**Trabajo Práctico N°1 – ISO.**

1. **GNU/LINUX:**
   1. Características de GNU/LINUX:

* Sistema operativo tipo Unix, pero LIBRE. Esto quiere decir que es un sistema que se comporta y opera de manera parecida a **Unix**, aunque no necesita ser certificado como una versión oficial de este. Unix es un tipo de S.O **multiusuario**, **multitarea** y **portátil**, conocido por su robustez y eficiencia.
* Gratuito y de libre distribución
* Posee diversas distribuciones y intérpretes de comando
* Es case sensitive y de código abierto, permite al usuario estudiarlo, personalizarlo, aprovecharse de la documentación y otras cosas.
  1. Sistemas Operativos en comparación a GNU/LINUX:
* Windows:
* Sistema operativo desarrollado por Microsoft, no es de tipo Unix, sino de MS-DOS
* Es de pago, con licencia. No es libre distribución
* Posee distintas versiones del sistema, no distribuciones
* No es de código abierto, por ende, no es accesible a su KERNEL para modificaciones del mismo
* MacOS:
* Es un sistema operativo tipo Unix, de pago. Es certificado como una versión oficial de este
* Es gratuito, pero no libre; viene incluido sin costo adicional en la compra de una Mac y se obtiene como una actualización de software gratuita para los equipos compatibles
* No posee distribuciones
* No es de código abierto
  1. Las siglas GNU vienen de = **G**NU **N**ot **U**nix (GNU No es Unix). Es un sistema operativo que permite a los usuarios acceder a su código para estudiarlo o realizar modificaciones de él.
  2. **GNU** es un proyecto iniciado por **Richard Stallman** con el fin de crear un Unix Libre (GNU). El mismo necesitó de un marco regulatorio conocido como **GPL** (General Public License de GNU).

Stallman creo la **Free Software Foundation** (FSF) para poder financiar el proyecto. El problema de GNU es que faltaba un componente principal, denominado **KERNEL.** Si bien se venía trabajando en un núcleo conocido como **TRIX**, el mismo fue abandonado dado su complejidad (Solamente corria en hardware muy costoso para la época). Cómo alternativa, Stallman, decidió adoptar como base el **Núcleo MACH** para crear **GNU Hurd**, que tampoco prosperó. Fue así cómo aparece **Linus Torvalds**, quien ya venia trabajando en un Núcleo (KERNEL) denominado **LINUX** el cuál se distribuiría bajo la licencia GPL. En el 92’, Torvalds y Stallman fusionan sus proyectos haciendo así GNU/Linux.

* 1. La multitarea es una función que poseen los sistemas operativos de realizar varias tareas al mismo tiempo por parte del usuario. Por ejemplo; Tener 2 ventanas abiertas, realizaron trabajos de impresión mientras el usuario escribe un texto, etc… GNU/Linux hace uso de la multitarea. En ello se destacan los tipos de S.O Unix.
  2. **POSIX** es una familia de estándares del IEEE que define un conjunto de interfaces de sistema, shells y utilidades comunes para garantizar la **compatibilidad** y **portabilidad** del software entre diferentes sistemas operativos. Esto significa que un programa escrito para cumplir con POSIX en una distribución GNU/Linux debería ejecutarse sin problemas en otra, así como en otros sistemas operativos basados en UNIX.

1. **Distribuciones de GNU/Linux:**
   1. Una distribución de GNU/Linux es una customización formada por una versión de **KERNEL** y determinados programas con sus configuraciones. Algunas de las distribuciones que podemos conocer son **DEBIAN, LINUX MINT, UBUNTU** o **FEDORA.**

**Ubuntu**, ideal para principiantes y usuarios en general; **Debian**, una base estable para muchas otras distros; **Linux** **Mint**, para una transición sencilla desde Windows; y **Fedora**, para usuarios que quieren software de vanguardia.

* 1. Una distribución de otra puede diferenciarse en base a su propósito, interfaz gráfica, utilidad, legibilidad y demás características. Como ejemplo, están aquellas que nombramos anteriormente.
  2. Debian es una distribución de GNU/Linux reconocida por varios usuarios, desarrolladores y entornos empresariales. Se valora a Debian por la estabilidad y fluidez de los procesos de actualización, como de los paquetes de toda la distro. Funciona en numerosas arquitecturas y dispositivos.

El proyecto **Debian** nace en Agosto de 1993, cuando **Ian Murdock** anuncia su creación en un mensaje a la lista de correo “**comp.os.linux.development”** . El nombre proviene de la combinación de su nombre y el de su entonces novia (esposa después), **Debra: Deb + Ian**. En los años 1994-1995 se publican **las primeras versiones** (0.9x) y se establecen los **Principios del Software Liibre de Debian**, donde se crea el **Contrato Social de Debian**, su base filosófica. De allí hasta la actualidad se han lanzado varias versiones de Debian con distintas características y soportes de arquitecturas. Al día de hoy su versión más reciente a lanzar es **Debian 13 “Trixie”**.

Tiene como objetivo crear un S.O libre, disponible para todos. Se refieren a “libre” como LIBERTAD del SOFTWARE. Esa es su filosofía.

1. **Estructura de GNU/Linux:**
   1. Los 3 componentes fundamentales de GNU/Linux son:

* **KERNEL**
* **HERRAMIENTAS DE GNU**
* **El Intérprete de comandos (Shell)**
  1. Estructura básica del sistema operativo de GNU/Linux:
* **NÚCLEO**: Conocido como **KERNEL**. Es el que ejecuta programas y gestiona dispositivos de hardware. Se encarga de tratar que el software y hardware trabajen juntos. Es **monolítico híbrido**, esto quiere decir que los drivers se ejecutan en modo privilegiado o modo kernel. Lo hace hibrido la capacidad de cargar y descargar funcionalidad a través de módulos
* **INTERPRETE** **DE** **COMANDOS**: Conocido como CLI (Command Line Interface), es el modo de comunicación entre un usuario y el S.O. Ejecuta programas a partir del ingreso de comandos.
* **SISTEMA DE ARCHIVOS**: Organiza la forma en que se almacenan los archivos en dispositivos de almacenamientos, el adoptado por GNU/Linux es el Extended (v2,v3,v4). Esta organizado según FHS (Filesystem Hierarchi Standard = Estándar de Jerarquía del Sistema de Archivos).
* **UTILIDADES:** Paquetes de software que permiten diferenciar las distribuciones. Poseen editores de texto, herramientas de networking, paquetes de oficina o interfaces gráficas.

1. **Kernel:**
   1. La funcion/es principal/es del kernel es gestionar los recursos de hardware y de software del sistema. Encargado de hacer que estos trabajen juntos.

Entre las funciones más importantes se encuentran la administración de memoria, la gestión de la CPU y E/S. Es decir, el corazón del S.O.

* 1. La versión actual es la **6.16.3**.

Definición del esquema de versionado del KERNEL en versiones anteriores a **2.4** y a partir de la **2.6**: Nomenclatura A.B.C.(D)

* **A:** Denota versión. Cambia con menor frecuencia. En 1994 (1.0), en 1996 (2.0), etc…
* **B:** Denota mayor revisión. Antes de la versión 2.6, los numeros impares indican desarrollo, los pares producción. Las versiones

impares eran consideradas más experimentales y menos estables, mientras que las versiones pares se consideraban más estables y adecuadas para su uso en sistemas de producción.

* **C:** Denota menor revisión. Solo cambia cuando hay nuevos drivers o características.
* **D:** Cambia cuando se corrige un error grave sin agregar nueva funcionalidad.

Esquema de Versionado Anterior a la 2.4 = Modelo "Par/Impar"). a partir de la versión 2.6 = Fin del Modelo Par/Impar.

* 1. Si, es posible tener más de 1 kernel GNU/Linux en nuestra máquina. Estos pueden iniciarse mediante un **gestor de arranque multiple** (**GRUB**), también conocido como un BOOTLOADER (Su finalidad es la de cargar la imagen del kernel -S.O- de alguna partición para su ejecución).
  2. El kernel, dentro del File System, se encuentra ubicado en **/boot**

1. **Intérprete de comandos (Shell):**
   1. Conocido como CLI (Command Line Interface), es el modo de comunicación entre un usuario y el S.O. Ejecuta programas a partir del ingreso de comandos.
   2. Su objetivo es brindar un modo de comunicación entre el usuario y el SO ejecutando programas a partir del ingreso de comandos.

* **Bourne Shell (sh);** Es el **estandar mínimo** Los scripts que empiezan con **#!/bin/sh** deben usar solo esta sintaxis básica para garantizar la máxima portabilidad en cualquier sistema Unix-like.
* **Korn Shell (ksh);** Fue una **mejora poderosa** que popularizó el shell scripting avanzado y la interactividad, dominando en entornos Unix comerciales.
* **Bourn Again Shell (bash);** Es el shell más popular y completo hoy en día. Es el estándar de facto en Linux y macOS, combinando lo mejor de sh y ksh y añadiendo muchas características propias. Es la mejor opción para uso interactivo y scripting moderno.
  1. Los comandos propios (internos) no tienen una ruta (path) física. Estan cargados en la memoria RAM del proceso del SHELL. Ejemplos: cd, echo, export, etc…

Los comandos externos (Ejecutables) se ubican (path) en directorios específicos.

* **Externos (Esencial):** /bin, /usr/bin
* **Externos (Admin):** /sbin, /usr/sbin
* **Externos (Usuario):** ~/.local/bin
* **Externos (Manual):** /usr/local/bin
  1. Considero que el SHELL no es parte del Kernel de GNU/Linux dado que Unix-Like tiene la arquitectura y filosofía de que cada componente debe hacer una sola cosa de manera independiente. El kernel y el shell tienen roles completamente distintos.

Además; El kernel opera en **modo kernel** (con acceso total al hardware), mientras que el shell se ejecuta en **modo usuario** (con restricciones). Si el shell fuera parte del kernel, cualquier vulnerabilidad en él (como inyección de comandos) comprometería directamente el núcleo del sistema, lo que sería un riesgo crítico de seguridad.

* 1. En GNU/Linux cada usuario tiene asignado un shell por defecto que se ejecuta cuando inicia sesión. Además, cada shell por defecto puede ser diferente (bash, zsh, fish, ksh, etc.).

El shell se define en el campo final de su entrada en el archivo /etc/passwd.

Un usuario puede cambiar su propio shell (usando chsh) siempre que el nuevo shell esté en **/etc/shells**. Solo el **root** puede cambiar el shell de otro usuario.

1. **El sistema de Archivos (File System) en Linux:**
   1. Un sistema de archivos organiza la forma en que se almacenan los archivos en dispositivos de almacenamientos, el adoptado por GNU/Linux es el Extended (v2,v3,v4).
   2. La **estructura básica** de los File System en GNU/Linux sigue el Estándar de **Jerarquía del Sistema de Archivos** (FHS - Filesystem Hierarchy Standard). El sistema de archivos se organiza como un único árbol que comienza desde el directorio raíz (/). Todos los demás directorios "cuelgan" de este.

Directorios más importantes:

* **/** = Tope de la estructura de directorios. Es como el C:\
* /**home** = Se almacenan archivos de USUARIOS (Mis Docs)
* /**var** = Información que varía de tamaño (Iogs, BD, spools)
* /**etc** = Archivos de configuración
* /**bin** = Archivos binarios y ejecutables
* /**dev** = Enlace a dispositivos
* /**usr** = Apps de usuarios.

Las siglas **FHS** hacen referencia a (Filesystem Hierarchi Standard = Estándar de Jerarquía del Sistema de Archivos).

* 1. Los sistemas de archivos **soportados** por GNU/Linux son:
* ext2, ext3, ext4 (Fourth Extended) 🡪 Estandar por defecto
* XFS
* Btrfs
* ZFS
* F2FS
* ReiserFS
* VFAT (FAT16/FAT32)
* exFAT
* NTFS
  1. Si, es posible dado que es soportado en GNU/Linux. Puede operar sobre ellos sin necesidad de un software adicional par FAT, sino con herramientas integradas en el kernel moderno o drivers para NTFS.

1. **Particiones:**
   1. Es una forma de dividir **lógicamente** el disco físico. Los tipos de particiones que existen son:

* **Partición Primaria**: División cruda del disco (Puede haber hasta 4 por disco). Se almacena información de la misma en el Master Boot Record (mbr)
* **Partición Extendida**: Sirve para contener unidades lógicas en su interior. Solo puede existir una partición de este tipo por disco. No se define un tipo de File System directamente sobre ella
* **Partición Lógica**: Ocupa la totalidad o parte de la partición extendida y se le define un tipo de File System. Las particiones de este tipo se conectan como una lista enlazada.

La **ventaja** de las particiones es que se considera una **buena práctica** al momento de separar los datos del usuario de las aplicaciones y/o sistemas operativos instalado. Se puede tener una partición de **restore** de todo el sistema. Poder **ubicar el KERNEL en una partición de solo lectura**, o una que ni siquiera se monta (No disponible para usuarios).

La **desventaja** de particionar demasiado un disco se nota al momento de gestionar sobre él. Se requiere decidir anticipadamente el tamaño de cada partición, dado que redimensionarlas después es **complejo y riesgoso**. Así mismo esto genera una **fragmentación de espacio**; El espacio libre se divide en muchos bloques pequeños imposibilitando almacenar archivos grandes, aunque el espacio total libre sea suficiente.

* 1. En GNU/Linux, las particiones se identifican mediante un esquema de nombres consistente basado en el tipo de controlador y la topología física.
* **SATA (Modernos):** Estos discos son tratados por el kernel como dispositivos SCSI, independientemente de su interfaz física. Utilizan la **Nomenclatura:** /dev/sdX#

🡪 SD: Indica un disco de tipo SCSI/SATA/USB.

🡪 X: Letra que identifica el disco.

-> #: El número que identifica la partición dentro del disco

* + Ejemplo:
    - **/dev/sda:** Primer disco SATA/SCSI/USB.
    - **/dev/sdb3**: Tercera partición del segundo disco.
* **IDE (Obsoletos, pero relevantes):** Aunque ya no son comunes, se identifican de forma diferente.
* Utilizan la **Nomenclatura:** /dev/hdX#

-> hd: Indica un disco IDE/PATA

-> X: Letra que identifica el disco y el canal IDE

-> #: Número de la partición

* + Ejemplo:
    - **/dev/hda:** Primer disco IDE (primary master).
    - **/dev/hdc2:** Segunda partición del tercer disco IDE (secondary master).
  1. Como mínimo se necesita 1 partición:
* **Tipo**: Primaria o Extendida/Logical
* **Identificación**: /dev/sda1 (Ejemplo)
* **FS**: ext4 (o XFS/Btrfs)
* **Punto de montaje**: **/** (Raíz)

De todas maneras, se **recomiendan** básicamente 2 particiones:

**Partición SWAP:**

* **Tipo**: Primaria o Extendida/Logical
* **Identificación**: /dev/sda2 (Ejemplo)
* **FS**: swap
* **Punto de montaje**: No se monta,s e activa con **swapon**

**Partición Raíz (/):**

* **Tipo**: Primaria o Extendida/Logical
* **Identificación**: /dev/sda1 (Ejemplo)
* **FS**: ext4
* **Punto de montaje**: **/** (Raíz)
  1. –
  2. Para crear particiones, se utiliza un software llamado **particionador**, que puede ser **destructivo**, permitiendo crear y eliminar particiones, o **no** **destructivo**, permitiendo crear, eliminar y modificar particiones existentes. Las herramientas de particionamiento incluyen **fdisk** para particionamiento destructivo y **gparted** o **ps** para no destructivo. Muchas distribuciones de sistema operativo tienen una interfaz gráfica de instalación que permite crear particiones durante el proceso de instalación.

1. **Arranque (bootstrap) de un Sistema Operativo:**
   1. El **BIOS** (SISTEMA BÁSICO DE ENTRADA/SALIDA) es un firmware de bajo nivel, que se encuentra almacenado en un chip de la placa madre. Se encarga de iniciar la computadora, verificar el hardware (POST – Prueba de Encendido) y cargar el sistema operativo. Gestiona la comunicación entre el software y los componentes de la PC. Proporciona una interfaz para configurar parámetros del sistema como la fecha, la hora, el orden de arranque, etc…
   2. El **UEFI** (Unified Extensible Firmware Interface) es una interfaz moderna de FIRMWARE que reemplaza al BIOS. Está diseñada para iniciar la PC y servir como puente entre el S.O y el hardware. Principalmente incluye un arranque más rapido, soporta disco duros de gran capacidad (Superiores a 2TB), posee opciones de seguridad avanzada como el Secure Boot y brinda la posibilidad de interactuar a través de una interfaz gráfica con el mouse.
   3. El **MBR**, también conocido como **Master Boot Record**, es un sector especial reservado en el primer sector (Cilindo 0, cabezal 0, sector 1) del disco físico. Es el primero en ser leído por el BIOS durante el proceso de arranque de la computadora y contiene información importante para cargar el S.O. El tamaño del MBR es de 512 Bytes (Tamaño estándar de sector en la mayoria de los discos). Los primeros bytes del MBR contienen el **Código de Arranque Principal** (Master Boot Code o **MBC)** que permite iniciar el S.O.
   4. Las siglas **GPT** hacen referencia a **GUID Partion Table**. El sistema GPT es parte de EFI (Extensible Firmware Interface) y puede ver como una sustitución del MBR. La misma se utiliza para solucionar limitaciones del MBR, como la cantidad de particiones (De igual manera se mantiene un MBR para mantener la compatilidad con el esquema BIOS).

**GPT** usa modo de direccionamiento lógico (Logical Block Addressing LBA) en lugar de “cylinder-header-sector”.

En el LBA 1 está la cabecera GPT. La tabla de particiones en sí está en los bloques sucesivos. La cabecera y la tabla están escritas al principio y al final del disco (redundancia).

* 1. La finalidad del bootloader (O **gestor** **de** **arranque**) es la de cargar una imagen de Kernel (sistema operativo) de alguna partición para su ejecución. Existen 2 modos de instalación:

1. En el MBR (Puede llegar a utilizar MBR GAP)
2. En el sector de arranque de la partición raíz o activa (Volume Boot Record)

Algunos de los gestores de arranque conocidos son; GRUB, LILO, NTLDR, GAG, YaST, etc.

* 1. El proceso de arranque se define como el proceso de inicio de una maquina y carga del sistema operativo, denominado **Bootstrap**.

En las arquitecturas x86, el BIOS es el responsable de iniciar la carga del SO a través del MBC.

En sí al iniciar una maquina se carga el programa de booteo (Desde el MBR) y se inicia el gestor de arranque lanzado desde el **MBC** que carga el kernel; Se prueba y hace disponible los dispositivos y luego pasa el control al proceso **init**.

* 1. Principales pasos del arranque de GNU/Linux:

1. **POST (Power-ON Self-Test)**: La BIOS/UEFI verifica el hardware
2. **Gestor de Arranque (Bootloader)**: Si es BIOS, se ejecuta el código del MBR y carga GRUB (o otro). Si es UEFI carga directamente GRUB desde la partición ESP (Efi System Partition)
3. **Ejecución del kernel**: El bootloader carga el kernel Linux y el **initramfs** en memoria
4. **Inicialización del kernel**: El kernel monta el sistema de archivos raíz y ejecuta systemd (o **init**)como primer proceso (PID 1)
5. **Ejecución de servicios**: systemd/init inician los servicios y runleveles/targets definidos.
6. **LOGIN:** Se inicia el servicio de login (getty) para permitir el acceso de usuarios.
   1. Pasos del proceso de parada (Shutdown) en GNU/Linux:
7. **Inicio del comando:** Un usuario ejecuta shutdown, poweroff, halt o reboot
8. **Notificación a los usuarios:** El sistema envía un mensaje a los usuarios logueados anunciando el apagado
9. **Señal a los procesos:** Se envía la señal **SIGTERM** a todos los procesos para que terminen de forma ordenada (Guardando datos y cerrando recursos)
10. **Fuerza la terminación**: Tras un tiempo, se envía la señal de **SIGKILL** a los procesos que no respondieron, forzando su cierre
11. **Desmontaje de sistema de archivos**: Se desmontan todas las particiones (/home, /var, etc…) para evitar corrupción de datos
12. **Señal al kernel**: El proceso **init** notifica al kernel que apague el sistema
13. **Apagado físico:** El **kernel** cierra todos los dispositivos y envía una señal de apagado al hardware
    1. Si, es posible tener un sistema GNU/Linux y otro sistema operativo instalados en una misma PC. Estos pueden iniciarse mediante un gestor de arranque multiple (**GRUB**), también conocido como un BOOTLOADER (Su finalidad es la de cargar la imagen del kernel -S.O- de alguna partición para su ejecución).
14. **Archivos y editores:**
    1. Los archivos en GNU/Linux se identifican mediante;

* **Rutas absolutas o relativa:**
  + Absolutas: Comienzan en la raíz del sistema (/). Ejemplo: /home/usuario/doc.txt
  + Relativa: Depende del directorio actual. Ejemplo: “./script.sh”
* **Inodo (inode)**: Estructura única que almacena metadatos del archivo. Se consulta con LS-I o STAT
* **Nombre de archivo:** Entrada en un directorio que asocia un nombre humano a un nodo. También múltiples nombres (enlaces duros) pueden apuntar al mismo inodo
* **Tipos de archivos:** Indicado por el primer carácter en LS -L
  + **-** : Archivo regular
  + **d**: Directorio
  + **l**: Enlace simbólico
  + etc…
* **Metadatos:** Permisos (rwx), propietario (usuario/grupo), timestamps (Acceso, modificación, cambio) y tamaño. Se visualizan con LS-L o STAT
* **Identificadores únicos:**
  + **UUID** (Universally Unique Identifier): Para sistema de archivos. Ejemplo: Particiones
  + **Major/Minor number**: Para dispositivos. Ejemplo “/dev/sda1”
  1. Editor de Texto **vim:**
* Posee 3 modos de ejecución:
  + Modo Insert (**i**)
  + Modo visual (**v**)
  + Modo de Órdenes o Normal (**esc**)
* Se le puede enviar una serie de comandos útiles:
  + **W:** escribir cambios
  + **Q** o **Q!**: Salir del editor
  + **dd:** Cortar
  + **y:** Copiar al portapapeles
  + **p:** Pegar desde el portapapeles
  + **u:** Deshacer
  + **/frase**: Busca “Frase” dentro del archivo

Editor de Texto **nano**:

* Posee un único modo de ejecución:
  + Modo Inserción directa
* **Comandos** **principales**; Se activan con teclas de control **Ctrl + Letra**
* **Caracteristicas**:
  + Los comandos se muestran en la parte inferior de la pantalla
  + No requiere aprendizaje de modos especiales (Es intuitivo para usuarios nuevos)
  + Soporta sintaxis coloreada para múltiples lenguajes de programación (-Y o --syntax)

Editor de Texto **mcedit (Midnight Commander)**:

* Posee un único modo de ejecución:
  + Modo Edición directa
* **Comandos** **principales**; Se activan con teclas de función o combinaciones. Ejemplo: F1, F2, Ctrl + U, etc…
* **Caracteristicas**:
  + Es parte del gestor de archivos Midnight Commander (mc), pero puede usarse de forma independiente.
  + Soporta resaltado de sintaxis para varios lenguajes de programación
  + Menú inferior con atajos visibles
  + No requiere modos especiales (Intuitivo para usuarios básicos)

El comando **cat:**

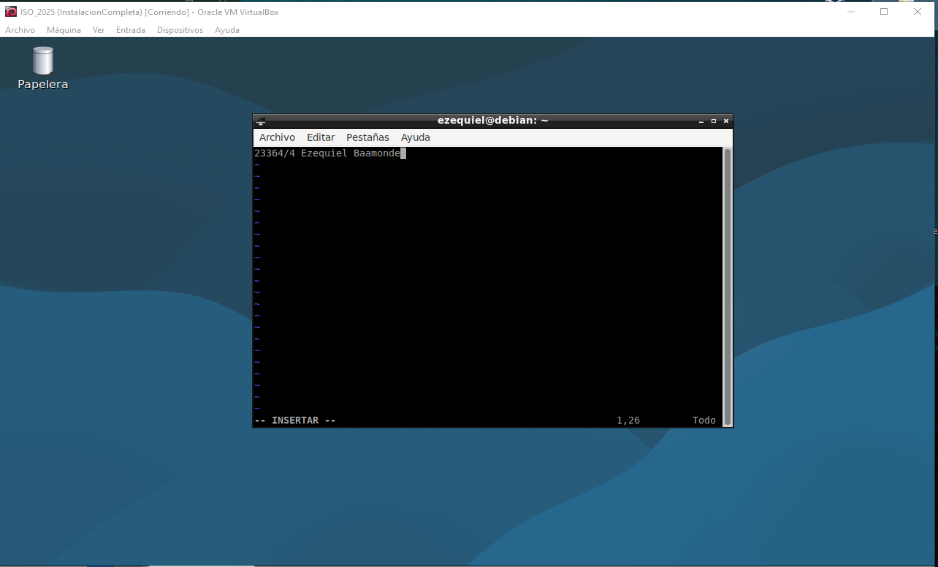
* **Función**: Concatena y muestra el contenido completo de archivos en la salida estándar (Terminal)
* **Uso**: cat archivo.txt
* **Caracteristicas**:
  + Muestra todo el contenido de una vez, sin pausas.
  + Ideal para archivos pequeños
  + No permite navegación o busqueda

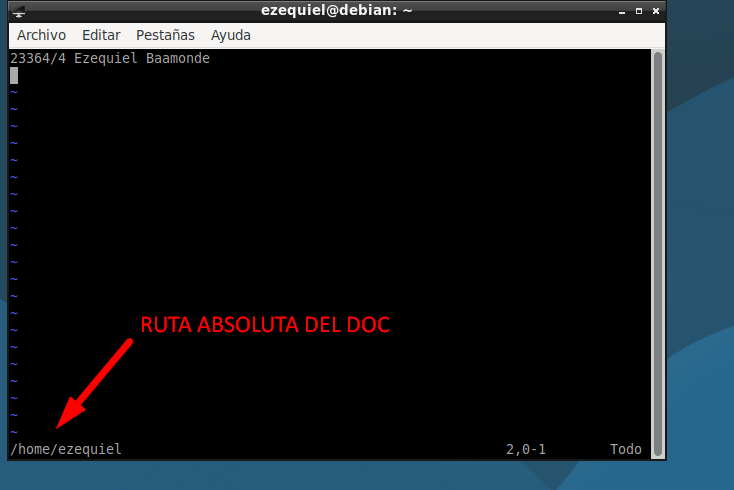
El comando **more**:

* **Función**: Muestra el contenido de archivos página por página
* **Uso**: more archivo.txt
* **Comandos internos:**
  + Enter: Avanza una línea
  + Espacio: Avanza una página
  + Q: Salir
* **Limitaciones**: No permite retroceder hacia atrás en el archivo

El comando **less:**

* **Función:** Mejora de **more**, permite navegación avanzada y busqueda
* **Uso:** less archivo.txt
* **Comandos internos:**
  + Avanzar**:** **Espacio** (Página), **Enter** (Línea), **Flecha abajo**
  + Retroceder: **B** (Página), **Flecha arriba**
  + Busqueda: /termino (buscar hacia adelante la palabra), ?/termino (buscar hacia atrás la palabra)
  + Navegación: **G** (ir al final), **g** (ir al inicio)
  + Salir: **q**
  1. **\*HACER EN PC\***
* Creación de prueba.exe con datos dentro (Insert):





* 1. El comando **file** se utiliza para determinar el **tipo** de un archivo analizando su contenido (No solo la extensión).
* Sintaxis básica: **file** documento.txt | Salida: UTF-8 Unicode text

Al probarlo con distintos archivos noto que cada uno de ellos tiene un tipo de archivo distinto según lo que contiene.

* 1. Funcionalidades y parámetros de cada comando:
* **cd** (Change Directory): Cambiar el directorio de trabajo actual
  + Parámetros comunes:
    - **cd [ruta]** -> Navega a la ruta especificada
    - **cd** o **cd** **~** -> Va al home del usuario
    - **cd** **-** -> Regresa al directorio anterior
    - **cd.**.-> Sube un nivel (Directorio padre)
* **mdkir** (Make Directory): Crear nuevo directorio
  + Parámetros comunes:
    - **mdkir [nombre]** -> Crea un directorio
    - **mkdir -p [ruta/anidada]** -> Crea directorios anidades (Crea padres si no existen)
    - **mkdir -m [permisos] [nombre]** -> Asigna permisos específicos
* **rmdir** (Remove Directory): Eliminar directorios vacios
  + Parámetros comunes:
    - **rmdir [nombre]** -> Elimina un directorio vacio
    - **rmdir -p [ruta/anidada]** -> Elimina directorios anidados si están vacios
* **ln** (Link): Crear enlaces entre archivos (Duros o simbólicos)
  + Parámetros comunes:
    - **ln [origen] [destino]** -> Crea un **enlace duro**
    - **ln -s [origen] [destino]** -> Crea un **enlace simbólico** (acceso directo)
    - **ln -f** -> Fuerza la creación, sobrescribiendo si existe
* **tail**: Muestra las últimas líneas de un archivo.
  + Parámetros comunes:
    - tail [archivo] → Muestra las últimas 10 líneas.
    - tail -n [número] [archivo] → Muestra las últimas n líneas (ej: tail -n 20 log.txt).
    - **tail -f [archivo]** → Sigue el archivo en tiempo real (útil para logs).
* **locate**: Busca archivos en una base de datos preconstruida (rápido, pero no en tiempo real).
  + Parámetros comunes:
    - **locate [patrón]** → Busca coincidencias del patrón.
    - **locate -i [patrón]** → Búsqueda insensible a mayúsculas/minúsculas.
    - **locate -l [número]** → Limita el número de resultados.
* **ls** (List): Lista archivos y directorios.
  + Parámetros comunes:
    - **ls** → Lista el contenido del directorio actual.
    - **ls -l** → Listado largo (detalles: permisos, propietario, tamaño).
    - **ls -a** → Muestra archivos ocultos (que empiezan con.)
    - **ls -h** → Tamaños legibles (ej: "1K", "2M").
    - **ls -t** → Ordena por fecha de modificación (más reciente primero).
* **pwd** (Print Working Directory): Muestra la ruta absoluta del directorio actual.
  + Parámetros comunes:
    - **pwd** → Sin parámetros, muestra la ruta.
    - **pwd** **-P** → Muestra la ruta física (resuelve enlaces simbólicos).
* **cp** (Copy): Copiar archivos o directorios.
  + Parámetros comunes:
    - **cp [origen] [destino]** → Copia un archivo.
    - **cp -r [origen] [destino]** → Copia directorios recursivamente.
    - **cp -i** → Modo interactivo (pregunta antes de sobrescribir).
    - **cp -u** → Copia solo si el origen es más reciente.
* **mv** (Move): Mover o renombrar archivos/directorios.
  + Parámetros comunes:
    - **mv [origen] [destino]** → Mueve o renombra.
    - **mv -i** → Pregunta antes de sobrescribir.
    - **mv -v** → Modo verbose (muestra lo que está haciendo).
* **find**: Busca archivos/directorios en tiempo real (potente pero lento en sistemas grandes).
  + Parámetros comunes:
    - **find [ruta] -name [patrón]** → Busca por nombre (ej: find /home -name "\*.txt").
    - **find [ruta] -type [tipo**] → Filtra por tipo: f (archivo), d (directorio).
    - **find [ruta] -size [+-]tamaño** → Busca por tamaño (ej: find . -size +10M).
    - **find [ruta] -exec** **[comando] {} \;** → Ejecuta un comando en cada resultado.

1. **Comandos:**
   1. Utilizamos el comadno **MKDIR** para la creación del directorio.
   2. Para ingresar a ella utilizamos el comando **CD**.
   3. Utilizando **vim**, **nano** o **mcedit** podemos crear los archivos.

* **Vim isocso.txt | vim isocso.csv**
  1. Utilizamos el comando **LS** para el listado del contenido del directorio.
  2. Utilizamos el comando **PWD** para obtener la ruta absoluta de mi ubicación.
  3. Utilizamos el comando **find [ruta] -name [patrón]** -> (ej: find /home -name "\*.txt").
  4. Utilizamos el comando **df -h** para informar la cantidad de espacio libre en disco.
* df: Reporta el espacio
* -h: Muestra el resultado en formato legible
  1. Utilizamos el comando **who**. Muestra una lista de los usuarios actualmente contectados, junto con la terminal y fecha/hora de inicio de sesión.
* **w:** Alternativa que muestra más detalles de los usuarios
* **users:** Lista solo los nombres sin detalles
* **last:** Muestra el historial de inicios de sesión
  1. Ingresando con nuestro editor de texto (Ejemplo VIM).
* **vim isocso.txt** -> Tipeo “I” y modifico el archivo
  1. Utilizamos **tail** para mostrar las últimas líneas del archivo.

1. **Comandos 3:**

* **man:** Proporciona información detallada sobre comandos, archivos de configuración, llamadas al sistema y bibliotecas. Sería como un “manual” de comandos del kernel.
  + Ej: **man ls | man passwd**
  + Ubicado en /usr/bin/man
* **shutdown:** Apaga, reinicia o suspende el sistema de forma segura.
  + Ej: **shutdown [opciones][tiempo][mensaje]**
  + Parametros:
    - -h: Apaga el sistema (Halt)
    - -r: Reinicia el sistema (reboot)
    - -c: Cancela un apagado programado
    - -p: Apaga y corta la energía (power off)
    - -k: Solo envìa el mensaje de advertencia sin apagar
    - [tiempo]: Especifica cuando ejecutar la acción
    - [mensaje]: Para notificar a los usuarios
  + Sin directorio específico, pero su comportamiento puede verse influenciado por /etc/systemd/system.conf
* **halt:** Detiene el sistema de manera abrupta (sin apagar la energía por defecto). Es un comando de bajo nivel que solo para el kernel.
  + Ej: **halt [opciones]**
  + Parametros:
    - -f: Fuerza el halt sin llamar a shutdown
    - -p: Apaga la energía después de detener el sistema (equivale a poweroff)
    - -w: Solo escribe en /var/log/wtmp sin realmente **haltear**
* **reboot:** Reinicia el sistema de manera inmediata. Al igual que halt, es un comando de bajo nivel.
  + Ej: **reboot [opciones]**
  + Parametros:
    - -f: Fuerza el reinicio sin llamar a shutdown
    - Solo escribe en /var/log/wtmp sin realmente **reiniciar**
  + Ubicación: /sbin/reboot
* **uname:** Muestra información básica sobre el sistema operativo y el hardware en el que se está ejecutando. Es útil para conocer detalles del kernel, arquitectura del procesador y más
  + Ej: **uname [opciones]**
  + Parametros:
    - -s: Muestra el nombre del kernel (por defecto si no se usan opciones).
    - -n: Muestra el nombre del host (nodename).
    - -r: Muestra la versión del kernel.
    - Etc…
  + Ubicación: /usr/bin/uname
* **dmesg:** Muestra y controla el búfer de mensajes del kernel (kernel ring buffer). Este búfer contiene mensajes generados por el kernel durante el arranque y la operación del sistema, incluyendo información sobre hardware, dispositivos, drivers y errores.
  + Ej: **dmesg [opciones]**
  + Parametros:
    - -h: Muestra la salida en formato legible para humanos (con timestamps convertidos).
    - -t: Muestra timestamps
    - -l: Filtra por nivel de mensaje
    - Etc…
  + Ubicación: /usr/bin/dmesg
* **lspci:** Lista todos los dispositivos PCI (y PCIe) conectados al sistema, incluyendo tarjetas gráficas, adaptadores de red, controladores USB, etc. Es esencial para diagnosticar hardware.
  + Ej: **lspci [opciones]**
  + Parametros:
    - -v: Modo verbose (muestra detalles adicionales
    - -vv: Modo muy verbose (aún más detalles)
    - -k: Muestra los controladores del kernel usados por cada dispositivo
    - Etc…
  + Ubicación: /usr/bin/lscpci
* **at:** Programa la ejecución de comandos o scripts en un momento específico del futuro. A diferencia de cron (para tareas recurrentes), at es para ejecuciones únicas.
  + Ej: **at [opciones][hora]**
  + Parametros:
    - -f archivo: Ejecuta los comandos contenidos en un archivo (en lugar de ingresarlos interactivamente)
    - -l: Lista las tareas programadas pendientes (equivale a atq)
    - -d ID: Elimina una tarea programada (equivale a atrm)
    - -m: Envía un correo al usuario cuando la tarea se completa (aunque no haya salida)
    - -v: Muestra la hora de ejecución antes de leer los comandos.
  + Ubicación: /usr/bin/at
* **netstat:** Muestra información sobre conexiones de red, interfaces, tablas de enrutamiento y estadísticas de protocolos. Es útil para diagnosticar problemas de red.
  + Ej: **netstat [opciones]**
  + Parametros:
    - -a: Muestra todas las conexiones y sockets en escucha
    - -t: Muestra solo conexiones **TCP**
    - -u: Muestra solo conexiones **UDP**
    - -n: Muestra direcciones numéricas (evita la resolución de nombres DNS)
    - -l: Muestra solo sockets en **escucha** (listening)
    - Etc…
  + Ubicación: /usr/bin/netsat
* **head:** Muestra las primeras líneas de un archivo o entrada estándar.
  + Ej: **head [opciones] archivo**
  + Parametros:
    - -n N: Muestra las primeras N líneas
    - -c N: Muestra los primeros N bytes.
  + Ubicación: /usr/bin/head
* **tail:** Muestra las últimas líneas de un archivo o entrada estándar.
  + Ej: **tail [opciones] archivo**
  + Parametros:
    - -n N: Muestra las ultimas N líneas
    - -f: Sigue el archivo en tiempo real
    - -c N: Muestra los últimos N bytes.
  + Ubicación: /usr/bin/tail

1. **Procesos**
   1. A

* **top:** Muestra información en tiempo real sobre los procesos en ejecución, el uso de recursos del sistema (CPU, memoria, swap) y una barra de estado resumida. Es una herramienta interactiva para monitorización.
  + Ej: **top [opciones]**
  + Parametros:
    - -d segundos: Intervalo de actualización (ej: -d 2 para 2 segundos).
    - -p PID: Monitoriza solo un proceso específico (Por PID)
    - -u usuario: Muestra solo procesos de un usuario.
    - -n N: Actualiza N veces y luego sale (Modo interactivo)
    - -b: Modo batch (Salida en texto plano, útil para scripts)
    - -H: Muestra hilos (threads) en lugar de procesos
  + Ubicación: /usr/bin/top
* **htop:** Es una versión mejorada e interactiva del comando top. Muestra procesos en tiempo real con una interfaz a color, barras visuales para CPU/memoria y soporte para ratón y scroll vertical/horizontal.
  + Ej: **htop [opciones]**
  + Parametros:
    - -d segundos: Intervalo de actualización (ej: -d 2 para 2 segundos).
    - -p PID: Monitoriza solo un proceso específico (Por PID)
    - -u usuario: Muestra solo procesos de un usuario.
    - -s columna: Ordenar por una columna
    - -c: Modo sin color (Monocromo)
  + Ubicación: /usr/bin/htop
* **ps:** Muestra una instantánea (snapshot) de los procesos actuales del sistema. A diferencia de top o htop (que son interactivos y en tiempo real), ps ofrece información estática en el momento de su ejecución.
  + Ej: **ps [opciones]**
  + Parametros:
    - -aux: Muestra todos los procesos de todos los usuarios con detalles extendidos
    - -ef: Muestra todos los procesos en formato estándar Unix
    - -u usuario: Muestra solo procesos de un usuario.
    - -p PID: Muestra información de un PID específico
    - -c nombre: Busca procesos pro nombre (Ej: ps -C firefox)
    - Etc…
  + Ubicación: /usr/bin/ps
* **pstree:** Muestra los procesos del sistema en forma de árbol jerárquico, ilustrando las relaciones padre-hijo entre procesos. Esto facilita la comprensión de cómo se estructuran los procesos y qué procesos iniciaron otros.
  + Ej: **pstree [opciones]**
  + Parametros:
    - pstree [PID]: Muestra el árbol de procesos a partir de un proceso específico (identificado por su PID)
    - pstree -p: Muestra los PID de cada proceso entre paréntesis
    - pstree -a: Muestra los argumentos de línea de comandos de los procesos
    - Etc…
  + Ubicación: *No viene instalado por defecto en todas las distribuciones. Si no lo posee se debe instalar con “sudo apt install psmisc”. El ejecutable figurará en* /usr/bin/pstree
* **kill:** Envía señales a procesos para controlar su comportamiento. Por defecto, envía la señal TERM (terminación suave), que solicita al proceso que finalice de manera ordenada. También puede forzar la terminación con la señal KILL.
  + Ej: **kill [opciones] <PID>…**
  + Parametros: *-s <señal> o -<señal> 🡪 Envía señal específica*
    - kill -9 1234: Fuerza la terminación del proceso con PID 1234 (señal KILL)
    - kill -TERM 5678: Terminación suave (equivalente al uso por defecto).
    - kill -HUP 9012: Recarga la configuración del proceso (señal HUP).
    - Etc…
  + Ubicación: /bin/kill
* **pgrep:** Busca procesos en ejecución basándose en su nombre u otros atributos y devuelve sus PID (Process ID). Es útil para scripts o para obtener rápidamente el ID de un proceso sin usar herramientas como ps o grep.
  + Ej: **prgrep [opciones] <patrón>…**
  + Parametros:
    - pgrep -l <PID>: Muestra el nombre del proceso junto al PID
    - pgrep -u <usuario>: Filtra procesos por usuario
    - pgrep -x <nombre>: Solo procesos llamados exactamente "<nombre>".
    - Etc…
  + Ubicación: /usr/bin/pgrep
* **pkill:** Envía señales a procesos basándose en su nombre u otros atributos, sin necesidad de especificar su PID. Es la versión "matadora" de pgrep, ya que busca procesos y les envía una señal directamente.
  + Ej: **pkill [opciones] <patrón>…**
  + Parametros (Y ejemplos):
    - pkill -9 firefox: Fuerza la terminación (señal KILL)
    - pkill -f <nombre script>: Mata procesos cuyo comando incluya "<nombredescript>". Ejemplo *"python script.py"*
    - pkill -x <nombre>: Solo procesos llamados exactamente "<nombre>".
    - Etc…
  + Ubicación: /usr/bin/pkill
* **killall:** Envía señales a todos los procesos que coincidan con un nombre específico. Similar a pkill, pero diseñado para actuar sobre grupos de procesos con el mismo nombre.
  + Ej: **pkill [opciones] <nombre\_del\_proceso>…**
  + Parámetros (Y ejemplos):
    - killall -9 apache2: Fuerza la terminación (señal KILL)
    - killall -i <script>: Modo interactivo (pide confirmación antes de terminar cada proceso)
    - killall -r “^**py**.\*”: Mata procesos cuyo nombre empiece con "py" u otro.
    - Etc…
  + Ubicación: /usr/bin/ killall | No suele venir instalado en algunas distros, en caso de instalarse se debe ejecutar ***sudo apt install psmisc***at
* **renice:** Cambia la prioridad de planificación (nice value) de uno o más procesos en ejecución.
  + Parámetros comunes:
    - renice -n <incremento> -p <PID>: Cambia la prioridad del proceso especificado.}
    - renice -n <incremento> -u <usuario>: Cambia la prioridad de todos los procesos de un usuario
    - Etc…
  + Ubicación: /usr/bin/renice
* **xkill:** Cierra ventanas gráficas de forma forzosa seleccionándolas con el cursor.
  + Parámetros comunes:
    - akill: Activa el modo selección (el cursor se convierte en una "X")
    - xkill -id <id\_ventana>: Cierra una ventana específica por su ID.
    - Etc…
  + Ubicación: /usr/bin/xkill | *Si no está instalado, este comando pertenece al paquete xorg-xkill, se instala ejecutando* ***sudo apt install x11-utils***
* **atop:** Monitor avanzado de rendimiento del sistema (similar a top pero con más detalles: disco, red, procesos).
  + Parámetros comunes:
    - atop: Inicia el monitor en tiempo real
    - atop -r <archivo\_log>: Reproduce un log grabado previamente
    - atop -c: Muestra la línea de comandos completa de los procesos
    - Etc…
  + Ubicación: /usr/bin/atop | *No viene preinstalado, se instala ejecutando* ***sudo apt install atop***
* **nice:** Ejecuta un programa con una prioridad de planificación (nice value) específica
  + Parámetros comunes:
    - nice -n <valor> <comando>: Ejecuta el comando con el valor de nice especificado (-20 a 19)
    - Etc…
  + Ubicación: /usr/bin/nice | *Viene preinstalado del paquete coreutils*

1. **Proceso de Arranque SysV init**
   1. Enumeración de pasos:
2. Ejecución del código del BIOS
3. El BIOS ejecuta el POST (Verificación de hardware)
4. El BIOS lee el sector de arranque (MBR)
5. Se carga el bootloader (Gestor de arranque MBC)
6. El bootloader carga el kernel y el initrd (Inital Ram Disk)
7. Se monta el initrd como sistema de archivos raíz y se inicializan componentes esenciales
8. El kernel ejecuta el proceso **inicit** y se desmonta el **initrd**
9. Se lee el /etc/inittab
10. Se ejecutan los scripts apuntados por el runlevel 1
11. El final del *runlevel 1* le indica que vaya al runlevel por defecto
12. Se ejecutan los scripts apuntados por el *runlevel por defecto*
13. El sistema está listo para ser usado
    1. El **proceso INIT** es ejecutado desde */sbin/init*, posee el PID 1. Su objetivo es cargar todos los subprocesos necesarios para el correcto funcionamiento del sistema operativo. Se encarga de montar los filesystems y de hacer disponible los demás dispositivos
    2. Los Runlevels son el modo en que arranca GNU/Linux. Es un proceso de arranque que se divide en niveles. Cada uno de ellos es responsable de iniciar o parar una serie de servicios, ya sea al entrar al Runlevel (Arranque) o al salir de este (Apagado).
    3. Acorde al estándar, existen 7 tipos de Runlevels:

* 0: Halt (Parada o apagado)
* 1: single-user mode (modo monousuario)
* 2: multi-user without network support (multiusuario sin soporte de red)
* 3: multi-user console mode (modo multiusuario en consola)
* 4: N/A (No se utiliza)
* 5: X11 (modo multiusuario con entorno gráfico basado en X.org)
* 6: reboot (Reinicio)

Los runlevel a ejecutar se encuentran definidos en el archivo */etc/inittab*.

No. La mayoría de las distribuciones modernas han reemplazado el sistema tradicional de runlevels (SysV init) por systemd, que introduce un concepto diferente: *targets* (objetivos). Sin embargo, systemd mantiene compatibilidad hacia atrás con los runlevels clásicos mediante alias simbólicos.

* 1. En el archivo */etc/inittab* se almacenan las definiciones de los runlevels con el siguiente formato: **id:runlevels:acción:proceso**
* **id:** identifica la entrada en inittab
* **runlevels:** el/los runlevels en los que se realizan la acción
* **acción:** índica como se ejecutará **proceso**
  + **wait, initdefault, ctrlaltdel, off, respawn, etc…**
* **proceso:** el comando exacto a ejecutar
  1. En los SysV init el cambio entre Runlevels (X --> Y) se realiza mediante el siguiente comando:
  + *init Y* **#Ejemplo: *init 3* (cambia al runlevel 3)**
  + *telinit Y* **#Alternativa equivalente a init**

Los cambios no son permanentes, son temporales. Solo afecta a la sesión actual del sistema. Al reiniciarse, el sistema volverá al runlevel o target predeterminado configurado inicialmente.

Es temporal dado que el *runlevel/target* por defecto se define en el archivo de configuración */etc/inittab***.**  Ejemplo: *id:5:initdefault:*

* 1. Los scripts RC son archivos de **shell script** (generalmente **ubicados** en */etc/init.d/*, */etc/rc.d/*, o directorios similares) que se ejecutan durante el arranque, cambio de runlevel o apagado del sistema. Su nombre suele comenzar con "**S**" (start) o "K**"** (kill) seguido de un número que indica su orden de ejecución (ej: **S10network, K80ssh**).

La **finalidad** de los Scripts RC es la de:

1. **Inicializar servicios:**
   * Iniciar demonios (Servicios) durante el arranque (Ej: Apache, MySQL) durante el arranque
   * Configurar recursos del sistema
2. **Detener servicios:**
   * Finalizar procesos de manera ordenada durante el apagado o cambio de runlevel
3. **Gestionar dependencias:**
   * Asegurar que los servicios se inicien o detengan en el orden correcto (ej: la red debe estar activa antes que un servidor web).
4. **Compatibilidad con runleves:**
   * Ejecutar acciones específicas según el runlevel (Ej: Modo gráfico vs modo consola)

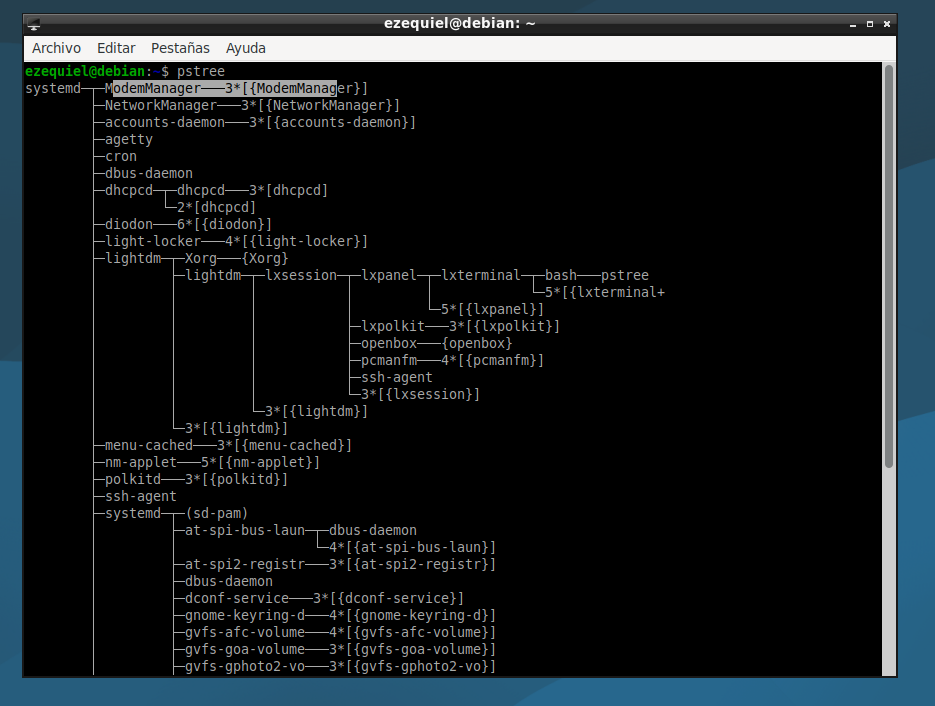
El sistema determina **que scripts ejecutar** basándose en el **runlevel** o **target** actual (estos tienen asociados un conjunto específico de scripts o servicios), en los **archivos** **de** **configuración** (Los directorios */etc/rcX.d/* \**donde X es el runlevel\** contienen enlaces simbólicos a scripts en */etc/init.d/*) y en las **etiquetas** **en** **los** **scripts** (Los scripts RC incluyen metadatos (comentarios) que indican en qué runleveles deben ejecutarse y sus dependencias).

Si, existe un orden para llamarlos… Este es un orden estricto y crítico para garantizar que los servicios se inicien o detengan de manera correcta, respetando las dependencias entre ellos.

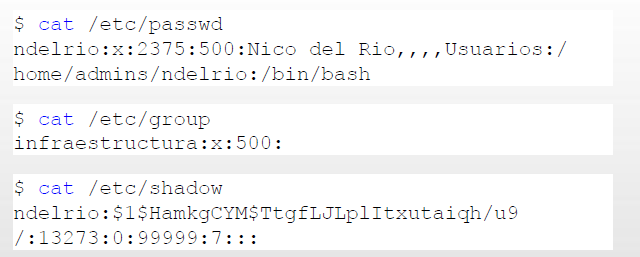
1. **SystemD**
   1. **SystemD** es un sistema de inicio (**init**) de administración de servicios y recursos del sistema. Actualmente se convirtió en el **estándar** de la mayoría de las distribuciones modernas de GNU/Linux. Reemplaza al antiguo sistema **SysV init** e introduce características avanzadas para mejorar la eficiencia, paralelización, el control de arranque y los procesos.
   2. El concepto de **Unit** en SystemD se refiere a “Unidad”. Este es el **bloque fundamental de configuración** que representa un recurso, servicio, o entidad que el sistema debe gestionar. Cada **Unit** (O unidad) se define en un archivo de texto plano (Con extensiones específicas) y permite controlar cómo **systemd** inicia, supervisa y organiza los componentes del sistema.

En síntesis; Las units **son el corazón de systemd**, permitiendo definir de manera declarativa cómo deben comportarse los recursos del sistema. Su diseño modular y basado en dependencias asegura un control preciso sobre el arranque, la supervisión y la organización de servicios.

* 1. El comando **systemctl** en SystemD es una **utilidad** de línea de comandos que **permite** a los usuarios **interactuar** con systemd, el sistema de inicio y gestor de servicios de Linux. **Sirve** para administrar y controlar el estado de los servicios, manejar los estados de arranque del sistema, habilitar o deshabilitar servicios para el inicio automático y supervisar el funcionamiento de las unidades del sistema.
  2. Un **target** (o destino) es un punto de sincronización que agrupa varias unidades de Systemd para **definir** un estado del sistema. Los targets actúan de manera similar a los niveles de ejecución (runlevels) de los sistemas SysV init antiguos, pero **ofrecen** una mayor flexibilidad para organizar servicios y dependencias. Por ejemplo, ***multi-user.target*** inicia los servicios necesarios para un sistema de consola sin interfaz gráfica, mientras que ***graphical.target*** incluye esos servicios más una interfaz gráfica.
  3. Al ejecutar el comando *pstree* se muestran los procesos del sistema en forma de árbol jerárquico, ilustrando las relaciones padre-hijo entre procesos.



1. **Usuarios**
   1. La información de los usuarios se almacena en diferentes archivos de configuración:



* 1. Las siglas **UID** significan **U**ser **ID**entifier. Es un identificador númerico único asignado a cada usuario en el sistema. Su propósito es la de utilizar este tipo de identificador para gestionar permisos y recursos en sistema. Además, determina los privilegios y accesos del usuario.

Las siglas **GID** significan **G**ropu **ID**entifier. Al igual que UID, esto es un identificar numérico, pero asignado a cada GRUPO de usuarios. Su propósito es gestionar permisos colectivos (varios usuarios pueden pertenecer a un mismo grupo) y controlar acceso a archivos, directorios y recursos compartidos.

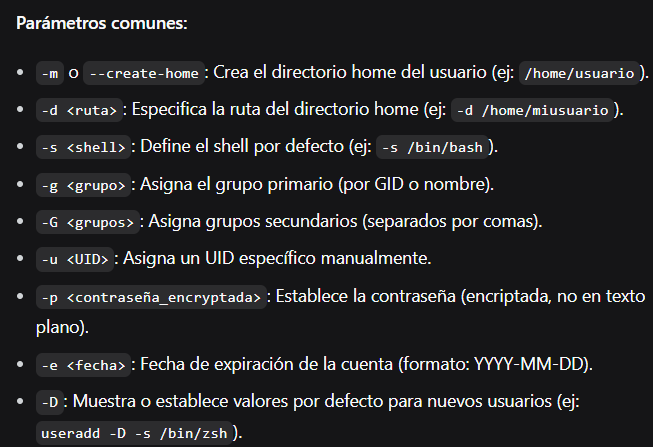
La relación entre ambos es que cada usuario tiene un **UID primario** y puede pertenecer a múltiples grupos (**GIDs secundarios**). El **GID** primario del usuario se almacena en */etc/passwd*, mientras que los grupos secundarios se definen en */etc/group*.

**No pueden coexistir** ***UID****s* iguales en un sistema GNU/Linux, deben ser únicos para cada usuario en un mismo sistema. Si dos usuarios compartieran el mismo **UID**, el sistema no podria distinguirlos, lo que provocaría problemas de seguridad y gestión de recursos.

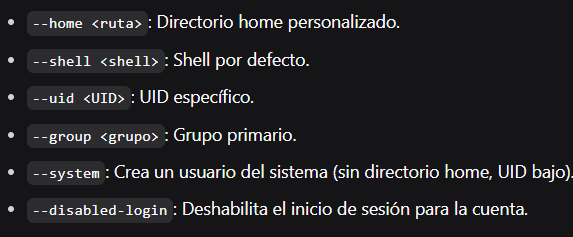
* 1. El usuario **root** es el administrador del sistema (SuperUsuario).

Sí, pueden existir varios usuarios **ROOT** pero no de la manera tradicional. No pueden existir múltiples cuentas con el mismo **UID 0 (que define al usuario root)**, pero hay formas alternativas de lograr privilegios equivalentes o acceso root sin duplicar el UID.

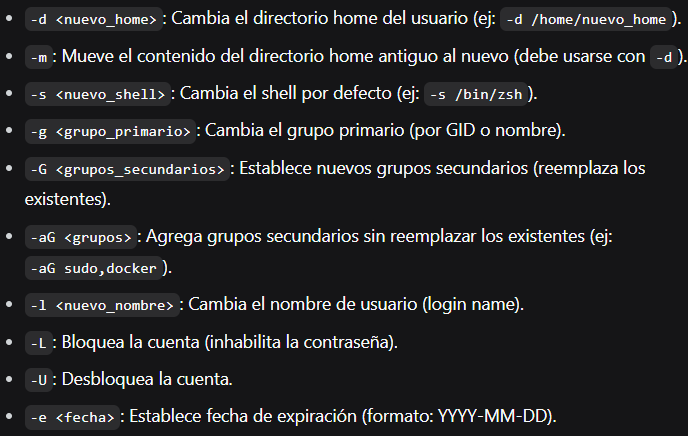
* 1. **Realizar en PC**
  2. Funcionalidad y parámetros de los siguientes comandos:
* **useradd**: Crea un nuevo usuario en el sistema de manera low-level. Es la herramienta básica y directa para agregar usuarios, pero requiere parámetros explícitos para configurar detalles como el directorio home o el shell.
  + **Parámetros:**



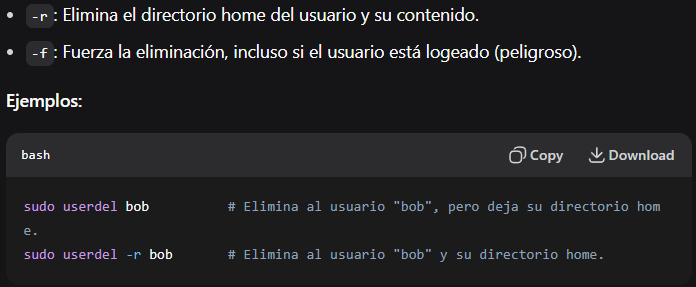
* **adduser**: Herramienta interactiva y amigable que simplifica la creación de usuarios. Automáticamente:
  + Crea el directorio home
  + Establece permisos básicos
  + Solicita contraseña y datos adicionales interactivamente
  + Configura valores por defecto de manera intuitiva
  + **Parámetros:**



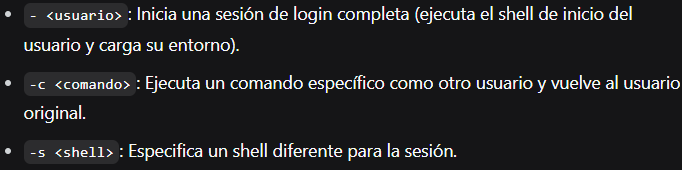
* **usermod**: Modifica las propiedades de un usuario existente en el sistema, como su grupo primario, shell, directorio home, o grupos secundarios.
  + **Parámetros:**



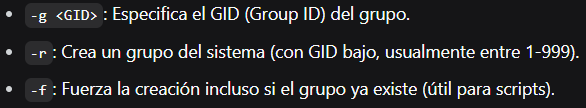
* **userdel**: Elimina un usuario del sistema y, opcionalmente, su directorio home y archivos.
  + **Parámetros:**



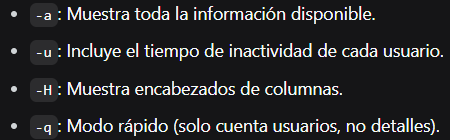
* **su**: Permite cambiar de usuario durante una sesión de terminal. Por defecto, cambia al usuario root si no se especifica otro.
  + **Parámetros:**



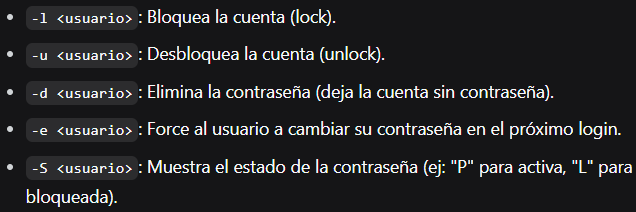
* **groupadd**: Crea un nuevo grupo en el sistema.
  + **Parámetros:**



* **who**: Muestra información sobre los usuarios actualmente logeados en el sistema, incluyendo terminal, fecha/hora de inicio de sesión y dirección IP (si aplica).
  + **Parámetros:**



* **groupdel**: Elimina un grupo existente del sistema.
  + **Restricciones:**
    - No se puede eliminar un grupo si es el grupo primario de algún usuario.
    - Primero debes cambiar el grupo primario de esos usuarios o eliminarlos.
* **passwd**: Cambia la contraseña de un usuario. Root puede cambiar cualquier contraseña; los usuarios solo pueden cambiar la suya.
  + **Parámetros:**



1. **FileSystem y permisos**
   1. Los permisos en un sistema GNU/Linux son definidos sobre archivos y directorios. Existen 3 posibles permisos: **R, W** y **X**. Se suelen expresar en OCTAL acorde a la siguiente tabla:

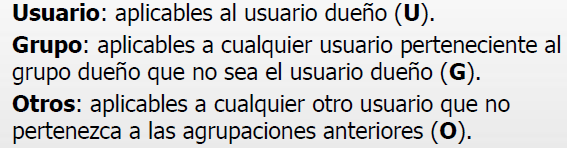
\*Las combinaciones de permisos se expresan como la suma de los valores octales de cada permiso que se desea incluir:

- **W + X =** 2 + 1 = 3

- **R + W** = 4 + 2 = 6

Cada archivo o directorio del sistema de archivos tiene un usuario dueño y un grupo dueño.

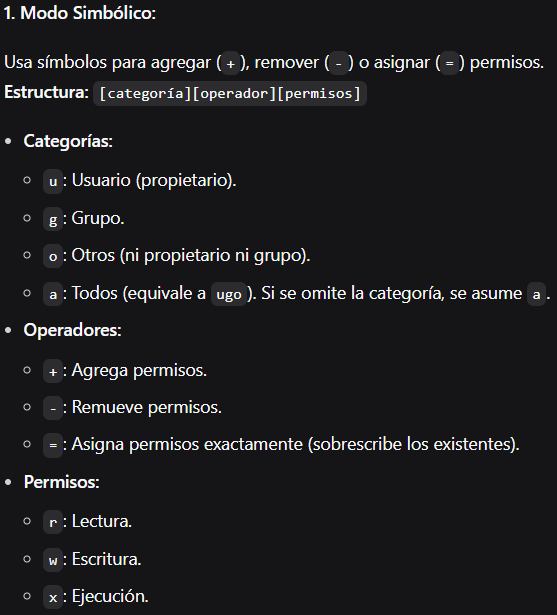
Estos, pueden definirse sobre las siguientes agrupaciones de usuario:



* 1. Funcionalidades y parámetros de:
* **chmod**: **Cambia los permisos** de acceso a archivos o directorios en sistemas GNU/Linux. Controla los permisos de lectura (**r**), escritura (**w**) y ejecución (**x**) para tres categorías de usuarios: propietario (owner), grupo (group) y otros (others).
  + Sintaxis básica:



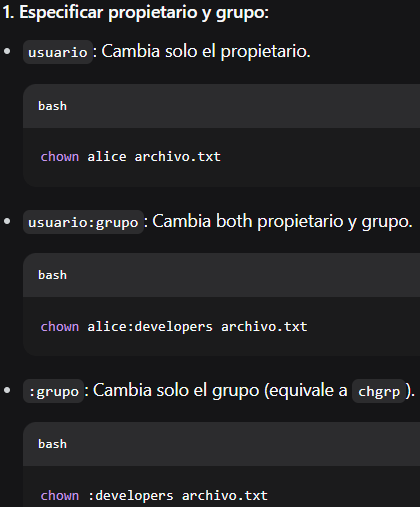
* + Parámetros:



* **chown**: **Cambia** el propietario (owner) y/o grupo de uno o más archivos o directorios. Es esencial para gestionar la propiedad de los recursos del sistema.
  + Sintaxis básica:



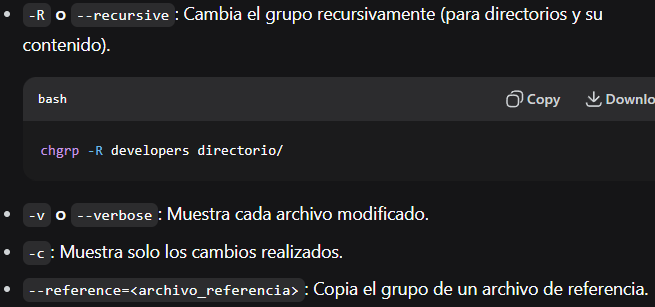
* + Parámetros:



* **chgrp**: Cambia el grupo de uno o más archivos o directorios. Es equivalente a *chown: grupo*, pero más específico para modificar solo el grupo.
  + Sintaxis básica:



* + Parámetros:



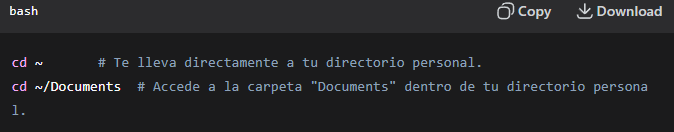
* 1. Se utiliza la notación octal para representar los permisos de archivos y directorios. Usa **tres dígitos octales** (Cada uno entre 0 y 7) para definir permisos de manera compacta y precisa. Cada dígito corresponde a una categoria de usuario (Propietario, grupo y otros). Referencias y valores de cada permiso en la tabla del **INCISO A)**
  2. Si, es posible que un usuario acceda a un archivo sin tener permisos directos en él, pero solamente si tiene **privilegios de root** (Sudo). Uno **normal** no puede acceder a un archivo sin permisos.
     + - **Realizar pruebas en PC**
  3. El **Path Absoluto (Full Path Name)** es la ruta completa que describe la ubicación de un archivo o directorio desde el **directorio raíz (/)**. Siempre comienza con ‘/’ y específica todos los directorios intermedios hasta el elemento destino.
* Ejemplo: Acceder al archivo ‘script.sh’ en el directorio home del usuario **Alice**:
  + Ruta 🡪 */home/alice/script.sh*
* Ejemplo 2: Editar el archivo de hosts del sistema:
  + Ruta 🡪 *sudo nano /etc/hosts*

En cambio, el **Path Relativo (Relative Path Name)** es una ruta que describe la ubicación de un archivo o directorio **RELATIVA al directorio de trabajo actual**. No comienza con ‘/’ y puede usar referencias especiales como **‘.’** (Directorio actual) o **‘..’** (Directorio padre).

* Ejemplo: Acceder a un archivo en el directorio actual:
  + Bash 🡪 *./config.yaml*
* Ejemplo 2: Acceder a un archivo en un subdirectorio:
  + Bash 🡪 *src/main.py*
* Ejemplo 3: Subir un nivel y acceder a un archivo en el directorio padre:
  + Bash 🡪 *../documentos/contrato.pdf*
  1. Con el comando **pwd** -> Muestra la ruta absoluta (path absoluto) del directorio de trabajo actual ***(Print Working Directory).***

Si, existen varias formas de ingresar a una dirección persoal sin escribir el PATH absoluto:

* Usando el **símbolo ~** podemos representar la ruta absoluta del directorio personal (/home/usuario):
  + Ejemplos:



* Usando la variable de entorno **$HOME**, quien almacena la ruta del directorio personal:
  + Ejemplo:



* Solo escribiendo **cd**, sin parámetros nos lleva automáticamente al directorio personal.

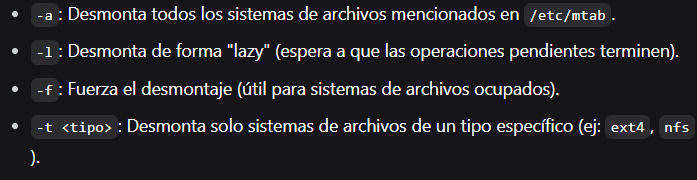
Acceso a otros directorios utilizando esta idea:

* Ejemplos:

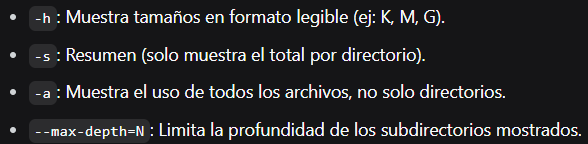


Estos métodos son **portables** y funcionan en cualquier sistema **GNU/Linux**, independientemente del nombre de tu usuario o la estructura del sistema.

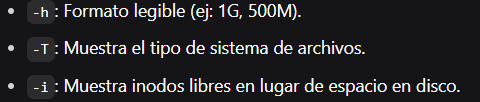
* 1. Funcionalidades y parámetros de:
* **umount**: Desmonta un sistema de archivos previamente montado.
  + Parámetros:



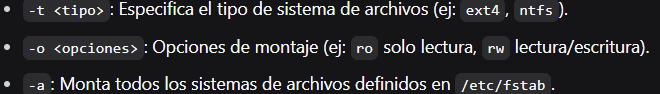
* **du** (Disk Usage): Muestra el espacio utilizado por archivos y directorios.
  + Parámetros:



* **df** (Disk Free): Muestra el espacio libre en los sistemas de archivos montados.
  + Parámetros:



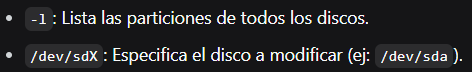
* **mount**: Monta sistemas de archivos en la jerarquía de directorios.
  + Parámetros:



* **mkfs** (Make FileSystem): Crea un sistema de archivos en una partición.
  + Parámetros:



* **fdsik:** Administra particiones de discos (crear, eliminar, modificar). **¡Precaución!:** Usar con cuidado, ya que puede causar pérdida de datos.
  + Parámetros:



* **write**: Envía un mensaje a la terminal de otro usuario logeado.
  + Parámetros:



* **losetup:** Gestiona dispositivos de loop (asocia archivos a dispositivos de bloque).
  + Parámetros:



* **stat**: Muestra información detallada de un archivo o directorio (metadatos).
  + Parámetros:



1. **Procesos**
   1. Que un proceso se esté ejecutando en **Foreground** (Primer Plano) quiere decir que el mismo se ejecuta en la terminal actual y **bloquea** la entrada de comandos hasta que se termina. En cambio, que un proceso se este ejecutando en **Background** (Segundo Plano) quiere decir que el mismo se ejecuta **sin bloquear** la terminal, permitiendo seguir usando la **Shell** para otras tareas.
   2. Para ejecutar un proceso en **BACKGROUND** se agrega el símbolo **&** al final del comando (*comando &)*. Ejemplo: *sleep 60 &* (Ejecuta “sleep 60” en segundo plano)

Para pasar un proceso de **background** a **foreground** se usa el comando **fg (foreground)** seguido del numero de trabajo con **%**.



Para volver a traerlo a **background** se pausa el proceso con **ctrl + z** (Si está en ejecución, sino no) y se reanuda con la instrucción **bg** seguido del número de trabajo con **%**.



* 1. El operador **| (pipe)** se utiliza para **redirigir la salida estándar (stdout)** de un comando hacia la **entrada estándar (stdin)** de otro comando. Esto permite **encadenar múltiples comandos** para procesar datos en flujos (*streams*) de manera eficiente, sin necesidad de archivos temporales.



El operador **PIPE** tiene alguna limitación:

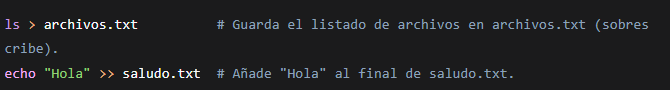
* **Error estándar (stderr):** Por defecto, el PIPE no redirige stderr. Para incluirla:



* **Datos binarios**: Algunos comandos no trabajan bien con pipes si la salida no es texto.
  1. La redirección permite controlar el flujo de entrada y salida de los comandos. Se utiliza para modificar hacia dónde se envía la salida de un comando o de dónde se recibe su entrada.

**Tipos de redirecciones:**

* Redirección de salida estándar (**stdout**):
  + Envia la salida de un comando a un archivo (sobrescribiendo o añadiendo)
  + Sintaxis:
    - **>**: Sobrescribe un archivo
    - **>>**: Añade al final de un archivo
  + **Ejemplo**:



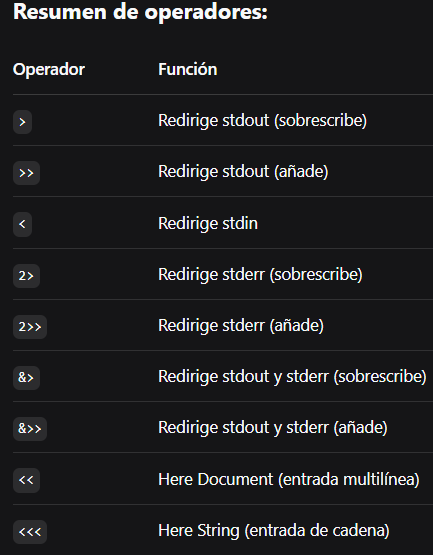
* Redirección de estándar (**stdin**):
  + Envía el contenido de un archivo como entrada a un comando
  + Sintaxis: **<**
  + **Ejemplo:**



* Redirección de error estándar (stderr):
  + Manejar los mensajes de error por separado
  + Sintaxis:
    - **2>**: Sobrescribe un archivo con errores
    - **2>>**: Añade errores al final de un archivo
  + **Ejemplo:**



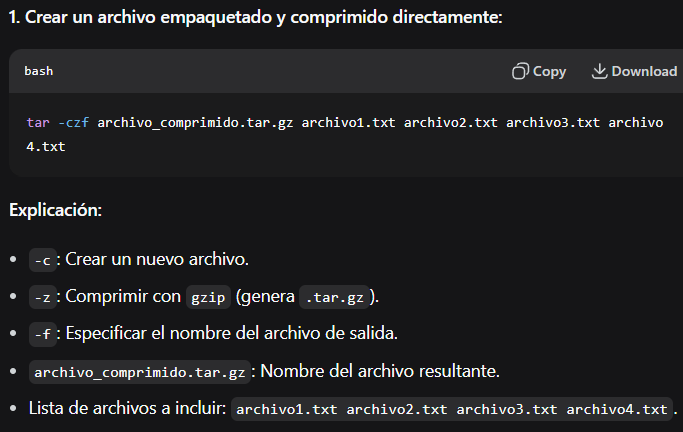
* Etc…

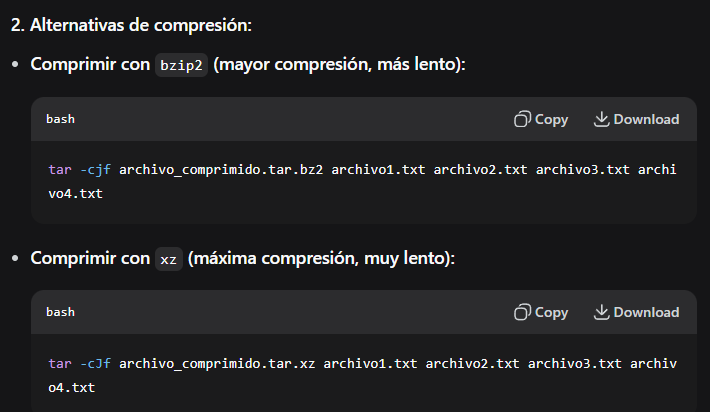


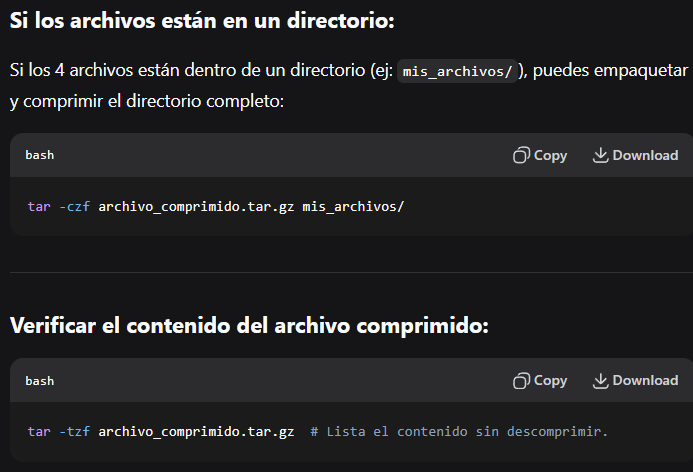
1. **Comandos de Linux, funcionalidad y parámetros:**
   1. **Empaquetar** **archivos** se refiere a la acción de combinar múltiples archivos y directorios en un único archivo contenedor (llamado "paquete" o "archivo empaquetado"), sin aplicar compresión en este paso. El objetivo principal es facilitar el transporte, almacenamiento o backup de grupos de archivos manteniendo su estructura jerárquica y metadatos (permisos, ownership, timestamps).

La **herramienta** más común para empaquetar es **tar (Tape Archive)**. Su sintaxis básica es: *tar -****cf*** *paquete.tar* ***archivo1.txt directorio/*** *#Crea el paquete.*

* ***-c****: Crear archivo.*
* ***-f****: Especificar nombre del archivo*
  1. \* Hacer en maquina virtual
  2. Para comprimir 4 archivos en un único archivo comprimido en GNU/Linux, se sigue un proceso de dos pasos: **empaquetas** los archivos (**usando TAR**) y **luego comprimir** el paquete (usando **gzip, bzip2, xz** etc…). Sin embargo, **tar** permite hacer ambos pasos a la vez con opciones integradas.

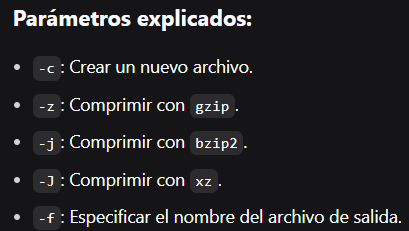






* 1. Si. En GNU/Linux, se puede empaquetar y comprimir múltiples archivos en un solo paso usando el comando **tar** con opciones de compresión integradas. No es necesario empaquetar primero y comprimir después.





* **Tar (Tape Archive):** Empaquetar y desempacar archivos en un único archivo contenedor (tarball), con opción de comprimir.
* **grep (Global Regular Expression Print):** Buscar patrones de texto en archivos o entrada estándar usando expresiones regulares.
* **gzip (GNU Zip)**: Comprimir archivos individuales (reemplaza el original con extensión .gz).
* **zgrep:** Equivalente a **grep** pero para archivos comprimidos con gzip (.gz). No requiere descomprimir el archivo primero.
* **wc (Word Count):** Contar líneas, palabras y caracteres en archivos o entrada estándar.