Presentación del curso

Hansel Tepal, Francisco Galindo
Estudiantes de Ingeniería en Computación

Universidad Nacional Autónoma de México
Facultad de Ingeniería

Curso de SysAdmin, 2025-1

Objetivo del curso

El curso está dirigido a personas sin conocimientos previos sobre el uso de sistemas Linux o tipo UNIX. Este tipo de personas tienen la necesidad o interés de utilizar este sistema en su día a día.

Por ello, el objetivo es que los alumnos sean capaces de interactuar con un sistema Linux para uso personal.

Temario

- Introducción al Software Libre e historia del sistema Linux. [30 min]
- Instalación de un ambiente Linux [1:30 h]
- Sistema de Archivos [1 h]
- Arquitectura de un sistema tipo UNIX [30 min]
- Shell [30 min]
- Shell (cont.) [1:30 h]
- Gestión de paquetes (nativos y Flatpak) [30 min]
- Edición de archivos [1 h]
- Gestión de servicios (SystemD) [1 h]

Requisitos

- Una computadora personal propia.
- 120 GB de espacio libre en su disco duro.
- Un respaldo de TODOS los archivos importantes de su computadora.
- De ser posible, una intalación ya realizada de un sistema Linux.

Metodología de enseñanza

Se utiliza un sistema teórico-práctico en el cual inicialmente se imparte la parte teórica para posteriormente asignar ejercicios de reforzamiento.

Evaluación

El curso se evalúa por medio de asistencia (mínima del 80%). Este es un requisito para obtener la constancia del curso.

Recursos

- G. Wolf, Fundamentos de sistemas operativos.
- B. Ward, How Linux Works.
- W. Shotts, The Linux command line: a complete introduction.
- OccupyTheWeb, Linux Basics for Hackers.

Introducción al Software Libre e historia del sistema Linux

Hansel Tepal, Francisco Galindo
Estudiantes de Ingeniería en Computación

Universidad Nacional Autónoma de México
Facultad de Ingeniería

Curso de SysAdmin, 2025-1

Información del tema

Tiempo estimado

Aproximadamente 30 minutos de clase, con participación de los alumnos para compartir su opinión y dudas respecto al tema.

Objetivos

- Familiarizarse con la idea del Software Libre, lo que representa y su historia.
- Conocer, de manera resumida, la historia de Linux.

Tiempos de UNIX

En 1969, el sistema operativo UNIX fue creado por Ken Thompson y Dennis Ritchie, que trabajaban para AT&T en *Bell Labs*

Este sistema operativo, junto a la filosofía de desarrollo que lo acompañaba, revolucionaron el mundo de la computación.

Thompson y Ritchie

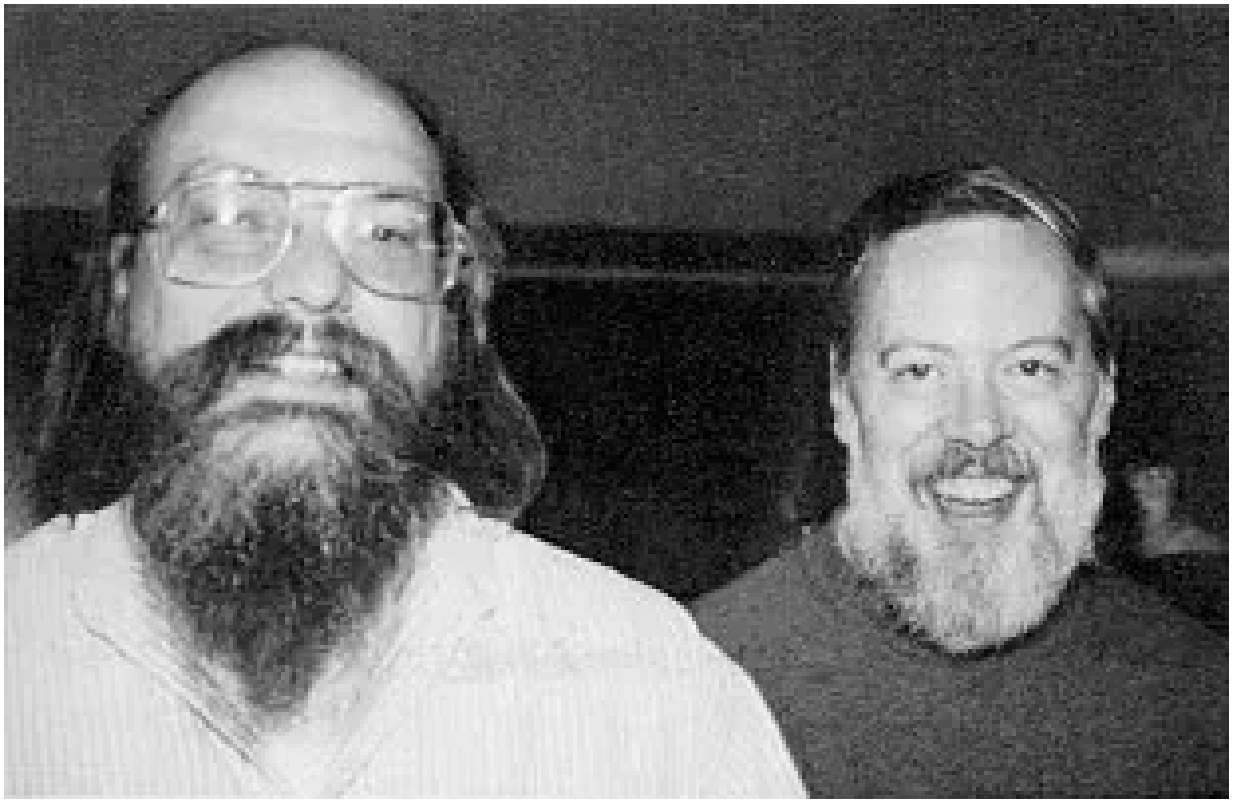

Figura 1: Ken Thompson (izquierda) y Dennis Ritchie (derecha).

Thompson y Ritchie

Figura 2: Thompson y Ritchie trabajando en una PDP-11.

Movimiento de Software Libre

Fue iniciado por Richard Stallman (RMS) en 1983, cuando inició el proyecto GNU (GNU's Not Unix).

En 1985. RMS creó la Fundación del Software Libre, que tiene la misión de avogar y compartir información sofre el Software Libre.

RMS

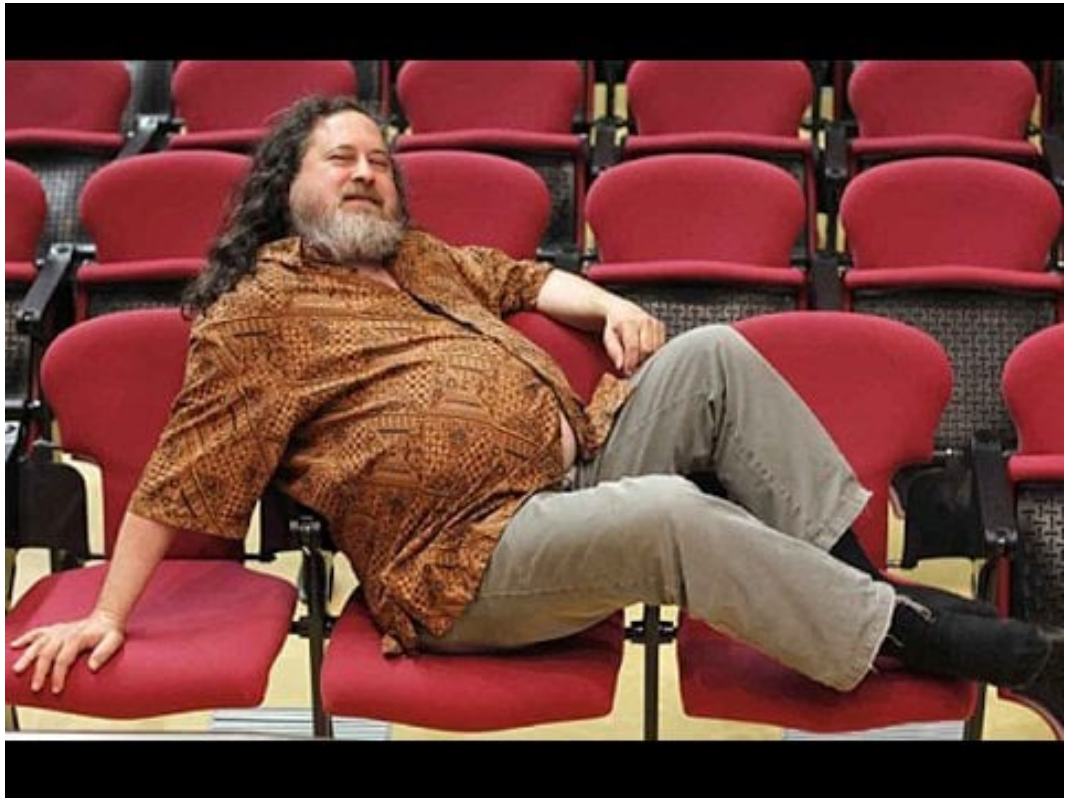

Figura 3: Richard Sallman haciendo no sé qué.

Las cuatro libertades

- Libertad de utilizar el programa como se desee, para cualquier propósito (libertad 0).
- Libertad de estudiar el funcionamiento del programa y modificarlo para que este haga lo que el usuario desee (libertar 1).
- Libertad de redistribuir copias del software para ayudar a los demás (libertad 2).
- Libertad de distribuir copias de versiones modificadas del software (libertad 3).

Berkeley vs AT&T

La Universidad de California en Berkeley tenía una licencia para el código de UNIX proveída por AT&T, gracias a la cuál se hizo mucha investigación relacionada con sistemas operativos.

Esta investigación se incorporó al sistema para dar lugar a la Berlekey Software Distribution (BSD).

Berkeley vs AT&T

Para liberarse de las restricciones impuestas por la licencia de AT&T, estudiantes de Berkeley empezaron a eliminar el código original y reemplazarlo con código nuevo, bajo una licencia libre.

Berkeley vs AT&T

Cuando se realizó el *port* de BSD hacia procesadores i386, que se creía que ya no contenía propiedad intelectual de AT&T, esta última inició un proceso legal para detenerlos que duró entre 1992 y 1994.

La pieza faltante de GNU

El proyecto de RMS consiguió crear muchas de las partes necesarias para conformar un sistema operativo en toda regla, como una shell y las *coreutils*.

Todavía había algo que faltaba, un pequeño detalle, casi insignificante. Faltaba un kernel para GNU.

Linux

Linus Torvalds, un estudiante de Ciencias de la Computación finlandés, empezó un proyecto de crear un kernel debido a su descontento con el kernel del sistema Minix (creado por Andrew Tanenbaum).

Lo hizo él porque el kernel de GNU (Hurd) no estaba listo todavía.

GNU / Linux

Dado que Torvalds había creado un kernel compatible con las partes que GNU ya había creado, se incorporó el kernel de Linux al sitema GNU dando lugar al sistema que ahora llamamos GNU / Linux.

Gracias al problema legal de BSD, GNU / Linux se volvió la única opción de sistema operativo libre, lo que impulsó su popularidad.

GNU / Linux

Prácticamente todas las distribuciones de Linux (pronto hablaremos de eso) son un sistema GNU / Linux.

Distribuciones de Linux

Dada la naturaleza libre de GNU / Linux, varios grupos de personas empaquetan programas y configuraciones extra para así distribuir un sistema operativo completamente funcional.

Estos sistemas ya preparados son conocidos como distribuciones (Distros).

Diferencias entre distribuciones

En general, lo que separa a una distribución de otra es:

- La gente que está detrás
- La manera en la que se instala software en el sistema
- Las configuraciones predeterminadas
- La agenda de lanzamientos de nuevas versiones

Por lo demás, un sistema Linux es un sistema Linux.

Instalación de Linux

Hansel Tepal, Francisco Galindo
Estudiantes de Ingeniería en Computación

Universidad Nacional Autónoma de México
Facultad de Ingeniería

Curso de SysAdmin, 2025-1

Información del tema

Tiempo estimado

Aproximadamente 1:30 horas de clase, donde los alumnos seguirán las instrucciones dadas para poder insalar el sistema operativo.

Objetivos

- Al finalizar esta clase, o antes de empezar la otra, el alumno contará con una instalación de Linux utilizable en su computadora.

Distribución a instalar

Para esta clase, se instalará Fedora Linux, pues es la distribución con la que se tiene más experiencia en este laboratorio.

Realmente cualquier distribución de su preferencia funcionará, así que si deseas usar otra, puedes hacerlo. También puedes pedirnos ayuda con esa instalación.

¿Hiciste tu respaldo?

No es común que se pierdan datos al instalar un nuevo sistema operativo en tu computadora, pero cometer algún error puede derivar en la pérdida de información en el disco duro.

¡Respalda todos los archivos que te importen!

Crea tu medio de arranque

Descarguemos una ISO de Linux y utilicemos un programa como *Rufus*[1] para cargarla en una memoria USB.

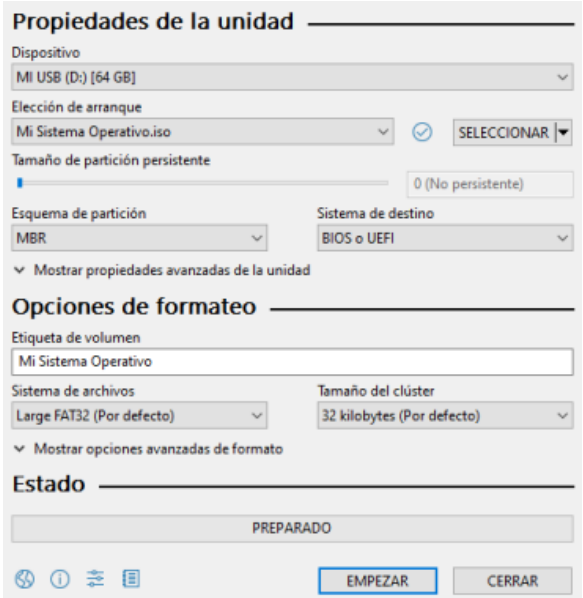

Figura 1: Eligiendo memoria para utilizar en *Rufus*.

[1] https://rufus.ie/es/

Conceptos fundamentales y jerarquía de directorios

Hansel Tepal, Francisco Galindo
Estudiantes de Ingeniería en Computación

Universidad Nacional Autónoma de México
Facultad de Ingeniería

Curso de SysAdmin, 2025-1

Información del tema

Tiempo estimado

Aproximadamente 1 hora de clase, repartido en secciones de explicación teórica y pequeños ejercicios para reforzar el aprendizaje.

Objetivos

- Conocer la organización canónica de los directorios en un sistema de archivos tipo UNIX.
- Que los alumnos estén acostumbrados al paradigma de UNIX donde todo es un archivo.

Lo más básico de la terminal

Abre tu programa de terminal. Ahí verás el prompt: la línea que termina con un $.

```
[usuario@hostname ~] $
```

Lo más básico de la terminal

Una vez aparecido el prompt, puedes escribir comandos. Un comando es un programa que hace algo en el sistema. Este programa puede ser de cualquier tipo.

Ejemplo: utiliza el comando `date`:

```
$ date
```

```
Thur Sep  5 01:05:16 PM CST 2024
```

Lo más básico de la terminal

Presiona Enter y no escribas el $, este último sólo indica al usuario que puede escribir un nuevo comando y se utilizará para indicar que cierta línea de código se trata de un comando.

Archivos

Un archivo en Linux es la manera en la que el sistema operativo almacena datos.

Un archivo físico tiene un nombre, contenidos, un lugar en el que se almacena, así com ouna serie de datos administrativos sobre quién es su dueño y su tamaño. En el sistema operativo estos atributos también existen.

Archivos

En un archivo tu puedes guardar la secuencia de bits que a tí se te antoje, puedes guardar el texto de una carta, de un reporte, código fuente de algún programa, o una imagen.

Comandos muy comúnes para trabajar con archivos

Empecemos por un editor de texto

Gracias a un editor de texto puedes... editar texto.

Utilizaremos `nano`, pues está hecho para ser fácil de usar.

```
$ nano nombre-de-un-archivo
```

Comandos muy comúnes para trabajar con archivos

Una vez abierto `nano`, puedes escribir tu texto directamente. Tambiénuedes mover tu cursor con las flechas direccionales

Para guardar tus cambios, presiona `Ctrl+O` (`^O`). Escoge el nombre con el que se guadrará y sal con `^X`.

Pequeña actividad

Utiliza `nano` para crear y llenar tres archivos de texto:

- `chiste`: Un chiste.
- `basura`: Cualquier cosa que quieras.
- `tareas`: Una lista de tareas que tengas que hacer.

Comandos muy comúnes para trabajar con archivos

Puedes ver una lista con los nombres de los archivos de cierto lugar con `ls`:

```
$ ls
```

```
basura  chiste  tareas
```

Puede que haya más archivos que ya existían desde antes.

Comandos muy comúnes para trabajar con archivos

Muchos comandos, como `ls`, tienen opciones que modifican su comportamiento por defecto.

Las opciones se escriben después del nombre del programa a ejecutar y generalmente llevan un - por delante.

¿Qué hace el siguiente comando?

```
ls -l
```

Comandos muy comúnes para trabajar con archivos

Generalmente, puedes pasarle muchas opciones a un mismo comando, aunque depende ya de cada programa:

```
$ ls -l -a
```

Que es lo mismo que:

```
$ ls -la
```

Comandos muy comúnes para trabajar con archivos

A veces no es tan conveniente usar un editor para ver los contenidos de unarchivo, especialmente si no vas a editar nada.

Puedes mostrar los contenidos de un archivo con el comando `cat`:

```
$ cat tareas
```

```
Estudiar para el examen de Bases de Datos
Comprar 1 kilo de huevo
Blah
Hlab
```

Moviendo, borrando, copiando archivos

Puedes mover un archivo de un lugar a otro con `mv`. Este comando también se puede utilizar para "renombrar" un archivo:

```
$ mv basura tesoro
```

Moviendo, borrando, copiando archivos

Para copiar un archivo se utiliza el comando cp. Todos los contenidos de un archivo se copian al otro

```
$ cp tesoro tesoro-respaldo
```

Moviendo, borrando, copiando archivos

Si quieres borrar un archivo, se utiliza el comando `rm`:

```
$ rm tesoro
$ ls

chiste  tareas  tesoro-respaldo
```

Pequeña actividad

A partir de los archivos creados en la actividad anterior, cambia sus nombres de manera que:

- El archivo `chiste` tenga los contenidos que antes tenía `tareas`
- El archivo `tareas` tenga los contenidos que antes tenía `basura`
- El archivo `basura` tenga los contenidos que antes tenía `chiste`

Y que al final sólo estén esos archivos (además de los que ya existían desde el principio de la clase).

Comandos muy comúnes para trabajar con archivos

Generalmente, los comandos en Linux son *silenciosos*, es decir, no envían información de salida si todo sale bien.

En muchos casos, cuando sale información en pantalla es porque son errores:

```
$ rm archivo-inexistente
```

```
rm: cannot remove 'archivo-inexistente': No such file or directory
```

Otros comandos

Para saber cómo se utiliza un comando, siempre es buena idea utilizar el comando `man` para ver un pequeño manual sobre el uso del comando:

```
$ man ls
```

Otros comandos

Si quieres conocer la cantidad de líneas, palabras o caracteres en un archivo, utiliza `wc`:

```
wc tareas
```

Con las opciones `-l`, `-c`, `-w`, puedes elegir qué contar.

Otros comandos

Un comando muy utilizado para el procesamiento simple de texto es `grep`. Es un programa que busca un patrón en las líneas de un archivo de texto.

El nombre viene de un comando del antiguo editor `ed`: `g/regular-expression/p`.

Otros comandos

Supongamos que tenemos este archivo de texto:

```
$ cat quijote
```

```
En un lugar de La Mancha, de cuyo nombre no quiero acordarm
no ha mucho tiempo que vivía un hidalgo de los de lanza y a
adarga antigua, rocín flaco y galgo corredor.
```

Puedo buscar la palabra "*un*" con

```
$ grep un quijote
```

Otros comandos

`sort` es un comando que, como su nombre lo indica, ordena los contenidos de un archivo e imprime el resultado:

```
cat arhivo
```

```
c
f
o
q
a
```

```
sort arhivo
```

```
a
c
f
o
```

Otros comandos

`sort` tiene algunas opciones muy útiles cuando se quiere ordenar un archivo por un criterio distinto al lexicográfico:

Comando	Significado
`sort -r`	Ordena en orden inverso
`sort -n`	Ordena en orden numérico
`sort -n`	Ordena en orden numérico
`sort -h`	Ordena tamaños (99 K antes de 1 G)

Otros comandos

Los comandos `head` y `tail` son útiles para mostrar sólo la primera o la última parte de un archivo, respectivamente:

```
cat archivo-largo
Elit aliquid quas ipsa
facere
facilis sequi
similique?
Laboriosam
iusto ipsam
...
```

```
$ head -n 3 archivo-largo
```

```
Elit aliquid quas ipsa
facere
facilis sequi
```

Directorios

En el sistema puede haber muchos archivos con el nombre `tareas`, pero no en el mismo lugar.

La manera en la que el S.O. puede distinguir cada uno de los archivos llamados `tareas` es mediante la grupación de archivos dentro de *directorios* (carpetas, folders, . . .) y *subdirectorios* (un directorio dentro de otro).

Directorios

Todos los archivos del sistema operativo tienen una organización lógica con la forma de un árbol.

Cada nodo de el árbol es un archivo que, en caso de ser un directorio, puede a su vez ser la raíz de otro árbol.

A la raíz del árbol entero se le conoce como el directorio raíz (/).

La jerarquía de directorios

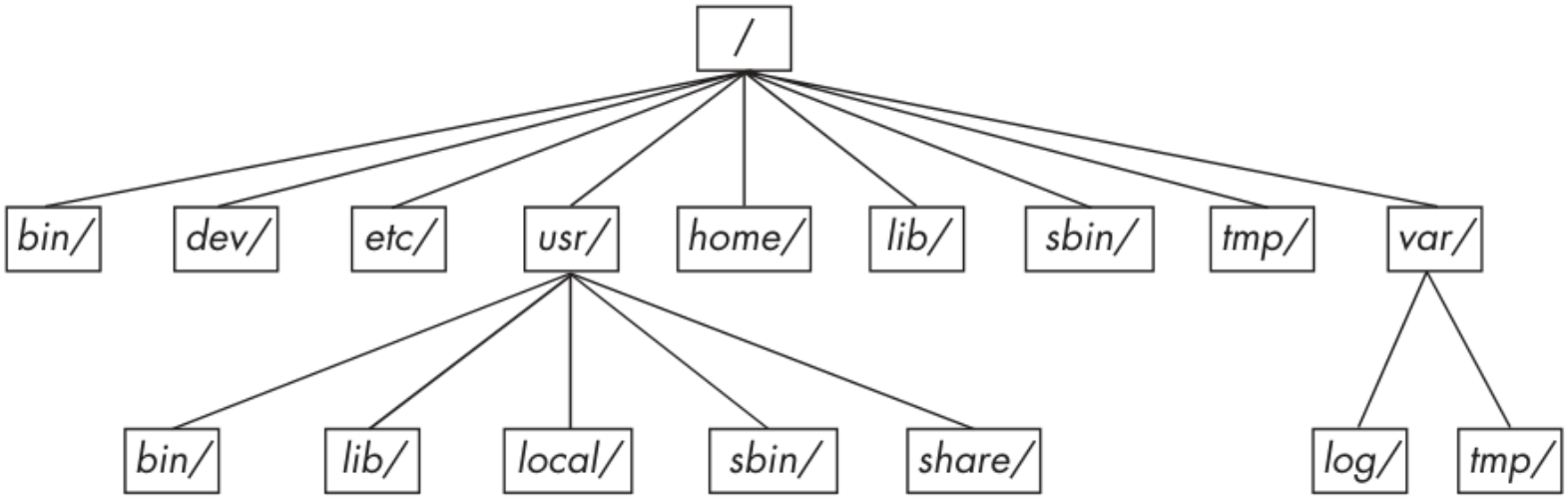

Figura 1: Jerarquía esencial de directorios en un sistema Linux.

Directorios

Cada usuario real del sistema tiene un directorio personal (casi siempre dentro del directorio `/home`), que contiene todos sus archivos.

Cuando inicias una sesión en la shell (cuando abres la terminal), empiezas trabajando dentro de tu directorio personal.

Directorios

Puedes cambiar el directorio en el que trabajas (*working directory*) con el comando `cd` (`pwd` es un comando que muestra tu *working directory*)

```
$ pwd

/home/paco

$ cd Downloads

/home/paco/Downloads
```

Directorios

De esta manera, puedes guardar archivos con el mismo nombre pero dentro de diferentes directorios para que no haya conflictos.

Directorios

Crea un nuevo directorio con el comando `mkdir`

```
$ mkdir nuevo-dir
```

Directorios

A pesar de que los directorios son archivos, para eliminar un directorio con contenidos, no puedes utilizar el comando `rm` así nada más. Necesitas hacer un eliminado recursivo:

```
rm -r nuevo-dir
```

Rutas

Una *ruta* es una manera de identificar cierta parte del árbol de archivos del sistema operativo. Una ruta consiste de una serie de nombres de directorios separados por diagonales (/) para indicar el recorrido que se debe hacer para llegar de un punto a otro.

Rutas abolutas

Una ruta absoluta inicia desde el directorio raíz (/) y describe el camino completo para llegar a cierto sitio:

`/home/paco/Pictures/vacaciones/museo.png`

Las rutas relativas siempre empiezan con una / (el directorio raíz)

Rutas relativas

En una ruta relativa se inicia desde el directorio de trabajo actual (puedes verlo con `pwd`) y describen cómo llegar a cierto sitio *desde* el *working directory*. Por ejemplo, si se trabaja desde la carpeta personal de un usuario:

`/home/paco/`

Escribir la ruta `Pictures/vacaciones/museo.png` es una ruta relativa equivalente a la ruta absoluta `/home/paco/Pictures/vacaciones/museo.png`

La jerarquía de directorios

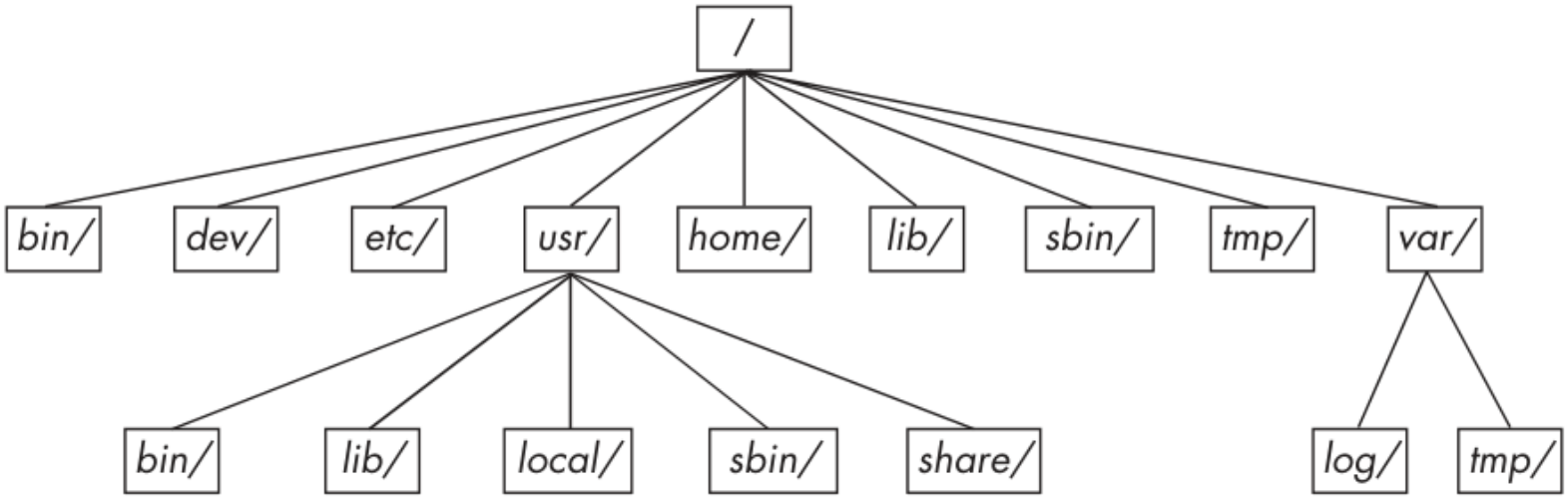

Figura 2: Jerarquía esencial de directorios en un sistema Linux.

La jerarquía de directorios

Aquí hay algunos de los *subdirectorios* más importantes dentro de la raíz.

- `/run` Contiene datos de tiempo de ejecución, archivos de socket, algunas bitácoras.
- `/sys` Similar a `/proc`, provee interfaces para dispositivos y el sistema.
- `/sbin` Lugar para ejecutables del sistemas, son binarios que se suelen solo utilizar mediante el usuario root.

La jerarquía de directorios

- `/tmp` Suele ser pequeño, contiene archivos temporales, todos tienen acceso a este directorio. No debe tener archivos importantes pues el sistema suele limpiarlo constantemente.
- `/usr` Contiene una copia similar a `/`, con subdirectorios como `/usr/bin`, `/usr/lib` y este tipo de archivos. Se mantiene separado históricamente para mantener `/` chico.
- `/var` Contiene datos variables a lo largo del tiempo de los programas, cosas como bitácoras, cache, etc.

Otros subdirectorios

- `/boot` Contiene los archivos del kernel y lo relacionado con el arranque.
- `/media` Un punto de montado de dispositivos que se pueden remover como memorias usb o sd.
- `/opt` Contiene software de terceros # Redirección de entrada y salida

Usuarios, grupos y permisos

Hansel Tepal, Francisco Galindo
Estudiantes de Ingeniería en Computación

Universidad Nacional Autónoma de México
Facultad de Ingeniería

Curso de SysAdmin, 2025-1

Información del tema

Tiempo estimado

Aproximadamente 1 hora de clase, repartido en secciones de explicación teórica y pequeños ejercicios para reforzar el aprendizaje.

Objetivos

- Aprender la utilidad de la existencia de usuarios, grupos y permisos en un sistema Linux
- Aprender a crear, modificar y eliminar usuarios y grupos del sistema.
- Se podrán interpretar y asignar diferentes permisos a los archivos del sistema de archivos.

Usuarios

Linux se trae consigo la noción de UNIX sobre *usuarios*. Un usuario es una entidad que puede ejecutar procesos y ser dueño de archivos. Se identifican mediante su `UID`.

Comúnmente se crea un usuario para cada persona que utilice la computadora (*normal user*).

Ciertos programas también crean sus propios usuarios para realizar sus tareas (*system user*).

Usuarios

La creación de usuarios permite configurar los permisos de los archivos de manera que sólo ciertos usuarios puedan hacer ciertas cosas, dependiendo de la jerarquía o reglas que se quieran implementar.

Usuarios

Puedes ver el `uid` y los `gid` (relacionado con los grupos) de tu usuario con el comando `id`:

```
uid=1000(paco) gid=100(users) groups=100(users),1(wheel),..
```

El usuario root

El usuario más importante del sistema es `root`. Este es el usuario con mayores privilegios. También se le conoce como administrador o *superuser*.

Advertencia

El usuario `root` puede hacer todo, incluso aquellas operaciones que atentan contra la integridad del sistema. ¡Ten cuidado al momento de ejecutar comandos como `root`!

Creando usuarios

Comando `useradd`

Eliminando usuario

Comando `userdel`. Este comando también puede elimidar el directorio personal del usuario

Grupos

Un grupo es un conjunto de usuario. El propósito de los grupos, pues, agrupar muchos usuarios así como los permisos que ellos tienen.

Cada grupo se identifica con un `id`

Grupos

Cada usuario tiene un grupo principal (`gid`). El `gid` se hereda hacia los archivos que crea el usuario, cosa a tener en cuenta al momento de asignar archivos.

Creando grupos

Comando `groupadd`

Eliminando grupos

Comando `groupdel`

Agregando usuarios a grupos

Comando `usermod -aG nombre-grupo nombre-usuario`

Permisos

En general, un usuario puede realizar tres operaciones básicas con un archivo:

- Lectura (`r`)
- Escritura (`w`)
- Ejecución (`x`)

Permisos

Estos permisos de un archivo se asignan para :

- El usuario dueño del archivo
- Los usuarios del grupo dueño del archivo
- Para el resto de usuarios

Permisos

```
$ ls -l 00-presentacion-curso.md
-rw-r--r-- 1 paco users 1737 Sep  3 07:46 00-presentacion-c
```

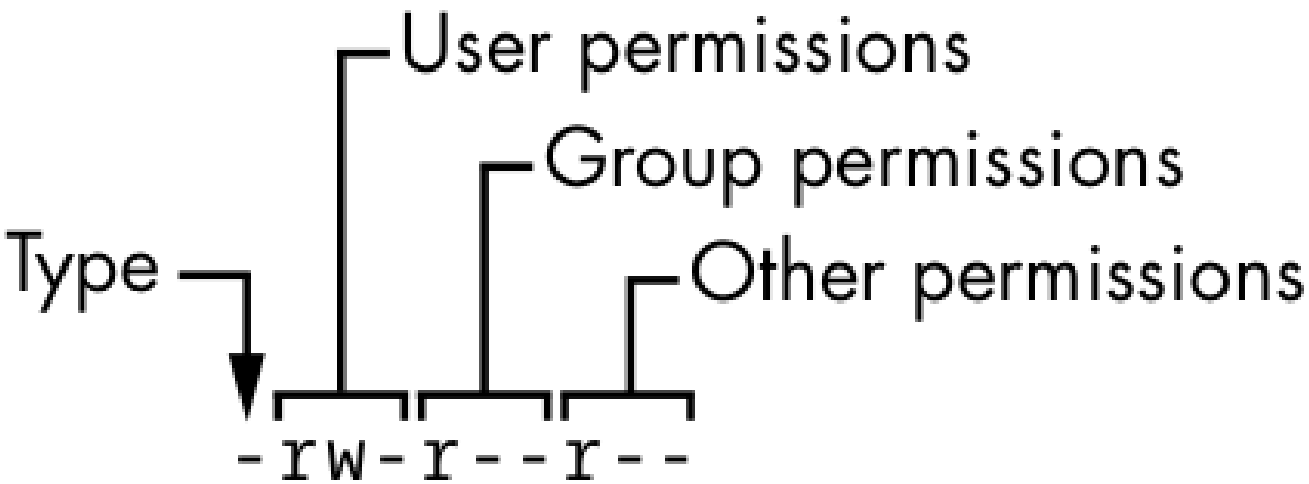

Figura 1: Viendo los permisos de un archivo

Formas de leer los permisos de un archivo

Representación	Tipo	Perm. dueño	Perm. grupo	Perm otros
Simbólica	-	`rw-`	`r--`	`r--`
		`110`	`100`	`100`
Octal		6	4	4

Formas de leer los permisos de un archivo

En su representación octal, los permisos del archivo se dividen en tres grupos de tres *bits* (octetos). La posición de cada *bit* hace referencia a cada uno de los permisos.

Cada una de estos octetos se convierte a decimal para describir los permisos con sólo tres números:

```
rw-r--r-- -> 110 100 100 -> 644
```

Haciendo la conversión rápidamente

- El permiso de escritura vale 4
- El permiso de lectura vale 2
- El permiso de ejecución vale 1

Actividad

Modificando los permisos de un archivo

El comando `chmod` permite cambiar los permisos de un archivo:

```
chmod u+x archivo # El usuario ahora puede ejecutar
chmod g+w archivo # El grupo ahora puede escribir
chmod o+r archivo # El resto ahora puede leer
chmod a+r archivo # Ahora todos puede leer
chmod o-r archivo # El resto ya no puede leer
```

También se puede utilizar la notación octal para definir por completo los permisos con un solo comando:

```
chmod 755 archivo # ¿Qué significa esto?
```

Cambiando el dueño de un archvo

Se puede cambiar tanto el usuario como grupo dueño de un archivo con el comando `chown`:

```
$ chown paco:estudiantes tarea.txt
```

La *shell*

Hansel Tepal, Francisco Galindo
Estudiantes de Ingeniería en Computación

Universidad Nacional Autónoma de México
Facultad de Ingeniería

Curso de SysAdmin, 2025-1

La *shell*

El programa encargado de interpretar qué programas debe ejecutar a partir de lo que escribas en la terminal se le llama *shell*.

Es sólo un programa como cualquier otro. Su objetivo es actuar como un intermediario entre el *kernel* del sistema operativo y la persona que utiliza la computadora.

La *shell* por defecto en Debian es `bash`, aunque existen muchas otras, como `zsh`, `csh`, `tcsh`, `dash`, etc.

Sintaxis de la ejecución de un comando en la *shell*

Un comando generalmente empieza indicando el nombre de un programa seguido de distintos *argumentos* que indican comportamiento. Los distintos argumentos se indican separados por espacios. Los argumentos que modifican el comportamiento por defecto de un programa suelen empezar con un guión (-), a estos se les suele conocer como banderas.

Un ejemplo de un comando es el siguiente:

```
$ cat /etc/passwd
```

Entrada y salida estándar

En otras palabras, cuando escribes datos a un programa a través de la terminal de un programa, estás escribiendo en su *stream* de **entrada estándar**. Cuando el programa escribe algo en la terminal, está utilizando su *stream* de **salida estándar**.

Entrada y salida estándar

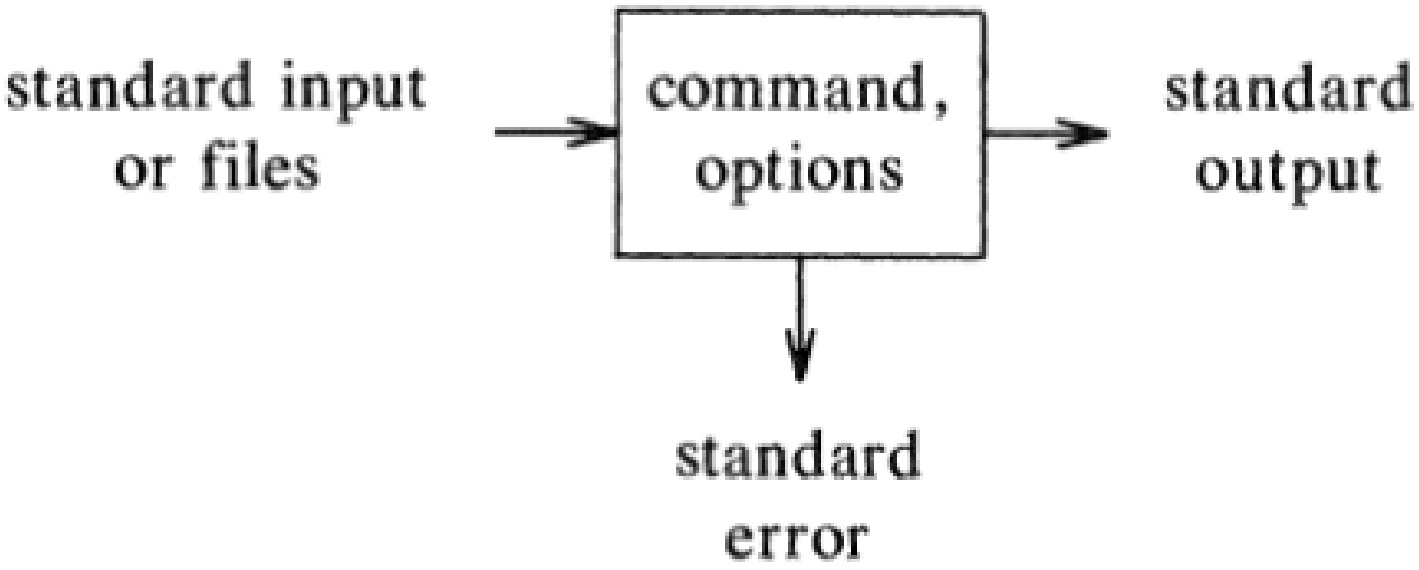

Figura 1: Entrada y salida estándar de un programa.

Redirección de salida

Si listamos los contenidos de la carpeta `home` de un usuario, es posible que veamos algo como esto:

```
$ ls -l
.  ..  .bash_logout  .bashrc  .config  .lesshst  .profile
```

Redirección de salida

Si se quiere es almacenar estos resultados a largo plazo, o para analizarlos después, puede hacerse que estos archivos se guarden en un archivo en lugar de mostrarse en la terminal:

```
$ ls -l >mi-home.txt
$ cat mi-home.txt
.  ..  .bash_logout  .bashrc  .config  .lesshst  .profile
```

Redirección de salida

El archivo al cual se hace referencia al redirigir la salida es creado en el caso de que no existiera anteriormente. Si el archivo ya existía, **los contenidos son sobreescritos por completo**. Así que cualquier cosa que estuviera almacenada en el archivo anteriormente se pierde.

Redirección de salida

Si no se desea sobreescribir los datos almacenados en el archivo de destino, puede utilizarse el símbolo >>, que opera de manera muy similar a >, con la diferencia de que >> agrega los nuevos datos al final de los ya existentes. Un ejemplo del uso de >> es el siguiente: en un principio concatenas varios archivos utilizando `cat`.

Redirección de entrada

Con < se puede redirigir la entrada. Por ejemplo, si se necesita ordenar la lista de alumnos de un curso, puede utilizarse el siguiente comando:

```
$ sort <alumnos.txt
Alberto
...
Yael
Zaír
```

Redirección de entrada

El resultado es el mismo que el que hubiéramos obtenido con el comando `sort alumnos.txt`. La diferencia aquí es que, cuando no se utiliza redirección, el programa `sort` se encarga de abrir y leer el archivo para posteriormente ordenarlo. Cuando uno redirige con <, es la *shell* la que se encarga de abrir y leer el archivo, así que desde la perspectiva de `sort`, hay una persona escribiendo los nombres uno a uno.

Redirección de entrada

Esta diferencia tiene la poderosa implicación de que ahora podemos utilizar archivos como datos de entrada para cualquier programa que lea desde la terminal, independientemente de que haya sido programado para ello.

Tuberías (pipes)

El concepto de redirección invita la posibilidad de ejecutar un comando, guardar su salida en un archivo, y luego utilizar ese archivo para alimentar a otro comando. Este proceso se vuelve tedioso muy rápido, por lo que se creó el concepto de *pipes*. Una *pipe* es una manera de conectar la salida estándar de un programa con la entrada estándar de otro. Se le llama *pipeline* a la conexión de dos o más programas por medio de *pipes*. Algunos ejemplos de pipes son:

```
$ ls | wc -l # Cuenta los archivos

# Muestra top 5 archivos pesados aquí
$ du -sh ./* | sort -rh | head -5
```

Tuberías (`pipes`)

Los programas dentro de una *pipeline* se ejecutan al mismo tiempo, de manera concurrente, no secuencialmente como parecería a primera vista. Ejecuta `cat | grep` para verificarlo.

Lo increíblemente poderoso de las tuberías es que permite la colaboración entre muchos programas de manera completamente transparente, sin necesidad de que estos estén programados explícitamente para interactuar entre sí. Los programas ni siquiera necesitan saber que están dentro de una *pipeline*. A partir de *pipelines* pueden generarse comportamientos extremadamente complejos con una serie de programas simples.

Gestión de Paquetes

Hansel Tepal, Francisco Galindo
Estudiantes de Ingeniería en Computación

Universidad Nacional Autónoma de México
Facultad de Ingeniería

Curso de SysAdmin, 2025-1

Información del tema

Tiempo estimado

Aproximadamente 1 hora de clase, repartido en secciones de explicación teórica y pequeños ejercicios para reforzar el aprendizaje.

Objetivos

- Aprender la utilidad de la existencia de usuarios, grupos y permisos en un sistema Linux
- Aprender a crear, modificar y eliminar usuarios y grupos del sistema.
- Se podrán interpretar y asignar diferentes permisos a los archivos del sistema de archivos.

Instalando Software en un sistema linux

En Linux (y en UNIX), el software se instalal de una manera distinta a otros sistemas operativos.

En lugar de utilizar una *AppStore* o descargar cosas desde el sitio web del desarrollador, se prefiere instalar programas mediante un *gestor de paquetes*.

Gestores de paquetes

Son herramientas que automatizan la instalación, actualización, configuración y eliminación de colecciones de software (programas, bibliotecas, . . .) que se conocen como *paquetes* en una computadora.

La manera principal de instalar software en una distribución Linux es mediante algún gestor de paquetes.

Gestores de paquetes

Un gestor de paquetes tiene una base de datos (repositorio) sobre muchos paquetes, así como las dependencias que tiene uno del otro para que todas las instalaciones sean correctas.

Gestores de paquetes

Algunos gestores de paquetes muy conocidos son:

- `apt`: El utilizado por Debian, Ubuntu y sus derivados (Mint, etc.). Los paquetes con los que trabaja tienen la extensión `.deb`
- `dnf`, `yum`: Utilizados en distribuciones basadas en *RHEL* (Fedora, CentOS, Rocky, etc.). Los paquetes están en formato `rpm`.
- `pacman`: Utilizado en Arch y sus derivados.

El gestor de paquetes es uno de los principales elementos que distinguen una distribución de otra.

Gestores de paquetes agnósticos

Para prevenir la fragmentación del "ecosistema" de Linux, existen algunos gestores de paquetes que intentan funcionar sim importar la distribución en la que se use:

- Flatpak
- Snap *
- AppImage