⚙️ Taller Práctico: Gestión de Procesos y Control de Trabajos

📝 Ejercicio: Lanzar procesos en segundo plano (Background)

- **Objetivo:** Ejecutar una tarea sin bloquear la terminal para poder seguir trabajando.
- **Paso a paso:** Ejecuta `sleep 1000 &`
- **Explicación:** El comando `sleep` simplemente espera el tiempo indicado (en segundos). Al añadir el símbolo `&` al final, le indicamos a Bash que lo ejecute en segundo plano. La terminal nos devolverá un número de trabajo (ej: `[1]`) y un PID (identificador de proceso). ⏳

📝 Ejercicio: Listar trabajos actuales (`jobs`)

- **Objetivo:** Ver qué tareas tenemos activas únicamente en la sesión actual del shell.
- **Paso a paso:** Ejecuta `jobs`
- **Explicación:** A diferencia de otros comandos, `jobs` solo muestra los procesos que han sido lanzados desde esa terminal específica. Verás el estado (Running/Stopped) y el número de trabajo. 📋

📝 Ejercicio: Recuperar un proceso al primer plano (`fg`)

- **Objetivo:** Traer una tarea que está en el fondo (background) al frente (foreground) para interactuar con ella.
- **Paso a paso:** Ejecuta `fg %1` (asumiendo que es el trabajo número 1).
- **Explicación:** El comando `fg` (Foreground) toma el control de la terminal para ese proceso. Ahora verás que no puedes escribir otros comandos hasta que el `sleep` termine o lo detengas. ⬆️

📝 Ejercicio: Suspender y enviar al fondo (`Ctrl+Z` y `bg`)

- **Objetivo:** Pausar un programa que está en primer plano y dejar que siga trabajando por detrás.
- **Paso a paso:**
  1. Con el proceso en primer plano, pulsa `Ctrl + Z`.
  2. Ejecuta `bg %1`
- **Explicación:** `Ctrl + Z` suspende (pausa) el proceso. Con `bg` (Background), le ordenamos que se reanude pero manteniéndose en segundo plano. ⏸️➡️▶️

📝 Ejercicio: Monitorización global con `ps`

- **Objetivo:** Ver todos los procesos del sistema, no solo los de nuestra terminal.
- **Paso a paso:** Ejecuta `ps aux | grep sleep`

- **Explicación:** `ps aux` muestra una "foto" instantánea de todos los procesos de todos los usuarios. Usamos el pipe y `grep` para filtrar y encontrar específicamente nuestro comando `sleep`. 🔍

📝 Ejercicio: Monitorización en tiempo real con `top`

- **Objetivo:** Identificar qué procesos consumen más CPU y memoria en vivo.
- **Paso a paso:** 1. Ejecuta `top` 2. Pulsa `M` para ordenar por memoria o `P` para ordenar por CPU. 3. Pulsa `q` para salir.
- **Explicación:** `top` es el administrador de tareas clásico de Linux. Se actualiza cada pocos segundos y es vital para diagnosticar lentitud en el servidor. 📊

📝 Ejercicio: Finalizar procesos de forma amistosa (`kill`)

- **Objetivo:** Detener un proceso usando su identificador numérico (PID).
- **Paso a paso:** 1. Busca el PID de tu sleep: `ps` 2. Ejecuta `kill 1234` (sustituye 1234 por el PID real).
- **Explicación:** `kill` envía una señal (por defecto SIGTERM) al proceso pidiéndole que cierre sus archivos y finalice. Es la forma segura de detener programas. 🔫

📝 Ejercicio: Cierre forzado de procesos (`kill -9`)

- **Objetivo:** Obligar a un proceso "colgado" o "zombie" a cerrarse inmediatamente.
- **Paso a paso:** Ejecuta `kill -9 1234`
- **Explicación:** La señal `-9` (SIGKILL) no puede ser ignorada por el proceso. El kernel lo elimina de la memoria de inmediato. Solo debe usarse si el `kill` normal no funciona. 💀

📝 Ejercicio: Matar por nombre de proceso (`pkill`)

- **Objetivo:** Detener todos los procesos que compartan un nombre sin buscar sus PIDs.
- **Paso a paso:** Ejecuta `pkill sleep`
- **Explicación:** `pkill` es muy útil cuando tienes muchas instancias de un mismo programa (como varios navegadores o procesos de base de datos) y quieres cerrarlos todos a la vez. 🧹

📝 Ejercicio: Prioridad de procesos (`nice`)

- **Objetivo:** Lanzar un proceso con menor prioridad para que no ralentice el resto del sistema.
- **Paso a paso:** Ejecuta `nice -n 19 sleep 500 &`
- **Explicación:** El valor de "niceness" va de -20 (máxima prioridad) a 19 (mínima). Un proceso "amable" (nice) deja que los demás usen la CPU antes que él. 😇