# Entornos de Programación (IA1.3)

Tecnicatura Universitaria en Inteligencia Artificial

Branco Blunda

2025

# Índice

# Parte I

# Fundamentos

## 1. Historia

### 1.1. Orígenes de la computadora

- **Pascalina (1642):** Desarrollada por Blaise Pascal. Una de las primeras calculadoras mecánicas, capaz de sumar usando engranajes. Sentó bases para la automatización y máquinas calculadoras modernas.
- **Máquina Analítica (1837):** Proyecto de Charles Babbage, considerado el primer prototipo de computadora moderna. Podía almacenar programas y realizar operaciones matemáticas complejas. Aunque no la completó, sentó bases para el procesamiento automático de datos.
- **Máquina Diferencial:** Otro proyecto de Babbage.
- **Telégrafo (1837):** Inventado por Samuel Morse. Primer sistema de comunicación eléctrica a larga distancia, usando codificación de letras/números en señales eléctricas.
- **Teletipo (TTY):** Utilizaba impresión automática para transmitir texto por red telegráfica (década de 1930). También se usaron como terminales de computadora.

### 1.2. Avances de la guerra

- **Z3 (1941):** Construida por Konrad Zuse. Máquina electromecánica basada en relés, usaba sistema binario y tenía memoria programable.
- **Harvard Mark I (1944):** Computadora electromecánica de IBM, diseñada por Howard Aiken, patrocinada por EEUU.
- **Tarjetas perforadas:** Medio de almacenamiento histórico, codificaban información mediante perforaciones en cartulinas.
- **ENIAC (1946):** Computadora electrónica programable (EEUU). Usaba válvulas electrónicas para cálculos numéricos complejos a alta velocidad. Se programaba mediante *plugboards* (paneles de conexiones).
- **Transistor (1947):** Invención clave que superó a las válvulas y relés, permitiendo un gran salto en el poder de cómputo.

### 1.3. La informática moderna

- **FORTRAN (1957):** Uno de los primeros lenguajes de programación (IBM, John Backus). Diseñado para científicos y matemáticos.
- **PDP-1 (1959):** De Digital Equipment Corporation. Ordenador de tiempo compartido, permitía acceso a múltiples usuarios simultáneamente.
- **BASIC (1964):** Lenguaje de programación diseñado para ser fácil de aprender y usar por principiantes.
- **NLS (oN-Line System) (1968):** Sistema de Douglas Engelbart. Demostró tempranamente GUI, ratón, hipertexto, ventanas, procesamiento de texto, email, videoconferencia, colaboración en tiempo real.

- **MULTICS (1969):** Sistema operativo de tiempo compartido (MIT, Bell Labs). Ofrecía servicios avanzados (archivos, BBDD, permisos). Tuvo problemas técnicos y financieros.
- **C (1971):** Lenguaje desarrollado por Dennis Ritchie. Eficiente, portable.
- **Unix (1972):** Sistema operativo creado por Ken Thompson y Dennis Ritchie (reescritura de UNICS en C). Portable y modificable, dio origen a muchas variantes.

## 1.4. Primeras computadoras personales

- **Xerox Alto (1973):** Ordenador personal temprano con GUI, ventanas, mouse, gestión de archivos, procesador de texto, red. No comercializado, pero influyente.
- **Altair 8800 (1974):** Una de las primeras PC comercializadas (en kit). Éxito de ventas, sentó bases para la PC moderna.
- **BSD (Berkeley Software Distribution) (1977):** Derivado de Unix, desarrollado en la Universidad de Berkeley gracias a que Bell Labs permitió a las universidades usar y adaptar el código fuente de Unix.
- **IBM PC (1981):** Computadora muy importante que estableció el estándar de PC. Su arquitectura abierta permitió un gran ecosistema de hardware y software de terceros.
- **System V (1983):** Versión comercial de Unix desarrollada por Bell Labs. Tuvo gran impacto, pero no era "libre"debido a restricciones de AT&T.
- **Movimiento GNU (1983):** Fundado por Richard Stallman. Objetivo: desarrollar un sistema operativo completo, gratuito y libre basado en Unix.

## 1.5. La era de la información

- **Software Privativo (Contexto):** Movimiento iniciado por Bill Gates en 1976 (carta abierta a los aficionados).
- **X Windows (1984):** Sistema de ventanas para sistemas Unix (MIT). Proporcionaba interfaz gráfica usable en diversos sistemas.
- **Minix (1987):** Sistema operativo educativo (Andrew Tanenbaum). Similar a Unix pero reducido, para enseñar diseño de SO.
- **World Wide Web (1989):** Propuesta por Tim Berners-Lee. Expansión de Internet en los 90, adopción de navegadores, servicios en línea (email, buscadores, e-commerce).
- **Linux (Kernel) (1991):** Desarrollado por Linus Torvalds como proyecto personal (basado en Minix). Se convirtió en el kernel faltante para el proyecto GNU, formando el sistema operativo GNU/Linux, muy popular y de código abierto.
- **Python (1991):** Lenguaje de alto nivel creado por Guido van Rossum. Enfocado en legibilidad y facilidad de uso. Popular por su sintaxis clara, versatilidad y amplias aplicaciones.
- **Windows 95 (1995):** SO de Microsoft con interfaz gráfica intuitiva (Botón Inicio, barra de tareas), Plug and Play, Internet Explorer. Marcó un hito.
- **Deep Blue (1997):** Supercomputadora de IBM que derrotó al campeón mundial de ajedrez Garry Kasparov, hito en IA.
- **Google (1998):** Motor de búsqueda fundado por Larry Page y Sergey Brin.
- **Wikipedia (2001):** Enciclopedia en línea, gratuita y colaborativa.

- **Ubuntu (2004):** Distribución de Linux basada en Debian (Canonical Ltd.). Popular por ser amigable y accesible.
- **YouTube (2005):** Plataforma de compartición de videos en línea.
- **Android (2005):** Sistema operativo móvil (adquirido por Google). Dominante en dispositivos móviles, también usado en tablets, TVs, etc.
- **GPT-3 (2020):** Modelo de lenguaje grande de OpenAI. Capaz de generar texto coherente y realizar diversas tareas de lenguaje natural.

# 2. Arquitectura de la Computadora

## 2.1. Gabinete

Carcasa que protege los componentes internos. Funciones:

- **Protección:** Contra daños físicos, polvo, etc.
- **Organización:** Mantiene componentes en su lugar.
- **Refrigeración:** Facilita circulación de aire, puede tener ventiladores integrados.

## 2.2. Placa madre

Pieza central que conecta y comunica todos los componentes esenciales (CPU, RAM, almacenamiento, tarjeta gráfica, etc.).

- Distribuye energía eléctrica.
- Controla puertos de E/S (USB, audio, red).
- Incluye la **BIOS** (en chip ROM) para configuración inicial y pruebas de hardware (POST).
- Puede tener chip de audio integrado.

## 2.3. Fuente de alimentación (PSU - Power Supply Unit)

Convierte la corriente eléctrica de la toma en voltajes necesarios para los componentes internos. Protege contra sobretensiones y cortocircuitos.

## 2.4. Microprocesador (CPU - Central Processing Unit)

Componente principal (çerebro"). Chip integrado en la placa madre (zócalo), con múltiples núcleos (cores).

- Procesa y ejecuta programas.
- Maneja E/S de datos.
- Controla otros componentes (RAM, disco duro).
- Su velocidad y capacidad determinan el rendimiento general.
- **iGPU (Integrated Graphics Processing Unit):** Algunos CPU modernos incluyen gráficos integrados para tareas básicas.

### 2.5. Memoria RAM (Random Access Memory)

Memoria de acceso aleatorio usada para almacenar *temporalmente* datos y programas en uso activo.

- Clave para el rendimiento (acceso rápido).
- **Volátil:** Pierde su contenido al apagar la computadora.
- El CPU también tiene memoria volátil más rápida y pequeña llamada **registros**.

### 2.6. Memoria secundaria

Almacenamiento *persistente* (no volátil).

- **Disco Duro (HDD - Hard Disk Drive):** Almacenamiento magnético con platos giratorios y cabezales de lectura/escritura. Más lento que la RAM debido al movimiento físico necesario.
- **SSD (Solid State Drive):** Usa memoria flash, sin partes móviles. Más rápido en lectura/escritura y tiempo de acceso que los HDD. (Ej: NVMe es un tipo de SSD muy rápido).
- **Jerarquía de memoria:** Concepto que organiza la memoria en niveles según velocidad y coste (Registros ¿Caché L1/L2/L3 ¿RAM ¿SSD/HDD ¿Almacenamiento externo/red).

### 2.7. Placa de video (GPU - Graphics Processing Unit)

Componente de hardware para procesar y generar imágenes en pantalla. Libera a la CPU del procesamiento gráfico.

- Esencial para juegos, diseño gráfico, etc.
- En IA, se usa para entrenar y ejecutar redes neuronales por su arquitectura paralela.

### 2.8. Teclado

Dispositivo de entrada para introducir texto, números y comandos.

- **Teclas especiales:** No introducen caracteres, realizan funciones (Inicio, Fin, Re Pág, Av Pág, Ctrl, Alt, Shift, etc.).
- **Combinaciones especiales:** Ej. Ctrl+Inicio (principio doc), Ctrl+Fin (fin doc), Ctrl+Flechas (mover por palabras).
- **Distribuciones:** Disposición de teclas (mapa), varía según idioma y preferencia (Ej: QWERTY, Dvorak, Colemak, Latinoamericana, Estadounidense).

## 3. Representación de la Información

### 3.1. Unidades de información

Basadas en el sistema binario (0s y 1s).

- **Bit (b):** Unidad más pequeña (0 o 1).
- **Byte (B):** Conjunto de 8 bits. Unidad básica.
- **Conversión:** 1 B = 8 b. 1,000,000$b$ = 125,000$B$. 1,024$B$ = 8,192$b$.
- Con $n$ bits se pueden formar $2^n$ valores diferentes (Ej: 32 bits -¿$2^{32}$ valores).

- **Prefijos Decimales (base 10):** Usados comercialmente.
  - Kilobyte (kB) = $10^3$ B = 1,000 B
  - Megabyte (MB) = $10^6$ B = 1,000 kB
  - Gigabyte (GB) = $10^9$ B = 1,000 MB
  - Terabyte (TB) = $10^{12}$ B = 1,000 GB
  - Petabyte (PB) = $10^{15}$ B = 1,000 TB
- **Prefijos Binarios (base 2):** Usados técnicamente (IEC).
  - Kibibyte (KiB) = $2^{10}$ B = 1,024 B
  - Mebibyte (MiB) = $2^{20}$ B = 1,024 KiB
  - Gibibyte (GiB) = $2^{30}$ B = 1,024 MiB
  - Tebibyte (TiB) = $2^{40}$ B = 1,024 GiB
  - Pebibyte (PiB) = $2^{50}$ B = 1,024 TiB
- **Diferencia:** Un disco de 500 GB (decimal) tiene $\approx$ 465,66 **GiB (binario).** $500 \times 10^9/2^{30} \approx 465{,}66$**.**

### 3.2. Sistemas de numeración

Conjuntos de reglas y símbolos para representar números.

- **No posicionales:** El valor del símbolo no depende de su posición (Ej: Unario '———'=3, Romano 'XIX'=19).
- **Posicionales:** El valor del dígito depende de su valor intrínseco y su posición (base).
  - **Base (b):** Determina cuántos símbolos se usan.
  - **Fórmula general (Entero N, n dígitos):** $N = d_{n-1} \cdot b^{n-1} + d_{n-2} \cdot b^{n-2} + \cdots + d_1 \cdot b^1 + d_0 \cdot b^0$
  - **Con parte decimal:** Se añaden potencias negativas $d_{-1} \cdot b^{-1} + d_{-2} \cdot b^{-2} + \ldots$
  - **Notación:** Se usa subíndice para indicar base no decimal, ej. $(1011,01)_2$.
  - **Decimal (base 10):** Dígitos 0-9. $345,67 = 3 \cdot 10^2 + 4 \cdot 10^1 + 5 \cdot 10^0 + 6 \cdot 10^{-1} + 7 \cdot 10^{-2}$.
  - **Binario (base 2):** Símbolos 0, 1. $(1011,01)_2 = 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 + 0 \cdot 2^{-1} + 1 \cdot 2^{-2} = 8 + 0 + 2 + 1 + 0 + 0,25 = 11,25_{10}$.
  - **Octal (base 8):** Símbolos 0-7. Fácil conversión con binario agrupando de 3 bits: $(101\ 010\ 110)_2 = (526)_8$.
  - **Hexadecimal (base 16):** Símbolos 0-9, A-F (A=10, B=11,..., F=15). $(1A)_{16} = 1 \cdot 16^1 + A \cdot 16^0 = 16 + 10 = 26_{10}$.

### 3.3. Codificación de caracteres

Sistemas que asignan valor numérico (código) a cada carácter para representación digital.

- Incluyen caracteres de control (no visibles) para formato, estructura, etc.
- **ASCII (American Standard Code for Information Interchange):**

- Una de las primeras codificaciones. 7 bits (0-127).
- Representa letras inglesas, números, símbolos.
- Caracteres no imprimibles (control): Nulo (0), Inicio texto (2), Fin texto (3), Tab (9), Nueva línea (10), Retorno carro (13), etc.
- Caracteres imprimibles: Espacio (32), '!' (33), ..., '0' (48), '1' (49), ..., 'A' (65), 'B' (66), ..., 'a' (97), etc.
- Palabra "HOLA": H(72) O(79) L(76) A(65) -¿Binario: 1001000 1001111 1001100 1000001.
- *Observación:* Código '0' (48) es diferente del carácter nulo (0).

- **ASCII Extendido:**
  - Usa 8 bits (0-255), añade 128 caracteres.
  - Primeros 128 (0-127) son idénticos a ASCII estándar (compatibilidad).
  - Cubre necesidades de otros idiomas (acentos, símbolos).
  - **No existe un único estándar.** Variantes comunes:
    - **Latin-1 (ISO-8859-1):** Para idiomas de Europa Occidental. Incluye acentos, etc.
    - **Windows-1252 (CP-1252):** Similar a Latin-1, usada en Windows, con algunos caracteres adicionales. Ejemplo: «10 €» en CP-1252 usa 128 para '€'. En Latin-1, 128 no es imprimible ('€' no existe).
  - *Observación:* Leer con codificación incorrecta produce resultados inesperados.
- **Unicode:**
  - Estándar universal para representar caracteres de *todos* los idiomas.
  - Soluciona problema de múltiples codificaciones incompatibles.
  - Define un conjunto de caracteres enorme (code points).
- **UTF (Unicode Transformation Format):**
  - Formatos para codificar code points Unicode en secuencias de bytes.
  - **UTF-32:** Siempre 4 bytes por carácter. Simple pero ineficiente.
  - **UTF-8:** Codificación de longitud variable (1 a 4 bytes). Eficiente para caracteres comunes (ASCII usa 1 byte). Compatible con ASCII. El más usado en la web y sistemas modernos.
- **Secuencias de escape:** Combinaciones (ej. '+ carácter) para representar símbolos no escribibles directamente o caracteres de control (ej. " nueva línea, '͡ tabulación, " nulo, ' ' barra invertida).

# 4. Conceptos de Programación

## 4.1. Introducción

**Programación:** Proceso de diseñar, escribir, probar y mantener código informático para crear software, aplicaciones y sistemas. Implica definir un problema, especificar requisitos, escribir

código en un lenguaje, probarlo y documentarlo.

## 4.2. Lenguaje de programación

Conjunto de reglas, símbolos y palabras clave para comunicar instrucciones a una computadora. Permiten describir acciones (cálculos, interacción con periféricos, E/S de datos). Varían en complejidad y enfoque. **Observación:** En esta materia se usa **bash**, en la carrera se aprende **python**.

## 4.3. Niveles de lenguajes

- **Alto Nivel:** Cercanos al lenguaje humano, abstraen detalles del hardware. Más fáciles y rápidos para programas complejos. Se encargan de gestión de memoria, errores. Ej: Python, Java, C++, Ruby, PHP.

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

Listing 1: Código Alto Nivel (Python)

- **Bajo Nivel:** Cercanos al lenguaje máquina, interactúan directamente con hardware. Requieren conocimiento detallado del hardware. Ej: Ensamblador, C.

```
.global _start
.text
mensaje: .ascii "Hola mundo!\n"
_start:
    mov $1, %rax           # syscall number (write)
    mov $1, %rdi           # file descriptor 1 (stdout)
    lea mensaje(%rip), %rsi # message address
    mov $12, %rdx          # message length
    syscall                # invoke kernel

    mov $60, %rax           # syscall number (exit)
    mov $0, %rdi           # exit code 0
    syscall                # invoke kernel
```

Listing 2: Código Bajo Nivel (Ensamblador x86-64 Linux)

- **Binario:** Lenguaje máquina (ceros y unos). Instrucciones directas para el procesador. Difícil y tedioso escribir directamente.

```
48 6f 6c 61 20 6d 75 6e 64 6f 21 0a  # "Hola mundo!\n" en ASCII Hex
48 c7 c0 01 00 00 00 # mov $1, %rax
48 c7 c7 01 00 00 00 # mov $1, %rdi
...
```

Listing 3: Código Binario (Fragmento Representativo)

## 4.4. Compiladores e intérpretes

Programas que convierten código fuente en instrucciones ejecutables.

- **Intérprete:** Lee, traduce y ejecuta el código fuente línea por línea, en tiempo real. Más lento en ejecución, pero permite ejecutar código sin paso previo de compilación. (Ej: Python, Bash).

- **Compilador:** Traduce todo el código fuente a lenguaje máquina de una vez, generando un archivo ejecutable. Ejecución más rápida. Proceso de compilación puede llevar tiempo. (Ej: C, C++).
- **Tiempo de Traducción vs Ejecución:**
  - Compilado: Traducción hecha por el desarrollador una vez. Ejecución rápida para el usuario final.
  - Interpretado: Traducción hecha por el usuario final cada vez que ejecuta. Ejecución más lenta.

### 4.5. Otros conceptos

- **Código fuente:** Conjunto de instrucciones escritas por un programador en un lenguaje legible por humanos.
- **Código abierto:** Software cuyo código fuente está disponible públicamente y puede ser modificado por cualquiera. Fomenta colaboración.
- **Software libre:** Software distribuido con una licencia que garantiza 4 libertades esenciales (usar, estudiar, compartir, modificar) sin pagar regalías. Puede ser distribuido comercialmente, no necesariamente gratuito. (Definido por la Free Software Foundation).
- **IDE (Integrated Development Environment):** Conjunto de herramientas en una aplicación para facilitar el desarrollo: editor de código (resaltado sintaxis, autocompletado), depurador (identificar/corregir errores), herramientas de compilación/construcción. (Ej: Visual Studio Code).

## 5. Sistemas Operativos

### 5.1. Descripción

Conjunto de programas y herramientas que controlan y coordinan las actividades de una computadora, permitiendo interacción usuario-hardware/software. Compuesto por un **núcleo (kernel)** (control total del hardware) y programas utilitarios.

### 5.2. Núcleo (Kernel)

Parte central y fundamental del SO.

- Controla acceso a recursos hardware.
- Gestiona procesos, memoria, E/S.
- Proporciona interfaz para que aplicaciones interactúen con hardware.
- Se ejecuta en **modo privilegiado** (acceso directo a hardware).
- Las aplicaciones de usuario piden permiso al kernel mediante **llamadas a sistema** para realizar tareas que requieren privilegios. La CPU detiene la app, ejecuta el código del kernel y luego reanuda la app.

**Funcionalidades Principales del SO Moderno:**

- **Abstracción de hardware:** Oculta detalles del hardware a las aplicaciones.
- **Gestión del hardware:** Controla y asigna recursos (CPU, memoria, disco).
- **Interfaz de usuario (UI):** Permite interacción usuario-computadora.

- **Gestión de archivos:** Permite crear, modificar, copiar, eliminar archivos/carpetas.
- **Multitarea:** Permite ejecutar varias aplicaciones .ªl mismo tiempo"(concurrencia).
- **Multiusuario:** Permite que varios usuarios usen el sistema simultáneamente.

### 5.3. Proceso de arranque

Pasos al encender una PC:

1. **Encendido:** Se activa la fuente de alimentación.
2. **POST (Power On Self Test):** La placa madre realiza autodiagnóstico del hardware.
3. **Boot Loader (Cargador de arranque):** Programa cargado por la placa madre. Su objetivo es cargar el núcleo del SO. Ej: GRUB (Linux), bootmgr (Windows).
4. **Núcleo:** Toma control, identifica hardware, carga controladores, monta sistema de archivos. Inicia el primer programa de usuario.
5. **Init (o systemd):** Primer programa de usuario en Linux. Carga scripts de arranque, inicia servicios, provee entorno gráfico o línea de comandos. (*systemd* reemplaza a init en muchas distros modernas).

### 5.4. Interfaz de usuario (UI)

Punto de interacción humano-programa. Tipos:

- **GUI (Graphical User Interface):** Interacción mediante elementos gráficos (ventanas, iconos, botones). Intuitiva, fácil de aprender. (Ej: KDE Plasma, Fluent Design de Windows). *Nota: En Linux, la GUI es opcional.*
- **CLI (Command Line Interface):** Interacción basada en texto, escribiendo comandos. Requiere conocimiento técnico, consume menos recursos. *Esta materia se enfoca en CLI.*
- **TUI (Text-based User Interface):** Híbrida. Entorno de texto pero con elementos visuales (menús, cuadros). (Ej: htop).

### 5.5. Terminal

Conceptos relacionados con la interacción basada en texto:

- **Terminal (física):** Históricamente, dispositivos físicos (teclado+monitor) para E/S de texto. Hoy, se refiere al conjunto teclado/monitor.
- **Consola virtual:** Aplicación del kernel que simula una terminal TTY. Accesible en sistemas Unix (Ctrl+Alt+F1, F2...).
- **Emulador de terminal:** Programa de usuario en entorno gráfico que simula una terminal. Permite interactuar con el shell. (Ej: Qterminal, GNOME Terminal). *Nota: Copiar/-pegar suele ser con Ctrl+Shift+C / Ctrl+Shift+V.*
- **Shell (Intérprete de línea de comandos):** Programa que interpreta los comandos del usuario y los ejecuta. (Ej: bash, sh, zsh, fish, cmd, PowerShell).

### 5.6. Procesos

Instancia de un programa en ejecución. Ciclo de vida (estados):

- **Creación:** El SO crea el proceso y asigna recursos (memoria, ID).

- **Listo:** Cargado en memoria, esperando ser seleccionado por el planificador para ejecutarse.
- **Ejecución:** El procesador ejecuta las instrucciones del proceso.
- **Bloqueado:** Esperando por un evento (E/S, red). Libera el procesador.
- **Finalización:** Ha terminado su ejecución (normal o forzada). Libera recursos.

### 5.7. Distribuciones

Sistemas operativos basados en el kernel de Linux, combinados con software libre/código abierto (aplicaciones, herramientas).

- Ejemplos: Debian, Ubuntu, Red Hat, Arch Linux.
- Heredan el concepto de versiones de Unix (System V, BSD).
- Existen por la naturaleza abierta del software, permitiendo personalización para diferentes necesidades (servidores, escritorio, programadores).

### 5.8. Proceso de apagado

Pasos controlados para finalizar la ejecución del SO:

- Cierre de programas y servicios.
- Guardado de datos pendientes.
- Desmontaje seguro de dispositivos de almacenamiento.
- Señal al hardware para cortar alimentación.

**Opciones comunes:**

- **Apagado:** Cierre total, requiere arranque completo al encender.
- **Reinicio:** Cierre total seguido de arranque automático. Útil para actualizaciones o solucionar problemas.
- **Hibernación:** Guarda estado del sistema (RAM) en disco duro y apaga. Restaura estado exacto al encender. *Requiere partición/archivo dedicado.*
- **Suspensión:** Estado de bajo consumo, mantiene RAM alimentada. Reanudación rápida. *Pierde estado si hay corte de energía.*

## 6. Sistema de Archivos

Estructura lógica para organizar, almacenar y recuperar datos en un medio de almacenamiento. Sin él, los datos serían un bloque informe.

### 6.1. Acceso aleatorio y secuencial

- **Acceso Aleatorio:** Permite acceso directo a cualquier ubicación (ej. SSD, USB). Rápido, ideal para programas.
- **Acceso Secuencial:** Requiere leer datos en orden desde el principio (ej. HDD, DVD, cintas magnéticas). Más lento, útil para grandes volúmenes o respaldos.
- *Observación:* Arquitecturas modernas suelen usar ambos tipos (RAM/SSD aleatorio, HDD/Cinta secuencial para respaldo).

## 6.2. Descripción

Funciones principales:

- **Gestión del espacio:** Asigna/administra espacio en disco para archivos, controla espacio libre.
- **Organización:** Estructura jerárquica (directorios/subdirectorios) para búsqueda y recuperación.
- **Permisos:** Medidas de seguridad (permisos de acceso, encriptación).
- **Metadatos:** Información sobre archivos/directorios (nombre, tamaño, fechas, permisos).
- **Integridad (Journaling):** Técnica que registra cambios en un diario *antes* de aplicarlos. Permite recuperar estado anterior en caso de fallo, evitando corrupción. (Ej: ext4 en Linux, NTFS en Windows).

## 6.3. Particiones

Sección lógica de un disco tratada como unidad separada.

- Se crean dividiendo el espacio del disco.
- Cada partición tiene su propio sistema de archivos.
- Usos: separar SO de datos de usuario, instalar múltiples SO, áreas para propósitos específicos.

## 6.4. Estructura de directorios

- **Directorio (Carpeta):** Tipo especial de archivo que contiene referencias a otros archivos y directorios.
- **Árbol de directorios:** Jerarquía que organiza el sistema de archivos.
- **Directorio Raíz:** Nivel superior del árbol. En Linux: '/'. En Windows: 'C:', 'D:', etc. (cada partición tiene su raíz).
- **Rutas de acceso:** Combinación de nombres de directorios/archivos para ubicar un elemento.
  - **Ruta Absoluta:** Especifica ubicación completa desde la raíz (Ej Linux: '/usr/bin/bash'). Siempre empieza con '/'.
  - **Ruta Relativa:** Especifica ubicación desde el directorio de trabajo actual (Ej: si estoy en '/usr', la ruta relativa a bash es 'bin/bash'). No empieza con '/'.
- **Administrador de archivos:** Programa GUI para navegar la estructura (Ej: PCManFM, Explorador de Windows).

## 6.5. El sistema de archivos de Linux

- **Esquema de particionado (Común):**
  1. Partición para gestor de arranque y núcleo (ej. montada en '/boot').
  2. Partición principal para SO, programas, documentos (montada en '/').
  3. Partición de intercambio ('swap'): Usada como extensión de RAM cuando esta se agota.

4. *Observación:* Esquemas antiguos podían instalar gestor en primer sector del disco (MBR).

- **Nomenclatura de dispositivos:**
  - Identificador de tipo: 'sd' para discos SATA/SCSI/USB.
  - Letra secuencial: 'a' (primer disco), 'b' (segundo), etc. -¿'sda', 'sdb'.
  - Número de partición: '1' (primera), '2' (segunda), etc. -¿'sda1', 'sda2'.
  - *Observación:* 'sda' se refiere a todo el dispositivo, 'sda1' a la primera partición.
- **Jerarquía de archivos (FHS - Filesystem Hierarchy Standard):** Estructura estándar. Directorios importantes:
  - '/': Raíz. Todo cuelga de aquí.
  - '/bin': Comandos binarios esenciales de usuario.
  - '/boot': Archivos necesarios para el arranque (kernel, gestor).
  - '/dev': Archivos especiales que representan dispositivos hardware.
  - '/etc': Archivos de configuración del sistema y aplicaciones.
  - '/home': Directorios personales de los usuarios.
  - '/lib': Bibliotecas compartidas esenciales.
  - '/mnt', '/media': Puntos de montaje temporales para sistemas de archivos externos (USB, CD).
  - '/proc': Directorio virtual con información del sistema y procesos (generado por kernel).
  - '/root': Directorio personal del usuario 'root'.
  - '/sbin': Comandos binarios esenciales del sistema (administración).
  - '/tmp': Archivos temporales.
  - '/usr': Utilidades y aplicaciones multiusuario (compartidas, sólo lectura). Contiene subdirectorios como '/usr/bin', '/usr/lib'.
  - '/var': Archivos variables (logs, bases de datos, correo).
  - '/opt': Software adicional .ºpcional", fuera de la distribución principal.

  *Observación: Linux es sensible a mayúsculas/minúsculas ('archivo' != 'Archivo'). Archivos/carpetas que empiezan con '.' son ocultos.*
- **Carpetas especiales (Interpretadas por el Shell):**
  - '.': Directorio actual.
  - '..': Directorio padre.
  - '~': Directorio principal (home) del usuario actual. (El shell reemplaza '~' por la ruta absoluta).
- **Montaje:** Hacer accesible un sistema de archivos (de una partición o dispositivo) en un directorio específico (punto de montaje) dentro de la jerarquía raíz '/'.
- **Enlaces:** Apuntadores a otros archivos/directorios.

- **Enlace Suave (Simbólico, Soft Link):** Archivo especial que contiene la *ruta* al archivo/directorio original. Si el original se borra, el enlace queda roto". Similar a accesos directos en Windows. Se crean con 'ln -s'.
- **Enlace Duro (Físico, Hard Link):** Entrada de directorio adicional que apunta al *mismo* inodo (datos) del archivo original. No son archivos separados. El archivo original no se borra hasta que se eliminan todos sus enlaces duros. No pueden apuntar a directorios ni a archivos en diferentes sistemas de archivos. Se crean con 'ln' (sin '-s'). *Observación: No se pueden crear enlaces duros a directorios.*

# Parte II

# Manejo de bash

## 7. Comandos Básicos de Bash

### 7.1. Introducción

**Bash:** Intérprete de línea de comandos y lenguaje de scripting, shell por defecto en muchas distros Linux. Permite interactuar con el SO, ejecutar programas, gestionar archivos, automatizar tareas.

**Sintaxis básica de un comando:**

```
comando [opciones] [argumentos]
```

- **comando:** Nombre del programa/acción.
- **opciones:** Modifican comportamiento. Suelen empezar con '-' (cortas) o '–' (largas).
- **argumentos:** Valores sobre los que actúa el comando (nombres de archivo, rutas, etc.).
- *Nota:* '[ ]' en la sintaxis indica elemento opcional.

Ejecución: Escribir comando en terminal y presionar `Enter`. Usar 'ṗara dividir comandos largos en varias líneas.

**Prompt:** Indicador del shell esperando entrada (ej. 'usuario@equipo:ruta').**Historial:***Bashguardacomandosej*

`Flecha Arriba/Abajo`: Navegar historial.

`Ctrl+R`: Búsqueda reversa.

'!¡: Repetir último comando.

'!n': Repetir comando número 'n' del historial.

**Autocompletado:** Presionar `Tab` para completar comandos/rutas.

#### 7.1.1. `man`

Muestra el manual de otros comandos.

```
man [opciones] comando
```

- Opciones: '-f' (descripción corta), '-k' (buscar páginas que contienen palabra clave).

- Navegación: Flechas, Espacio, ‘/‘ (buscar), ‘q‘ (salir).

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

Listing 4: Ejemplo man

#### 7.1.2. `clear`

Limpia la pantalla del terminal.

```
clear [-x]
```

Opción ‘-x‘ evita borrar historial de desplazamiento del emulador. Atajo: `Ctrl+L`.

#### 7.1.3. `echo`

Imprime texto en pantalla.

```
echo [opciones] texto
```

- Opciones: ‘-n‘ (no añadir nueva línea al final), ‘-e‘ (habilitar interpretación de secuencias de escape como “, “).

```
echo Hola mundo!
echo -e "Hola\nmundo"
```

Listing 5: Ejemplo echo

#### 7.1.4. `history`

Muestra la lista de comandos ejecutados anteriormente.

```
history [-c]
```

Opción ‘-c‘ borra el historial. *Nota: ‘history‘ es una funcionalidad del shell, no un programa externo.*

### 7.2. Sistema de archivos

#### 7.2.1. `pwd`

Muestra la ruta completa del directorio de trabajo actual (Print Working Directory).

```
pwd [-P]
```

Opción ‘-P‘ muestra la ruta física (sin resolver enlaces simbólicos).

```
pwd
```

Listing 6: Ejemplo pwd

Salida ej: ‘/home/entorno‘

### 7.2.2. `cd`

Cambia el directorio de trabajo actual (Change Directory).

```
cd [ruta]
```

- ‘cd‘ (sin ruta): va al directorio home (‘ ‘).
- ‘cd -‘: vuelve al directorio anterior.
- ‘cd /‘: va al directorio raíz.
- ‘cd ..‘: va al directorio padre.
- *Nota: ‘cd‘ es un comando interno del shell.*

```
1 cd /
2 cd ~/documentos
3 cd ..
```

Listing 7: Ejemplo cd

### 7.2.3. `ls`

Lista archivos y directorios.

```
ls [opciones] [ruta]
```

Si se omite ruta, lista directorio actual. Opciones comunes:

- ‘-l‘: Formato largo (permisos, enlaces, dueño, grupo, tamaño, fecha, nombre).
- ‘-a‘: Muestra todos (incluyendo ocultos que empiezan con ‘.‘).
- ‘-h‘: Tamaño legible por humanos (KB, MB).

```
1 ls
2 ls -a
3 ls -lh /
```

Listing 8: Ejemplo ls

### 7.2.4. `tree`

Muestra estructura de directorios en forma de árbol.

```
tree [opciones] [directorio]
```

Opciones: ‘-d‘ (solo directorios), ‘-L nivel‘ (limitar profundidad), ‘-a‘ (todos), ‘-f‘ (ruta completa).

```
1 tree -L 1 /
```

Listing 9: Ejemplo tree

### 7.2.5. `mkdir`

Crea nuevos directorios (Make Directory).

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

Opciones: ‘-p‘ (crea directorios padres si no existen), ‘-v‘ (muestra detalles).

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

Listing 10: Ejemplo mkdir

#### 7.2.6. `rmdir`

Elimina directorios **vacíos** (Remove Directory).

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

Opciones: '-p' (elimina directorio y padres si quedan vacíos), '-v' (detalles).

#### 7.2.7. `touch`

Crea archivos vacíos o actualiza fecha/hora de acceso/modificación de archivos existentes.

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

Opciones: '-a' (actualiza solo hora acceso), '-m' (actualiza solo hora modificación).

```
touch archivo1 archivo2
```

Listing 11: Ejemplo touch

#### 7.2.8. `rm`

Elimina archivos y directorios (Remove). **Precaución: Borrado permanente.**

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

Opciones:

- '-r': Elimina directorios y su contenido recursivamente. **Muy peligroso si se usa mal.**
- '-f': Forzar eliminación sin pedir confirmación, ignora archivos inexistentes.
- '-i': Pide confirmación antes de borrar cada archivo.

```
rm -i archivo1
rm -r directorio_con_contenido
rm -rf / # !!! NO EJECUTAR ESTO !!!
```

Listing 12: Ejemplo rm

#### 7.2.9. `cp`

Copia archivos y directorios (Copy).

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

**Precaución:** Sobrescribe destino si existe sin preguntar (a menos que se use '-i'). Opciones:

- '-r': Copia directorios recursivamente.
- '-v': Muestra detalles.
- '-i': Pide confirmación antes de sobrescribir.

```
cp archivo.txt copia.txt
cp archivo.txt directorio/
cp -r directorio1 directorio2
```

Listing 13: Ejemplo cp

#### 7.2.10. `mv`

Mueve o renombra archivos y directorios (Move).

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

Si origen y destino están en la misma ruta, renombra. Si destino es un directorio, mueve origen dentro de él. Opciones: '-i' (confirmar), '-f' (forzar), '-v' (detalles).

```
mv nombre_original nombre_nuevo # Renombrar
mv archivo.txt directorio/       # Mover
```

Listing 14: Ejemplo mv

#### 7.2.11. `df`

Muestra espacio libre y utilizado en sistemas de archivos montados (Disk Free).

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

Opciones: '-h' (legible), '-T' (mostrar tipo sistema archivos).

```
df -Th
```

Listing 15: Ejemplo df

#### 7.2.12. `du`

Muestra el espacio utilizado por directorios (Disk Usage).

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

Opciones: '-h' (legible), '-s' (solo total), '-a' (incluir archivos). Por defecto, muestra uso por cada subdirectorio en la ruta.

```
du -h
du -sh /home
```

Listing 16: Ejemplo du

#### 7.2.13. `ln`

Crea enlaces (links) a archivos o directorios.

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

Opción '-s' crea enlace simbólico (suave). Por defecto crea enlace duro.

```
ln -s /usr/bin/python python_link # Enlace suave
ln archivo.txt hard_link          # Enlace duro
```

Listing 17: Ejemplo ln

### 7.2.14. `lsblk`

Lista información sobre dispositivos de bloque (discos, particiones).

```
lsblk [opciones]
```

Opciones: ‘-a‘ (todos), ‘-l‘ (lista simple). Útil para ver nombres de discos (sda, sdb) y sus particiones (sda1, sda2).

### 7.2.15. `mount`

Monta un sistema de archivos en un directorio. Requiere privilegios.

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

Opción ‘-r‘ monta en modo solo lectura.

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

Listing 18: Ejemplo mount

### 7.2.16. `umount`

Desmonta un sistema de archivos. Requiere privilegios.

```
umount [opciones] directorio | dispositivo
```

Opciones: ‘-f‘ (forzar), ‘-l‘ (desmontaje perezoso).

```
1 sudo umount /mnt
```

Listing 19: Ejemplo umount

### 7.2.17. `find`

Busca archivos y directorios.

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

Expresiones comunes:

- ‘-type d‘ (directorios), ‘-type f‘ (archivos).
- ‘-name ”patron« (buscar por nombre, puede usar comodines).
- ‘-empty‘ (archivos/directorios vacíos).

  ‘-exec comando ‘: Ejecuta ‘comando‘ para cada resultado (‘‘ se reemplaza por el nombre).

```
1 find . -type d                    # Busca directorios desde aqu
2 find /home -name "*.txt"          # Busca .txt en /home
3 find . -type f -empty -exec rm {} \; # Borra archivos vac os
```

Listing 20: Ejemplo find

### 7.2.18. Expansiones (Comodines)

Caracteres especiales para representar patrones en nombres de archivos/directorios. Interpretados por el shell.

- '*': Cualquier cadena de caracteres (incluyendo vacía). Ej: 'ls *.txt'
- '¿: Cualquier carácter individual. Ej: 'ls foto?.jpg'
- '[]': Un conjunto de caracteres posibles. Ej: 'ls [abc]*.txt' (empiezan con a, b o c), 'ls [0-9]*.log' (empiezan con dígito).
- '': Producto cartesiano (expansión de llaves). Ej: 'touch a,b1,2' crea 'a1 a2 b1 b2'. No considera archivos existentes.

```
ls /dev/sd?2
rm [a-z].{gif,jpg}
touch {a..h}{1..8}
```

Listing 21: Ejemplo Expansiones

## 7.3. Contenido y filtros

- **Flujos estándar:** stdin (entrada, teclado por defecto), stdout (salida normal, pantalla), stderr (salida de error, pantalla).
- `nano`: Editor de texto en terminal. Interactivo. Guardar (`Ctrl+O`), Salir (`Ctrl+X`), Cortar (`Ctrl+K`), Pegar (`Ctrl+U`).
- `cat`: Concatena y muestra contenido de archivos. Opción '-n' numera líneas.
- `less`: Visor de texto paginado (similar a 'man'). Navegación: flechas, espacio, 'g', 'G', '/' (buscar), 'q' (salir). Interactivo.
- `head`: Muestra primeras líneas (10 por defecto). '-n X' muestra X líneas.
- `tail`: Muestra últimas líneas (10 por defecto). '-n X' muestra X líneas. '-f' sigue cambios en tiempo real (logs).
- `sort`: Ordena líneas de texto. Opciones: '-n' (numérico), '-r' (reverso).
- `uniq`: Elimina/muestra líneas duplicadas *consecutivas*. Opciones: '-c' (contar), '-d' (solo duplicadas), '-u' (solo únicas). Usualmente se usa después de 'sort'.
- `strings`: Extrae cadenas de texto legibles de archivos binarios. '-n MINIMO' especifica longitud mínima.
- `wc`: Cuenta líneas ('-l'), palabras ('-w'), bytes ('-c'), caracteres ('-m').
- `file`: Identifica tipo de archivo.
- `cut`: Corta secciones de líneas. '-c RANGO' (caracteres), '-d DELIM' (delimitador), '-f CAMPOS' (campos).
- **Expresiones regulares:** Patrones para definir lenguajes/buscar texto.
  - Literales: 'a', 'b'.
  - Concatenación: 'ab'.
  - Unión: 'a—b'.
  - Agrupación: '(a—b)c'.

- Repetición: ‘¿ (0 o 1), ‘n,m‘ (n a m), ‘+‘ (1 o más), ‘*‘ (0 o más).
- Anclas: ‘‘(*iniciolínea*), ‘‘ (fin línea).
- Clases: ‘.‘ (cualquier caracter), ‘[abc]‘, ‘[a-z]‘, ‘[[:digit:]]‘, ‘[[:alpha:]]‘, etc.

- `tr`: Traduce o elimina caracteres. ‘tr ’[:lower:]’ ’[:upper:]’‘ (mayúsculas), ‘tr -d ’ ’‘ (elimina espacios).
- `grep`: Busca patrones (texto o regex) en archivos. Opciones: ‘-i‘ (ignorar mayús), ‘-v‘ (invertir), ‘-c‘ (contar), ‘-n‘ (núm línea), ‘-r‘ (recursivo), ‘-E‘ (regex extendida), ‘-o‘ (solo match), ‘-w‘ (palabra completa).

### 7.4. Secuenciación, redirección y tuberías

- **Secuenciación (Shell):**
  - ‘;‘: Ejecuta comandos en secuencia, independientemente del resultado.
  - ‘‘: Ejecuta siguiente comando solo si el anterior tuvo éxito (código salida 0).
  - ‘——‘: Ejecuta siguiente comando solo si el anterior falló (código salida != 0).
- **Redirección:** Cambia flujos estándar.
  - ‘¿‘: Redirige stdout a archivo (sobrescribe). ‘comando ¿archivo‘
  - ‘‘: Redirige stdout a archivo (añade al final). ‘comando  archivo‘
  - ‘2¿‘: Redirige stderr a archivo. ‘comando 2¿errores.log‘
  - ‘¿‘: Redirige stdout y stderr a archivo. ‘comando ¿salida$_{ye}$*rrores.log*“ < ‘ : *Redirigestdindesdearchi*
    *entrada.txt*‘

- ‘/dev/null‘: Archivo especial para descartar salida. ‘comando ¿/dev/null‘

**Tuberías (—):** Conecta stdout de un comando con stdin del siguiente. ‘comando1 — comando2 — comando3‘

`tee`: Lee de stdin y escribe a stdout Y a uno o más archivos. Opción ‘-a‘ para añadir. ‘comando — tee archivo.log‘

## 8. Comandos Avanzados de Bash

### 8.1. Usuarios, grupos y permisos

- `whoami`: Muestra usuario actual.
- `id`: Muestra UID, GID y grupos del usuario. ‘id [usuario]‘
- `who`: Muestra usuarios conectados.
- `su`: Cambia al usuario root o a [USUARIO]. Pide contraseña del usuario destino.
- `sudo`: Ejecuta un comando como otro usuario (root por defecto). Pide contraseña del usuario actual. ‘sudo comando‘
- `passwd`: Cambia contraseña del usuario actual o de [usuario].
- **Permisos Linux:** ‘rwx‘ para ‘u‘suario, ‘g‘rupo, ‘o‘tros.

- r (lectura=4), w (escritura=2), x (ejecución=1).
- Notación octal: Suma de valores (ej. ‘rwxr-xr-x‘ = ‘755‘).

- `chown`: Cambia propietario (usuario:grupo) de archivo/directorio. ‘-R‘ recursivo. ‘chown [-R] usuario[:grupo] archivo‘ (requiere ‘sudo‘ usualmente).
- `chmod`: Cambia permisos. ‘-R‘ recursivo.
  - Modo numérico: ‘chmod 755 archivo‘
  - Modo simbólico: ‘[ugoa][+-=][rwx]‘. Ej: ‘chmod u+x script‘, ‘chmod go-w datos‘.
- `useradd`: Crea nueva cuenta de usuario. ‘-m‘ crea home dir, ‘-g GRUPO‘ asigna grupo primario. Requiere ‘sudo‘.
- `userdel`: Elimina cuenta de usuario. ‘-r‘ elimina home dir. Requiere ‘sudo‘.

- `usermod`: Modifica cuenta de usuario. ‘-l‘ (nuevo nombre), ‘-d‘ (nuevo home), ‘-g‘ (nuevo grupo primario), ‘-G‘ (grupos secundarios), ‘-a‘ (añadir grupo secundario con -G), ‘-s‘ (nuevo shell). Requiere ‘sudo‘. ‘usermod -a -G sudo nuevo$_u$*suario*‘

## 8.2. Procesos y tareas

- `ps`: Muestra procesos (Process Status). Opciones: ‘-e‘ (todos), ‘-f‘ (full format), ‘-U USUARIO‘, ‘–forest‘ (árbol). ‘ps aux‘ es común.
- `nice`: Ejecuta comando con prioridad específica (‘-n NIVEL‘, -20 mayor, 19 menor).
- `top`: Monitor interactivo de procesos y uso de recursos. Actualiza periódicamente. Salir con ‘q‘. ‘-d n‘ cambia intervalo.
- **Señales:** Mecanismo para comunicar eventos a procesos (interrupciones HW, eventos SO, solicitudes usuario).
  - SIGINT (2): Interrupción (Ctrl+C).
  - SIGKILL (9): Matar proceso (forzado).
  - SIGTSTP (20): Detener proceso (Ctrl+Z).
  - SIGCONT (18): Continuar proceso detenido.
- `kill`: Envía señal a proceso. ‘kill -s SEÑAL PID...‘. Ej: ‘kill -9 1234‘.
- **Tareas (Jobs):** Procesos ejecutados en el shell (primer plano o segundo plano).
  - Primer plano (foreground): Bloquea terminal.
  - Segundo plano (background): No bloquea. Ejecutar con ‘‘ al final. ‘comando ‘
- `jobs`: Lista tareas activas en el shell actual. ‘-p‘ muestra PIDs.
- `bg`: Reanuda una tarea detenida en segundo plano. ‘bg
- `fg`: Trae una tarea (detenida o en segundo plano) a primer plano. ‘fg
- `disown`: Desasocia una tarea del shell, permitiendo que continúe si se cierra el shell. ‘disown
- `wait`: Espera a que terminen procesos hijos o un PID específico. ‘wait [PID...]‘.
- `exec`: Reemplaza el proceso actual del shell con un nuevo comando (no crea proceso nuevo). ‘exec comando‘.

### 8.3. Gestión

- `apt`: Herramienta de gestión de paquetes (Debian/Ubuntu).
  - ‘apt update‘: Actualiza lista de paquetes.
  - ‘apt install paquete‘: Instala paquete.
  - ‘apt upgrade‘: Actualiza paquetes instalados.
  - ‘apt remove paquete‘: Desinstala paquete.
  - (Alternativa: ‘snap‘).
- `free`: Muestra uso de memoria RAM e intercambio (swap). ‘-h‘ legible.
- `loadkeys`: Configura distribución teclado en consola virtual. ‘loadkeys es‘. Mapas en ‘/usr/share/kbd/keymaps/...‘.
- `setfont`: Cambia fuente en consola virtual. ‘setfont nombre$_{f}uente$‘.$Fuentesen$‘$/usr/share/console$
  $Configuradistribuciónteclad oenentornográfico$.‘$setxkbmapes$‘.$Mapasen$‘$/usr/share/X11/xkb/$‘
- `which`: Muestra ruta absoluta de un comando ejecutable. ‘-a‘ muestra todas las ubicaciones.
- `startx`: Inicia entorno gráfico (si no se inicia automáticamente).

### 8.4. Internet

- `ping`: Verifica conectividad con host (IP o nombre). ‘ping DESTINO‘.
- `ssh`: Conexión segura a máquina remota (Secure Shell). ‘ssh [-p puerto] usuario@equipo‘. Salir con ‘exit‘.
- `scp`: Copia segura de archivos entre hosts vía SSH. ‘scp [-P puerto] origen usuario@destino:ruta‘.
- `wget`: Descarga archivos desde URL. ‘-O archivo‘ (guardar como), ‘-b‘ (background), ‘-r‘ (recursivo).
- `curl`: Transfiere datos desde/hacia servidor (muy versátil, APIs). ‘-o archivo‘ (guardar salida).
- `w3m`: Navegador web en modo texto.

### 8.5. Compresión y backup

- `tar`: Empaqueta/desempaqueta archivos (Tape Archiver). Opciones: ‘-c‘ (crear), ‘-x‘ (extraer), ‘-t‘ (listar), ‘-f archivo.tar‘ (especificar archivo), ‘-r‘ (agregar), ‘-v‘ (verbose). ‘tar -cvf backup.tar directorio/‘.
- `gzip`: Comprime/descomprime un único archivo (formato ‘.gz‘). ‘-d‘ (descomprimir), ‘-k‘ (mantener original). ‘gzip archivo‘.
- `rsync`: Sincroniza/transfiere archivos eficientemente (local/remoto). Opciones: ‘-r‘ (recursivo), ‘-v‘ (verbose), ‘-n‘ (simular), ‘-z‘ (comprimir). Copia solo diferencias.
- `dd`: Copia y convierte datos a bajo nivel (ej. clonar discos). Peligroso si se usa mal.

### 8.6. Otros comandos

- `alias`: Crea/muestra atajos para comandos. ‘alias nombre=çomando«. Persistencia: editar ‘ /.bashrc‘.
- `unalias`: Elimina alias. ‘-a‘ todos.
- `bc`: Calculadora de precisión arbitraria (interactiva o con pipes). ‘echo ”3+4”— bc‘.
- `sha256sum`: Calculo/verifica hash SHA-256 de archivo (integridad). ‘-c archivo$_s$*umas*‘*verifica*.`time`
  $Midetiempodeejecucióndecomando(real, user, sys)$.‘*timecomando*‘.

- `watch`: Ejecuta comando periódicamente y muestra salida. ‘-n segs‘ (intervalo), ‘-d‘ (resaltar diffs), ‘-t‘ (sin hora).
- `xclip`: Maneja portapapeles (X Window). ‘-o‘ (pegar), ‘-se c‘ (usar clipboard). ‘echo texto — xclip -se c‘ (copiar).
- `xargs`: Construye y ejecuta líneas de comando desde entrada estándar.
- `screen`: Multiplexor de terminal (sesiones persistentes).
- `ranger`: Administrador de archivos en terminal.

## 9. Shell Scripting

Archivo de texto con comandos Bash para automatizar tareas. Extensión ‘.sh‘.

### 9.1. Introducción

- **Comentarios:** Empiezan con ‘‘. Ignorados por el intérprete.
- **Ejecución:**
  - ‘bash script.sh‘ (Pasa script al intérprete).
  - ‘./script.sh‘ (Si tiene permiso de ejecución ‘chmod +x script.sh‘).
- **Shebang:** Primera línea ‘!/bin/bash‘ (o ruta al intérprete deseado). Indica qué intérprete usar si se ejecuta directamente.

### 9.2. Variables

Símbolos para almacenar valores.

- **Variables locales:** Definidas por usuario. Alcance local. Sintaxis: ‘NOMBRE=VALOR‘ (sin espacios). Referencia: ‘$NOMBRE$‘*o*‘NOMBRE‘.

```
1        NOMBRE="Damian"
2        echo "Hola $NOMBRE."
3
```

Listing 22: Variable Local

- **Variables de entorno:** Especiales, disponibles para todos los procesos. Ej: ‘HOME‘, ‘PATH‘, ‘USER‘, ‘HOSTNAME‘, ‘SHELL‘, ‘PWD‘, ‘OLDPWD‘, ‘PS1‘ (prompt). El shell usa ‘PATH‘ para buscar comandos.
- **Variables especiales:** Actualizadas por el shell.
  - ‘0‘ : $Nombredelscript/comando$.‘1‘, ‘2‘, ... : $Argumentosposicionalespasadosalscript$.

- ‘@‘ : $Todoslosargumentoscomolistaseparada$.‘*‘: Todos los argumentos como una sola cadena.

- ‘‘ : $Númerodeargumentospasados$.‘¿: Código de salida del último comando (0 = éxito, != 0 = error).

- ‘

  ‘ : $PIDdelshellactual$.‘

  ¡: PID del último proceso en segundo plano.

### 9.3. Comandos (Built-in)

Comandos integrados en el shell.

- `test / [ expression ]`: Evalúa expresiones condicionales. Devuelve código de salida 0 (true) o != 0 (false).
  - Comparación cadenas: ‘=‘, ‘!=‘.
  - Comparación números: ‘-eq‘ (igual), ‘-ne‘ (no igual), ‘-gt‘ (mayor), ‘-ge‘ (mayor/igual), ‘-lt‘ (menor), ‘-le‘ (menor/igual).
  - Chequeo archivos: ‘-e‘ (existe), ‘-d‘ (es directorio), ‘-f‘ (es archivo regular).
  - Chequeo cadenas: ‘-n‘ (no vacía), ‘-z‘ (vacía).
  - Lógicos: ‘¡ (negar), ‘-a‘ (AND), ‘-o‘ (OR).

```
1         if test "$USER" = "root"; then echo "Eres root"; fi
2         if [ 1 -eq 2 ]; then echo "Iguales"; else echo "Distintos
    "; fi
3
```

Listing 23: Ejemplo test

- `exit [CODIGO]`: Finaliza script/shell con código de salida opcional (por defecto, el del último comando).
- `source ARCHIVO` (o `. ARCHIVO`): Ejecuta comandos de ARCHIVO en el shell *actual* (no crea subproceso). Útil para cargar configuraciones/variables.
- `read [opciones] var1 [var2...]`: Lee entrada del usuario y la asigna a variables. ‘-p ”prompt« (mostrar mensaje), ‘-s‘ (ocultar entrada).
- `seq [-s SEP] INICIO [INC] FINAL`: Genera secuencia de números.
- `shift [n]`: Desplaza argumentos posicionales ‘1,2,...‘ hacia la izquierda ‘n‘ veces (por defecto 1). ‘1‘$sepierde$.`export VAR[=VALOR]` : $Marcaunavariablelocalparaserexporta$
- `trap ACCION SEÑAL...`: Establece una acción a ejecutar cuando se recibe una señal. ‘trap ” INT‘ (ignora Ctrl+C). ‘trap ’rm tempfile; exit’ EXIT‘ (limpia al salir).

### 9.4. Encomillados y escapes

Controlan la interpretación y expansión de caracteres.

- **Carácter de escape (‘):** Quita significado especial al siguiente carácter. ‘echo $USER‘ imprime ‘$USER$‘.‘$toucharchivo\ con\ espacios$‘.**Comillas simples (’’text** imprime ‘Hola $USER$‘.

- **Comillas dobles (‘”texto«): Permiten expansión de variables (‘$VAR‘), sustituciónde comandos(“‘coma y secuencias de escape (“, ‘$‘). Preservan espacios. ‘”Hola $USER”‘imprime‘Hola < usuario > ‘.**

## 9.5. Expresiones aritméticas

- **Sintaxis: ‘((EXPRESION))‘.Evalúaexpresionesaritméticasenteras.‘echo((5+**

## 9.6. Control de flujo

- **`if / else / fi`: Ejecución condicional basada en código de salida de un comando.**

```
if comando; then
  # si comando exito (0)
elif otro_comando; then
  # si otro_comando exito (0)
else
  # si ambos fallaron
fi

```

**`[[ expression ]]`: Variante de bash (no ‘test‘/‘[ ]‘) más flexible para comparaciones y lógica. Soporta ‘==‘, ‘!=‘, ‘¡‘, ‘¿‘, ‘“, ‘——‘, ‘¡, ‘= ‘ (regex).**

```
if [[ $# -eq 0 || ! $1 =~ ^[0-9]+$ ]]; then
  echo "Error"
fi

```

- **`for variable in lista; do ... done`: Itera sobre elementos de una lista (separados por espacio/salto línea).**

```
for ARCHIVO in *.txt; do
  echo "Procesando $ARCHIVO"
done

```

- **`while comando; do ... done`: Repite mientras comando tenga éxito (salida 0).**
- **`until comando; do ... done`: Repite mientras comando falle (salida != 0).**
- **`case variable in patron1) ... ;; patron2) ... ;; *) ... ;; esac`: Evalúa variable contra patrones (con comodines ‘*‘, ‘¿, ‘[]‘). Similar a switch. ‘;;‘ termina cada bloque.**
- **`select variable in opcion1 opcion2 ...; do ... done`: Crea menú interactivo numerado. Elige opción por número (guardado en ‘REPLY‘), título en ‘variable‘. Usado con ‘case‘.**
- **Funciones: Agrupan comandos para reutilización.**

```
nombre_funcion() {
  # Codigo, $1, $2.. son argumentos
  echo "Hola, $1"
  return 0 # Opcional: codigo salida
}
# Llamada:
```

```
7 saludar "Mundo"
8
```

# Parte III

# Herramientas auxiliares

## 10. Control de Versiones (Git)

### 10.1. Introducción

**VCS (Version Control System): Herramienta para gestionar y controlar cambios en archivos/proyectos. Rastrea quién hizo qué cambio, cuándo y por qué. Esencial para desarrollo colaborativo. Ej: Git, Subversion, Mercurial.**

**Git: VCS distribuido, el más popular. Áreas de trabajo Git:**

1. **Directorio de trabajo (Working Tree): Carpeta actual con archivos (modificados o no).**
2. **Área de preparación (Staging Area / Index): Área intermedia donde se preparan los cambios para el próximo commit.**
3. **Repositorio (.git): Base de datos que almacena historial completo de cambios (commits).**

**Estados de un archivo en Git:**

1. **No rastreado (Untracked): Nuevo archivo que Git no sigue.**
2. **Modificado (Modified): Archivo rastreado con cambios no preparados.**
3. **Preparado (Staged): Archivo con cambios añadidos al área de preparación.**
4. **Confirmado (Committed): Cambios guardados permanentemente en el repositorio.**

**Configuración (`git config`): Establece opciones a nivel local (repositorio), global (usuario) o sistema.**

```
git config [--global | --system] configuracion valor
```

**Configuraciones iniciales importantes: ‘user.name‘, ‘user.email‘. Ver config: ‘git config -l‘.**

### 10.2. Comandos básicos

- **`git init [ruta]`: Inicializa un nuevo repositorio Git (crea carpeta ‘.git‘).**
- **`git status [-s]`: Muestra estado del directorio de trabajo y área de preparación. ‘-s‘ para formato corto.**

- ◇ `git add [opciones] archivo...`: Añade cambios del directorio de trabajo al área de preparación. ‘git add .‘ añade todo.
- ◇ `git commit [-a] [-m mensaje]`: Guarda los cambios preparados (staged) en el repositorio. ‘-m‘ para mensaje corto, ‘-a‘ para añadir y confirmar archivos modificados/eliminados rastreados. Mensajes claros y concisos son importantes.
- ◇ `git log [opciones]`: Muestra historial de commits. Opciones: ‘–oneline‘, ‘–graph‘, ‘-n ¡num¿‘. Cada commit tiene un hash único (SHA-1).
- ◇ `git restore [opciones] archivo...`: Restaura archivos.
- ◇ ‘git restore archivo‘: Descarta cambios en directorio de trabajo (vuelve a versión staged/committed).
- ◇ ‘git restore –staged archivo‘: Saca archivo del área de preparación (unstage), pero mantiene cambios en dir. trabajo.
- ◇ ‘-s ¡fuente¿‘: Especifica desde dónde restaurar (ej. un commit específico).
- ◇ `git diff [opciones] [archivo...]`: Muestra diferencias.
- ◇ ‘git diff‘: Cambios en dir. trabajo vs área de preparación.
- ◇ ‘git diff –staged‘: Cambios en área de prep. vs último commit.
- ◇ ‘git diff commit1 commit2‘: Diferencias entre dos commits.
- ◇ `git reset`: (No detallado en PDF, comando avanzado para deshacer commits, mover HEAD).
- ◇ `git rm`: Elimina archivo del directorio de trabajo Y del área de preparación. ‘git rm archivo‘. Equivalente a ‘rm archivo  git add archivo‘.

## 10.3. Trabajando con ramas

- ◇ Rama (Branch): Línea de desarrollo independiente. Permite trabajar en paralelo sin afectar la línea principal (usualmente ‘master‘ o ‘main‘). Es un puntero a un commit.
- ◇ HEAD: Puntero especial que indica la rama/commit actual en el que se está trabajando.
- ◇ `git branch [-a | nombre]`: Lista ramas locales (‘-a‘ todas, locales y remotas), o crea una nueva rama ‘nombre‘.

◇ **git switch nombre**$_{r}ama : Cambiaalaramaespecificada(mueveHEADyactualizadirectoriodetrabajo).(Reco$

### 10.4. Repositorios remotos

**Versiones del proyecto almacenadas en otro lugar (servidor, otra máquina). Plataformas: GitHub, GitLab, Bitbucket.**

- **`git remote [-v | add nombre ruta | remove nombre]`: Gestiona conexiones remotas. '-v' lista remotos y URLs. 'add' añade uno nuevo (usualmente llamado 'origin'). 'remove' lo quita.**
- **`git push [-u remoto rama]`: Envía commits locales a la rama del repositorio remoto. '-u' establece seguimiento (upstream) para futuros push/pull.**
- **`git fetch [remoto] [rama]`: Descarga cambios y referencias del remoto al repositorio local, *sin* fusionarlos con el trabajo local. Actualiza las ramas remotas locales (ej. 'origin/master').**
- **`git pull [remoto] [rama]`: Recupera cambios del remoto y los fusiona automáticamente con la rama local actual. Equivale a 'git fetch' seguido de 'git merge'.**
- **`git clone ruta [directorio] [--depth num]`: Crea una copia local de un repositorio remoto. Configura 'origin' automáticamente. '–depth' clona historial limitado.**
- **Fork: Copia personal de un repositorio (en la plataforma remota).**
- **Pull Request (Merge Request): Solicitud para incorporar cambios de tu fork/rama en el repositorio original/otra rama.**
- **LFS (Large File Storage): Extensión de Git para manejar archivos grandes eficientemente.**

## 11. Contenedores

**Conceptos relacionados (resumen muy breve del PDF):**

- **Simulación: Imitar comportamiento de un sistema.**
- **Emulación: Replicar funcionalidad de un hardware/software en otro diferente.**
- **Virtualización: Crear máquinas virtuales (VMs) con SO completos sobre un hardware físico.**
- **Contenedores: Empaquetan aplicaciones y sus dependencias, aislándolas pero compartiendo el kernel del SO anfitrión. Más ligeros que VMs. (Ej. Docker).**

## 12. Otros Conceptos

- **`venv`: Herramienta de Python para crear entornos virtuales aislados.**

- ◇ `chroot`: Cambia el directorio raíz aparente para un proceso y sus hijos.
- ◇ `Colab` (Google Colaboratory): Entorno de notebooks Jupyter en la nube, permite ejecutar código (Python, Bash con ‘¡).

# Apéndice

## A. Instalación de Linux

### A.1. Máquina Virtual (VM)

**Software que permite ejecutar un SO (ïnvitado”) dentro de otro SO (.ªnfitrión”).**

- ◇ **VirtualBox: Software de virtualización de código abierto (Oracle).**
- ◇ **Lubuntu: Distribución Linux ligera (usada como ejemplo).**
- ◇ **Proceso Creación VM (VirtualBox):**
  1. **Nueva VM.**
  2. **Configurar SO: Nombre, carpeta, ISO de Lubuntu.**
  3. **Hardware: Asignar RAM (mín 2GB recomendado), CPUs.**
  4. **Disco Virtual: Crear disco duro virtual (mín 15GB recomendado).**
  5. **Resumen y Terminar.**
  6. **Iniciar VM (arrancará desde la ISO).**
- ◇ **Proceso Instalación (Lubuntu LiveCD):**
  1. **Elegir Ïnstall Lubuntu”.**
  2. **Idioma.**
  3. **Región/Zona horaria.**
  4. **Distribución de teclado.**
  5. **Personalización: Instalación mínima/completa, software de terceros (opcional).**
  6. **Particionado: Elegir ”borrar disco”(opción simple para VM).**
  7. **Crear usuario principal: Nombre, usuario, nombre equipo, contraseña.**
  8. **Resumen.**
  9. **Instalar.**
  10. **Reiniciar VM (quitar ISO virtual al reiniciar).**

### A.2. Guest Additions (VirtualBox)

**Controladores y utilidades para mejorar integración entre anfitrión e invitado.**

- **Integración ratón/teclado fluida.**
- **Pantalla completa/redimensionable.**
- **Portapapeles compartido.**
- **Carpetas compartidas.**

**Instalación (ej. en Lubuntu/Debian):**

```
sudo apt update
sudo apt install virtualbox-guest-x11
```

### A.3. Instalación en pendrive

- **Herramienta: Rufus (u otras como Etcher).**
- **Requisito: Pendrive (mín 16GB, USB 3.0 preferible).**
- **Precaución: Se borrarán datos del pendrive.**
- **Configuración (Rufus): Seleccionar dispositivo (pendrive), imagen ISO, esquema partición, sistema archivos.**
- **Persistencia (Opcional): Permite guardar cambios entre sesiones. Mover deslizador en Rufus. *Hace el sistema más lento.* Recomendado sin persistencia para instalación limpia.**
- **Arranque: Reiniciar PC, entrar al menú de booteo (tecla varía: Esc, F9, F11, F12...) y elegir el pendrive USB.**

### A.4. Otras alternativas

- **WSL (Windows Subsystem for Linux): Ejecutar entorno Linux en Windows sin VM. Usa capa de compatibilidad. Instalación: 'wsl –install -d ¡distro¿' (ej. ubuntu). Ejecutar con 'wsl'.**
- **Colab: Entorno online para Bash. Requiere instalar 'colab-xterm' ('!pip install colab-xterm'), cargar extensión ('**
- **Replit, CoCalc, Termux (Android): Otras plataformas/apps para entornos Linux/terminal.**

## B. Resolución de Problemas

### B.1. Activar IVT/AMD-V

**Tecnologías de virtualización hardware (Intel/AMD). Necesarias para VMs eficientes. Se activan en configuración BIOS/UEFI del PC (tecla al arrancar: F2, F10, DEL...).**

### B.2. Librerías de C++

**Programas Windows creados con Visual C++ pueden requerir "Visual C++ Redistributable Package". Descargar desde Microsoft si es necesario.**

### B.3. Poner man en español

**Instalar páginas del manual en español (ej. Debian/Ubuntu):**

```
sudo apt update
sudo apt install manpages-es
```

### B.4. Actualizar Kernel para WSL

**Si WSL2 da error 0x800701bc, requiere actualización del kernel.**

```
wsl --update
wsl --install -d <distro> # Reintentar instalaci n
```

## C. Programando un Shell (Python Examples)

**El PDF incluye código Python para ilustrar cómo implementar funciones básicas de un shell:**

- **15.1 Linea de comandos: Bucle simple 'while True', lee comando con 'input()', usa 'os.fork()' para crear proceso hijo, 'os.execv()' para ejecutar comando en hijo, 'os.waitid()' para esperar en padre.**
- **15.2 Variable PATH: Función 'buscar(comando)' que obtiene 'PATH' ('os.getenv'), divide por ':', prueba cada ruta ('os.path.isfile'), devuelve ruta completa o comando original. 'ejecutar' ahora usa 'buscar'.**
- **15.3 Manejo de argumentos: Función 'interpretar(comando)' que divide comando en programa y argumentos ('split()'), maneja alias (si 'programa' está en dict 'alias'), devuelve 'programa', '[programa] + argumentos'. 'ejecutar' recibe argumentos.**
- **15.4 Cambiar de directorio: Función 'comando**$_c d(argumentos)$'$usa$'$os.chdir$ '$os.getenv("HOME")$').$Bucleprincipalllamaa$'$comando_c d$'$si$'$programa ==$ "$cd$"'.**15.5 Prompt:**$Función$'$prompt()$'$queobtienedirectorioactual$('$os.getcwd$'),

- **15.6 Alias: Diccionario global 'alias = '. Función 'comando**$_a lias(argumentos, alias)$'$quemaneja$'$aliasno$ $comando$'$o$'$alias$'$(listar)$.$Bucleprincipallollama$.**15.7 Segundo plano:**'$interpretar$'$detecta$"$al final, devuelve$
- **15.8 Secuenciación: 'interpretar' ahora divide por ';' y usa 'yield' para devolver '(programa, argumentos, background)' por cada comando en la secuencia. Bucle principal itera sobre lo que devuelve 'interpretar'.**
- **15.9 Exit: Bucle principal cambia condición 'while comandos != .exit":'. Llama a 'interpretar' solo si 'comandos != .exit«.**
- **15.10 Historial: Lista global 'historial = []'. Guarda comando en 'historial' (si no empieza con '¡). 'interpretar' maneja '!¡ (reemplaza con 'historial[-1]') y '!n' (reemplaza con 'historial[n]'). Añade comando 'history' que itera e imprime 'historial'.**