Terminal de Comandos (CMD)

Informática Aplicada I
Undécimo (A)
20 de julio del 2025

Interfaz de Línea de Comandos (CLI)

1.1. Definición y Propósito

La **Interfaz de Línea de Comandos (CLI)** es un método de interacción con el sistema operativo o una aplicación de software mediante la introducción de comandos de texto. A diferencia de las interfaces gráficas de usuario (GUI), donde se interactúa con elementos visuales (iconos, ventanas, botones), la CLI se basa en texto y permite a los usuarios ejecutar operaciones directamente escribiendo instrucciones.

El propósito principal de la CLI es proporcionar un control más directo, preciso y potente sobre el sistema. Es fundamental para:

- **Automatización de Tareas:** Ejecutar secuencias de comandos (scripts) para automatizar procesos repetitivos.
- **Administración de Servidores:** Muchos servidores no tienen interfaz gráfica, por lo que la CLI es la única forma de gestionarlos.
- **Diagnóstico y Resolución de Problemas:** Permite acceder a funcionalidades de bajo nivel para identificar y solucionar fallos.
- **Eficiencia:** Para usuarios avanzados, la CLI puede ser más rápida que la GUI para ciertas tareas.
- **Acceso Remoto:** Permite controlar sistemas a distancia a través de conexiones seguras.

1.2. CLI vs. GUI: Ventajas y Desventajas

Característica	Interfaz de Línea de Comandos (CLI)	Interfaz Gráfica de Usuario (GUI)
Interacción	Basada en texto, comandos escritos.	Basada en elementos visuales (iconos, ventanas).
Curva de Aprendizaje	Más empinada, requiere memorizar comandos.	Más intuitiva, fácil de aprender para principiantes.
Precisión/Control	Muy alta, control granular sobre el sistema.	Menor, limitada a las opciones visuales predefinidas.
Velocidad (Usuario Experto)	Muy rápida para tareas repetitivas o complejas.	Puede ser más lenta para tareas repetitivas o complejas.
Automatización	Excelente, ideal para scripting.	Limitada, requiere herramientas adicionales.
Uso de Recursos	Bajo consumo de recursos del sistema.	Mayor consumo de recursos (memoria, CPU).
Entorno Típico	Servidores, desarrollo de software, administración de sistemas.	Usuarios finales, aplicaciones de escritorio.
Ejemplos	CMD (Windows), Bash (Linux/macOS), PowerShell (Windows).	Windows Desktop, macOS Finder, Android, iOS.

Comandos Básicos de Terminal

Los siguientes comandos son fundamentales para interactuar con el sistema de archivos y realizar operaciones básicas, tanto en Windows (CMD/PowerShell) como en Linux/macOS (Bash/Shell).

2.1. Comandos de Navegación

- `cd` **(Change Directory)**
  - **Función:** Permite navegar por el sistema de archivos, cambiando el directorio de trabajo actual.
  - **Sintaxis:**
    - `cd <ruta>`: Cambia al directorio especificado por `<ruta>`.
    - `cd ..`: Sube un nivel en la jerarquía de directorios (va al directorio padre).
    - `cd \` (Windows): Va a la raíz de la unidad actual.
    - `cd /` (Linux/macOS): Va a la raíz del sistema de archivos.
    - `cd ~` (Linux/macOS): Va al directorio de inicio del usuario.
  - **Ejemplos:**
    - `cd Documentos`
    - `cd ..\..\Proyectos (Windows)`
    - `cd /home/usuario/descargas (Linux/macOS)`
- `pwd` **(Print Working Directory)**
  - **Función:** Muestra la ruta absoluta (completa) del directorio actual en el que te encuentras.
  - **Nota:** En Windows CMD, el prompt (`C:\Users\TuUsuario>`) ya muestra el directorio actual, por lo que `pwd` no es un comando nativo, aunque PowerShell sí lo tiene (`Get-Location` o su alias `pwd`).
  - **Ejemplo (Linux/macOS):**
  - `$ pwd`
  - `/home/usuario/documentos`

2.2. Comandos de Gestión de Archivos y Directorios

- `dir` **(Windows)** **/** `ls` **(Linux/macOS)**
  - **Función:** Lista el contenido de un directorio (archivos y subdirectorios).
  - **Sintaxis y Opciones Comunes:**
    - `dir`: Lista el contenido del directorio actual (Windows).
    - `dir /w`: Lista en formato ancho (solo nombres, múltiples columnas) (Windows).
    - `dir /p`: Lista paginada (pausa después de cada pantalla) (Windows).
    - `ls`: Lista el contenido del directorio actual (Linux/macOS).
    - `ls -l`: Lista en formato largo (detalles de permisos, tamaño, fecha, propietario) (Linux/macOS).
    - `ls -a`: Muestra todos los archivos, incluyendo los ocultos (Linux/macOS).
    - `ls -lh`: Formato largo con tamaños legibles para humanos (Linux/macOS).
  - **Ejemplos:**
    - `dir`
    - `ls -l`

- **`mkdir` (Make Directory)**
  - **Función:** Crea uno o más directorios nuevos.
  - **Sintaxis:** `mkdir <nombre_directorio>`
  - **Opción Común:** `mkdir -p <ruta_anidada>` (Linux/macOS): Crea directorios padres si no existen.
  - **Ejemplos:**
    - `mkdir MisDocumentos`
    - `mkdir -p Proyectos/2025/Reportes` (Linux/macOS)
- **`rmdir` (Remove Directory) / `rm` (Remove)**
  - **Función:** Elimina directorios. `rmdir` solo elimina directorios vacíos en Windows. `rm` es más potente y se usa con opciones para directorios no vacíos.
  - **Sintaxis y Opciones Comunes:**
    - `rmdir <nombre_directorio>`: Elimina un directorio vacío (Windows).
    - `rmdir /s /q <nombre_directorio>`: Elimina un directorio no vacío y su contenido sin pedir confirmación (Windows).
    - `rm -r <nombre_directorio>`: Elimina un directorio y su contenido de forma recursiva (Linux/macOS).
    - `rm -rf <nombre_directorio>`: Elimina un directorio y su contenido de forma recursiva y forzada (sin pedir confirmación). ¡Usar con extrema precaución! (Linux/macOS).
  - **Ejemplos:**
    - `rmdir CarpetaVacia`
    - `rm -r ProyectoAntiguo/`
- **`copy` (Windows) / `cp` (Linux/macOS)**
  - **Función:** Copia archivos y directorios.
  - **Sintaxis y Opciones Comunes:**
    - `copy <origen> <destino>`: Copia un archivo (Windows).
    - `xcopy <origen> <destino> /e /i`: Copia directorios y subdirectorios (Windows).
    - `cp <origen> <destino>`: Copia un archivo (Linux/macOS).
    - `cp -r <origen> <destino>`: Copia directorios de forma recursiva (Linux/macOS).
  - **Ejemplos:**
    - `copy informe.docx C:\Backup\`
    - `cp -r Documentos/ RespaldoDocumentos/`
- **`move` (Windows) / `mv` (Linux/macOS)**
  - **Función:** Mueve archivos y directorios de una ubicación a otra. También se usa para renombrar.
  - **Sintaxis:** `move <origen> <destino>`
  - **Ejemplos:**
    - `move archivo.txt C:\ArchivosTemporales\`
    - `mv informe_viejo.txt informe_final.txt` (Renombrar)
- **`del` (Windows) / `rm` (Linux/macOS)**
  - **Función:** Elimina archivos.
  - **Sintaxis y Opciones Comunes:**
    - `del <nombre_archivo>`: Elimina un archivo (Windows).
    - `del /q *.tmp`: Elimina todos los archivos `.tmp` sin pedir confirmación (Windows).
    - `rm <nombre_archivo>`: Elimina un archivo (Linux/macOS).
    - `rm -f <nombre_archivo>`: Elimina un archivo sin pedir confirmación (Linux/macOS).
  - **Ejemplos:**
    - `del archivo_temporal.log`
    - `rm imagen_duplicada.jpg`
- **`type` (Windows) / `cat` (Linux/macOS)**
  - **Función:** Muestra el contenido de un archivo de texto directamente en la terminal.
  - **Sintaxis:**
    - `type <nombre_archivo>` (Windows)
    - `cat <nombre_archivo>` (Linux/macOS)

    - **Ejemplo:**
        - `type readme.txt`
        - `cat script.sh`
- **`help` (Windows) / `man` (Linux/macOS)**
    - **Función:** Proporciona ayuda y manuales para los comandos de la terminal.
    - **Sintaxis:**
        - `help <comando>` (Windows)
        - `man <comando>` (Linux/macOS)
    - **Ejemplos:**
        - `help cd`
        - `man ls`
- **`cls` (Windows) / `clear` (Linux/macOS)**
    - **Función:** Limpia la pantalla de la terminal, eliminando el historial de comandos y la salida anterior.
    - **Sintaxis:**
        - `cls` (Windows)
        - `clear` (Linux/macOS)
    - **Ejemplo:**
        - `cls`
- **`sudo` (Linux/macOS)**
    - **Función:** Permite ejecutar un comando con privilegios de superusuario (administrador). Es esencial para tareas que requieren acceso al sistema a un nivel más profundo.
    - **Sintaxis:** `sudo <comando>`
    - **Ejemplo:**
        - `sudo apt update` (Actualiza la lista de paquetes en sistemas basados en Debian/Ubuntu)

Shell Scripting Básico

¿Qué es Shell Scripting?

El **Shell Scripting** es la capacidad de escribir secuencias de comandos (scripts) que el intérprete de comandos (shell) puede ejecutar automáticamente. Un script es un archivo de texto que contiene una serie de comandos de la terminal, lógica de programación (variables, condicionales, bucles) y comentarios, que se ejecutan en orden.

El objetivo principal del shell scripting es **automatizar tareas repetitivas**, lo que ahorra tiempo, reduce errores manuales y mejora la eficiencia en la administración de sistemas y el desarrollo de software.

Creación y Ejecución de un Script Básico

Pasos Generales:

1. **Crear el archivo:** Abre un editor de texto (Bloc de notas en Windows, Gedit/Nano/Vim en Linux).
2. **Escribir los comandos:** Introduce los comandos de la terminal línea por línea.
3. **Guardar el archivo:**
    - **Windows:** Guarda con extensión `.bat` (ej. `mi_script.bat`).
    - **Linux/macOS:** Guarda con extensión `.sh` (ej. `mi_script.sh`).
4. **Permisos de Ejecución (Linux/macOS):** Antes de ejecutar un script `.sh`, debes darle permisos de ejecución:
    - `chmod +x mi_script.sh`
5. **Ejecutar el script:**
    - **Windows:** Haz doble clic en el archivo `.bat` o ejecútalo desde CMD: `mi_script.bat`.
    - **Linux/macOS:** Ejecútalo desde la terminal: `./mi_script.sh` (el `./` indica que está en el directorio actual).

Elementos Básicos de un Script

- **Comentarios:** Líneas que no se ejecutan, usadas para documentar el código.
  - Windows (`.bat`): Comienzan con `REM` o `::`.
  - Linux (`.sh`): Comienzan con `#`.
- **`echo`:** Muestra texto en la terminal.
  - Windows: `echo Hola Mundo`
  - Linux: `echo "Hola Mundo"`
- **`@echo off` (Windows `.bat`):** Se coloca al inicio de un script `.bat` para evitar que los comandos se muestren en la terminal mientras se ejecutan, solo se muestra la salida de los comandos.
- **`pause` (Windows `.bat`):** Detiene la ejecución del script y espera a que el usuario presione una tecla. Útil para ver la salida antes de que la ventana se cierre.
- **Redirección de Salida (>):** Envía la salida de un comando a un archivo en lugar de a la pantalla.
  - `comando > archivo.txt`: Sobreescribe el archivo.
  - `comando >> archivo.txt`: Añade al final del archivo.
  - `>nul 2>&1` (Windows): Redirige la salida estándar y los errores a "ningún lugar", haciendo que el comando sea silencioso.

Ejemplos de Scripts Sencillos

Ejemplo 1: Script para Organizar Archivos (Windows .bat)

Este script crea una carpeta y mueve todos los archivos de un tipo específico a esa carpeta.

```
:: mi_organizador.bat
@echo off
echo Iniciando organización de archivos...

:: Crear una carpeta para documentos de texto si no existe
mkdir DocumentosTexto >nul 2>&1

:: Mover todos los archivos .txt a la carpeta DocumentosTexto
move *.txt DocumentosTexto >nul 2>&1

echo Archivos .txt movidos a DocumentosTexto.
echo Proceso de organización finalizado.
pause
```

Ejemplo 2: Script para Copia de Seguridad Simple (Linux .sh)

Este script crea una copia de seguridad de un directorio en un archivo comprimido con la fecha actual.

```
#!/bin/bash
# script_backup.sh

echo "Iniciando copia de seguridad..."

# Definir el directorio a respaldar
SOURCE_DIR="/home/usuario/Documentos"
# Definir el directorio de destino para el backup
BACKUP_DIR="/home/usuario/Backups"
# Nombre del archivo de backup con la fecha actual
DATE=$(date +%Y%m%d_%H%M%S)
BACKUP_FILE="$BACKUP_DIR/documentos_backup_$DATE.tar.gz"

# Crear el directorio de backups si no existe
mkdir -p "$BACKUP_DIR"

# Comprimir y copiar el directorio
```

```
tar -czf "$BACKUP_FILE" "$SOURCE_DIR"

echo "Copia de seguridad completada en: $BACKUP_FILE"
```

Ejercicios Prácticos

Ejercicio 1: Organización de Archivos por Tipo

Objetivo: Crear un script que organice archivos de un tipo específico en una nueva carpeta.

Instrucciones:

1. Abre tu terminal (CMD en Windows o Bash en Linux/macOS).
2. Navega a un directorio donde tengas algunos archivos de prueba (ej. `.jpg`, `.png`, `.txt`).
3. Crea un nuevo archivo de script:
    - **Windows:** `organizar_imagenes.bat`
    - **Linux/macOS:** `organizar_imagenes.sh`
4. Escribe el código en el script para realizar las siguientes tareas:
    - Crear una carpeta llamada `MisImagenes`.
    - Mover todos los archivos con extensión `.jpg` y `.png` del directorio actual a la carpeta `MisImagenes`.
    - Mostrar un mensaje en la terminal confirmando que los archivos han sido movidos.
    - **(Windows)** Añade `pause` al final para que la ventana no se cierre inmediatamente.

Ejercicio 2: Monitoreo Básico de Sistema

Objetivo: Crear un script que muestre información básica del sistema y la registre en un archivo de log.

Instrucciones:

1. Abre tu terminal.
2. Crea un nuevo archivo de script:
    - **Windows:** `info_sistema.bat`
    - **Linux/macOS:** `info_sistema.sh`
3. Escribe el código en el script para realizar las siguientes tareas:
    - Mostrar la fecha y hora actual.
    - Mostrar el espacio libre en disco de la unidad principal (ej. `C:` en Windows o el sistema de archivos raíz `/` en Linux).
    - Redirigir toda esta información a un archivo de texto llamado `monitoreo.log` (asegúrate de que la nueva información se añada al final del archivo si ya existe).
    - **(Windows)** Añade `pause` al final.

Comandos útiles para el Ejercicio 2:

- **Fecha y Hora:**
    - Windows: `date /t` y `time /t`
    - Linux/macOS: `date`
- **Espacio en Disco:**
    - Windows: `wmic logicaldisk get caption,freespace,size` o `fsutil volume diskfree C:`
    - Linux/macOS: `df -h /`