nombre del archivo. Es como un alias del archivo.
- **Directorios**: Archivos especiales que contienen referencias a otros archivos o directorios.
- **Archivos especiales**: Representan dispositivos físicos: unidades de almacenamiento, impresoras, terminales.

Visualizar archivos: cat, head

- **cat**
  - Visualizar el contenido de un fichero.
  - **cat fichero_de_texto.** Se pueden indicar varios ficheros: **cat f1 f2 f3**
  - **cat** (sin parámetros) repite la entrada estándar. Cortar con Control+d (envía un EOF).
  - **cat -n** numera todas las líneas del texto, incluso las vacías. En concreto, pone un número delante de cada línea al mostrarlas por pantalla.
  - **cat -b** numera todas líneas del texto que no sean vacías. Las líneas vacías las muestra pero no les pone ningún número delante.

- **head**
  - Muestra las primeras líneas de un fichero. Por defecto, son 10. Se puede indicar el número de líneas a mostrar.
  - Las 5 primeras: **head -5 fichero_texto**
  - **head -c <bytes>** (*para especificar los primeros bytes a mostrar)*

Visualizar archivos: tail, more

- **tail**
  - Muestra las últimas líneas de un fichero de texto. Como **head** pero por el final del fichero.
  - Mostrar las 4 últimas filas del fichero: **tail -4 fichero_texto**
  - **tail -n fichero** *(especifica las últimas líneas)*
  - **tail -c <bytes>** (*para especificar los últimos bytes a mostrar)*

- **more**
  - Muestra el contenido de un fichero de texto. Lo hace forma paginada, pulsando una tecla, va mostrando el resto del fichero: **more fichero_texto**
  - Avanza página a página con la barra del espacio.
  - Con Enter una sola línea.

  - **more -X** permite especificar el número de líneas que queremos que muestre por cada página. Por ejemplo, si queremos visualizar 6 líneas por página ejecutaríamos **more -6**.

  - **more +X** comienza la visualización del fichero en la línea X. Por ejemplo, si queremos ver un fichero a partir de su línea 13 pondríamos **more +13**.

  - *Si estamos visualizando con el comando more y pulsamos la (**h**)elp mostrará todas las opciones del comando more.*

Visualizar archivos: less

- **less**
  - Mas potente que more.
  - Permite **avanzar** y **retroceder** línea a línea con las **flechas de dirección**.
  - **Avanzar página a página** mediante **la barra espaciadora** y **retroceder** una página mediante la tecla **w**.
  - Al igual que con **more**, si visualizamos un fichero con **less** y presionamos la tecla **h** veremos una ayuda que nos indicará las teclas que podemos usar para desplazarnos por el documento.
  - Con varios ficheros de texto como argumentos, podemos movernos entre ellos pulsando **:n** para ir al siguiente documento o **:p** para ir al documento anterior.
  - Para salir pulsar la tecla **q**.
  - El **comando man** para mostrar las páginas del manual, utiliza **less**.

Visualizar archivos: less

- **less -e**
  - Hace que se salga automáticamente de la visualización cuando se llega **por segunda vez** al final del fichero. La primera vez que se llega al final del fichero se permanece en la visualización. Es muy útil para no tener que acordarse de salir con la **q.**

- **less -E**
  - Es muy similar a la anterior opción, pero hace que se salga de la visualización cuando se llega al final del fichero por primera vez.

- **less -F**
  - Hace que se salga de la visualización automáticamente si el fichero puede ser mostrado en una sola página.

- **less -N**
  - Muestra un número de línea delante de cada línea de texto que contiene el fichero.

- **less -w**
  - Al avanzar una página nos señala cual es la primera línea que no estaba en la página anterior. Sirve para saber por donde nos habíamos quedado leyendo cuando cambiamos a la siguiente página.

PRACTICAS VISUALIZACION

Operaciones con archivos

- **cp**
  - Copia un fichero.
  - **cp fichero_origen fichero_destino**
  - Se pueden utilizar comodines / rutas de directorios: . .. *
  - **cp -R /home/alumno/directorio1 /tmp**

  - copiará todo el contenido de la carpeta directorio1 en el destino indicado, que en este caso es /tmp.

  - También -r ←→-R (es lo mismo)

opción	descripción
-a	archive files
-f	copia forzada al remover el archivo destino si es necesario
-i	interactivo, pide confirmación del usuario antes de sobre escribir
-l	enlaces en vez de copia
-L	seguir enlaces simbólicos
-n	no sobre escribir archivos
-R	copia recursiva recursive copy (including hidden files)
-u	actualizar, copia cuando la fuente es mas reciente que el destino
-v	muestra mensajes informativos

Operaciones con archivos

- **ln**
  - Crea un **enlace duro** al fichero.
  - Para poner otro nombre al fichero. Los dos nombres son equivalentes. Es como un alias. Un fichero puede tener varios nombres o enlaces. Cuando se van borrando y este número llega a 0, el fichero original se elimina.
  - **ln fichero_origen alias**
  - Para crear **enlaces simbólicos** (en vez de duros) **utilizar la opción -s**
    - **$ ln -s fichero_origen enlace_fichero**

Ejemplo ln

- Partimos de un fichero prueba en directorio actual.
- Crear un **enlace duro**
  - **$ ln prueba enlace_duro** (*El enlace duro apunta al mismo inodo*)

- Crear enlace **simbólico**
  - $ **ln -s prueba enlace_simbo** *(El enlace simbólico apunta al nombre del archivo)*

  - Se pueden comprobar con el comando: **$ ls -l enlace***

```
antonio@antonio-VirtualBox:~$ ls -l enlace*
-rw-r--r-- 2 antonio antonio 5 dic 27 10:20 enlace_duro
lrwxrwxrwx 1 antonio antonio 6 ene 19 10:55 enlace_simbo -> prueba
```

Operaciones con archivos

- **unlink:**
  - Para borrar un enlace, también podemos utilizar la orden rm, pero es más indicado utilizar unlink.
  - La finalidad de esta orden es borrar un enlace existente.
    - En el caso en que dicho enlace sea el último que hace referencia al inodo, el fichero se borrará dejando el espacio disponible. En caso de que exista algún otro, sólo se borra el enlace.
  - Si, por ejemplo, deseamos borrar el enlace duro enldurofichero, ejecutaremos:
    - **$ unlink enldurofichero.**

Operaciones con archivos

- **mv**
  - Sirve para renombrar un fichero / directorio o moverlo a otra ubicación.
  - **mv nombre_origen nombre_destino**

  - **<u>Opciones del comando:</u>**
  - **-b** (backup) Si el fichero destino existe, crea una copia de seguridad.
  - **-f** Force, fuerza a sobrescribir archivos existentes (sin preguntar).
  - **-i** Interactivo para que pregunte antes de sobrescribir un fichero destino.
  - **-n** Para que NO sobrescriba archivos destino.
  - **-v** Proporciona una salida detallada (verbose).

Operaciones con archivos

- **rm**
  - Quita un nombre al fichero. En el caso de que el fichero tenga varios enlaces. Los otros nombres se mantienen. Se pueden utilizar comodines: *
  - **rm fichero**

  - **Opciones del comando**:
  - **-r, -R** Recursivo. Borrar en profundidad directorios y sus contenidos.
  - **-d, -dir** Para eliminar directorios vacíos. Es equivalente: rm -d y rmdir.
  - **-v** Modo detallado.
  - **-i** Interactivo. Confirmar antes de borrar.
  - **-I** (i mayúscula). Indicar una vez antes de eliminar más de tres archivos.

PRACTICAS FICHEROS-1 & BORRADOS & COPIA

Impresión de archivos: lp, lpq

- **lp**
  - Para imprimir un archivo.
  - Sintaxis: **lp documento**, ejemplo: lp fichero.pdf
  - lp **-P** 3,5,10-15 documento. Con el parámetro -P se pueden indicar las páginas a imprimir, separadas por comas y con el guión podemos indicar un rango.
  - lp **-d** impresora documento. Se puede indicar la impresora con el parámetro -d.
  - lp **-n** número_copias documento. Con -n se indica el número de copias que queremos imprimir.
  - Formatos permitidos: txt / pdf / ps

- **lpq**
  - Para consultar la cola de impresión. Ver que trabajos están pendientes.
  - **EPSONSCX1500 is ready and printing**
    Rank Owner Job File(s) Total Size
    active M@th 15 documento.txt 105472 bytes

  - Muestra información como:
    - **Rank:** indica cual es el estado del trabajo (active, inactive, waiting, etc...).
      Owner: indica el usuario que mandó a imprimir el trabajo.
      Job: indica el número de trabajo. (Sirve para hacer referencia a dicho trabajo, por ejemplo si queremos eliminarlo de la cola de impresión).
      File(s): indica el nombre del archivo a imprimir.
      Total Size: indica el peso del archivo a imprimir.

Impresión de archivos: lprm, lpstat

- **lprm**
  - Para eliminar trabajos de la cola de impresión.
  - **lprm número_trabajo**

- **lpstat**
  - Muestra información sobre las clases, trabajos e impresoras actuales. Sin argumentos muestra la lista de trabajos en la cola de impresión por el usuario actual.
  - lpstat forma parte de **CUPS** (a partir de Ubuntu 18 ya viene instalado).
  - Puede que haya que añadir el usuario al sistema de administración de impresión → **sudo adduser <nombre de usuario> lpadmin**
  - Podemos entrar en una página de administración: http://localhost:631

Buscar archivos: find

- **find**
  - Sirve para **buscar ficheros dentro de directorios**.
  - **Sintaxis: find <dir_inicial> <opciones> <término_búsqueda>**
  - Imprime todo el subárbol del directorio indicado. Si no se indica directorio, muestra todo lo que cuelgue del directorio actual. Comando similar: **du**

  - Búsqueda por nombre:
    - **find . -name "mi_fichero"** (no son necesarias las comillas)

  - Para no tener en cuenta las mayúsculas / minúsculas:
    - **find . -iname "mi_fichero"**

  - Se pueden utilizar comodines:
    - **find . -name "*.txt"**

Buscar archivos : find

- **find**
  - **find . not -name “mi_fichero”**
    - Archivos que no se llamen … mi_fichero.

  - Búsqueda por tipo de archivo:
    - Argumento -type

    - **f** – archivo normal
    - **d** – directorio o carpeta
    - **l** – enlace simbólico
    - **c** – dispositivos de caracteres
    - **b** – dispositivos de bloque

    - Ejemplo**: find / -type d** *Buscar todos los directorios a partir de la raíz*.
    - **find / -type d -name “mi_fichero”** *Combinando búsquedas*.

Buscar archivos : find (por días)

- **find**
  - Búsqueda por fecha:
    - **Tiempo de acceso (-atime)** – Fecha más reciente en que el archivo fue leído o escrito.
    - **Tiempo de modificación (-mtime)** – Fecha más reciente en que se modificó el archivo.
    - **Hora de cambio (-ctime)** – Fecha más reciente en que se actualizaron los metadatos del archivo.
  - Por ejemplo: Indicando el número de días que hace que se accedió:
    - **find / -atime 1**
  - Podemos hacer que nuestras consultas sean más precisas con los signos más (**+**) y menos (**–**) precediendo al número de días. Sin signo (+,-) es exactamente ese tiempo.
    - Por ejemplo: **find / -mtime +2**
  - Esto listará todos los archivos que tienen un tiempo de modificación de más de dos días.

Buscar archivos : find (por minutos)

- **-amin X**
  - Busca ficheros cuya fecha de último acceso sea de hace X minutos
  - +X hace mas de X minutos
  - -X menos de X minutos
  - X exactamente X minutos
  - Ejemplo, para ficheros que hayan sido accedidos en los últimos 5 minutos **find / -amin -5**.

- **-cmin X**
  - Busca ficheros cuya fecha de último cambio de estado sea de hace X minutos.
  - +X ficheros que hayan cambiado de estado hace más de X minutos.
  - -X ficheros que hayan cambiado de estado hace menos de X minutos.
  - X exactamente X minutos
  - Ejemplo, para ficheros que hayan cambiado de estado hace más de 30 minutos **find / -cmin +30**.

- **-mmin X**
  - Busca ficheros cuya fecha de última modificación sea de hace X minutos.
  - **+X** buscamos ficheros que hayan sido modificados hace más de X minutos.
  - -X Si buscamos son ficheros que hayan sido modificados hace menos de X minutos.
  - Si no ponemos ni **+** ni **-** buscamos ficheros que hayan sido modificados hace exactamente X minutos.
  - Ejemplo, para ficheros que hayan sido modificados hace exactamente 1 minuto **find / -mmin 1**.

Buscar archivos : find

- **-newer** se refiere a ficheros que hayan sido modificados más recientemente que un fichero determinado. Por ejemplo, si queremos saber que ficheros se han modificado más recientemente que /etc/sos.conf tendremos **find / -newer /etc/sos.conf**.

- **-ctime X**
  - Busca ficheros cuya fecha de último cambio de estado sea de hace X días.
  - +X buscamos ficheros que hayan cambiado de estado hace más de X días.
  - -X buscamos son ficheros que hayan cambiado de estado hace menos de X días.
  - X exactamente X días.
  - Ejemplo, para ficheros que hayan cambiado de estado hace más de 1 mes **find / -ctime +30**.

Buscar archivos : find

- **find**
  - Búsqueda por tamaño de los archivos
  - find <dir_inicial> -size <tamaño> <unidad>
    - **c** – bytes
    - **k** – kilobytes
    - **M** – megabytes
    - **G** – gigabytes
    - **b** – bloques de 512 bytes

    - **find / -size 10M** Archivos de 10 mb
    - **find / -size +5G** Más de 5 Gb.

  - Búsqueda por permisos:
    - **find / -perm 644 → permisos rw-xr--r--**

Buscar archivos : find

- **find**
  - **-maxdepth** sirve para indicarle la profundidad en subdirectorios a la que queremos buscar. Por ejemplo, **find / -maxdepth 2** buscará en el directorio raíz y sus subdirectorios directos, pero no más allá.
  - **-user** permite buscar ficheros que pertenecen a determinado usuario. Por ejemplo, si queremos ver los ficheros que pertenecen al usuario alumno debemos ejecutar **find / -user alumno**.
  - **-group** permite buscar ficheros que pertenecen a determinado grupo. Por ejemplo, si queremos buscar ficheros que pertenezcan al grupo root haremos **find / -group root**.

PRACTICAS BUSQUEDAS CON FIND

Permisos de archivos

- Permisos de archivos en Linux
- Gestión de permisos con chmod y umask
- Gestión de atributos de archivos con chgrp, chown y touch
- Seguridad de contraseñas

Permisos de archivos en Linux

- Linux es **multiusuario**.
- Cada usuario tiene “**una cuenta de usuario**”.
- Un usuario tiene que ser **miembro de un grupo** obligatoriamente.
- Existe la posibilidad de **compartir archivos** entre los distintos usuarios.
- Su grupo principal se utiliza al crear archivos.
- Puede pertenecer a **otros grupos** (son los grupos secundarios)
- Los grupos secundarios determinan sus derechos de acceso a los archivos creados por otros miembros de los grupos

Identificadores

- **UID**
  - User ID → Cada usuario tiene un identificador único.
  - Este número determina el propietario de un archivo en el sistema.
  - Los usuarios también tienen un nombre de usuario único (login)y de una contraseña (password). Para autenticarse en el sistema.

- **GID**
  - Group ID → Cada grupo tiene un identificador único.
  - El número también determina el grupo propietario de un archivo.

Comandos: id, groups

- Para consultar el UID y el GID de un usuario: comando **id**
- Muestra el uid, gid y a los grupos que pertenece un usuario.

- Para consultar la lista de grupos: **groups**
- Se puede pasar como parámetro el nombre de un usuario.

Jerarquías de usuarios

- Todas las cuentas NO son iguales.

- Hay 3 tipos de cuentas:
  - **root**
    - El super usuario
    - El usuario más importante del sistema.
    - No le afectan los permisos de los archivos.

  - **bin, daemon, sync, apache** ...
    - Cuentas que no se asignan a personas físicas.
    - Facilitan la administración de los derechos de acceso.
    - Estas cuentas utilizan los UID comprendidos entre 1 y 999

  - **Los usuarios**
    - Para personas reales.
    - El UID >= 1000

Gestión de permisos

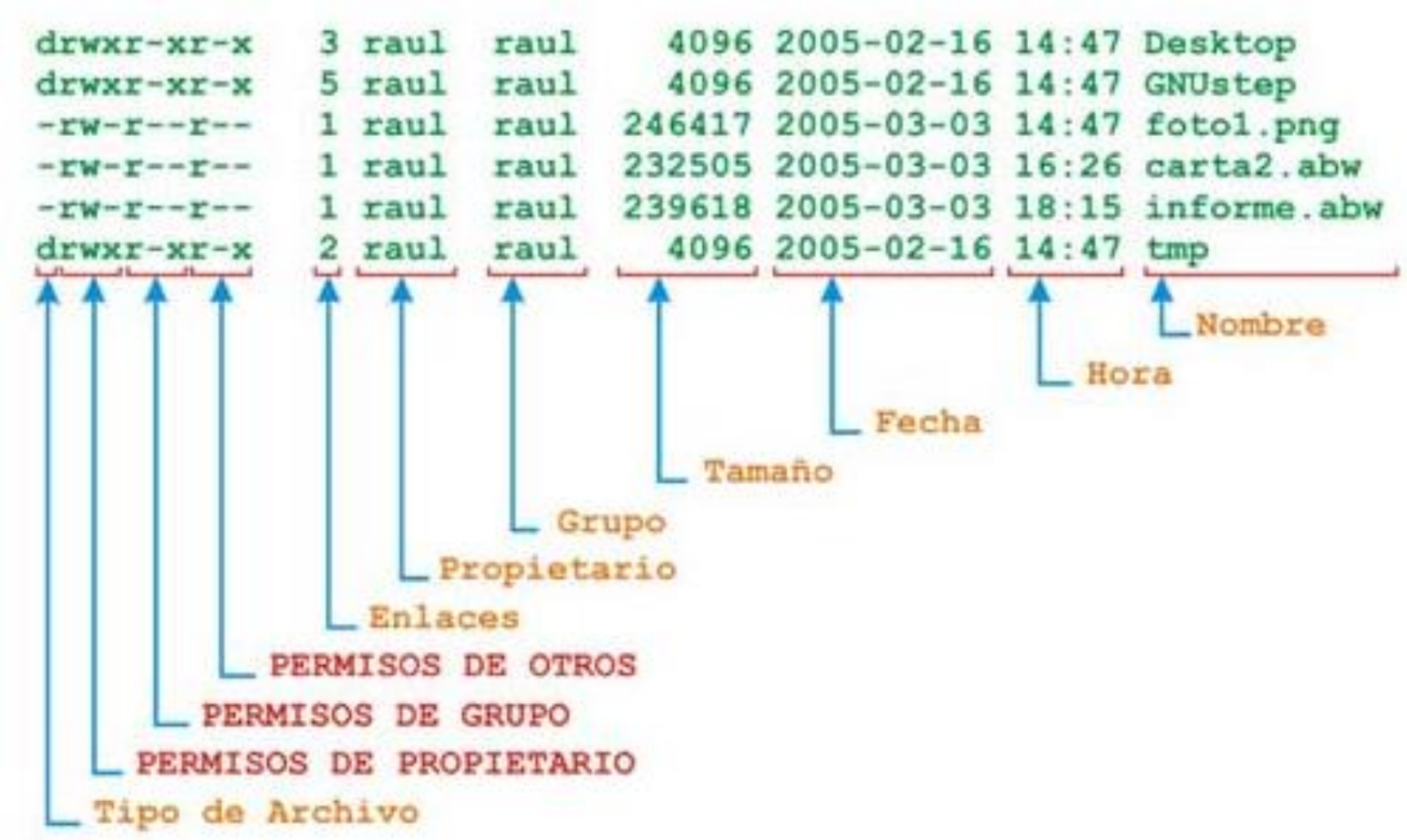
- Los permisos de un archivo se definen por:
  - Para cambiar los permisos de un fichero. Se indican con grupos de 3. Ver comando **ls -l**
  - **rwx** Los 3 primeros para el dueño.
  - **rwx** Usuarios del mismo grupo.
  - **rwx** Otros usuarios.
  - Delante de estos 3 grupos puede haber una d o _ para indicar si la entrada es un directorio o no.
  - Los 3 números se interpretan en binario:
  - 110 100 100 → indica: rw-r--r--

read	Lectura
write	Escritura
execute	Ejecución

Usuario Grupo

```
drwxr-xr-x 2 antonio antonio 4096 jul 11  2019 Descargas
drwxr-xr-x 2 antonio antonio 4096 jul 11  2019 Documentos
drwxrwxr-x 4 antonio antonio 4096 nov  1 18:02 eclipse-workspace
-rw-r--r-- 1 antonio antonio  171 dic 27 10:46 ejemplo
-rw-r--r-- 1 antonio antonio  247 dic 27 10:33 errores
```

Gestión de permisos



Gestión de permisos

- El permiso de acceso a un archivo no se define únicamente por los permisos otorgados al archivo y al directorio que lo contiene.

- Hay que tener permiso por todos los directorios intermedios para poder acceder al archivo interno.

- Por ejemplo para un archivo situado en la ruta: /dir1/dir2/dir3/arch
- Hay que tener permiso lectura **r** en el archivo **arch** pero también los permisos r y x en los tres directorios: **dir1**, **dir2** y **dir3**

Gestión de permisos

- **chmod**
  - Se utiliza para cambiar los permisos de un archivo.
  - Sintaxis: chmod [-R] <permisos> <archivo …>
  - Ejemplo:
  - chmod 644 fichero1 → **Notación Octal**
  - 644 → 110 100 100 → **rw**x **r**wx **r**wx
  - **-R** indica recursivo

  - ***Los permisos se puede indicar de forma octal o de forma simbólica.***

Gestión de permisos

- **Notación simbólica**:
  - Lectura: **r**
  - Escritura: **w**
  - Ejecución: **x**
  - Usuario: **u**
  - Grupo: **g**
  - Otros: **o**

  - Se utilizan también los caracteres: **+ - =**
  - La letra '**a**' equivale a: **ugo**
  - *(son las 3 entidades)*

- Sintaxis general:

  <entidad(es)>[+-=]<permiso(s)>

- **Ejemplos**:
  - **u+x** Agrega permiso exec para el propietario.
  - **g-w** Quitar permiso escritura al grupo.
  - **a+r** Añadir permiso de lectura a las 3 entidades.

Ejemplo (I)

- Asignar permisos a carpetas:
- Creamos las carpetas: **$ mkdir carpeta1 carpeta2**
- Permisos iniciales:

```
antonio@antonio-VirtualBox:~/pruebas$ ls -dl carpeta?
drwxr-xr-x 2 antonio antonio 4096 ene 19 11:34 carpeta1
drwxr-xr-x 2 antonio antonio 4096 ene 19 11:34 carpeta2
```

- Cambiar permisos:
  - $ chmod u=**rwx**, g=, o= carpeta1
  - $ chmod u=**rwx**, g=**rx**, o= carpeta2

```
drwx------ 2 antonio antonio 4096 ene 19 11:34 carpeta1
drwxr-x--- 2 antonio antonio 4096 ene 19 11:34 carpeta2
```

Ejemplo II

- Creamos dos ficheros (vacíos) dentro de carpeta1:
- $ touch carpeta1/{f1,f2}
- $ chmod = carpeta1/{f1,f2} → Quitamos todos los permisos.

```
antonio@antonio-VirtualBox:~/pruebas$ touch carpeta1/{f1,f2}
antonio@antonio-VirtualBox:~/pruebas$ ls -l carpeta1
total 0
-rw-r--r-- 1 antonio antonio 0 ene 19 11:42 f1
-rw-r--r-- 1 antonio antonio 0 ene 19 11:42 f2
antonio@antonio-VirtualBox:~/pruebas$ chmod = carpeta1/{f1,f2}
antonio@antonio-VirtualBox:~/pruebas$ ls -l carpeta1
total 0
---------- 1 antonio antonio 0 ene 19 11:42 f1
---------- 1 antonio antonio 0 ene 19 11:42 f2
```

Gestión de permisos

- **umask**
  - La máscara del usuario (para permisos).
  - Cuando se crean nuevos archivos y directorios se crean por defecto con unos permisos, que son distintos según sea un fichero o un directorio.
    - En el caso de los ficheros, se crean con los permisos: **-rw-r--r--**
    - En el caso de los directorios, **drwxr-xr-x**
  - Esto se establece a partir del comando umask
  - Por defecto, umask , muestra **0022**
  - Se puede cambiar la máscara con el comando umask.
    - umask 002 → pasa a ser 002

Gestión de permisos

- Si la máscara de usuario cambia, solo será en la sesión actual.
- Cuando arranque el sistema, volverá a estar al valor inicial.
- Esto tiene que ver con los ficheros de arranque del sistema hay que indicarlo en los ficheros: **/etc/profile** o **/etc/bash.bashrc**

Gestión de atributos

- **umask funcionamiento**.
  - Los permisos base para un fichero son 666 → 110 110 110
  - La operación que hace umask es:
  - 666 - 022 → **644** (**rw-r--r--**)
  - Los permisos base para un directorio son 777 → 111 111 111
  - 777 - 022 → **755** (**drwxr-xr-x**)

Gestión de atributos

- **umask funcionamiento (ejemplo, para los ficheros)**
- Para realizar la diferencia se realiza la operación:
- 666 AND NOT umask

- umask = 022 → 000 010 010
- Not umask = 755 → 111 101 101

- 666 → 110 110 110
- 755 → 111 101 101 AND
- ----------

110 100 100 → **rwxr--r--** ***Para los archivos***

Gestión de atributos

- **chgrp**
  - Sintaxis: **chgrp [-R] <grupo> <archivo …>**

  - **chgrp** forma parte del paquete **coreutils**, el cual se instala de forma predeterminada en todas las distribuciones de GNU/Linux debido a que se trata componente esencial.

  - ***Se utiliza para cambiar el grupo al cual pertenece un archivo o directorio***. Puede especificarse tanto el nombre de un grupo, así como un número de identidad de grupo (**GID**).

Gestión de atributos:

- **chown**:
  - **chown [-R] usuario[:grupo] archivo(s) o directorio(s)**

  - Se utiliza para **cambiar el propietario** al cual pertenece un **archivo** o **directorio**.
  - Puede especificarse tanto el nombre de un usuario, así como un número de identidad de usuario (**UID**).
  - De manera opcional, utilizando un signo de dos puntos (**:**) o bien un punto (**.**), permite especificar también un nombre de grupo.

  - Ejemplo:
    - chown otro_usuario fichero
    - chown otro_usuario:otro_grupo fichero

PRACTICAS SOBRE PERMISOS

Gestión de atributos

- **touch**
  - El comando **touch** de Linux se usa principalmente para manipular el tiempo de acceso y modificación de los archivos.

- Los archivos y directorios tienen una marca de tiempo:
  - Tiempo de **acceso** (**atime**): la última vez que se leyó un archivo.
  - Tiempo de **modificación** (**mtime**): la última vez que se modificó el contenido de un archivo. Al igual que el tiempo de acceso, también forma parte de los metadatos del estado de los archivos.
  - Tiempo de cambios realizados (**ctime**): la última vez que se **modificaron** los metadatos de un archivo (por ejemplo, permisos)

Gestión de atributos

- **touch**
  - **touch [opciones] archivo**

- El comando touch sin ninguna opción crea un nuevo archivo. Si el archivo existe, el comando actualizará el tiempo de acceso y de modificación a la hora actual sin cambiar su contenido:
  - **touch nombre_archivo.txt**

- También es posible modificar varios archivos en el mismo comando touch

Gestión de atributos

- **touch**
  - Dispone de varias opciones:
    - **-a** cambiar el tiempo de **acceso** al archivo.
    - **-m** cambiar el tiempo de **modificación** de un archivo.
    - **-c** cambiar el tiempo de acceso y modificación a un archivo.
  - Indicar un tiempo específico.
    - touch **-t 201903081047.30** nombre_archivo.txt

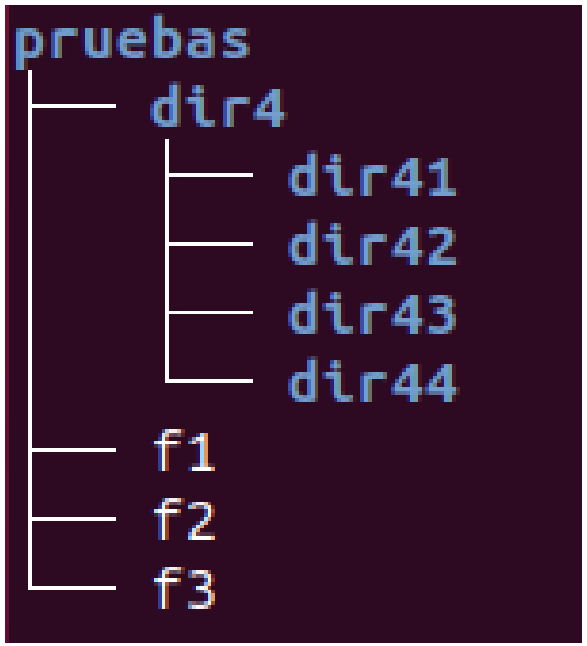
- El formato de fecha y hora debe estar en **CCYYMMDDhhmm.ss** donde:
  - CC: los dos primeros dígitos del año
  - YY: los dos segundos dígitos del año
  - MM: El mes del año [01-12]
  - DD: El día del mes [01-31]
  - hh: La hora del día [00-23]
  - mm: El minuto de la hora [00-59]
  - ss: El segundo del minuto [00-59]

La fecha / hora también se puede especificar:
touch **-d '8 Mar'** nombre_archivo.txt
touch **-d '20:10** ‘ nombre_archivo.txt

Gestión de atributos

- Si al comando **touch** le pasamos el nombre de un archivo que no existe lo crea vacío.

- Podemos utilizarlo para crear varios ficheros vacíos:
- Ejemplo: crear los ficheros f1,f2,f3 a partir del directorio “pruebas”
- $ **touch pruebas/{f1,f2,f3}**



Seguridad de contraseñas

- Las contraseñas se almacenan en el fichero: **/etc/shadow**

- Los usuarios disponen del comando **passwd** para cambiar la contraseña.
  - Solicita la contraseña actual
  - Y luego 2 veces la nueva.

Redirección entrada y salida

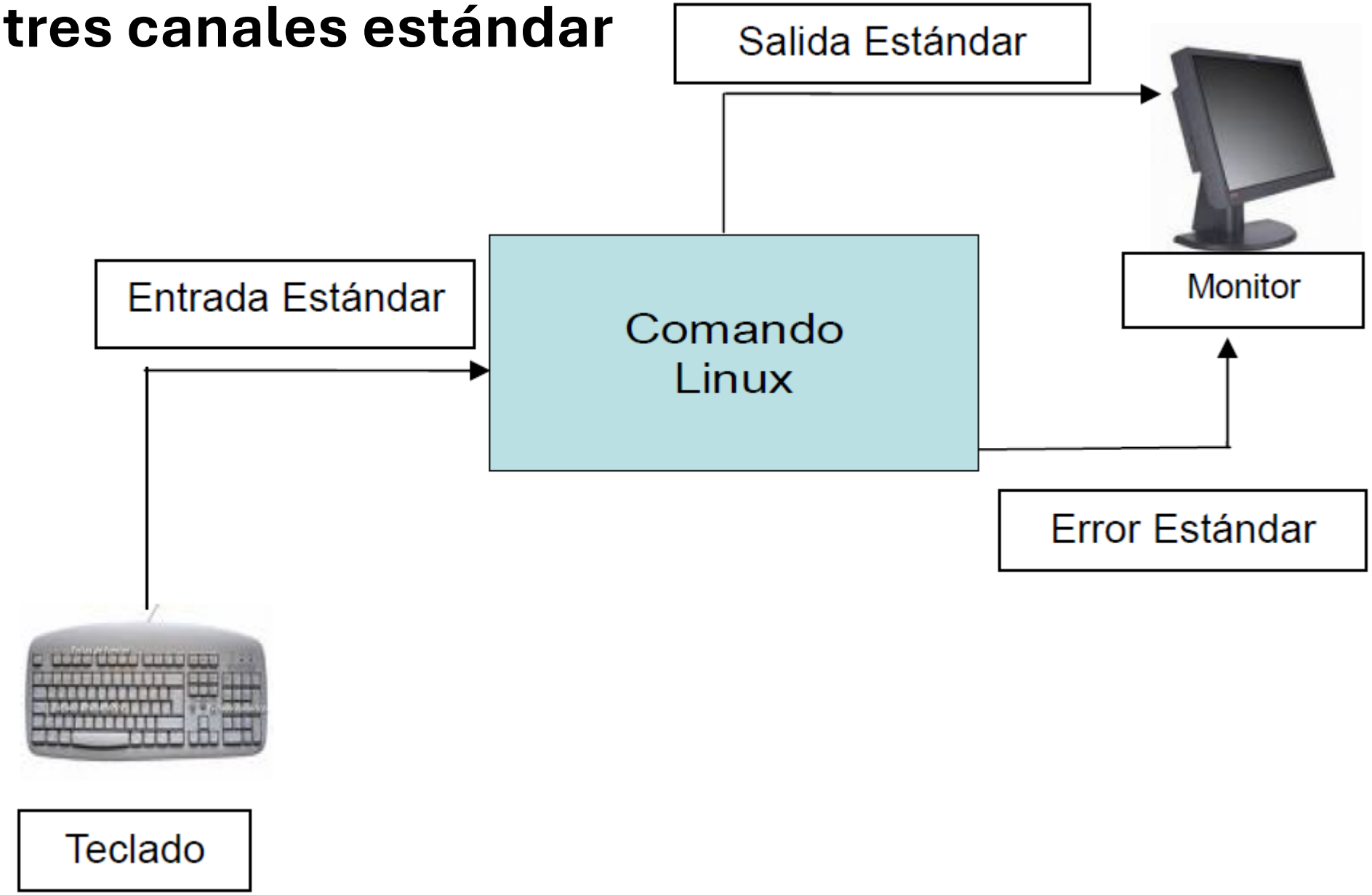
- Redirección de la entrada estándar, la salida estándar y el error estándar.
- Usar sort, wc y filtros grep

Redirecciones

- Todo proceso en Linux se asocia a 3 descriptores de archivos que permiten administrar: entrada estándar, salida estándar y errores estándar.
- Los descriptores de archivo son:
  - 0 Entradas (asociado al teclado)
  - 1 Salidas (terminal)
  - 2 Errores (terminal)

Name	Abbreviation	File Descriptor	Standard Device
Standard Input	*stdin*	0	Keyboard
Standard Output	*stdout*	1	Console
Standard Error	*stderr*	2	Console

Los tres canales estándar



Redirecciones

- Para redirigir la **entrada estándar** de un proceso que no trate los datos desde el teclado se utiliza el símbolo **<** seguido del nombre de un fichero.

- Si tecleamos un comando, por ejemplo: wc (sin indicar fichero) el comando tomará como datos lo que tecleemos hasta el EOF (al teclear **control + d**).

- La **salida estándar** será con el símbolo **>** seguido del nombre de un fichero.
  - El archivo si no existe se crea.
  - Si existe se sobrescribe el contenido.
  - Para añadir al final del archivo se utiliza **>>** si no existe también se crea.

Redirecciones

- Para redirigir la salida estándar de errores se utiliza **2>**. El 2 representa el descriptor 2 (salida errores).

- Es conveniente anular la presentación de mensajes de errores conocidos. Por ejemplo, el comando ls -R / recorre todo el árbol de directorios y aparecen mensajes de error cuando el usuario no tiene permisos para acceder a ciertos directorios.

- **ls -R / 2>/dev/null**
  - Con esta redirección los errores se destruyen automáticamente.

Redirecciones

- En el caso de la salida estándar de errores también se puede extender el fichero si ya existía.

- Utilizar: comando **2>>** nombre_fichero

- Las redirecciones se pueden combinar en el mismo comando:
  - comando **<** archivo_in **>** archivo_out **2>** archivo_error

Comandos

- **sort**
  - Ordena el contenido de un fichero. Ordena las líneas del fichero por orden alfabético.
  - sort fichero1
  - La sintaxis general:
    - **sort -t delimitador -k campo.carácter opciones archivo**
    - Se puede añadir -r para invertir la ordenación.
    - **sort -r** ordena los resultados en orden inverso
    - **sort -f** ordena sin tener en cuenta las mayúsculas / minúsculas
  - La clasificación puede empezar a partir de cualquier carácter de cualquier campo en el archivo.
  - Se utiliza la sintaxis **-kn.m** n es el número de campo y m el número del carácter en el campo (si se omite m, se toma el primer carácter).

Comandos

- Ejemplos comando **sort**

  - Partiendo del fichero: frutas
    - **m**anzana.2
    - **t**omate.10
    - **p**era.4

  - sort -t. -k2 frutas, primer carácter del segundo campo de cada línea.
    - tomate.**1**0
    - manzana.**2**
    - pera.**4**

  - sort -t . -n -k2 frutas (-n tiene en cuenta el valor numérico del campo).
    - manzana.**2**
    - pera.**4**
    - tomate.**10**

Comandos

- **wc**
  - Se utiliza para hacer recuentos en un fichero, si no se indican parámetros devuelve toda la información: número de líneas ´, de palabras, de caracteres.
    - wc **-l** archivo Número de líneas.
    - wc **-w** archivo Número de palabras.
    - wc **-c** archivo Número de caracteres.
- **grep**
  - Para buscar texto dentro del contenido de uno o varios ficheros.
  - Mostrar las líneas de todos los ficheros cpp del directorio actual que contienen la palabra include:
    - **grep include *.cpp**

Expresiones regulares para grep

- El comando grep utiliza expresiones regulares para filtrar la información:
  - Los principales meta-caracteres:

.	Cualquier carácter una vez (equivale al carácter genérico ?)
*****	Cero instancias o más del carácter anterior.
[a-n]	Un carácter de la lista.
^	Principio de la línea.
$	Final de la línea
****	Inhibición del meta-carácter siguiente.

Opciones de grep

- **-número** Las líneas concordantes se mostrarán acompañadas del número especificado de líneas anteriores y posteriores.
- **-A X** Muestra X líneas de contexto después de las que concuerden con el patrón.
- **-B X** Muestra X líneas de contexto antes de las que concuerden con el patrón.
- -**c** En lugar de mostrar el contenido de las líneas que conciden con el patrón de búsqueda muestra el número de líneas que concuerdan con el patrón para cada fichero de entrada.
- **-h** Suprime la impresión de los nombres de ficheros antes de las líneas concordantes en la salida, cuando se busca en varios ficheros.
- **-i** No hace caso de si las letras son mayúsculas o minúsculas ni en el patrón ni en los ficheros de entrada.
- **-L** Suprime la salida normal. En su lugar muestra el nombre de cada fichero de entrada donde no se encuentre ninguna concordancia y por lo tanto de cada fichero que no produciría ninguna salida. La búsqueda se detendrá al llegar a la primera concordancia.
- **-l** Suprime la salida normal. En su lugar muestra el nombre de cada fichero de entrada que produciría alguna salida. La búsqueda se detendrá en la primera concordancia.
- **-n** Prefija cada línea de salida con el número de línea de su fichero de entrada correspondiente.
- **-q** Modo silencioso; suprime la salida normal. La búsqueda finaliza en la primera concordancia.
- **-v** Invierte el sentido de la concordancia, para seleccionar las líneas donde no hay coincidencias con el patrón.
- **-w** Selecciona solamente aquellas líneas que contienen concordancias que forman palabras completas.
- **-x** Selecciona solamente aquellas concordancias que constan de toda la línea.

Tuberías

- Conceptos sobre Shell pipeline
- Usar tee, cut, tr, more y pr

- Los pipes permiten enlazar comandos:

Shell pipeline

- Mediante los **pipes** **|** vamos a poder enlazar comandos. La salida de un comando se pasa con entrada al siguiente comando y así sucesivamente.

- No es necesario utilizar redirecciones para grabar en ficheros los resultados intermedios.

- Por ejemplo, contar el número de archivos:
  - **ls | wc -l**
  - Lista los ficheros del directorio actual y el resultado se le pasa al comando wc que lo recibe como entrada. Con el parámetro -l cuenta líneas, es decir, ficheros.

Shell pipeline

- Se pueden enlazar varios pipes, pero solo se muestra el resultado del último comando.

- Si quisiéramos ver la salida de un comando intermedio se puede almacenar en un fichero.
  - **ls | tee ejemplo | wc -l**

- Debido a que se pueden crear comandos realmente largos, Linux permite la definición de alias para comandos que utilizamos repetidamente.

Comandos

- **tee**
  - Tiene que ver con los pipes.
  - Si queremos redirigir la salida de un comando como entrada de otro, pero además queremos ver la salida de ese comando, podemos utilizar el comando **tee**.
  - Por ejemplo: guardar la salida del comando ls en un fichero pero además queremos ver la salida del comando ls (para este cometido se utiliza tee).
  - **ls -l | tee fichero.txt**

  - Es útil, si estamos trabajando con MySQL y a la vez que queremos ver el resultado de la consulta además queremos guardar los resultados en un archivo.
    - mysql -u root -p -e "USE gestion; SELECT * FROM tbl_articulos;" | **tee** datos.txt
    - mysql -u root -p -e "USE gestion; SELECT * FROM tbl_presupuestos;" | **tee --append** datos.txt

  - Con la opción **--append** añade al final del archivo sin borrar los datos anteriores.

Comandos

- **cut**
  - Permite extraer columnas o campos seleccionados a partir de su entrada estándar o de archivos.
  - cut -f campo(s) -d delimitador archivos(s)
  - Ejemplo: Extraer columnas del fichero /etc/passwd
  - cut -f 1-4 -d : /etc/passwd
    - Columnas de la 1 a la 4.
    - El separador :
  - Para columnas separadas: Utilizar la coma
  - cut -f 1,3,6 -d : /etc/passwd

Comandos

- **tr**
  - Cambia caracteres de un fichero.
  - tr a x
  - Cambia la aes por x del fichero1. Ver **tr --help**
  - cat fichero1 | tr a x

- **more**
  - El comando more permite mostrar el resultado de la ejecución de un comando en la terminal de a una página a la vez.
  - Esto es especialmente útil cuando se ejecuta un comando que causa un gran desplazamiento, como el **comando ls** o el **comando du**. Pagina el resultado.
  - Ejemplos:
    - ls -l | more
    - ps | more

Comandos

- **pr**
  - Para ver las páginas que nos va a ocupar un documento antes de imprimirlo.
  - Prepara el fichero para imprimir. Por defecto, son 66 líneas y le formatea las cabeceras.
  - pr fichero1 fichero2 … ficheroN

- Opciones:
  - **pr -d** pone el documento a doble espacio.
  - **pr -l X** especifica que la página tendrá X líneas. Por ejemplo, para poner páginas de 30 líneas sería **pr -l 30**.
  - **pr -h "texto"** permite poner en la cabecera de cada página el texto que le pongamos entre comillas.
  - **pr -n** imprime también el número de línea asignado a cada línea del texto.

PRACTICAS IMPRESIÓN

Comandos

- **uniq**
  - Quitar filas repetidas, deben estar ordenadas previamente
  - Ideal para combinar en un pipe con cut, cat | sort | uniq

PRACTICAS FICHEROS CSV (PIPES)

Creación de scripts

- Creación y ejecución de scripts
- Variables y estructuras de control
- Expresiones y funciones

Introducción

- Linux permite escribir script para automatizar tareas.
- La Shell tiene un lenguaje de programación con operadores, bucles, condiciones, se pueden pasar parámetros, definir y operar con variables.
- Los ficheros o scripts tienen extensión **sh**.
- Los scripts se pueden ejecutar con la Shell de bash: **bash prog.sh**
- **También con sh prog.sh**
- Si tiene los permisos de ejecución se puede llamar: **./prog.sh**

Ficheros sh

- No nombrar a los scripts con la palabra script o test puede confundir con los comandos test.

- Para que el script lo ejecute la Shell de bash en la 1ª línea del fichero se escribe: **#!/bin/bash**

- Podemos comprobar cuál es la Shell que nos responderá por defecto: imprimir en el **prompt**:
  - **echo $SHELL**

Comentarios

- Dentro de los ficheros de script, tenemos

```
# Comentario de una línea
```

- Comentario en bloque : ’ … ‘ *(entre los : y la ‘ va un espacio)*

```
: ’
    Líneas de comentario
    O comentario en bloque
‘
```

Más opciones de echo

- Por defecto, **echo salta de línea** cuando se imprime la cadena por consola.
- Con la opción **-n** → **echo -n “mensaje”** NO salta de línea.

- Con la opción -e sustituye los caracteres especiales por el carácter correspondiente. \n Salto de línea, \t tabulador …
- **echo -e** “**\t**un tabulador y un salto de línea**\n**”

Códigos de retorno

- Todos los comandos de Linux devuelven un código al terminar su ejecución.

- Devuelven **0** cuando el programa **ha terminado correctamente** y **distinto de cero en caso contrario** (entre 0 y 255).

- La variable especial **$?** Contiene el **código de retorno** del último comando ejecutado.

- Se puede mostrar con **echo $?**

Códigos de retorno

- Podemos hacer que nuestro script devuelva un valor hacia fuera mediante el comando exit.

programa1.sh

```
#!/bin/bash
pwd
exit 2
```

Ejecución del script:

```
./programa1.sh
/home/antonio
```

echo $? → 2

Ejecución secuencial

- En lugar de escribir los comandos uno detrás de otro, se pueden encadenar separándolos con un ;

- Sintaxis:

  Comando1 **;** comando2 **;** …

- Ejemplo:

~$ **date ; ps**

Ejecución condicional

- Se ejecuta el comando2 sólo si el comando1 se ha ejecutado correctamente.

- Para ello se separan los dos comandos con **&&**
  - Sintaxis: **comando1 && comando2**
  - pwd **&&** ls fichero_inexistente
  - Si pwd va bien, ejecuta ls

- Si queremos que el comando2 se ejecute sólo si el primer comando va mal, se separan los comandos con **||**
  - Sintaxis: **comando1 || comando2**
  - ls fichero_inexistente **||** pwd
  - Si falla ls ejecuta pwd.

Variables especiales

- **$$**
  - Contiene el PID de la Shell
- **$PPID**
  - Contiene el PID del proceso padre de la Shell.
- **$0**
  - Contiene el nombre del script que se está ejecutando.
- **$1,$2,$3 ...**
  - Representan parámetros posicionales de la Shell

Variables especiales

- Comando **Shift**
  - shift n
    - Permite desplazar los parámetros de la Shell n veces
    - El parámetro 1 → $1, 2 → $2, 3 → $3 …
- **$#**
  - El número de parámetros posicionales de la Shell
- **$*, $@**
  - Ambas variables contienen el conjunto de parámetros posicionales pasados por la línea de comandos.
- **$LINENO**
  - Se utiliza para depurar. Es el número de línea del script del Shell donde aparece.

Comando test

- Este comando permite efectuar una serie de pruebas sobre los archivos, las cadenas de caracteres, los valores aritméticos y el entorno de usuarios.

- Dos posibles sintaxis:
  - **test expresión**
    - **if test $a –lt $b then**
  - **[expresión]**
    - **if [$a –lt $b] then**

- True → 0
- False → 1

Test de archivos

- **-f archivo**
  - Devuelve true si el archivo es de tipo estándar
  - test -f /etc/passwd → true
- **-d archivo**
  - Devuelve true si es de tipo directorio
  - test -d /
- **-r archivo**
  - Devuelve true si el archivo tiene permiso de lectura
- **-w archivo**
  - Devuelve true si el archivo tiene permiso de escritura
- **-x archivo**
  - Devuelve true si el archivo tiene permiso de ejecución

Test de archivos

- **-e archivo**
  - True si el archivo existe
- **-s archivo**
  - True si el archivo no está vacío
- **archivo1 -nt archivo2**
  - Devuelve true si archivo1 es más reciente archivo2. (nt newer than)
- **archivo1 -et archivo2**
  - Devuelve si el archivo1 es idéntico a archivo2 (equal file). Los dos archivos poseen el mismo inodo.

Test de cadenas de caracteres

- **-n cadena**
  - Devuelve true si la cadena no es nula.
- **-z cadena**
  - Devuelve true si la cadena es nula
- **cadena1 = cadena2**
  - Devuelve true si las dos cadenas son iguales
- **cadena1 != cadena2**
  - Devuelve true si las dos cadenas son distintas

Test aritmético

- **valor1 -eq valor2**
  - Devuelve true si los dos valores son iguales
- **valor1 -ne valor2**
  - Devuelve true si los dos valores son distintos
- **valor1 -le valor2**
  - Devuelve true si valor1 <= valor2
- **valor1 -lt valor2**
  - Devuelve true si valor1 < valor2
- **valor1 -gt valor2**
  - Devuelve true valor1 > valor2
- **valor1 -ge valor2**
  - Devuelve true si valor1 >= valor2

Test del entorno de usuario

- Opción **-o**
- **set -o**
  - test -o vi ; echo $?
  - Devuelve 1 → false

  - test -o emacs ; echo $?
  - Devuelve 0 → true

```
antonio@antonio-VirtualBox:~$ set -o
allexport       off
braceexpand     on
emacs           on
errexit         off
errtrace        off
functrace       off
hashall         on
histexpand      on
history         on
ignoreeof       off
interactive-comments    on
keyword         off
monitor         on
noclobber       off
noexec          off
noglob          off
nolog           off
notify          off
nounset         off
onecmd          off
physical        off
pipefail        off
posix           off
privileged      off
verbose         off
vi              off
xtrace          off
```

Combinación de expresiones

- **!** Expresión
  - Negación si es true → false y viceversa.
- Expresion1 **-a** expresion2
  - Representa **AND** lógico
- Expresion1 **-o** expresion2
  - Representa **OR** lógico
- **\(**expresión**\)**
  - Se pueden agrupar varias expresiones.

Operaciones aritméticas

- Disponemos principalmente de los comandos: **expr**, **let**

- **expr**
  - Podemos ver la versión y las opciones de la siguiente forma.
  - **expr -- version** y **expr --help**

- Ojo al indicar las expresiones dejar espacios en blanco
  ~$ expr 3 + 7 → 10
  ~$ expr 3+7 → 3+7

```
ARG1 | ARG2       ARG1 si no es nulo ni 0, de otra manera ARG2

ARG1 & ARG2       ARG1 si ningún argumento es nulo o 0, de otra manera 0

ARG1 < ARG2       ARG1 es menor que ARG2
ARG1 <= ARG2      ARG1 es menor o igual que ARG2
ARG1 = ARG2       ARG1 es igual a ARG2
ARG1 != ARG2      ARG1 es distinto de ARG2
ARG1 >= ARG2      ARG1 es mayor o igual que ARG2
ARG1 > ARG2       ARG1 es mayor que ARG2

ARG1 + ARG2       suma aritmética de ARG1 y ARG2
ARG1 - ARG2       diferencia aritmética de  ARG1 y ARG2

ARG1 * ARG2       producto aritmético de ARG1 y ARG2
ARG1 / ARG2       cociente aritmético de ARG1 dividido entre ARG2
ARG1 % ARG2       residuo aritmético de ARG1 dividido entre ARG2

CADENA : EXPREG   búsqueda de expresiones regulares REGEXP en CADENA

match CADENA EXPREG        igual que CADENA : EXPREG
substr CADENA POS LONG     subcadena de CADENA, POS se cuenta partiendo de 1
index CADENA CARacteres    índice en CADENA donde cualquier CARácter es
                           encontrado, ó 0
length CADENA              longitud de CADENA
+ TOKEN                    interpreta TOKEN como una cadena, incluso si es
                             una palabra clave como 'match' o un operador
                             como '/'

( EXPRESIÓN )              valor de EXPRESIÓN
```

Operaciones matemáticas: expr

- \| or lógico
- \& and lógico
- \< menor
- \<= menor igual
- \> mayor
- \>= mayor igual
- = igual
- != distinto

expr devuelve 0 con falso, 1 true
Es coherente con el álgebra de Boole
En la consola en cambio es al revés.

Expresiones aritméticas: let, (())

- Está integrado en bash y permite realizar operaciones como el comando **expr**.

- Ventajas de **let** frente a e**x**pr:
  - Más rápido porque está integrado en la Shell.
  - Operadores utilizados en matemáticas y otros lenguajes de programación.
  - No hay meter espacios en las expresiones, ni poner una \ delante de los símbolos: *, < y >
  - Los nombres de las vars. No se preceden de $.
  - Hay más operadores.

- Se puede llamar: **let expresión** o **((expresión))**

Operadores

- Ejemplo: **let var=9**2 ; echo $var** → **El ; une sentencias**

- Aritméticos:
  +, -, *, /, %, **, =
  - Cuando copiamos una variable, el origen necesita el $ → **var1=$var2** y sin espacios

- Lógicos:
  &&, ||, !,

- Relacionales:
  <, <=, >, >=, ==, !=

- Otros operadores (a nivel de bits):
  - ~, >>, << , &, |, ^

- Incrementos / Decrementos:
  - ++var, var++, --var, var--

let dentro del script

- #!/bin/bash
- # Se pueden indicar las expresiones con espacios en blanco
- ((suma=6+7))

- Se pueden imprimir las variables y expandirlas dentro de una cadena.
- echo $suma
- echo “La suma es: $suma”

Comando read

- El comando **read** interrumpe la ejecución de la Shell hasta que el usuario introduzca una cadena de caracteres (aunque sea vacía) en su entrada estándar.

- **Ejemplo**:
  - $ read a b c
  - 1 2 3
  - echo $a ; echo $b ; echo $c

- Si llamamos a **read** sin parámetros, la respuesta del usuario se asigna a la variable de entorno $**REPLY**

Operaciones con String

- Obtener una subcadena.
- #!/bin/bash
  Str="Learn Linux from LinuxHint"
  subStr=${Str:6:5}
  echo $subStr

- El 6 indica a partir de donde y 5 la longitud de la subcadena.

Creación de funciones

- #!/bin/bash
  F1()
  {
  echo ’Dentro de la function F1’
  }

Llamada a la función F1
F1

- Si la función tiene parámetros no es necesario declararlos se recibe con $1, $2, etc.
- Se llama a la función sin parámetros

- #!/bin/bash

- Rectangle_Area() {
- area=$(($1 * $2))
- echo "Area is : $area"
- }

- Rectangle_Area 10 20

Creación de funciones

- Recoger el valor de retorno de una función.

- ```
  #!/bin/bash
  greeting() {

  str="Hello, $name"
  echo $str

  }

  echo "Enter your name"
  read name

  val=$(greeting)
  echo "Return value of the function is $val"
  ```
  ```

Ejemplo

```
#!/bin/bash

# Ejemplo de paso de parámetros y recogida del valor retornado.

suma(){
        r=$(($1+$2))
        echo $r
}

echo "El resultado de la suma"
i=9
j=10
resul=$(suma $i $j)
echo "Suma: $resul"
```

Estructuras de control

- Permiten ejecutar uno o más comandos en función del resultado de una expresión.
  - Condicional → **if / case**
  - Bucles → **while** / **for**
  - En los bucles se puede utilizar **break** para salir del bucle.

- Principalmente se utilizan los comandos **test** o **let** como condiciones.

Instrucción if

if condición
then
    Comandos si la condición es true
else
    Comandos si la condición es false
fi

- *Al lado de **then** podemos tener un **break**, no hace falta romper la línea*

```
if ((i==5))
then break
fi
```

- *Las expresiones del **if** van entre **(())** las variables no necesitan el **$***

- **Ejemplo**:
  - cat programa.sh
  - #!/bin/bash
  - if [“$1” = “vale”]
  - then
    - echo “está bien”
  - else
    - echo “no está ok”
  - fi

Ojo, con espacios en Blanco entre los [ ] y la Expr.

- ***Las expresiones del if si referencian a variables llevan el $***
- **La llamada: ./programa.sh vale**

Instrucción if

- Si utilizamos operador lógico AND (&&) la expresión se coloca entre doble corchete.
- **if [[** ($cadena1=="hola" && $cadena2=="adios“) **]]**

- Para anidar varios if else if ... utilizar elif:
- #!/bin/bash

```
echo "Enter your lucky number"
read n

if [ $n -eq 101 ];
then
echo "You got 1st prize"
elif [ $n -eq 510 ];
then
echo "You got 2nd prize"
elif [ $n -eq 999 ];
then
echo "You got 3rd prize"

else
echo "Sorry, try for the next time"
fi
```

Instrucción case

- Evaluación múltiple:
- case ... esac

```
case expression in
   pattern1 )
      statements ;;
   pattern2 )
      statements ;;
   ...
esac
```

```
#!/bin/bash
 read x
  case $x in
    1)
      echo "uno"
    ;;
    2)
      echo "dos"
    ;;
    *)
      echo "no sé qué número es"
    ;;
  esac
```

Instrucción while

while condición
do
Serie de comandos
done

- Con las expresiones let no utilizamos el $ para las variables

```
#!/bin/bash
# Ejemplo de bucle while, con expresiones let
echo “numero de iteraciones:”
read n
echo “el valor leído es:  $n“

i=0
while [ $i –lt $n ]; do
        echo “Valor de $i”
        i = $(($i+1))
done
```

Instrucción for

for variable in lista de valores
do
    Serie de comandos
Done

```
#!/bin/bash
for var in a b c 1 5 4
do
    echo “\$var=$var”
done
echo fin
```

- Salida:
- $var=a
- $var=b
- …
- $var=4
- fin

Instrucción for

- Podemos iterar por los parámetros recibidos:

```
for arg in "$@"
do
    echo "Parámetro $arg"
done
```

- Para iterar por una secuencia de valores:

```
max=10
for i in `seq 1 $max`
do
    echo “valor de i: $i”
done
```

Características avanzadas de Shell

- Variables locales y de entorno
- Sustitución de variables
- Sustitución de alias
- Sustitución de comandos

Variables locales y globales

- La Shell tiene dos tipos de variables: **Locales** y **Globales**

  - Las variables **locales**:
    - Se escriben en minúsculas (para distinguir)
    - Son visibles en la Shell donde se definieron.
    - Pero se pueden exportar a un proceso hijo (*siempre de Padre a Hijo, NO al revés*).

  - Las variables **globales**:
    - Se escriben en minúsculas.
    - Hay que exportarlas con: **export**
    - Son visibles para cualquier proceso en ejecución o que se ejecute desde el Shell.
    - Se pueden consultar con el comando: **set (muestra variables y funciones de la Shell) → set | more**
    - En Ubuntu el comando: **env (muestra también las de entorno)**

  - Las variables de **entorno**:
    - Se escriben en **MAYÚSCULAS**.
    - Las establece el sistema cuando se inicia una sesión.
    - Se muestran con el comando: **printenv**

Variables Locales

- El nombre: caracteres alfanuméricos y el subrayado.

- Para definirlas:
  - **var=valor** (sin espacios)

- Si el valor tiene más de una palabra tiene que ir entrecomillas.

- Ejemplos:
  - var1=valor1
  - var2=palabra
  - var3=“mas palabras”

  - Con cadena vacía:
    - var4=“”
    - var5=

Variables Locales

- Si una variable ya existe se machaca.

- Para ver el contenido de la variable (***OJO con $***)
  - echo **$**var

- Si el contenido de la variable lleva caracteres especiales: $ * …
  - Utilizar comillas simples.

  - Prueba:
    - var1=‘$$$$****’
    - var2=“$$$$****”

Variables Globales

- Las **variables globales** se crean igual que las locales, pero hay que exportarlas con el comando: **export**
  - var1=valor1
  - **export var1**
  - echo $var1

  - También es válido:
    - **export var2=valor2**

- Cuando la variable se exporta y creamos otra Shell (dentro de la propia terminal, llamando al comando **bash**) se puede consultar el valor de la variable (se transfiere al proceso hijo)

- En otra terminal nueva NO se vería, habría que hacerla permanente para poder mantenerla.

- De la Shell Hija, exportar a la Padre NO se puede.

Sustitución de alias

- Linux permite crear y eliminar alias de comandos.

- Se pueden añadir nuevos alias a la lista.

  alias cont='ls | wc -l'

- Se pueden eliminar los alias con:

  unalias cont

Sustitución de comandos

- Permite que la salida estándar de un programa se utilice como parte de la línea que se va a interpretar.

- Existen dos opciones, con el mismo funcionamiento:
  - $(comando)
  - `comando`

- En el segundo caso se está utilizando la tilde francesa o acento grave, que no debe confundirse con las comillas simples.
  - Para escribirla, hay que pulsar la tecla correspondiente a ` y pulsar espacio.

- El Shell ejecutará comando, capturará su salida estándar y sustituirá $(comando) por la salida capturada.

- Por ejemplo, para almacenar en una variable el nombre de todos los ficheros con extensión .sh del directorio actual, podría escribir:
  - VAR=` ls *.sh`
  - O, por ejemplo, para matar el proceso con nombre firefox-bin, podría usar:
    - kill -9 $(pidof firefox-bin)

Expansión de ~

- Las apariciones de la virgulilla (o tilde de la ñ) dentro de una línea, que no se encuentren entrecomilladas, se expanden de la siguiente manera:
  - **~** → Valor de la variable HOME.
  - Por ejemplo: **cd ~** Nos lleva al directorio personal (HOME).

  - **~login** → Si login es un nombre de usuario del sistema, se expande la ruta absoluta del directorio de inicio de sesión de ese usuario.

  - Por ejemplo: **var=~antonio ; echo $var** (se imprime: /home/antonio)

Generación de nombres de archivos

- Conceptos sobre caracteres genéricos.
- Caracteres genéricos.

Conceptos sobre caracteres genéricos

- Los caracteres genéricos permiten escribir patrones (como las expresiones regulares) para los nombres de archivos.

- Se utilizan en búsquedas o en comandos para copiar, borrar, buscar, etc. Ficheros que cumplan una serie de características.

- Dentro de estos caracteres especiales tenemos:
  - Asterisco *****
  - Interrogación **?**
  - Corchetes **[]**
  - Clases predefinidas **[:alpha:] [:digit:]**
  - Llaves **{}**

Caracteres genéricos

- **Asterisco ***
  - Se sustituye por una cadena de caracteres que puede ser nula.
  - Por ejemplo: ls r*o
  - El archivo **ro** se tomaría y **r**osad**o** también.
  - Se puede utilizar más de un asterisco.

- **La interrogación ?**
  - Sustituye un único carácter, excepto el punto en el inicio del nombre del fichero.
  - Por ejemplo, ls ro?o → rojo

Caracteres genéricos

- **Corchetes []**
  - Dentro de los corchetes se especifica una lista de caracteres.
  - Cada lista representa un único carácter.
  - Por ejemplo: [**abc**] se reemplaza por una **a**, una **b**, una **c**.
  - Si incluimos un espacio dentro, formará parte de la lista de caracteres.
  - Se pueden especificar rangos: [a-z][A-Za-z0-9]
  - Si el guión se coloca así: [ab-] el guión es tal cual, no indica rango.
  - Dentro del corchete como primer carácter ^ o !, indica que niega esos caracteres. [^0-9] → Que no sea un dígito.

- Todos los caracteres genéricos se pueden combinar: ls ?[ol]*
  - Listar archivos cuyo nombre contiene una ‘o’ o una ‘l’ en la 2ª posición.

Caracteres genéricos

- **Clases predefinidas:**
  - [:alpha:] → letras [a-zA-Z]
  - [:digit:] → números
  - [:alnum:] → Alfanumérico, equivale a [0-9a-z-A-Z]
  - [:lower:] → [a-z]
  - [:upper:] → [A-Z]
  - [:space:] → Espacio y tabulador horizontal
  - [:xdigit:] → Hexadecimal, [0-9A-Fa-f]
  - [:graph:] → Caracteres visible entre el ascii 33 y 126.
  - [:print:] → Igual que graph pero se añade el espacio.

Caracteres genéricos

- **Las llaves: {}**
  - Permiten especificar listas de patrones.
  - La sintaxis: **{abc, def}** se reemplaza por la cadena **abc** o la cadena **def**.
  - Por ejemplo: **{az, ro}*** corresponde a todos los nombres de archivos que empiezan por la cadena **az** o la cadena **ro**.
  - Dentro de las llaves se pueden utilizar los caracteres anteriores.
  - Por ejemplo: **ls {*ro[sj]*, ?a*}**

Caracteres genéricos

- Operadores de correspondencia extendida:
  - Si la opción **extglob** está activada, por el comando: **shopt**

  - Se pueden utilizar los siguientes patrones:
    - **?(patrón)**
      - Cero o una instancia de los motivos indicados.
    - ***(patrón)**
      - Cero o N-instancias.
    - **+(patrón)**
      - Una o N-instancias.
    - **@(patrón)**
      - Exactamente una instancia.
    - **!(patrón)**
      - Todo, menos lo indicado en el patrón.

Entrecomillado

- Conceptos de entrecomillado
- Caracteres de entrecomillado

Concepto de entrecomillado

- El **shell** entiende los siguientes caracteres como **metacaracteres** o caracteres que describen otros caracteres: ****, ^, $, ., [,],* , ?

- El carácter **** permite representar en la línea de comandos los caracteres especiales anteriores.
  - Por ejemplo, ***** no representaría cualquier cadena de caracteres sino el carácter ***.**
  - Esto se denomina: **escapar un carácter especial**.

Caracteres de entrecomillado

- **Comillas simples**, ('). El shell **no procesa ni variables ni comandos**.
- **$ echo 'mi espacio es $HOME y mi ordenador se llama hostname'**
- $ mi espacio es $HOME y mi ordenador se llama hostname

- **Comillas dobles**, ("). El shell procesa y **expande** las variables, pero no los comandos.
- $ echo "mi espacio es **$HOME** y mi ordenador se llama hostname"
- $ mi espacio es /home/antonio y mi ordenador se llama hostname

- **Backticks**, (`). El shell intenta procesar cada palabra entre comillas como un comando. Si hay variables, se evalúan primero y luego se procesan como un comando.
- $ echo `**ls $HOME**`
- $ En este caso ejecuta el comando **ls** sobre la variable $**HOME** y lista el contenido del directorio.