**¿Por qué un SO es un software complejo?**

Un Sistema Operativo (SO) es complejo porque debe gestionar y coordinar múltiples tareas, recursos de hardware (CPU, memoria, almacenamiento, dispositivos), proporcionar interfaces para el usuario y desarrolladores, y garantizar la seguridad y estabilidad del sistema. También debe ser capaz de interactuar con diversos dispositivos y soportar aplicaciones concurrentemente.

**¿Qué recursos administra un SO?**

Un SO administra los siguientes recursos:

* **Procesador (CPU):** Gestión de procesos y planificación de tareas.
* **Memoria:** Asignación y liberación de memoria para procesos.
* **Dispositivos de E/S:** Control de dispositivos como discos duros, impresoras, etc.
* **Almacenamiento:** Gestión del sistema de archivos.
* **Red:** Manejo de conexiones y comunicaciones en red.

**¿Qué ventajas ofrece al usuario y a los desarrolladores un SO?**

* **Usuarios:** Facilidad de uso, multitarea, acceso a hardware sin conocimientos técnicos específicos, seguridad, y estabilidad.
* **Desarrolladores:** APIs estandarizadas, abstracción del hardware, gestión de recursos simplificada, y herramientas para desarrollo y depuración.

**¿Cuáles son las 4 libertades de un software libre?**

1. **Libertad 0:** Usar el software para cualquier propósito.
2. **Libertad 1:** Estudiar cómo funciona el software y modificarlo.
3. **Libertad 2:** Distribuir copias del software.
4. **Libertad 3:** Mejorar el software y compartir esas mejoras.

**Históricamente, ¿cómo se relacionan entre sí Multics, Unix, GNU y el kernel Linux?**

* **Multics:** Fue un proyecto pionero de SO que inspiró la creación de **Unix** después de que los desarrolladores decidieran crear un sistema más simple.
* **Unix:** Creado por Ken Thompson y Dennis Ritchie, fue la base de muchos otros SO y estableció principios que influyeron en GNU.
* **GNU:** Proyecto iniciado por Richard Stallman para crear un SO libre compatible con Unix. Sin embargo, faltaba un kernel.
* **Linux:** Kernel desarrollado por Linus Torvalds, que al combinarse con el software GNU, formó el **sistema operativo GNU/Linux**.

**¿Qué es un Shell? Nombrar al menos 2 implementaciones.**

Un **Shell** es una interfaz que permite a los usuarios interactuar con el sistema operativo mediante comandos de texto. Ejemplos de implementaciones:

1. **Bash** (Bourne Again Shell)
2. **Zsh** (Z Shell)

**Nombrar al menos 3 editores de terminal.**

1. **Nano**
2. **Vim**
3. **Emacs**

**¿Por qué es conveniente agregar el operador & al ejecutar un editor de texto GUI desde terminal?**

Agregar el operador & permite ejecutar el editor de texto en segundo plano, liberando la terminal para que puedas seguir ingresando otros comandos sin esperar a que cierres el editor.

**¿Qué hacen los comandos pwd, cat, ls, ls -l, ls -a, cd, cd ..?**

* **pwd**: Muestra el directorio de trabajo actual.
* **cat**: Muestra el contenido de un archivo.
* **ls**: Lista los archivos y directorios en el directorio actual.
* **ls -l**: Lista los archivos y directorios con detalles como permisos, tamaño y fecha de modificación.
* **ls -a**: Lista todos los archivos, incluyendo los ocultos (que comienzan con .).
* **cd**: Cambia al directorio especificado.
* **cd ..**: Cambia al directorio padre (un nivel hacia arriba en la jerarquía).

**¿Qué hacen los comandos mkdir, rm, rm -R?**

* **mkdir**: Crea un nuevo directorio.
* **rm**: Elimina un archivo.
* **rm -R**: Elimina un directorio y todo su contenido recursivamente.

**¿Cuál es la diferencia entre terminal y shell?**

* **Terminal**: Es una interfaz que permite al usuario interactuar con el Shell. Es el emulador de terminal que muestra la salida del Shell y permite la entrada de comandos.
* **Shell**: Es el programa que interpreta y ejecuta los comandos ingresados en la terminal.

**¿Cuál es la diferencia entre Bash y Sh?**

* **Bash**: Es una versión mejorada del Shell original sh, con características adicionales como historial de comandos, alias, y soporte para scripting avanzado.
* **Sh**: Es el Shell original de Unix (Bourne Shell). Es más básico y menos interactivo que Bash.

En un **Shell script**, puede ser necesario usar el comando chmod para modificar los permisos del script y permitir su ejecución.

### Ejemplo de cuándo y por qué usar chmod:

Cuando escribes un script en Shell, por defecto puede que no tenga permisos de ejecución. Esto significa que, aunque el script esté correctamente escrito, no podrás ejecutarlo directamente con ./script.sh. Para otorgarle permisos de ejecución, utilizas chmod.

**Razones para usar chmod en Shell scripting:**

1. **Permitir la ejecución**: chmod +x script.sh hace que el script sea ejecutable.
2. **Controlar quién puede ejecutar el script**: Puedes especificar qué usuarios (propietario, grupo, otros) pueden ejecutar, leer o modificar el script.
3. **Mejorar la seguridad**: Ajustando los permisos, puedes limitar quién tiene acceso a tu script, reduciendo el riesgo de que usuarios no autorizados lo ejecuten o modifiquen.

Sin chmod, podrías tener un script perfectamente funcional que no se ejecuta simplemente porque no tiene los permisos adecuados.

#!/bin/bash cd /home/Documents ls -l | grep "script" chmod u+x script.sh ./script.sh &

 Navega al directorio /home/Documents.

 Lista los archivos en ese directorio y filtra aquellos que contienen "script" en su nombre.

 Da permisos de ejecución al archivo script.sh.

 Ejecuta script.sh en segundo plano.

#!/bin/bash echo "nombre de archivo :"; read archivo; echo "buscar :"; read patron; cat $archivo | grep -n $patron >>out.txt

 Pide al usuario que introduzca el nombre de un archivo y un patrón de búsqueda.

 Busca el patrón en el archivo especificado, incluyendo los números de línea donde se encuentra el patrón.

 Añade los resultados de la búsqueda al archivo out.txt.