Ejercicio: llamar desde Kotlin a un proceso en Linux con un script en Bash

1. Crea un archivo llamado script.sh con el siguiente contenido:

[crayon-6a9d57a3656d9692002257/]

2. En Kotlin, puedes usar la clase ProcessBuilder para ejecutar el script en Bash. Asegúrate de tener el archivo script.sh en la misma ubicación que tu código Kotlin.

[crayon-6a9d57a3656e1632605465/]

Este código en Kotlin utiliza la clase ProcessBuilder para ejecutar un script de shell (script.sh) con un argumento desde un programa Kotlin, y luego captura e imprime la salida del script. A continuación, se explica el código línea por línea:

1. **Importación de la clase File**: import java.io.File Se importa la clase File de la biblioteca estándar de Java, que se utiliza para manejar archivos.
2. **Función principal main**: fun main() { Se define la función main, que es el punto de entrada del programa.
3. **Creación del objeto File para el script**: val scriptFile = File("script.sh") Se crea un objeto File que representa el archivo del script (script.sh). Esto asume que script.sh está en el mismo directorio desde donde se ejecuta el programa.
4. **Definición del comando a ejecutar**: val command = "bash ${scriptFile.absolutePath} argumento" Se define una

cadena de caracteres command que contiene el comando a ejecutar. Utiliza bash para ejecutar el script de shell con su ruta absoluta y le pasa un argumento (argumento).

5. **Creación del ProcessBuilder**: val processBuilder = ProcessBuilder("/bin/bash", "-c", command) Se crea un objeto ProcessBuilder para ejecutar el comando definido. El comando se ejecuta utilizando /bin/bash con la opción -c, que permite ejecutar un comando en forma de cadena de caracteres.
6. **Redirección del flujo de errores**: processBuilder.redirectErrorStream(true) Se configura el ProcessBuilder para redirigir el flujo de errores estándar (stderr) al flujo de salida estándar (stdout). Esto significa que cualquier error producido por el script se incluirá en la salida capturada.
7. **Inicio del proceso**: val process = processBuilder.start() Se inicia el proceso utilizando el ProcessBuilder.
8. **Esperar a que el proceso termine**: process.waitFor() Se espera a que el proceso termine antes de continuar. Esto asegura que el programa principal no continúe antes de que el proceso hijo haya completado su ejecución.
9. **Captura de la salida del proceso**: val output = process.inputStream.bufferedReader().readText() Se lee la salida del proceso desde el flujo de entrada (inputStream) del proceso. Utiliza un BufferedReader para leer el texto completo de la salida.
10. **Imprimir la salida**: println(output) Se imprime la salida capturada del script.