Bash Scripting y Python

Redirecting streams

Para redirigir flujos de entrada y salida en Python, puedes usar los módulos sys y io. A continuación se muestra cómo redirigir la salida estándar a un archivo.

```
import sys

# Redirigir la salida estándar a un archivo
sys.stdout = open('output.txt', 'w')

print("Este texto se escribe en el archivo 'output.txt'.")

# Restaurar la salida estándar
sys.stdout = sys.__stdout__
```

Pipes and pipelines

Para implementar tuberías y pipelines en Python, puedes usar el módulo subprocess para ejecutar comandos y conectar sus entradas y salidas.

```
import subprocess

# Crear un pipeline que pasa la salida de 'echo' a 'grep'
p1 = subprocess.Popen(['echo', 'Hello World'], stdout=subprocess.PIPE)
p2 = subprocess.Popen(['grep', 'World'], stdin=p1.stdout, stdout=subprocess.PIPE)
p1.stdout.close()  # Permitir que p1 termine
output = p2.communicate()[0]

print(output.decode())
```

Signaling processes

Para enviar señales a procesos en Python, puedes usar el módulo os para enviar señales a procesos específicos.

```
import os
import signal
import time

# Proceso hijo que espera señales
pid = os.fork()
if pid == 0:
    while True:
        print("Esperando señales...")
        time.sleep(1)
else:
    # Esperar un momento y luego enviar una señal SIGTERM
    time.sleep(5)
    os.kill(pid, signal.SIGTERM)
    print(f"Señal SIGTERM enviada al proceso {pid}")
```

Creating bash scripts

En Python, puedes generar y ejecutar scripts de bash utilizando el módulo subprocess.

```
import subprocess

# Crear un archivo de script bash
script_content = """#!/bin/bash
echo "Este es un script Bash ejecutado desde Python"
"""

with open('script.sh', 'w') as file:
    file.write(script_content)

# Hacer el script ejecutable
subprocess.run(['chmod', '+x', 'script.sh'])
```

```
# Ejecutar el script
subprocess.run(['./script.sh'])
```

Using variables and globs

Para usar variables y globs en Python, puedes utilizar el módulo glob para la expansión de patrones en nombres de archivo.

```
import glob

# Usar globs para encontrar archivos .txt en el directorio actual
files = glob.glob('*.txt')

print("Archivos .txt en el directorio actual:")
for file in files:
    print(file)
```

Conditional execution in bash

Para la ejecución condicional de comandos en bash, puedes usar operadores lógicos como && y ||. Aquí un ejemplo en Python para crear un script bash con condicionales.

```
import subprocess

# Crear un script bash con ejecución condicional
script_content = """#!/bin/bash
if [ -f "file.txt" ]; then
    echo "file.txt existe"
else
    echo "file.txt no existe"
fi
"""

with open('conditional_script.sh', 'w') as file:
    file.write(script_content)
```

```
# Hacer el script ejecutable
subprocess.run(['chmod', '+x', 'conditional_script.sh'])

# Ejecutar el script
subprocess.run(['./conditional_script.sh'])
```

While loops in bash scripts

En bash, los bucles while se usan para ejecutar un bloque de código mientras se cumpla una condición. A continuación se muestra cómo crear un script bash con un bucle while desde Python.

```
import subprocess

# Crear un script bash con un bucle while
script_content = """#!/bin/bash
counter=0
while [ $counter -lt 5 ]; do
    echo "Contador: $counter"
    counter=$((counter + 1))
    sleep 1
done
"""

with open('while_loop_script.sh', 'w') as file:
    file.write(script_content)

# Hacer el script ejecutable
subprocess.run(['chmod', '+x', 'while_loop_script.sh'])

# Ejecutar el script
subprocess.run(['./while_loop_script.sh'])
```

For loops in bash scripts

Los bucles for en bash se utilizan para iterar sobre una lista de elementos. El siguiente código muestra cómo crear un script bash con un bucle for desde Python.

```
import subprocess

# Crear un script bash con un bucle for
script_content = """#!/bin/bash
for i in {1..5}; do
    echo "Número: $i"
done
"""

with open('for_loop_script.sh', 'w') as file:
    file.write(script_content)

# Hacer el script ejecutable
subprocess.run(['chmod', '+x', 'for_loop_script.sh'])

# Ejecutar el script
subprocess.run(['./for_loop_script.sh'])
```

Advanced command interaction

Para la interacción avanzada con comandos, puedes combinar subprocess con técnicas de procesamiento de entrada y salida. A continuación se muestra un ejemplo de interacción avanzada con comandos en Python.

```
import subprocess

# Ejecutar un comando y capturar su salida
process = subprocess.Popen(['grep', 'pattern'], stdin=subprocess.PIPE, stdout=subprocess.PIPE, text=True)
output, _ = process.communicate(input='line with pattern\nline without pattern')

print("Salida del comando grep:")
print(output)
```

Choosing between bash and python

Elegir entre Bash y Python depende del contexto. Bash es útil para scripts de shell y tareas de administración del sistema, mientras que Python ofrece más flexibilidad y potencia para aplicaciones complejas.

```
# Ejemplo en Bash
# !/bin/bash
echo "Hello from Bash"

# Ejemplo en Python
# print("Hello from Python")

# Ambas opciones tienen sus casos de uso específicos.
```