









Trabajo Practico N° 1

1.Características de GNU/Linux:

a. Una principal e importante característica de GNU/Linux, es que es de software libre, esto hace que posea las siguientes características:

-  Puede ser usado, copiado, modificado, estudiado y redistribuido libremente
-  Generalmente es de costo nulo.
-  Es común que se distribuya junto con su código fuente.
-  Corrección más rápida ante fallas.

b. Otros sistemas operativos como Windows o Mac Os, al ser de software propietario:

-  Generalmente tiene un costo asociado.
-  No se lo puede redistribuir libremente.
-  Generalmente no permite su modificación.
-  La corrección de fallas está a cargo del propietario(actualizaciones).

c. GNU es un acrónimo recursivo que significa “GNU No es Unix”. Se refiere a un sistema operativo similar a Unix, constituido en su totalidad por software libre.

d. Fue iniciado por Richard Stallman en 1983 con el objetivo de crear un sistema Unix completamente libre. El sistema GNU fue diseñado para ser totalmente compatible con Unix.

En 1985 Richard Stallman creo la Free Software Foundation(FSF) para proporcionar soporte logístico, legal y financiero al proyecto GNU.

Para 1990 el sistema ya contaba con programas importantes como el editor de texto EMACS. En 1992, el núcleo Linux se combinó con el sistema GNU, dando el origen al sistema operativo GNU/Linux.

e. La multitarea es una característica de los sistemas operativos que les permite ejecutar múltiples tareas o procesos aparentemente de forma simultánea. Esto significa que el sistema puede gestionar y alternar rápidamente entre diferentes programas o hilos de ejecución. GNU hace uso de la multitarea.

f. Posix significa Portable Operating System Interface, es un conjunto de estándares definidos por IEEE para garantizar la compatibilidad entre sistemas operativos tipo Unix.

La idea de Posix es que los programas escritos, siguiendo sus especificaciones puedan compilarse y ejecutarse en distintos sistemas Unix sin apenas modificaciones.

2.Distribuciones de GNU/Linux:

a. Una distribución de GNU/Linux es un conjunto de aplicaciones reunidas que facilitan la instalación de un sistema operativo basado de GNU/Linux. También se puede entender como una customización de GNU/Linux, la cual está compuesta por una versión de Kernel y un conjunto específico de programas junto con sus configuraciones.

Distribuciones como Ubuntu se centran en ser lo más amigables posible a la hora de instalarse o descargar programas. Linux Mint aprovecha el hardware potente para competir con Windows o MacOS. Para computadoras viejas hay distribuciones ligeras como Puppy Linux. Para Linux en servidores, Debian, y para jugar videojuegos, SteamOS es la mejor.

Debian: Es considerada una de las "distribuciones principales". Se caracteriza por su estabilidad y su sistema de gestión de paquetes APT.

Ubuntu: Está orientado al usuario promedio, con un fuerte enfoque en la facilidad de uso y en mejorar la experiencia del usuario. Ubuntu es uno de los más populares, estables y mejor equipados de las distribuciones Linux basadas en Debian.




Linux Mint: Linux Mint es la distribución basada en Ubuntu más popular y fácil de usar disponible en el mercado.

b. Las distribuciones se diferencian en cómo se empaquetan, configuran y que software adicional se ofrece, así como su origen y las herramientas específicas que emplean para la gestión del sistema.

c. El Proyecto Debian es una asociación de personas que han hecho causa común para crear un sistema operativo (SO) libre. Este sistema operativo que hemos creado se llama Debian. Debian lo producen cerca de un millar de desarrolladores activos, dispersos por el mundo que ayudan voluntariamente en su tiempo libre. Son pocos los desarrolladores que realmente se han encontrado en persona. La comunicación se realiza principalmente a través de correo electrónico (listas de correo en lists.debian.org) y a través de IRC (canal #debian en irc.debian.org). Debian comenzó en agosto de 1993 gracias a Ian Murdock, como una nueva distribución que se realizaría de forma abierta, en la línea del espíritu de Linux y GNU. Debian estaba pensado para ser creada de forma cuidadosa y concienzuda, y ser mantenida y soportada con el mismo cuidado. Comenzó como un grupo de pocos y fuertemente unidos hackers de Software Libre, y gradualmente creció hasta convertirse en una comunidad grande y bien organizada de desarrolladores y usuarios 4 Matías Guaymas El nombre tiene su origen en los nombres del creador de Debian, Ian Murdock, y su esposa, Debra. Debian ha tenido varios líderes desde sus comienzos en el año 1993.

3. Estructura de GNU/Linux

a. Los tres componentes fundamentales de GNU/Linux, a los que se hace referencia cuando se habla de un sistema operativo GNU, son:

-  El kernel (núcleo)
-  El Shell (intérprete de comandos)
-  El FileSystem (sistema de archivos)

b. La estructura básica del Sistema Operativo es el Kernel, que ejecuta programas y gestiona dispositivos de hardware. Sus funciones más importantes son la administración de memoria, CPU y la E/S. El Kernel en un sentido estricto, es el sistema operativo.

4. Kernel

a. Kernel es la parte fundamental del Sistema Operativo, es el encargado de que el software y el hardware de una computadora puedan trabajar juntos, ejecuta programas y gestiona los dispositivos de hardware, sus funciones más importantes son la administración de memoria, la CPU y la Entrada/Salida.

b. Actualmente el Kernel de Linux ya supera la versión 6.0. Antes de la serie de Linux 2.6.x, el número B (el segundo dígito en el formato A.B.C[.D]) indicaba el tipo de versión. Los números pares, como el 1.2 o 2.4, indicaban una versión "estable" destinada para uso de fabricación. Los números impares, en cambio, como la serie 2.5.x, son versiones de desarrollo, es decir que no son consideradas de producción.

c. Sí, es totalmente posible tener más de un kernel de GNU/Linux instalado en la misma máquina.

d. En /boot. El primer sector del disco se llama boot sector. Contiene información general de donde se almacena el Kernel y como se arranca.

5. Intérprete de comandos (Shell):

a. El intérprete de comandos conocido como Shell, es un programa fundamental en SO como GNU/Linux y Unix, que actúa como interfaz entre el usuario y el SO.

b. Su función principal es recibir comandos ingresados por el usuario, interpretarlos y convertirlos en instrucciones para el SO. Actúa como interfaz, ejecuta programas a partir del ingreso de comandos, procesa órdenes.

c. Algunos tipos de intérpretes de comandos que posee GNU/Linux



Bourne Shell (/bin/sh):

- Está disponible en todas las versiones de UNIX.
- Es lo suficientemente básico como para funcionar en todas las plataformas.
- El prompt (el indicador de que la shell espera órdenes) suele ser el carácter \$ para usuarios y # para el administrador.



C Shell (/bin/csh):

- Debe su nombre al lenguaje de programación C, lo que permite utilizar una sintaxis similar a la de C al crear scripts.
- Es el estándar en los sistemas BSD y sus derivados.



Korn Shell (/bin/ksh):

- Es el estándar de SYSV.
- Está basado en sh, pero incorpora agregados para hacerlo más amigable.
- Maneja un historial de comandos

d. Comandos propios (internos al Shell): Cuando el Shell recibe una orden, primero verifica si se trata de una de sus órdenes internas, como cd, export, return o exit. Estos comandos son reconocidos y ejecutados directamente por el Shell.

Comandos externos: Si la orden no es un comando interno ni un alias, el Shell busca el comando en las ubicaciones especificadas por la variable PATH. Los comandos externos pueden encontrarse en directorios como:

- /bin
- /usr/bin
- /usr/local/bin

La diferencia fundamental es que los internos están incorporados a la consola y se pueden ejecutar directamente, mientras que para los externos hay que indicar la ruta hasta la ubicación del comando.

En bash: para saber si un comando es externo o interno usar el comando interno type: \$ type cd.

e. La shell no forma parte del kernel básico del sistema operativo; en cambio, la misma "dialoga" con el kernel. Esto implica que son entidades separadas que interactúan entre sí, en lugar de ser componentes intrínsecos uno del otro.

f. Si es posible definir un intérprete de comandos distintos para cada usuario ya que la shell es iniciada por un proceso denominado "login" en donde cada usuario tiene asignado una shell por defecto que se puede personalizar.

6. Intérprete de comandos (Shell):

a. El Sistema de Archivos (File System) es la manera en que un sistema de cómputo organiza y administra los archivos. Constituye una parte fundamental de cómo se estructura la información dentro de un sistema operativo.

b. La estructura de directorios en GNU/Linux es generalmente jerárquica (en árbol), y donde el directorio principal (raíz) es el directorio "/", del que cuelga toda la estructura del sistema.

- /boot: Aquí se encuentran los cargadores de inicio y los archivos de inicio del sistema.
- /bin: Contiene la mayoría de los programas ejecutables esenciales del sistema, como cp, ls y mv.
- /dev: Contiene los controladores de dispositivo (device drivers) que se utilizan para acceder a los dispositivos del sistema y recursos, como discos duros y memoria.
- /etc: Almacena una serie de archivos de configuración del sistema, incluyendo /etc/passwd (base de datos de usuarios) y /etc/fstab (scripts de inicialización del sistema).
- /home: Contiene los directorios personales de los usuarios, por ejemplo, /home/ISO_CS0 para el usuario ISO_CS0.
- /lib: Aquí se guardan las librerías de los paquetes instalados ya que estas librerías son compartidas por todos los paquetes.
- /root: Es el directorio home del usuario root (administrador del sistema).
- /sbin: Se utiliza para almacenar programas ejecutables esenciales del sistema que son para uso del administrador del sistema
- /tmp: Un directorio para archivos temporales generados por programas
- /usr: Es un directorio muy importante que contiene programas y archivos de configuración útiles, muchos de los cuales son opcionales pero hacen que el sistema sea más funcional

La sigla FHS hace referencia al Filesystem Hierarchy Standard. Este estándar define los directorios principales y sus contenidos en el sistema operativo GNU/Linux y otros sistemas de la familia Unix.

c. GNU/Linux soporta una variedad de sistemas de archivos, incluyendo aquellos que son nativos de otros sistemas operativos. Los principales sistemas de archivos soportados por GNU/Linux, según las fuentes, son:

- Familia Extended (Ext)
- Btrfs (B-tree FS)
- ReiserFS
- XFS
- FAT (File Allocation Table):
- NTFS (New Technology File System)

d. Sí, es posible visualizar particiones del tipo FAT y NTFS en GNU/Linux.

GNU/Linux organiza la forma en que se almacenan los archivos en dispositivos de almacenamiento y que soporta sistemas de archivos como FAT y NTFS.

7. Particiones:

a. Las particiones son una forma de dividir lógicamente un disco físico. Esta división permite organizar el almacenamiento de datos y es fundamental para la instalación de sistemas operativos.

Hay distintos tipos de particiones, los tipos de particiones se refieren a las diferentes maneras en que un disco físico puede ser dividido lógicamente para organizar el almacenamiento de datos. Principalmente, se distinguen tres tipos: particiones primarias, particiones extendidas y particiones lógicas

Partición Primaria: Es una división “cruda” del disco. En un esquema MBR, se pueden tener hasta cuatro particiones por disco. Solo las particiones primarias pueden ser marcadas como activas o “booteables”.

Partición Extendida: Una de las cuatro particiones posibles en el esquema MBR puede ser designada como partición extendida. Su propósito principal es contener unidades lógicas en su interior.

Partición Lógica: Estas particiones residen dentro de una partición extendida. A las particiones lógicas se les define un tipo de sistema de archivos específico.

Ventajas de las Particiones:

- 🐧 Instalación de Múltiples Sistemas Operativos
- 🐧 Separación de Datos y Sistema
- 🐧 Partición de Recuperación (Restore)
- 🐧 Seguridad y Disponibilidad del Kernel

Desventajas de las Particiones:

- 🐧 Exceso de Particionamiento

b. En GNU/Linux, la identificación de las particiones en los discos se ha basado tradicionalmente en el tipo de interfaz del disco y su ubicación física, aunque ha evolucionado con el tiempo.






Discos IDE(antiguos): Los discos IDE se nombran como /dev/hdX, las particiones se numeran hda1, hda2, ... -> particiones primarias/lógicas del primer disco

Discos SCSI y SATA (y también USB, modernos): En Linux, SCSI y SATA se identifican igual con /dev/sdX, las particiones se numeran sda1, sda2, ... -> particiones en el primer disco SATA/SCSI.

Discos NVMe(mas nuevos, SSD PCIe): Los discos NVMe se nombran distinto /dev/nvmeXnTpZ.

c. Como mínimo es necesario una partición (para el /), esta se utiliza para el directorio raíz. Es recomendable crear al menos 2 (/ y SWAP), swap es el área de intercambio. Usualmente se suelen tener tres, una para el sistema/programas (/), otra para los datos (/home) y otra para swap.

d. Dependiendo del tipo de tarea , se usan distintos esquemas de Particionamiento:

-  **Particionamiento Fijo**: La memoria se divide en bloques del mismo tamaño.
Ej: procesos batch en mainframes antiguos.
-  **Particionamiento Variable**: La memoria se divide según el tamaño del proceso.
Ej: Servidores web con cargas variables.
-  **Particionamiento por Paginación**: La memoria se divide en páginas del mismo tamaño. Ej: PC con Windows/Linux ejecutando varios programas.
-  **Particionamiento por Segmentación**: La memoria se organiza en segmentos lógicos (código, datos, pila). Ej: compiladores o programas modulares.
-  **Particionamiento Mixto**: Combina la Segmentación + la Paginación.
Ej: Sistemas Operativos modernos(Linux, Windows).

e. Existen 2 tipos:

- Destructivos**: permiten crear y eliminar particiones (fdisk)
- **No destructivo**: permiten crear, eliminar y modificar particiones (fips, gparted) ← generalmente las distribuciones permiten hacerlo desde la interfaz de instalación

partman :Herramienta original de Linux para particionar discos.

fdisk: Es la herramienta original de Linux para particionar discos, buena para expertos.

cdisk: Una herramienta para particionar a pantalla completa, muy fácil de usar.

8. Arranque (bootstrap) de un Sistema Operativo:

a. El BIOS (Basic Input Output System) es un software de bajo nivel que se encuentra en la placa base (motherboard) de las computadoras IBM PC y sus "clones". Su función principal es la de iniciar los componentes de hardware y lanzar el sistema operativo de un ordenador cuando lo encendemos (a través del MBC). También carga las funciones de gestión de energía y temperatura del ordenador. Cuando enciendes tu ordenador lo primero que se carga en él es el BIOS. Este firmware entonces se encarga de iniciar, configurar y comprobar que se encuentre en buen estado el hardware del ordenador, incluyendo la memoria RAM, los discos duros, la placa base o la tarjeta gráfica. Cuando termina selecciona el dispositivo de arranque (disco duro, CD, USB etcétera) y procede a iniciar el sistema operativo, y le cede a él el control de tu ordenador.

b. UEFI (Unified Extensible Firmware Interface) es una especificación que surgió de la evolución de EFI (Extensible Firmware Interface), la cual fue desarrollada originalmente por Intel. Es un nexo entre el sistema operativo y el firmware. Puede verse como una alternativa para reemplazar la BIOS.

Un firmware es un software que maneja físicamente al hardware.

La función principal del UEFI es inicializar el hardware del sistema cuando la computadora se enciende y cargar el sistema operativo.

c. El MBR (Master Boot Record) es un componente fundamental en el proceso de arranque de las computadoras, especialmente en sistemas basados en BIOS. El MBR es el primer sector del disco de inicio que lee BIOS cuando arranca la computadora.

El MBC (Master Boot Code) es un pequeño código de arranque que se encuentra en el MBR, este permite arrancar el Sistema Operativo. La última acción del BIOS es leer el MBC, lo lleva a memoria y lo ejecuta.

d. Las siglas GPT hacen referencia a GUID Partition Table (Tabla de Particiones GUID). En el contexto de la especificación EFI, se lo considera una alternativa o reemplazo del MBR tal como se concebía en el BIOS.

e. La funcionalidad de un Gestor de Arranque (Bootloader) es cargar una imagen Kernel (Sistema Operativo) de alguna partición para su ejecución. Es un pequeño programa diseñado específicamente para arrancar un sistema operativo.

Tipos de Gestores de Arranque:

Tipos de Gestores de Arranque

1. De primera etapa (Stage 1)
Muy pequeños, se almacenan en el MBR (en BIOS) o en la partición de arranque (en UEFI). Su función es cargar un gestor más completo.
2. De segunda etapa (Stage 2)
Más avanzados, con menús y soporte para varios sistemas operativos. Permiten elegir configuraciones y opciones de arranque.

3. Monocargadores
Cargan directamente un único sistema operativo.
4. Multicargadores
Permiten arrancar varios sistemas operativos desde un menú (Windows, Linux, etc).

Ejemplos de gestores de arranque conocidos

- GRUB (Grand Unified Bootloader)
- LILO (Linux Loader)
- Windows Boot Manager (bootmgr)

f. El proceso de arranque o bootstrap es la secuencia de pasos que realiza una computadora desde que se enciende hasta que el Sistema Operativo está listo para usarse.

Pasos del bootstrap

1. Enciendo: La energía llega a los componentes.
2. BIOS/UEFI: Hace el POST, inicializa el hardware y busca el dispositivo de arranque.
3. Gestor de arranque (Bootloader): Se carga desde el MBR o la partición EFI.
4. Bootloader: Selecciona y carga el kernel del sistema operativo en memoria.
5. Kernel: inicializa drivers, servicios y monta el sistema de archivos.
6. Sistema Operativo: Arranca procesos base e interfaz de usuario.

g. El proceso de arranque en GNU/Linux es una compleja orquestación de hardware, firmware y software, que ha evolucionado significativamente con tecnologías como UEFI y SystemD para ofrecer mayor seguridad, flexibilidad y eficiencia.

Pasos del proceso de arranque en GNU/Linux:

1.Inicio del Hardware y Firmware:

Cuando se enciende la computadora, se ejecuta el BIOS en sistemas antiguos o el UEFI en sistemas modernos. El BIOS realiza un POS (Power-on-self-test), que incluye rutinas para fijar valores de señales internas de componentes esenciales como la RAM y el Teclado.

2.Carga del Gestor de arranque (Bootloader):

La forma en que se carga el gestor de arranque difiere entre sistemas BIOS y UEFI:

En sistemas basados en BIOS:

Después del POST, el BIOS lee el primer sector del disco de inicio, el MBR (Master Boot Record), el MBR se carga en memoria y se ejecuta.

En sistemas basados en UEFI:

UEFI moderniza el proceso de arranque y rompe las limitaciones de uso de memoria y disco del BIOS. Utiliza el sistema GPT para el particionado de discos. El firmware de UEFI busca una partición identificada por un código GUID específico en discos GPT o una partición 0xEF en discos MBR.

3.Carga e inicialización del Sistema Operativo:

Una vez que el gestor de arranque ha tomado el control, este carga el Kernel del sistema operativo y el initrd.

4.Proceso de inicialización del Sistema:

El proceso de init puede ser gestionado por diferentes sistemas de inicialización como SysV init, SystemD.

h. El proceso de parada (shutdown) en GNU/Linux, implica una secuencia de pasos para detener el sistema de manera segura.

1. Inicio del comando de parada: El proceso de parada puede ser iniciado por un usuario (generalmente el administrador root) utilizando comandos específicos. Por ejemplo, `init 0` o `shutdown -t1 -h now` son comandos para apagar la máquina.

2. Transición al Runlevel 0: En sistemas basados en SysV init, el comando `init 0` instruye al sistema para que cambie al runlevel 0. El runlevel 0 está explícitamente definido como el modo de "halt" o "parada/apagado" del sistema.

3. Ejecución de scripts de detención: Al entrar en el runlevel 0, el sistema ejecuta una serie de scripts diseñados para detener los servicios y procesos en ejecución. Estos scripts suelen almacenarse en `/etc/init.d` y se vinculan simbólicamente en directorios específicos para cada runlevel, como `/etc/rc0.d` para el runlevel 0. Los nombres de estos enlaces simbólicos en `/etc/rcX.d` a menudo comienzan con K (de "kill" o "detener"), seguidos de un número de orden y el nombre del script (ej. `K<orden><nombre>`).

4. Parada del sistema: Una vez que todos los servicios necesarios han sido detenidos y los filesystems desmontados de forma segura, el sistema procede a apagar completamente la máquina.

i. Si, ya que un disco rígido puede particionarse y en cada partición puede tener un sistema de archivos distinto (partición primaria), sería como tener varios discos distintos, uno con cada Sistema Operativo, por lo tanto, se necesitará un gestor de arranque como los ya descritos anteriormente.

9. Archivos y Editores:

a. En GNU/Linux, los archivos se identifican de varias maneras clave, principalmente a través de su ruta(path) y su nombre, y se rigen por ciertas características del sistema de archivos.

🐧 Ruta (Path): La ubicación precisa de un archivo se declara mediante una cadena de texto llamada "ruta" o "path". Una ruta se forma por una sucesión de nombres de directorios y subdirectorios, ordenados jerárquicamente de izquierda a derecha.

🐧 Separador de directorios: Los directorios y subdirectorios en una ruta se separan utilizando el carácter de barra '/ '.

🐧 Nombres de archivos: Los nombres de archivos son simplemente cadenas de texto.

- 🐧 Sensibilidad a mayúsculas y minúsculas(Case Sensivite): significa que distingue entre mayúsculas y minúsculas en los nombres de archivos y directorios.

b. Los editores vim, nano y mcedit son editores de texto en modo consola muy usados en GNU/Linux.

🐧 VIM

Es un editor modal que deriva del editor clásico VI de Unix(1976), esta diseñado para ser eficiente con el teclado, sin depender del mouse. Que sea modal significa que tiene varios modos de operación. Los principales son:

- Normal: En este modo no se escribe texto, sino que se navega, se borra, se copia, se pega, etc. Ejemplos:

dd -> borra una línea

yy -> copia una línea

p -> pega

- Inserción: En este modo si se puede escribir texto como en cualquier editor normal, se accede con i, a, o, etc. Para volver al modo normal se presiona Esc.

- Comando (línea de comandos): A este modo se accede con : desde el modo normal, en este modo se puede dar órdenes:

:w -> guardar

:q -> salir

:wq -> guardar y salir

:q! -> salir sin guardar

- Las ventajas que tiene este editor es que es muy rápido una vez aprendido, es extensible con plugins y reconoce cientos de lenguajes con resaltado de sintaxis.

Las desventajas es que es difícil de aprender y es poco intuitivo al inicio.

🐧 NANO

Este editor está pensado para ser fácil, directo y usable sin manuales.

Al abrir un archivo con nano archivo.txt, ya estás en modo escritura. En nano no hay modos es todo inmediato.

En la parte inferior se muestran los atajos de teclado mas usados ^ significa Ctrl.

Ctrl + O -> Guardar archivo

Ctrl + X -> Salir

Ctrl + W -> K

Ctrl + K -> Cortar línea

Ctrl + U -> Pegar linea

- Nano es muy fácil para principiantes, todo lo necesario visible está en pantalla, es ideal para ediciones rapidas, pero es limitado en comparación con Vim y es más eficiente para usuarios más avanzados que editan mucho código.



MCEDIT

Este editor forma parte de Midnight Commander (MC), un administrador de archivos de consola estilo Norton Commander.

Se puede usar independientemente como editor `mcedit archivo.txt`.

Tiene una interfaz semi-gráfica en consola, con menús accesibles por teclas de función (F1-F10). Es más “amigable” que Vim porque los menús son visibles, pero más completo que Nano.

Este ofrece:

- Resultados de sintaxis.
- Barra de menú superior.
- Búsqueda y reemplazo.
- Soporte para múltiples archivos.

Ejemplos de atajos:

F2 -> Guardar

F10 -> Salir

F7 -> Buscar texto

F4 -> Reemplazar

- `mcedit` tiene una interfaz más visual, soporta colores, sintaxis y menús claros. Es menos “minimalista” que `nano` y no es tan rápido ni extensible como Vim.

Los comandos `cat`, `more` y `less` en GNU/Linux se usan para visualizar el contenido de archivos de texto en la terminal, pero cada uno funciona de forma distinta.



Cat: su nombre viene de concatenate, se usa principalmente para mostrar el contenido completo de un archivo en la salida estándar o concatenar varios archivos.

`cat archivo.txt` -> muestra todos los archivos de golpe.

También se pueden unir archivos:

`cat archivo1.txt archivo2.txt > combinado1y2.txt`

Se usa en tuberías:

`cat archivo.txt | grep “palabra”`

esto muestra solo las líneas que contienen “palabra”



more: Es un paginador básico, sirve para mostrar archivos página por página, permitiendo avanzar hacia adelante.

`more archivo.txt` -> muestra una parte de un archivo en pantalla.

Espacio -> avanza una página.

Tecla -> avanza una línea.

q -> salir.

Solo permite avanzar, no retroceder.

lees: Es un paginador mejorado que supera a more, permite navegar hacia adelante y hacia atrás dentro de un archivo, sin cargarlo completo en memoria.

lees archivo.txt -> abre el archivo en un visor interactivo en la terminal.

Espacio -> página siguiente.

b -> página anterior.

↑ / ↓ -> moverte línea por línea.

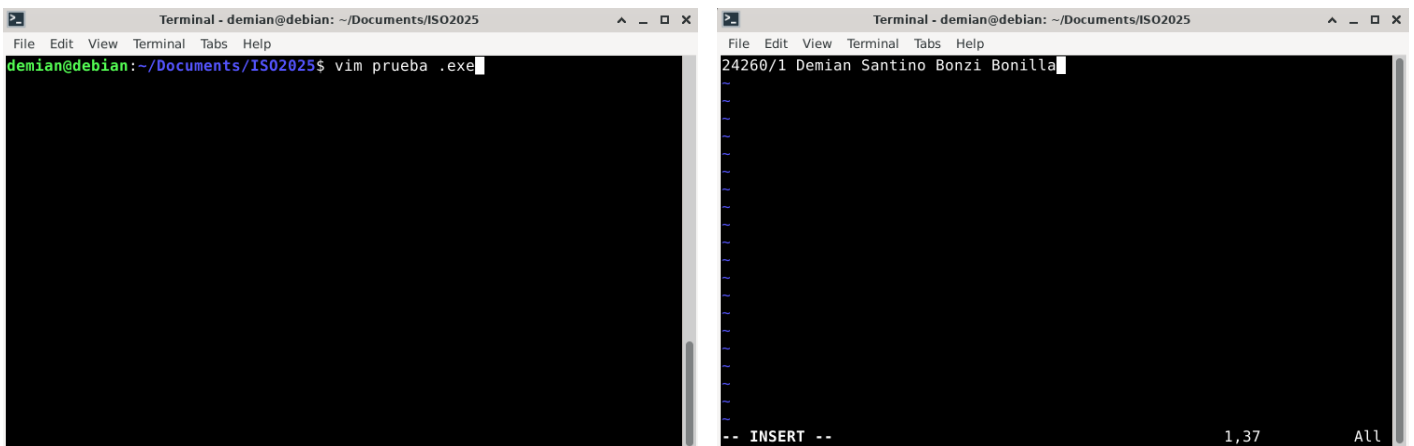
/palabra -> buscar hacia adelante.

?palabra -> buscar hacia atrás.

n / N -> siguiente o anterior coincidencia.

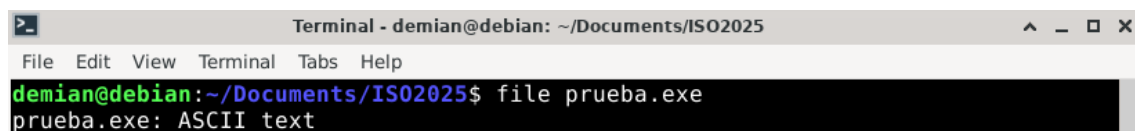
q -> salir.

c.



d. El comando file en GNU/Linux sirve para identificar el tipo de contenido de un archivo, sin importar su extensión, ya que la extensión (.exe, .txt, etc) en Linux no determina nada.

file analiza los primeros bytes del archivo y con eso deduce que tipo de archivo es.



Como es este caso el archivo que cree anteriormente llamado "prueba.exe" es en realidad un archivo de texto.

e. • cd (change directory): Cambiar de directorio.

• mkdir (make directory): Crear directorios, crea directorios anidados si no existen.

• rmdir (remove directory): Eliminar directorios vacíos.

• ln (link): Crear enlaces a archivos..

• tail (cola): Mostrar las últimas líneas de un archivo.

• locate (localizar): Buscar archivos rápidamente mediante una base de datos.

- ls (list): Listar archivos y directorios. Parametros útiles:
 ls -l: formato largo(permisos, dueño, tamaño, fecha).
 ls -a: incluye archivos ocultos.
 ls -h:tamaños legibles (K, M, G).
 ls -R: listado recursivo.
- pwd (print working directory): Mostrar la ruta absoluta del directorio actual.
- cp (copy): Copiar archivos o directorios.
- mv (move): Mover o renombrar archivos y directorios.
- find (encontrar): Buscar archivos y ejecutar acciones sobre ellos.

10. Comandos I:

a. Crear la carpeta ISOCSO (mkdir)

```
demian@debian:~/Documents/ISO2025$ mkdir ISOCSO
```

b. Acceder a la carpeta (cd)

```
demian@debian:~/Documents/ISO2025$ cd ISOCSO
demian@debian:~/Documents/ISO2025/ISOCSO$
```

c. Crear dos archivos con nombres isocso.txt e isocso.csv (touch)

```
demian@debian:~/Documents/ISO2025/ISOCSO$ touch isocso.txt
demian@debian:~/Documents/ISO2025/ISOCSO$ touch isocso.csv
```

d. Listar el contenido del directorio actual (ls)

```
demian@debian:~/Documents/ISO2025/ISOCSO$ ls
isocso.csv  isocso.txt
demian@debian:~/Documents/ISO2025/ISOCSO$ ls -l
total 0
-rw-rw-r-- 1 demian demian 0 Sep  7 22:35 isocso.csv
-rw-rw-r-- 1 demian demian 0 Sep  7 22:35 isocso.txt
```

e. Visualizar la ruta donde estoy parado(pwd)

```
demian@debian:~/Documents/ISO2025/ISOCSO$ pwd
/home/demian/Documents/ISO2025/ISOCSO
```

f. Buscar todos los archivos con los nombres que contienen "iso" (find)

```
demian@debian:~/Documents/ISO2025/ISOCSO$ find iso*
isocso.csv
isocso.txt
```

g. Informar la cantidad de espacio libre en el disco (df)

```
demian@debian:~/Documents/ISO2025/ISOCSO$ df
Filesystem      1K-blocks    Used Available Use% Mounted on
udev             974248         0   974248    0% /dev
tmpfs            202104         964   201140    1% /run
/dev/sda3       5151720 4028920   840332   83% /
tmpfs           1010520         0  1010520    0% /dev/shm
tmpfs             5120          8     5112    1% /run/lock
tmpfs            1024          0     1024    0% /run/credentials/systemd-journal
d.service
tmpfs           1010520          8  1010512    1% /tmp
/dev/sda5       941740  169208   707356   20% /boot
tmpfs            1024          0     1024    0% /run/credentials/getty@tty1.serv
ice
tmpfs           202104          84   202020    1% /run/user/1000
```

h. Verificar los usuarios conectados al sistema (who)

```
demian@debian:~/Documents/ISO2025/ISOCS0$ who
demian    seat0          2025-09-07 21:25 (:0)
```

i. Editar el archivo isocso.txt e ingresar nombre y apellido (vim)

```
demian@debian:~/Documents/ISO2025/ISOCS0$ vim isocso.txt
```

j. Mostrar en pantalla las últimas líneas de un archivo (tail)

```
demian@debian:~/Documents/ISO2025/ISOCS0$ tail isocso.txt
Demian Bonzi
```

11. Comandos II:

- man (manual): Muestra el manual de ayuda de un comando.

- Parametros útiles:

- man 1 ls -> manual sección 1 (comandos de usuario).

- man -k palabra -> buscar palabra clave .

- Ubicación: /usr/bin/man

- shutdown: Apagar o reiniciar el sistema de manera segura.

- Parametros útiles:

- shutdown -h now -> apagar inmediatamente.

- shutdown -r now -> reiniciar inmediatamente.

- shutdown +10 -> apagar en 10 minutos.

- Ubicación: /usr/sbin/shutdown

- reboot: Reinicia el sistema.

- Ubicación: /usr/sbin/reboot

- halt: Detiene el sistema.

- Parametros útiles:

- halt -p -> detener y apagar el sistema.

- Ubicación: /usr/sbin/halt

- uname (Unix name): Muestra información del sistema operativo y kernel.

- Parametros útiles:

- uname -a -> todo.

- uname -r -> versión del kernel.

- uname -m -> arquitectura .

- Ubicación: /usr/bin/uname

- dmesg (diagnostic message): Muestra mensajes del kernel ring buffer.

- Parametros útiles:

- dmesg | less -> ver con paginador.

- dmesg -k -> solo mensajes del kernel.

- dmesg -T -> mostrar hora legible.

- Ubicación: /usr/bin/dmesg

- lsblk (list block devices): Lista dispositivos de almacenamiento (discos, particiones).

- Parametros útiles:

- lsblk -f -> muestra sistema de archivos.

- Ubicación: /usr/bin/lsblk

- **at:** Programa tareas para ejecutarse una sola vez en un momento específico.

Parametros útiles:

at -l -> lista tareas pendientes.

at -c N -> muestra tarea numero N.

Ubicación: /usr/bin/at

- **netstat:** Muestra conexiones red, tablas de enrutamiento, estadísticas.

Parametros útiles:

netstat -tuln -> puertos abiertos.

netstat -anp -> conexiones activas con PID.

Ubicación: /usr/bin/netstat

- **head:** Muestra las primeras líneas de un archivo.

Parametros útiles:

head archivo.txt -> primeras 10 líneas.

head archivo -n 20 archivo.txt -> primeras 20 líneas.

Ubicación: /usr/bin/head

12. Procesos:

a. Un proceso es una instancia de un programa en ejecución. Es una parte esencial de cualquier sistema de cómputo, y el sistema operativo se encarga de gestionar su correcto funcionamiento.

En el contexto de un proceso en GNU/Linux, las siglas PID y PPID hacen referencia a lo siguiente:

- **PID (Process ID):** Es el identificador único de un proceso. Todos los procesos tienen este atributo en GNU/Linux.

- **PPID (Parent Process ID):** Es el identificador del proceso padre. Este atributo se utiliza para los procesos hijos, indicando quién los originó

No todos los procesos tienen ambos atributos (PID y PPID) en GNU/Linux. Todos los procesos en GNU/Linux se identifican con un PID (Process ID), que es un identificador único. Sin embargo, no todos los procesos tienen un PPID (Parent Process ID). El ejemplo clave es el proceso init (o systemd en sistemas modernos), que es el primer proceso en ejecutarse.

Este proceso tiene el PID 1, pero no tiene padre este es el padre de todos los demás procesos del sistema.

Además del PID (Process ID) y el PPID (Parent Process ID), un proceso en GNU/Linux tiene otros atributos importantes que el sistema operativo gestiona para su correcto funcionamiento y planificación:

- **Prioridad:** Cada proceso tiene un valor que representa su prioridad, donde un valor menor generalmente indica una mayor prioridad. Esta prioridad influye en qué proceso es seleccionado para ejecutarse en la CPU.

- **Tiempos Asociados:** Los procesos tienen varios tiempos que son cruciales para su administración y evaluación de rendimiento:
 - **Tiempo de CPU (TCPU):** Es el tiempo que el proceso utiliza efectivamente la CPU.
 - **Tiempo de Retorno (TR):** Es el tiempo total que transcurre desde que el proceso llega al sistema hasta que completa su ejecución.
 - **Tiempo de Espera (TE):** Es el tiempo que el proceso permanece en el sistema esperando, es decir, el tiempo que pasa sin ejecutarse ($TR - TCPU$).
 - **Tiempos Promedio de Retorno (TPR) y Espera (TPE):** Se refieren a los promedios calculados de los tiempos individuales de cada proceso en un lote.

- **Modo de Ejecución:** Un proceso puede ejecutarse en Background o en Foreground, lo que determina cómo interactúa con el usuario y la terminal.

- **Contador (en Round Robin):** En algoritmos de planificación como Round Robin, un proceso puede tener un "contador" que indica las unidades de CPU que ha utilizado. Este contador puede ser global o local (en el PCB - Process Control Block)

b. • top: Muestra en tiempo real los procesos activos, consumo de CPU, RAM, etc.

-Parámetros útiles:

top -u usuario -> solo procesos de un usuario

dentro de top:

k -> matar proceso.

q -> salir.

-Ubicación: /usr/bin/top

- **htop:** Similar a top pero con interfaz interactiva y colores.

-Parametros utiles:

htop -u usuario -> filtrar por usuario.

F2 (Setup), F9 (Kill), F10 (Salir).

Ubicación: /usr/bin/htop

- **ps (process status):** Muestra procesos en ejecución.

-Parametros utiles:

ps -> procesos actuales de la terminal.

ps -e -> todos los procesos.

ps -ef -> formato extendido.

ps aux -> con usuario, memoria y CPU.

-Ubicación: /usr/bin/ps

- **pstree:** Muestra los procesos en forma de árbol.

-Parametros utiles:

pstree -> ver árbol completo.

pstree -p -> incluir PID.

pstree -u -> incluir usuarios.

-Ubicación: /usr/bin/pstree

- **kill:** Envía señales a un proceso (normalmente para detenerlo).

-Parametros utiles:

kill -9 PID -> matar forzosamente.

kill -15 PID -> terminar de forma elegante (default).

-Ubicación: /bin/kill

- **pgrep**: Busca procesos por nombre y devuelve su PID.
 - Parametros utiles:
 - pgrep Firefox -> PID de Firefox.
 - pgrep -u usuario -> procesos de un usuario.
 - Ubicación: /usr/bin/pgrep.
- **pkill**: Mata procesos por un nombre en lugar de PID.
 - Parametros utiles:
 - pkill Firefox
 - pkill -u usuario
 - Ubicación: /usr/bin/pkill
- **killall**: Mata todos los procesos con un nombre especifico.
 - Parametros utiles:
 - killall Firefox
 - Ubicación: /usr/bin/killall
- **renice**: Cambia la prioridad de un proceso ya en ejecución.
 - Parametros utiles:
 - renice +5 -p 1234 -> PID 1234 menos prioridad
 - renice -10 -p 1234 -> PID 1234 más prioridad
 - Ubicación: /usr/bin/renice
- **xkill**: Permite cerrar ventanas graficas haciendo clic sobre ellas
 - Ubicación /usr/bin/xkill
- **atop**: Monitor de procesos y recursos, mas completo que top
 - Parametros utiles:
 - atop -> modo interactivo.
 - atop -r archivo -> leer registros guardados.
 - Ubicación: /usr/bin/atop
- **nice**: Ejecutar un proceso con una prioridad especifica.
 - nice -n 10 comando -> menor prioridad
 - nice -n -5 comando -> mayor prioridad
 - Ubicación: /usr/bin/nice

13. Proceso de Arranque SystemV

a. El proceso de inicio de un sistema GNU/Linux, desde que se enciende la PC hasta que se obtiene la pantalla de inicio de sesión, sigue una serie de pasos secuenciales:

🔔 1. Inicio del BIOS/UEFI y POST:

- Al encender la computadora, se ejecuta el BIOS (Basic Input Output System), un software de bajo nivel ubicado en la placa base.
- El BIOS realiza el POST (Power-on self-test), que incluye rutinas para fijar valores de señales internas y ejecutar pruebas de componentes como la RAM y el teclado.
- Luego, el BIOS lee el primer sector del disco de inicio (seleccionado de entre los posibles dispositivos de arranque), conocido como MBR (Master Boot Record). Este MBR se carga en memoria y se ejecuta.

- Alternativamente, en sistemas modernos, el UEFI (Unified Extensible Firmware Interface) es una especificación que busca modernizar el proceso de arranque. UEFI aporta criptografía, autenticación por red y una interfaz gráfica.
- El proceso de arranque de UEFI busca una partición identificada por un código GUID específico o una partición con tipo 0xEF marcada como de arranque (conocida como "EFI System Partition", que debe tener formato UEFI filesystem, como FAT12, FAT16 o FAT32). Si se encuentra, el firmware accede a su sistema de archivos y carga el Bootloader por defecto.

2. Carga y Ejecución del Gestor de Arranque (Bootloader):

- El MBR puede contener un código de arranque llamado MBC (Master Boot Code).
- La última acción del BIOS es leer el MBC, cargarlo en memoria y ejecutarlo.
- Un gestor de arranque es un programa pequeño que se encarga de arrancar un sistema operativo. Habitualmente, se instala en el MBR y asume el rol de MBC.
- Su finalidad es cargar una imagen del Kernel (núcleo del sistema operativo) desde alguna partición para su ejecución.
- Ejemplos de gestores de arranque incluyen GRUB, LILO, NTLDR, GAG o YaST.
- En sistemas BIOS, bootloaders como GRUB Legacy pueden usar los 446 bytes del MBR y luego sectores adyacentes del disco (conocido como "MBR Gap") para cargar sus fases posteriores.
- En el entorno UEFI, el bootloader es una aplicación UEFI especial que se ejecuta en "modo protegido" o "long mode", sin las restricciones de tamaño del BIOS antiguo. Puede cargar el sistema deseado utilizando los servicios de acceso a archivos provistos por el firmware UEFI.

3. Carga e Inicialización del Kernel:

- Una vez ejecutado, el gestor de arranque carga el Kernel (núcleo del sistema operativo).
- El Kernel realiza pruebas y pone a disposición los dispositivos del sistema.
- También carga el initrd (initial ram disk), que se monta como sistema de archivos raíz, inicializando componentes esenciales como el planificador.
- Posteriormente, el Kernel ejecuta el proceso init.

Proceso init (SysV init o SystemD) y Runlevels/Targets:

- El proceso init (ejecutado desde /sbin/init) es el responsable de cargar todos los subprocesos necesarios para el correcto funcionamiento del sistema operativo.
- Tiene el PID 1 (Process ID 1) y es el padre de todos los demás procesos del sistema.
- También se encarga de montar los sistemas de archivos y de hacer disponibles los demás dispositivos.
- En sistemas que utilizan SysV init:
 - El proceso init lee el archivo /etc/inittab.
 - Se ejecutan los scripts correspondientes al "runlevel 1".
 - Luego, el sistema pasa al "runlevel" por defecto, donde se ejecutan los scripts asociados a dicho nivel.
 - Los runlevels, que son modos de arranque de GNU/Linux, están numerados del 0 al 6: 0 (apagado), 1 (monousuario), 2 (multiusuario sin red), 3 (multiusuario en consola), 4 (no utilizado), 5 (multiusuario con entorno gráfico X11), y 6 (reinicio).
 - Los scripts de inicio/parada se guardan en /etc/init.d y se crean enlaces simbólicos en directorios como /etc/rcX.d (donde X es el número de runlevel).
- En sistemas que utilizan SystemD:
 - SystemD reemplaza el proceso init y también tiene el PID 1.

- En lugar de runlevels, SystemD utiliza "targets" para agrupar unidades y establecer puntos de sincronización durante el arranque.
- No utiliza el archivo `/etc/inittab` para su configuración.
- Los servicios y otras funcionalidades se gestionan a través de "units" (unidades de trabajo), que pueden ser de tipo service, socket, target o snapshot.
- El comando `systemctl` se utiliza para controlar SystemD.

🔔 5. Sistema Listo para Uso:

- Una vez que se han ejecutado los scripts de inicio del runlevel (en SysV init) o se han procesado los targets (en SystemD), el sistema está completamente iniciado y listo para que el usuario inicie sesión, presentando el prompt de login.

b. El proceso `init` es un componente fundamental en el arranque de un sistema GNU/Linux, y sus funciones son cruciales para el funcionamiento del sistema operativo.

El proceso `init` es ejecutado por el Kernel (núcleo) del sistema operativo. Después de que el gestor de arranque carga el Kernel y este realiza las pruebas iniciales y hace disponibles los dispositivos, el control es transferido al proceso `init`.

El objetivo principal del proceso `init` es cargar todos los subprocesos necesarios para el correcto funcionamiento del Sistema Operativo.

c. Los Runlevels son modos de ejecución en los que puede arrancar un sistema GNU/Linux. En el contexto del proceso de arranque SysV `init`, el sistema se divide en diferentes niveles de operación

Su objetivo principal es gestionar la inicialización o detención de una serie de servicios al entrar o salir de cada nivel. Esto permite que el sistema operativo funcione en distintos estados, cada uno con un conjunto específico de servicios activos.

d. Según el estándar, existen siete runlevels, numerados del 0 al 6, y cada uno tiene una referencia específica para el estado del sistema

- 0: Indica la parada o apagado del sistema (`halt`).
- 1: Representa el modo monousuario (`single-user mode`).
- 2: Corresponde al modo multiusuario sin soporte de red (`multi-user without network support`).
- 3: Se refiere al modo multiusuario en consola (`multi-user console mode`). Por defecto, Red Hat utiliza este runlevel.
- 4: No se utiliza (N/A).
- 5: Es el modo multiusuario con entorno gráfico basado en X11 (X11).
- 6: Significa el reinicio del sistema (`reboot`)

En los sistemas que utilizan SysV `init`, la configuración de los runlevels se encuentra definida en el archivo `/etc/inittab`.

No todas las distribuciones actuales respetan el uso de SysV `init` y sus runlevels de la misma manera. Algunas distribuciones pueden tener diferentes runlevels por defecto (por ejemplo, Runlevel 3 en Red Hat y Runlevel 2 en Debian), pero aun así se adhieren a las definiciones generales de cada nivel.

e. El archivo `/etc/inittab` es fundamental en los sistemas GNU/Linux que utilizan el sistema de inicialización SysV init.

La finalidad principal del archivo `/etc/inittab` es configurar el proceso init.

El archivo `/etc/inittab` almacena información sobre los runlevels y las acciones asociadas a cada uno. Esto incluye:

- El runlevel por defecto que el sistema debe adoptar al arrancar.
- Scripts de inicialización del sistema que deben ejecutarse en momentos específicos del arranque.
- Acciones a tomar ante eventos específicos, como la combinación de teclas Ctrl+Alt+Del
- Comandos exactos que se ejecutarán para una acción y runlevel dados.

Cada entrada en `/etc/inittab` sigue una estructura específica:

- id: Un identificador único para la entrada en inittab, que puede tener de 1 a 4 caracteres.
- runlevels: Indica el/los runlevels en los que se debe realizar la acción. Puede ser un solo número (0-6) o una combinación de ellos.
- acción: Especifica cómo se ejecutará el proceso. Algunos valores comunes para la acción son wait, initdefault, ctrlaltdel, off, respawn, once, sysinit, boot, bootwait, powerwait, entre otros.
- proceso: Es el comando exacto que será ejecutado.

f. Para cambiar de un runlevel <X> a un runlevel <Y> en un sistema que utiliza SysV init, es el comando `init <Y>`.

Por ejemplo:

- Para apagar la máquina (Runlevel 0), usaría `init 0`.
- Para reiniciar la máquina (Runlevel 6), usaría `init 6`

Este cambio no es permanente, para hacer un cambio permanente en el runlevel de arranque, sería necesario modificar el archivo `/etc/inittab` para ajustar la entrada `initdefault` al runlevel deseado.

g. Los Scripts RC (Run Command) son esenciales en los sistemas GNU/Linux que utilizan el sistema de inicialización SysV init para gestionar los servicios durante el arranque y la detención del sistema.

La finalidad de los scripts RC es iniciar o detener una serie de servicios cada vez que el sistema entra o sale de un determinado runlevel (modo de ejecución). Son los encargados de asegurar que los servicios necesarios para cada estado del sistema (como el modo multiusuario o el modo gráfico) estén correctamente activos o inactivos.

Los scripts originales, que contienen la lógica para iniciar y detener los servicios, se suelen guardar en el directorio `/etc/init.d`. Sin embargo, para cada runlevel (del 0 al 6), existen directorios específicos como `/etc/rc0.d`, `/etc/rc1.d`, etc. (donde X es el número del runlevel). Dentro de estos directorios `/etc/rcX.d`, se almacenan enlaces simbólicos a los scripts que residen en `/etc/init.d`.

Cuando un sistema GNU/Linux arranca o se detiene, el proceso init (que tiene el PID 1) es el encargado de orquestar la ejecución de estos scripts.

- El proceso init lee el archivo `/etc/inittab` para determinar el runlevel por defecto y qué scripts ejecutar.
- Los enlaces simbólicos en los directorios `/etc/rcX.d` (donde X es el runlevel) siguen un patrón de nomenclatura
- Este valor numérico garantiza el orden de ejecución de los scripts y puede interpretarse como una prioridad. específico: `[S|K]<orden><nombre>`

Existe un orden para llamar a estos scripts, y es fundamental para el correcto funcionamiento del sistema.

- El componente `<orden>` en el nombre del enlace simbólico es un valor numérico de dos dígitos.

14. SystemD

a. Systemd es un sistema que centraliza la administración de los demonios (servicios) y las librerías del sistema. Fue diseñado para modernizar el proceso de arranque en GNU/Linux, trayendo consigo varias mejoras y cambios significativos respecto a sistemas anteriores como SysV init.

b. En SystemD, el concepto de Unit (unidad de trabajo) hace referencia a las diferentes piezas o componentes que SystemD administra y gestiona. Estas unidades son los elementos fundamentales con los que SystemD opera y pueden tener distintos tipos, cada uno con una función específica.

c. El comando `systemctl` sirve para controlar SystemD. SystemD, a su vez, es un sistema que centraliza la administración de los demonios (servicios) y las librerías del sistema

d. En SystemD, el concepto de target (objetivo) hace referencia a un elemento que agrupa unidades (units) o establece puntos de sincronización durante el proceso de arranque del sistema. Los targets son importantes porque gestionan las dependencias entre las unidades.

c. Al ejecutar el comando `ps-tree`, se puede observar los procesos en curso en forma de un árbol.

```
demian@debian:~$ ps-tree
systemd├─ModemManager─3*[{ModemManager}]
│   └─NetworkManager─3*[{NetworkManager}]
│       └─accounts-daemon─3*[{accounts-daemon}]
│           └─agetty
│               └─avahi-daemon─avahi-daemon
│                   └─colord─3*[{colord}]
│                       └─cron
│                           └─cups-browsed─3*[{cups-browsed}]
│                               └─cupsd
│                                   └─dbus-daemon
│                                       └─dhcpcd─4*[{dhcpcd}]
│                                           └─dhcpcd─2*[{dhcpcd}]
│                                               └─Xorg
│                                                   └─lightdm
│                                                       └─xfce4-session
│                                                           └─Thunar─3*[{Thunar}]
│                                                               └─aplet.py
│                                                                   └─light-locker─4*[{light-locker}]
│                                                                       └─nm-applet─5*[{nm-applet}]
│                                                                           └─polkit-mate-aut─3*[{polkit-mate+}]
│                                                                               └─xfce4-panel─wrapper-2.0─3*[{w+}]
│                                                                                   └─wrapper-2.0─4*[{w+}]
│                                                                                       └─xfce4-power-man─3*[{xfce4-power+}]
│                                                                                           └─xfdesktop─6*[{xfdesktop}]
│                                                                                               └─xfsettingsd─4*[{xfsettingsd}]
│                                                                                                   └─xfwm4─9*[{xfwm4}]
│                                                                                                       └─xiced─3*[{xiced}]
│                                                                                                           └─3*[{xfce4-session}]
│                                                                                                               └─3*[{lightdm}]
│                                                                                                                   └─3*[{lightdm}]
│                                                                                                                       └─3*[{lightdm}]
│                                                                                                                           └─polkitd─3*[{polkitd}]
```

15. Usuarios

a. En un sistema GNU/Linux, la información de los usuarios se guarda en varios archivos de configuración específicos:

- `/etc/passwd`: Este archivo contiene la base de datos de usuarios. Aquí se almacena información básica de cada usuario, incluyendo la shell que tienen asignada por defecto.
- `/etc/group`: En este archivo se registra la información de los grupos de usuarios. Los usuarios pueden pertenecer a uno o más grupos.
- `/etc/shadow`: Este archivo almacena la información de las contraseñas de los usuarios de forma segura. Cuando se asigna o cambia la contraseña de un usuario con el comando `passwd`, este archivo es modificado.

b. En un sistema GNU/Linux, las siglas UID y GID hacen referencia a identificadores numéricos únicos asociados a usuarios y grupos, respectivamente.

• UID (User ID): Hace referencia al identificador único del usuario. No pueden coexistir UIDs iguales en un sistema GNU/Linux. Los usuarios se identifican con un nombre de usuario, el cual es único en el sistema.

• GID (Group ID): Hace referencia al identificador único del grupo.

c. El usuario root es el administrador del sistema o superusuario en GNU/Linux. Es el usuario con los máximos privilegios y control sobre el sistema operativo. No puede existir más de un usuario con el perfil de root en GNU/Linux. Los usuarios se identifican con un nombre de usuario que es único en el sistema. Aunque un usuario regular puede tener la capacidad de realizar tareas de superusuario (es decir, tener los permisos del root), aunque no sea el usuario root en sí mismo.

d. - Creo el grupo informática

```
demian@debian:~$ sudo groupadd -f informatica
[sudo] password for demian:
```

- Creo el usuario isocso con home en `/home/isocso` y miembro del grupo informática

```
demian@debian:~$ sudo useradd -m -d /home/isocso -g informatica isocso
```

-m -> crea el directorio home si no existe

-d `/home/isocso` -> especifica el directorio home

-g informática -> establece el grupo

-Creo un archivo en su home sin iniciar sesión como él

```
demian@debian:~$ sudo touch /home/isocso/archivo_prueba
demian@debian:~$ sudo chown isocso:informatica /home/isocso/archivo_prueba
```

El comando `chown` en Linux sirve para cambiar la propiedad de archivos o directorios. Básicamente, define quién es el dueño del archivo y a qué grupo pertenece.

-Verifico el archivo y los permisos

```
demian@debian:~$ sudo ls -l /home/isocso
total 0
-rw-r--r-- 1 isocso informatica 0 Sep 12 01:19 archivo_prueba
```

-Borro el usuario isocso y su home

```
demian@debian:~$ sudo userdel -r isocso
```

-Verifico que no queden registros

```
demian@debian:~$ sudo grep isocso /etc/passwd
demian@debian:~$ sudo grep isocso /etc/shadow
demian@debian:~$ sudo grep isocso /etc/group
```

Uso los archivos de configuración que ya mencioné, en donde se guardan los datos de usuario, como no me muestra nada, significa que se borró todo bien.

e. i. useradd y adduser

Ambos crean usuarios, pero:

useradd -> comando de bajo nivel, no interactivo.

adduser -> script amigable (en Debian/Ubuntu) que usa useradd por debajo, pero hace preguntas interactivas.

Parámetros comunes de useradd:

- m -> crea el home directory (/home/usuario).
- d /ruta -> especifica una ruta distinta para el home.
- s /bin/bash -> asigna un shell específico.
- u UID -> define un UID personalizado.

ii. usermod

Modifica cuentas de usuario ya existentes.

Parámetros más usados:

- l nuevo_nombre -> cambia nombre de usuario.
- d /nueva/ruta -> cambia home (usar -m para mover archivos).
- s /bin/zsh -> cambia shell.

iii. userdel

Elimina un usuario del sistema.

Parámetros:

- r -> borra también el home directory y mail spool.

iv. su

Permite cambiar de usuario dentro de la terminal.

su [opciones] [usuario]

v. groupadd

Parámetros:

- g GID -> asigna un ID específico al grupo.
- r -> crea un grupo del sistema (GID < 1000 normalmente).

vi. who

Muestra quién está logueado en el sistema.

Parámetros comunes:

-a -> muestra toda la info disponible (incluye tiempo de arranque, procesos, etc.).

-b -> muestra fecha y hora de último arranque.

-q -> muestra solo usuarios logueados y cantidad.

vii. groupdel

Elimina un grupo.

sudo groupdel grupo

viii. passwd

Sirve para cambiar contraseñas de usuarios.

passwd [opciones] [usuario]

16. FileSystem y permisos

a. En un sistema GNU/Linux, los permisos sobre archivos se definen a nivel del sistema de archivos, aplicándose tanto a directorios como a archivos.

Existen tres posibles permisos: Lectura (R), Escritura (W) y Ejecución (X).

Estos permisos se suelen expresar mediante valores octales:

- Lectura (R) tiene un valor octal de 4.

- Escritura (W) tiene un valor octal de 2.

- Ejecución (X) tiene un valor octal de 1.

Cada archivo o directorio en el sistema de archivos posee un usuario dueño y un grupo dueño. Los permisos se pueden definir para las siguientes agrupaciones de usuarios:

- Usuario (U): Son aplicables al usuario dueño del archivo o directorio.

- Grupo (G): Son aplicables a cualquier usuario que pertenezca al grupo dueño, siempre y cuando no sea el usuario dueño.

- Otros (O): Son aplicables a cualquier otro usuario que no forme parte de las agrupaciones anteriores.

En el caso de los directorios, los permisos tienen un efecto particular:

- r (lectura) permite ver su contenido (pero no el contenido de sus archivos internos).

- w (escritura) permite añadir o eliminar ficheros dentro del directorio (pero no modificarlos).

- x (ejecución) permite acceder al directorio

b. El comando `chmod` se utiliza para cambiar los permisos de acceso de un archivo o directorio. Los permisos pueden definirse para el usuario dueño (U), el grupo dueño (G) y otros usuarios (O).

- `chmod [-R][ugo][+/- rwx]` fichero: Esta es la sintaxis general para cambiar los permisos de acceso de un fichero.
 - `[ugo]`: Indica para quién se aplican los permisos (User, Group, Other)
 - `[+/- rwx]`: Indica si se añaden (+) o se quitan (-) permisos, y cuáles permisos (read, write, execute, sticky bits, etc.).
 - `fichero`: El nombre del archivo o directorio al que se le aplicarán los cambios.

El comando `chown` se utiliza para cambiar el propietario de un fichero o directorio. Parámetros y ejemplos:

- `chown [-R] usuario fichero`: Esta es la sintaxis para cambiar el propietario de un fichero o directorio.
 - `usuario`: El nombre del nuevo usuario propietario.
 - `fichero`: El nombre del archivo o directorio.
 - El parámetro `[-R]` se utiliza para aplicar los cambios de forma recursiva.

El comando `chgrp` se utiliza para cambiar el grupo de un fichero o directorio. Parámetros y ejemplos:

- `chgrp [-R] grupo fichero`: Esta es la sintaxis para cambiar el grupo de un fichero o directorio.
 - `grupo`: El nombre del nuevo grupo propietario.
 - `fichero`: El nombre del archivo o directorio.
 - El parámetro `[-R]` se utiliza para aplicar los cambios de forma recursiva.

d. Sí, existe la posibilidad de que un usuario del sistema acceda a un archivo para el cual no posee permisos, específicamente el usuario `root` (superusuario). Normalmente, si un usuario que no es el propietario, ni pertenece al grupo propietario, e incluso si los permisos para "otros" no otorgan acceso, el usuario no podrá acceder al archivo. Sin embargo, el usuario `root` es el administrador del sistema y, como tal, tiene autoridad absoluta para realizar cualquier operación, incluyendo la lectura, escritura o modificación de archivos, independientemente de los permisos establecidos para otros usuarios o grupos.

- Creo dos usuarios de prueba:

```
demian@debian:~$ sudo useradd -m -s /bin/bash user1
demian@debian:~$ sudo useradd -m -s /bin/bash user2
```

- Creo el archivo como `user1` y me aseguro de que `user1` sea el dueño

```
demian@debian:~$ sudo -u user1 bash -c 'echo "Contenido Archivo user1" > /home/user1/archivo.txt'
demian@debian:~$ sudo chown user1:user1 /home/user1/archivo.txt
```

- Pongo permisos 600 y muestro el estado

```
demian@debian:~$ sudo chmod 600 /home/user1/archivo.txt
demian@debian:~$ sudo ls -l /home/user1/archivo.txt
-rw----- 1 user1 user1 24 Sep 20 14:23 /home/user1/archivo.txt
```

Esto último significa que solo el dueño puede leer y escribir, el grupo y los demás usuarios no tienen ningún permiso.

- Intento leer el archivo con el user2 (no me deja)

```
demian@debian:~$ sudo -u user2 cat /home/user1/archivo.txt
cat: /home/user1/archivo.txt: Permission denied
```

- Cambio permisos a 777

```
demian@debian:~$ sudo chmod 777 /home/user1/archivo.txt
demian@debian:~$ sudo ls -l /home/user1/archivo.txt
-rwxrwxrwx 1 user1 user1 24 Sep 20 14:23 /home/user1/archivo.txt
```

Esto último significa que ahora el user2 ahora puede leer y modificar

- Trato de leer con el user2 (ahora si puedo), si quisiera escribir también ya puedo

```
demian@debian:~$ sudo -u user2 cat /home/user1/archivo.txt
Contenido Archivo user1
```

Cómo se leen los permisos:

En Linux, cada archivo tiene tres grupos de permisos:

Usuario/propietario → el dueño del archivo.

Grupo → todos los usuarios que pertenecen al grupo del archivo.

Otros → cualquier otro usuario del sistema.

Y para cada grupo existen 3 permisos:

r = read (leer).

w = write (escribir/modificar).

x = execute (ejecutar, si es un programa o script).

Como se representan: Representación numérica (chmod)

Cada permiso tiene un valor en octal:

r = 4

w = 2

x = 1

Se suman los valores para cada grupo.

Ejemplo:

rw- = 4 + 2 = 6

r-x = 4 + 1 = 5

rwX = 4 + 2 + 1 = 7

🔔 Cada posición es el valor para cada grupo, el primer dígito es para el usuario/propietario, el segundo es para el grupo y el tercero es para otros.
Ejemplo: como hice anteriormente si pongo 777 hace que los tres grupos puedan leer, escribir y ejecutar (rwx).

e. El concepto de "ruta" (path) es fundamental en los sistemas de archivos jerárquicos como GNU/Linux, ya que es una cadena de texto que especifica la ubicación precisa de un archivo.

Existen dos maneras principales de nombrar o referenciar la ubicación de un archivo o directorio: la ruta absoluta y la ruta relativa.

1. Full Path Name (Path Absoluto)

El full path name o path absoluto declara la ubicación de un archivo o directorio comenzando siempre desde el directorio raíz (/). El directorio raíz es el tope de la estructura de directorios. En GNU/Linux, todos los archivos y directorios aparecen bajo el directorio raíz (/), aunque estén físicamente ubicados en distintos dispositivos.

Ejemplo de Path Absoluto:

Directorio Raíz:

- / (El punto de partida de toda la estructura de archivos).

2. Relative Path Name (Path Relativo)

El relative path name o path relativo define la ubicación de un archivo o directorio con respecto al directorio de trabajo actual. Este path no comienza con el directorio raíz (/).

En lugar de listar la ruta completa desde el inicio del sistema, se utilizan referencias para moverse desde la ubicación actual. Para facilitar la navegación relativa, se utilizan caracteres especiales:

El comando `pwd` se utiliza para visualizar el directorio actual, que es el punto de referencia para cualquier path relativo.

Ejemplos de Path Relativo:

Supongamos que el directorio de trabajo actual es `/home/usuario_a/Documentos`.

1. Acceder a un subdirectorio: Para ir al directorio Reportes dentro de Documentos:

- `cd Reportes` (o `cd ./Reportes`).

2. Subir al directorio padre (usuario_a): Para moverse un nivel arriba, a `/home/usuario_a`:

- `cd ..`

f. El comando utilizado para visualizar el directorio de trabajo actual es `pwd`.

La función del comando `pwd` (print working directory) es mostrar el path o ruta completa del directorio donde el usuario está posicionado en la consola o shell. Existe una forma sencilla de ingresar al directorio personal (home) sin necesidad de escribir su ruta completa, en el intérprete de comandos (Shell), si se ejecuta el comando `cd` (change directory) sin especificar ningún argumento (sin nombre de directorio), el Shell te dirige automáticamente a su directorio personal.

También se puede utilizar esta idea de evitar escribir el path completo de dos maneras fundamentales en GNU/Linux, basadas en el uso de rutas relativas y variables de entorno:

1. Uso de Rutas Relativas

La ruta relativa define la ubicación de un archivo o directorio con respecto a su directorio de trabajo actual. En lugar de comenzar desde el directorio raíz (/), se utilizan notaciones especiales:

- .. (Punto punto): Hace referencia al directorio superior o padre.
- . (Punto): Hace referencia al directorio actual.

Ejemplo de uso de Ruta Relativa (..):

Nos encontramos en el directorio
/home/usuario/Documentos/Reportes.

Si se desea subir un nivel para acceder al directorio Documentos, se puede usar la ruta relativa ".."

Comando: cd ".."

Después de ejecutar este comando, el directorio de trabajo actual sería
/home/usuario/Documentos.

2. Uso de Variables de Entorno

El Shell puede almacenar rutas importantes en variables de entorno, lo que permite utilizarlas en lugar de escribir la ruta completa, imitando la "idea" de un atajo para ubicaciones específicas:

El directorio personal de un usuario se identifica frecuentemente con la variable \$HOME.

Ejemplo de uso de Variable de Entorno (\$HOME):

Supongamos que su directorio personal es /home/miusuario. Si desea acceder a un subdirectorio llamado Desktop, podría usar:

Comando: cd \$HOME/Desktop

(en este caso, \$HOME actúa como un atajo a la ruta completa).

g. Comandos de Administración del FileSystem (Sistema de Archivos)

i. umount

Se utiliza para desmontar un dispositivo. Desmontar implica separar lógicamente un sistema de archivos de su punto de montaje en la estructura de directorios, haciéndolo inaccesible a través de esa ruta.

Parametros:

umount /dev/hda2 -> desmonta un dispositivo identificado por /dev/hda2.

ii. du (disk usage)

Se utiliza para mostrar el espacio ocupado en disco.

Parametros:

Si no se indica ningún argumento, muestra el espacio ocupado por el directorio en curso.

iii. df (disk free)

Se utiliza para mostrar información sobre particiones montadas. Este comando es relevante para tareas como informar la cantidad de espacio libre en disco

iv. mount

Se utiliza, en su forma simple, para ver el listado de dispositivos montados. En su forma completa, se usa para montar una partición.

Parametros:

mount -t ext3 /dev/hda2 /disco -> monta una partición.

-t ext3 -> Especifica el tipo de filesystem (sistema de archivos)

v. mkfs (make filesystem)

Formatea una partición o dispositivo de almacenamiento (disco, partición, USB, etc.). Le crea un sistema de archivos (ext4, xfs, vfat, etc.), que es la estructura que usa Linux para organizar y guardar archivos y directorios.

Parámetros:

`mkfs -t ext4 /dev/sdb1` -> crea un sistema de archivos ext4

vi. fdisk

Es un software particionador que permite inspeccionar, crear y eliminar particiones del disco. Es un ejemplo de software destructivo para particionar.

Inspección: Al usar fdisk, se puede seleccionar la opción 4 para ver las particiones del disco. Eliminación: Se puede eliminar una partición usando la opción 3, y luego la opción 1 para seleccionar la partición a eliminar (por ejemplo, la partición 2).

vii. write

Es un comando que sirve para enviar mensajes directamente a la terminal de otro usuario conectado al mismo sistema.

`write nombre_usuario` -> envía mensaje a nombre_usuario

viii. losetup

Es una herramienta de Linux que sirve para asociar archivos regulares con dispositivos de bloque de tipo loop (loop devices).

Parámetros:

`losetup -a` -> lista todos los loop devices activos y a qué archivo están asociados.

ix. stat

El comando stat en Linux muestra información detallada sobre un archivo, directorio o sistema de archivos. Es más completo que `ls -l`, porque enseña metadatos internos directamente del *inode*.

`stat archivo` -> muestra todos los metadatos del archivo

17. Procesos

a. Un proceso en Background (segundo plano) se ejecuta “detrás”, liberando la terminal para seguir usando otros comandos.

Un proceso en Foreground (primer plano) es aquel que se ejecuta ocupando la terminal, no se puede usar hasta que el proceso termine o lo interrumpas.

b. Para ejecutar un proceso en Background, se coloca el carácter & al final del comando, lo que permite que el proceso se ejecute en segundo plano y la terminal quede disponible para otras tareas.

comando &

Para pasar un proceso de background a foreground, se utiliza el comando `fg %n`, donde `n` corresponde al número de *job* obtenido con `jobs`. Si solo existe un proceso en segundo plano, basta con `fg`.

```
fg %1
```

Para pasar un proceso de foreground a background, primero se suspende el proceso en primer plano con la combinación `Ctrl + Z`. Luego se reanuda en segundo plano con el comando `bg %n`.

```
bg %1
```

c. El pipe (`|`) es un operador de la línea de comandos que se utiliza para redirigir la salida estándar de un comando como entrada estándar de otro.

Su finalidad es encadenar comandos, permitiendo crear flujos de procesamiento de datos sin necesidad de archivos temporales.

Ejemplos de utilización

Filtrar resultados con grep:

```
ls -l | grep ".txt"
```

Lista los archivos en formato largo y filtra solo aquellos que terminan en `.txt`.

Ejemplos de utilización

Ordenar resultados:

```
cat archivo.txt | sort
```

Muestra el contenido de `archivo.txt` ordenado alfabéticamente.

d. La redirección permite controlar el flujo de entrada y salida de los comandos en la terminal, enviando la entrada estándar (`stdin`), la salida estándar (`stdout`) o los errores (`stderr`) hacia archivos, dispositivos o incluso descartándolos.

Tipos de redirecciones y finalidad

🔥 Redirección de salida estándar (`>` y `>>`)

Finalidad: Guardar la salida de un comando en un archivo en lugar de mostrarla en pantalla.

Ejemplos:

```
ls > archivos.txt    -> Sobrescribe el archivo con la salida
```

```
ls >> archivos.txt   -> Agrega la salida al final del archivo
```

🔥 Redirección de entrada estándar (`<`)

Finalidad: Tomar la entrada de un comando desde un archivo en lugar de desde el teclado.

Ejemplo:

```
sort < datos.txt     -> Ordena el contenido de datos.txt
```

🔔 Redirección de errores (2> y 2>>)

Finalidad: Guardar los mensajes de error en un archivo separado.

Ejemplos:

ls /carpetalnexistente 2> errores.txt -> Guarda errores en el archivo
ls /carpetalnexistente 2>> errores.txt -> Los agrega al final del archivo

🔔 Redirección combinada (&>, 2>&1)

Finalidad: Combinar salida estándar y errores en un mismo archivo.

Ejemplos:

comando &> salida.log -> stdout y stderr en el mismo archivo
comando > salida.log 2>&1 -> Alternativa equivalente

🔔 Redirección a /dev/null

Finalidad: Descartar la salida o errores.

Ejemplos:

comando > /dev/null -> Descarta la salida estándar
comando 2> /dev/null -> Descarta solo los errores
comando &> /dev/null -> Descarta todo

18. Otros comandos de Linux:

a. El concepto de empaquetar archivos en GNU/Linux hace referencia a unir varios archivos y directorios en un único archivo sin comprimirlos necesariamente. Su finalidad es facilitar el almacenamiento, distribución, copia o respaldo de conjuntos de archivos.

b. Visualización de los archivos y sus tamaños (en bytes)

```
demian@debian:~/Documents/ISO2025/ISOCS0$ ls
archivo1.txt archivo2.txt archivo3.txt archivo4.txt
demian@debian:~/Documents/ISO2025/ISOCS0$ stat -c "%s" archivo1.txt archivo2.txt archivo3.txt archivo4.txt
21
11
19
34
```

Sumo el tamaño de cada uno de los archivos

```
demian@debian:~/Documents/ISO2025/ISOCS0$ du -b archivo1.txt archivo2.txt archivo3.txt archivo4.txt | awk '{sum += $1} END {
print sum "bytes"}'
85bytes
```

Empaqueto los 4 archivos

```
demian@debian:~/Documents/ISO2025/ISOCS0$ tar -cvf archivos.tar archivo1.txt archivo2.txt archivo3.txt archivo4.txt
esto crea un archivo archivos.tar que contiene los 4 archivos empaquetados juntos.
```

Veo el tamaño que tiene este nuevo archivo

```
demian@debian:~/Documents/ISO2025/ISOCS0$ stat -c "%s" archivos.tar
10240
```

Lo que podemos ver es que tar no reduce espacio sino que solo empaqueta a las 4 archivos, si quisiera reducir espacio tendría que comprimir los 4 archivos.

c. Visualizo los archivos que quiero comprimir y sus tamaños

```
demian@debian:~/Documents/ISO2025/ISOCS0$ ls
archivo1.txt archivo2.txt archivo3.txt archivo4.txt
demian@debian:~/Documents/ISO2025/ISOCS0$ stat -c "%s" archivo1.txt archivo2.txt archivo3.txt archivo4.txt
21
11
19
34
```


Creo el archivo empaquetado y comprimido (con tar y gzip)

```
demian@debian:~/Documents/ISO2025/ISOCS0$ tar -cvzf comprimido.tar.gz archivo1.txt archivo2.txt archivo3.txt archivo4.txt
```

Veo el tamaño que tiene el nuevo archivo comprimido

```
demian@debian:~/Documents/ISO2025/ISOCS0$ stat -c "%s" comprimido.tar.gz
222
```

c. Si se pueden comprimir un conjunto de archivos utilizando un único comando, como en el caso de tar (empaquetar + comprimir en un paso)

```
tar -cvzf comprimido.tar.gz arch1 arch2 arch3 arch4
```

Otro ejemplo es con zip

```
zip comprimido.zip arch1 arch2 arch3 arch4
```

d. i. tar

Empaqueta múltiples archivos y directorios en un solo archivo. No comprime solo agrupa, aunque puede usarse junto con opciones para comprimir.

```
tar -cvf archivo.tar arch1 arch2 arch3
```

ii. grep

Busca patrones de texto en archivos o entrada estándar.

```
grep "error" archivo.log -> busca la palabra 'error' en el archivo
```

iii. gzip

Comprime archivos individuales usando el algoritmo DEFLATE.

A diferencia de tar solo trabaja sobre un archivo a la vez.

```
gzip archivo.txt -> Comprime archivo.txt = archivo.txt.gz
```

iv. zgrep

Igual que grep, pero trabaja directamente sobre archivos comprimidos con gzip (.gz)

```
zgrep "error" archivo.log.gz -> Busca 'error' dentro de un archivo comprimido
```

v. wc (word count)

Muestra estadísticas de archivos: líneas, palabras caracteres.

```
wc archivo.txt -> Muestra líneas, palabras y bytes.
```

```
wc -l archivo.txt -> Solo cuenta líneas.
```

19. Comandos I

```
demian@debian:~/Documents/ISO2025/ISOCS0$ ls -l > prueba
```

Ejecuta ls -l y redirige la salida estándar al fichero "prueba" (osea crea "prueba" en el directorio actual)

```
demian@debian:~/Documents/ISO2025/ISOCS0$ ps > PRUEBA
```

Ejecuta ps (lista procesos) y redirige la salida estándar al archivo "PRUEBA" (osea crea PRUEBA en el directorio actual)

```
demian@debian:~/Documents/ISO2025/ISOCS0$ chmod 710 prueba
```

Cambia los permisos de "prueba" a 710 (propietario: rwx (puede leer, escribir y ejecutar), grupo: x (puede ejecutar), otros: 0 (no puede hacer nada).

```
demian@debian:~/Documents/ISO2025/ISOCS0$ chown root:root PRUEBA
chown: changing ownership of 'PRUEBA': Operation not permitted
```

Intenta cambiar propietario y grupo de PRUEBA a root:root, no se puede porque solo root puede reasignar la propiedad de un archivo a otro usuario/grupo.

```
demian@debian:~/Documents/ISO2025/ISOCS0$ chmod 777 PRUEBA
```

Pone permisos rwx (lectura, escritura y ejecución) a todos.

```
demian@debian:~/Documents/ISO2025/ISOCS0$ passwd root
passwd: You may not view or modify password information for root.
```

Intenta cambiar la contraseña del usuario root, no se puede porque solo root (o un administrador a través de sudo) puede cambiar la contraseña de otro usuario.

```
demian@debian:~/Documents/ISO2025/ISOCS0$ rm PRUEBA
```

Borra el fichero "PRUEBA" del directorio actual.

```
demian@debian:~/Documents/ISO2025/ISOCS0$ man /etc/shadow
man: can't open /etc/shadow: Permission denied
```

man busca paginas de manual por nombre, no se abre esa dirección porque es un lugar legible solo por root.

```
demian@debian:~/Documents/ISO2025/ISOCS0$ find / -name *.conf
```

Busca recursivamente desde / archivos cuyo nombre coincida con *.conf.

Se puede ejecutar bien, pero hay muchos directorios a los que el usuario no tiene acceso y salta "Permission denied"

```
demian@debian:~/Documents/ISO2025/ISOCS0$ usermod root -d /home/newroot -L
bash: usermod: command not found
```

Intenta cambiar home de root y bloquear la cuenta, no se puede ejecutar ya que usermod requiere privilegios para modificar cuentas del sistema.

```
demian@debian:~/Documents/ISO2025/ISOCS0$ cd /root
bash: cd: /root: Permission denied
```

Intenta entrar al directorio /root (home de root), pero no se tiene permiso para eso.

```
demian@debian:~/Documents/ISO2025/ISOCS0$ rm *
```

Borra todos los archivos del directorio actual, solo borra lo que el usuario tiene permiso a borrar.

```
demian@debian:~/Documents/ISO2025/ISOCS0$ cd /etc
demian@debian:/etc$
```

Entra al directorio /etc

```
demian@debian:/etc$ cp * /home -R
cp: cannot create regular file '/home/adduser.conf': Permission denied
```

Intenta copiar ficheros/directorios a home, pero no se puede porque se necesitan permisos root para escribir en la raíz del árbol (/home).

```
demian@debian:/etc$ shutdown
bash: shutdown: command not found
```

Ordena apagar la máquina, no se puede porque requiere privilegios de root o sudo.

20. Comandos II

a. Terminar el proceso con PID 23:

```
kill 23
```

b. Terminar el proceso llamado init o systemd:

```
kill init
```

```
kill systemd
```

Esto no se puede hacer con usuario normal, solo root puede intentar matarlo, pero lo mas probable es que tampoco se lo permite porque apagaría o bloquearía el sistema.

c. Buscar todos los archivos de usuarios en los que su nombre contiene la cadena ".conf":

```
find /home -name "*.conf"
```

Esto busca recursivamente en /home.

d. Guardar una lista de procesos en ejecución el archivo /home/<su nombre de usuario>/procesos:

```
ps -ef > /home/demian/procesos
```

> Crea el archivo (si existe, lo sobrescribe).

e. Cambiar permisos de /home/demian/xxxx:

```
chmod 751 /home/demian/xxxx
```

7 -> rwx (Para usuario), 5 -> rx (Para grupo), 1 -> x (Para otros)

f. Cambiar permisos del archivo /home/<su nombre de usuario>/yyyy:

```
chmod 650 /home/demian/yyyy
```

6 -> rw (Para usuario), 5 -> rx (Para grupo), 0 -> ninguno (Para otros).

g. Borrar todos los archivos del directorio /tmp:

```
rm -f /tmp/*
```

h. Cambiar el propietario del archivo /opt/isodata al usuario isocso:

```
sudo chown isocso /opt/isodata
```

Esto requiere privilegios (sudo).

i. Guardar en el archivo /home/<su nombre de usuario>/donde el directorio donde me encuentro en este momento:

```
pwd >> /home/demian/donde
```

>> me asegura que el contenido se agregue sin borrar lo que ya tiene.

21. Comandos III

a. Ingresar al sistema como root

```
demian@debian:~$ su -
```

```
Password:
```

```
root@debian:~#
```

b. Crear un usuario y asignarle una contraseña

```
demian@debian:~$ sudo useradd -m dbonzi
```

```
[sudo] password for demian:
```

c. Archivos y directorios modificados al crear usuario:

Archivos:

- /etc/passwd -> se agrega la entrada del nuevo usuario.
- /etc/shadow -> se guarda la contraseña cifrada.
- /etc/group -> puede actualizarse con su grupo principal.
- /etc/gshadow -> se gestiona seguridad de grupos.

Directorios creados:

- /home/dbonzi/ (directorio personal del usuario).

Posiblemente archivos de configuración inicial copiados desde /etc/skel.

d. Crear un directorio en /tmp llamado miCursada

```
demian@debian:~$ mkdir /tmp/miCursada
```

e. Copiar todos los archivos de /var/log al directorio antes creado.

```
demian@debian:~$ cp /var/log/* /tmp/miCursada
```

f. Para el directorio antes creado (y los archivos y subdirectorios contenidos en él) cambiar el propietario y grupo al usuario creado y grupo users.

```
demian@debian:~$ sudo chown -R dbonzi:users /tmp/miCursada
```

-R lo aplica recursivamente

g. Agregar permiso total al dueño, de escritura al grupo y escritura y ejecución a todos los demás usuarios para todos los archivos dentro de un directorio en forma recursiva.

```
demian@debian:~$ chmod -R 723 /tmp/miCursada
```

7 -> rwx (Para usuario), 2 -> w (Para grupo), 3 -> wx (Para otros)

h. Acceda a otra terminal para loguearse con el usuario antes creado.

```
demian@debian:~$ su - dbonzi
```

Password:

i. ver cuál es el nombre de su terminal.

```
$ tty
```

/dev/pts/1

j. Verifique la cantidad de procesos activos que hay en el sistema.

```
demian@debian:~$ ps -e | wc -l
```

154

k. Verifiqué la cantidad de usuarios conectados al sistema.

```
demian@debian:~$ who | wc -l
```

1

l. Vuelva a la terminal del usuario root y envíele un mensaje al usuario anteriormente creado enviándole que el sistema va a ser apagado.

```
root@debian:~# echo "El sistema va a ser apagado" | sudo tee /dev/pts/1
```

El sistema va a ser apagado

Mensaje visto desde la terminal de dbonzi

```
demian@debian:~$ su - dbonzi
```

Password:

```
$ tty
```

/dev/pts/1

```
$ El sistema va a ser apagado
```

22. Comandos IV

n. Crear un directorio cuyo nombre sea su num de legajo e ingresar a el

```
demian@debian:~$ mkdir 24260
demian@debian:~$ cd 24260
demian@debian:~/24260$
```

o. Crear un archivo utilizando el editor de textos vi, introduzca su información personal: Nombre, Apellido, Número de alumno y dirección de correo electrónico. El archivo debe llamarse "LEAME".

```
demian@debian:~/24260$ vi LEAME
demian@debian:~/24260$ cat LEAME
Nombre: Demian
Apellido: Bonzi
Numero de alumno: 24260
Correo: demianbonzi@gmail.com
```

p. Cambiar los permisos del archivo

```
demian@debian:~/24260$ chmod 017 LEAME
0 -> ninguno (Para usuario), 1 -> x (Para grupo), 7 -> rwx (Para otros)
```

q. Ir al directorio /etc y verificar su contenido.

```
demian@debian:~/24260$ cd /etc
demian@debian:/etc$ ls
```

Crear un archivo dentro de su directorio personal cuyo nombre sea leame donde el contenido del mismo sea el listado de todos los archivos y directorios contenidos en /etc.

```
demian@debian:/etc$ ls -l > ~/leame
```

La razón por la cual se puede crear "leame" aunque exista "LEAME" es porque en Linux los nombres de archivos son sensible a mayúsculas y minúsculas, entonces LEAME y leame son archivos diferentes.

r. localizar un archivo puntual

```
demian@debian:/etc$ find / -name "archivo.txt"
```

Localizar varios archivos con un patrón

```
demian@debian:/etc$ find / -name "*.conf"
```

find recorre el árbol de directorios aplicando condiciones (-name, -size, -type, etc.). Sirve para buscar un archivo exacto o un conjunto de archivos que cumplan criterios.

s. Buscar todos los archivos .so y guardarlos en ejercicioF dentro del directorio legajo

```
demian@debian:~$ find / -name "*.so" > ~/24260/ejercicioF 2>/dev/null
```

> ~/12345/ejercicioF -> guarda los resultados en el archivo.

2>/dev/null -> descarta los mensajes de "Permission denied" que aparecerían en directorios sin acceso.

23. Comandos V

01. mkdir iso

Crea un directorio llamado iso dentro del directorio actual (\$HOME/iso).

02. cd ./iso; ps > f0

cd ./iso → entra en el directorio iso.

ps > f0 → ejecuta ps (muestra procesos en ejecución) y redirige la salida al archivo f0 en iso.

03. ls > f1

Lista el contenido del directorio actual (iso) y lo guarda en el archivo f1 (sobrescribiéndolo si ya existe).

04. cd /

Cambia el directorio actual a la raíz del sistema (/).

05. echo \$HOME

Muestra el valor de la variable de entorno HOME (Ej: /home/usuario).

06. ls -l \$> \$HOME/iso/ls

Este comando es inválido.

\$> no es un identificador de variable válido → el shell marcará error de sintaxis.

07. cd \$HOME; mkdir f2

cd \$HOME → vuelve al directorio personal del usuario.

mkdir f2 → crea el directorio f2 dentro de \$HOME.

08. ls -ld f2

Muestra información detallada del directorio f2 (permisos, propietario, grupo, etc.), sin listar su contenido.

09. chmod 341 f2

Cambia permisos de f2 a:

3 (propietario: -wx) → puede escribir y ejecutar, pero no leer.

4 (grupo: r--) → solo lectura.

1 (otros: --x) → solo ejecución.

10. touch dir

Crea un archivo vacío llamado dir en \$HOME (si no existía).

Si ya existía, actualiza su fecha de acceso/modificación.

11. cd f2

El usuario entra en el directorio f2.

Después del chmod 341, el dueño no tiene permiso de lectura en f2.

→ El cd f2 funciona, porque el propietario tiene permiso de ejecución (x), suficiente para acceder.

12. cd ~/iso

Va al directorio \$HOME/iso.

13. `pwd > f3`

Guarda la ruta completa del directorio actual (`$HOME/iso`) en el archivo `f3` (dentro de `iso`).

14. `ps | grep 'ps' | wc -l >> ../f2/f3`

`ps` → lista procesos.

`grep 'ps'` → filtra los que contienen "ps".

`wc -l` → cuenta cuántas líneas.

`>> ../f2/f3` → añade el resultado al archivo `f3` dentro de `f2`.

Puede fallar si los permisos de `f2` no permiten escritura.

15. `chmod 700 ../f2 ; cd ..`

`chmod 700 ../f2` → da permisos completos (lectura, escritura, ejecución) solo al propietario de `f2`.

`cd ..` → sube un nivel (de `iso` a `$HOME`).

16. `find . -name etc/passwd`

Busca dentro de `$HOME` (y subdirectorios) algún archivo con nombre exacto `etc/passwd`.

17. `find / -name etc/passwd`

Busca desde la raíz del sistema.

Como usuario normal, recibirá muchos "Permission denied", pero puede encontrar rutas si tiene permiso de lectura (por ejemplo, en `/usr/share/...`).

18. `mkdir ejercicio5`

Crea un directorio llamado `ejercicio5` en `$HOME`.

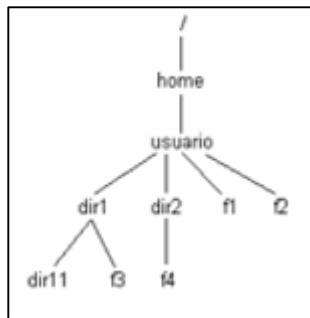
19. `cp -r iso ejercicio5/`

Copiar el directorio y todo su contenido al directorio `ejercicio5`

20. `cp -r f2 dir f0 f1 f3 ejercicio5/`

Copiar el resto de los archivos y directorios creados en este ejercicio al directorio `ejercicio5`

24. Directorios, subdirectorios y archivos.



Creo el directorio dir1, con dir11 dentro y el directorio dir 2

```
demian@debian:~$ mkdir -p "/home/demian/dir1/dir11"
demian@debian:~$ mkdir -p "/home/demian/dir2"
```

Creo archivos f1 y f2

```
demian@debian:~$ touch f1
demian@debian:~$ touch f2
```

Creo archivos f3 y f4 dentro de dir 2

```
demian@debian:~$ touch "/home/demian/dir2/f3" "/home/demian/dir2/f4"
```

Muevo f3 al directorio /home/usuario

```
demian@debian:~$ mv "/home/demian/dir2/f3" "/home/demian"
```

Compruebo si se movió correctamente

```
demian@debian:~$ ls -l "/home/demian/f3"
-rw-rw-r-- 1 demian demian 0 Sep 22 14:55 /home/demian/f3
```

Copio f4 en el directorio dir11

```
demian@debian:~$ cp "/home/demian/dir2/f4" "/home/demian/dir1/dir11/"
```

Comprobar si se copió correctamente

```
demian@debian:~/dir1/dir11$ ls -l "f4"
-rw-rw-r-- 1 demian demian 0 Sep 22 15:06 f4
```

Copio f4 en el directorio dir11, pero el archivo destino debe llamarse f7

```
demian@debian:~$ cp "/home/demian/dir2/f4" "/home/demian/dir1/dir11/f7"
```

Compruebo

```
demian@debian:~/dir1/dir11$ ls -l "f7"
-rw-rw-r-- 1 demian demian 0 Sep 22 15:20 f7
```

Creo el directorio copia en home/demian y copio en el mismo el contenido de dir1

```
demian@debian:~$ mkdir -p "/home/demian/copia"
demian@debian:~$ cp -r "/home/demian/dir1/*" "/home/demian/copia/"
```

Compruebo

```
demian@debian:~/copia$ ls
dir11
```

Renombro f1 por "archivo" y visualizo los permisos

```
demian@debian:~$ mv "/home/demian/f1" "/home/demian/archivo"
demian@debian:~$ ls -l "/home/demian/archivo"
-rw-rw-r-- 1 demian demian 0 Sep 22 14:52 /home/demian/archivo
```

Permisos: rw -> lectura/escritura (Para usuario), rw -> lectura/escritura (Para grupo), r -> lectura (Para otros)

Cambio los permisos de "archivo"

```
demian@debian:~$ chmod 617 "/home/demian/archivo"
```

6 -> rw (Para usuario), 1 -> x (Para grupo), 7 -> rwx (Para otros)

```
demian@debian:~$ ls -l "/home/demian/archivo"
```

```
-rw--x-rwx 1 demian demian 0 Sep 22 14:52 /home/demian/archivo
```

Renombro los archivos "f3" y "f4" a "f3.exe" y "f4.exe"

```
demian@debian:~$ mv ~/f3 ~/f3.exe
```

```
demian@debian:~$ mv ~/dir2/f4 ~/dir2/f4.exe
```

El símbolo ~ es un atajo que representa el directorio personal

24. Directorios, subdirectorios y archivos.

a. Crear un directorio llamado logs en el directorio /tmp

```
mkdir /tmp/logs
```

b. Copiar todo el contenido del directorio /var/log en el directorio creado en el punto anterior.

```
cp -r /var/log/* /tmp/logs
```

c. Empaquetar el directorio creado en a), el archivo resultante se debe llamar "misLogs.tar".

```
tar -cvf misLogs.tar -C /tmp logs
```

d. Empaquetar y comprima el directorio creado en a), el archivo resultante se debe llamar "misLogs.tar.gz".

```
tar -czvf misLogs.tar.gz -C /tmp logs
```

e. Copie los archivos creados en c) y d) al directorio de trabajo de su usuario.

```
cp misLogs.tar misLogs.tar.gz ~/
```

f. Elimine el directorio creado en a), logs.

```
rm -r /tmp/logs
```

g. Desempaquete los archivos creados en c y d en 2 directorios diferentes.

```
//Crear directorios de destino
```

```
mkdir ~/logs_tar ~/logs_targz
```

```
//Desempaquetar misLogs.tar
```

```
tar -xvf ~/misLogs.tar -C ~/logs_tar
```

```
//Desempaquetar misLogs.tar.gz
```

```
tar -xzvf ~/misLogs.tar.gz -C ~/logs_targz
```