SISTEMAS OPERATIVOS

Contenido

1. SISTEMAS OPERATIVOS

Un **sistema operativo (SO)** es el software fundamental que permite que el hardware de un ordenador funcione y se comunique con los programas que utilizamos. Es como un **intermediario** entre el usuario, las aplicaciones y el hardware.

¿Qué hace un sistema operativo?

- **Gestiona los recursos del sistema**: CPU, memoria, discos, dispositivos de entrada/salida.
- **Permite ejecutar programas**.
- **Ofrece una interfaz al usuario**: gráfica (GUI) o de línea de comandos.
- **Controla el acceso a los archivos y la seguridad del sistema**.
- **Organiza tareas y procesos** (multitarea, planificación).

Tipos de sistemas operativos

1. Por el número de usuarios y tareas

Tipo	Características
Monousuario	Solo un usuario puede usar el sistema a la vez. Ej: MS-DOS.
Multiusuario	Varios usuarios pueden usar el sistema al mismo tiempo. Ej: Unix, Linux, Windows Server.
Monotarea	Solo ejecuta una tarea a la vez. Ej: sistemas antiguos.
Multitarea	Ejecuta varias tareas simultáneamente. Ej: Windows, macOS, Linux.

2. Por el tipo de interfaz

Tipo	Características
De línea de comandos (CLI)	Interacción mediante comandos. Ej: MS-DOS, Bash en Linux.
Gráficos (GUI)	Uso de ventanas, íconos y ratón. Ej: Windows, macOS.

3. Por el tipo de uso o entorno

Tipo	Características
Sistemas de escritorio	Orientados a usuarios finales. Ej: Windows, macOS, Ubuntu.
Sistemas de servidor	Optimizados para servicios de red y alto rendimiento. Ej: Windows Server, CentOS, Ubuntu Server.
Sistemas embebidos	SO ligeros integrados en dispositivos (coches, microondas, TV, IoT). Ej: FreeRTOS, VxWorks.
Sistemas móviles	Diseñados para smartphones y tablets. Ej: Android, iOS.
Sistemas en tiempo real (RTOS)	Responden rápidamente a eventos. Usados en robótica, medicina, aviación. Ej: QNX, RTLinux.
Sistemas distribuidos	Coordinan múltiples ordenadores para funcionar como uno solo. Ej: Hadoop, sistemas de computación en la nube.

Tipos de sistemas operativos (monousuario, multiusuario, monotarea, multitarea)

Existen cuatro combinaciones básicas:

1. **Sistemas operativos monousuario**

   Permiten que un solo usuario utilice el sistema en un momento dado. No significa que no pueda haber varias cuentas, sino que solo una sesión de trabajo está activa al mismo tiempo.

   Ejemplo: MS-DOS.

2. **Sistemas operativos multiusuario**

Permiten que varios usuarios trabajen simultáneamente con el sistema, ya sea desde distintos terminales físicos o mediante conexiones remotas. El sistema reparte los recursos entre ellos y aísla los procesos de cada usuario para que no interfieran.

Ejemplo: Unix, Linux, Windows Server.

3. **Sistemas operativos monotarea**

   Solo permiten ejecutar una tarea o programa a la vez. Si el usuario quiere abrir otra aplicación, debe cerrar la anterior.

   Ejemplo: sistemas antiguos como MS-DOS.

4. **Sistemas operativos multitarea**

   Pueden ejecutar varios programas a la vez. Aunque en realidad el procesador solo ejecuta una instrucción en cada momento, el sistema reparte el tiempo entre las tareas de forma tan rápida que parecen funcionar en paralelo. Ejemplo: Windows, Linux, macOS.

En la práctica, los sistemas modernos como Windows, Linux o macOS son **multiusuario y multitarea**, lo que los hace más eficientes y versátiles. Esta clasificación ayuda a entender las capacidades del sistema y su adecuación a diferentes contextos, como el uso personal, profesional o empresarial.

Interfaces del sistema operativo (línea de comandos vs interfaz gráfica)

Existen principalmente dos tipos:

1. **Interfaz de línea de comandos (CLI)**

   Es una forma de comunicación basada en texto. El usuario escribe comandos en una consola o terminal para dar instrucciones al sistema. Características:

   - Requiere conocimientos de sintaxis y comandos específicos
   - Es más rápida y precisa para tareas repetitivas o complejas

- Ocupa muy pocos recursos del sistema
  Ejemplos: MS-DOS, Bash en Linux, PowerShell en Windows

2. **Interfaz gráfica de usuario (GUI)**

   Es una forma visual de interactuar con el sistema usando ventanas, menús, botones e iconos. Se maneja normalmente con un ratón y teclado, o con los dedos en pantallas táctiles.
   Características:

- Más fácil de usar, ideal para usuarios sin conocimientos técnicos
- Permite trabajar de forma intuitiva y multitarea
- Requiere más recursos del sistema
  - Ejemplos: Windows, macOS, entornos de escritorio en Linux como GNOME o KDE

En muchos sistemas operativos actuales se pueden usar ambos tipos de interfaz. Por ejemplo, en Windows puedes abrir una consola de comandos, y en Linux puedes alternar entre la terminal y el entorno gráfico según lo que necesites hacer. Esta combinación permite un control avanzado para usuarios expertos y facilidad de uso para principiantes.

Ejemplos de sistemas operativos actuales (Windows, Linux, macOS, Android)

1. **Windows**

   Desarrollado por Microsoft. Es el sistema operativo más usado en ordenadores personales. Tiene una interfaz gráfica fácil de usar y es compatible con una gran variedad de programas y hardware.

   Versiones actuales: Windows 10, Windows 11

   Uso: ordenadores de escritorio, portátiles, empresas

2. **macOS**

   Desarrollado por Apple. Es exclusivo de los ordenadores Mac. Está basado en Unix y se caracteriza por su diseño, estabilidad y rendimiento en tareas creativas como edición de vídeo o diseño gráfico.

Uso: ordenadores Mac, entornos creativos

3. **Linux**

   Es un sistema operativo libre y de código abierto. Existen muchas versiones llamadas distribuciones, como Ubuntu, Debian o Fedora. Es muy usado en servidores, centros de datos, supercomputadoras y también por usuarios que valoran la personalización.

   Uso: servidores, programación, ciberseguridad, educación, ordenadores personales

4. **Android**

   Basado en Linux, está diseñado para dispositivos móviles como teléfonos y tablets. Es el sistema operativo móvil más utilizado en el mundo.
   Uso: smartphones, tablets, televisores inteligentes

5. **iOS**

   Desarrollado por Apple para sus dispositivos móviles. Está basado en una versión reducida de macOS. Tiene un sistema cerrado y muy controlado, centrado en la seguridad y la experiencia de usuario.

   Uso: iPhone, iPad, iPod Touch

6. **Otros ejemplos**

   - ChromeOS: desarrollado por Google, basado en Linux, pensado para trabajar en la nube
   - Unix y derivados (como FreeBSD): usados principalmente en servidores y sistemas industriales
   - Sistemas embebidos: como FreeRTOS, usados en coches, electrodomésticos o dispositivos IoT

Cada uno de estos sistemas operativos está diseñado para un tipo de dispositivo y usuario distinto, según las necesidades de rendimiento, facilidad de uso, seguridad y compatibilidad.

Estructura interna del sistema operativo

El sistema operativo está compuesto por varios componentes que trabajan juntos para controlar el hardware y ejecutar programas. Los principales son:

- **Núcleo o kernel**: es el corazón del sistema operativo. Se encarga de gestionar la CPU, la memoria, los dispositivos y la comunicación entre procesos.
- **Gestión de procesos**: coordina la ejecución de programas, asigna tiempo de CPU y controla su estado (en ejecución, esperando, detenido).
- **Gestión de memoria**: reparte la memoria RAM entre los programas activos y se asegura de que no interfieran entre ellos.
- **Sistema de archivos**: organiza los datos en carpetas y archivos dentro de los discos.
- **Controladores (drivers)**: permiten que el sistema operativo se comunique con el hardware como impresoras, tarjetas gráficas, ratón o teclado.
- **Interfaz de usuario**: puede ser gráfica o por línea de comandos. Permite que el usuario interactúe con el sistema.

Gestión de procesos

Un proceso es un programa en ejecución. El sistema operativo gestiona todos los procesos activos del sistema.

Funciones principales:

- **Planificación de procesos**: decide qué proceso usa la CPU en cada momento, usando algoritmos como round-robin o prioridades.
- **Creación y destrucción de procesos**: cuando abres o cierras un programa, el sistema crea o elimina su proceso correspondiente.
- **Estados del proceso**: un proceso puede estar en ejecución, en espera, listo para ejecutarse, o terminado.
- **Multiprocesamiento**: si hay varios núcleos, el sistema puede ejecutar varios procesos realmente al mismo tiempo.

Esto permite que el ordenador ejecute muchos programas a la vez, incluso si tú solo ves uno en pantalla.

Gestión de memoria

El sistema operativo controla cuánta memoria RAM se asigna a cada proceso y cómo se organiza.

Funciones principales:

- **Asignación de memoria**: da espacio a cada proceso cuando se inicia.
- **Protección de memoria**: evita que un proceso acceda a la memoria de otro, lo que puede causar errores o fallos.
- **Memoria virtual**: si la RAM se llena, se usa parte del disco duro como memoria temporal (swap), aunque sea más lenta.
- **Liberación de memoria**: cuando un proceso termina, la memoria que usaba se libera y queda disponible para otros.

Una buena gestión de memoria permite que el sistema funcione rápido y estable, incluso con muchos programas abiertos.

Sistema de archivos

El sistema de archivos es la parte del sistema operativo que organiza y gestiona los datos almacenados en los discos duros, memorias USB, discos SSD u otros dispositivos de almacenamiento.

Funciones principales:

- **Organización de datos**: los datos se estructuran en archivos y carpetas, lo que facilita su acceso y uso.
- **Jerarquía**: los archivos se organizan en una estructura de árbol, con una carpeta raíz y subcarpetas.
- **Gestión del espacio**: controla qué partes del disco están ocupadas o libres y almacena los datos sin que se sobreescriban entre sí.
- **Tipos de sistema de archivos**: según el sistema operativo se usan distintos formatos, como NTFS en Windows, HFS+ o APFS en macOS, ext4 en Linux.

- **Permisos y propiedad**: cada archivo tiene un propietario y unos permisos que controlan quién puede leerlo, escribirlo o ejecutarlo.

Esto permite guardar, acceder, modificar y eliminar información de manera segura y eficiente.

Controladores y dispositivos

Los controladores (o drivers) son programas que permiten al sistema operativo comunicarse con el hardware del equipo.

Funciones principales:

- **Interfaz entre el hardware y el sistema**: traducen las órdenes del sistema operativo a instrucciones que el dispositivo entiende, y viceversa.
- **Tipos de dispositivos**: impresoras, tarjetas gráficas, discos duros, teclados, cámaras, micrófonos, etc.
- **Instalación de drivers**: muchos dispositivos necesitan un controlador específico para funcionar correctamente. Algunos vienen preinstalados, otros se descargan automáticamente.
- **Actualización y compatibilidad**: un controlador desactualizado puede causar errores o impedir el funcionamiento correcto de un dispositivo.

Gracias a los controladores, el sistema operativo puede funcionar con diferentes tipos de hardware sin necesidad de modificar su núcleo.

Seguridad del sistema operativo

El sistema operativo tiene mecanismos para proteger el equipo, los datos y los usuarios frente a accesos no autorizados, errores y amenazas externas.

Principales elementos de seguridad:

- **Cuentas de usuario**: cada persona que usa el sistema tiene una cuenta con permisos definidos. Existen cuentas estándar y cuentas de administrador.
- **Contraseñas y autenticación**: controlan el acceso al sistema. Algunos sistemas también permiten autenticación biométrica.

- **Permisos y control de acceso**: determinan qué archivos y programas puede usar cada usuario.
- **Actualizaciones de seguridad**: corrigen vulnerabilidades que podrían ser aprovechadas por malware o hackers.
- **Firewall o cortafuegos**: filtra las conexiones de red para evitar accesos no autorizados.
- **Antivirus y herramientas antimalware**: detectan y eliminan software malicioso.
- **Cifrado**: protege los datos almacenados para que no puedan ser leídos sin autorización.

La seguridad es esencial para mantener la integridad del sistema y proteger tanto los archivos como la información personal del usuario.

Historia breve de los sistemas operativos

Los sistemas operativos han evolucionado a lo largo del tiempo para adaptarse a los avances tecnológicos y a las necesidades de los usuarios.

- **Años 50**: no existían sistemas operativos como tal. Se programaba directamente en el hardware usando tarjetas perforadas.
- **Años 60**: aparecen los primeros sistemas operativos básicos para gestionar tareas. Nace Unix, uno de los primeros SO multitarea y multiusuario.
- **Años 70**: se populariza Unix en universidades y empresas. Comienzan a aparecer los primeros sistemas con interfaz de línea de comandos.
- **Años 80**: se desarrolla MS-DOS para los primeros ordenadores personales. En 1984 aparece el primer sistema con interfaz gráfica: el Macintosh.
- **Años 90**: nace Windows 95 con entorno gráfico integrado y Linux como alternativa libre y gratuita a Unix.
- **Siglo XXI**: los sistemas operativos se hacen más seguros, visuales y compatibles con internet. Aparecen los sistemas móviles como Android e iOS. También se expande el uso de sistemas embebidos y en la nube.

Diferencias entre Unix, Linux, Windows y macOS

Aunque todos son sistemas operativos, tienen orígenes, licencias y usos distintos.

- **Unix**: sistema desarrollado en los años 70. Originalmente comercial, muy usado en servidores y entornos industriales. Muy estable y seguro.
- **Linux**: es un sistema operativo libre inspirado en Unix. Tiene muchas distribuciones distintas y es muy usado en servidores, supercomputadoras y desarrolladores.
- **Windows**: sistema propietario de Microsoft. Es el más usado en ordenadores personales. Muy compatible con programas comerciales. Interfaz gráfica fácil de usar.
- **macOS**: sistema propietario de Apple. Está basado en Unix (específicamente en BSD) y se usa solo en ordenadores Mac. Muy orientado al diseño, multimedia y rendimiento.

Característica	Windows	Unix	macOS
Desarrollador	Microsoft	AT&T (originalmente), ahora varias entidades	Apple
Interfaz	Gráfica (GUI), también CLI (cmd/powershell)	Originalmente solo CLI, ahora también GUI	Gráfica (GUI) basada en Unix
Licencia	Comercial, propietario	Mayormente libre o con licencia BSD	Comercial, propietario
Núcleo (kernel)	Windows NT	Varios: System V, BSD, etc.	Basado en XNU (derivado de BSD y Mach)
Seguridad	Mejorada en versiones recientes, pero históricamente vulnerable	Alta seguridad, orientado a entornos multiusuario	Alta seguridad, muy estable

Característica	Windows	Unix	macOS
Compatibilidad de software	Muy alta para videojuegos y programas comerciales	Alta en servidores, menor para software comercial	Alta en entorno creativo (diseño, edición)
Usos más comunes	Escritorio, juegos, empresas	Servidores, supercomputación, sistemas críticos	Diseño, multimedia, usuarios Apple

Qué son las distribuciones de Linux y Unix

Una distribución es una versión personalizada de un sistema operativo que incluye el núcleo (kernel) junto con otras herramientas, programas y configuraciones.

- **Distribuciones de Linux**: son muchas y variadas, adaptadas a distintos tipos de usuarios.
    - **Ubuntu**: fácil de usar, ideal para principiantes
    - **Debian**: muy estable, base de muchas otras distribuciones
    - **Fedora**: orientada a desarrolladores, patrocinada por Red Hat
    - **Arch Linux**: muy configurable, orientada a usuarios avanzados
    - **Kali Linux**: especializada en seguridad informática
- **Distribuciones de Unix**: muchas son comerciales, con fines empresariales o industriales.
    - **AIX**: desarrollada por IBM
    - **HP-UX**: desarrollada por Hewlett-Packard
    - **Solaris**: desarrollada por Sun Microsystems (ahora Oracle)
    - **BSD (Berkeley Software Distribution)**: versión libre de Unix con variantes como FreeBSD, OpenBSD, NetBSD
    - **macOS**: aunque es propietario, internamente está basado en BSD

Estas distribuciones permiten adaptar el sistema operativo a diferentes necesidades técnicas y tipos de usuarios.

2. UBUNTU SERVER

Ubuntu Server es una versión del sistema operativo Ubuntu diseñada específicamente para ser utilizada en servidores. A diferencia de Ubuntu Desktop, no incluye una interfaz gráfica por defecto (aunque puede instalarse), ya que está optimizado para funcionar de forma ligera y eficiente en entornos donde lo más importante es el rendimiento, la seguridad y la estabilidad.

Características principales de Ubuntu Server:

1. **Sin entorno gráfico por defecto**
    - Se administra principalmente mediante línea de comandos (CLI).
    - Esto reduce el consumo de recursos y aumenta la seguridad.
2. **Soporte para múltiples arquitecturas**
    - Compatible con arquitecturas x86_64 (AMD64), ARM y otras, lo que lo hace útil desde servidores en la nube hasta dispositivos IoT.
3. **Gran comunidad y soporte empresarial**
    - Canonical, la empresa detrás de Ubuntu, ofrece soporte LTS (Long Term Support) de 5 años.
    - También existen versiones intermedias con soporte de 9 meses.
4. **Amplia compatibilidad con software y servicios**
    - Ideal para servicios como web servers (Apache, Nginx), bases de datos (MySQL, PostgreSQL), contenedores (Docker), virtualización (KVM), etc.
    - Compatible con herramientas de automatización como Ansible, Puppet o Terraform.
5. **Seguridad**
    - Incluye actualizaciones de seguridad frecuentes.
    - Tiene funcionalidades como AppArmor, actualizaciones automáticas y soporte para UFW (Uncomplicated Firewall).
6. **Despliegue en la nube**
    - Muy usado en plataformas como AWS, Azure o Google Cloud.
    - Canonical ofrece *Ubuntu Pro*, una versión con soporte extendido para entornos empresariales.

7. **Instalación sencilla y flexible**
    - El instalador permite elegir entre distintas configuraciones de servidor como: servidor SSH, servidor de base de datos, servidor de archivos, etc.

SSH (Secure Shell)

SSH (Secure Shell) es un **protocolo de red seguro** que permite a los usuarios controlar y modificar un servidor remoto a través de Internet. Es una herramienta fundamental en administración de sistemas y desarrollo backend, especialmente en sistemas como **Ubuntu Server**, donde se trabaja mayoritariamente por terminal.

SSH permite:

- **Conectarse a un servidor remoto** de forma segura usando línea de comandos.
- **Ejecutar comandos a distancia**, como si estuvieras sentado delante del servidor.
- **Copiar archivos** entre máquinas (con scp o sftp).
- **Crear túneles seguros** para servicios como bases de datos o entornos de desarrollo.
- **Administrar múltiples máquinas** sin necesidad de acceso físico

¿Cómo funciona SSH?

SSH funciona mediante un sistema **cliente-servidor**:

- **Servidor SSH**: debe estar instalado y en ejecución en la máquina remota (por ejemplo, sshd en Ubuntu Server).
- **Cliente SSH**: es la máquina desde la que te conectas.

El cliente establece una conexión con el servidor en el **puerto 22** (por defecto) y se autentica mediante usuario y contraseña o mediante un **par de claves (autenticación por llave pública/privada)**.

SSH permite conectarse de forma segura a otro sistema remoto. Desde un Ubuntu Server, puedes conectarte a otro servidor con:

```
ssh usuario@192.168.1.100
```

Te pedirá la contraseña del usuario remoto.

Si quieres copiar un archivo desde tu máquina a otra mediante SSH:

```
scp archivo.txt usuario@192.168.1.100:/home/usuario/
```

Para usar llaves SSH en lugar de contraseñas, primero se genera un par de claves:

```
ssh-keygen
```

Luego se copia la clave pública al servidor remoto:

```
ssh-copy-id usuario@192.168.1.100
```

A partir de ahí podrás conectarte sin introducir contraseña.

Introducción al entorno

Ubuntu Server no tiene entorno gráfico por defecto. Esto significa que todo se hace a través de una interfaz de texto llamada shell. La shell más común en Ubuntu Server es Bash. A través de ella se escriben comandos para interactuar con el sistema operativo.

Por ejemplo, para saber en qué directorio estás situado puedes usar:

```
pwd
```

Este comando muestra la ruta completa del directorio actual.

Para cambiar de directorio se utiliza:

```
cd /etc
```

Este comando te lleva al directorio /etc.

Si quieres limpiar la pantalla puedes escribir:

```
clear
```

El comando man permite consultar los manuales de uso de los comandos. Por ejemplo:

```
man ls
```

Esto abre la ayuda sobre el comando ls.

Comando ls

El comando ls en Linux (incluido en Ubuntu Server) se utiliza para **listar el contenido de un directorio**. Es uno de los comandos más usados al trabajar con la terminal, porque te permite ver qué archivos y carpetas hay, así como sus permisos, propietarios, tamaños y fechas.

Uso básico de ls

```
ls
```

Este comando muestra los archivos y carpetas del directorio actual, pero de forma muy básica: solo nombres.

Opciones más útiles de ls

ls -l → formato largo (long listing)

Muestra información detallada de cada archivo: permisos, número de enlaces, propietario, grupo, tamaño, fecha y nombre.

```
ls -l
```

Ejemplo de salida:

-rw-r--r-- 1 juan alumnos 1240 jul 17 12:00 informe.txt

drwxr-xr-x 2 juan alumnos 4096 jul 17 11:45 documentos

Significado de cada columna:

- Tipo de archivo (- archivo, d directorio, l enlace)
- Permisos
- Número de enlaces
- Propietario
- Grupo
- Tamaño en bytes
- Fecha y hora de última modificación
- Nombre

ls -a → incluye archivos ocultos

Muestra **todos los archivos**, incluidos los ocultos (los que empiezan por . como .bashrc o .profile).

```
ls -a
```

Útil para ver configuraciones ocultas o comprobar si un archivo .git existe.

ls -la o ls -al → largo + ocultos

Combina ambas opciones para mostrar información detallada de **todos** los archivos, incluidos los ocultos.

```
ls -la
```

Muy usado por administradores de sistemas.

ls -h → human-readable (junto con -l)

Hace que los tamaños de archivo se muestren en **formato legible** (KB, MB, GB) en vez de bytes.

```
ls -lh
```

Ejemplo:

-rw-r--r-- 1 juan alumnos 1.2K jul 17 12:00 informe.txt

ls -R → recursivo

Lista **también el contenido de todos los subdirectorios**.

```
ls -R
```

Útil para ver toda la estructura de carpetas desde una raíz.

ls -t → ordenado por fecha de modificación (más reciente primero)

```
ls -lt
```

Esto es útil para ver qué archivos han sido modificados más recientemente.

Puedes combinarlo con -h:

```
ls -lth
```

ls -S → ordenado por tamaño (de mayor a menor)

```
ls -lS
```

Para ver qué archivos ocupan más espacio.

ls -d */ → solo muestra directorios

```
ls -d */
```

Esto te da un listado de solo carpetas, útil si estás en un directorio con cientos de archivos.

ls -1 → un archivo por línea

```
ls -1
```

Muestra la lista de archivos con **uno por línea**, útil en scripts o para procesar salidas.

Combinaciones útiles de ls

- Listar todo con detalles legibles y ordenado por tamaño:

`ls -lhSa`

- Ver solo los archivos ocultos:

`ls -ld .*`

- Listar todos los archivos y subdirectorios de manera recursiva, con detalles:

`ls -lR`

Colores en ls

En muchas distribuciones (incluida Ubuntu Server si tienes un terminal compatible), ls resalta tipos de archivos con colores:

- Azul: directorio
- Verde: ejecutable
- Rojo: archivo comprimido
- Gris o apagado: archivo oculto

Esto se controla con la variable de entorno LS_COLORS. Puedes personalizarla o dejarla por defecto.

Alias comunes

Muchas distribuciones (incluido Ubuntu) ya configuran alias como:

`alias ll='ls -l'`

`alias la='ls -la'`

Puedes ver los alias definidos con:

`alias`

Y añadirlos a tu archivo .bashrc si quieres que se guarden.

El archivo .bashrc

El archivo .bashrc es un **script de inicialización** que se ejecuta automáticamente cada vez que un usuario abre una **nueva terminal interactiva** en un sistema basado en Linux, como Ubuntu Server, cuando está usando el shell Bash.

Su función es **personalizar el entorno de la línea de comandos del usuario**, configurando variables, alias, funciones y otros comportamientos que afectan a la sesión.

¿Dónde se encuentra el .bashrc?

Cada usuario tiene su propio .bashrc dentro de su directorio personal. Por ejemplo:

/home/juan/.bashrc

Cuando el usuario juan inicia sesión o abre una terminal, su shell ejecuta ese archivo si está usando bash.

¿Cuándo se ejecuta .bashrc?

- Se ejecuta cuando se abre **una nueva terminal interactiva no de login** (por ejemplo, al abrir una consola o una nueva ventana de terminal).
- No se ejecuta directamente en sesiones de **login por consola** o SSH, a menos que el archivo .profile esté configurado para cargarlo (lo cual es común en Ubuntu).

En Ubuntu Server, el archivo .profile contiene esta línea:

```
if [ -n "$BASH_VERSION" ]; then
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi
```

Esto hace que .bashrc también se ejecute al iniciar sesión por SSH.

¿Para qué sirve .bashrc? Usos comunes

1. Definir alias

Permite crear atajos para comandos frecuentes.

```
alias ll='ls -l'
alias gs='git status'
alias ..='cd ..'
```

Después de añadirlos, puedes hacer que surtan efecto con:

```
source ~/.bashrc
```

O simplemente abriendo una nueva terminal.

2. Definir variables de entorno personalizadas

```
export EDITOR=nano
export PATH=$PATH:$HOME/bin
```

Esto añade el directorio ~/bin al PATH del usuario.

3. Cambiar el prompt del shell (PS1)

```
PS1='\u@\h:\w\$ '
```

Esto configura el aspecto del prompt. En este ejemplo, mostrará:

usuario@maquina:/ruta$

4. Activar colores en ls y otros comandos

```
export LS_OPTIONS='--color=auto'
alias ls='ls $LS_OPTIONS'
```

Aunque muchas distribuciones ya lo traen configurado por defecto.

5. Cargar scripts o funciones propias

Puedes definir funciones útiles:

```
extract () {
    if [ -f $1 ] ; then
        case $1 in
            *.tar.bz2)   tar xvjf $1    ;;
            *.tar.gz)    tar xvzf $1    ;;
            *.zip)       unzip $1       ;;
            *)           echo "Archivo no soportado: $1" ;;
        esac
    else
        echo "Fichero no encontrado"
    fi
}
```

Y luego usar extract archivo.zip como comando.

¿Cómo editar .bashrc?

1. Abre el archivo con un editor de texto:

```
nano ~/.bashrc
```

2. Haz los cambios que necesites (por ejemplo, añadir alias).
3. Guarda con Ctrl + O, luego Enter y sal con Ctrl + X.
4. Para aplicar los cambios sin reiniciar la terminal, ejecuta:

```
source ~/.bashrc
```

¿Qué diferencia hay con .profile o .bash_profile?

- .bashrc: se ejecuta en terminales interactivas **no de login**.
- .profile y .bash_profile: se ejecutan en sesiones de **login**.

Ubuntu suele usar .profile y desde allí carga .bashrc cuando corresponde. En otros sistemas como CentOS se usa .bash_profile.

Comando su

El comando su en Linux significa **"substitute user"** o **"switch user"**, y permite cambiar de usuario dentro de la misma sesión de terminal. Es especialmente conocido por su uso para convertirse en **superusuario o root**, es decir, obtener **acceso total al sistema**.

En sistemas como **Ubuntu Server**, su es útil, pero no tan común como sudo, que es el mecanismo recomendado por seguridad. Aun así, es importante entender su, cómo funciona y cuándo se puede usar.

¿Qué hace su?

Cuando escribes:

```
su
```

Sin argumentos, el sistema interpreta que quieres cambiar al **usuario root**, por lo que te pedirá la contraseña del root (si está habilitado).

Una vez autenticado, accedes a una **shell como el usuario root**, con todos los permisos del sistema. Puedes hacer cualquier cosa: borrar archivos de sistema, cambiar contraseñas, instalar o eliminar software, etc.

Ejemplo:

```
su
```

Password:

root@servidor:/home/usuario#

Aquí estás trabajando como root. Para salir y volver a tu usuario normal, basta con escribir:

```
exit
```

Cambiar a otro usuario (no root)

También puedes cambiar a cualquier otro usuario si sabes su nombre y su contraseña:

su pedro

El sistema te pedirá la contraseña del usuario pedro, y luego estarás logueado como él dentro de la misma terminal.

Diferencias entre su y su -

Ambos comandos te permiten convertirte en otro usuario, pero con diferencias importantes:

- su cambia de usuario **manteniendo el entorno actual** (variables, rutas, etc.)
- su - inicia una **sesión de login completa** del nuevo usuario, como si se hubiera conectado desde cero. Carga su entorno, su .bashrc, su $PATH, etc.

Ejemplo:

```
su - root
```

Esto es más seguro y predecible, especialmente si vas a ejecutar scripts o usar rutas específicas del sistema.

Activar la cuenta root en Ubuntu (por defecto desactivada)

En Ubuntu, la cuenta root viene desactivada para forzar el uso de sudo.

Para activar root y asignarle una contraseña:

```
sudo passwd root
```

Introduce tu contraseña de usuario (si tiene permisos de sudo), y luego una nueva contraseña para root.

A partir de ahí, podrás usar:

```
su
```

Y entrar como root con esa contraseña.

Para **volver a desactivar la cuenta root** (recomendado en muchos casos):

```
sudo passwd -l root
```

¿Qué diferencia hay entre su y sudo?

Característica	su	sudo
Cambia de usuario	Sí, cambia al root u otro	No cambia de usuario, ejecuta como root
Pide contraseña	Del usuario **al que te conectas**	Tu propia contraseña
Control de permisos	Todos los permisos de root	Solo comandos concretos permitidos
Seguridad	Menor (si root está habilitado)	Mayor (registro de acciones, control por grupos)
Uso recomendado	En Debian, CentOS, sistemas sin sudo	En Ubuntu y sistemas modernos

En Ubuntu Server, el uso recomendado es:

`sudo comando`

Ejemplo:

`sudo apt update`

`sudo reboot`

Pero si activas root por necesidad y usas su, asegúrate de saber lo que haces.

Gestión de archivos y directorios

Para crear un archivo vacío puedes usar el comando touch:

`touch ejemplo.txt`

Para crear un directorio:

`mkdir documentos`

Para copiar un archivo a otro nombre o a otra ubicación:

`cp ejemplo.txt copia.txt`

`cp ejemplo.txt /home/usuario/`

Para mover un archivo (o renombrarlo):

`mv ejemplo.txt nuevo_nombre.txt`

Para borrar un archivo o directorio:

```
rm nuevo_nombre.txt
```

```
rm -r documentos
```

Para ver qué tipo de archivo es uno determinado:

```
file ejemplo.txt
```

Para buscar una palabra dentro de un archivo:

```
grep "clave" ejemplo.txt
```

Para buscar archivos en todo el sistema puedes usar find:

```
find / -name ejemplo.txt
```

Lectura y edición de ficheros

COMANDOS PARA LEER FICHEROS

cat

Muestra el contenido completo de un archivo directamente en la terminal.

cat archivo.txt

- Útil para archivos pequeños.
- No permite navegación ni edición.

less

Muestra el contenido de un archivo **página a página** y permite desplazarse.

less archivo.txt

- Usa flechas o AvPág / RePág para moverse.
- Pulsa q para salir.
- Ideal para archivos largos.

more

Parecido a less, pero con menos funcionalidades.

```
more archivo.txt
```

- Se usa igual que less, aunque es menos flexible.

head

Muestra las primeras líneas de un archivo (por defecto, 10).

```
head archivo.txt
```

```
head -n 20 archivo.txt
```

tail

Muestra las últimas líneas del archivo (por defecto, 10).

```
tail archivo.txt
```

```
tail -n 15 archivo.txt
```

Muy útil para ver el final de logs, como:

```
tail -f /var/log/syslog
```

-f mantiene la salida actualizándose en tiempo real.

COMANDOS PARA EDITAR FICHEROS (EDITORES DE TEXTO EN TERMINAL)

nano

Editor de texto fácil y recomendado para principiantes.

nano archivo.txt

- Navegación con flechas.
- Para guardar: Ctrl + O, luego Enter.
- Para salir: Ctrl + X.
- Para buscar texto: Ctrl + W.

Ejemplo: editar configuración de red.

```
sudo nano /etc/netplan/00-installer-config.yaml
```

vim (o vi)

Editor avanzado, muy usado por administradores. Tiene curva de aprendizaje.

```
vim archivo.txt
```

- **Modo normal** (navegación): al abrir el archivo.
- **Modo inserción**: pulsa i para escribir.
- **Guardar y salir**: Esc, luego :wq y Enter.
- **Salir sin guardar**: Esc, luego :q!.

OTROS COMANDOS ÚTILES PARA EXPLORAR TEXTO

grep

Busca líneas que contienen una palabra clave.

```
grep "error" /var/log/syslog
```

Combinado con cat, less o tail:

```
cat archivo.txt | grep "clave"
```

wc (word count)

Cuenta líneas, palabras y caracteres.

`wc archivo.txt`

Solo número de líneas:

`wc -l archivo.txt`

Comparativa rápida de usos

Comando	¿Lee?	¿Edita?	¿Interfaz?	Ideal para…
cat	Sí	No	No	Leer archivos pequeños
less	Sí	No	Parcial	Leer archivos largos
nano	Sí	Sí	Sí	Editar fácilmente
vim	Sí	Sí	Sí	Edición profesional avanzada
head	Sí	No	No	Ver primeras líneas
tail	Sí	No	No	Ver últimas líneas / logs en vivo

Gestión de usuarios y grupos

Ubuntu Server gestiona el acceso al sistema mediante usuarios y grupos.

Para crear un nuevo usuario:

`adduser juan`

Para borrar un usuario:

`deluser juan`

Para añadir un usuario a un grupo:

`usermod -aG sudo juan`

Este ejemplo permite que el usuario juan pueda usar comandos como administrador.

Para cambiar la contraseña de un usuario:

`passwd juan`

Puedes ver los grupos a los que pertenece un usuario con:

`groups juan`

Y ver su UID y GID con:

`id juan`

La información de los usuarios está en el archivo /etc/passwd y la de los grupos en `/etc/group.`

Permisos de archivos y directorios

¿Qué son los permisos en Linux?

En Ubuntu Server, **cada archivo o directorio tiene permisos que determinan quién puede leerlo, escribir en él o ejecutarlo**. Estos permisos están asociados a tres tipos de usuarios:

1. **Propietario (user)**: el usuario que creó el archivo.
2. **Grupo (group)**: el grupo al que pertenece el archivo.
3. **Otros (others)**: cualquier otro usuario del sistema.

Cada uno de estos tres puede tener uno o más de los siguientes permisos:

- **r (read)**: permiso de lectura.
- **w (write)**: permiso de escritura.
- **x (execute)**: permiso de ejecución (para archivos ejecutables o acceso a directorios).

Cómo ver los permisos de un archivo o directorio

Usa el comando ls -l para ver los permisos:

```
ls -l ejemplo.txt
```

Salida típica:

```
-rw-r--r-- 1 juan juan  123 jul 17 12:00 ejemplo.txt
```

Significado:

- - → es un archivo (si fuera un directorio sería una d).
- rw- → permisos del propietario (lectura y escritura).
- r-- → permisos del grupo (solo lectura).
- r-- → permisos para otros usuarios (solo lectura).
- juan → propietario.
- juan → grupo propietario.

Cómo cambiar los permisos: el comando chmod

Para cambiar los permisos puedes usar chmod de dos formas: simbólica y numérica.

Forma simbólica

```
chmod u+x script.sh   # añade permiso de ejecución al propietario
chmod g-w ejemplo.txt # quita permiso de escritura al grupo
chmod o=r archivo.txt # da solo lectura a otros
chmod u=rwx,g=rx,o= ejemplo.txt # permisos detallados
```

Forma numérica

Los permisos se expresan como números:

- r = 4
- w = 2
- x = 1

Se suman según los permisos deseados para cada grupo:

- 7 = rwx (4+2+1)
- 6 = rw- (4+2)
- 5 = r-x (4+1)
- 4 = r-- (solo lectura)

Ejemplo:

```
chmod 755 script.sh
```

Significa:

- Propietario: rwx (7)
- Grupo: r-x (5)
- Otros: r-x (5)

Otro ejemplo:

```
chmod 644 ejemplo.txt
```

- Propietario: rw-
- Grupo: r--
- Otros: r--

Cómo cambiar propietario o grupo: chown y chgrp

Para cambiar el propietario:

```
sudo chown pedro archivo.txt
```

Para cambiar propietario y grupo a la vez:

```
sudo chown pedro:admin archivo.txt
```

Para cambiar solo el grupo:

```
sudo chgrp profesores archivo.txt
```

Permisos en directorios

En un directorio:

- **r (read)**: permite listar el contenido del directorio.
- **w (write)**: permite crear, borrar o renombrar archivos dentro del directorio.
- **x (execute)**: permite entrar al directorio (hacer cd).

Por ejemplo, si quieres que un usuario pueda acceder al directorio /datos, pero no ver lo que hay dentro, puedes hacer:

```
chmod 701 /datos
```

- Propietario: rwx (puede todo)
- Grupo: --- (nada)
- Otros: --x (solo puede entrar si sabe el nombre exacto de los archivos)

Tabla resumen de permisos de archivos y directorios en Linux (Ubuntu Server)

Permiso	Letra	Valor	Significado en archivos	Significado en directorios
Read	r	4	Permite leer el contenido del archivo	Permite listar los archivos del directorio
Write	w	2	Permite modificar o borrar el archivo	Permite crear, borrar o renombrar archivos dentro del directorio
Execute	x	1	Permite ejecutar el archivo si es un script o binario	Permite acceder al directorio con cd

Red en Linux: conceptos y pruebas básicas

Para ver la configuración de red actual, incluyendo la IP del servidor:

```
ip a
```

o bien, si está instalado net-tools:

```
ifconfig
```

Para comprobar la conexión a otra máquina o a internet:

```
ping google.com
```

Puedes detener el ping con Control + C.

Para ver la ruta que sigue un paquete:

```
traceroute google.com
```

Para comprobar los servidores DNS:

```
nslookup google.com
```

o también:

```
dig google.com
```

Para ver puertos y conexiones abiertas:

```
netstat -tuln
```

```
ss -tuln
```

FTP y SFTP

FTP es un protocolo para transferir archivos, aunque se considera inseguro si no se cifra.

Para eso existe SFTP, que funciona sobre SSH.

Conexión con SFTP:

```
sftp usuario@192.168.1.100
```

Dentro de la sesión puedes usar comandos como:

```
ls          # lista archivos remotos
cd          # cambia directorio remoto
get archivo.txt       # descarga archivo
put archivo.txt       # sube archivo
exit        # salir
```

Si usas ftp (menos recomendable), los comandos son similares, pero debes tener un servidor FTP instalado como vsftpd.

Shell Scripting básico

Un script de shell es un archivo de texto que contiene comandos Bash que se ejecutan en orden.

Ejemplo de script para mostrar fecha y directorio actual:

```
#!/bin/bash
echo "Hoy es:"
date
echo "Estás en el directorio:"
pwd
```

Se guarda como info.sh, se hace ejecutable con:

```
chmod +x info.sh
```

Y se ejecuta así:

```
./info.sh
```

Puedes usar variables:

```
nombre="Juan"
echo "Hola $nombre"
```

Estructuras de control:

```
if [ -f archivo.txt ]; then
  echo "El archivo existe"
else
  echo "No existe"
fi
```

Bucles:

```
for i in {1..5}
do
  echo "Número $i"
done
```

Uso de argumentos:

```
echo "Primer argumento: $1"
```

Compresión y archivado

Para empaquetar y comprimir archivos se usan tar y gzip.

Crear un archivo comprimido:

```
tar -czvf archivo.tar.gz carpeta/
```

Extraerlo:

```
tar -xzvf archivo.tar.gz
```

Otros formatos útiles:

```
zip -r archivo.zip carpeta/
```

```
unzip archivo.zip
```

Para archivos individuales:

```
gzip archivo.txt
```

```
gunzip archivo.txt.gz
```

Supervisión del sistema

Para ver los procesos en ejecución:

```
ps aux
```

Para una vista interactiva:

```
top
```

O más visual, si tienes instalado htop:

```
htop
```

Para matar un proceso:

```
kill PID
```

Por ejemplo:

```
kill 1234
```

Para ver el uso del disco:

```
df -h
```

Para ver cuánto ocupa una carpeta:

```
du -sh carpeta/
```

Para ver el uso de la RAM:

```
free -h
```

Para saber cuánto tiempo lleva encendido el sistema:

```
uptime
```

También puedes ver el historial de comandos usados con:

```
history
```