Taller de GNU/Linux

Conceptos

GNU/Linux: GNU/Linux es un sistema operativo de código abierto, compuesto por el núcleo Linux y las herramientas del proyecto GNU. Es conocido por su estabilidad, seguridad y flexibilidad, lo que lo hace popular entre desarrolladores, administradores de sistemas y usuarios avanzados. Está disponible en múltiples distribuciones, cada una adaptada a diferentes necesidades y preferencias. Por lo general, hay mucha similitud entre distribuciones, una de las diferencias más notables siendo la gestión e instalación de paquetes.

Además de ser conocido por las razones anteriores, también lo es por ser más complicado de usar que otros sistemas operativos, por lo que no es una opción popular entre personas que no requieren mucho conocimiento informático para su uso día a día. Pero esta complicación irradia de la filosofía principal de Linux: asume que el usuario sabe lo que hace y le deja hacerlo. Esto puede sonar a que es fácil de romper, pero en realidad permite al usuario usar todas las herramientas del sistema para poder administrar su sistema de la forma que desee, aportando mayor libertad. [1]

Terminal: La terminal, también conocida como consola, es una interfaz de texto que permite a los usuarios interactuar con el sistema operativo o el software mediante la escritura de comandos. A través de la terminal, los usuarios pueden controlar todo el sistema, además de ejecutar programas, manipular archivos y directorios, configurar ajustes del sistema y muchas cosas más de manera rápida y eficiente.
Se recuerda que en la interfaz de comandos de Linux, se distingue entre mayúsculas y minúsculas. El comando `Ls` no existe, es `ls`. También va por los directorios, `~/Documentos/` no es lo mismo que `~/documentos/`. Esto se llama case-sensitive.

Comandos: Un comando es una instrucción dada a un sistema informático para realizar una tarea específica. En Linux, los comandos se introducen en la línea de comandos (CLI) y son interpretados por el intérprete de comandos (shell) para ejecutar una acción, como manipular archivos, consultar datos, lanzar programas, etc. Los comandos suelen tener la forma `PROGRAMA OPCIONES ARGUMENTOS`. Ejemplo: `ls -l /`, `ls -l -a / /home`, `ls -la / /home`

Por qué usar la terminal: Nos permite interactuar directamente con el sistema operativo, ofreciéndonos una forma rápida, eficiente y flexible de realizar tareas complejas. El uso de la terminal nos permite automatizar procesos, gestionar archivos y configuraciones del

sistema con mayor control que a través de interfaces gráficas.

Gestión de paquetes y programas

¿Qué es un gestor de paquetes y por qué usarlos?

Comencemos con lo básico. Un gestor de paquetes es una herramienta que facilita la instalación, actualización, configuración y eliminación de software en tu sistema operativo Linux. En lugar de buscar e instalar cada programa manualmente, el gestor de paquetes se encarga de todo el proceso por ti, lo que ahorra tiempo y garantiza la consistencia del sistema.

Dependiendo de la distribución Linux que utilices vas a utilizar un gestor de paquetes o otro:
Para Debian y sus derivados (Ubuntu, Linux Mint): APT (Advanced Package Tool)
Para Arch Linux y sus derivados: Pacman
Para Red Hat Enterprise Linux y sus derivados (Fedora, CentOS): DNF (Dandified YUM)
Cada uno de estos tiene sus distintas opciones y posibilidades. En este taller nos centraremos en APT, ya que estamos usando una distro basada en Ubuntu.

El comando sudo

`sudo` ("super user do") es un comando de Unix y sistemas operativos basados en Unix que permite a los usuarios ejecutar comandos con los privilegios de otro usuario, generalmente el superusuario o administrador del sistema. Es una herramienta poderosa que permite a los usuarios ejecutar comandos con privilegios elevados de manera segura y controlada, lo que es fundamental para realizar tareas de administración en sistemas Unix y garantizar la seguridad del sistema.
Debe de usarse en moderación, solo cuando se hagan cambios importantes persistentes en el sistema.

¿Cómo funciona APT?

APT utiliza repositorios, que son servidores en línea que contienen una gran cantidad de paquetes de software disponibles para su instalación en tu sistema Linux. Generalmente, cada distribución tiene sus propios repositorios, y si un paquete está disponible en los repositorios oficiales, casi siempre es seguro instalarselo.

Opciones comunes de APT:

- `update` : Actualiza la base de datos local de paquetes con la última información de los repositorios.
- `upgrade` : Actualiza todos los paquetes instalados en tu sistema a sus últimas

versiones disponibles en los repositorios.

- `install` : Descarga e instala el paquete especificado junto con sus dependencias.
- `remove` : Elimina el paquete especificado del sistema, junto con cualquier otro paquete que haya sido instalado como dependencia y que ya no sea necesario.
- `search` : Busca en la base de datos local de paquetes coincidencias con el término de búsqueda proporcionado.
- `autoremove` : Elimina automáticamente los paquetes que fueron instalados como dependencias de otros paquetes pero que ya no son necesarios.

Cuestión: ¿Porque crees que es necesario tener permisos de superusuario para ejecutar estos comandos?

Ejercicio: Escribe parcialmente un comando, y pulsa la tecla `TAB` . ¿Que ha ocurrido?

Ejemplo divertido con "cowsay":

Ahora que entendemos la teoría, ¡vamos a practicar con algo divertido! Vamos a instalar y utilizar "cowsay", un programa que muestra mensajes de texto en forma de burbujas de diálogo de una vaca. Para instalarlo, simplemente ejecutamos el siguiente comando en nuestra terminal:

```
sudo apt install cowsay
```

Ahora que hemos instalado "cowsay", probemos con un mensaje divertido. Por ejemplo:

```
cowsay "¡Hola a todos! ¡Bienvenidos al taller de Linux!"
```

Como podéis ver el programa funciona perfectamente. Podemos intentar instalar ahora algo un poco más útil, como por ejemplo, el programa `htop` .

`htop` es un comando de terminal en sistemas Linux que proporciona una vista interactiva y mejorada del uso del sistema y de los procesos en ejecución. Es una alternativa más avanzada y visual al comando top, mostrando información como el uso de CPU, memoria, procesos y su consumo de recursos en tiempo real. Es útil para supervisar el rendimiento del sistema y para gestionar procesos de manera eficiente desde la línea de comandos.

Ejercicio: ¿Como crees que se instalaría htop?

Una vez instalado ejecutad el comando `htop` y comprobad que funciona bien. Para salir de la aplicación podeis usar ctrl + c .

El comando `man`

`man` es un comando utilizado en sistemas operativos basados en Unix y Linux para acceder al manual de referencia del sistema. Proporciona información detallada sobre el uso y la sintaxis de otros comandos, así como información de referencia sobre archivos, funciones del sistema y configuraciones del sistema.

Ejemplo de uso:
Para acceder al manual de referencia de un comando específico, simplemente escribe man seguido del nombre del comando. Por ejemplo, para obtener información sobre el comando ls que se utiliza para listar archivos en un directorio, puedes ejecutar: `man ls`

Esto mostrará el manual de referencia de `ls`, que incluye una descripción del comando, opciones disponibles, argumentos aceptados y ejemplos de uso. Puedes desplazarte hacia arriba y hacia abajo en el manual utilizando las teclas de dirección, y salir del manual presionando la tecla `q`.

Directorios

Directorio Raíz (`/`):

El directorio raíz (`/`) es el directorio principal en el sistema de archivos de Linux. Es dónde se guarda toda la información del sistema, basándose en subdirectorios y ficheros de texto plano (si, os podéis olvidar de las extensiones).
No es necesario saberse por completo su contenido y para que se usa cada directorio para este taller, y en uso general no se suelen tocar mucho (a excepción de etc en casos muy particulares).
Aquí un listado de los subdirectorios bajo (`/`) y uso resumido de ellos: [2]

- `/bin/`
  - Contiene los binarios esenciales del sistema. Son los programas ejecutables disponibles para todos los usuarios.
- `/boot/`
  - Almacena archivos necesarios del sistema, incluyendo el kernel y configuración de arranque.
- `/dev/`
  - Contiene archivos que representan dispositivos en el sistema, como discos duros, dispositivos USB, y otro tipo de dispositivos especiales.
- `/etc/`
  - Contiene configuración vital para el sistema, como scripts de inicio, servicios, configuración de redes, diversas aplicaciones, etc.
- `/home/`
  - Directorio raíz de los usuarios, contiene toda la información respecto a cada uno.
- `/lib/`
  - Colección de código pre-compilado que los ejecutables de `/bin/` usan.
- `/media/`
  - Directorio temporal en el que se montan automáticamente los nuevos

dispositivos de almacenaje como discos duros adicionales, USB y más. Cuando se desconectan, la carpeta desaparece.

- `/mnt/`
  - Directorio en el que el sistema de datos monta temporalmente dispositivos adicionales. Al contrario que `/media/`, para montar en este directorio se necesita que manualmente se añada a este (o mediante el uso de fstab)(no lo vemos en este taller).
- `/opt/`
  - Se usa para guardar información de paquetes instalados que nos están disponibles en los repositorios oficiales y/o para evitar conflictos con la configuración del sistema (p.ej. discord suele estar aquí, pero está en los repositorios oficiales)
- `/proc/`
  - Este es un directorio especial, tiene su propio pseudo-sistema de ficheros, conteniendo informaciónsobre procesos y parámetros de kernel. Se llena de información al enceder la máquina, y se vacía una vez se apaga.
- `/root/`
  - Directorio home, pero para el superusuario.
- `/run/`
  - Información del sistema des de el inicio. Contiene información sobre los procesos de fondo, usuarios logeados.
- `/sbin/`
  - Programas que solo son ejecutables por el adimnistrador.
- `/tmp/`
  - Guarda información de programas de forma temporal.
- `/usr/`
  - Todos los binarios y ficheros solo lectura del sistema.
- `/var/`
  - Contiene varios ficheros que varian de tamaño durante operación normal.

El directorio /home/*usuario*/es donde se guarda toda la información específica de ese usuario. Generalmente se encuentra las típicas carpetas de Descargas, Documentos, Escritorio, etc, además de las carpetas ocultas de configuración.

Comandos Básicos:

Volvemos a recordar que los comandos son case-sensitive.

`cd [dir | ..]` : El comando "cd" se utiliza para cambiar de directorio. Por ejemplo:

- `cd Documentos` : Cambia al directorio "documentos".
- `cd ..` : Retrocede al directorio anterior.

`ls [and options]` : El comando `ls` se utiliza para mostrar el contenido de un directorio. Algunas opciones comunes incluyen:

- `-l` : Muestra el contenido en formato detallado.
- `-a` : Muestra todos los archivos, incluidos los ocultos.

Cuestión: Al usar `-a` podemos ver dos secciones destacables: `.` y `..`, ¿Que significan?

- `-h` : Muestra el tamaño de los archivos en un formato legible por humanos.

Directorios/ficheros ocultos: Los archivos y directorios que comienzan con un punto (`.`) se consideran ocultos en Linux. Se pueden mostrar utilizando la opción `-a` con el comando `ls`.
¡Prueba a combinarlos! En abreviaciones de opciones, se pueden combinar sin separarlos. Ejecuta `ls -lah` y mira lo que hace.

Creación y Gestión de Directorios:

`mkdir` : Se utiliza para crear nuevos directorios. Por ejemplo:

- `mkdir documentos` : Crea un nuevo directorio llamado "documentos".

`mv` : Se utiliza para mover archivos o directorios de una ubicación a otra. Por ejemplo:

- `mv archivo.txt documentos/arxiu.txt` : Mueve el archivo "archivo.txt" al directorio "documentos", con el nuevo nombre "arxiu.txt".

`rm` : El comando "rm" se utiliza para eliminar archivos o directorios. Por ejemplo:

- `rm archivo.txt` : Elimina el archivo "archivo.txt".
- `rm -r documentos` : Elimina el directorio "documentos" y su contenido de forma recursiva.

Cuestion: ¿Como crees que es el comando para renombrar archivos/directorios? Pista: no es `rm`, no cometas el mismo error que una vez me pasó.

Ejercicios Prácticos:

- Cambiar al directorio personal del usuario.
- Crear un nuevo directorio llamado "proyectos".

- Mover un archivo de texto a este nuevo directorio.
- Mostrar el contenido del directorio actual, incluidos los archivos ocultos.
- Eliminar el directorio junto con el archivo de debajo.
- Crea un directorio, déjalo vacío e intenta eliminar ese directorio.

Gestión de Permisos, Grupos y Usuarios

La gestión de usuarios y permisos en Linux es una parte fundamental de la administración de sistemas operativos, ya que permite controlar quién tiene acceso al sistema, qué recursos pueden utilizar y qué acciones pueden realizar. Esta gestión se realiza a través de la creación, modificación y eliminación de cuentas de usuario y grupos de usuarios, así como también mediante la asignación de permisos y privilegios específicos.

Revisión de permisos:

Para ver los permisos sobre un fichero o directorio, se puede usar `ls -l`. Se separan en 3 apartados, donde cada uno representa permisos para un cierto grupo o personas. Ejemplo:

```
drwxrw-r-- 10 usuario grupo 4096 mar 21 12:00 Documents
```

- `d`: Esto indica el tipo del objeto. `d` significa un directorio, `-` significa un archivo cualquiera, `l` indica un link simbólico.
- `rwxrw-r--` Son los permisos pertenecientes a ese objeto. Se separan en tres secciones, cada una representando 3 letras. `r` significa permisos de lectura, `w` son permisos de escritura y `x` son de ejecución. Si no tiene un permiso, se representa con `-`.
    - Permisos específicos del usuario propietario
        - Son las 3 primeras letras, e indica los permisos que tiene el propietario del archivo. El propietario se puede ver después de los permisos, en este caso `usuario`. Lo más común es que tenga todos los permisos, siendo nuestro caso.
    - Permisos específicos de grupo
        - Las 3 siguientes letras indican los permisos que tienen las personas que pertenecen a ese grupo. Más adelante vemos como administrar estos.
    - Permisos para cualquier otro usuario
        - Las últimas 3 letras estos son los permisos que engloba cualquier otro usuario que ni es el propietario, ni se encuentra en el grupo. Generalmente, no tienen ningun permiso en el directorio.

Cuestión: ¿Podrías reconocer a que y quien tiene los permisos del directorio del ejemplo? ¿Y que tal el siguiente? ¿Serías capaz de encontrar la posible vulnerabilidad de este grupo de permisos?

```
lrwxr-xr-x 10 usuario grupo 4096 <fecha> Directorio
```

¿Por qué el propietario y el grupo tienen el mismo nombre?
Cuando se crea un usuairo, se crea un grupo con el mismo nombre de usuario, siendo ese el grupo principal. Más en las diferencias entre estos en el siguiente apartado.

Modificación de permisos:

El comando `chmod` es el comando estándar en sistemas linux para cambiar permisos. Hay dos maneras de cambiar permisos usando los comandos: usando letras o los valores binarios.

Usando letras

Este método se basa en usar las letras que representan los permisos.

- `r` - "Read"; lectura
- `w` - "Write"; escritura
- `x` - "eXecute"; ejecución
- `u` - "User (owner)"; Usuario propietario
- `g` - "Group"; Grupo al que pertenece
- `o` - "Others"; Cualquier otro

Usando operadores podemos especificar permisos:

- `+` Añade permisos
- `-` Quita permisos
- `=` Cambia los permisos a los especificados

Con esto, podemos combinar opciones. Por ejemplo, si queremos añadir permisos de ejecución para el propietario y grupo, y quitar permisos de lectura para los demás, el comando queda como:

```
chmod ug+x,o-r directorio
```

Si queremos quitar todos los permisos para un grupo tenemos dos opciones igualmente válidas:

```
chmod o=- directorio
chmod o-rwx directorio
```

Cuestión: ¿Quien crees que es capaz de cambiar los permisos? Pista: hay dos usuarios

capaces.

Práctica:

Crea un directorio, y, usando el método de las letras, ponle los siguientes permisos:

- `rwxr-xr--`
- `rw-rw----`
- `rwx------`
- `rwxrwxrwx`

Método Octal

Aquí, cada grupo de permisos se les asigna un valor entre 0 y 7.

- Lectura: 4
- Escritura: 2
- Ejecución: 1

Al sumar los valores tenemos los permisos deseados. Esta opción es más rápida si ya tienes los permisos claros. Hay que tener en cuenta que esta opción solo reemplaza permisos.

Por ejemplo: si queremos los siguientes permisos `rwxr-xr--` se traduce a `chmod 754 directorio` .

Práctica:

Repite la práctica anterior de asignar permisos, pero en vez de usar letras, usa el método octal.

Cambio de propietario

Para cambiar el propietario y el grupo se usa el mismo comando:
`chown [USUARIO]:[GRUPO] directorio` .

Ejercicio: a través del uso de la opción `-h` , averigua como replicar los cambios de permisos de chwon y chmod a todos los subdirectorios.

Creación de Usuarios:

adduser / **useradd:** Existen diferentes comandos para agregar usuarios en Linux, como "adduser" o "useradd".

- `adduser nuevo_usuario` : Crea un nuevo usuario llamado "nuevo_usuario" con un

directorio personal por defecto en /home.

Ejercicio: Usa `man` para ver la diferencia entre los dos comandos.

passwd: Después de crear un usuario, es importante establecer una contraseña para la cuenta. El comando "passwd" se utiliza para esto:

- `passwd nuevo_usuario` : Establece una nueva contraseña para el usuario "nuevo_usuario".

Cuestión: Es posible que el comando anterior fallase por una simple razón. ¿Quien crees que es el úncio capaz de añadir usuarios, grupos, cambiar contraseñas, etc?

Eliminación de Usuarios:

userdel: Para eliminar un usuario, utilizamos el comando "userdel".

- `userdel usuario_a_eliminar` : Elimina el usuario "usuario_a_eliminar".

Ejercicio: Usando `ls /home/` , mira los directorios existentes. ¿Notas algo raro? Usa `man` para averiguar la opción que permite solucionarlo.

Gestión de Grupos:

Para ver los grupos que pertenece un usuario, podemos usar el comando `groups <usuario>` .
Cuestión: ¿Serías capaz de identificar los grupos primarios y los secundarios del usuario?

groupadd: Para crear un nuevo grupo en Linux, utilizamos el comando "groupadd". Por ejemplo:

- `groupadd nuevo_grupo` : Crea un nuevo grupo llamado "nuevo_grupo".

Cuestión: ¿Serias capaz de adivinar cual es el comando para eliminar grupos?

Modificación de Usuarios:

`usermod` : El comando "usermod" se utiliza para modificar las propiedades de un usuario existente.

Un usuario puede tener dos tipos de grupos: primario y secundario. La diferencia reside en que cuando un usuario crea un archivo, se le asigna automáticamente el grupo propietario como el grupo principal del usuario.

- `usermod -g <grupo> <usuario>` : Para cambiar el grupo primario.

- `usermod -aG <grupo> <usuario>` : Para añadir grupo secundario al usuario.

Ejercicios: Usa `man` o la opción `-h` para averiguar las siguientes opciones:

- Cambiar el nombre de usuario
- Quitar grupo secundario del usuario
- Cambiar directorio `home`
- ¿Que crees que pasaría si no usasemos la opción `-a` junto con `-G` ? Si te sientes atrevido, lo puedes probar.
- Cambiar la contraseña de usuario.
- Cuestión complicada: ¿Cual crees que es la diferencia entre `passwd <usuario>` y `usermod -p <contraseña> <usuario>` ? ¿Por qué hay dos comandos diferentes que hacen técnicamente lo mismo?

Visualización de Usuarios y Grupos:

`getent passwd` : Muestra una lista de todos los usuarios en el sistema.

`getent group` : Muestra una lista de todos los grupos en el sistema.

Ejercicios Prácticos:

- Crear un nuevo usuario.
- Establecer una contraseña para el nuevo usuario.
- Agregar el nuevo usuario a un grupo existente.
- Cambiar el nombre del usuario.
- Eliminar el usuario creado anteriormente.

Gestión de permisos de superusuario

Antes que nada, crea un nuevo usuario.
Si se quiere logear como un nuevo usuario en la terminal, podemos usar `su <usuario>` .
Con esto, podemos estar en entorno gráfico con un usuario, mientras que ponemos comandos como otro.

Inicia sesión con el nuevo usuario creado. Ahora, prueba a hacer un comando inofensivo con permisos de superusario, como `sudo ls /` . ¿Que ha pasado?

Esto ocurre porque el nuevo usuario no tiene permisos de administrador, y por lo tanto, no solo no puede usar `sudo` , sino que tampoco puede logearse como superusuario a través de ese usuario.

Para salir del entorno escribe en la CLI `exit` . vemos que hemos vuelto al usuario original.

Ejercicio Práctico: Añadir permisos de superusuario al nuevo usuario creado

¡Alerta, la siguiente parte es complicada! ¡No te desanimes si no se consigue por cuenta propia!
Ahora que hemos vuelto al usuario original, volvemos a tener acceso al comando sudo. El fichero que controla quien tiene permisos de usuario se encuentra es el `/etc/sudoers` .

Importante: ¡este fichero no se edita! Solo vamos a ver el contenido con `sudo less /etc/sudoers/` . Si bajamos un poco, deberíamos de encontrar los usuarios con permisos y los grupos que tienen acceso.

Así que, con ese grupo en mente, simplemente usamos `sudo usermod -aG <usuario> <grupo>` para añadir ese grupo a la lista de grupos secundarios al usuario.

Prueba a volver a conectarte al usuario y a intentar usar un comando con `sudo` para comprobar que puede.

Expansión de shell y expresiones regulares

Expansión de shell

Para facilitar el uso de la terminal, hay unos carácteres especiales que se usan para sustituir valores comunes. Por ejemplo:
`cd ~` se expande a `cd /home/<usuario>`
`mkdir d{1..3}` se expande a `mkdir d1 d2 d3` .

Después de ejecutar el comando anterior, prueba a usar:
```
ls d?
```

La expresión anterior se puede combinar con otras opciones:
```
mkdir dir{1..3,6,8..10}
```

Después de ejecutar el comando anterior, prueba a usar:
```
ls dir?
```

Cuestión: ¿Serías capaz de encontrar la razón por la que no lista uno de los directorios recientemente creados?

Prueba los siguientes comandos:
```
mkdir d
ls d*
```
Cuestión: ¿Que ha cambiado con el uso del anterior comando?

```
mkdir -p {2020..2024}/{001..012}
ls 202?
tree
```

Ejercicio: Es posible que el comando anterior no esté disponible. ¿Como lo solucionarias?

Vamos a probar a combinar diferentes opciones:

```
mkdir -p Asig_{1..3}/{Teoria/Tema_{1..9},Practicas/Lab_{0..8}}
cd & ls
tree
ls Asig_?/*
```

Expresiones regulares

Metacaracteres

`.` : Cualquier carácter.
`^` : Inicio de línea.
`$` : Fin de línea.
`*` : Cero o más ocurrencias del carácter anterior.
`+` : Una o más ocurrencias del carácter anterior.
`?` : Cero o una ocurrencia del carácter anterior.
`[...]` : Cualquier carácter dentro de los corchetes.
`|` : Operador OR.
`(...)` : Agrupa patrones.
`\` : Escapa el siguiente carácter (si es un metacarácter).
`[a-z]` : Cualquier carácter en minúsculas.
`[A-Z]` : Cualquier carácter en mayúsculas.
`[0-9]` : Cualquier dígito.
`{n}` : Exactamente n ocurrencias.
`{n,}` : n o más ocurrencias.
`{n,m}` : Entre n y m ocurrencias.

Ejemplos:

`texto.txt` -> Fichero provisionado por los coordinadores del taller.

1. Buscar una palabra en un archivo
   Utiliza `grep` para buscar la palabra "linux" en un archivo llamado "texto.txt".

   ```
   grep "Linux" texto.txt
   ```

2. Buscar todas las líneas que comienzan con una letra mayúscula

Usando `grep` y expresiones regulares, puedes buscar todas las líneas en un archivo que comiencen con una letra mayúscula.

```
grep '^[A-Z]' texto.txt
```

3. Sustituir una palabra por otra en un archivo con `sed` Para reemplazar todas las instancias de "linux" por "UNIX" en el archivo "texto.txt", puedes usar `sed` . Este comando edita el archivo en el lugar y guarda el original con una extensión de respaldo.

```
sed -i.bak 's/linux/UNIX/g' texto.txt
```

4. Filtrar líneas que contengan números de teléfono
   Si quieres extraer líneas que contengan números de teléfono en un formato específico (por ejemplo, 123-456-7890), puedes usar `grep` de la siguiente manera:

```
grep -E '[0-9]{3}-[0-9]{3}-[0-9]{4}' texto.txt
```

5. Extraer todas las direcciones de correo electrónico
   Usa `grep` para encontrar todas las direcciones de correo electrónico en el archivo.

```
grep -oP "[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}" texto.txt
```

6. Buscar números de teléfono con diferentes formatos
   Encuentra todos los números de teléfono, independientemente de su formato.

```
grep -oP "(\+\d{1,2}\s)?\(?\d{3}\)?[\s.-]?\d{3}[\s.-]?\d{4}" texto.txt
```

7. Extraer fechas en cualquier formato
   Extrae todas las fechas, sin importar el formato en que se presenten.

```
grep -oP "\b\d{2}/\d{2}/\d{4}\b|\b\d{4}-\d{2}-\d{2}\b|\b\d{2} \w{3} \d{4}\b|\b\w{3,} \d{2}, \d{4}\b" texto.txt
```

8. Encontrar todas las URLs
   Busca todas las URLs presentes en el texto.

```
grep -oP "https?://\S+|www\.\S+|\b\S+\.\com\b" texto.txt
```

9. Identificar direcciones IPv4
   Encuentra todas las direcciones IPv4 en el archivo.

```
grep -oP "\b\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}\b" texto.txt
```

10. Filtrar códigos mixtos de letras y números
    Extrae códigos que contienen una mezcla de letras y números.

```
grep -oP "\b[A-Za-z0-9]+\b" texto.txt | grep -P "[A-Za-z].*[0-9]|[0-9].*[A-Za-z]"
```

Ficheros

Para crear un fichero vacío en Linux, podemos utilizar la instrucción `touch` (`touch fichero`).

También podemos crear y modificar el contenido de un fichero con un editor de texto, como `nano` y `vim` (`nano fichero` , `vim fichero`). `nano` es un editor de texto con interfaz de terminal (TUI) que es fácil de usar y común en los sistemas GNU/Linux. `vi` / `vim` es un editor de texto TUI que se encuentra en todas las distribuciones de GNU/Linux, es un programa muy potente, pero un tanto difícil de dominar.

Para mostrar el contenido de un fichero, podemos utilizar `cat` (`cat fichero`). Y para leer el fichero de forma que podamos "paginar" su contenido, podemos usar `less` (`less fichero`).
Otra forma de crear fichero es mediante la redirección de salida estándar (stdout) de una instrucción a un fichero `> fichero` :
`cat > fichero`
`echo hola > fichero` (sobreescritura)
`echo adios >> fichero` (concatenación)

¡Pruébalo con diferentes comandos!
Unos ejemplos interesantes:

- `tree ~/`
- `sudo echo "¡Hola!" > /tmp.log`
  Cuestión: ¿Por qué crees que no funciona?
- `ps -eFL --sort -pcpu > process.log`
  Cuestión: ¿Podrías identificar que significan las opciones en el comando?

Pipes

Los pipes son uno de los superpoderes de las terminales UNIX. Sirven para conectar la salida (stdout) de un comando con la entrada (stdin) de otro. Para crear un pipe entre dos instrucciones, se debe escribir entre ellos el carácter `|` .

Veamos un ejemplo del uso de pipes. Queremos crear un pipe que termine la ejecución del proceso `htop`. Antes de empezar: abre otra terminal y ejecuta `htop`.

Para terminar la ejecución de un proceso desde la terminal, podemos usar la instrucción `kill PID` donde "PID" es el número de proceso que queremos terminar.
Para mostrar la información de los procesos en ejecución podemos ejecutar la instrucción `ps aux`. Los PIDs de los procesos están en la segunda columna de la salida del comando anterior. Podríamos copiarnos el PID de `htop` y ejecutar `kill PID_HTOP`, pero queremos que sea la máquina la que lo haga.

Podemos pasar la salida de la instrucción `ps aux` a la entrada de la instrucción `grep htop` (que sirve para buscar el texto `htop` en el texto/fichero que le pasemos a `grep`), de forma que `ps aux | grep htop` nos devolverá todas las líneas de la salida de `ps aux` donde exista la cadena de texto "htop".

Ahora que tenemos la información de los procesos que queremos terminar, tenemos que extraer de ella sólo lo que nos interesa: el PID. Como sabemos que se encuentra en la segunda columna de la salida, la siguiente instrucción de nuestro pipe será `awk '{print $2}'`, que muestra la segunda columna de cada línea de entrada.

Nuestro pipe ahora devuelve sólo los PIDs de los procesos que queremos terminar (los que contenían la cadena "htop"). Para terminar estos procesos debemos pasarlos al comando `kill` (`kill PID`). Como kill no acepta entrada estándar (sólo acepta una serie de PIDs) tenemos que usar un comando que tome lo que viene por entrada estándar y se lo dé a otro comando como argumento. El comando que buscamos es `xargs`. Para entender cómo funciona `xargs` ejecuta `ls hola` y `echo hola | xargs ls` para ver cómo el resultado es exactamente el mismo. Lo que hace `xargs` es coger la salida de `echo hola`, que es "hola" y pasársela a `ls` como argumento, de forma que se ejecuta `ls hola`. También puedes pasarle varias líneas de entrada estándar a `xargs`. Intenta entender qué pasa cuando haces `echo "hola\nadios" | xargs ls`. Volviendo a nuestro pipe, bastaría con añadirle la instrucción `xargs kill` para conseguir nuestro objetivo, de forma que el pipe quedaría así:

```
ps aux | grep htop | awk '{print $2}' | xargs kill
```

Verás que ejecutar el pipe anterior da un error. Este error ocurre porque la instrucción `ps aux | grep htop` lista dos procesos que contienen en su nombre la cadena "htop", uno de ellos es el propio htop y el otro corresponde al comando que estamos ejecutando (`grep htop`), que también contiene la cadena "htop". Como se trata de una instrucción `grep`, podemos eliminar esa línea con `grep -v grep` que invierte la búsqueda, mostrando, en este caso, sólo las líneas que no contengan la cadena "grep". El pipe quedaría así:

```
ps aux | grep htop | grep -v grep | awk '{print $2}' | xargs kill
```

El siguiente pipe lista los cinco ficheros más grandes (con sus tamaños) a partir del

directorio actual:
`find . -type f | grep a | xargs ls -ldh | awk '{print $5"\t"$9}' | sort -gr | head -n5`

Alias y scripts

Alias

Un alias es un nombre alternativo o un atajo a una instrucción o serie de instrucciones.
`alias duh="du -hd 1 . | sort -rh"`
Podemos crear un alias con el pipe de la sección anterior de esta forma: `alias killfind="ps aux | grep htop | grep -v grep | awk '{print $2}' | xargs kill"`
Los alias no persisten entre distintas sesiones de terminal. Si cerramos la terminal donde hemos creado el alias perderemos el alias. Para hacer el alias persistente tendremos que añadirlo a nuestro `~/.bashrc` (o equivalente). El fichero `~/.bashrc` es un script de configuración de la shell que se ejecuta cada vez que abrimos una terminal. Por tanto, para hacer que el alias persista basta con añadir la instrucción de creación del alias al fichero.

Scripts

Un script es un programa, es decir, una serie de instrucciones que se ejecutan para llevar a cabo una tarea. La gracia de los scripts es que pueden hacer uso de otros programas, programas como los que hemos visto a lo largo de este taller.

Hola, Mundo!

Vamos a crear un simple script de "Hola, Mundo!". Para ello debemos crear un fichero:
`nano script.sh` (la extensión no importa, podría no tenerla)

El contenido del fichero debe ser el siguiente:

```
#!/bin/sh
echo "Hola, Mundo!"
```

La primera línea se llama la línea de Shebang, y sirve para indicarle al sistema con qué shell debe interpretar las instrucciones del script. La segunda línea muestra "Hola, Mundo!"" por pantalla.

Ejercicio: Ahora debemos hacer el script ejecutable. Recordando el apartado de permisos, ¿cual sería la opción más simple para permitir que el propietario tenga permisos de ejecución?

Ahora ejecutaremos el script: `./script.sh` (`bash script.sh` ó `sh script.sh`)

Ahora modificaremos el script para que guarde en una variable la salida de un comando, de forma que el script quedará así:

```
#!/bin/sh
numero="$(ls ~ | wc -l)"
echo "Hay $numero ficheros y directorios en HOME"
```

Ahora modificaremos el script con un `if-else` para que se muestre una cosa u otra en función del valor de la variable numero:

```
#!/bin/sh
numero="$(ls ~ | wc -l)"
if [ "$numero" -gt 10 ]; then
        echo "Hay más de 10 ficheros y directorios en HOME"
else
echo "No hay más de 10 ficheros y directorios en HOME"
fi
```

Ahora modificaremos el script para que tome un argumento, de forma que muestre el número de ficheros/directorios en el directorio que pasemos como argumento:

```
#!/bin/sh
numero="$(ls "$1" | wc -l)"
echo "Hay $numero ficheros y directorios en $1"
```

Para ejecutar un script con argumentos se hace como para un comando normal: `./script.sh /home`

Extendiendo el pipe

Si quisieramos crear un script a partir del pipe que terminaba la ejecución del programa "htop" de la sección anterior, podríamos pasarle el nombre del programa a terminar de ejecutar como argumento:

```
#/bin/sh
ps aux | grep "$1" | grep -v grep | awk '{print $2}' | xargs kill
```

Gestión de paquetes

```
#!/bin/bash

# Función para actualizar la lista de paquetes disponibles
actualizar_paquetes() {
        echo "Actualizando la lista de paquetes..."
        sudo apt update
}

# Función para buscar paquetes
buscar_paquete() {
        echo "Buscando el paquete: $1"
        apt search "$1"
}

# Función para instalar un paquete
instalar_paquete() {
        echo "Instalando el paquete: $1"
        sudo apt install -y "$1"
}

# Función para eliminar un paquete
eliminar_paquete() {
        echo "Eliminando el paquete: $1"
        sudo apt remove -y "$1"
}

# Función para comprobar actualizaciones de paquetes instalados
comprobar_actualizaciones() {
        echo "Comprobando actualizaciones de paquetes..."
        apt list --upgradable
}

# Menú principal
echo "Gestión de paquetes de Debian"
echo "1. Actualizar lista de paquetes"
echo "2. Buscar paquete"
echo "3. Instalar paquete"
echo "4. Eliminar paquete"
echo "5. Comprobar actualizaciones de paquetes"
read -rp "Seleccione una opción: " opcion

case $opcion in
        1) actualizar_paquetes ;;
        2) read -rp "Paquete a buscar: " paquete ; buscar_paquete "$paquete"
        3) read -rp "Paquete a instalar: " paquete ;  instalar_paquete "$paqu
        4) read -rp "Paquete a eliminar: " paquete ; eliminar_paquete "$paque
        5) comprobar_actualizaciones ;;
        *) echo "Opción no válida." ;;
esac
```

Anime desde la terminal

```
#!/bin/sh

wget -p "https://jkanime.bz/one-piece/$1/" -P /tmp

url="$(grep -A 1 video /tmp/jkanime.bz/um.php* | grep url | cut -d"'" -f2 | h

rm -rf /tmp/jkanime.bz

mpv "$url"
#yt-dlp -o "OnePiece.%(ext)s" "$url"
```

Ejercicios Extra

Esta colección de ejercicios se creó a base de la posibilidad de los usuarios avanzados acabasen el taller. No esperamos que los principiantes puedan entender o llegar a este apartado, es normal que sea confuso.

Systemd e Inits

Init es la terminología de "inicialización", siendo no solo el primer proceso al arrancar, sino que también controla los scripts de inicialización.
El Init más popular entre servidores y sistemas linux es Systemd, y el comando para administrarlo es `systemctl`.

Ejercicios:

- Prueba a ejecutar `systemctl` sin parámetros. ¿Que puedes ver?
- Ahora prueba a ejecutar `systemctl status`. ¿Que crees que estás viendo?
- Ahora prueba `systemctl status firewalld`. ¿Serías capaz de identificar cada campo?
- Mira el directorio `/usr/lib/systemd/system`. ¿Que puedes encontrar?
- Ahora mira el directorio `/etc/systemd/system`. ¿Cual crees que es la diferencia?
- Mira el contenido de cualquier fichero `.service`. ¿Que puedes ver?

Bajo linux, los procesos en en ejecución de segundo plano se llaman `daemon`. Por eso, algunos programas de fondo añaden una `d` al final del nombre, o el directorio de configuración del servicio acaba con `.d`.

Utilizando el init y `systemctl` podemos administrar y crear scripts que se ejecuten en el inicio del sistema.

Creación de scripts de inicio

Primero vamos a crear un script simple que nos ponga en el directorio de usuario un fichero con la data exacta que se ejecutó el script.
`/home/<usuario>/script.sh`

```
#!/bin/bash
echo "¡Script ejecutado el $(date)!" >> /home/<usuario>/exec.log
```

Ahora que tenemos el script (recuerda hacerlo ejecutable), vamos a crear un archivo `.service` en `/usr/lib/systemd/system`.

Hay tres apartados a tener en cuenta dentro de un archivo de servicio:

- `[UNIT]`
  - Campos como nombre, descripción y dependencias.
- `[SERVICE]`
  - Tipo de arranque y localización de script/binario de ejecución. Necesariamente con ruta absoluta. [3]
- `[INSTALL]`
  - Le dice a systemd en que momento de la iniciación del sistema ejecuta el servicio. Las opciones más usadas son `multi-user.target`, `graphical.target`, `network-online.target`.

Con esto, podemos escribir nuestro script de inicio:

```
[Unit]
Description=Script de inicio

[Service]
#Forking es el más común, pero oneshot nos sirve.
Type=oneshot
ExecStart=/home/<usuario>/script.sh

[Install]
WantedBy=multi-user.target
```

¡Importante! No hay comentarios in-line. Eso significa que cualquier comentario debe ser hecho exclusivamente en una línea nueva. Si no, va a fallar.

Para que el sistema detecte el nuevo servicio, debemos usar `systemctl daemon-reload`.
Ahora si usaríamos `systemctl status <nombre-de-servicio>` y nos debería de salir el estado.
Para permitir que un servicio se inicie en el boot, usamos la opción `enable` del `systemctl`. Para que no haga falta reiniciar, usa `start` y mira si el fichero se ha

creado.

Nota: esta no es la única manera para crear scripts de inicio. Pero con esto puedes crear scripts para que se inicien programas, y administrarlos a través de `systemctl`.

Tareas Cron

Cron es una herramienta para automatizar tareas a través de scripts que se ejecutan automáticamente después de un tiempo especificado por el usuario.

Primero, miramos si tenemos el servicio en marcha con `systemctl status cron` **P.D: el nombre podría ser diferente**. El comando para cron es `crontab`

Ejercicio: Usando `man`, ¿serías capaz de encontrar la opción para editar las tareas del usuario?

Nota: El editor que usa cron para las tareas puede llegar a ser distinto en cada sistema, así que si de repente no puedes escribir una vez abierto, es posible que estés en el editor `vi`, `vim` o parecidos. Eso se puede cambiar usando variables del entorno.

Vamos a poner un ejemplo:

```
* * * * * /comando.sh
```

1. El primer asterisco significa minutos
2. El segundo horas
3. Dias, o dias del mes
4. Meses
5. Dias de la semana (0 = Domingo, 6 = Sabado) [4]

Este script se ejecutaría cada minuto desde el inicio de cron (generalmente al inicio). Esta tambien es una alternativa a los scripts de inicio, con usar el alias `@reboot` en vez de especificar el tiempo, podemos que se ejecute una vez al inicio.

Enlaces externos

Este apartado aporta información adicional a los usuarios que quieran saber más sobre un tema, no es necesario para completar el taller.

1. The Unix and Linux Philosophy - http://www.linux-databook.info/?page_id=2178 (http://www.linux-databook.info/?page_id=2178) ↩

2. Linux Directory Structure Explained for Beginners - https://linuxhandbook.com/linux-directory-structure/ (https://linuxhandbook.com/linux-directory-structure/) ↩

3. Systemd Type Options ArchiWiki - https://man.archlinux.org/man/systemd.service.5#OPTIONS (https://man.archlinux.org/man/systemd.service.5#OPTIONS) ↩

4. Crontab, Quick Reference - https://www.adminschoice.com/crontab-quick-reference (https://www.adminschoice.com/crontab-quick-reference) ↩