

ENTORNO DE PROGRAMACIÓN

TECNICATURA UNIVERSITARIA EN INTELIGENCIA ARTIFICIAL

9 de mayo de 2023.

ÍNDICE GENERAL

I	FUNDAMENTOS	5
1	HISTORIA	6
1.1	Orígenes de la computadora	6
1.2	Avances de la guerra	8
1.3	La informática moderna	10
1.4	Primeras computadoras personales	14
2	ARQUITECTURA DE LA COMPUTADORA	17
2.1	Gabinete	17
2.2	Placa madre	18
2.3	Fuente de alimentación	18
2.4	Microprocesador	19
2.5	Memoria RAM	20
2.6	Memoria secundaria	21
2.7	Placa de video	23
3	SISTEMAS OPERATIVOS	24
3.1	Proceso de arranque	24
3.2	Descripción	26
3.3	Núcleo	27
3.4	Terminal	28
3.5	Interfaz gráfica	30
3.6	Distribuciones	32
3.7	Proceso de apagado	33
4	CONCEPTOS DE PROGRAMACIÓN	36
4.1	¿Que es la programación?	36
4.2	Lenguaje de programación	37
4.3	Niveles de lenguajes	37
4.4	Compiladores e interpretes	39
4.5	Otros conceptos	39
5	SISTEMA DE ARCHIVOS	41
5.1	Acceso aleatorio y secuencial	42
5.2	Particiones	43
5.3	Descripción	44
5.4	Estructura de directorios	45

5.5	El sistema de archivos de Linux	48
5.5.1	Esquema de particionado	48
5.5.2	Nomenclatura de dispositivos	49
5.5.3	Jerarquía de archivos	50
5.5.4	Carpetas especiales	52
5.5.5	Montaje	52
5.5.6	Enlaces	53
 II MANEJO DE BASH		55
6	COMANDOS BÁSICOS	56
6.1	Introducción	56
6.1.1	man	57
6.1.2	clear	59
6.1.3	echo	60
6.1.4	history	61
6.1.5	Ejercicios	61
6.2	Sistema de archivos	62
6.2.1	pwd	62
6.2.2	cd	62
6.2.3	ls	63
6.2.4	mkdir	65
6.2.5	rmdir	66
6.2.6	touch	66
6.2.7	rm	67
6.2.8	cp	68
6.2.9	mv	70
6.2.10	df	70
6.2.11	du	71
6.2.12	ln	72
6.2.13	mount	72
6.2.14	umount	73
6.2.15	find	74
6.2.16	Comodines	76
6.2.17	Ejercicios	77
6.3	Contenido y filtros	79
6.3.1	nano	79
6.3.2	cat	80
6.3.3	less	81
6.3.4	head	82
6.3.5	tail	82
6.3.6	sort	83
6.3.7	uniq	84
6.3.8	strings	84
6.3.9	wc	85

6.3.10	file	86
6.3.11	cut	86
6.3.12	Expresiones regulares	87
6.3.13	tr	89
6.3.14	grep	90
6.3.15	Ejercicios	91
6.4	Secuenciación, redirección y tuberías	93
6.4.1	Secuenciación	93
6.4.2	Redirección	94
6.4.3	Tuberías	96
6.4.4	Ejercicios	96
6.5	Usuarios y grupos	97
6.5.1	whoami	97
6.5.2	id	97
6.5.3	who	98
6.5.4	su	98
6.5.5	sudo	99
6.5.6	passwd	100
6.5.7	Permisos en Linux	101
6.5.8	chown	103
6.5.9	chmod	104
6.5.10	useradd	105
6.5.11	userdel	106
6.5.12	Ejercicios	107
6.6	Procesos y tareas	107
6.6.1	ps	107
6.6.2	jobs	107
6.6.3	bg	107
6.6.4	fg	107
6.6.5	Señales	107
6.6.6	kill	107
6.6.7	Ejercicios	107
6.7	Gestión	107
6.7.1	apt	107
6.7.2	chsh	107
6.7.3	free	107
6.7.4	loadkeys	107
6.7.5	lsof	107
6.7.6	reboot	107
6.7.7	setxkbmap	107
6.7.8	startx	107
6.7.9	top	107
6.7.10	uname	107
6.7.11	uptime	107
6.7.12	which	107
6.7.13	Ejercicios	107

6.8	Internet	107
6.8.1	curl	107
6.8.2	ping	107
6.8.3	scp	107
6.8.4	ssh	107
6.8.5	wget	107
6.8.6	Ejercicios	107
6.9	Otros comandos	107
6.9.1	alias	107
6.9.2	bc	107
6.9.3	dd	107
6.9.4	rsync	107
6.9.5	sha256sum	107
6.9.6	tar	107
6.9.7	time	107
6.9.8	tldr	107
6.9.9	watch	107
6.9.10	Ejercicios	107
7	SHELL SCRIPTING	108
7.1	Comandos	108
7.1.1	exit	108
7.1.2	export	108
7.1.3	read	108
7.1.4	seq	108
7.1.5	shift	108
7.1.6	source	108
7.1.7	test	108
7.1.8	trap	108
7.2	Control de flujo	108
7.2.1	if / else	108
7.2.2	for	108
7.2.3	while / until	108
7.2.4	case	108
7.2.5	select	108
7.2.6	Funciones	108
III	CONCEPTOS ADICIONALES	109
8	CONTROL DE VERSIONES	110
9	CONTENEDORES	111
9.1	Emulación	111
9.2	Virtualización	111
9.3	Contenedores	111

Parte I

FUNDAMENTOS

«Una computadora es para mí la herramienta más sorprendente que hayamos ideado. Es el equivalente a una bicicleta para nuestras mentes».

Steve Jobs

HISTORIA

1.1 ORÍGENES DE LA COMPUTADORA

👤 *Blaise Pascal* había desarrollado en 1642 una de las primeras calculadoras mecánicas de la historia, conocida como la «📺 Pascalina». Esta máquina, capaz de realizar operaciones aritméticas básicas mediante el uso de engranajes y ruedas dentadas, fue un importante avance en el campo de la mecánica y la automatización, y sentó las bases para el desarrollo de las máquinas calculadoras modernas. La Pascalina fue utilizada por matemáticos y científicos de la época, y su diseño influyó en el desarrollo de posteriores calculadoras mecánicas.



Figura 1.1: Pascalina

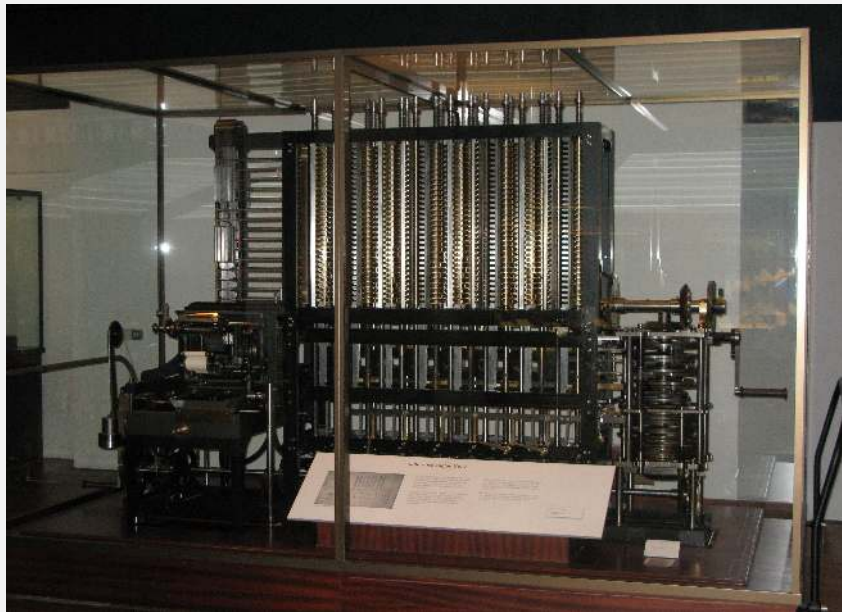


La «📺 máquina analítica» fue un proyecto desarrollado por 👤 *Charles Babbage* en 1837, considerado como el primer prototipo de una computadora moderna. Babbage, matemático, filósofo e inventor inglés, diseñó una máquina capaz de realizar cálculos automáticamente mediante el uso de engranajes



y ruedas dentadas, similar a la Pascalina de Blaise Pascal pero con un alcance mucho mayor. La máquina analítica tenía la capacidad de almacenar programas y realizar operaciones matemáticas complejas, y se considera el primer ejemplo de un sistema de procesamiento de datos automático. Aunque Babbage nunca logró construir la máquina completa debido a problemas financieros y técnicos, su proyecto sentó las bases para el desarrollo de las computadoras modernas y tuvo un gran impacto en el campo de la informática y la automatización.

Figura 1.2: Máquina diferencial



El 📺 teletipo fue una invención que permitió la transmisión automática de texto a través de una red telegráfica. Fue inventado a finales del siglo XIX, y su desarrollo permitió una comunicación más rápida y eficiente que el telégrafo de 📡 Samuel Morse. El teletipo utilizaba un sistema de impresión automática para recibir y transmitir texto, lo que permitió una comunicación más precisa y legible.

! Observación

Las terminales de teletipo (TTY) también se utilizaron como terminales de computadoras en las primeras décadas de la historia de las computadoras.

Figura 1.3: Teletipo Siemens t37h



1.2 AVANCES DE LA GUERRA

El comienzo de la Segunda Guerra Mundial en 1939 motivó un gran avance en la computación debido a la necesidad de procesar grandes cantidades de datos y realizar cálculos complejos para apoyar los esfuerzos de la guerra.



El ordenador  Z3 es una máquina de computación construida en 1941 por  Konrad Zuse, un ingeniero y matemático alemán. Era una máquina electromecánica basada en relés, que utilizaba un sistema binario para representar los datos y los cálculos. El Z3 contaba con una memoria programable, lo que permitía al usuario programar la máquina para llevar a cabo diferentes tareas. Fue destruido en 1943 durante un bombardeo en Berlín.



Figura 1.4: Réplica del Zuse Z3 exhibida en Múnich






La  Harvard Mark I es una computadora electromecánica construida en 1944 por IBM y patrocinada por el gobierno de los Estados Unidos. Fue diseñada por el matemático  *Howard Aiken* y se encuentra en el Museo de la historia de la computación en la Universidad de Harvard. La Harvard Mark I fue una máquina de gran tamaño, medía 8 metros de largo, 2 metros de alto y 2 metros de ancho, y utilizaba relés electromecánicos para llevar a cabo cálculos. La máquina era capaz de realizar aritmética básica y funcionaba con un sistema de tarjetas perforadas.



Figura 1.5: Harvard Mark I

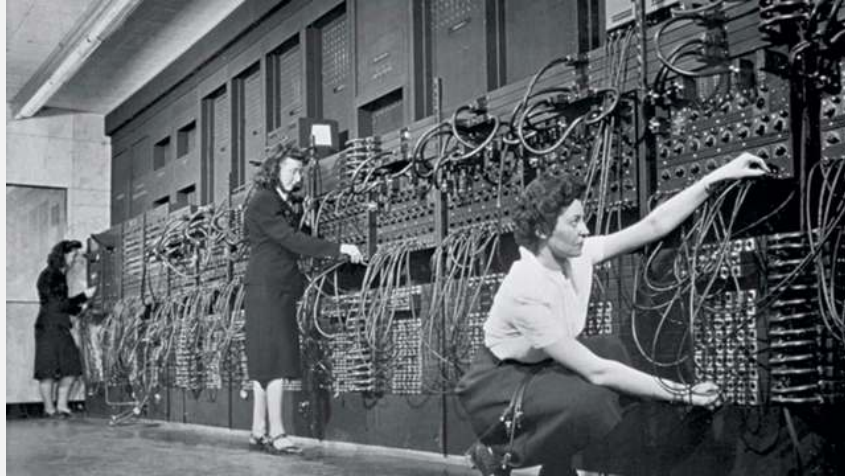


La  ENIAC (Electronic Numerical Integrator And Computer) fue una computadora electrónica programable construida por el gobierno de los Estados Unidos en 1946. Era una máquina enorme, medía 30 metros de largo, 2 metros de alto y 3 metros de ancho, y pesaba 27 toneladas. Utilizaba tecnología de válvulas electrónicas y era capaz de realizar cálculos numéricos complejos a alta velocidad. Fue utilizada para calcular los resultados de las explosiones nucleares y para resolver problemas científicos y técnicos en la industria. Fue inicialmente diseñada para calcular tablas de tiro de artillería destinadas al Laboratorio de Investigación Balística del Ejército de los Estados Unidos.

! Observación

Aunque la tecnología de válvulas era superior a la de relés, no fue hasta la invención del transistor en 1947 que las computadoras ganaron su verdadero poder de computo.

Figura 1.6: Electronic Numerical Integrator And Computer



1.3 LA INFORMÁTICA MODERNA

En la década de 1950, las computadoras eran enormes máquinas electro-mecánicas y electrónicas que ocupaban grandes espacios y requerían un equipo especializado para operarlas. Estas máquinas eran principalmente utilizadas para tareas de cálculo y procesamiento de datos, como la contabilidad y la investigación científica, y eran utilizadas principalmente por grandes empresas, organizaciones gubernamentales y universidades. A pesar de su tamaño y complejidad, estas computadoras marcaron el comienzo de la era informática moderna y sentaron las bases para el desarrollo de las computadoras personales y de escritorio de las décadas siguientes.

FORTRAN (Formula Translation) es uno de los primeros lenguajes de programación de computadora. Fue desarrollado en 1957 por un equipo de ingenieros de IBM liderado por 🧑 John Backus. El objetivo de FORTRAN era proporcionar un lenguaje de programación que permitiese a los científicos y matemáticos escribir programas de manera eficiente y fácilmente para ser utilizado en las computadoras de ese entonces.





📺 PDP-1 (Programmed Data Processor-1) es un ordenador construido por la compañía Digital Equipment Corporation (DEC) en 1959. Fue el primer ordenador de la serie PDP y uno de los primeros de tipo minicomputadora. El PDP-1 fue un ordenador de tiempo compartido, lo que significa que varios usuarios podían acceder al sistema al mismo tiempo y compartir los recursos del ordenador. Esto fue un gran avance en comparación con los ordenadores anteriores, que solían ser utilizados por un solo usuario a la vez.

Figura 1.7: Programmed Data Processor-1



BASIC (Beginners All-Purpose Symbolic Instruction Code) es un lenguaje de programación creado en el verano de 1964, con el objetivo de desarrollar un lenguaje de programación fácil de aprender y usar para estudiantes no especialistas y principiantes en la programación. El lenguaje se basó en el lenguaje FORTRAN, pero con un enfoque en la simplicidad y la facilidad de uso.

El sistema  NLS (oN-Line System) fue un sistema de información desarrollado por  Douglas Engelbart y su equipo en 1968. Fue una de las primeras demostraciones de una interfaz gráfica de usuario (GUI), con un puntero del ratón sistema de hipertexto y ventanas. También introdujo varias características que se consideran fundamentales en la computación moderna, como el procesamiento de textos, el correo electrónico, la videoconferencia y la colaboración en tiempo real.



MULTICS (Multiplexed Information and Computing Service) fue un sistema operativo desarrollado en 1969 por un equipo liderado por el MIT y Bell Labs. El objetivo de MULTICS era crear un sistema operativo de tiempo compartido que pudiera ser utilizado por varios usuarios simultáneamente y ofrecer servicios avanzados, como el procesamiento de archivos, el manejo de bases de datos y el procesamiento de informes. A pesar de sus avances, MULTICS tuvo problemas financieros y técnicos que retrasaron su desarrollo y limitaron su adopción. Aun así, muchas de las características y conceptos de MULTICS se convirtieron en estándar en la industria de los sistemas ope-

Figura 1.8: NLS (oN-Line System)



rativos modernos, como el manejo de permisos, el sistema de archivos y la seguridad.

Tras haber participado en el desarrollo de MULTICS, 🧑 Ken Thompson y 🧑 Dennis Ritchie iniciaron en 1970 la creación de un nuevo sistema operativo para la computadora DEC PDP-7. El proyecto fue bautizado originalmente como UNICS e inicialmente no tuvo apoyo económico por parte de los laboratorios Bell. Al año siguiente, Dennis Ritchie desarrolla el lenguaje de programación C, un sistema diseñado para ser eficiente en términos de tiempo de ejecución y uso de recursos, y fácil de portar a diferentes plataformas de hardware.



Los sistemas operativos existentes hasta el momento eran propietarios y solo funcionaban en una plataforma específica, es por esto que en 1972, Ken Thompson y Dennis Ritchie decidieron reescribir el código de UNICS pero esta vez en lenguaje C, dando así origen a UNIX. Este cambio significaba que UNIX podría ser fácilmente modificado para funcionar en otras computadoras y así otras variaciones podían ser desarrolladas por otros programadores. Ahora, el código era más conciso y compacto, lo que se tradujo en un aumento en la velocidad de desarrollo de UNIX.


En 1973 en el Xerox PARC (Palo Alto Research Center), se desarrolló el  Xerox Alto: un ordenador de tipo personal, con una interfaz gráfica de usuario, soporte para ventanas y un mouse. Además, Alto también tenía un sistema de gestión de archivos, un procesador de texto, una herramienta de dibujo y soporte para redes de computadoras. Aunque el Xerox Alto nunca fue comercializado, muchas de sus características y conceptos se convirtieron en estándar en la industria de los ordenadores personales.

Figura 1.9: Xerox Alto



1.4 PRIMERAS COMPUTADORAS PERSONALES


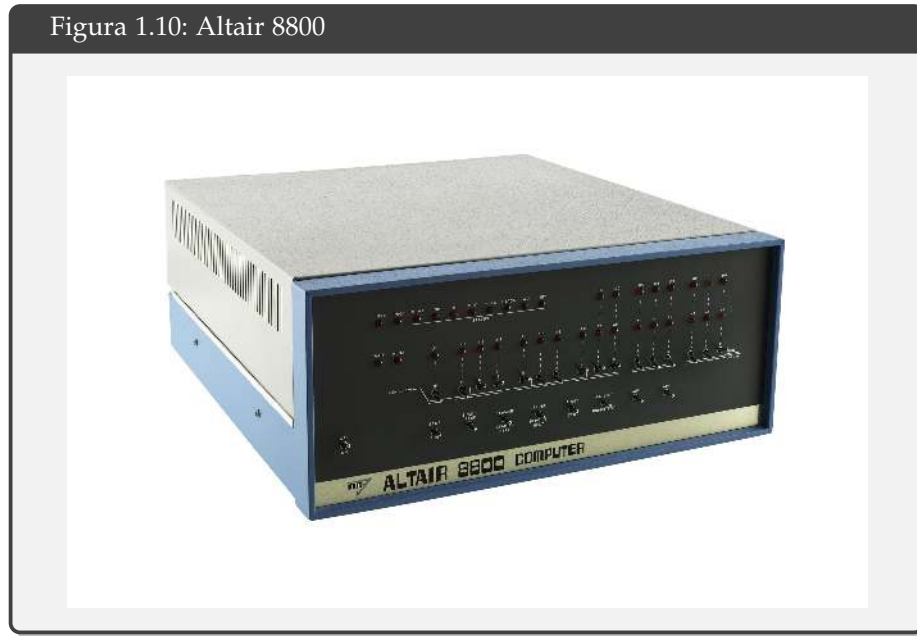
La  Altair 8800 fue una de las primeras computadoras personales en ser comercializadas, lanzada en 1974. Fue un gran éxito de ventas y sentó las bases para el desarrollo de las computadoras personales que conocemos hoy en día. La Altair 8800 se vendía en kit y los usuarios debían armarla ellos mismos.

Figura 1.10: Altair 8800



En los primeros años del sistema Unix los Laboratorios Bell autorizaron a las universidades, a utilizar el código fuente y adaptarlo a sus necesidades. A partir de dicha iniciativa, en 1977 nace en la universidad de Berkley el sistema operativo BSD (Berkeley Software Distribution). Sus principales contribuciones fueron la implementación de mejoras significativas en el sistema de archivos y en la red.

La IBM PC fue una de las computadoras más importantes en la historia de la informática, ya que sentó las bases para el estándar de computadora personal que se utiliza en la actualidad. Fue introducida por IBM en 1981 y se convirtió rápidamente en el estándar de la industria para las computadoras personales. Una de las características más importantes de la IBM PC fue su arquitectura abierta. A diferencia de otras computadoras personales de la época, la IBM PC tenía un diseño abierto que permitía a los usuarios y terceros desarrollar sus propios productos y programas para ella. Esto ayudó a impulsar un gran ecosistema de desarrolladores y fabricantes de periféricos que crearon una gran variedad de software y hardware para la computadora.

Figura 1.11: IBM PC



System V es una versión del sistema operativo UNIX desarrollado por Bell Labs en 1983. Fue una de las primeras versiones de UNIX en ser comercializada y distribuida ampliamente, y tuvo un gran impacto en el desarrollo de los sistemas operativos tipo UNIX. Aunque UNIX se convirtió en un estándar en la industria de la computación, no se consideraba «libre» debido a las restricciones en su uso y distribución impuestas por AT&T.


Ese mismo año fue fundado el movimiento GNU por 🧑 *Richard Stallman*, un programador y defensor de la libertad de software. El objetivo principal del movimiento GNU es desarrollar un sistema operativo completo y gratuito basado en el estándar UNIX, de manera que cualquier persona pueda usar, estudiar, compartir y modificar el software sin restricciones.




! Observación

El movimiento de software privativo había sido fundado con anterioridad en 1976 por Bill Gates a través de la denominada «**W** carta abierta a los aficionados».

X Windows es un sistema de ventanas para sistemas tipo UNIX. Fue desarrollado en el Massachusetts Institute of Technology (MIT) en 1984. El objetivo principal de X Windows era proporcionar un sistema de ventanas que pudiera ser utilizado en una variedad de sistemas, permitiendo una interfaz gráfica de usuario (GUI).

Minix fue un sistema operativo educativo desarrollado en el año 1987 por  *Andrew S. Tanenbaum*, para enseñar principios de diseño y funcionamiento de sistemas operativos a estudiantes universitarios. Originalmente diseñado para ser utilizado en computadoras IBM PC y compatibles, Minix tenía un diseño similar al de UNIX, pero con un conjunto reducido de herramientas y utilidades.



Mientras estudiaba informática en la Universidad de Helsinki en 1991,  *Linus Torvalds* desarrolla un núcleo de sistema operativo como proyecto personal, basándose en el diseño de Minix. Hasta el momento el proyecto GNU había desarrollado una amplia gama de software, incluyendo un compilador, un intérprete de línea de comandos y diversas herramientas de programación. Sin embargo, faltaba un kernel, que es la parte del sistema operativo que administra los recursos del sistema, como la memoria y los procesos.



En resumen, Linux surgió como un proyecto personal de Linus Torvalds, pero con el tiempo se convirtió en una parte fundamental del proyecto GNU y en uno de los sistemas operativos de código abierto más utilizados y respetados de la industria tecnológica.

ARQUITECTURA DE LA COMPUTADORA

2.1 GABINETE

El gabinete de la PC es una carcasa que cubre y protege los componentes de una computadora. Sus principales funciones son:

PROTECCIÓN El gabinete protege los componentes de la computadora de daños físicos, polvo y otros factores ambientales que pueden dañarlos.

ORGANIZACIÓN Los gabinetes de PC están diseñados para mantener todos los componentes en su lugar y organizados de manera eficiente. Esto hace que sea más fácil para el usuario trabajar en la computadora y realizar mejoras o reparaciones.

REFRIGERACIÓN: Los gabinetes de PC también ayudan a mantener los componentes frescos mediante la circulación de aire a través de la carcasa. Muchos gabinetes tienen ventiladores y otros sistemas de enfriamiento integrados para evitar el sobrecalentamiento de la computadora.

Figura 2.1: Gabinete Phobos Tg Xtech



2.2 PLACA MADRE

La placa madre es la pieza central de una computadora, encargada de conectar y comunicar todos los componentes esenciales del sistema. A través de sus conectores, la placa madre une la CPU, la memoria RAM, las unidades de almacenamiento, la tarjeta gráfica y otros dispositivos.

Además, la placa madre distribuye la energía eléctrica necesaria a todos los componentes a través de los conectores de alimentación, y controla los puertos de entrada/salida que permiten la comunicación de la computadora con dispositivos externos, como los puertos USB, de audio y de red.

La placa madre también incluye un chip de memoria ROM donde se almacena la BIOS, un programa que se encarga de configurar la computadora al encenderla y realizar pruebas iniciales del hardware.

La mayoría de las placas madre tienen un chip de audio integrado que proporciona capacidades de audio.

Figura 2.2: Placa madre



2.3 FUENTE DE ALIMENTACIÓN

La PSU (Power Supply Unit) o fuente de alimentación se encarga de convertir la corriente eléctrica de la toma de corriente en la energía eléctrica necesaria para alimentar los componentes internos de la computadora.

Esta recibe la corriente eléctrica de la toma de corriente a través del cable de alimentación y la convierte en diferentes voltajes que son suministrados a los componentes de la computadora.

La PSU también protege los componentes de la computadora de sobretensiones, cortocircuitos y otros problemas eléctricos que pueden ocurrir. En caso de que se detecte una sobrecarga o falla, la PSU puede cortar el suministro de energía para proteger los componentes de la computadora.

Figura 2.3: Fuente de Alimentación



2.4 MICROPROCESADOR

La CPU (Unidad Central de Procesamiento) es el componente principal de una computadora que realiza la mayoría de las operaciones de procesamiento de datos. Es un chip integrado que se coloca en el zócalo de la placa madre y está compuesto por varios núcleos (o cores) que trabajan en conjunto para ejecutar instrucciones y procesar datos.

La CPU es responsable de procesar y ejecutar los programas de software, manejar la entrada y salida de datos, y controlar los componentes del sistema, como la memoria RAM, el disco duro y las tarjetas de expansión. La velocidad y la capacidad de la CPU son factores clave que determinan el rendimiento

general de una computadora.

Algunos procesadores modernos tienen gráficos integrados en su diseño. Estos gráficos integrados se denominan iGPU (unidad de procesamiento de gráficos integrados) y están diseñados para proporcionar capacidades gráficas básicas para aplicaciones informáticas y de juegos de baja exigencia.

Figura 2.4: Intel Core i9



2.5 MEMORIA RAM

La memoria RAM (Random Access Memory o Memoria de Acceso Aleatorio) es un tipo de memoria que se utiliza en las computadoras para almacenar temporalmente los datos y programas que están en uso. La RAM es un componente clave en el rendimiento general de una computadora, ya que proporciona un acceso rápido y aleatorio a los datos y programas que el procesador necesita para operar.

Cuando una aplicación o un programa se ejecuta en la computadora, los datos y las instrucciones necesarios se cargan en la memoria RAM desde el disco duro. La RAM permite que el procesador acceda rápidamente a estos datos y programas, lo que acelera el tiempo de ejecución y la velocidad de la computadora en general.

Además, la memoria RAM es una memoria *volátil*, lo que significa que pierde todos los datos almacenados en ella cuando se apaga la computadora. Por lo tanto, es importante guardar los archivos y datos importantes en el disco duro o en otro dispositivo de almacenamiento persistente.


! Observación

Aunque el microprocesador también tiene una memoria volátil llamada *registros*, estos son mucho mas caros y en consecuencia pequeños.

Figura 2.5: Memoria RAM Corsair Vengeance DDR4



2.6 MEMORIA SECUNDARIA

Un  disco duro es un dispositivo de almacenamiento de datos magnético que se utiliza en las computadoras para almacenar permanentemente archivos y programas. A diferencia de la memoria RAM, que es una memoria volátil y pierde todos los datos almacenados en ella cuando la computadora se apaga, el disco duro mantiene los datos almacenados incluso después del apagado de la computadora.

Una de las principales diferencias entre el disco duro y la memoria RAM es la velocidad. La memoria RAM proporciona un acceso rápido y aleatorio a los datos y programas, lo que permite al procesador acceder a ellos rápidamente. En comparación, los discos duros son mucho más lentos en términos de velocidad de acceso, ya que el brazo de lectura/escritura necesita moverse físicamente para acceder a los datos en los platos.

Figura 2.6: Disco rígido Seagate Barracuda 1TB



Una SSD (Solid State Drive) es un dispositivo de almacenamiento de datos que utiliza memoria flash para almacenar permanentemente archivos y programas en la computadora. A diferencia de un disco duro tradicional, que utiliza platos magnéticos giratorios y cabezas de lectura/escritura para acceder a los datos, una SSD no tiene partes móviles y utiliza chips de memoria flash para almacenar y acceder a los datos.

La tecnología SSD es más rápida que la de un disco duro porque no hay partes mecánicas que necesiten moverse para acceder a los datos. En lugar de eso, los datos se almacenan en chips de memoria flash, que son mucho más rápidos para acceder y leer que los discos duros. Como resultado, las SSD proporcionan un mejor rendimiento en términos de velocidad de lectura/escritura y tiempo de acceso.

Figura 2.7: SSD Kingston



2.7 PLACA DE VIDEO

Una placa de video, también conocida como tarjeta gráfica, es un componente de hardware de la computadora que tiene como objetivo procesar y generar imágenes en la pantalla. Su función es liberar a la CPU (unidad central de procesamiento) de la computadora de la tarea de procesamiento gráfico, lo que permite que la CPU se concentre en otras tareas.

! Observación

Además de su uso en gráficos, en el campo de la inteligencia artificial son comúnmente utilizadas para entrenar y ejecutar redes neuronales profundas. Esto se debe a que las placas de video tienen una arquitectura altamente paralela que les permite procesar grandes cantidades de datos de manera eficiente. Como resultado, las placas de video son ideales para el procesamiento masivo de datos que se requiere en la inteligencia artificial.

Figura 2.8: Tarjeta Gráfica NVIDIA RTX 2080 Ti



SISTEMAS OPERATIVOS

3.1 PROCESO DE ARRANQUE

Durante el arranque de una PC, ocurren varias cosas importantes que permiten que el sistema operativo se inicie correctamente y la computadora esté lista para su uso.

Al presionarse el botón de arranque se activa la fuente de alimentación de la computadora, la cual suministra la energía necesaria para que la placa madre comience a funcionar. A partir de aquí, comienza un proceso que consiste en varias etapas:

POST (POWER ON SELF TEST) La placa madre realiza un autodiagnóstico para verificar que todos los componentes de hardware de la computadora estén funcionando correctamente. Si detecta algún problema, emitirá un mensaje de error y detendrá el proceso de arranque.

Figura 3.1: Etapa de Autodiagnóstico





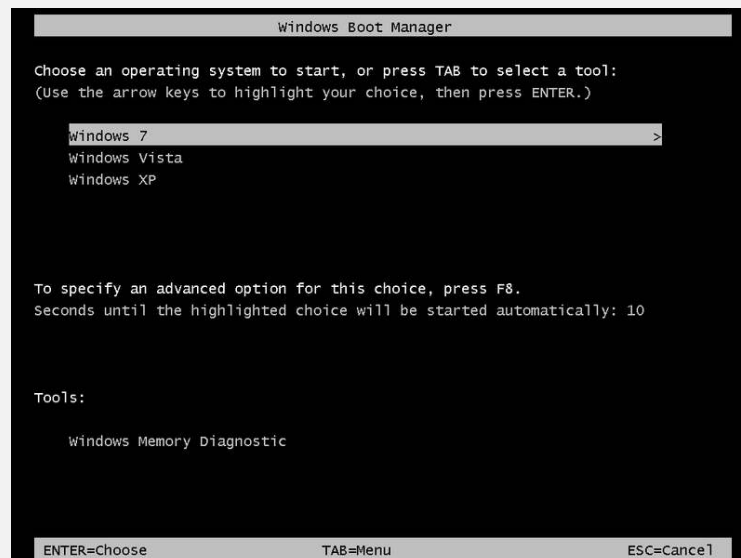
BOOT LOADER A continuación la placa madre debe cargar un programa llamado «*cargador de arranque*». El cargador de arranque es un programa cuyo objetivo principal es cargar el núcleo del sistema operativo. El cargador de arranque mas utilizado en Linux es « GRUB»; y « *bootmgr*» es el proporcionado por los sistemas modernos de Windows.

Figura 3.2:  GRUB



Figura 3.3:  bootmgr



NÚCLEO A continuación, el núcleo del sistema operativo toma el control de la computadora. Durante sus tareas iniciales se encargará de identificar el hardware disponible, cargar los controladores necesarios y montar el sistema de archivos del sistema. Finalmente dará comienzo al primer programa de usuario, a partir del cual se ejecutarán todos los demás programas.

INIT En los sistemas Linux, el programa inicial del sistema operativo se llama «*init*». *init* se encargará de cargar los scripts de arranque del sistema, así como también ejecutar los servicios esenciales para el funcionamiento del mismo, y proveer al usuario de un entorno gráfico o de línea de comandos.

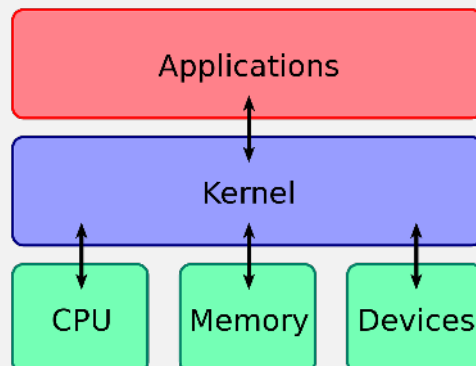
! Observación

En la actualidad se desarrolló «*systemd*» para reemplazar el sistema de inicio (*init*) heredado de los sistemas operativos estilo UNIX System V y Berkeley Software Distribution (BSD).

3.2 DESCRIPCIÓN

Un sistema operativo es un conjunto de programas y herramientas que controlan y coordinan las actividades de una computadora o dispositivo electrónico, y permiten a los usuarios interactuar con el hardware y el software de manera sencilla y eficiente. Está compuesto por un núcleo (o *kernel*) que tiene control completo sobre el hardware en el que corre, y una serie de programas utilitarios que se comunican con el.

Figura 3.4: Comunicación entre aplicaciones, núcleo y hardware



Las funciones principales de un sistema operativo incluyen:

- Gestionar el hardware: El sistema operativo es responsable de gestionar el hardware de la computadora, como el procesador, la memoria, el disco duro, la tarjeta gráfica, entre otros. Controla cómo se utilizan estos recursos y asigna la cantidad adecuada de memoria y procesador a cada aplicación.
- Proporcionar una interfaz de usuario: El sistema operativo proporciona una interfaz de usuario que permite a los usuarios interactuar con el ordenador y ejecutar aplicaciones y programas.
- Gestionar los archivos y directorios: El sistema operativo se encarga de gestionar los archivos y directorios del ordenador, lo que permite a los usuarios crear, modificar, copiar y eliminar archivos y carpetas.

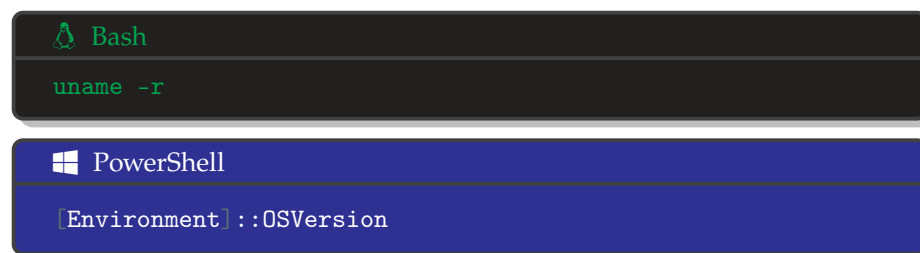
Entre las principales características de un sistema operativo se encuentran:

- Multitarea: Un sistema operativo permite que varias aplicaciones se ejecuten al mismo tiempo y asigna los recursos necesarios para que funcionen correctamente.
- Multiusuario: Un sistema operativo puede ser utilizado por varios usuarios al mismo tiempo y garantiza que cada usuario tenga sus propios archivos y configuraciones.
- Portabilidad: Los sistemas operativos pueden ser instalados en diferentes tipos de hardware, lo que los hace altamente portables.

3.3 NÚCLEO

El *kernel* (o núcleo) de un sistema operativo es la parte central y más fundamental del mismo. Es responsable de controlar el acceso a los recursos del hardware, gestionar los procesos, la memoria y la entrada/salida, y proporcionar una interfaz para que las aplicaciones interactúen con el hardware del sistema.

Podemos consultar cual es el núcleo que se esta ejecutando con el comando:



El núcleo se ejecuta en modo privilegiado, lo que significa que tiene acceso directo al hardware y puede ejecutar instrucciones que otros programas no pueden.

Para casi cualquier tarea las aplicaciones de usuario necesitan pedirle permiso al kernel, a través de una instrucción denominada «**W** llamada a sistema». Cuando se produce una llamada a sistema el CPU deja de ejecutar el programa, y comienza a ejecutar la funcionalidad del núcleo requerida, luego de la cual se continua con la ejecución del programa.

3.4 TERMINAL

A menudo se utilizan términos como «terminal», «consola virtual», «emulador de terminal» o «intérprete de línea de comandos» de forma indistinta, lo que puede llevar a cierta confusión. A continuación, se explican las diferencias entre estos términos:

TERMINAL Se refiere a el o los dispositivos físicos que se utilizan para interactuar con un ordenador mediante la entrada y salida de texto. En la actualidad está compuesta principalmente por el teclado y el monitor.



CONSOLA VIRTUAL Es una aplicación implementada dentro del núcleo que provee acceso al sistema simulando una terminal de teletipo. En los sistemas tipo Unix se puede acceder a ellas presionando  Ctrl+Alt+F1,  Ctrl+Alt+F2, etc.

Figura 3.5:  Consola virtual en Ubuntu

```
Ubuntu 18.04 ubuntu tty1
ubuntu login: Ubuntu
Password:
Welcome to Ubuntu 18.04 (GNU/Linux 4.15.0-23-generic)

 * Documentation:  https://help.ubuntu.com/

278 packages can be updated.
71 updates are security updates.

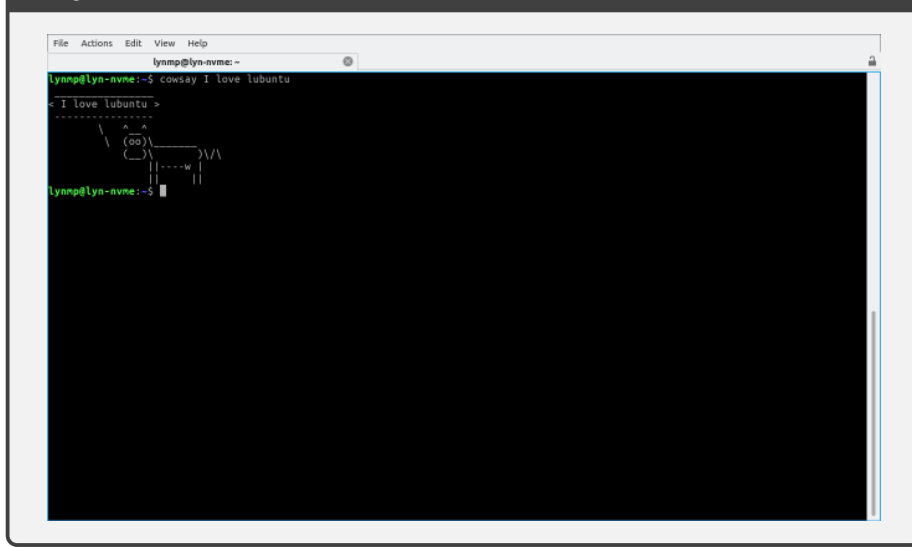
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

Ubuntu@ubuntu:~$
```

EMULADOR DE TERMINAL Es un programa de usuario que permite interactuar con un sistema operativo a través de una ventana en un entorno gráfico. Los emuladores de terminal son comúnmente utilizados para acceder a sistemas remotos o para ejecutar aplicaciones de línea de comandos en sistemas operativos.

Figura 3.6:  Qterminal en Lubuntu



Bash

Para saber que emulador de terminal se está utilizando se puede escribir el comando:

```
echo $TERM
```

SHELL También llamado «interprete de linea de comandos», es un programa que permite a un usuario interactuar con el sistema operativo mediante la ejecución de comandos a través de una interfaz de línea de comandos.

Bash

Para saber que interprete de linea de comandos se está utilizando se puede escribir el comando:

```
echo $SHELL
```

3.5 INTERFAZ GRÁFICA

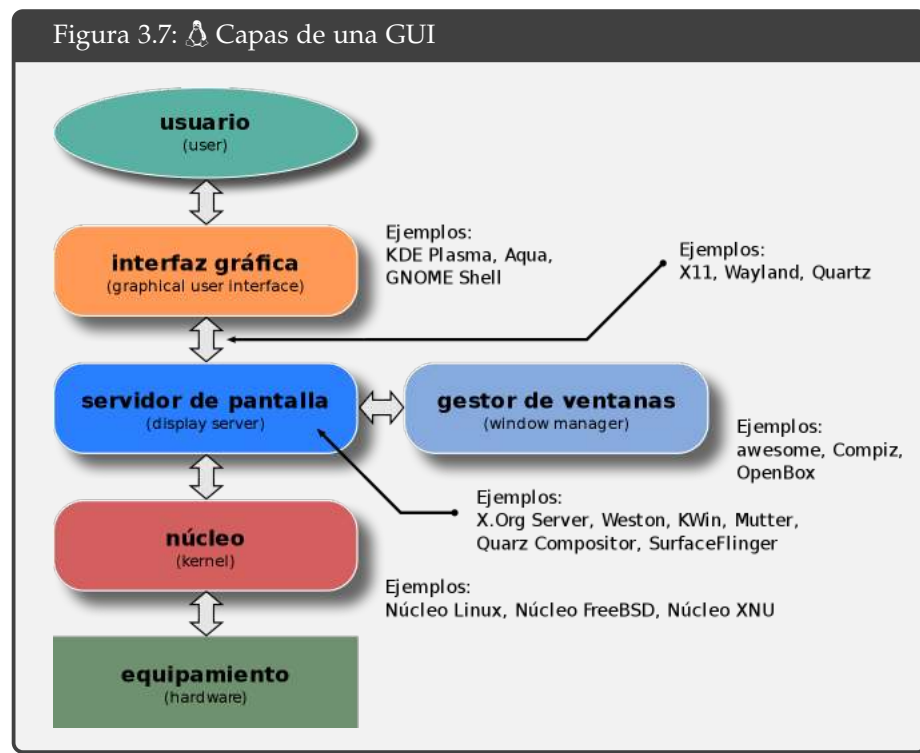
Una interfaz gráfica de usuario es una forma de interactuar con un programa o sistema operativo mediante el uso de elementos gráficos, como ventanas, iconos, botones y menús, en lugar de usar comandos de texto en una línea de comandos.

! Observación

A las interfaces de usuario gráficas las llamamos «GUI» por sus siglas en inglés «*Graphical User Interface*»; en cambio a las interfaces de texto las llamamos «CLI» por las siglas «*Command Line Interface*».

El entorno de escritorio, el sistema de ventanas, el servidor de pantalla y el gestor de ventanas son componentes importantes de un sistema operativo gráfico que trabajan juntos para proporcionar una interfaz de usuario intuitiva y fácil de usar.

Figura 3.7: Capas de una GUI



A continuación, se describen las funciones de cada uno de ellos:

ENTORNO DE ESCRITORIO Es un conjunto de aplicaciones, herramientas y utilidades que proporcionan una interfaz de usuario gráfica para un sistema operativo. El entorno de escritorio incluye menús, barras de herramientas, iconos, fondos de pantalla, gestores de archivos y otras herramientas que hacen que el uso del sistema operativo sea más fácil e intuitivo para el usuario. Algunos ejemplos de entornos de escritorio son 🐧 GNOME, 🐧 KDE, 🐧 XFCE y 🐧 LXDE.

Figura 3.8: 🐧 Unity en Ubuntu 22.10



SISTEMA DE VENTANAS Es un sistema que permite la creación y manipulación de ventanas de aplicaciones en la pantalla. El sistema de ventanas se encarga de administrar la posición, tamaño, apariencia y eventos de las ventanas en la pantalla. También se encarga de la gestión de los recursos gráficos, como el uso de la memoria, la gestión de la entrada y salida de datos, y el manejo de la interacción entre aplicaciones. Algunos ejemplos de sistemas de ventanas son X11 y 🐧 Wayland.

SERVIDOR DE PANTALLA Es un programa que se ejecuta en el sistema operativo y se encarga de controlar la pantalla, el teclado y el ratón del sistema. El servidor de pantalla recibe la entrada de teclado y ratón y la envía a las aplicaciones en ejecución en el sistema. También se encarga de mostrar la salida gráfica de las aplicaciones en la pantalla. El servidor de pantalla más utilizado en Linux es 🐧 Xorg.




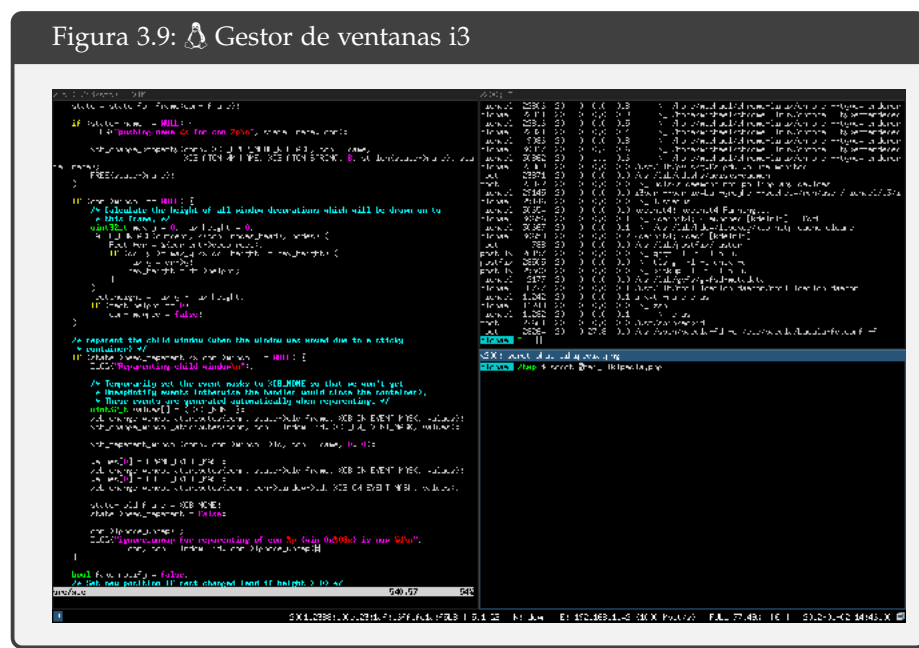




GESTOR DE VENTANAS Es un programa que se ejecuta en el entorno de escritorio y que se encarga de administrar la apariencia y el comportamiento de las ventanas de las aplicaciones. El gestor de ventanas proporciona una variedad de características, como la decoración de ventanas, la administración de escritorios virtuales, la configuración de atajos de teclado, y la gestión de la colocación de ventanas en la pantalla. Algunos ejemplos de gestores de ventanas son  Compiz,  Openbox y  i3.

Figura 3.9:  Gestor de ventanas i3



3.6 DISTRIBUCIONES

Las distribuciones de Linux son sistemas operativos basados en el kernel de Linux, que están compuestos por una combinación de software libre y de código abierto, como aplicaciones, controladores, herramientas de gestión de paquetes, etc.

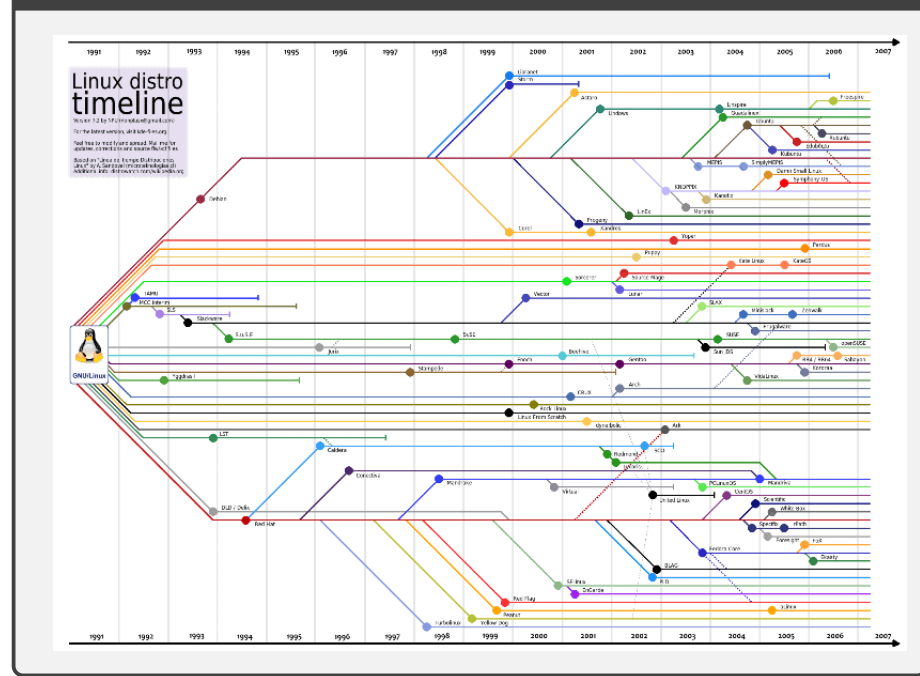
Existen muchas distribuciones de Linux diferentes, como  Debian,  Ubuntu,  Red Hat,  Arch Linux, entre otras. Cada distribución tiene sus propias características, objetivos y filosofía, y están diseñadas para satisfacer las necesidades de diferentes usuarios y aplicaciones.

! Observación

El concepto de distribución de Linux se hereda de los sistemas operativos estilo UNIX donde las universidades comenzaron a desarrollar sus propias versiones, como System V y BSD.

Las distribuciones de Linux existen porque el software de código abierto permite a los usuarios y desarrolladores acceder, modificar y distribuir el código fuente del software. Esto ha permitido que muchas personas y comunidades puedan desarrollar y distribuir sus propias versiones personalizadas de Linux. Además, al ser un sistema operativo altamente personalizable y adaptable, cada distribución puede estar diseñada para satisfacer las necesidades específicas de diferentes usuarios, como por ejemplo para usuarios de servidores, programadores, usuarios de escritorio, entre otros.

Figura 3.10:  Línea de tiempo de distribuciones de Linux



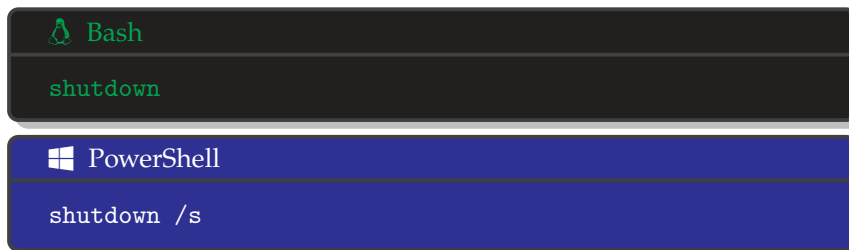
3.7 PROCESO DE APAGADO

Antes de finalizar la ejecución del sistema operativo, se inicia un proceso que cierra todos los programas y servicios que se están ejecutando en la computadora. Luego, se guardan todos los datos pendientes y se asegura que

todos los dispositivos de almacenamiento, como los discos duros o las unidades flash USB, estén en un estado seguro antes de apagar la alimentación. Finalmente, se envía una señal al hardware para que se apague por completo y se desconecte la alimentación en caso de ser necesario.

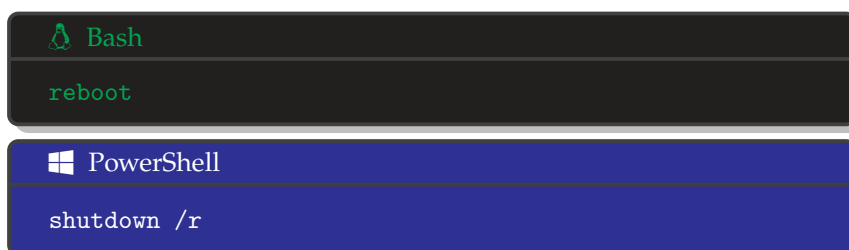
Hay varias opciones disponibles para «apagar» una computadora, cada una con diferentes efectos.

APAGADO Cuando se selecciona la opción de «apagar», la computadora cierra todos los programas y procesos en ejecución y se apaga completamente. La próxima vez que se encienda la computadora, se iniciará el proceso de arranque completo.



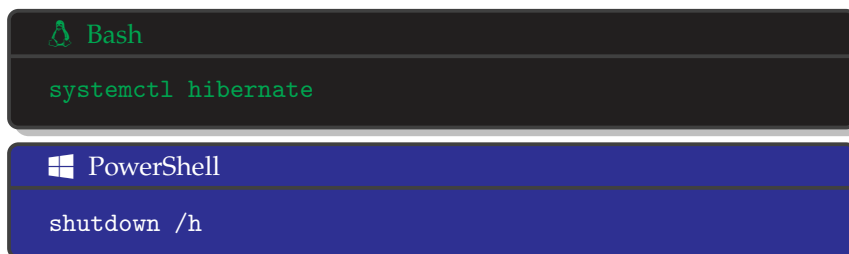
Two terminal window screenshots. The first is a Bash terminal with a dark background, showing the command `shutdown` in green text. The second is a PowerShell terminal with a blue background, showing the command `shutdown /s` in white text.

REINICIO Al seleccionar la opción «reiniciar», la computadora cierra todos los programas y procesos en ejecución, se apaga brevemente y luego se reinicia automáticamente. Esta opción es útil para solucionar problemas de hardware o software y para actualizar el sistema operativo.




Two terminal window screenshots. The first is a Bash terminal with a dark background, showing the command `reboot` in green text. The second is a PowerShell terminal with a blue background, showing the command `shutdown /r` in white text.

HIBERNACIÓN La opción de «hibernar» guarda todos los datos y configuraciones del sistema en el disco duro y luego apaga la computadora. Cuando se vuelve a encender la computadora, el sistema restaura automáticamente los datos y la configuración de la sesión anterior.



Two terminal window screenshots. The first is a Bash terminal with a dark background, showing the command `systemctl hibernate` in green text. The second is a PowerShell terminal with a blue background, showing the command `shutdown /h` in white text.

SUSPENSIÓN La opción «suspender» pone la computadora en un estado de bajo consumo de energía, dejando alimentada solamente la memoria RAM. De esta manera los programas y procesos en ejecución se conservan y la computadora puede volver a su estado anterior cuando se reanude la actividad. Esta opción es útil para ahorrar energía y reanudar rápidamente el trabajo en curso.

 Bash

```
systemctl suspend
```

! Observación

Si se llega a producir un corte en el suministro eléctrico, el estado de la computadora se pierde pues la RAM será incapaz de retener su información.

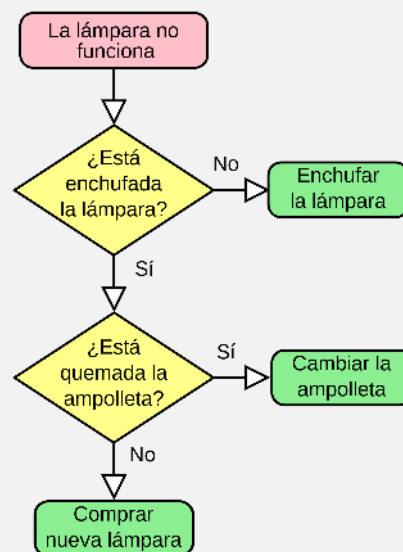
CONCEPTOS DE PROGRAMACIÓN

4.1 ¿QUE ES LA PROGRAMACIÓN?

La programación es el proceso de diseñar, escribir, probar y mantener el código informático. En términos más simples, la programación es la forma en que los desarrolladores crean software, aplicaciones y sistemas.

El proceso de programación implica varios pasos, que incluyen la definición del problema que se está tratando de resolver, la identificación de los requisitos y especificaciones necesarios para crear la solución, la escritura del código en un lenguaje de programación, la realización de pruebas para asegurarse de que el software funciona correctamente y la documentación del código para que otros desarrolladores puedan entender y trabajar en él.

Figura 4.1: Diagrama del flujo de un programa



4.2 LENGUAJE DE PROGRAMACIÓN

Un lenguaje de programación es un conjunto de reglas, símbolos y palabras clave que se utilizan para comunicar instrucciones a una computadora. Estos lenguajes permiten a los programadores crear software, aplicaciones y sistemas, al proporcionar un medio para escribir código en un formato que la máquina pueda entender y ejecutar.

Los lenguajes de programación se utilizan para describir las acciones que una computadora debe realizar, como realizar cálculos matemáticos, interactuar con dispositivos periféricos, almacenar y recuperar datos, y responder a las entradas del usuario. Los lenguajes de programación pueden variar en complejidad y enfoque, y algunos se centran en tareas específicas, mientras que otros son más generales y versátiles.

En esencia, un lenguaje de programación es una herramienta para crear software y sistemas informáticos, permitiendo a los programadores transformar sus ideas en código ejecutable.

! Observación

En esta materia aprenderemos el lenguaje de programación de «W bash». A lo largo de la carrera también aprenderán el lenguaje «W python».

4.3 NIVELES DE LENGUAJES

Los lenguajes de programación se pueden clasificar en tres niveles: alto, bajo y binario.

ALTO Estos lenguajes son más cercanos al lenguaje natural humano y se utilizan para escribir programas complejos de manera más fácil y rápida. Ejemplos de lenguajes de programación de alto nivel incluyen Python, Java, C++, Ruby, PHP, entre otros. Estos lenguajes tienen una sintaxis más amigable para el programador y se encargan de muchos de los detalles de bajo nivel, como la administración de memoria y el manejo de errores.

</> Código

```
print("Hola mundo!")
```

BAJO Estos lenguajes están más cerca del lenguaje de la máquina y se utilizan para escribir programas que interactúan directamente con el hardware del ordenador. Ejemplos de lenguajes de programación de bajo nivel

incluyen el lenguaje ensamblador y el lenguaje C. Estos lenguajes requieren que el programador tenga un conocimiento más detallado del hardware y de la forma en que se maneja la memoria.

</> Código

```
.global _start          # gcc -nostdlib -no-pie hola.s
.text

mensaje: .ascii "Hola mundo!\n"
_start:
    mov $1, %rax        # La llamada a sistema 1 es write
    mov $1, %rdi        # El descriptor 1 es la salida estandar
    mov $mensaje, %rsi  # Direccion de memoria del mensaje
    mov $12, %rdx       # Cantidad de bytes
    syscall             # write(1, mensaje, 12)


    mov $60, %rax       # La llamada a sistema 60 es exit
    mov $0, %rdi        # Queremos devolver el numero 0
    syscall             # exit(0)
```

BINARIO El lenguaje binario es el lenguaje de máquina utilizado por los ordenadores para ejecutar programas. Este lenguaje está compuesto de ceros y unos, que representan instrucciones que el procesador del ordenador puede entender y ejecutar directamente. Es muy difícil y tedioso para los programadores escribir directamente en lenguaje binario, por lo que se utilizan los lenguajes de programación de nivel alto y bajo para crear programas que luego se traducen a lenguaje binario.

</> Código

```
401000: 48 6f 6c 6c          # H o l a
401004: 20 6d 75 6e 64 6f   # m u n d o
40100a: 21 0a              # ! \n
40100c: 48 c7 c0 01 00 00 00 # mov $1, %rax
401013: 48 c7 c7 01 00 00 00 # mov $1, %rdi
40101a: 48 c7 c6 00 10 40 00 # mov $mensaje, %rsi
401021: 48 c7 c2 0c 00 00 00 # mov $12, %rdx
401028: 0f 05             # syscall
40102a: 48 c7 c0 3c 00 00 00 # mov $60, %rax
401031: 48 c7 c7 00 00 00 00 # mov $0, %rdi
401038: 0f 05             # syscall
```

4.4 COMPILADORES E INTERPRETES

Un  intérprete y un compilador son dos tipos de programas que se utilizan para convertir el código fuente escrito por un programador en instrucciones ejecutables por una computadora.

INTERPRETE Un intérprete es un programa que lee el código fuente de un programa y lo traduce en instrucciones ejecutables en tiempo real. El intérprete lee una línea de código fuente, la traduce a lenguaje de máquina y la ejecuta antes de pasar a la siguiente línea. Debido a que el intérprete realiza la traducción y la ejecución de cada línea a medida que se lee el código, puede ser más lento que un compilador. Sin embargo, el intérprete tiene la ventaja de que el programador puede ejecutar el código directamente sin necesidad de compilarlo previamente.

COMPILADOR Un compilador, por otro lado, es un programa que traduce todo el código fuente a lenguaje de máquina en un solo paso antes de su ejecución. El compilador lee todo el código fuente del programa y lo traduce en un archivo ejecutable que puede ser utilizado por la computadora sin necesidad de leer el código fuente original nuevamente. Debido a que el código fuente se traduce y compila solo una vez, el código compilado se ejecuta generalmente más rápido que el código interpretado. Sin embargo, el proceso de compilación puede llevar más tiempo que el proceso de interpretación.

4.5 OTROS CONCEPTOS

CÓDIGO FUENTE El código fuente de un programa es el conjunto de instrucciones o comandos escritos por un programador en un lenguaje de programación determinado; es el «texto» legible por humanos que se escribe en un archivo de código fuente y que debe ser traducido o compilado en lenguaje de máquina para que la computadora pueda entenderlo y ejecutar el programa.

CÓDIGO ABIERTO El término «código abierto» se refiere a un tipo de software en el cual el código fuente está disponible públicamente y puede ser modificado por cualquier persona. En el modelo de código abierto, los desarrolladores de software pueden trabajar juntos para mejorar y expandir el software existente.

SOFTWARE LIBRE El software libre es aquel que se distribuye con una licencia que permite a los usuarios tener acceso al código fuente del programa y redistribuir versiones modificadas del mismo sin tener que pagar regalías o cargos adicionales. En general, dicha licencia establece los términos y condiciones bajo los cuales se puede usar, modificar y distribuir el software.



! Observación

El software libre suele estar disponible gratuitamente, sin embargo no es obligatorio que sea así, por lo tanto no hay que asociar software «libre» a «gratuito» ya que, conservando su carácter de libre, puede ser distribuido comercialmente.

IDE Un IDE (Integrated Development Environment) es un conjunto de herramientas y características que se combinan en una única aplicación para hacer más fácil y eficiente el proceso de desarrollo de software.

Típicamente incluye un editor de código que proporciona herramientas de resaltado de sintaxis y autocompletado, lo que facilita la escritura de código. Además, un IDE suele incluir un depurador que ayuda a identificar y corregir errores en el código. También puede tener herramientas de compilación y construcción, que permiten compilar y empaquetar el código en un archivo ejecutable o en otro formato adecuado para su distribución.

Figura 4.2: Visual Studio Code

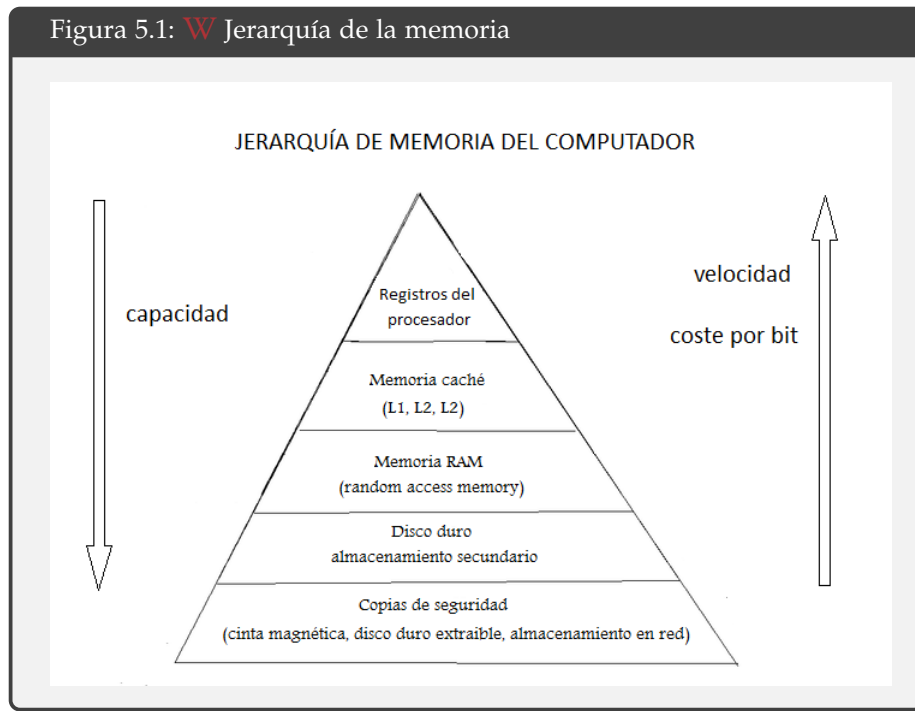


SISTEMA DE ARCHIVOS

Aunque la «**W** memoria principal» es esencial para el funcionamiento de una computadora, esta es de acceso volátil, lo que significa que se borra cuando se apaga la computadora. Por lo tanto, la memoria RAM solo <puede almacenar temporalmente los programas y datos que están en uso en ese momento.

En este capítulo explicaremos como funcionan las llamadas «**W** memorias secundarias» de la computadora, como el disco rígido o una unidad de estado solido.

Figura 5.1: **W** Jerarquía de la memoria



5.1 ACCESO ALEATORIO Y SECUENCIAL

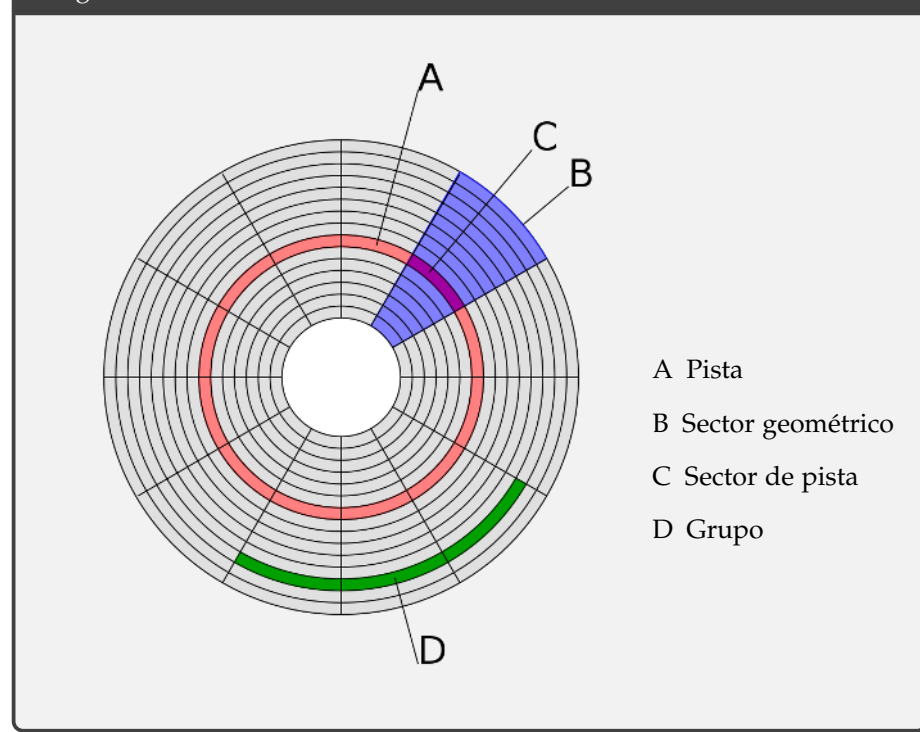
Un dispositivo de acceso aleatorio, como una unidad de estado sólido o USB, permite el acceso directo a cualquier ubicación en el medio de almacenamiento. Esto significa que se puede acceder a cualquier archivo o dato almacenado en el dispositivo sin tener que pasar por los datos que están almacenados «antes» o «después» en el dispositivo.

! Observación

Puesto que los dispositivos de acceso aleatorio son muy rápidos, son ideales para almacenar programas y mejorar el rendimiento general de la computadora.

Por otro lado, un dispositivo de acceso secuencial, como un disco rígido o DVD, requiere que los datos se lean en secuencia, desde el principio del medio de almacenamiento hasta el final. Esto significa que para acceder a un archivo o dato específico, el dispositivo debe leer los datos almacenados antes de llegar al archivo o dato deseado.

Figura 5.2: Estructura de un disco



! Observación

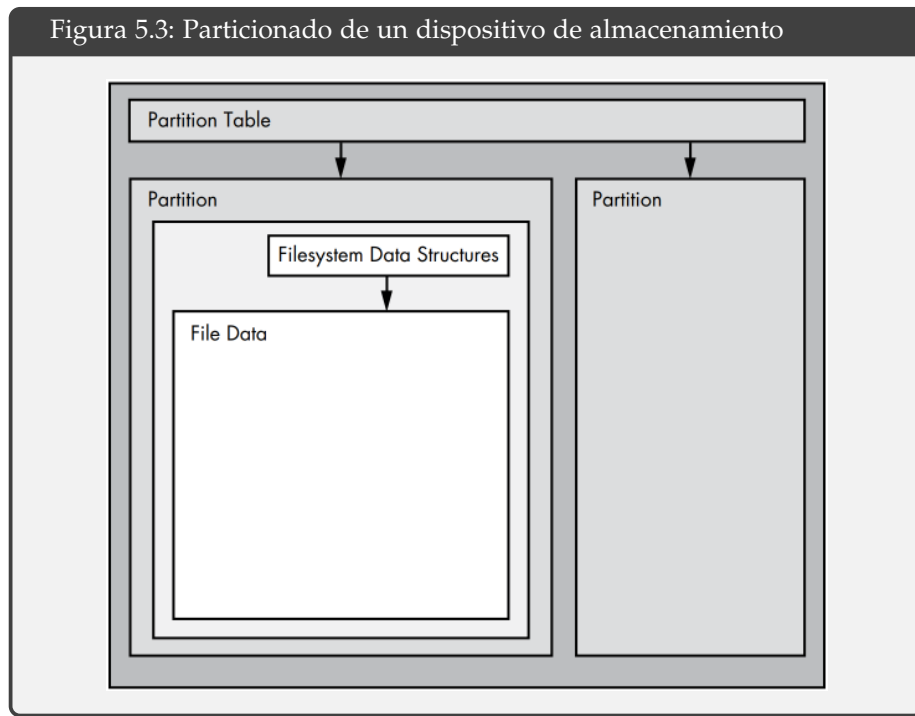
Las arquitecturas de computadora modernas suelen utilizar además de una memoria de acceso directo, otra de acceso secuencial. Esto es puesto que pese a son mas lentas, son mas baratas y de mayor capacidad, lo que los hace ideales como medios de respaldo.

5.2 PARTICIONES

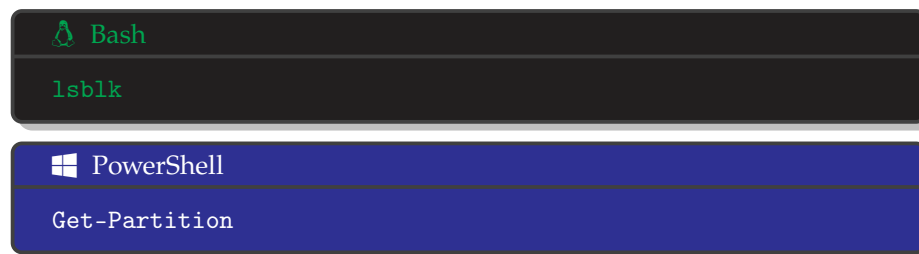
Una partición es una sección lógica de un disco duro u otro dispositivo de almacenamiento que se trata como si fuera un disco separado. Una partición se crea mediante la división del espacio de almacenamiento disponible en el dispositivo en secciones separadas. Cada partición tiene su propio sistema de archivos y puede contener archivos y datos independientes de las demás particiones.

Las particiones se utilizan para varios fines, como separar el sistema operativo y los archivos del usuario, o para crear diferentes áreas de almacenamiento para diferentes propósitos, como la música, los documentos y las imágenes. También se pueden utilizar para instalar varios sistemas operativos en una sola unidad de disco duro.

Figura 5.3: Particionado de un dispositivo de almacenamiento



Para observar los discos y particiones de un sistema podemos usar el comando:



5.3 DESCRIPCIÓN

Un sistema de archivos es una estructura lógica y organizada de datos que se utiliza para almacenar y recuperar información en un medio de almacenamiento. Sin un sistema de archivos, los datos colocados en un medio de almacenamiento serían un gran cuerpo de datos sin manera de saber dónde termina un dato y comienza el siguiente.

Sus principales funciones son:

GESTIÓN DEL ESPACIO El sistema de archivos se encarga de gestionar la manera en que los archivos son almacenados en un dispositivo de almacenamiento de datos. Esto incluye entre otras cosas, la asignación de espacio en disco para cada archivo, la administración del espacio libre y la organización de los datos de manera eficiente.

ORGANIZACIÓN El sistema de archivos organiza los archivos en una jerarquía de directorios y subdirectorios, lo que facilita su búsqueda y recuperación.

PERMISOS El sistema de archivos puede incluir medidas de seguridad para proteger los datos almacenados en el medio de almacenamiento, como permisos de acceso, y encriptación de datos.

METADATOS Un sistema de archivos almacena una variedad de metadatos que proporcionan información sobre los archivos y directorios que se almacenan en el dispositivo de almacenamiento. Algunos de los metadatos más comunes que se almacenan en un sistema de archivos son el nombre del archivo, su tamaño, fecha de creación y modificación, etc.

INTEGRIDAD Un sistema de «*W journaling*» (o diario) es una técnica utilizada para proteger la integridad de los datos. Este sistema funciona registrando todos los cambios que se realizan en el sistema de archivos en un registro o diario antes de que los cambios sean efectuados en el sistema de archivos. En caso de un fallo del sistema o un corte de energía, el

sistema de journaling puede utilizar la información en el registro para volver al estado anterior del sistema de archivos, evitando así la pérdida de datos o la corrupción del sistema de archivos.

! Observación

En los sistemas Linux, el sistema de archivos mas utilizados es «**W** *ext4*», mientras que en los sistemas Windows se utiliza «**W** *NTFS*».

5.4 ESTRUCTURA DE DIRECTORIOS

Los sistemas operativos utilizan archivos y directorios para organizar y almacenar información en un dispositivo de almacenamiento. Los archivos contienen los datos que los usuarios crean y manipulan, mientras que los directorios proporcionan una forma de organizar y acceder a los archivos de manera lógica y fácil de entender.

! Observación

Los términos carpeta y directorio tienen el mismo significado.

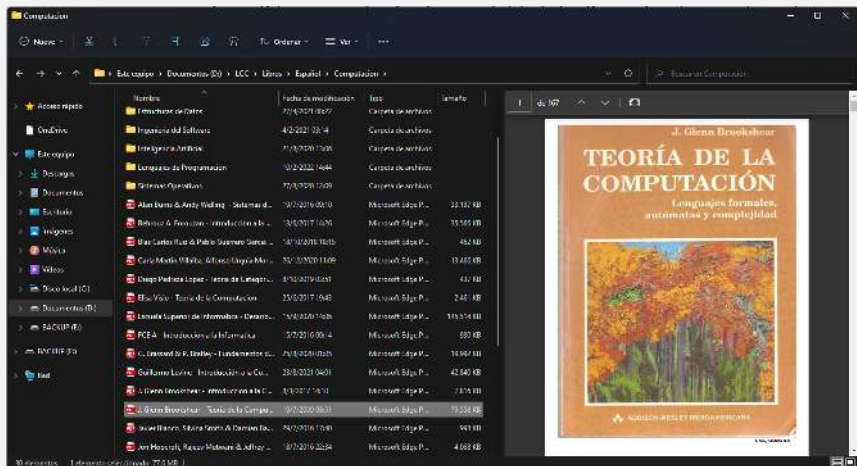
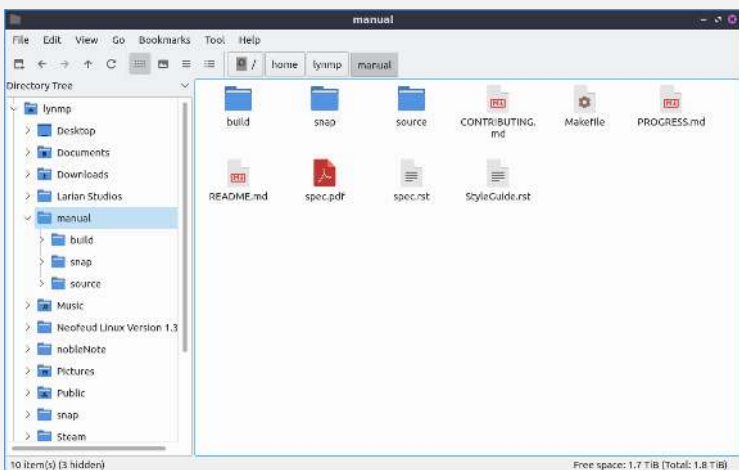
La jerarquía de directorios en un sistema de archivos se llama árbol de directorios. El nivel superior del árbol de directorios se llama directorio raíz, que contiene todos los demás directorios y archivos en el sistema de archivos. A medida que se desciende por el árbol de directorios, los nombres de los directorios y los archivos se combinan para formar rutas de acceso a los archivos y directorios individuales.

! Observación

En los sistemas Linux existe un solo directorio raíz llamado «/», en donde luego se «montan» el resto de los dispositivos. En los sistemas Windows cada partición tiene su propio directorio raíz llamado «\».

Para navegar a través de los directorios del sistema en un entorno gráfico, utilizamos un programa denominado «administrador de archivos». En dicho programa podemos recorrer la estructura del sistema de archivos mediante una serie de clicks de mouse en iconos, botones y menús.

En esta sección aprenderemos a llevar a cabo esas mismas acciones, pero desde una ventana de terminal.



En la barra de dirección del administrador de archivos, podemos observar en que posición de la estructura de directorios nos encontramos en este momento. Esa misma información podemos obtenerla en una terminal con el siguiente comando:

 Bash`pwd` PowerShell`pwd`

! Observación

El programa «pwd» recibe su nombre de las siglas en ingles «*Print Working Directory*» (mostrar directorio de trabajo).

El administrador de archivos nos muestra también en su ventana el contenido del directorio de trabajo actual. Para lograr lo mismo desde la consola escribimos:

 Bash`ls` PowerShell`ls`

Para cambiar la carpeta en la que estamos desde el entorno gráfico simplemente hacemos doble click en la carpeta que pretendemos. Para cambiar de carpeta desde una terminal disponemos del siguiente comando:

 Bash`cd carpeta` PowerShell`cd carpeta`

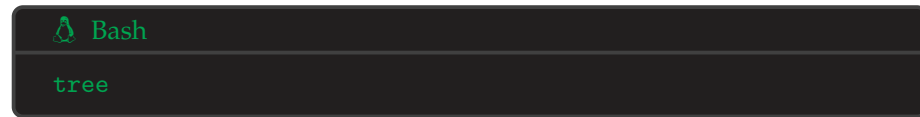
En el entorno gráfico para acceder al directorio padre del directorio actual hacemos click en la flecha hacia arriba. El comando para lograr lo mismo es:

 Bash`cd ..`



```
PowerShell
cd ..
```

Si queremos observar el árbol de directorios desde la posición actual, podemos escribir lo siguiente en la consola:



```
Bash
tree
```



```
PowerShell
tree
```

Existen dos tipos de rutas que se utilizan para referirse a la ubicación de un archivo o directorio en un sistema de archivos: rutas absolutas y rutas relativas.

RUTA ABSOLUTA Una ruta absoluta es una ruta que especifica la ubicación completa de un archivo o directorio, desde el directorio raíz hasta el archivo o directorio en cuestión. En otras palabras, una ruta absoluta es la ruta completa que describe la ubicación exacta de un archivo o directorio, independientemente de la ubicación del usuario o del directorio de trabajo actual.

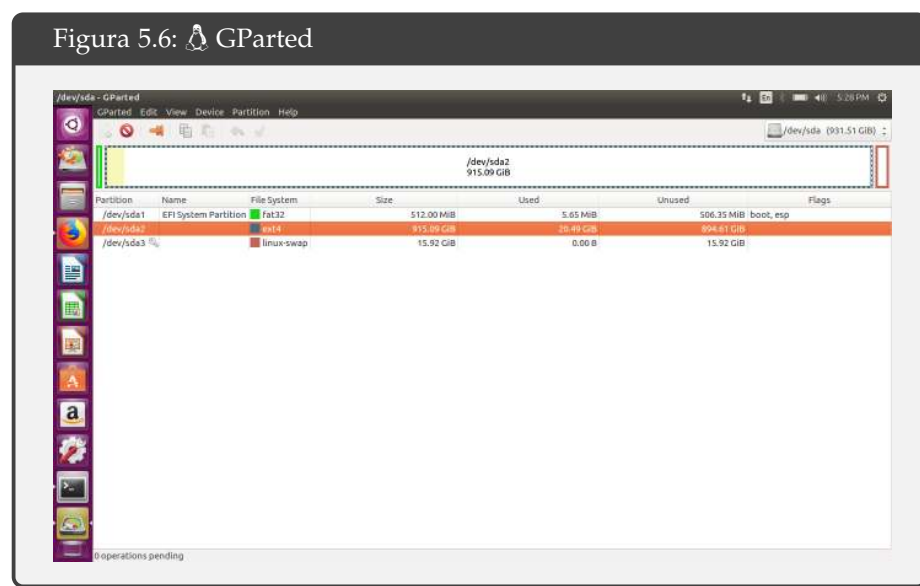
RUTA RELATIVA Una ruta relativa es una ruta que describe la ubicación de un archivo o directorio en relación con el directorio de trabajo actual. En otras palabras, una ruta relativa es una ruta que se especifica desde el directorio actual hasta el archivo o directorio en cuestión.

5.5 EL SISTEMA DE ARCHIVOS DE LINUX

5.5.1 Esquema de particionado

Linux puede instalarse en una variedad de esquemas de particionado, pero uno de los más comunes es el esquema de particionado convencional, que utiliza tres particiones para el sistema:

- Una partición en donde se instala el gestor de arranque y el núcleo del sistema operativo.
- Una partición destinada a almacenar los programas de usuario y documentos.
- Una partición de intercambio, reservada para ser utilizada cuando la memoria RAM se agota.



5.5.2 Nomenclatura de dispositivos

En Linux, los discos duros y particiones se nombran usando un esquema de nomenclatura estandarizado. Cada dispositivo se identifica por un nombre que se genera automáticamente y que puede estar compuesto de diferentes elementos.

El nombre asignado a un dispositivo de almacenamiento consta de tres partes:

- Una sigla identificadora del tipo de dispositivo. En general, los discos rígidos, unidades de estado sólido y dispositivos USB empiezan con las letras «*sd*».
- Una letra secuencial que identifica el dispositivo en relación con otros dispositivos del mismo tipo. El primer dispositivo se denominará «*sda*», el segundo «*sdb*» y así sucesivamente.
- Un número adicional que indica la partición específica del dispositivo. La primera partición en el primer dispositivo será «*sda1*», la segunda será «*sda2*», y así sucesivamente.

! Observación

Es muy importante notar que la denominación «*sda*» hace referencia a todo el dispositivo, mientras que «*sda1*» solo hace referencia a la primera partición del primer dispositivo.

5.5.3 Jerarquía de archivos

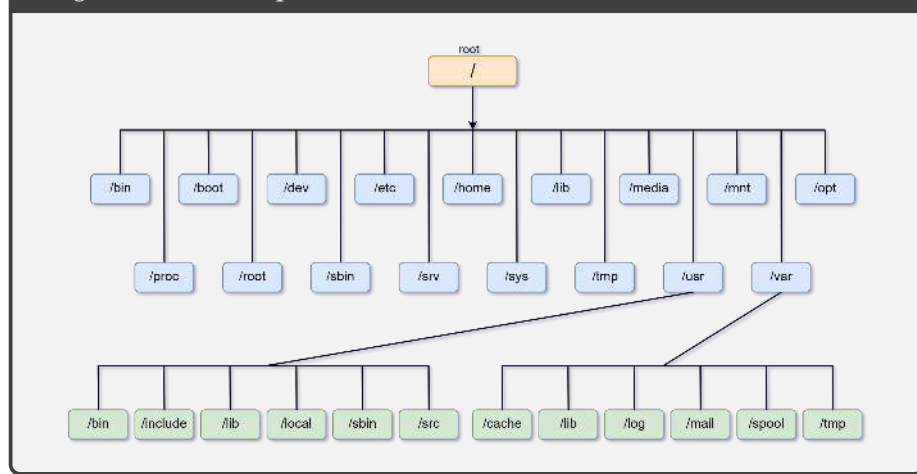
La jerarquía de archivos en el sistema de archivos se organiza en un árbol con una estructura específica. El objetivo de una estructura estandarizada es que los desarrolladores de software y los administradores de sistemas puedan predecir con precisión dónde se encuentran los archivos importantes del sistema y cómo acceder a ellos.

A continuación, se presentan algunos de los directorios más importantes de la jerarquía de archivos:

- / Es la raíz del sistema de archivos. Cualquier otro directorio debe encontrarse dentro de él, independientemente de en cual dispositivo se encuentre.
- /BIN Este directorio contiene los archivos binarios esenciales del sistema, como comandos básicos de GNU/Linux.
- /BOOT Contiene los archivos necesarios para el proceso de arranque del sistema, como el gestor de arranque o el núcleo.
- /DEV En Linux, los dispositivos del hardware se representan como archivos especiales. En esta carpeta se encuentran esos archivos.
- /ETC Este directorio contiene los archivos de configuración del sistema y de las aplicaciones instaladas en el mismo.
- /HOME Este directorio se utiliza para almacenar los directorios personales de los usuarios.

- /LIB Contiene las bibliotecas compartidas necesarias para el funcionamiento del sistema.
- /MNT Este directorio se utiliza para montar sistemas de archivos externos, como unidades flash USB, discos duros externos, etc. En algunas distribuciones esto también se hace en el directorio «/media».
- /PROC El directorio «/proc» es un directorio virtual que contiene información sobre el sistema y los procesos en ejecución. Es generado en tiempo real por el kernel del sistema operativo.
- /ROOT Este directorio es el directorio personal del usuario «root». El usuario *root* es el único que tiene acceso completo al sistema.
- /SBIN Contiene los archivos binarios esenciales del sistema, utilizados principalmente para la administración del sistema.
- /TMP Este directorio se utiliza para almacenar archivos temporales creados por el sistema y los usuarios.
- /USR Contiene la mayoría de las utilidades y aplicaciones multiusuario, es decir, accesibles para todos los usuarios. En otras palabras, contiene los archivos compartidos, pero que no obstante son de sólo lectura.
- /VAR Este directorio contiene archivos que cambian frecuentemente durante la operación del sistema, como archivos de registro y archivos de bases de datos.

Figura 5.7: 🐧 Jerarquía de directorios estándar en Linux.



5.5.4 *Carpetas especiales*

Existen tres carpetas especiales en los sistemas Linux:

- El punto (.) es una carpeta especial que representa el directorio actual en el que se encuentra el usuario. Por ejemplo, si el usuario se encuentra en el directorio `/home/user`, entonces el comando `«ls .»` mostrará el contenido del directorio actual, es decir, `/home/user`.
- Los dos puntos (..) representan el directorio padre del directorio actual. Por ejemplo, si el usuario se encuentra en el directorio `/home/user`, el comando `«ls ..»` mostrará el contenido del directorio padre, es decir, `/home`.
- El símbolo de tilde (~) representa el directorio principal del usuario. Por ejemplo, si el usuario es «user», entonces el comando `«cd ~»` lo llevará al directorio `/home/user`.

5.5.5 *Montaje*

«Montar» significa hacer que un sistema de archivos esté disponible en un punto de montaje especificado en el sistema de archivos raíz. En otras palabras, montar un sistema de archivos implica hacer que el sistema operativo sea consciente de la existencia de un sistema de archivos y asignarlo a un directorio específico en la jerarquía de archivos del sistema. Cuando se monta un sistema de archivos, el sistema operativo lo asigna a un punto de montaje específico en la jerarquía de archivos. El punto de montaje es un directorio en el sistema de archivos raíz, que se utiliza para acceder al contenido del sistema de archivos que se va a montar. Una vez montado, el contenido del sistema de archivos se vuelve accesible en el punto de montaje especificado.

Salida en pantalla						
NAME	MAJ:MIN	RM	SIZE	RO	TYPE	MOUNTPPOINTS
sda	8:0	0	10G	0	disk	
└sda1	8:1	0	100M	0	part	/boot
└sda2	8:2	0	9,9G	0	part	/
sdb	8:16	0	7,5G	0	disk	
└sdb1	8:17	0	7,4G	0	part	/mnt
└sdb4	8:20	0	2,4G	0	part	
sr0	11:0	1	810,3M	0	rom	

El comando «mount» se utiliza para montar un sistema de archivos en Linux. El comando «mount» requiere varios parámetros, incluyendo la ubicación del dispositivo de almacenamiento y el punto de montaje. Por ejemplo, para montar un dispositivo de almacenamiento USB en el punto de montaje «/mnt», el siguiente comando puede ser utilizado:

 Bash

```
sudo mount /dev/sdb1 /mnt
```

! Observación

El uso del comando «sudo» en el principio del comando indica que se necesita permisos de administrador para montar el dispositivo de almacenamiento.

Para desmontar un sistema de archivos, se utiliza el comando «umount». El comando «umount» se utiliza para desmontar un sistema de archivos que se encuentra actualmente en uso y liberar el punto de montaje. Por ejemplo, para desmontar el dispositivo de almacenamiento USB montado anteriormente podemos usar el siguiente comando:

 Bash

```
sudo umount /mnt
```

5.5.6 Enlaces

En Linux, existen dos tipos de enlaces para archivos: enlaces suaves (también conocidos como enlaces simbólicos o «*soft links*») y enlaces duros (también conocidos como enlaces físicos o «*hard links*»).

Un enlace suave es un archivo especial que apunta a otro archivo en el sistema de archivos. El enlace suave no contiene los datos del archivo al que apunta, sino simplemente la ruta del archivo. Si se elimina el archivo original, el enlace suave se vuelve inútil. Los enlaces suaves se utilizan comúnmente para hacer referencia a archivos en diferentes ubicaciones y para crear accesos directos en el sistema de archivos.

! Observación

Los enlaces suaves de Linux son equivalentes a los «*accesos directos*» en los sistemas Windows.

Para crear un enlace suave en Linux, se utiliza el comando «ln». El comando «ln» crea un enlace entre dos archivos y la opción «-s» indica que se creará un enlace suave. Por ejemplo, para crear un enlace suave llamado «enlace» que apunte al archivo «archivo», se puede usar el siguiente comando:

 Bash

```
ln -s archivo enlace
```

Un enlace duro es una entrada de directorio adicional que apunta al mismo archivo en el sistema de archivos. A diferencia de los enlaces suaves, los enlaces duros no son archivos independientes y no pueden apuntar a archivos en diferentes sistemas de archivos o particiones. Si se elimina el archivo original, el enlace duro sigue siendo válido y no se elimina hasta que todos los enlaces se eliminan.

Para crear un enlace duro en Linux, se utiliza el comando «ln» sin la opción «-s». Por ejemplo, para crear un enlace duro llamado «enlace» que apunte al archivo «archivo», se puede usar el siguiente comando:

 Bash

```
ln archivo enlace
```

Parte II

MANEJO DE BASH

«Pensar no garantiza que no cometeremos errores. Pero no pensar garantiza que lo haremos».

Leslie Lamport

COMANDOS BÁSICOS

6.1 INTRODUCCIÓN

Bash es un intérprete de línea de comandos y un lenguaje de programación de scripting utilizado principalmente en sistemas operativos tipo Unix. Es el intérprete de comandos por defecto en la mayoría de las distribuciones de Linux.

Permite a los usuarios interactuar con el sistema operativo mediante la entrada de comandos en un emulador de terminal. Con Bash, los usuarios pueden ejecutar programas, gestionar archivos y directorios, automatizar tareas y realizar una amplia variedad de operaciones de administración del sistema.

Además, también es un lenguaje de scripting que permite a los usuarios escribir programas para automatizar tareas repetitivas o complejas. Estos scripts pueden incluir una serie de comandos de Bash y se pueden ejecutar en una terminal o programarse para que se ejecuten automáticamente en momentos específicos.

SINTAXIS La sintaxis de un comando es la estructura y el formato que se deben seguir para escribir y ejecutar correctamente un comando en un sistema operativo. La sintaxis incluye el nombre del comando, las opciones y los argumentos necesarios, así como la forma en que deben escribirse y ordenarse.

Cada comando tiene su propia sintaxis única, pero en general, la mayoría sigue una estructura básica:

`comando [opciones] [argumentos]`

- El comando es el nombre del programa o acción que se desea ejecutar.
- Las opciones son indicadores adicionales que se agregan al comando para modificar su comportamiento. Las opciones generalmente se escriben después del comando y comienzan con un guion (-) o dos guiones (--).




Es importante seguir la sintaxis correcta de un comando, ya que de lo contrario, el comando puede no funcionar como se espera o puede producir errores.



- Los argumentos son valores adicionales que se pasan al comando para que actúe sobre ellos. Los argumentos generalmente se escriben después de las opciones y pueden ser nombres de archivo, rutas de directorio, nombres de usuario, números, etc.
- La mayoría de los comandos admiten la opción «--help» para consultar su ayuda.

! Observación

En la sintaxis de un comando, los corchetes se utilizan para indicar que un elemento es opcional. Esto significa que puede omitirse o incluirse según sea necesario.

HISTORIAL Cada vez que se escribe y ejecuta un comando en Bash, se agrega al historial de comandos. Esto permite acceder a los comandos previamente ejecutados para volver a ejecutarlos o modificarlos en lugar de volver a escribirlos desde cero. Para poder acceder fácilmente al historial, se pueden presionar las siguientes teclas:

-  Flecha hacia arriba: Muestra el comando anterior en el historial. Si se sigue presionando, se mostrarán comandos anteriores en el historial en orden cronológico inverso.
-  Flecha hacia abajo: Análogo pero en orden cronológico.
-  Ctrl+R: Abre una búsqueda reversa en el historial de comandos. Al escribir una palabra clave o frase, se mostrarán los comandos anteriores que coinciden con la búsqueda.

AUTOCOMPLETADO Cuando se escribe un comando o una ruta en el terminal y se presiona la tecla  Tab, Bash intenta completar automáticamente el comando o la ruta. Si hay una sola opción disponible, Bash la completará automáticamente. Si hay varias opciones disponibles, Bash mostrará una lista de opciones que coinciden con lo que se ha escrito hasta el momento. Si se sigue presionando la tecla  Tab, Bash continuará mostrando opciones adicionales.

6.1.1 *man*

El comando `man` es una herramienta de línea de comandos que proporciona información detallada sobre otros comandos, funciones y bibliotecas.

«*man*» es una abreviatura de «*manual*».





La sintaxis básica del comando «man» es la siguiente:

```
man [opciones] comando
```


Las opciones más comunes incluyen:

- «-f» para buscar descripciones cortas de comandos relacionados con una palabra clave.
- «-k» para buscar páginas del manual que contienen una palabra clave específica.

Para navegar dentro del programa «man», se pueden usar varias teclas moverse por la página, buscar texto y salir del programa:

- Las teclas de  flecha arriba y  abajo: se pueden utilizar para desplazarse hacia arriba y hacia abajo por la página del manual.
- La tecla « /» seguida de un texto de búsqueda: se puede utilizar para buscar una palabra o frase en la página del manual actual.
- La tecla « q»: se puede utilizar para salir del programa man y volver al terminal.

! Observación


Las páginas del manual también pueden consultarse en internet: « Linux Manpages Online»

Ejemplo 6.1. Para poder observar la documentación del comando man podemos usar el siguiente comando:

```
 Bash
```

```
man man
```

Ejemplo 6.2. Podemos consultar cuales comandos sirven para «listar directorios» con el siguiente comando:


```
 Bash
```

```
man -k "list directory"
```

 Salida en pantalla

```
dir (1) - list directory contents
ls (1) - list directory contents
ls (1p) - list directory contents
vdir (1) - list directory contents
```

Ejemplo 6.3. La descripción corta del comando «clear» la podemos observar con el siguiente comando:

 Bash

```
man -f clear
```

 Salida en pantalla

```
clear (1) - clear the terminal screen
clear (3x) - clear all or part of a curses window
```

6.1.2 clear

El comando `clear` es un comando que se utiliza para limpiar la pantalla del terminal, eliminando todos los comandos y resultados previos que se hayan mostrado. Al ejecutar este comando, la pantalla del terminal se restablecerá, dejando la línea de comando en la parte superior de la pantalla.

La sintaxis básica del comando `clear` es la siguiente:

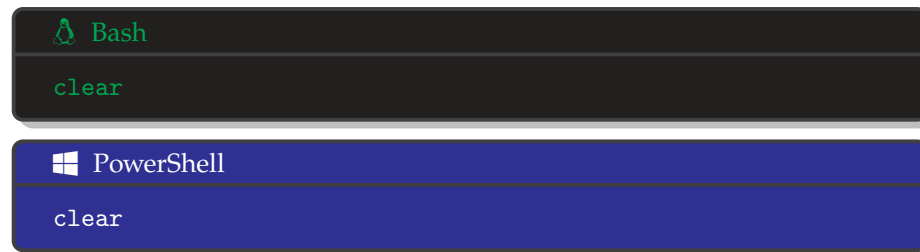
```
clear [-x]
```

La opción «-x» puede agregarse para evitar que se borre el historial desplazamiento del emulador de terminal.

! Observación

El atajo de teclado  Ctrl+L es equivalente al comando `clear`.

Ejemplo 6.4. Para borrar el contenido de la pantalla se utiliza el siguiente comando:



6.1.3 *echo*

El comando `echo` es un comando de la línea de comandos que se utiliza para imprimir texto en la pantalla del terminal.

La sintaxis básica del comando `echo` es la siguiente:

```
echo [opciones] texto
```

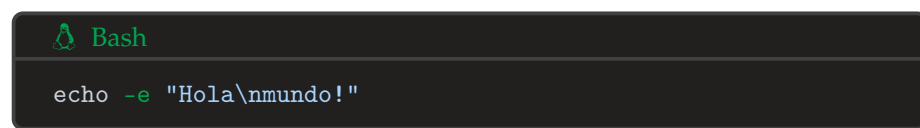
El comando `echo` también admite varias opciones y parámetros adicionales, que permiten ajustar el formato de la salida y realizar otras tareas:

- «-n»: evita que el comando agregue un carácter de nueva línea al final de la salida. En otras palabras, la siguiente línea del terminal comenzará en la misma línea en la que termina la salida del comando `echo`.
- «-e»: activa el soporte de secuencias de escape en la salida. Por ejemplo, se pueden utilizar secuencias de escape para agregar saltos de línea al texto.

! Observación

La utilidad del comando `echo` resultará mas evidente una vez que se estudien las redirecciones y los scripts.

Ejemplo 6.5. Para escribir dos palabras separadas por una línea se escribe el siguiente comando:



 PowerShell
`echo` Hola mundo!

6.1.4 *history*

El comando `history` muestra una lista de los comandos que se han ejecutado anteriormente. La sintaxis básica del comando «`history`» es la siguiente:

```
history [-c]
```

Puede agregarse la opción «`-c`» para borrar el historial.

! Observación

El comando `history` no es un programa de usuario, sino una funcionalidad que provee Bash.


Ejemplo 6.6. Para ver los últimos comandos ingresados se usa el siguiente comando:

 Bash

```
history
```

 Salida en pantalla

```
1 cd ~  
2 ls  
3 history
```

 PowerShell
`history`

6.1.5 *Ejercicios*

Ejercicio 6.7. Utilice el comando «`man`» para averiguar el propósito y funcionamiento de los comandos «`date`» y «`cal`». Pruebe cada uno de ellos.

6.2 SISTEMA DE ARCHIVOS

6.2.1 *pwd*

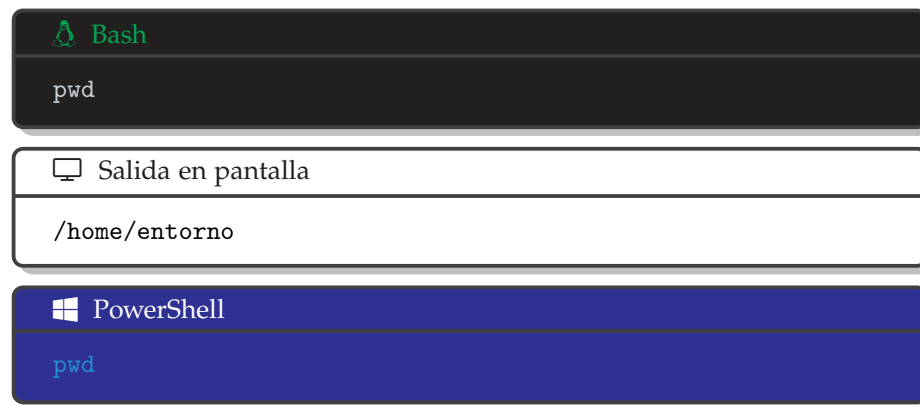
El comando «pwd» muestra la ruta completa del directorio de trabajo actual en el sistema de archivos. La sintaxis básica del comando «pwd» es la siguiente:

```
pwd [-P]
```

«*pwd*» son las siglas de «Print Working Directory» («mostrar directorio de trabajo»).

Puede agregarse la opción «-P» para mostrar el directorio sin enlaces simbólicos.

Ejemplo 6.8. Para mostrar la ruta del directorio de trabajo actual usamos:



6.2.2 *cd*

El comando «cd» se utiliza para cambiar el directorio de trabajo actual en el sistema de archivos. La sintaxis básica del comando «cd» es la siguiente:

```
cd [ruta]
```

«*cd*» son las iniciales de «cambiar directorio».

- Si no se especifica una ruta, el comando es equivalente a «cd ~».
- Si se utiliza el argumento «-», el comando «cd» vuelve al directorio de trabajo anterior.

! Observación

A diferencia de los programas anteriores, «cd» es un comando interno del shell.

Ejemplo 6.9. Para ir al directorio que contiene la carpeta personal del usuario actual usamos el siguiente comando:

 Bash

```
cd ~/. ..
```

 PowerShell

```
cd ~/. ..
```

6.2.3 `ls`

El comando «ls» se utiliza para listar los archivos y directorios en el directorio de trabajo actual o en una ubicación específica del sistema de archivos. La sintaxis básica del comando «ls» es simplemente:

«ls» es una abreviación de «listar».

```
ls [ruta]
```

Si se omite la ruta, se asume que se desea listar los archivos y directorios en el directorio de trabajo actual. Además, se pueden utilizar opciones para personalizar la salida del comando «ls». Algunas opciones comunes incluyen:


- «-l»: muestra la lista en formato largo, que incluye información detallada sobre cada archivo o directorio.
- «-h»: muestra el tamaño de los archivos en formato legible por humanos, como «KB» o «MB».
- «-a»: muestra todos los archivos y directorios, incluyendo los que comienzan con un punto.

! Observación

En el sistema de archivos de Linux, los archivos que comienzan con un punto se consideran archivos ocultos.

Ejemplo 6.10. Para mostrar el contenido del directorio actual usamos el siguiente comando:

```

 Bash
ls

```


Salida en pantalla

```

dsh dsh.c entornos final hola.c hola.py hola.s

```


```

 PowerShell
ls

```

Ejemplo 6.11. Si además queremos ver información adicional, agregamos la opción «-l»:

```

 Bash
ls -l

```

Salida en pantalla


```

-rwxr-xr-x 1 damian wheel 15832 abr 12 19:33 dsh
-rw-r--r-- 1 damian wheel 1130 abr 12 19:32 dsh.c
drwxr-xr-x 6 damian wheel 4096 abr 17 13:54 entornos
drwxr-xr-x 4 damian wheel 4096 abr 19 01:03 final
-rw-r--r-- 1 damian wheel 60 mar 20 15:05 hola.c
-rw-r--r-- 1 damian wheel 35 mar 28 19:06 hola.py
-rw-r--r-- 1 damian wheel 462 mar 28 19:40 hola.s

```

Ejemplo 6.12. Para mostrar el contenido del directorio raíz podemos usar el siguiente comando:

```

 Bash
ls /

```

Salida en pantalla

```

bin boot dev etc home lib lib64 lost+found mnt opt proc
root run sbin srv sys tmp usr var

```

```
PowerShell
ls /
```

6.2.4 *mkdir*

El comando «mkdir» se utiliza para crear nuevos directorios en el sistema de archivos.

La sintaxis básica del comando «mkdir» es la siguiente:

```
mkdir [opciones] directorio...
```

«mkdir» significa «make directory» (o «crear directorio» en español).

Donde «opciones» son las opciones que se pueden utilizar para personalizar la creación del directorio y «directorio» es el nombre del directorio que se desea crear.

Algunas de las opciones comunes incluyen:

- «-p»: crea todos los directorios en la ruta especificada, incluso si no existen. Si ya existe un directorio con el mismo nombre que uno de los directorios de la ruta, se ignora sin dar errores.
- «-v»: muestra los detalles de la creación del directorio en la salida estándar de la terminal.

Ejemplo 6.13. Para crear un directorio llamado «carpeta» usamos el siguiente comando:

```
Bash
mkdir carpeta
```

```
PowerShell
mkdir carpeta
```

Ejemplo 6.14. Para crear un directorio llamado «carpeta1» y otro llamado «carpeta2» podemos usar este comando:

```
Bash
mkdir carpeta1 carpeta2
```

Ejemplo 6.15. Para crear un directorio llamado «carpeta1» con una carpeta «carpeta2» adentro, usamos el siguiente comando:

 Bash

```
mkdir -p carpeta1/carpeta2
```

 PowerShell

```
mkdir carpeta1/carpeta2
```

6.2.5 *rmdir*

El comando «*rmdir*» es utilizado para eliminar directorios *vacíos* en el sistema de archivos. La sintaxis básica del comando «*rmdir*» es la siguiente:

```
rmdir [opciones] directorio...
```

«*rmdir*» es la abreviación de «*remove* directorio».

Algunas de las opciones comunes del comando «*rmdir*» incluyen:

- «-p»: dada una ruta, elimina un directorio y sus predecesores.
- «-v»: muestra los detalles de la eliminación del directorio en la salida estándar de la terminal.

! Observación

Es importante tener en cuenta que solo se pueden eliminar directorios vacíos con el comando «*rmdir*».

6.2.6 *touch*

El comando «*touch*» se utiliza para crear archivos vacíos o actualizar la fecha y hora de acceso o modificación de un archivo. La sintaxis básica del comando «*touch*» es la siguiente:

```
touch [opciones] archivo...
```

Algunas de las opciones comunes del comando «touch» incluyen:

- «-a»: actualiza sólo la fecha y hora de acceso del archivo.
- «-m»: actualiza sólo la fecha y hora de modificación del archivo.

Si se utiliza el comando «touch» para crear un archivo que no existe, se crea un archivo vacío con el nombre especificado. Si en cambio se utiliza con un archivo existente, se actualiza la fecha y hora de acceso y/o modificación del archivo sin cambiar su contenido.

! Observación

El comando «touch» es útil en situaciones en las que se necesita establecer manualmente la fecha y hora de acceso o modificación de un archivo, o para crear rápidamente un archivo vacío sin tener que abrir un editor de texto y guardar un archivo vacío.

6.2.7 rm

El comando «rm» es utilizado para eliminar archivos y directorios en el sistema de archivos. La sintaxis básica del comando «rm» es la siguiente:

«*rm*dir» es la abreviación de «*remove*».

```
rm [opciones] archivo...
```

Algunas de las opciones comunes del comando «rm» incluyen:

- «-r»: se utiliza para eliminar directorios y su contenido de forma recursiva.
- «-f»: se utiliza para forzar la eliminación de un archivo si este está protegido contra escritura. Además no da error si los archivos no existen.
- «-i»: pide una confirmación antes de borrar cada archivo.

! Observación

Si se utiliza el comando «rm» para eliminar un archivo, el archivo se eliminará de forma permanente del sistema de archivos; no se mueve a una papelera de reciclaje.

Ejemplo 6.16. Para borrar tres archivos pidiendo confirmación podemos usar el siguiente comando:

 Bash

```
rm -i archivo1 archivo2 archivo3
```

 Salida en pantalla

```
rm: ¿borrar el fichero regular vacío 'archivo1'? (s/n) s
rm: ¿borrar el fichero regular vacío 'archivo2'? (s/n) s
rm: ¿borrar el fichero regular vacío 'archivo3'? (s/n) s
```

Ejemplo 6.17. Para borrar un directorio no vacío junto con todo su contenido usamos el siguiente comando:

 Bash

```
rm -r directorio
```

 PowerShell

```
rm -r directorio
```

6.2.8 cp

El comando «cp» que se utiliza para copiar uno o más archivos o directorios de una ubicación a otra en el sistema de archivos.

La sintaxis básica del comando «cp» es la siguiente:

```
cp [opciones] origen destino
```

«cp» es una
abreviación de
«copiar».

! Observación

Es importante tener en cuenta que si se copia un archivo a un directorio que ya contiene un archivo con el mismo nombre, el archivo existente se sobrescribirá automáticamente sin pedir confirmación.

Algunas de las opciones comunes del comando «cp» incluyen:

- «-r»: indica que se deben copiar los directorios y su contenido de forma recursiva.
- «-v»: muestra los detalles de la copia en la salida estándar de la terminal.
- «-i»: pide confirmación antes de sobrescribir archivos existentes.

Ejemplo 6.18. Para copiar un archivo a otro directorio usamos el siguiente comando:

 Bash

```
cp archivo directorio
```

 PowerShell

```
cp archivo directorio
```

Ejemplo 6.19. Para crear una copia de un archivo en el directorio actual podemos usar el siguiente comando:

 Bash

```
cp archivo copia
```

 PowerShell

```
cp archivo copia
```

Ejemplo 6.20. Para copiar un directorio y su contenido a otro directorio:

 Bash

```
cp -r directorio1 directorio2
```

 PowerShell

```
cp -r directorio1 directorio2
```

6.2.9 *mv*

El comando «mv» se utiliza para mover o renombrar archivos o directorios. Su sintaxis básica es la siguiente:

«mv» es una abreviatura de «mover».

```
mv [opciones] origen destino
```

El comando «mv» también ofrece algunas opciones útiles, entre las que se incluyen:

- «-i»: Solicita confirmación antes de sobrescribir archivos existentes.
- «-f»: Sobrescribe archivos existentes sin preguntar.
- «-v»: Muestra información detallada sobre lo que se está haciendo.

! Observación

Si origen y destino se encuentran dentro de la misma ruta, entonces el comando «mv» cambia el nombre del archivo o directorio.

6.2.10 *df*

El comando «df» sirve para mostrar información sobre el espacio libre y utilizado en sistemas de archivos montados. Su sintaxis básica es la siguiente:

«df» es una abreviatura de "Disk Free" (disco libre en inglés)

```
df [opciones] [ruta]
```

El comando «df» también ofrece algunas opciones útiles, entre las que se incluyen:


- «-h»: Muestra los tamaños de bloque en un formato legible por humanos (por ejemplo, en *kilobytes*, *megabytes*, *gigabytes*, etc.).
- «-T»: Muestra el tipo de sistema de archivos.

! Observación

Por defecto, si se ejecuta sin opciones, «df» muestra información sobre todos los sistemas de archivos montados en el sistema, incluyendo el espacio total, el espacio utilizado, el espacio libre y el porcentaje de uso.

Ejemplo 6.21. Para ver el tipo de sistema de archivos y espacio libre en forma legible, usamos el siguiente comando:

```

 Bash
df -Th

```

Salida en pantalla

S.ficheros	Tipo	Tamaño	Usados	Disp	Uso%	Montado en
dev	devtmpfs	971M	0	971M	0%	/dev
run	tmpfs	979M	644K	978M	1%	/run
/dev/sda2	ext4	9,7G	7,1G	2,1G	78%	/
tmpfs	tmpfs	979M	0	979M	0%	/dev/shm
tmpfs	tmpfs	979M	72M	907M	8%	/tmp
/dev/sda1	vfat	100M	97M	2,9M	98%	/boot
tmpfs	tmpfs	196M	16K	196M	1%	/run/user/1000

6.2.11 du

El comando «du» sirve para mostrar información sobre el espacio utilizado por directorios. Su sintaxis básica es la siguiente:

```
du [opciones] [ruta]
```

«du» es una abreviatura de «Disk Usage» (uso de disco en inglés)

Algunas opciones útiles son:

- «-h»: Muestra los tamaños de archivo en un formato legible por humanos.
- «-s»: Muestra sólo el tamaño total de la ruta especificada, sin detalles de tamaño por directorio.
- «-a»: Muestra información de tamaño también para los archivos, no sólo para los directorios.

! Observación

Por defecto, si se ejecuta sin opciones, «du» muestra la cantidad de espacio utilizado por cada directorio en la ruta actual.

Ejemplo 6.22. Para mostrar el uso del directorio actual usamos el siguiente comando:

```
Bash
du

Salida en pantalla

4  ./semi1/cuartos1
4  ./semi1/cuartos2
12 ./semi1
4  ./semi2/cuartos4
4  ./semi2/cuartos3
12 ./semi2
28 .
```

6.2.12 *ln*

El comando «ln» se utiliza en sistemas Linux para crear enlaces simbólicos o enlaces duros a archivos y directorios. Su sintaxis básica es la siguiente:

```
ln [-s] [archivo original] [nombre del enlace]
```

! Observación

Por defecto «ln» crea enlaces duros, a no ser que se utilice la opción «-s».

6.2.13 *mount*

El comando «mount» se utiliza en sistemas operativos tipo Linux para montar sistemas de archivos en un directorio del sistema de archivos. Su sintaxis básica es la siguiente:

```
mount [-r] dispositivo directorio
```

! Observación

Por defecto «mount» monta los sistemas de archivos en modo de lectura y escritura, a no ser que se use la opción «-r» (sólo lectura). También es importante destacar que para montar un sistema de archivos, normalmente se necesitan privilegios de superusuario, por lo que se debe utilizar el comando «sudo» para ejecutar el comando «mount».

Ejemplo 6.23. Si se desea montar la partición «/dev/sda1» en el directorio «/mnt», se puede utilizar el siguiente comando:

 **Bash**

```
sudo mount /dev/sda1 /mnt
```

Una vez que se ha montado la partición, cualquier archivo que se encuentre en la partición estará disponible en el directorio «/mnt».

6.2.14 *umount*

El comando «umount» se utiliza para desmontar un sistema de archivos previamente montado en un directorio. Su sintaxis básica es la siguiente:

```
umount [opciones] directorio
```

Algunas opciones útiles son las que siguen:

- «-f»: Forzar el desmontaje, aunque la partición esté siendo utilizada por otros procesos. El sistema puede quedar inconsistente.
- «-l»: Desmonta la partición permitiendo que los procesos que utilizan la partición terminen antes de desmontarla.

! Observación

Es importante destacar que para desmontar un sistema de archivos, normalmente se necesitan privilegios de superusuario, por lo que se debe utilizar el comando «sudo» para ejecutar el comando «umount».

Ejemplo 6.24. Si se desea desmontar la partición que previamente se montó en el directorio «/mnt», se puede utilizar el siguiente comando:

```
Bash
sudo umount /mnt
```

6.2.15 *find*

El comando «find» es una herramienta muy útil para buscar archivos y directorios en el sistema de archivos. Su sintaxis básica es la siguiente:

```
find [ruta inicial] [expresiones de búsqueda]
```

Algunas de las expresiones de búsqueda mas importantes son:

- «-type»: Especifica el tipo de archivo que se está buscando.
- «-name»: Especifica el nombre del archivo que se está buscando.
- «-empty»: Busca archivos vacíos.

! Observación

Si no se especifica una ruta de inicio, se buscará a partir del directorio de trabajo actual.

Es posible ejecutar un comando para cada uno de los archivos o directorios encontrados. Para esto debemos agregar al final:

```
find [ruta] [expresiones de búsqueda] -exec comando {} \;
```

donde «{ }» se reemplazará por los nombres de los archivos o directorios encontrados.

Ejemplo 6.25. Para buscar en el directorio de trabajo actual todos los *directorios* que existan se usa el comando:

```
Bash
find -type d
```

Salida en pantalla

```
.  
./semi1  
./semi1/cuartos1  
./semi1/cuartos2  
./semi2  
./semi2/cuartos4  
./semi2/cuartos3
```

Ejemplo 6.26. Para buscar en el directorio de trabajo actual todos los *archivos* que existan se usa el comando:

Bash

```
find -type f
```

Salida en pantalla

```
./francia  
./semi1/croacia  
./semi1/cuartos1/holanda  
./semi1/cuartos1/argentina  
./semi1/argentina  
./semi1/cuartos2/croacia  
./semi1/cuartos2/brasil  
./argentina  
./semi2/cuartos4/marruecos  
./semi2/cuartos4/portugal  
./semi2/marruecos  
./semi2/francia  
./semi2/cuartos3/inglaterra  
./semi2/cuartos3/francia
```

Ejercicio 6.27. Para actualizar la fecha de modificación y accesos de todos los archivos vacíos puede usarse el siguiente comando:

Bash

```
find -type f -empty -exec touch {} \;
```

6.2.16 Comodines

Los comodines, también conocidos como wildcards en inglés, son caracteres especiales que se utilizan para representar patrones de búsqueda en *nombres de archivos y directorios*.

Los comodines pueden ser muy útiles para simplificar tareas que involucren la manipulación de muchos archivos y directorios.

! Observación

La expansión de comodines, es otra de las tareas que llevan a cabo los intérpretes de línea de comandos.

- «*»: Representa cualquier cadena de caracteres, incluyendo una vacía.
- «?»: Representa cualquier carácter individual.
- «[]»: Representa un conjunto de caracteres posibles.
- «{}»: Representa un producto cartesiano de caracteres. No se tienen en cuenta los archivos de la carpeta actual.

Ejemplo 6.28. Para borrar todos los archivos que terminen en «.txt» se puede usar el comando:

 Bash

```
rm *.txt
```

 PowerShell

```
rm *.txt
```

Ejemplo 6.29. Para mostrar todos los programas de usuario que tienen «64» en su nombre usamos el siguiente comando:

 Bash

```
ls /bin/*64*
```

Ejemplo 6.30. Para mostrar las segundas particiones de todas las unidades de almacenamiento podemos usar:

 Bash

```
ls /dev/sd?2
```

Ejemplo 6.31. Para ver cuando ocupan los archivos que comienzan con dos letras y terminan con un número usamos el siguiente comando:

```
Bash
du [a-z][a-z][0-9]
```

Ejemplo 6.32. Para borrar todos los archivos de una sola letra, que terminen en «.jpg» o «.gif» escribimos:

```
Bash
rm [a-z].{gif,jpg}
```

6.2.17 Ejercicios

Ejercicio 6.33. Investigue con el comando «man», que otras opciones tiene el comando «ls». Pruebe algunas de ellas.

Ejercicio 6.34. Averigüe para que sirve el comando «stat».

Ejercicio 6.35. Considere la siguiente salida en pantalla del comando «ls -log»:

```
Salida en pantalla

total 4
drwxr-xr-x 2 4096 may 1 23:46 archivo
-rw-r--r-- 1    0 may 1 23:46 carpeta
```

¿Por que da error el comando «cd carpeta»?


Ejercicio 6.36. Cree un directorio vacío con «mkdir» y luego bórralo con «rmdir».

Ejercicio 6.37. Escriba la siguiente secuencia de comandos:

```
Bash
mkdir carpeta
touch carpeta/archivo
rmdir carpeta
```

Explique por que se produce el error. Proponga una forma de solucionarlo.

Ejercicio 6.38. Escriba la siguiente secuencia de comandos:

```
 Bash
touch carpeta
rmdir carpeta
```


Explique por que se produce el error. Proponga una forma de solucionarlo.

Ejercicio 6.39. Escriba el siguiente comando:

```
 Bash
mkdir carpeta1/carpeta2
```

Explique por que se produce el error. Proponga una forma de solucionarlo.

Ejercicio 6.40. Considere la siguiente salida en pantalla del comando «rm»:

```
 Salida en pantalla
rm: ¿borrar el fichero regular vacío 'archivo' protegido
contra escritura? (s/n) s
```

¿Cuál fue el comando completo? ¿Cómo podría haberse omitido la comprobación de eliminación?

Ejercicio 6.41. Utilice el comando «man» para averiguar sobre el comando «tree». Logre la siguiente salida del comando «tree»:

```
 Salida en pantalla
.
├── argentina
├── francia
├── semi1
│   ├── argentina
│   └── croacia
├── semi2
│   ├── francia
│   └── marruecos
3 directories, 6 files
```

Ejercicio 6.42. Cree una copia idéntica a la estructura anterior en otro directorio.

Ejercicio 6.43. Elimine ambas estructuras de directorios.

Ejercicio 6.44. Investigue con el comando «man», las opciones «-b» y «-u» del comando «cp». Pruebe ambas opciones.

Ejercicio 6.45. Utilice el comando «man» para aprender sobre el comando «readlink».

Ejercicio 6.46. Inserte un *pendrive* en la computadora y utilice el comando «lsblk» para ver el nombre del dispositivo. Intente montarlo y desmontarlo.

Ejercicio 6.47. Utilice el comando «man» y averigüe como buscar con «find», los archivos modificados durante la última semana.

Ejercicio 6.48. Utilice «find» para crear una copia de todos los directorios vacíos del directorio de trabajo actual.

Ejercicio 6.49. Averigüe cuales son los programas de usuario instalados en su computadora, que solo tienen dos letras en su nombre.

Ejercicio 6.50. Cree un archivo con el nombre de cada fecha del año.

Ejercicio 6.51. Escriba la siguiente secuencia de comandos:

```
Bash
mkdir carpeta
touch carpeta/archivo1 carpeta/.archivo2
rm -rf carpeta/*
rmdir carpeta
```

Explique por que se produce el error. Proponga una forma de solucionarlo.

Ejercicio 6.52. Escriba un comando para mostrar los archivos ocultos de la carpeta actual.

6.3 CONTENIDO Y FILTROS













6.3.1 nano

«nano» es un editor de texto de la línea de comandos que se utiliza para crear y modificar archivos en sistemas Linux. Su sintaxis es la siguiente:

```
nano [archivo]
```

Si no se especifica un archivo, se comenzará un documento nuevo. Una vez que se encuentra en el editor, se puede usar el teclado para escribir o editar el archivo de texto.

Para utilizar el programa se pueden usar varias combinaciones de teclas que se señalan en la parte inferior de la pantalla. Algunas de ellas son:

-  Ctrl+O: Para guardar los cambios en un archivo, se utiliza la combinación de teclas  Ctrl + O, que guarda el archivo en su ubicación actual.
-  Shift+flechas: Se puede seleccionar texto manteniendo presionada la tecla  Shift y presionando las flechas del teclado.
-  Ctrl+K: Para cortar un texto seleccionado, se utiliza la combinación de teclas  Ctrl+K.
-  Ctrl+U: Para pegar un texto seleccionado, se utiliza la combinación de teclas  Ctrl+U.
-  Alt+U: Para deshacer la última acción, usamos la combinación de teclas  Alt+U.
-  Ctrl+X: Para salir del editor, se utiliza la combinación de teclas  Ctrl+X, lo que devuelve al usuario a la línea de comandos.

! Observación

Al igual que «man» y al contrario que la mayoría de los comandos que veremos, «nano» es un programa interactivo. Esto quiere decir que después de ingresar el comando, se ingresa a un programa que espera más entrada desde el teclado.

6.3.2 cat


El comando «cat» es una herramienta que se utiliza para concatenar y mostrar el contenido de uno o más archivos de texto. La sintaxis básica del comando «cat» es:

«cat» es una abreviación de «concatenar».

```
cat [-n] archivo...
```

Puede agregarse la opción «-n» para enumerar las líneas emitidas.

Ejemplo 6.53. Para conocer información sobre el CPU podemos observar el contenido del archivo especial «/proc/cpuinfo» con el siguiente comando:

```
 Bash
cat /proc/cpuinfo
```

Ejemplo 6.54. Para ver información sobre la memoria podemos observar el contenido del archivo especial «/proc/meminfo» con el siguiente comando:

```
Bash
cat /proc/meminfo
```

Ejemplo 6.55. Para mostrar información sobre la línea de comando del kernel utilizada para iniciar el sistema operativo y a continuación la versión del mismo, podemos usar el siguiente comando:

```
Bash
cat /proc/cmdline /proc/version
```

6.3.3 less

El comando «less» es una utilidad en línea de comandos que se utiliza para ver el contenido de un archivo de texto de manera paginada. La sintaxis básica es la siguiente:

```
less archivo
```

El nombre del comando «less» deriva de «more» que fue el programa que se utilizaba anteriormente para este propósito

Una vez que se ejecuta el comando «less», se abrirá una vista de página en la terminal que muestra el contenido del archivo. El programa «less» se puede manejar igual que «man».

! Observación

«less» es otro de los programas interactivos que veremos, junto con «man» y «nano».

Ejemplo 6.56. Para ver un registro de los últimos sucesos del sistema podemos paginar el archivo «/var/log/syslog» con el comando:

```
Bash
less /var/log/syslog
```

6.3.4 *head*

El comando «head» se utiliza para mostrar las primeras líneas de un archivo de texto. La sintaxis básica del comando es:

```
head [-n X] archivo...
```

Puede agregarse la opción «-n X» para mostrar las primeras «X» líneas del archivo.

! Observación

Por defecto, «head» muestra las primeras 10 líneas de un archivo.

Ejemplo 6.57. Para ver las primeras cuatro líneas del archivo «/etc/fstab» usamos el siguiente comando:

```
Bash
head -n 4 /etc/fstab
```

Salida en pantalla

```
# Static information about the filesystems.
# See fstab(5) for details.

# <file system> <dir> <type> <options> <dump> <pass>
```

6.3.5 *tail*

El comando «tail» se utiliza para mostrar las últimas líneas de un archivo de texto. La sintaxis básica del comando es:

```
tail [opciones] archivo...
```

Algunas opciones comunes incluyen:

- «-n X»: muestra las últimas «X» líneas del archivo en lugar de las 10 últimas.
- «-f»: sigue en tiempo real los cambios que se realizan en el archivo.

! Observación

La opción «-f» puede ser especialmente útil para monitorear en tiempo real la actividad en un archivo que está siendo constantemente actualizado.

Ejemplo 6.58. Para monitorear los últimos sucesos del sistema podemos observar las últimas líneas del archivo «/var/log/syslog» con el comando:

 Bash

```
tail -f /var/log/syslog
```

6.3.6 *sort*

El comando «sort» es una herramienta que se utiliza para ordenar líneas de texto de un archivo de entrada. La sintaxis básica del comando es:

```
sort [opciones] archivo...
```

Algunas opciones comunes incluyen:

- «-n»: ordena las líneas en orden numérico en lugar de orden alfabético.
- «-r»: ordena las líneas en orden reverso (descendente).
- «-m»: fusiona archivos previamente ordenados.
- «-R»: en vez de ordenar, mezcla el contenido.

Ejemplo 6.59. Para ordenar alfabéticamente los shells del sistema podemos usar el comando «sort» sobre el archivo «/etc/shells»:

 Bash

```
sort /etc/shells
```



Salida en pantalla

```
# /etc/shells: valid login shells
/bin/bash
/bin/dash
/bin/rbash
/bin/sh
/usr/bin/bash
/usr/bin/dash
/usr/bin/rbash
/usr/bin/sh
```

6.3.7 *uniq*

El comando «*uniq*» es una herramienta que se utiliza para encontrar y eliminar líneas duplicadas *consecutivas* en un archivo de entrada. La sintaxis básica del comando es:

```
uniq [opciones] archivo...
```

Algunas opciones comunes incluyen:

- «-c»: para contar el número de ocurrencias de cada línea en el archivo de entrada.
- «-d»: para mostrar sólo las líneas duplicadas.
- «-u»: para mostrar sólo las líneas únicas.

! Observación

Puesto que el comando solo funciona con líneas duplicadas consecutivas, es probable que primero se deba ordenar el archivo.

6.3.8 *strings*

El comando «*strings*» es una herramienta que se utiliza para imprimir las cadenas de texto legibles que se encuentran en los archivos binarios. Esto es útil para buscar información dentro de archivos binarios que no son legibles para un usuario común.

La sintaxis básica es la siguiente:


```
strings [-n MINIMO] archivo...
```

Puede agregarse la opción «- n» para especificar el *MÍNIMO* de caracteres buscado.

Ejemplo 6.60. Para buscar las cadenas de 70 caracteres o mas dentro del archivo «/bin/echo» escribimos:

 Bash

```
strings -n 70 /bin/echo
```

 Salida en pantalla

```
-E disable interpretation of backslash escapes (default)
NOTE: your shell may have its own version of %s, which
usually supersedes
the version described here. Please refer to your shell's
documentation
```

6.3.9 wc

El comando «wc» se utiliza para contar el número de líneas, palabras y bytes en un archivo de texto.

«wc» es una abreviatura de "Word Count" (contar palabras).

```
wc [opciones] archivo...
```

A continuación, se describen las opciones más comunes:

- «-l»: cuenta el número de líneas en el archivo especificado.
- «-w»: cuenta el número de palabras en el archivo especificado.
- «-c»: cuenta el número de bytes en el archivo especificado.
- «-m»: cuenta el número de caracteres en el archivo especificado.

! Observación


Si no se especifican opciones, «wc» cuenta líneas, palabras y caracteres.

6.3.10 *file*

El comando «file» es útil para identificar el formato de un archivo cuando la extensión del archivo no es suficiente para determinar su tipo. La sintaxis básica del comando es:

```
file archivo...
```

Ejemplo 6.61. Podemos averiguar que formato tiene una imagen con el siguiente comando:

 Bash

```
file imagen
```

 Salida en pantalla

```
imagen: JPEG image data, JFIF standard 1.01, resolution  
(DPI), density 300x300, segment length 16, progressive,  
precision 8, 739x688, components 3
```

6.3.11 *cut*

El comando «cut» es una herramienta de procesamiento de texto que se utiliza para cortar secciones de líneas de texto. Permite seleccionar una o varias columnas de texto de un archivo de entrada separado por delimitadores como espacios, tabulaciones o comas.


La sintaxis básica del comando es:

```
cut opciones archivo
```

Algunas de las opciones más comunes son las siguientes:

- «-c CANTIDAD»: Corta una *CANTIDAD* de caracteres.
- «-d DELIMITADOR»: Especifica el *DELIMITADOR* de columna utilizado en el archivo de entrada.
- «-f CAMPOS»: Selecciona *CAMPOS* específicos separados por el delimitador.

Ejemplo 6.62. Podemos consultar cuales son los shells usados por los usuarios del sistema con el comando:

```
 Bash
cut -d : -f 7 /etc/passwd
```

6.3.12 Expresiones regulares

Las expresiones regulares son un conjunto de caracteres especiales que se utilizan para definir *lenguajes*. Un *lenguaje* es simplemente un conjunto de palabras.

Los conceptos básicos de las expresiones regulares son:

CARACTERES LITERALES Los caracteres literales son aquellos que se escriben directamente en la expresión regular.

Ejemplo 6.63. La expresión regular «'h'» define el lenguaje $\{h\}$, cuya única palabra es la letra *h*.

Ejemplo 6.64. El lenguaje con la palabra vacía ($\{\epsilon\}$) se define mediante la expresión regular «''».

CONCATENACIÓN La concatenación de lenguajes permite definir un lenguaje nuevo, a partir de otros dos. Las palabras del nuevo lenguaje se construyen teniendo como prefijos a las palabras del primero y como sufijos a las palabras del segundo. La concatenación de lenguajes no utiliza ningún carácter especial, simplemente se escribe un lenguaje a continuación de otro.

Ejemplo 6.65. El lenguaje definido por la expresión regular «'si'» es la concatenación del lenguaje definido por las expresiones «'s'» y «'i'»; formando el conjunto $\{si\}$.

UNION La unión de lenguajes permite definir un lenguaje con palabras de otros dos. La denotamos con el carácter «|».

Ejemplo 6.66. El lenguaje definido por la expresión regular «'a|b|c'» es el lenguaje $\{a, b, c\}$.

PARÉNTESIS Es posible agrupar expresiones regulares mediante la utilización de paréntesis.

Ejemplo 6.67. La expresión regular «'(a|b|c)(1|2|3)''' es la concatenación de los lenguajes definidos por «'a|b|c'» y «'1|2|3'»; en definitiva es el lenguaje $\{a1, a2, a3, b1, b2, b3, c1, c2, c3\}$.

REPETICIONES Es posible concatenar un lenguaje consigo mismo, para dar lugar a nuevos lenguajes:

- El operador «?» permite concatenar un lenguaje a lo sumo una sola vez.

Ejemplo 6.68. El lenguaje definido por la expresión regular «'a(1)?z'» es el lenguaje {az, a1z}.

- El operador «{n,m}» permite concatenar un lenguaje entre n y m veces.

Ejemplo 6.69. Puede definirse el lenguaje {aaa,aaaa,aaaaa} con la expresión regular «'a{3,5}'».

! Observación

Puede omitirse el límite inferior para repetir hasta m veces, o el límite superior para repetirse a partir de n veces.

- El operador «+» permite concatenar un lenguaje entre una e infinitas veces.

Ejemplo 6.70. El lenguaje definido por la expresión regular «'x+'» es el lenguaje {x, xx, xxx, xxxx, ...}.

- El operador «*» permite concatenar un lenguaje entre cero e infinitas veces.

Ejemplo 6.71. El lenguaje definido por la expresión regular «'x*'» es el lenguaje {", x, xx, xxx, xxxx, ...}.

ANCLAS Las anclas son caracteres especiales que indican el inicio o el final de una cadena.

- El ancla «^» se utiliza para indicar el inicio de una cadena.
- El ancla «\$» indica el final de una cadena.

! Observación

La utilidad de las anclas quedará en evidencia cuando las utilicemos para buscar contenido dentro de archivos.

Además, contamos con algunos lenguajes ya definidos, como por ejemplo:

- La clase «[:digit:]» representa cualquier carácter numérico.

Ejemplo 6.72. La expresión regular «[:digit:]» define el lenguaje {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.

- La clase «[:alpha:]» representa cualquier carácter alfabético.
- La clase «[:alnum:]» representa cualquier carácter alfanumérico.
- La clase «[:lower:]» representa cualquier carácter en minúscula.
- La clase «[:upper:]» representa cualquier carácter en minúscula.
- La clase «[:space:]» representa cualquier carácter en blanco.
- La clase «[d-r]» representa cualquier carácter entre la letra «d» y la «r».

Ejemplo 6.73. La expresión regular «[a-c]» define el lenguaje $\{a, b, c\}$.

- La clase «.» representa cualquier carácter.

6.3.13 *tr*

El comando «tr» es una herramienta modificar caracteres en una cadena de texto. El comando toma la entrada estándar y la traduce o modifica de acuerdo con los parámetros especificados enviando el resultado se envía a la salida estándar.

«tr» es una
abreviación de
«traducir».

Se puede utilizar para realizar diversas operaciones de manipulación de texto, como eliminar o reemplazar caracteres, cambiar mayúsculas y minúsculas, y eliminar espacios en blanco. También pueden usarse clases de caracteres.

La sintaxis básica es la siguiente:

```
tr [opciones] CADENA1 [CADENA2]
```

Algunas de las opciones mas comunes son:

- «-d»: sirve para eliminar los caracteres especificados en el conjunto de caracteres a traducir.
- «-s»: sustituye secuencias de caracteres repetidos, por uno solo de ellos.

! Observación

La utilidad de «tr» resultara mas evidente cuando se estudien las re-direcciones.

6.3.14 *grep*

El comando «grep» se utiliza para buscar patrones de texto en el contenido de archivos. La sintaxis básica del comando es la siguiente:

```
grep [opciones] patron archivo
```

«*grep*» son las iniciales de «Global Regular Expresion Print».

Algunas opciones comunes son:

- «-i»: busca el patrón de forma insensible a mayúsculas y minúsculas.
- «-v»: muestra las líneas que no contienen el patrón.
- «-c»: muestra el número de líneas que contienen el patrón.
- «-n»: muestra el número de línea de cada línea que contiene el patrón.
- «-r»: busca recursivamente en todos los archivos dentro de un directorio y sus subdirectorios.
- «-E»: habilita el uso de expresiones regulares extendidas. Las expresiones regulares extendidas permiten caracteres especiales como «?», «+», «{ }», «|» y «()».
- «-o»: muestra sólo la parte de la línea que coincide con el patrón de búsqueda.

! Observación

Por defecto, «grep» muestra toda la línea que contiene la coincidencia.

Ejemplo 6.74. Para buscar la línea donde se encuentra definido el usuario «root» podemos usar el siguiente comando:


 Bash

```
grep root /etc/passwd
```


 Salida en pantalla

```
root:x:0:0::/root:/bin/bash
```

Ejemplo 6.75. Para buscar cuantas veces aparece la palabra «GNU» en la documentación de Bash usamos el siguiente comando:

 Bash

```
grep -c GNU /usr/share/doc/bash/*
```

 Salida en pantalla

```
/usr/share/doc/bash/bash.html:9
/usr/share/doc/bash/bashref.html:49
/usr/share/doc/bash/CHANGES:18
/usr/share/doc/bash/COMPAT:0
/usr/share/doc/bash/FAQ:10
/usr/share/doc/bash/INTRO:1
/usr/share/doc/bash/NEWS:10
/usr/share/doc/bash/POSIX:0
/usr/share/doc/bash/RBASH:0
/usr/share/doc/bash/README:2
```


6.3.15 Ejercicios

Ejercicio 6.76. Utilice «nano» para crear un archivo de texto. Luego visualice dicho contenido en el emulador de terminal con el comando «cat».


Ejercicio 6.77. En una *consola virtual*, utilice «cat» para ver el contenido de «/usr/share/doc/bash/README». Tome nota de las dificultades que se presentan al ver el archivo y proponga una forma de solucionarlo.

Ejercicio 6.78. Utilice el comando «man» para averiguar como mostrar con «head» todas las líneas de un archivo, salvo las últimas 3.

Ejercicio 6.79. Cree un archivo vacío y, en un emulador de terminal, utilice «tail» para monitorear su contenido. Modifique el contenido del archivo en otra aplicación y observe como esos cambios se reflejan en el emulador de terminal.

Ejercicio 6.80. Descargue la siguiente  fotografía y utilice el comando «strings» para averiguar con que cámara se tomo dicha imagen.

Ejercicio 6.81. Cree un archivo de python con el siguiente código:

 Código

```
print("Hola mundo!")
```

Utilice el comando «file» para averiguar que tipo de archivo es.

Agregue la siguiente línea al comienzo del archivo:

```
</> Código
```

```
#!/bin/python
```

Vuelva a averiguar el tipo del archivo.

Ejercicio 6.82. Cree un archivo con el siguiente contenido:

```
</> Código
```

```
Año,Marca,Modelo,Descripción,Precio
1997,Ford,E350,"ac, ABS, moon",3000.00
1999,Chevy,Venture,Extended Edition,4900.00
1999,Chevyr,Venture,"Extended Edition, Very Large",5000.00
1996,Jeep,Grand Cherokee,"MUST SELL! air, moon roof",4799.00
```

Utilice el comando «cut» para obtener todas las marcas.

Ejercicio 6.83. Utilice el comando «man» para averiguar el propósito y funcionamiento de los comandos «comm» y «diff». Pruebe cada uno de ellos.

Ejercicio 6.84. Escriba expresiones regulares que definan los siguientes lenguajes:

1. $\{xx, yx\}$.
2. Palabras con una cantidad par de letras x .
3. Palabras con una cantidad impar de letras x .
4. Palabras formadas solamente por letras x , o solamente por letras y .
5. Palabras que comienzan por letras x y siguen por letras y .
6. Palabras donde cada letra y se encuentra entre un par de letras x .
7. Palabras que terminen en *.jpg* o *.jpeg*.

Ejercicio 6.85. Utilice el comando «man» para consultar que hacen las opciones «-A» y «-B» del comando «grep».

6.4 SECUENCIACIÓN, REDIRECCIÓN Y TUBERÍAS

6.4.1 Secuenciación

La secuenciación de comandos en Bash se utiliza para ejecutar varios comandos de forma consecutiva en una sola línea. Se pueden utilizar diferentes operadores de secuenciación para controlar cómo se ejecutan los comandos en función de sus resultados.

! Observación

Los operadores de secuenciación, son otras de las cuestiones de las cuales se encarga el shell.

- «;»: Se utiliza para ejecutar varios comandos en secuencia, independientemente del resultado de cada uno de ellos.
- «&&»: Se utiliza para ejecutar el siguiente comando solo si el comando anterior tuvo éxito.
- «||»: Se utiliza para ejecutar el siguiente comando solo si el comando anterior falló.

Ejemplo 6.86. Para crear dos archivos y verificar que se hayan creado, podemos usar el siguiente comando:

 Bash

```
touch archivo1 archivo2; ls archivo1 archivo2
```

Ejercicio 6.87. Podemos intentar borrar un directorio y mostrar un mensaje en caso de éxito con el comando:

 Bash

```
rmmdir directorio && echo "El directorio se ha borrado."
```

Ejercicio 6.88. Análogamente, podemos mostrar el mensaje si el directorio no se creó:

 Bash

```
rmmdir directorio || echo "El directorio no se ha borrado."
```

6.4.2 Redirección

Las redirecciones en Bash son una forma de cambiar la entrada y/o salida de un comando a diferentes archivos o dispositivos.

Hay varios tipos de redirecciones en Bash:

- Redirección de salida estándar «>»: esta redirección cambia la salida en pantalla del comando a un archivo. La sintaxis básica de la redirección de salida estándar es:

```
comando > archivo
```

! Observación

La redirección de salida estándar substituye por completo el contenido de un archivo.

Ejemplo 6.89. Si queremos que la salida del comando «ls» se guarde en el archivo «lista.txt», podemos usar la redirección de salida de la siguiente manera:

 Bash

```
ls > lista.txt
```

 PowerShell

```
ls > lista.txt
```

- Anexación de salida estándar «>>»: La redirección «>>» en Bash se utiliza para redirigir la salida estándar de un comando a un archivo y agregarla al final del archivo en lugar de reemplazar su contenido, como hace la redirección ">". La sintaxis básica de la anexación de salida estándar es:

```
comando >> archivo
```

Ejemplo 6.90. Si queremos crear un archivo con los textos «*Hola mundo!*» y «*Chau mundo!*» en dos líneas, puedes usar el siguiente comando:

 Bash

```
echo "Hola mundo!" > saludo  
echo "Chau mundo!" >> saludo
```

 PowerShell

```
echo "Hola mundo!" > saludo  
echo "Chau mundo!" >> saludo
```

- Redirección de salida de error «2>»: Esta redirección cambia la salida de error de un comando a un archivo. La sintaxis básica de la redirección de salida estándar es:

```
comando 2> archivo
```

! Observación

La salida de error en sistemas Linux es una salida de información que se genera cuando un comando o programa encuentra un error durante su ejecución. Esta salida suele estar separada de la salida estándar; aunque se suele mostrar en la misma pantalla.

Ejemplo 6.91. Para escribir los mensajes de error de «mkdir» en un archivo usamos el siguiente comando:

 Bash

```
mkdir . 2> error
```

 PowerShell

```
mkdir . 2> error
```

- Redirección de salidas «&>»: La redirección «&>» en Bash es una forma de redireccionar tanto la salida estándar como la salida de error estándar de un comando a un archivo o dispositivo. La sintaxis básica es la siguiente:

```
comando &> archivo
```

Ejemplo 6.92. Para redirigir ambas salidas del comando «mkdir» en un archivo usamos el siguiente comando:

 Bash

```
mkdir -v carpeta carpeta &> salidas
```


En Linux y sistemas similares, la carpeta «/dev» contiene algunos archivos especiales que son útiles a la hora de combinarlos con redirecciones. Algunos de ellos son:

- «/dev/null»: un archivo que no almacena ningún dato, sino que simplemente los descarta.
- «/dev/tty*»: son archivos que representan a las consolas virtuales.
- «/dev/pts/*»: son archivos que representan a los emuladores de terminales.
- «/dev/zero»: un archivo infinito que contiene datos cero (0).
- «/dev/random»: un archivo que genera datos aleatorios.

6.4.3 Tuberías

En Bash, una tubería (también conocida como «*pipe*» en inglés) permite conectar la *salida* de un comando con la *entrada* de otro, lo que permite crear una cadena de procesos que trabajan juntos para realizar una tarea más compleja. La sintaxis de una tubería es la siguiente:


```
comando | comando
```

6.4.4 Ejercicios

Ejercicio 6.93. Utilice «man» para averiguar sobre los comandos «true» y «false». Utilice secuenciación para mostrar un mensaje en pantalla luego de la ejecución de cada uno de ellos.

Ejercicio 6.94. Piense que ocurre tras ejecutar el comando «echo 2 * 3 >10». Verifíquelo.

Ejercicio 6.95. Busque en Internet o consulte a alguna IA sobre la redirección de entrada «<». Explique que hace el siguiente comando:

```
 Bash  
cat < archivo1 > archivo2
```

Ejercicio 6.96. ¿Cómo puede utilizarse la redirección de salida estándar para crear archivos vacíos, sin usar «touch»?

Ejercicio 6.97. ¿Cuanto ocupa un archivo con todas las «contraseñas» posibles de cuatro letras formadas por caracteres alfabéticos en minúscula?

Ejercicio 6.98. Cree un archivo con una lista de todos los archivos del sistema.

Ejercicio 6.99. Escriba una secuencia de tuberías para observar la tercer línea de un archivo.

Ejercicio 6.100. Escriba una secuencia de tuberías para averiguar cual es el archivo con mayor cantidad de líneas.

Ejercicio 6.101. Escriba una secuencia de tuberías para averiguar cuantos emuladores de terminal hay abiertos.

Ejercicio 6.102. Escriba una secuencia de tuberías para mostrar los tamaños de los archivos del directorio actual.

Ejercicio 6.103. Escriba una secuencia de tuberías para averiguar cuantas veces se ejecutó el comando «cd».

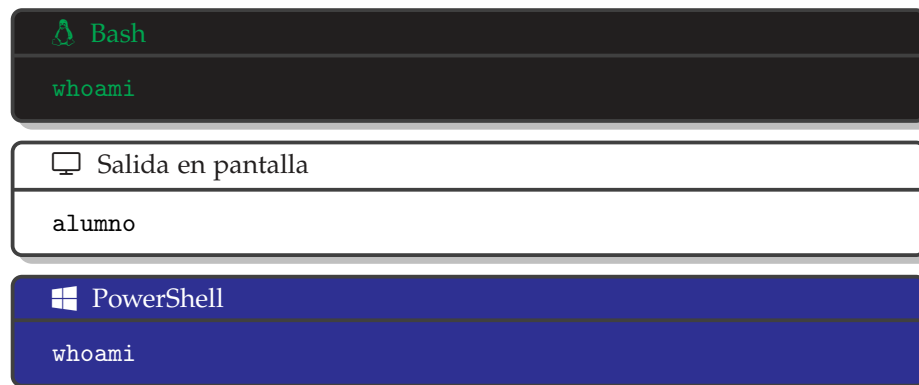
6.5 USUARIOS Y GRUPOS

6.5.1 *whoami*

El comando «whoami» muestra el nombre de usuario de la cuenta actualmente activa en la sesión de terminal. Es útil cuando se necesita saber el nombre de usuario para realizar una acción o para verificar que la cuenta activa es la deseada.

El comando «whoami» significa «Who Am I» (quién soy yo).

Ejemplo 6.104. Para saber cuál es el usuario activo, simplemente escribimos:



6.5.2 *id*

El comando «id» obtiene información sobre la identidad de un usuario, como su nombre de usuario, identificador de usuario (*UID*), identificador de grupo (*GID*) y los grupos a los que pertenece. La sintaxis básica del comando es la siguiente:

```
id [usuario]
```

Si no se especifica el usuario, se considera el usuario que lo ejecuta.

! Observación

Dicha información se obtiene del archivo «/etc/passwd».

Ejemplo 6.105. Para conocer información del usuario *root* usamos el siguiente comando:

```
Bash
```

```
id root
```

Salida en pantalla

```
uid=0(root) gid=0(root) grupos=0(root)
```

6.5.3 *who*

El comando «who» muestra información acerca de los usuarios que están actualmente conectados al sistema. Esta información incluye el nombre de usuario, la consola virtual en la que están conectados, la fecha y hora en que iniciaron sesión y la dirección IP de la máquina desde la que se conectaron.

Ejemplo 6.106. Para saber cuáles usuarios están conectados podemos escribir:

```
Bash
```

```
who
```

Salida en pantalla

```
damian tty1 2023-05-01 04:21  
root tty2 2023-05-03 06:01 (190.2.103.252)
```

6.5.4 *su*

El comando «su» es utilizado para cambiar de usuario en una sesión de terminal. La básica del comando es la siguiente:

«su» es un acrónimo de «substituir usuario».

```
su [USUARIO]
```

Si no se especifica un *USUARIO*, se cambia al usuario *root*.

! Observación

Al ingresar el comando «su», el sistema solicitará la contraseña *del usuario al que se quiere cambiar*. Si la contraseña es correcta se iniciará una shell para el usuario indicado.

Ejemplo 6.107. Para empezar a utilizar la terminal como el usuario root escribimos:

 Bash

```
su
```

6.5.5 sudo

Con «sudo», un usuario común puede obtener temporariamente los permisos de un usuario con otros privilegios, como el superusuario (*root*), para realizar tareas que requieren permisos especiales. La sintaxis básica del comando «sudo» es la siguiente:

```
sudo [-u USUARIO] comando
```

Si no se especifica un *USUARIO*, se obtienen los privilegios del usuario *root*.

! Observación

Al ingresar el comando «sudo», el sistema solicitará la contraseña *del usuario que ejecuta el comando*. Si la contraseña es correcta y el usuario tiene permisos para usar «sudo», el comando especificado se ejecutará con los permisos correspondientes.

Ejemplo 6.108. Para ver el archivo de contraseñas de Linux podemos usar el siguiente comando:

 Bash

```
sudo head -n 5 /etc/shadow
```

 Salida en pantalla

```
root:x:0:0::/root:/bin/bash
bin:x:1:1:::/usr/bin/nologin
daemon:x:2:2:::/usr/bin/nologin
mail:x:8:12::/var/spool/mail:/usr/bin/nologin
ftp:x:14:11::/srv/ftp:/usr/bin/nologin
```

Para usar la consola como superusuario, pero sin saber la contraseña de *root* podemos usar:

 Bash

```
sudo su
```

6.5.6 *passwd*

El comando «*passwd*» es utilizado para cambiar la contraseña de un usuario. La básica del comando es la siguiente:

```
passwd [opciones] [usuario]
```

Si se ejecuta sin argumentos, se cambiará la contraseña del usuario actual. El comando pedirá al usuario que introduzca la contraseña *actual* y, a continuación, que introduzca la nueva contraseña dos veces para confirmarla.

! Observación

Las contraseñas de los usuarios en Linux están cifradas y almacenadas en un archivo protegido llamado «*/etc/shadow*».

Algunas de las opciones más comunes son:

- «-l»: Bloquea la cuenta del usuario. Esto significa que el usuario no podrá iniciar sesión hasta que la cuenta sea desbloqueada.
- «-u»: Desbloquea una cuenta que ha sido bloqueada.

- «-d»: Elimina la contraseña del usuario. Esto permite que el usuario inicie sesión sin una contraseña.
- «-e»: Hace que la contraseña del usuario expire. Esto significa que el usuario deberá cambiar su contraseña la próxima vez que inicie sesión.

6.5.7 Permisos en Linux

Los permisos Linux son utilizados para proteger los archivos y directorios del sistema de accesos no autorizado. Cada archivo y directorio es propiedad de un usuario y un grupo; y el propietario del archivo o directorio puede especificar quién tiene acceso a ellos y en qué nivel.

Existen tres tipos de permisos: de usuario (*u*), de grupo (*g*) y otros (*o*), y cada uno de ellos puede tener tres permisos posibles: lectura (*r*), escritura (*w*) y ejecución (*x*).

- El permiso de lectura (*r*) permite al usuario ver el contenido de un archivo o listar los archivos de un directorio.
- El permiso de escritura (*w*) permite al usuario modificar el contenido de un archivo o crear, eliminar y renombrar archivos dentro de un directorio.
- El permiso de ejecución (*x*) permite al usuario ejecutar un archivo o entrar en un directorio.

Ejemplo 6.109. Observemos la salida del comando `ls -l ~`:

 Salida en pantalla

```
-rwxr-xr-x 1 alumno wheel 15832 abr 12 19:33 dsh*  
-rw-r--r-- 1 alumno wheel 1130 abr 12 19:32 dsh.c
```

- Podemos observar que el archivo «dsh» tiene los permisos «rwxr-xr-x». Los primeros tres permisos se corresponden con los permisos de usuario (*u*), los siguientes con los permisos del grupo (*g*) y los últimos son los permisos de los otros (*o*) usuarios. En definitiva el usuario «alumno» tiene todos los permisos, pero los del grupo «wheel» y demás usuarios no tienen permiso de escritura.
- Para el caso del archivo «dsh.c», la única diferencia es que nadie puede ejecutarlo.

En ocasiones estos permisos se representan mediante un número de 3 dígitos en base 8, donde cada dígito representa los permisos para un grupo diferente

de usuarios: el propietario del archivo, el grupo al que pertenece el archivo y otros usuarios. Cada dígito es una suma de valores numéricos que representan los permisos para leer, escribir y ejecutar.

Los valores numéricos correspondientes a cada permiso son:

- 4 para el permiso de lectura (*r*).
- 2 para el permiso de escritura (*w*).
- 1 para el permiso de ejecución (*x*).

Por lo tanto, la suma de los valores numéricos para los distintos permisos es:

- 0: sin permisos (*---*).
- 1: permiso de ejecución (*--x*).
- 2: permiso de escritura (*-w-*).
- 1 + 2 = 3: permiso de escritura y ejecución (*-wx*).
- 4: permiso de lectura (*r--*).
- 1 + 4 = 5: permiso de lectura y ejecución (*r-x*).
- 2 + 4 = 6: permiso de lectura y escritura (*rw-*).
- 1 + 2 + 4 = 7: permiso de lectura, escritura y ejecución (*rwx*).

Ejemplo 6.110. Si vemos que un archivo tiene permisos 644, esto significa que el propietario del archivo tiene permisos de lectura y escritura (6), mientras que el grupo y otros usuarios solo tienen permiso de lectura (4).

6.5.8 *chown*

El comando «*chown*» se utiliza para cambiar el propietario y/o grupo de un archivo o directorio. La sintaxis general del comando es:

```
chown [-R] usuario[:grupo] archivo
```

«*chown*» es una abreviación de «Change Owner» (cambiar dueño).

Puede agregarse la opción «-R» para operar recursivamente sobre los directorios.

! Observación

Es importante tener en cuenta que generalmente solo el usuario *root* puede cambiar la propiedad de un archivo o directorio.

Ejemplo 6.111. El usuario *root* puede apropiarse de un «archivo» con el comando:

```
Bash
chown root archivo
```

6.5.9 *chmod*

El comando «*chmod*» se utiliza para cambiar los permisos de acceso a archivos y directorios. La sintaxis básica del comando «*chmod*» es la siguiente:

```
chmod [-R] modo archivo
```

«*chmod*» significa «cambiar modo» (del inglés: «Change Mode»).

Puede agregarse la opción «-R» para operar recursivamente sobre los directorios.

El modo puede especificarse en forma numérica o usando caracteres especiales. La sintaxis básica del modo simbólico es:

```
[ugoa] [+ -=] [rwx]
```

Donde:

- «u»: representa al usuario propietario del archivo/directorio.
- «g»: representa al grupo al que pertenece el archivo/directorio.
- «o»: representa a otros usuarios que no son el propietario ni pertenecen al grupo.
- «a»: representa a todos los usuarios.
- «+»: agrega los permisos especificados al archivo/directorio.
- «-»: elimina los permisos especificados del archivo/directorio.
- «=»: establece los permisos especificados y elimina cualquier otro permiso que no sea el especificado.

Ejemplo 6.112. Para agregar permiso de *lectura* al usuario propietario de «archivo» podemos usar el comando:

```
Bash
chmod u+r archivo
```

Ejemplo 6.113. Para eliminar permiso de *escritura* al grupo al que pertenece «archivo» podemos usar el comando:

```
Bash
chmod g-w archivo
```

Ejemplo 6.114. Para establecer permiso de *ejecución* a otros usuarios que no son propietarios de «archivo» ni pertenecen al grupo, podemos usar el comando:

```
Bash
chmod o=x archivo
```

Ejemplo 6.115. Para darles todos los permisos a todos los usuarios sobre «archivo» podemos escribir:

```
Bash
chmod =777 archivo
```

6.5.10 *useradd*

El comando «useradd» es utilizado para crear una nueva cuenta de usuario en el sistema. A continuación se explican en detalle los aspectos más importantes del comando.

Sintaxis básica:

```
useradd [opciones] usuario
```

Algunas de las opciones más comunes son:

- «-m»: Crea automáticamente el directorio personal del usuario en «/home/usuario».
- «-g GRUPO»: Setea el *GRUPO* al que pertenece el usuario.

! Observación

Es importante destacar que para crear una cuenta de usuario con «useradd», es necesario tener privilegios de superusuario en el sistema.

6.5.11 userdel

El comando «userdel» se utiliza para eliminar una cuenta de usuario existente. La sintaxis básica del comando "userdel" es la siguiente:

```
userdel [opciones] usuario
```

Algunas de las opciones más comunes son:

- «-r»: esta opción se utiliza para eliminar la cuenta de usuario y sus archivos personales.
- «-f»: esta opción se utiliza para forzar la eliminación de la cuenta de usuario, incluso si está actualmente en uso o si tiene procesos en ejecución.

! Observación

Al igual que «useradd», para usar «userdel» es necesario tener privilegios de superusuario en el sistema.

6.5.12 *Ejercicios*

6.6 PROCESOS Y TAREAS

6.6.1 *ps*6.6.2 *jobs*6.6.3 *bg*6.6.4 *fg*6.6.5 *Señales*6.6.6 *kill*6.6.7 *Ejercicios*

6.7 GESTIÓN

6.7.1 *apt*6.7.2 *chsh*6.7.3 *free*6.7.4 *loadkeys*6.7.5 *lsof*6.7.6 *reboot*6.7.7 *setxkbmap*6.7.8 *startx*6.7.9 *top*6.7.10 *uname*6.7.11 *uptime*6.7.12 *which*6.7.13 *Ejercicios*

6.8 INTERNET

6.8.1 *curl*6.8.2 *ping*6.8.3 *scp*6.8.4 *ssh*

SHELL SCRIPTING

7.1 COMANDOS

7.1.1 *exit*

7.1.2 *export*

7.1.3 *read*

7.1.4 *seq*

7.1.5 *shift*

7.1.6 *source*

7.1.7 *test*

7.1.8 *trap*

7.2 CONTROL DE FLUJO

7.2.1 *if / else*

7.2.2 *for*

7.2.3 *while / until*

7.2.4 *case*

7.2.5 *select*

7.2.6 *Funciones*

Parte III

CONCEPTOS ADICIONALES

CONTROL DE VERSIONES

CONTENEDORES

9.1 EMULACIÓN

9.2 VIRTUALIZACIÓN

9.3 CONTENEDORES