

# Bitácora Semanal: Administración de Sistemas Unix Bitácora Semanal - Administración de Unix

Equipo Alpine White

[2026-02-09 Mon]

## Contents

<b>1 Información General</b>	<b>2</b>
<b>2 Sesión 1: 3 de Febrero 2026</b>	<b>2</b>
2.1 Conceptos nuevos . . . . .	2
2.2 Conceptos de repaso . . . . .	4
2.3 Máximas . . . . .	4
2.4 Sección libre . . . . .	5
<b>3 Sesión 2: 4 de Febrero 2026</b>	<b>5</b>
3.1 Conceptos nuevos . . . . .	5
3.2 Conceptos de repaso . . . . .	5
3.3 Máximas . . . . .	5
3.4 Sección libre . . . . .	5
<b>4 Sesión 3: 5 de Febrero 2026</b>	<b>5</b>
4.1 Conceptos nuevos . . . . .	5
4.2 Conceptos de repaso . . . . .	6
4.3 Máximas . . . . .	6
4.4 Sección libre . . . . .	6
<b>5 Sesión 4: 6 de Febrero 2026</b>	<b>7</b>
5.1 Conceptos nuevos . . . . .	7
5.2 Conceptos de repaso . . . . .	12
5.3 Máximas . . . . .	12
5.4 Sección libre . . . . .	12
<b>6 Comandos de Administración en Linux</b>	<b>12</b>
<b>7 Montaje de Particiones, EFI y Administración del Sistema</b>	<b>12</b>
<b>8 1. Identificación y Montaje de Particiones</b>	<b>13</b>
<b>9 2. EFI (EFI System Partition)</b>	<b>13</b>
9.1 Características principales: . . . . .	13
9.2 Proceso de arranque en sistemas UEFI: . . . . .	13

<b>10 3. Dispositivos tipo /dev/sdX y contexto de “sd3”</b>	<b>14</b>
<b>11 4. Buenas prácticas: uso de usuario con privilegios sudo</b>	<b>14</b>
11.1 Justificación técnica: . . . . .	14
11.2 Buena práctica recomendada: . . . . .	15
<b>12 5. Sistema de archivos Btrfs</b>	<b>15</b>
<b>13 5.1 Opción compress=zstd:1 0 0</b>	<b>15</b>
13.1 Componentes: . . . . .	15
<b>14 5.2 Subvolúmenes (subvol)</b>	<b>16</b>
<b>15 6. Archivo /etc/fstab</b>	<b>16</b>
<b>16 7. Conceptos Complementarios Propuestos</b>	<b>17</b>

## 1 Información General

- **Semana:** Del 3 de Febrero 2026 al 6 de Febrero 2026
- **Equipo:**
  - Arreguín Salgado Gael Emiliano
  - Gramer Muñoz Omar Fernando
  - López Pérez Mariana
  - Nieto Gallegos Isaac Julián

## 2 Sesión 1: 3 de Febrero 2026

Esta sesión fue una introducción general a la clase y a conceptos iniciales de Linux.

### 2.1 Conceptos nuevos

- **GRUB (bootloader):** GRUB (GNU GRand Unified Bootloader) es un **cargador de arranque**; es el programa que aparece al iniciar y se encarga de cargar el sistema operativo (o una ISO) y pasarle el control.
  - **Ventoy y su GRUB:** Ventoy no usa exactamente el mismo GRUB que el del sistema instalado; incluye y arranca con su propia variante de **GRUB2** para poder gestionar el menú y el arranque de múltiples ISOs desde una sola USB.
  - **Consola de GRUB en Ventoy:** En el menú de Ventoy/GRUB, al presionar la tecla C se abre la **consola** (tipo terminal) de GRUB.
    - \* En esta consola podemos inspeccionar y explorar qué ve el bootloader (discos/particiones/archivos) y consultar variables o configuración.
    - \* Comandos:

```

ls
help
set
ls (hd0,gpt1) /
cat (hd0,gpt1)/boot/grub/grub.cfg

```

- \* Esto sirve para tipo depurar, inspeccionar por qué algo no arranca como debe, confirmar rutas, o entender cómo Ventoy/GRUB está resolviendo la configuración.

- El comando ls nos permite listar los contenidos del directorio en el que nos encontramos actualmente. Usando los parámetros -la, podemos colocarlos en forma de lista y mostrar archivos ocultos para tener una vista más general y completa del directorio. Por ejemplo:

```
ls -la
```

```

total 456
drwxr-xr-x 1 marlop18 marlop18 132 Feb 18 19:22 .
drwxr-xr-x 1 marlop18 marlop18 44 Feb 10 09:22 ..
-rw-r--r-- 1 marlop18 marlop18 28160 Feb 18 19:20 bitacoral.org
-rw-r--r-- 1 marlop18 marlop18 259064 Feb 18 19:22 bitacoral.pdf
-rw-r--r-- 1 marlop18 marlop18 35070 Feb 18 19:22 bitacoral.tex
drwxr-xr-x 1 marlop18 marlop18 142 Feb 18 19:24 .git
-rw-r--r-- 1 marlop18 marlop18 25 Feb 18 19:22 test.txt
-rw-r--r-- 1 marlop18 marlop18 133687 Feb 10 09:22 ventoy_boot.jpg

```

- Ventoy es una herramienta que nos permite poder arrancar varias imágenes ISO desde una misma memoria USB. Se instala sobre la memoria USB, y recursivamente busca las imágenes ISO que contenga para mostrarlas en el menú de arranque. Se nos queda de tarea para la próxima semana traer una memoria USB con Ventoy instalado.



- Existen dos formas de referenciar un archivo en Linux: Mediante su ruta relativa y mediante su ruta absoluta. Generalmente una ruta relativa empieza por un ./ o ../, mientras que una ruta absoluta empieza por /, ya que es la ruta completa al archivo desde la raíz del sistema de archivos. Por ejemplo:

```
echo este es el mismo archivo > test.txt
cat /home/mrtaichi/Documents/escuela/adminUnix/test.txt # Ruta absoluta
cat ./test.txt # Ruta relativa
```

este es el mismo archivo

- LAN? (en proceso)
- En Linux, 2.3.1. Esto incluye a los dispositivos, ya que la carpeta /dev guarda representaciones en archivo de todos los dispositivos que están conectados a la computadora. Esto incluye desde los mismos dispositivos de almacenamiento, hasta cosas más “virtuales” como las terminales y pseudoterminales del sistema.
- Dispositivo psar? (en proceso)
- Gparted es una herramienta con interfaz gráfica para crear y modificar particiones en un disco.

## 2.2 Conceptos de repaso

- **Proceso de arranque (boot process):** Secuencia general: BIOS/UEFI → bootloader (GRUB) → kernel → init/systemd. Punto clave: el sistema operativo **no existe** mientras estamos en GRUB; solo existe firmware con bootloader.
- **Bootloader vs Kernel:** GRUB no es el sistema operativo ni el kernel. Su única responsabilidad es localizar, cargar el kernel (y el initramfs) en memoria y cederle el control. No ejecuta procesos ni gestiona recursos del sistema.
- **Separación de responsabilidades:** Ventoy delega completamente el arranque a GRUB y evita modificar el disco. Ejemplo de buen diseño: cada componente hace una sola cosa bien definida.
- **Acceso a disco antes del kernel:** GRUB puede leer sistemas de archivos (FAT, EXT, ISO) sin drivers del kernel.
- **Lectura de archivos sin montaje:** El bootloader accede directamente a estructuras del sistema de archivos. Montar un FS es una abstracción que pertenece al kernel, no al bootloader.

## 2.3 Máximas

### 2.3.1 Todo es un archivo

En los sistemas derivados de Unix, como Linux

## **2.4 Sección libre**

# **3 Sesión 2: 4 de Febrero 2026**

### **3.1 Conceptos nuevos**

- Estuvimos hablando de la importancia de la redundancia para preservar la información. Como ejemplo usamos el sistema RAID, que utiliza varios discos duros para crear replicación de información y tolerancia a fallos.
- La necesidad principal de tener redundancia en la información es para no tener un sólo punto de falla en nuestros sistemas que pueda fallar. Si alguna parte de nuestro sistema falla, siempre existe otro componente listo para tomar el lugar y mantener el sistema funcionando.
- Generalmente la emulación es más lenta que la virtualización. El primer sistema de WSL usaba emulación de Linux, ahora el segundo utiliza virtualización.
- El propósito de que cada quien utilice una distribución distinta es demostrar la máxima: “3.3.1”. Vamos a observar como -en esencia- todas las distribuciones de Linux tienen tantas cosas en común que podemos hacer cualquier cosa en cualquier distro.
- Distrowatch es un sitio web en el que se pueden observar todas las distribuciones de Linux y muchos datos acerca de ellas. Lo que resulta interesante es que incluso un ranking de cuáles son las distros más usadas hasta el momento.
- SCSI (que se pronuncia “scuzzy”) es una interfaz de intercambio de datos que se utiliza sobre todo en servidores y computadoras de alto desempeño. Hablando del sistema RAID, es generalmente sobre esta interfaz sobre la que se construyen varios controladores RAID que están en la industria.

### **3.2 Conceptos de repaso**

### **3.3 Máximas**

#### **3.3.1 Todos los Linux son iguales**

En cualquier distribución de Linux podemos -a final de cuentas- hacer cualquier cosa. Todas funcionan esencialmente para lo mismo, y aunque evidentemente hay diferencias, se puede trabajar con ellas sin problema.

## **3.4 Sección libre**

# **4 Sesión 3: 5 de Febrero 2026**

### **4.1 Conceptos nuevos**

- Unix nace en 1969 como un sistema operativo desarrollado en Bell Labs, al principio con fines académicos y distribuido para varias universidades. De ahí nace BSD cuando se vuelve comercial
- Y de ahí también nace Linux, como un kernel Unix-like desarrollado en 1991 por Linus Torvalds, el cual se termina fusionando con GNU para formar un sistema operativo completo.

- Un proceso es un programa que se está ejecutando en el sistema y constantemente le está pidiendo recursos al kernel. Los procesos se comunican con el kernel a través de syscalls

## 4.2 Conceptos de reposo

## 4.3 Máximas

## 4.4 Sección libre

Para entender mejor el funcionamiento de un proceso, se me ocurrió meterme al código fuente de Linux y, con ayuda de la IA, aprender acerca de la estructura que conforma un proceso como tal.

Primero había que ubicarla, y después de indagar un rato en la documentación me pude dar cuenta de que el término usado en el kernel para referirse a los procesos es una task (o tarea). Esto debido a que esta estructura sirve tanto para procesos de programas como para los hilos de un mismo programa. La estructura que buscamos es entonces `task_struct`, la cual se encuentra declarada en `include/linux/sched.h`

Como el código que define esta estructura es muy extenso, es aquí donde me tuve que servir de la IA para entenderle. Le pasé el archivo y le pedí una explicación sobre cada uno de los componentes de esta estructura. Este fue el resultado:

Entiendo perfectamente, la ‘`struct task_struct`’ es intimidante porque es el “corazón” de Linux y contiene prácticamente todo lo que el kernel sabe sobre un proceso. Basado en el archivo ‘`sched.h`’ que compartiste, vamos a despedazarla en partes lógicas para que sea más fácil de digerir.

Piensa en la ‘`task_struct`’ como el **expediente completo** de una persona en una oficina gubernamental; tiene su nombre, su familia, su dinero, sus propiedades y qué está haciendo en este momento.

### ### 1. Estado y Propósito (¿Qué está haciendo?)

Lo primero que el kernel necesita saber es si el proceso puede correr o si está esperando algo.

**‘`_state`**: Define si el proceso está ‘`TASK_RUNNING`’ (listo para usar la CPU), ‘`TASK_INTERRUPTIBLE`’ (dormiendo pero puede despertar con una señal) o en otros estados como ‘`TASK_DEAD`’.

**‘`stack`**: Un puntero al stack del kernel para este proceso específico. **‘`flags`**: Banderas que indican atributos especiales (por ejemplo, si es un hilo del kernel o si está saliendo).

### ### 2. Identificación (¿Quién es?)

**‘`pid`**: El **Process ID** único que ves cuando usas comandos como ‘`top`’. **‘`tgid`**: El **Thread Group ID**. En Linux, para el hilo principal, el PID y el TID son iguales. Si creas más hilos en ese proceso, todos compartirán el mismo TID pero tendrán PIDs distintos. **‘`comm[TASK_COMMLEN]`**: El nombre del ejecutable (máximo 16 caracteres).

### ### 3. El Planificador (¿Cuándo le toca la CPU?)

Esta sección decide la prioridad del proceso.

**‘`prio`**, ‘`static_prio`, ‘`normal_prio`’’: Las distintas prioridades que el planificador usa para decidir a quién darle la CPU. **‘`se`**, ‘`rt`, ‘`dl`’’: Estas son sub-estructuras (`‘sched_entity, ‘‘sched_rtentity, etc.) que contienen los datos específicos para diferentes tipos de procesos (normales, de tiempo real, etc.).`

### ### 4. Relaciones Familiares (¿Quién lo creó?)

Linux organiza los procesos en un árbol genealógico.

\*'parent': Puntero al proceso padre. \*'children': Una lista de todos los procesos hijos que este proceso ha creado. \*'sibling': Enlaces a sus "hermanos" (otros procesos creados por el mismo padre).

### ### 5. Recursos (¿Qué posee?)

Aquí es donde la estructura se conecta con el resto del sistema.

\*'mm': Puntero a 'struct mm\_struct'. Aquí reside toda la información de la memoria virtual del proceso (sus páginas, segmentos, etc.). \*'files': Puntero a 'struct files\_struct'. Es la tabla que guarda todos los archivos abiertos (tus descriptores de archivo 0, 1, 2, etc.). \*'fs': Información del sistema de archivos, como el directorio actual ('pwd'). \*'cred': Las credenciales de seguridad (quién es el dueño: UID, GID).

### ### 6. Señales (¿Qué le han dicho?)

\*'signal' y 'sighand': Manejan qué debe hacer el proceso cuando recibe señales como un 'SIGKILL' o 'SIGTERM'. \*'pending': Una lista de señales que han llegado pero que el proceso aún no ha procesado.

**Resumen rápido para no volverse loco:** Si estás programando en el kernel y tienes un puntero a un proceso (usualmente llamado 'p' o 'current'), lo más común es que solo te interesen 'p->pid', 'p->comm', 'p->mm' (memoria) o 'p->files' (archivos). El resto son detalles técnicos para que el kernel gestione la magia por debajo.

¿Hay alguna de estas secciones que te interese explorar más a fondo con el código que pasaste?

De esto ya pude entender mejor cómo funciona el tema de los procesos en Linux y la relación de parentesco que debe existir entre todos ellos que los lleva al proceso inicial (systemd).

## 5 Sesión 4: 6 de Febrero 2026

### 5.1 Conceptos nuevos

- Linux tiene un directorio llamado /tmp el cual es volátil. Ahí se guardan todos los archivos que se necesitan para el funcionamiento de los programas y SO durante la sesión, pero que no son necesarios de conservarse y por lo tanto se pueden borrar en el siguiente apagado. En Windows existe la misma implementación en una carpeta llamada Temp
- nohup es un comando que nos permite ejecutar procesos y evitar que mueran cuando el padre muera.

man nohup

NOHUP (1)

User Commands

NOHUP(1)

NAME

nohup - run a command immune to hangups, with output to a non-tty

SYNOPSIS

```
nohup COMMAND [ARG] ...
nohup OPTION
```

#### DESCRIPTION

Run COMMAND, ignoring hangup signals.

--help display this help and exit

--version

output version information and exit

If standard input is a terminal, redirect it from an unreadable file. If standard output is a terminal, append output to 'nohup.out' if possible, '\$HOME/nohup.out' otherwise. If standard error is a terminal, redirect it to standard output. To save output to FILE, use 'nohup COMMAND > FILE'.

Your shell may have its own version of nohup, which usually supersedes the version described here. Please refer to your shell's documentation for details about the options it supports.

#### Exit status:

125 if the nohup command itself fails

126 if COMMAND is found but cannot be invoked

127 if COMMAND cannot be found

- the exit status of COMMAND otherwise

#### AUTHOR

Written by Jim Meyering.

#### REPORTING BUGS

Report bugs to: bug-coreutils@gnu.org

GNU coreutils home page: <<https://www.gnu.org/software/coreutils/>>

General help using GNU software: <<https://www.gnu.org/gethelp/>>

Report any translation bugs to <<https://translationproject.org/team/>>

#### COPYRIGHT

Copyright © 2025 Free Software Foundation, Inc. License GPLv3+: GNU General Public License version 3 or later <<https://gnu.org/licenses/gpl.html>>.

This is free software: you are free to change and redistribute it. There is NO WARRANTY, to the extent permitted by law.

#### SEE ALSO

Full documentation <<https://www.gnu.org/software/coreutils/nohup>> or available locally via: info '(coreutils) nohup invocation'

- Con el comando `ps tree` podemos ver un árbol de los procesos en ejecución actualmente en nuestro sistema. Esto nos da una vista muy clara de las relaciones de parentesco de todos estos.

`pstree`

```

|      |-code---10*[{code}]
|      |-code---15*[{code}]
|      |-code---17*[{code}]
|      |-code---16*[{code}]
|      `--38*[{code}]
|-dbus-broker-lau---dbus-broker
|-dconf-service---3*[{dconf-service}]
|-deja-dup---5*[{deja-dup}]
|-deja-dup-monito---4*[{deja-dup-monito}]
|-evolution-addre---6*[{evolution-addre}]
|-evolution-alarm---7*[{evolution-alarm}]
|-evolution-calen---9*[{evolution-calen}]
|-evolution-sourc---4*[{evolution-sourc}]
|-2*[gjs---11*[{gjs}]]
|-gnome-calendar---7*[{gnome-calendar}]
|-gnome-keyring-d---4*[{gnome-keyring-d}]
|-gnome-session-c---{gnome-session-c}
|-gnome-session-s---4*[{gnome-session-s}]
|-gnome-shell-+-Xwayland---15*[{Xwayland}]
|          |-bwrap---bwrap---glycin-image-rs---2*[{glycin-image}]
|          |-mutter-x11-fram---16*[{mutter-x11-fram}]
|          `--34*[{gnome-shell}]
|-gnome-shell-cal---6*[{gnome-shell-cal}]
|-gnome-text-edit---11*[{gnome-text-edit}]
|-goa-daemon---3*[{goa-daemon}]
|-goa-identity-se---3*[{goa-identity-se}]
|-gsd-a11y-settin---4*[{gsd-a11y-settin}]
|-gsd-color---4*[{gsd-color}]
|-gsd-datetime---4*[{gsd-datetime}]
|-gsd-disk-utilit---3*[{gsd-disk-utilit}]
|-gsd-housekeepin---4*[{gsd-housekeepin}]
|-gsd-keyboard---4*[{gsd-keyboard}]
|-gsd-media-keys---4*[{gsd-media-keys}]
|-gsd-power---5*[{gsd-power}]
|-gsd-print-notif---3*[{gsd-print-notif}]
|-gsd-printer---3*[{gsd-printer}]
|-gsd-rfkill---3*[{gsd-rfkill}]
|-gsd-screensaver---3*[{gsd-screensaver}]
|-gsd-sharing---4*[{gsd-sharing}]
|-gsd-smartcard---4*[{gsd-smartcard}]
|-gsd-sound---4*[{gsd-sound}]
|-gsd-usb-protect---4*[{gsd-usb-protect}]
|-gsd-wwan---4*[{gsd-wwan}]
|-gsd-xsettings---4*[{gsd-xsettings}]
|-gvfs-afc-volume---4*[{gvfs-afc-volume}]
|-gvfs-goa-volume---3*[{gvfs-goa-volume}]
|-gvfs-gphoto2-vo---3*[{gvfs-gphoto2-vo}]
|-gvfs-mtp-volume---3*[{gvfs-mtp-volume}]

```

```

| -gvfs-udisks2-vo---4*[{gvfs-udisks2-vo}]
| -gvfsd---gvfsd-dnssd---3*[{gvfsd-dnssd}]
| | -gvfsd-network---4*[{gvfsd-network}]
| | -gvfsd-recent---3*[{gvfsd-recent}]
| | -gvfsd-trash---4*[{gvfsd-trash}]
| | -gvfsd-wsdd---python3
| | | -3*[{gvfsd-wsdd}]
| | | -3*[{gvfsd}]
| -gvfsd-fuse---6*[{gvfsd-fuse}]
| -gvfsd-metadata---3*[{gvfsd-metadata}]
| -ibus-daemon---ibus-dconf---4*[{ibus-dconf}]
| | -ibus-engine-sim---3*[{ibus-engine-sim}]
| | -ibus-extension---5*[{ibus-extension-}]
| | | -3*[{ibus-daemon}]
| -ibus-portal---3*[{ibus-portal}]
| -ibus-x11---10*[{ibus-x11}]
| -keyboxd
| -kgx---zsh---emacs---bash---pstree
| | -bwrap---bwrap---glycin-svg---2*[{glycin-svg}]
| | | -8*[{emacs}]
| | -zsh
| | | -10*[{kgx}]
| -localsearch-3---7*[{localsearch-3}]
| -nautilus---13*[{nautilus}]
| -obexd
| -pipewire---2*[{pipewire}]
| -pipewire-pulse---2*[{pipewire-pulse}]
| -wireplumber---5*[{wireplumber}]
| -xdg-desktop-por---6*[{xdg-desktop-por}]
| | -2*[xdg-desktop-por---5*[{xdg-desktop-por}]]
| -xdg-document-po---fusermount3
| | | -6*[{xdg-document-po}]
| -xdg-permission---3*[{xdg-permission-}]
| | -2*[zsh---gitstatusd---16*[{gitstatusd}]]
| | | -4*[zsh]
|-systemd-journal
|-systemd-logind
|-systemd-machine
|-systemd-timesyn---{systemd-timesyn}
|-systemd-udevd
|-systemd-userdbd---3*[{systemd-userwor}]
|-udisksd---6*[{udisksd}]
|-upowerd---3*[{upowerd}]
`-wpa_supplicant

```

- De aquí podemos observar una vez más como el proceso más importante y del que nacen todos los procesos en espacio de usuario es el proceso systemd. Además, observamos algunas cosillas interesantes como los procesos que generan nuestro entorno de escritorio, pipewire

para el sonido, etc.

## 5.2 Conceptos de repaso

- Me parece que nohup ya había sido visto?

## 5.3 Máximas

## 5.4 Sección libre

- **Semana:** Del 9 de Febrero al 13 de Febrero 2026

Durante la semana vimos una variedad de comandos que nos permiten ver la configuración que hay dentro de bas

# 6 Comandos de Administración en Linux

Definiciones de Comandos de Administración en Linux

### Comando

---

dmesg \

Muestra el registro del kernel (mensajes del sistema), incluyendo detección de hardware, particiones, errores de

fdisk -l /dev/sdf

p (en fdisk)

n (en fdisk)

gpart

smart

efibootmgr

df -l

df -lh

mount /dev/sda3 /mnt

umount /mnt

mount -o subvol=home /dev/sda3 /mnt

mount -bind /dev /mnt/dev

mount -bind /sys /mnt/sys

less /etc/fstab

less /etc/mtab

cat /etc/os-release

chroot /mnt bash

# 7 Montaje de Particiones, EFI y Administración del Sistema

Durante esta semana se abordaron conceptos fundamentales relacionados con la arquitectura de almacenamiento en sistemas Unix/Linux, el proceso de arranque, el montaje de particiones y buenas prácticas administrativas.

---

## **8 1. Identificación y Montaje de Particiones**

En clase analizamos cómo identificar las particiones del sistema y montarlas manualmente utilizando el directorio /mnt como punto temporal de montaje.

El directorio /mnt está definido por el FHS (Filesystem Hierarchy Standard) como un punto destinado a montajes temporales realizados por el administrador.

El proceso general consistió en:

- Identificar las particiones con herramientas como fdisk o lsblk.
  - Reconocer la partición raíz (/), la partición /boot y la partición /boot/efi.
  - Montar manualmente las particiones para inspeccionar su contenido.
  - Analizar la estructura interna del sistema de archivos.
- 

## **9 2. EFI (EFI System Partition)**

La EFI System Partition (ESP) es una partición especial utilizada en sistemas modernos que emplean UEFI (Unified Extensible Firmware Interface) en lugar del BIOS tradicional.

### **9.1 Características principales:**

- Formateada generalmente en FAT32.
- Contiene los cargadores de arranque (.efi).
- Se monta comúnmente en /boot/efi.
- Es requerida en sistemas con tabla de particiones GPT.

### **9.2 Proceso de arranque en sistemas UEFI:**

1. Encendido del equipo.
2. Ejecución del firmware (UEFI).
3. Localización de la partición EFI.
4. Carga del bootloader (ej. GRUB).
5. Selección y carga del kernel.
6. Inicio del sistema operativo.

La EFI permite mayor flexibilidad, soporte para discos mayores a 2TB y mecanismos como Secure Boot.

---

## **10 3. Dispositivos tipo /dev/sdX y contexto de “sd3”**

En Linux, los dispositivos de almacenamiento se representan como block devices dentro del directorio /dev.

Ejemplos:

- /dev/sda → Primer disco detectado
- /dev/sdb → Segundo disco
- /dev/sda1 → Primera partición del disco sda
- /dev/sda3 → Tercera partición del disco sda

Cuando se mencionó “sd3”, se hacía referencia probablemente a /dev/sda3, que corresponde a una partición específica del disco principal. Esta partición puede contener:

- La raíz del sistema (/)
- Un subvolumen en Btrfs
- El directorio /home
- Otro sistema de archivos independiente

En sistemas modernos es común que la raíz esté en /dev/sda3 utilizando Btrfs.

---

## **11 4. Buenas prácticas: uso de usuario con privilegios sudo**

Una recomendación importante fue evitar trabajar directamente como root.

### **11.1 Justificación técnica:**

En sistemas Unix existe separación de privilegios:

- Usuario normal → permisos limitados.
- Root (UID 0) → control total del sistema.

Trabajar siempre como root implica riesgos:

- Eliminación accidental de archivos críticos.
- Modificación indebida de configuraciones.
- Mayor impacto ante errores humanos.

## **11.2 Buena práctica recomendada:**

- Crear un usuario administrador.
- Asignarlo al grupo sudo o wheel.
- Elevar privilegios únicamente cuando sea necesario usando sudo.

Esto sigue el principio de:

Principio de mínimo privilegio (Principle of Least Privilege)

Este principio es ampliamente recomendado en comunidades técnicas, documentación oficial y foros especializados en administración Linux.

---

## **12 5. Sistema de archivos Btrfs**

Se revisó el uso del sistema de archivos Btrfs, el cual es moderno y avanzado, diseñado para ofrecer:

- Copy-on-Write (CoW)
  - Snapshots
  - Subvolúmenes
  - Compresión
  - Integridad de datos
- 

## **13 5.1 Opción compress=zstd:1 0 0**

La opción:

`compress=zstd:1`

es una opción de montaje en Btrfs que habilita compresión usando Zstandard.

### **13.1 Componentes:**

- `compress` → activa compresión.
- `zstd` → algoritmo Zstandard.
- `:1` → nivel de compresión (1 = bajo, más rápido).

Ventajas:

- Reduce uso de espacio en disco.
- Mejora rendimiento en ciertos escenarios (menos I/O).

- Transparente para el usuario.

En /etc/fstab suele verse algo como:

UUID=xxxxxx / btrfs defaults,compress=zstd:1 0 0

Los últimos dos valores (0 0) indican:

- Dump (respaldo) → generalmente 0.
  - Orden de chequeo con fsck → generalmente 0 en Btrfs.
- 

## 14 5.2 Subvolúmenes (subvol)

Un subvolumen en Btrfs es una división lógica dentro del mismo sistema de archivos.

No es una partición física, sino una estructura interna.

Ejemplo común:

- Subvolumen @ → raíz del sistema
- Subvolumen @home → directorio /home

Ventajas:

- Permite snapshots independientes.
- Separación lógica sin particionar físicamente.
- Facilita recuperación y administración.

Montaje ejemplo:

```
mount -o subvol=@home /dev/sda3 /mnt
```

---

## 15 6. Archivo /etc/fstab

El archivo /etc/fstab define qué sistemas de archivos se montan al inicio.

Incluye:

- UUID
- Punto de montaje
- Tipo de sistema de archivos
- Opciones de montaje
- Flags de dump y fsck

Es clave en administración Unix para garantizar persistencia de configuración.

---

## **16 7. Conceptos Complementarios Propuestos**

A continuación se listan temas estructurales para desarrollo posterior dentro de la materia Administración de Unix.

Esta semana permitió comprender cómo Unix/Linux organiza el almacenamiento desde el nivel físico hasta el nivel lógico, cómo se monta el sistema de archivos y cómo el proceso de arranque interactúa con la EFI y el bootloader.

Asimismo, se reforzaron buenas prácticas administrativas como el uso de usuarios con privilegios controlados y la comprensión del sistema de archivos Btrfs y sus capacidades avanzadas.

Estos conocimientos constituyen una base esencial para la administración segura y eficiente de sistemas Unix.