tp2

1: Editor de textos:

(a) Nombre al menos 3 editores de texto que puede utilizar desde la línea de comandos.

Nvim, bash, nano

(b)¿En qué se diferencia un editor de texto de los comandos cat, more o less? Enumere
los modos de operación que posee el editor de textos vi.

En un editor de texto, podes leer y escribir, tambien se tiene un manejo mas facil dentro del archivo.. En cambio cat,more, less muestran los archivos en modo lectura

cat: muestra el archivo, normalmente de forma concatenada
more: muestra el contenido del archivo, solo en modo lectura

more: muestra el archivo y permite navegar dentro de el con la barra espaciadora para avanzar una pagina o “enter” para avanzar una linea

less: permite moverse con las flechitas y tiene un comando para buscar palabras clave: /palabraclave

(c) Nombre los comandos más comunes que se le pueden enviar al editor de textos vi

El editor de textos `vi` (y su versión mejorada, `vim`) tiene una amplia gama de comandos que permiten a los usuarios editar texto de manera eficiente. A continuación, se enumeran algunos de los comandos más comunes que se utilizan en `vi`:

Comandos de Navegación

1. `h`: Mueve el cursor una posición a la izquierda.

2. `j`: Mueve el cursor una línea hacia abajo.
3. `k`: Mueve el cursor una línea hacia arriba.
4. `l`: Mueve el cursor una posición a la derecha.
5. `w`: Mueve el cursor al comienzo de la siguiente palabra.
6. `b`: Mueve el cursor al comienzo de la palabra anterior.
7. `e`: Mueve el cursor al final de la palabra actual.
8. `0`: Mueve el cursor al comienzo de la línea actual.
9. `$`: Mueve el cursor al final de la línea actual.
10. `gg`: Mueve el cursor al principio del archivo.
11. `G`: Mueve el cursor al final del archivo.
12. `Ctrl-u`: Desplaza hacia arriba media pantalla.
13. `Ctrl-d`: Desplaza hacia abajo media pantalla.
14. `H`: Mueve el cursor a la parte superior de la pantalla.
15. `M`: Mueve el cursor al medio de la pantalla.
16. `L`: Mueve el cursor a la parte inferior de la pantalla.

Comandos de Inserción y Edición

1. `i`: Entra en el modo de inserción antes del cursor.
2. `I`: Entra en el modo de inserción al comienzo de la línea actual.
3. `a`: Entra en el modo de inserción después del cursor.
4. `A`: Entra en el modo de inserción al final de la línea actual.
5. `o`: Abre una nueva línea debajo de la línea actual y entra en el modo de inserción.
6. `O`: Abre una nueva línea encima de la línea actual y entra en el modo de inserción.
7. `r`: Reemplaza el carácter bajo el cursor.
8. `R`: Entra en el modo de reemplazo, reemplazando texto en línea hasta que se presione `Esc`.
9. `x`: Elimina el carácter bajo el cursor.

10. `dd`: Elimina (corta) la línea actual.
11. `yy`: Copia (yank) la línea actual.
12. `p`: Pega el texto después del cursor o la línea siguiente.
13. `P`: Pega el texto antes del cursor o la línea anterior.
14. `u`: Deshace la última acción.
15. `Ctrl-r`: Rehace la última acción deshecha.
16. `c` **+ movimiento**: Cambia (corta y entra en el modo de inserción) la parte del texto especificada por el movimiento.
17. `s`: Elimina el carácter bajo el cursor y entra en modo de inserción.

Comandos de Línea de Comando

1. `:w`: Guarda el archivo actual.
2. `:q`: Sale de `vi`.
3. `:wq` o `:x`: Guarda y sale de `vi`.
4. `:q!`: Sale de `vi` sin guardar los cambios.
5. `:e` **nombre_de_archivo**: Abre el archivo especificado para editar.
6. `:r` **nombre_de_archivo**: Inserta el contenido de un archivo en la posición del cursor.
7. `:/patrón`: Busca hacia adelante el patrón especificado.
8. `:?patrón`: Busca hacia atrás el patrón especificado.
9. `:s/patrón/reemplazo`: Sustituye el primer patrón encontrado en la línea actual por el reemplazo especificado.
10. `:%s/patrón/reemplazo/g`: Sustituye todos los patrones en todo el archivo.

Estos comandos cubren las operaciones básicas y avanzadas que puedes realizar en `vi`. Aprender estos comandos te permitirá ser más eficiente en la edición de texto desde la línea de comandos

2: Proceso de Arranque SystemV (https://github.com/systeminit/si):

(a) Enumere los pasos del proceso de inicio de un sistema GNU/Linux, desde que se prende

la PC hasta que se logra obtener el login en el sistema

El proceso de inicio de un sistema GNU/Linux, desde el encendido de la computadora hasta la aparición de la pantalla de inicio de sesión (login), implica varios pasos que son críticos para cargar el sistema operativo y preparar el entorno de usuario. Aquí se detallan estos pasos:

1. Encendido y POST (Power-On Self-Test)

- **Encendido**: Cuando se enciende la computadora, la fuente de alimentación proporciona energía a todos los componentes del sistema.
- **POST**: La BIOS (Basic Input/Output System) o UEFI (Unified Extensible Firmware Interface) realiza una serie de pruebas de diagnóstico conocidas como Power-On Self-Test (POST) para verificar que el hardware básico (memoria, teclado, etc.) funciona correctamente.

2. BIOS/UEFI y Cargador de Arranque (Boot Loader)

- **BIOS/UEFI**: Después del POST, la BIOS o UEFI busca un dispositivo de arranque (disco duro, SSD, USB, etc.) que contenga un sector de arranque (boot sector) válido. En sistemas modernos, UEFI se utiliza más comúnmente, proporcionando capacidades más avanzadas como el arranque seguro (Secure Boot).
- **Cargador de Arranque (Boot Loader)**: El BIOS/UEFI carga el primer sector de arranque del dispositivo (MBR - Master Boot Record en sistemas BIOS, o EFI System Partition en sistemas UEFI), que contiene el cargador de arranque (bootloader). Ejemplos de cargadores de arranque son **GRUB (GRand Unified Bootloader)** y **LILO (Linux Loader)**.
    - **GRUB** es el más común y tiene la capacidad de presentar un menú para seleccionar diferentes sistemas operativos o modos de arranque.

3. Carga del Núcleo (Kernel)

- **Cargar el Núcleo**: El cargador de arranque carga el núcleo de Linux (kernel) en la memoria. El núcleo es el componente central del sistema

operativo que gestiona el hardware y proporciona servicios básicos a los programas.

- **Transferencia de Control**: Una vez cargado, el control se transfiere al núcleo de Linux, que comienza su proceso de inicialización.

4. Inicialización del Núcleo

- **Inicialización de Controladores y Dispositivos**: El núcleo inicializa todos los controladores de hardware necesarios para interactuar con los dispositivos del sistema (como discos duros, tarjetas de red, USB, etc.).
- **Montaje del Sistema de Archivos Root**: El núcleo monta el sistema de archivos raíz (`/`) que contiene los archivos y directorios necesarios para que el sistema funcione.
- **Ejecutar el proceso `init`**: El núcleo carga y ejecuta el proceso inicial (`init`). Este es el primer proceso que se ejecuta en un sistema Linux y es el ancestro de todos los procesos que se ejecutarán después.

5. Sistema de Inicialización (init system)

- **Sistema de Init**: Históricamente, el sistema de inicialización más utilizado era **SysVinit**. Sin embargo, muchas distribuciones modernas de Linux utilizan **systemd**, que es un sistema de inicialización más nuevo y avanzado.
- **Ejecución de Scripts de Inicio**: El sistema de inicialización ejecuta una serie de scripts de inicio (`/etc/rc.d/`, `/etc/init.d/` o unidades de servicio en systemd) para configurar el sistema y preparar el entorno de usuario. Esto incluye:
  - Configuración de la red.
  - Montaje de otros sistemas de archivos (como `/home`, `/var`, etc.).
  - Inicio de servicios y demonios (como el servidor SSH, servicios de base de datos, etc.).
  - Configuración de dispositivos adicionales.

6. Mostrar el Prompt de Login

- **Lanzamiento de Getty**: Finalmente, el sistema de inicialización lanza el proceso `getty` o `agetty`, que prepara el terminal para el inicio de sesión

del usuario. Este proceso muestra el prompt de login en la pantalla (o en un terminal virtual en sistemas sin interfaz gráfica).

- **Login**: Cuando el usuario introduce sus credenciales de inicio de sesión, `getty` lanza el shell de usuario o un gestor de inicio de sesión gráfico (como **LightDM**, **GDM**, **SDDM** o **KDM**).

7. Entorno de Usuario

- **Cargar Entorno de Escritorio (Opcional)**: Si el sistema está configurado para utilizar una interfaz gráfica, el gestor de inicio de sesión (display manager) lanzará el entorno de escritorio (como GNOME, KDE, XFCE, etc.) una vez que el usuario se autentique.
- **Sistema Listo para Usar**: Una vez que el entorno de usuario (CLI o GUI) está cargado, el sistema está listo para que el usuario comience a interactuar.

  Estos pasos representan el proceso típico de inicio en una distribución GNU/Linux, que puede variar ligeramente dependiendo de la configuración específica del hardware y del software del sistema.

(b)Proceso INIT. ¿Quién lo ejecuta? ¿Cuál es su objetivo?

El proceso `init` es ejecutado por el **núcleo (kernel) de Linux** después de que se haya cargado completamente en la memoria y haya terminado su proceso de inicialización básica.

Una vez que el núcleo está en funcionamiento y ha montado el sistema de archivos raíz, busca y ejecuta el programa `init`. En la mayoría de los sistemas Linux, `init` es el primer proceso que se inicia (tiene el **PID 1**) y, por lo tanto, se considera el proceso padre de todos los demás procesos en el sistema.

¿Cuál es el objetivo del proceso `init`?

El objetivo principal del proceso `init` es:

1. **Inicialización del Sistema**: `init` es responsable de inicializar y preparar el entorno del sistema operativo para que sea operativo y funcional para los usuarios. Esto incluye:
   - Montar sistemas de archivos adicionales.
   - Configurar dispositivos de hardware.

- Establecer configuraciones de red.
- Iniciar servicios y demonios esenciales para el funcionamiento del sistema (como servidores web, servidores de bases de datos, y otros servicios de red).

2. **Gestión de Procesos**: `init` se encarga de iniciar todos los procesos de fondo necesarios para el sistema operativo y su correcto funcionamiento. Esto puede incluir tanto servicios del sistema como demonios y procesos de usuario.
   - En sistemas basados en **SysVinit**, `init` lee archivos de configuración como `/etc/inittab` y scripts de inicio en directorios como `/etc/init.d/` y `/etc/rc.d/`.
   - En sistemas modernos basados en **systemd**, `init` es reemplazado por `systemd`, que gestiona procesos a través de unidades de servicio (service units) y otros tipos de unidades (como mount units y socket units).
3. **Cambiar Niveles de Ejecución (Runlevels)**: En sistemas que usan **SysVinit**, `init` gestiona los "niveles de ejecución" (runlevels), que determinan qué servicios y demonios deben estar activos en un momento dado. Por ejemplo:
   - Runlevel 0: Apagar el sistema.
   - Runlevel 1: Modo de usuario único (modo de recuperación).
   - Runlevel 3: Modo multiusuario sin entorno gráfico.
   - Runlevel 5: Modo multiusuario con entorno gráfico.
   - Runlevel 6: Reiniciar el sistema.
   - En `systemd`, los runlevels son reemplazados por "targets", que proporcionan una forma más flexible y modular de gestionar el estado del sistema (por ejemplo, `multi-user.target`, `graphical.target`).
4. **Mantenimiento del Sistema**: `init` también es responsable de monitorear procesos en el sistema. Si un proceso que fue iniciado por `init` se termina, `init` puede reiniciarlo automáticamente o tomar las medidas necesarias para mantener el sistema en un estado coherente.

En resumen, el proceso `init` o su equivalente moderno (`systemd`) es fundamental para la gestión del sistema operativo Linux, ya que garantiza

que el sistema se inicialice correctamente y permanezca operativo, gestionando tanto los servicios esenciales como los procesos de usuario.

(c) RunLevels

¿Qué son?

Los **RunLevels** (o niveles de ejecución) son estados operativos predefinidos en los sistemas basados en **SysVinit**, un sistema de inicialización tradicional en Linux. Cada RunLevel define un conjunto de servicios y procesos que se deben iniciar o detener en ese estado particular del sistema.

¿Cuál es su objetivo?

El objetivo de los RunLevels es proporcionar diferentes configuraciones del sistema que puedan activarse en función de las necesidades del administrador del sistema o del usuario. Por ejemplo, se pueden definir RunLevels para:

- Modo de usuario único (para tareas de mantenimiento).
- Modo multiusuario sin entorno gráfico (para servidores).
- Modo multiusuario con entorno gráfico (para estaciones de trabajo).
- Apagar o reiniciar el sistema.

(d) Niveles de Ejecución (RunLevels) Según el Estándar

¿A qué hace referencia cada nivel de ejecución según el estándar?

Según el estándar SysVinit, los RunLevels comunes son:

- **RunLevel 0**: Apagar el sistema.
- **RunLevel 1**: Modo de usuario único (recuperación o mantenimiento).
- **RunLevel 2**: Modo multiusuario sin servicios de red (definido de manera distinta según la distribución).
- **RunLevel 3**: Modo multiusuario con servicios de red (sin interfaz gráfica).
- **RunLevel 4**: No está definido y se deja para uso personalizado.
- **RunLevel 5**: Modo multiusuario con entorno gráfico (X11).
- **RunLevel 6**: Reiniciar el sistema.

¿Dónde se define qué RunLevel ejecutar al iniciar el sistema operativo?

El RunLevel que se ejecuta al iniciar el sistema operativo se define en el archivo `/etc/inittab`. Este archivo contiene la configuración de inicialización, incluyendo el RunLevel predeterminado que el sistema debe utilizar al arrancar.

¿Todas las distribuciones respetan estos estándares?

No, no todas las distribuciones respetan estos estándares de RunLevels. Aunque los RunLevels son bastante consistentes en sistemas basados en **SysVinit**, algunas distribuciones los implementan de manera diferente o utilizan sistemas de inicialización modernos como **systemd**, que no utilizan RunLevels tradicionales y en su lugar utilizan **targets** que son más flexibles y específicos.

(e) Archivo `/etc/inittab`

¿Cuál es su finalidad?

El archivo `/etc/inittab` es utilizado en sistemas basados en **SysVinit** para definir la configuración del proceso de inicialización, incluyendo:

- El RunLevel predeterminado al inicio.
- Las acciones a ejecutar en ciertos eventos o RunLevels.
- La configuración de terminales y otros procesos de mantenimiento.

¿Qué tipo de información se almacena en él?

El archivo almacena información sobre:

- RunLevel predeterminado y su descripción.
- Qué procesos deben iniciarse, reiniciarse o detenerse en cada RunLevel.
- Configuraciones específicas del sistema que se deben ejecutar al iniciar el sistema o al cambiar de RunLevel.

¿Cuál es la estructura de la información que en él se almacena?

La estructura básica de una línea en `/etc/inittab` es:

```
arduinoCopiar código
id:runlevels:action:process
```

- `id`: Identificador único para la entrada.
- `runlevels`: Lista de RunLevels en los que la entrada es válida.
- `action`: La acción que debe realizarse (por ejemplo, `respawn`, `wait`, `once`).
- `process`: El comando o proceso que se ejecutará.

(f) Cambiar de RunLevel

Suponga que se encuentra en el RunLevel `<X>`. Indique qué comando(s) ejecutaría para cambiar al RunLevel `<Y>`.

Para cambiar al RunLevel `<Y>`, utilizarías el comando:

```
bashCopiar código
init <Y>
```

Por ejemplo, para cambiar al RunLevel 3, se usaría:

```
bashCopiar código
init 3
```

¿Este cambio es permanente? ¿Por qué?

No, este cambio no es permanente. El comando `init <Y>` cambia el RunLevel solo para la sesión actual. Cuando el sistema se reinicia, volverá al RunLevel predeterminado definido en `/etc/inittab`. Para cambiar el RunLevel de inicio permanentemente, se debe editar el archivo `/etc/inittab` y modificar el RunLevel predeterminado.

(g) Scripts RC

¿Cuál es su finalidad?

Los **scripts RC** (scripts de inicio o de ejecución) son utilizados para iniciar o detener servicios y configurar el entorno del sistema cuando el sistema cambia de RunLevel o se inicializa.

¿Dónde se almacenan?

Generalmente, los scripts RC se almacenan en directorios como `/etc/init.d/`, `/etc/rc.d/`, o en subdirectorios específicos de RunLevel como `/etc/rc0.d/`, `/etc/rc1.d/`, etc.

Cuando un sistema GNU/Linux arranca o se detiene, ¿cómo determina qué script ejecutar ante cada acción?

Cuando el sistema cambia de RunLevel o se inicia, el proceso de inicialización (`init`) consulta los directorios de RunLevel (`/etc/rc#.d/` donde `#` es el número del RunLevel) para determinar qué scripts ejecutar. Los scripts en estos directorios están nombrados con un prefijo que indica el orden de ejecución y la acción:

- `S##script`: Inicia (Start) un servicio. El número `##` indica el orden de ejecución.
- `K##script`: Detiene (Kill) un servicio.

¿Existe un orden para llamarlos? Justifique.

Sí, hay un orden para llamarlos, determinado por el prefijo numérico en los nombres de los scripts (`S##` o `K##`). Este orden es importante para asegurar que los servicios se inicien y se detengan en la secuencia correcta, respetando las dependencias entre ellos. Por ejemplo, un servidor de bases de datos debería iniciarse después de que los sistemas de archivos estén montados y la red esté configurada.

En resumen, los scripts RC están cuidadosamente organizados y numerados para garantizar que el sistema se inicie y se detenga de manera ordenada y coherente.

3.SystemD(https://github.com/systemd/systemd)

(a) ¿Qué es systemd?

systemd es un sistema de inicialización y administrador de servicios para sistemas operativos Linux. Fue diseñado para mejorar la eficiencia del proceso de inicio y la gestión de servicios en comparación con el sistema de inicialización tradicional **SysVinit**. Introducido por primera vez en 2010 por Lennart Poettering y Kay Sievers, systemd reemplaza a SysVinit en muchas distribuciones Linux modernas debido a sus ventajas en términos de velocidad, eficiencia y características adicionales.

Algunas características clave de systemd son:

- **Inicio paralelizado**: Permite iniciar múltiples servicios en paralelo, reduciendo significativamente el tiempo de arranque del sistema.
- **Soporte para dependencias explícitas**: systemd puede manejar dependencias de servicios, asegurando que los servicios se inicien en el orden correcto.
- **Journald**: Un sistema de registro centralizado que reemplaza al syslogd tradicional.
- **Gestión de unidades (Units)**: systemd utiliza "units" (unidades) para gestionar diversos recursos del sistema, incluyendo servicios, dispositivos, puntos de montaje, sockets, y más.

(b) ¿A qué hace referencia el concepto de Unit en systemd?

En systemd, una **"Unit" (unidad)** es un archivo de configuración que define cómo se gestiona un recurso específico del sistema. Cada unidad tiene un tipo específico que determina su propósito y funcionalidad. Los tipos de unidades más comunes incluyen:

- **Service Unit (`.service`)**: Define un servicio del sistema (por ejemplo, un servidor web o un demonio SSH).
- **Target Unit (`.target`)**: Agrupa otras unidades y define estados del sistema (similares a los RunLevels en SysVinit).
- **Mount Unit (`.mount`)**: Controla la montura de sistemas de archivos.
- **Socket Unit (`.socket`)**: Define un socket que se puede usar para activar servicios bajo demanda.
- **Timer Unit (`.timer`)**: Define temporizadores que pueden ejecutar comandos o activar servicios en horarios específicos.
- **Device Unit (`.device`)**: Representa dispositivos de hardware y permite gestionar su estado.

Cada unidad tiene un archivo de configuración en `/etc/systemd/system/`, `/usr/lib/systemd/system/`, o en otros directorios específicos de systemd, y estos archivos terminan con una extensión correspondiente al tipo de unidad.

(c) ¿Para qué sirve el comando `systemctl` en systemd?

El comando `systemctl` es la principal herramienta de línea de comandos para interactuar con systemd. Se utiliza para administrar y controlar el estado de las unidades en un sistema que usa systemd. Algunas de las funciones más comunes de `systemctl` incluyen:

- **Iniciar, detener, reiniciar servicios**:
  - `systemctl start <nombre_del_servicio>`: Inicia un servicio.
  - `systemctl stop <nombre_del_servicio>`: Detiene un servicio.
  - `systemctl restart <nombre_del_servicio>`: Reinicia un servicio.
  - `systemctl reload <nombre_del_servicio>`: Recarga la configuración de un servicio sin detenerlo.
- **Habilitar o deshabilitar servicios para que se inicien al arrancar**:
  - `systemctl enable <nombre_del_servicio>`: Habilita un servicio para que se inicie al arrancar el sistema.
  - `systemctl disable <nombre_del_servicio>`: Deshabilita un servicio para que no se inicie al arrancar.
- **Mostrar el estado de los servicios y otras unidades**:
  - `systemctl status <nombre_del_servicio>`: Muestra el estado actual de un servicio.
  - `systemctl list-units`: Muestra una lista de todas las unidades cargadas actualmente.
- **Reiniciar, apagar, suspender o hibernar el sistema**:
  - `systemctl reboot`: Reinicia el sistema.
  - `systemctl poweroff`: Apaga el sistema.
  - `systemctl suspend`: Suspende el sistema.
  - `systemctl hibernate`: Hiberna el sistema.

(d) ¿A qué hace referencia el concepto de target en systemd?

En systemd, un **"target"** es una unidad que agrupa otras unidades para alcanzar un estado específico del sistema. Los targets son utilizados para representar un estado del sistema y pueden ser considerados como una forma moderna de los RunLevels en SysVinit, aunque son mucho más flexibles.

Algunos targets comunes incluyen:

- `default.target`: El target por defecto en el que arranca el sistema, generalmente es un alias para `graphical.target` o `multi-user.target`.
- `graphical.target`: Proporciona un entorno gráfico (inicia el servidor gráfico y el gestor de ventanas).
- `multi-user.target`: Proporciona un entorno multiusuario sin entorno gráfico (similar al RunLevel 3 de SysVinit).
- `rescue.target`: Proporciona un entorno de recuperación (similar al RunLevel 1 de SysVinit, pero con más servicios).
- `emergency.target`: Proporciona un entorno de emergencia con muy pocos servicios, solo el mínimo necesario para reparar el sistema.

Los targets permiten iniciar o detener grupos de servicios y otros recursos del sistema de manera organizada y eficiente.

(e) Comando `pstree`

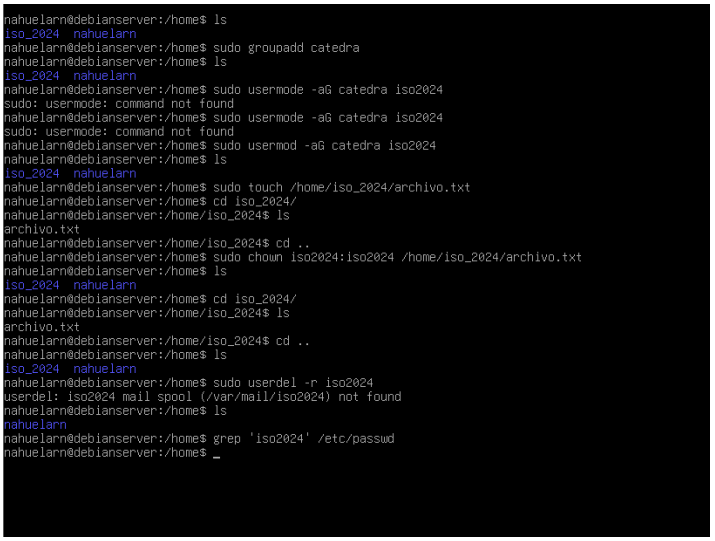
El comando `pstree` muestra una representación en forma de árbol de los procesos en ejecución en el sistema. Este comando es útil para ver cómo los procesos se relacionan entre sí, mostrando qué procesos son padres e hijos de otros.

¿Qué es lo que se puede observar a partir de la ejecución de este comando?

Al ejecutar `pstree`, puedes observar:

- **Estructura Jerárquica de Procesos**: Muestra cómo los procesos están organizados jerárquicamente, lo que te permite ver qué procesos son padres y cuáles son hijos.
- **Relaciones entre Procesos**: Puedes ver cómo los procesos están relacionados entre sí, lo que puede ser útil para entender la estructura del sistema y para la resolución de problemas (por ejemplo, ver qué servicios se han iniciado bajo un demonio padre).
- **Uso de Recursos y Dependencias**: Al observar el árbol de procesos, puedes identificar qué procesos se han iniciado como resultado de otros, lo cual es útil para entender el uso de recursos del sistema y las dependencias entre procesos.

El uso de `pstree` es especialmente útil para administradores de sistemas y desarrolladores que necesitan entender cómo están organizados los procesos y para identificar rápidamente los procesos secundarios de un servicio o aplicación.



4 Usuarios

(a) ¿Qué archivos son utilizados en un sistema GNU/Linux para guardar la información de los usuarios?

En un sistema GNU/Linux, la información de los usuarios se guarda principalmente en los siguientes archivos:

- `/etc/passwd`: Contiene la información básica de cada usuario, como el nombre de usuario, UID, GID, home directory, y shell predeterminado.
- `/etc/shadow`: Guarda las contraseñas encriptadas y la información relacionada con las contraseñas de los usuarios, como la fecha de cambio de la contraseña y la fecha de expiración.
- `/etc/group`: Almacena información sobre los grupos del sistema, incluyendo el nombre del grupo, GID y la lista de usuarios miembros.

- `/etc/gshadow`: Similar a `/etc/shadow`, pero para los grupos. Contiene contraseñas de los grupos y otra información relacionada.

(b) ¿A qué hacen referencia las siglas UID y GID? ¿Pueden coexistir UIDs iguales en un sistema GNU/Linux? Justifique.

- **UID (User Identifier)**: Es un número único asignado a cada usuario en el sistema. Es utilizado por el sistema para identificar de manera única a un usuario.
- **GID (Group Identifier)**: Es un número que identifica de manera única a un grupo en el sistema.

Coexistencia de UIDs iguales:

- Por lo general, cada usuario debe tener un UID único en el sistema. Sin embargo, es técnicamente posible asignar el mismo UID a varios usuarios. En tal caso, esos usuarios compartirían los mismos permisos de acceso a archivos y recursos, ya que el sistema no podría diferenciarlos. Esto no es una práctica recomendada, ya que puede llevar a problemas de seguridad y administración.

(c) ¿Qué es el usuario root? ¿Puede existir más de un usuario con este perfil en GNU/Linux? ¿Cuál es la UID del root?

- **Usuario root**: Es el superusuario en sistemas GNU/Linux. Tiene acceso completo a todos los comandos y archivos en el sistema. El usuario root puede realizar cualquier acción administrativa, incluyendo la creación y eliminación de usuarios, la instalación de software, y la modificación de archivos de sistema.
- **Múltiples usuarios con perfil root**: En GNU/Linux, la UID 0 está reservada para el superusuario. Técnicamente, es posible crear múltiples usuarios que tengan UID 0, lo que les otorgaría los mismos privilegios que el usuario root. Sin embargo, no es una práctica común ni recomendada debido a los riesgos de seguridad.
- **UID del root**: La UID del root es `0`.

(d) Agregar un nuevo usuario llamado iso2017 a su instalación de GNU/Linux:

1. **Crear el usuario `iso2017`:**

```
sudo useradd -m -d /home/iso_2017 iso2017
```

   Esto crea el usuario `iso2017` con su directorio personal en `/home/iso_2017`.

2. **Crear el grupo `catedra`** (si no existe):

```
sudo groupadd catedra
```

3. **Agregar el usuario `iso2017` al grupo `catedra`:**

```
sudo usermod -aG catedra iso2017
```

4. **Crear un archivo en su directorio personal que le pertenezca:**

```
sudo touch /home/iso_2017/archivo.txt
sudo chown iso2017:iso2017 /home/iso_2017/archivo.txt
```

5. **Eliminar el usuario `iso2017` y verificar:**

```
sudo userdel -r iso2017
```

   Esto eliminará al usuario y su directorio personal. Luego, verifica que no quede ningún registro en los archivos de información de usuarios y grupos:

```
grep 'iso2017' /etc/passwd
grep 'iso2017' /etc/group
```

(e) Funcionalidad y parámetros de los siguientes comandos:

1. **`useradd` o `adduser`:**

- `useradd`: Comando utilizado para crear un nuevo usuario. Tiene muchas opciones como `m` (crea el directorio personal), `d` (especifica el directorio personal), `s` (especifica el shell), `G` (añadir a un grupo secundario).
- `adduser`: Es un script que usa `useradd` internamente, pero proporciona una interfaz más amigable. No es común en todos los sistemas, ya que depende de la distribución.

2. `usermod`:
   - Modifica una cuenta de usuario existente. Ejemplos de uso:
     - `usermod -aG groupname username`: Añadir a un grupo secundario.
     - `usermod -d /new/home/dir username`: Cambiar el directorio personal.
3. `userdel`:
   - Elimina una cuenta de usuario. La opción `r` elimina también el directorio personal y los archivos del usuario.
4. `su`:
   - Cambia el usuario activo a otro. Por ejemplo, `su -` cambia al usuario root. La opción `-` carga el entorno del usuario.
5. `groupadd`:
   - Crea un nuevo grupo en el sistema. Ejemplo: `groupadd groupname`.
6. `who`:
   - Muestra quién está actualmente conectado al sistema. Incluye información como el nombre del usuario, terminal, fecha y hora.
7. `groupdel`:
   - Elimina un grupo del sistema. Ejemplo: `groupdel groupname`.
8. `passwd`:
   - Cambia la contraseña de un usuario. Si se ejecuta como root, puedes cambiar la contraseña de cualquier usuario. También puedes bloquear o desbloquear cuentas usando opciones como `l` (bloquear) y `u` (desbloquear).

Estas respuestas te deberían ayudar a completar la práctica sobre administración de usuarios en sistemas GNU/Linux.

5. FileSystem:

Como son definidos los permisos sobre archivos en un sistema GNU/LINUX?
En GNU/LINUX, los permisos sobre archivos y directorios son gestionados utilizando un esquema de permisos que incluye:

a) ¿Cómo son definidos los permisos sobre archivos en un sistema GNU/Linux?

En GNU/Linux, los permisos sobre archivos y directorios son gestionados utilizando un esquema de permisos que incluye:

1. **Usuario (Owner)**: El propietario del archivo.
2. **Grupo**: Un grupo de usuarios que tienen ciertos permisos sobre el archivo.
3. **Otros (Others)**: Cualquier usuario que no es ni el propietario ni un miembro del grupo.

Cada archivo o directorio tiene permisos que controlan las acciones que estos tres tipos de usuarios pueden realizar. Los permisos se dividen en:

- **Lectura (r)**: Permite ver el contenido de un archivo o listar los contenidos de un directorio.
- **Escritura (w)**: Permite modificar el contenido de un archivo o añadir/eliminar archivos en un directorio.
- **Ejecución (x)**: Permite ejecutar un archivo si es un programa o script, o acceder a un directorio y sus archivos.

Estos permisos se pueden ver y modificar usando comandos como `ls -l` para listar y `chmod` para cambiar los permisos.

(b) Funcionalidad y parámetros de los comandos relacionados con los permisos en GNU/Linux

`chmod` (change mode)

- **Funcionalidad**: Cambia los permisos de un archivo o directorio.
- **Parámetros**:
  - **Notación simbólica**: `chmod u+x archivo` añade permisos de ejecución para el usuario (owner).
  - **Notación octal**: `chmod 755 archivo` establece permisos específicos utilizando números octales.
- **Ejemplo**:

```
bashCopiar código
chmod 644 archivo.txt  # Establece permisos de lectura y escritura para el propietario, y solo lectura para grupo y otros.
```

`chown` (change owner)

- **Funcionalidad**: Cambia el propietario y/o el grupo de un archivo o directorio.
- **Parámetros**:
  - `chown usuario archivo`: Cambia solo el propietario.
  - `chown usuario:grupo archivo`: Cambia el propietario y el grupo.
- **Ejemplo**:

```
bashCopiar código
chown alice archivo.txt  # Cambia el propietario de archivo.txt a alice.
```

`chgrp` (change group)

- **Funcionalidad**: Cambia el grupo asociado a un archivo o directorio.
- **Parámetros**:
  - `chgrp grupo archivo`: Cambia el grupo del archivo.
- **Ejemplo**:

```
bashCopiar código
chgrp developers archivo.txt  # Cambia el grupo de archi
vo.txt a developers.
```

(c) Notación octal en `chmod`

La notación octal en `chmod` se utiliza para definir los permisos de un archivo o directorio de manera compacta. Cada dígito octal representa un conjunto de permisos para el propietario, grupo, y otros:

- **4**: Permiso de lectura (`r`).
- **2**: Permiso de escritura (`w`).
- **1**: Permiso de ejecución (`x`).
- **0**: Sin permisos.

Los números octales se suman para combinar los permisos:

- **7**: Lectura, escritura y ejecución (`rwx`).
- **6**: Lectura y escritura (`rw-`).
- **5**: Lectura y ejecución (`r-x`).
- **4**: Solo lectura (`r--`).

Por ejemplo, `chmod 755 archivo.txt` establece:

- **Propietario**: Lectura, escritura, ejecución (`7` → `rwx`).
- **Grupo**: Lectura y ejecución (`5` → `r-x`).
- **Otros**: Lectura y ejecución (`5` → `r-x`).

(d) ¿Existe la posibilidad de que algún usuario del sistema pueda acceder a determinado archivo para el cual no posee permisos?

Sí, es posible mediante varios métodos, dependiendo de la configuración del sistema y los permisos específicos:

1. **Acceso como root**: El superusuario (root) tiene permisos para acceder a todos los archivos del sistema, independientemente de sus permisos individuales.

2. **Setuid, Setgid y Sticky bit**:
    - **Setuid**: Cuando un archivo ejecutable tiene el bit Setuid activado, se ejecuta con los permisos del propietario del archivo en lugar de los del usuario que lo ejecuta.
    - **Setgid**: Similar al Setuid, pero aplica al grupo.
    - **Sticky bit**: Aplica principalmente a directorios; cuando está activo, solo el propietario del archivo (o el root) puede borrar o renombrar los archivos.
3. **ACLs (Access Control Lists)**: Proporcionan un nivel adicional de control de acceso, permitiendo especificar permisos más granulares.

Para realizar pruebas:

- Crea un archivo con permisos restrictivos:

```
touch prueba.txt
chmod 000 prueba.txt  # Sin permisos para nadie
```

- Intenta acceder al archivo como usuario normal (fallará).
- Usa `sudo` para acceder como root:

```
sudo cat prueba.txt  # Como root, se podrá acceder
```

(e) Conceptos de "full path name" y "relative path name"

- **Full path name (Ruta absoluta)**: Especifica una ruta completa desde el directorio raíz (`/`) hasta el archivo o directorio. Siempre comienza con `/`.

  Ejemplo:

```
/home/usuario/documentos/informe.txt
```

- **Relative path name (Ruta relativa)**: Especifica una ruta en relación al directorio actual en el que se encuentra el usuario. No comienza con `/`.

  Ejemplo:

```
documentos/informe.txt  # Asumiendo que estás en /home/u
suario
```

(f) Determinar el directorio actual y acceder al directorio personal

- **Comando para determinar el directorio actual**:

  ```
  pwd
  ```

  `pwd` (print working directory) muestra el directorio en el que te encuentras actualmente.

- **Acceder al directorio personal**:
  Puedes usar el símbolo
  `~` para referirte al directorio personal. Por ejemplo:

  ```
  cd ~  # Te lleva a tu directorio home
  ```

  También puedes acceder a otros directorios relativos al directorio home usando `~usuario`:

  ```
  cd ~usuario  # Cambia al directorio home de "usuario"
  ```

Ejemplo:

```
cd ~/documentos  # Accede a la carpeta "documentos" en tu d
irectorio
```

(g) Investigue la funcionalidad y parámetros de los siguientes comandos relacionados con
el uso del FileSystem:

- **Funcionalidad**: Cambia el directorio actual de trabajo.

- **Sintaxis**: `cd [directorio]`
- **Parámetros**:
  - `cd [ruta]`: Cambia al directorio especificado.
  - `cd ..`: Cambia al directorio padre (nivel superior).
  - `cd ~`: Cambia al directorio personal del usuario.
  - `cd -`: Cambia al último directorio en el que estabas.

Ejemplo:

```
cd /etc  # Cambia al directorio /etc
cd ..    # Vuelve al directorio anterior (padre)
```

2. `umount` (Unmount)

- **Funcionalidad**: Desmonta un sistema de archivos montado.
- **Sintaxis**: `umount [opciones] [punto de montaje | dispositivo]`
- **Parámetros**:
  - `umount [punto de montaje]`: Desmonta el sistema de archivos montado en el punto de montaje especificado.
  - `l` o `-lazy`: Desmontaje "perezoso". Desmonta el sistema de archivos cuando ya no esté en uso.
  - `f` o `-force`: Forzar el desmontaje (útil para sistemas de archivos de red).

Ejemplo:

```
umount /mnt/usb  # Desmonta el sistema de archivos montado
en /mnt/usb
umount -f /mnt/usb  # Fuerza el desmontaje del dispositivo
```

3. `mkdir` (Make Directory)

- **Funcionalidad**: Crea uno o más directorios.

- **Sintaxis**: `mkdir [opciones] directorio...`
- **Parámetros**:
  - `p` o `-parents`: Crea directorios padres si no existen (no lanza error si ya existen).
  - `m [modo]` o `-mode=[modo]`: Establece los permisos del nuevo directorio.

Ejemplo:

```
mkdir proyecto  # Crea un directorio llamado proyecto
mkdir -p /ruta/a/mi/directorio  # Crea todos los directorio
s necesarios en la ruta especificada
```

4. `du` (Disk Usage)

- **Funcionalidad**: Muestra el uso del disco de archivos y directorios.
- **Sintaxis**: `du [opciones] [archivo/directorio]`
- **Parámetros**:
  - `h` o `-human-readable`: Muestra tamaños en formato legible por humanos (KB, MB, GB).
  - `s` o `-summarize`: Muestra solo el total para cada argumento.
  - `a` o `-all`: Incluye archivos individuales además de los directorios.

Ejemplo:

```
du -h  # Muestra el uso del disco de todos los archivos y d
irectorios en el directorio actual
du -sh /home/usuario  # Muestra el uso total del disco en /
home/usuario
```

5. `rmdir` (Remove Directory)

- **Funcionalidad**: Elimina directorios vacíos.
- **Sintaxis**: `rmdir [opciones] directorio...`
- **Parámetros**:

- `-ignore-fail-on-non-empty`: Ignora errores al intentar eliminar directorios no vacíos.
- `p` o `-parents`: Elimina directorios padres si están vacíos después de eliminar el directorio especificado.

Ejemplo:

```
rmdir test  # Elimina el directorio 'test' si está vacío
rmdir -p /ruta/a/mi/directorio  # Elimina 'directorio' y lo
s directorios padres si están vacíos
```

6. `df` (Disk Free)

- **Funcionalidad**: Muestra el espacio libre y usado en los sistemas de archivos.
- **Sintaxis**: `df [opciones] [archivo/sistema de archivos]`
- **Parámetros**:
  - `h` o `-human-readable`: Muestra tamaños en formato legible por humanos.
  - `T` o `-print-type`: Muestra el tipo de sistema de archivos.
  - `i` o `-inodes`: Muestra información sobre los inodos en lugar de bloques.

Ejemplo:

```
df -h  # Muestra el uso de disco en un formato legible por
humanos
df -T  # Muestra el tipo de sistema de archivos además del
uso de disco
```

7. `mount` (Mount)

- **Funcionalidad**: Monta un sistema de archivos.
- **Sintaxis**: `mount [opciones] [dispositivo] [punto de montaje]`
- **Parámetros**:
  - `t [tipo]` o `-types [tipo]`: Especifica el tipo de sistema de archivos.

- `o [opciones]`: Opciones de montaje (como `ro` para solo lectura, `rw` para lectura/escritura).
- `a` o `-all`: Monta todos los sistemas de archivos listados en `/etc/fstab`.

Ejemplo:

```
mount /dev/sdb1 /mnt/usb  # Monta el dispositivo /dev/sdb1
en /mnt/usb
mount -t ext4 /dev/sdb1 /mnt/usb  # Monta /dev/sdb1 como un
sistema de archivos ext4
```

8. `ln` (Link)

- **Funcionalidad**: Crea enlaces (links) a archivos.
- **Sintaxis**: `ln [opciones] archivo destino`
- **Parámetros**:
  - `s` o `-symbolic`: Crea un enlace simbólico (enlace suave).
  - `f` o `-force`: Forza la creación de un enlace, sobrescribiendo archivos existentes.
  - `b` o `-backup`: Realiza un respaldo de los archivos de destino.

Ejemplo:

```
ln archivo1 enlace1  # Crea un enlace duro de archivo1 llam
ado enlace1
ln -s archivo1 enlace1  # Crea un enlace simbólico de archi
vo1 llamado enlace1
```

9. `ls` (List)

- **Funcionalidad**: Lista el contenido de un directorio.
- **Sintaxis**: `ls [opciones] [archivo/directorio]`
- **Parámetros**:

- `l`: Lista en formato largo, mostrando detalles como permisos, propietario, tamaño y fecha de modificación.
- `a` o `-all`: Muestra todos los archivos, incluidos los ocultos (los que empiezan con `.`).
- `h` o `-human-readable`: Muestra tamaños en un formato legible por humanos cuando se usa con `l`.

Ejemplo:

```
ls -l  # Lista en formato largo los archivos y directorios
en el directorio actual
ls -a  # Lista todos los archivos, incluidos los ocultos
```

10. `pwd` (Print Working Directory)

- **Funcionalidad**: Muestra el directorio de trabajo actual.
- **Sintaxis**: `pwd`
- **Parámetros**: No tiene opciones generalmente utilizadas, aunque puede soportar `P` para mostrar la ruta física, resolviendo enlaces simbólicos.

Ejemplo:

```
pwd  # Muestra la ruta completa del directorio actual
```

11. `cp` (Copy)

- **Funcionalidad**: Copia archivos y directorios.
- **Sintaxis**: `cp [opciones] origen destino`
- **Parámetros**:
  - `r` o `-recursive`: Copia directorios de forma recursiva.
  - `i` o `-interactive`: Solicita confirmación antes de sobrescribir archivos existentes.
  - `v` o `-verbose`: Muestra los archivos a medida que se copian.

Ejemplo:

```
cp archivo1 archivo2  # Copia archivo1 a archivo2
cp -r dir1 dir2  # Copia todo el contenido de dir1 a dir2
```

12. `mv` (Move)

- **Funcionalidad**: Mueve o renombra archivos y directorios.
- **Sintaxis**: `mv [opciones] origen destino`
- **Parámetros**:
  - `i` o `-interactive`: Solicita confirmación antes de sobrescribir archivos.
  - `f` o `-force`: Sobrescribe archivos sin preguntar.
  - `v` o `-verbose`: Muestra los archivos a medida que se mueven.

Ejemplo:

```
mv archivo1 archivo2  # Renombra o mueve archivo1 a archivo
mv dir1 dir2  # Renombra o mueve dir1 a dir2
```

6. Procesos:

(a) ¿A qué hace referencia el concepto de empaquetar archivos en GNU/Linux?

Empaquetar archivos en GNU/Linux se refiere al proceso de agrupar varios archivos y/o directorios en un único archivo. Esto se hace principalmente con el propósito de simplificar el manejo, la transferencia o el almacenamiento de estos archivos. Empaquetar no comprime los archivos; simplemente los agrupa en un solo archivo contenedor. El comando más común para empaquetar archivos en GNU/Linux es `tar`.

(b) Seleccione 4 archivos dentro de algún directorio al que tenga permiso y sume el tamaño de cada uno de estos archivos. Cree un archivo empaquetado conteniendo estos 4

archivos y compare los tamaños de los mismos. ¿Qué característica nota?

1. **Seleccionar archivos y calcular tamaño:**

   Supongamos que los archivos seleccionados son `file1.txt`, `file2.txt`, `file3.txt`, y `file4.txt` y están ubicados en el directorio `/home/user/`. Puedes usar el comando `du` para sumar el tamaño de los archivos:

   ```
   du -b /home/user/file1.txt /home/user/file2.txt /home/us
   er/file3.txt /home/user/file4.txt
   ```

   El parámetro `-b` muestra el tamaño en bytes. Para sumar los tamaños, puedes usar `awk` o una simple suma:

   ```
   du -b /home/user/file1.txt /home/user/file2.txt /home/us
   er/file3.txt /home/user/file4.txt | awk '{sum += $1} END
   {print sum}'
   ```

2. **Crear un archivo empaquetado:**

   Usa el comando `tar` para empaquetar los archivos:

   ```
   tar -cvf archivos.tar /home/user/file1.txt /home/user/fi
   le2.txt /home/user/file3.txt /home/user/file4.txt
   ```

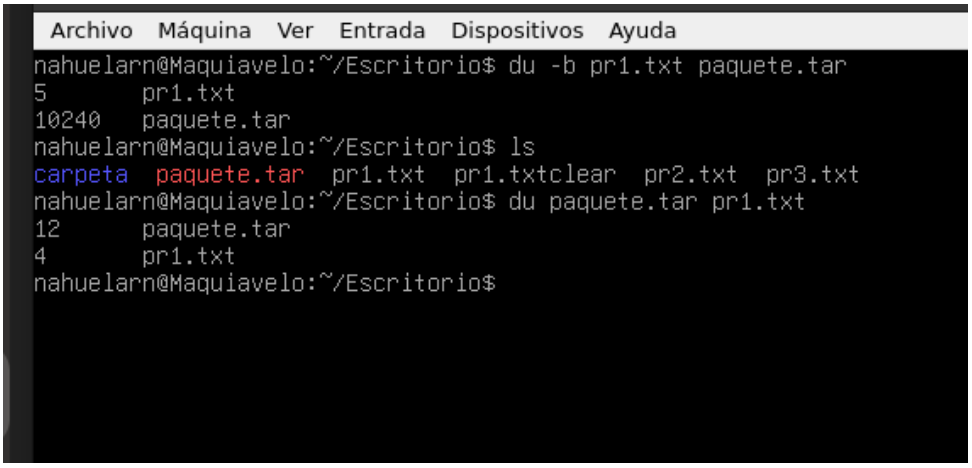
   El parámetro `-c` crea el archivo, `-v` muestra el progreso y `-f` especifica el nombre del archivo.

3. **Comparar tamaños:**

   Calcula el tamaño del archivo empaquetado:

   ```
   du -b archivos.tar
   ```

Característica a notar:
El archivo empaquetado (
`archivos.tar`) suele ser más grande que la suma de los tamaños individuales de los archivos, ya que `tar` no realiza compresión, solo empaqueta. El tamaño adicional proviene de la metadata y la estructura del archivo tar.



(c) ¿Qué acciones debe llevar a cabo para comprimir 4 archivos en uno solo? Indique la secuencia de comandos ejecutados.

Para comprimir 4 archivos en uno solo, primero debes empaquetarlos y luego comprimir el archivo empaquetado. La secuencia de comandos es:

1. **Empaquetar los archivos:**

```
bash
Copiar código
tar -cvf archivos.tar /home/user/file1.txt /home/user/fi
le2.txt /home/user/file3.txt /home/user/file4.txt
```

2. **Comprimir el archivo empaquetado:**

   Usa `gzip` para comprimir el archivo `archivos.tar`:

```
bash
Copiar código
```

```
gzip archivos.tar
```

Esto generará `archivos.tar.gz`, que es el archivo comprimido.