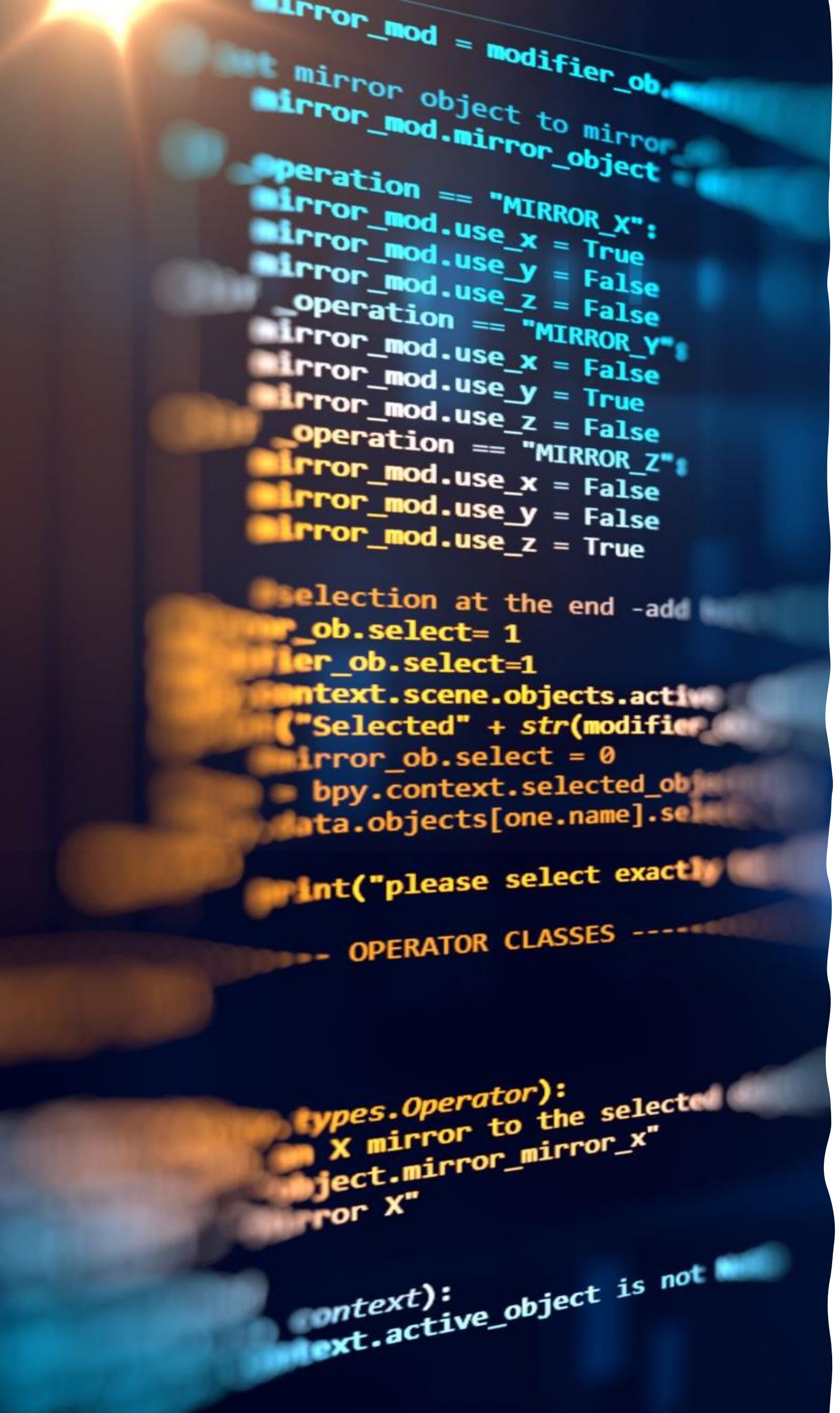


Bloque 1 –Tema 2. Bash Scripting.

Automatización y Optimización de Flujos de Trabajo para Data Engineering

Bash

Bash significa 'Bourne Again Shell’

Bash es el intérprete de comandos predeterminado en la mayoría de los sistemas basados en Unix (Linux, macOS).

Su potencia reside en la capacidad de automatizar tareas repetitivas y optimizar procesos en flujos de trabajo de datos.

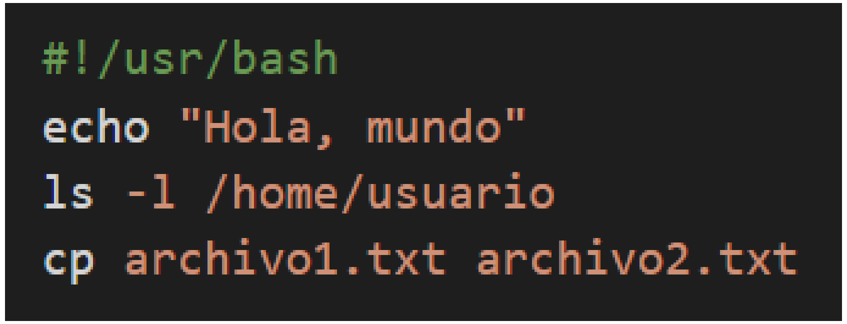
Permite integrar múltiples herramientas del ecosistema Unix y gestionar procesos, archivos y sistemas de almacenamiento de manera eficiente.

Bash script anatomy

Un script de Bash tiene algunas características clave:

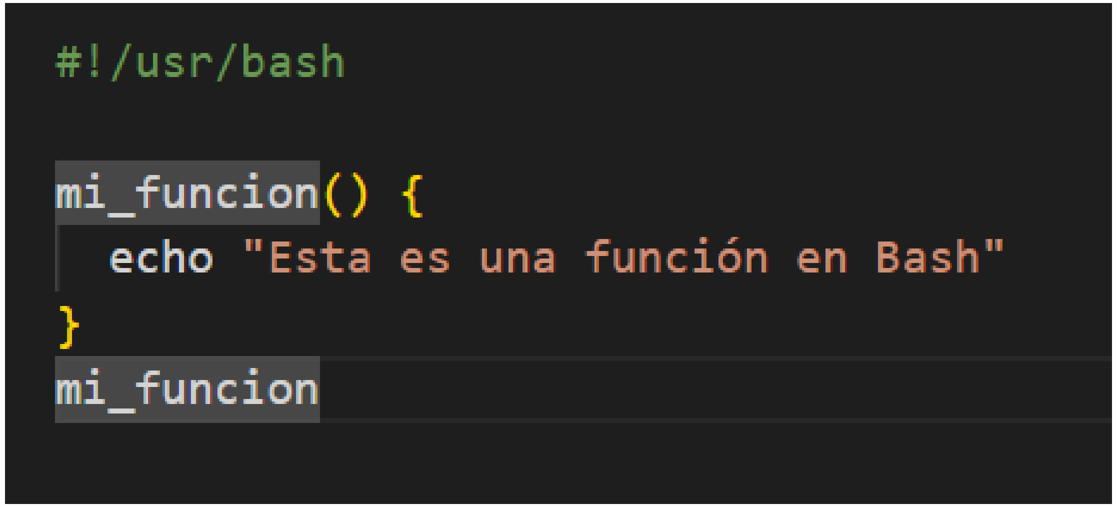
- Generalmente comienza con #!/usr/bash (en su propia línea)
- Esto indica al intérprete que es un script de Bash y que debe usar Bash ubicado en /usr/bash.
- Esta ruta podría ser diferente si instalaste Bash en otro lugar, como /bin/bash. (Escribe which bash para verificar la ubicación).
- Las líneas intermedias contienen el código. Esto puede ser comandos línea por línea o estructuras de programación.

Bash script anatomy



```bash
#!/usr/bash
echo "Hola, mundo"
ls -l /home/usuario
cp archivo1.txt archivo2.txt
```

```bash
#!/usr/bash

if [ -f "archivo.txt" ]; then
  echo "El archivo existe"
else
  echo "El archivo no existe"
fi
```



```bash
#!/usr/bash

mi_funcion() {
  echo "Esta es una función en Bash"
}
mi_funcion
```

Bash script anatomy

Para guardar y ejecutar un script:

- Tiene una extensión de archivo .sh. Técnicamente no es necesario si la primera línea tiene el shebang y la ruta a Bash (#!/usr/bash), pero es una convención.
- Se puede ejecutar en la terminal usando bash nombre_del_script.sh.
- O si has especificado la primera línea (#!/usr/bash), simplemente puedes ejecutar usando ./nombre_del_script.sh.

Los scripts se utilizan para automatizar tareas repetitivas, lo que permite ahorrar tiempo y esfuerzo. Algunas ventajas son:

Automatización: Ahorran tiempo y esfuerzo al automatizar tareas repetitivas.

Eficiencia: Permiten realizar tareas complejas de forma rápida y sencilla.

Precisión: Reducen el riesgo de errores al ejecutar comandos manualmente.

Flexibilidad: Se pueden personalizar para adaptarse a tus necesidades específicas.

Aplicacioness

En ingeniería de datos, los scripts de Bash son una herramienta fundamental para automatizar, optimizar y orquestar tareas repetitivas en el procesamiento y manejo de datos. Aquí están algunos casos comunes en los que los scripts de Bash se utilizan:

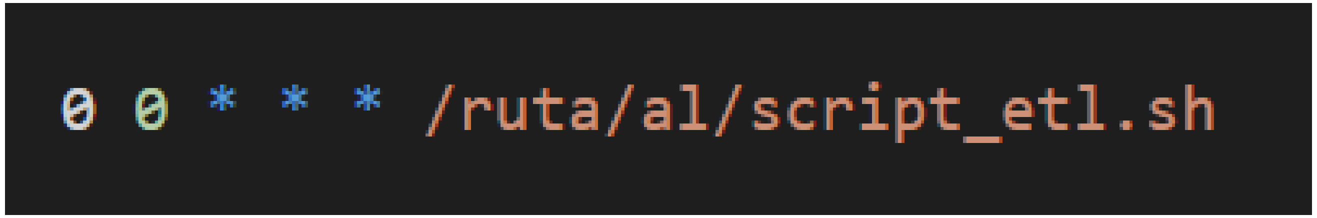
1. Automatización de Flujos de Trabajo (ETL)

- Extracción: Los scripts de Bash se utilizan para extraer datos de diversas fuentes, como bases de datos, archivos CSV, API o sistemas de archivos distribuidos (HDFS). Comandos como curl, wget o utilidades para bases de datos (mysql, psql) son comunes.
- Transformación: Procesar archivos para limpiarlos, formatearlos o integrarlos con otras fuentes de datos. Herramientas como awk, sed, o incluso Python, se combinan en scripts de Bash para transformar los datos en el formato adecuado.
- Carga: Mover datos transformados a un almacén de datos, data lake o base de datos. Comandos como scp, rsync, o utilidades de bases de datos se usan para cargar datos en sus ubicaciones finales.

2. Programación de Tareas Periódicas

Bash se usa junto con herramientas de programación de tareas, como cron, para ejecutar scripts periódicamente. Por ejemplo, puede ser útil programar un script que recolecte y procese datos todos los días a una hora específica.

Ejemplo de uso de cron para ejecutar un script diariamente a la medianoche:

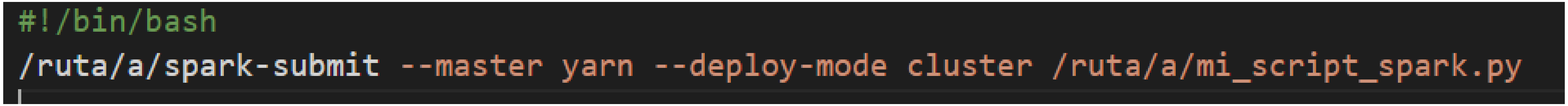


```
0 0 * * * /ruta/al/script_etl.sh
```

3. Integración con Big Data y Ecosistemas en la Nube

Bash permite interactuar con herramientas de Big Data como Hadoop, Spark, o herramientas de la nube (AWS, GCP, Azure). Scripts de Bash se utilizan para enviar trabajos a clusters de procesamiento, mover datos entre sistemas distribuidos y ejecutar pipelines de datos.

Ejemplo: Ejecutar un trabajo de Spark con spark-submit desde un script de Bash:



```
#!/bin/bash
/ruta/a/spark-submit --master yarn --deploy-mode cluster /ruta/a/mi_script_spark.py
```

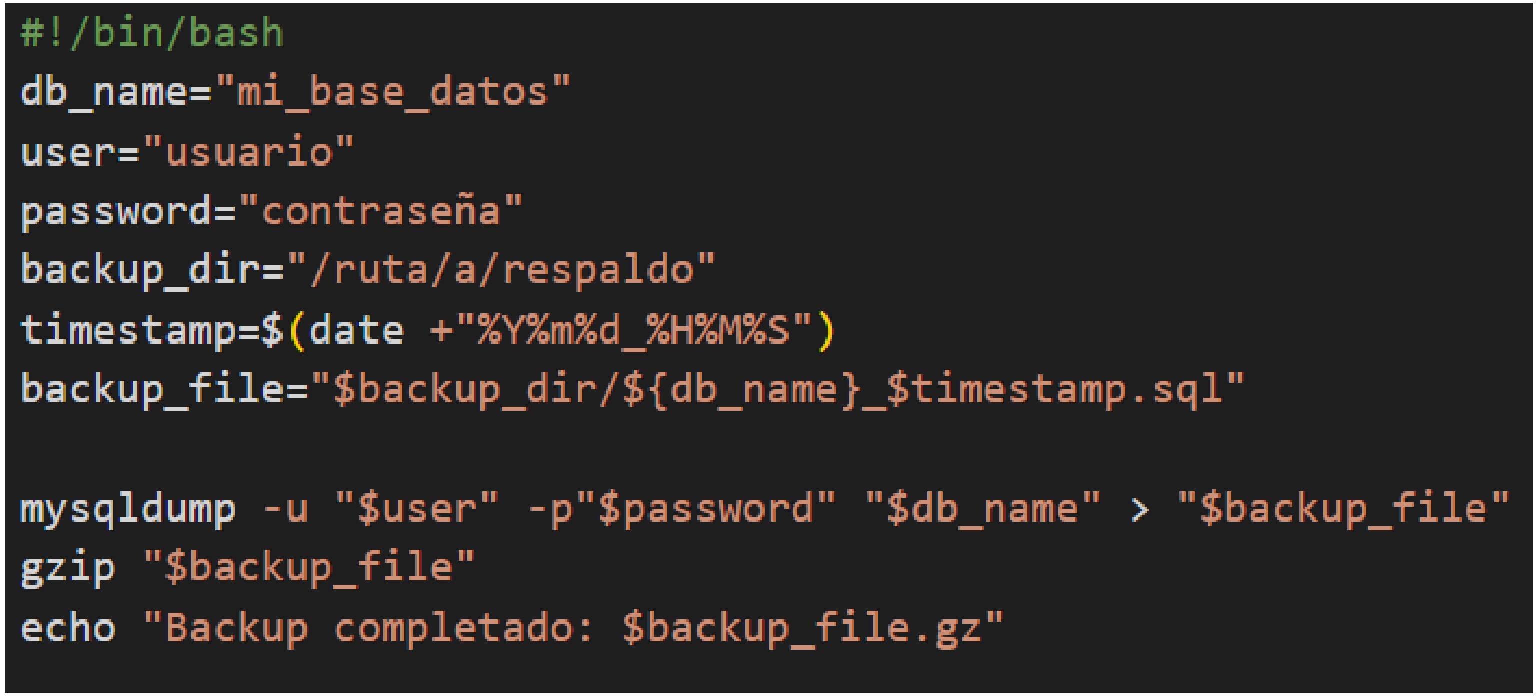
4. Manipulación de Archivos y Datos

- Procesamiento de Archivos de Log: Filtrar, buscar patrones o dividir archivos de log en función de ciertos criterios utilizando grep, awk, y sed. Estos scripts pueden ser usados para analizar grandes volúmenes de logs generados por sistemas distribuidos.
- Consolidación y Compresión de Datos: Agrupar y comprimir archivos para archivar o mover grandes volúmenes de datos. Por ejemplo, usar tar para comprimir y gzip para reducir el tamaño de archivos de datos.
- Renombrar o reorganizar archivos: Los scripts de Bash pueden renombrar y mover archivos en función de fechas, patrones de nombres o contenido del archivo.

5. Automatización del Backup y Recuperación de Datos

Los scripts de Bash se usan para programar respaldos automáticos de bases de datos, sistemas de archivos y servidores. Se pueden combinar con herramientas de compresión (gzip, tar) y sincronización (rsync) para crear copias de seguridad incrementales o completas.

Ejemplo de script de respaldo de una base de datos MySQL



```bash
#!/bin/bash
db_name="mi_base_datos"
user="usuario"
password="contraseña"
backup_dir="/ruta/a/respaldo"
timestamp=$(date +"%Y%m%d_%H%M%S")
backup_file="$backup_dir/${db_name}_$timestamp.sql"

mysqldump -u "$user" -p"$password" "$db_name" > "$backup_file"
gzip "$backup_file"
echo "Backup completado: $backup_file.gz"
```

6. Orquestación y Ejecución de Pipelines

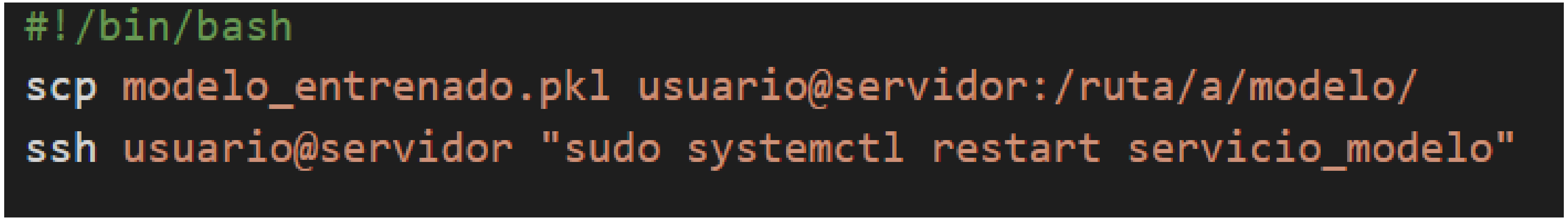
Bash se usa para orquestar la ejecución de múltiples scripts o trabajos, facilitando la creación de pipelines complejos que incluyen procesamiento de datos, limpieza, entrenamiento de modelos de machine learning, y almacenamiento de resultados.

Estos scripts pueden coordinar la ejecución de diferentes lenguajes o entornos (como Python, R, Spark), permitiendo integrar diferentes etapas de procesamiento en un flujo continuo.

7. Despliegue de Modelos de Machine Learning

Los scripts de Bash pueden automatizar el proceso de despliegue de modelos, gestionando la transferencia de archivos, configuración de entornos, y ejecución de servicios.

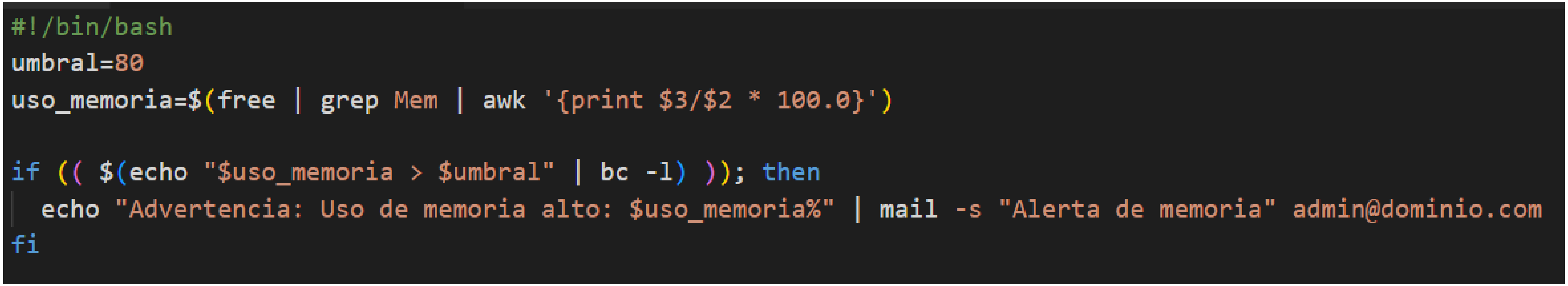
Ejemplo de uso: Copiar un modelo entrenado a un servidor remoto y reiniciar el servicio que lo usa.



```bash
#!/bin/bash
scp modelo_entrenado.pkl usuario@servidor:/ruta/a/modelo/
ssh usuario@servidor "sudo systemctl restart servicio_modelo"
```

8. Monitoreo y Alertas

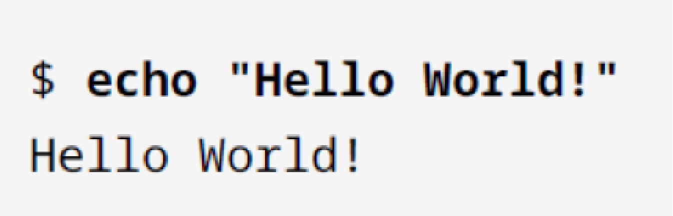
Bash se utiliza para crear scripts que monitorizan servicios, rendimiento de sistemas, o la disponibilidad de datos. Los resultados se pueden usar para generar alertas por correo electrónico o integrarse con herramientas de monitoreo (como Prometheus). Ejemplo de monitoreo de uso de memoria:



```bash
#!/bin/bash
umbral=80
uso_memoria=$(free | grep Mem | awk '{print $3/$2 * 100.0}')

if (( $(echo "$uso_memoria > $umbral" | bc -l) )); then
  echo "Advertencia: Uso de memoria alto: $uso_memoria%" | mail -s "Alerta de memoria" admin@dominio.com
fi
```

Imprimiendo salidas (Printing Output)

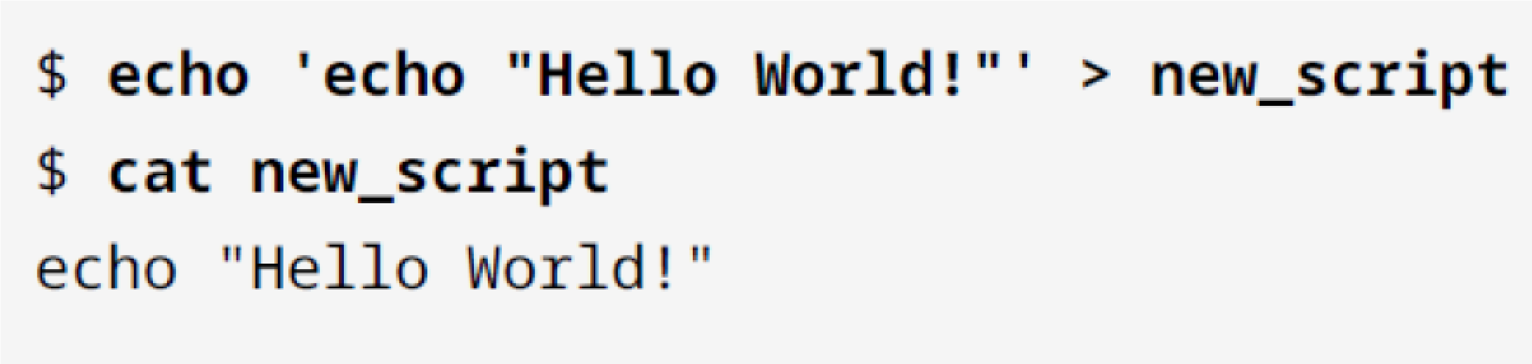
Comencemos construyendo un ejemplo con el comando **echo** el cual imprime un argumento en la salida estándar.



```
$ echo "Hello World!"
Hello World!
```

Ahora, usaremos la redirección de archivos para enviar este comando a un nuevo archivo llamado **new_script**.

Imprimiendo salidas (Printing Output)

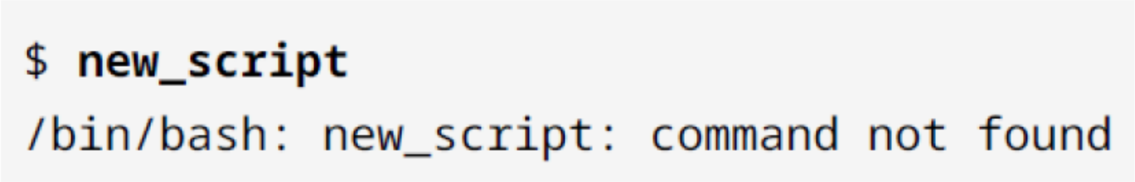


```
$ echo 'echo "Hello World!"' > new_script
$ cat new_script
echo "Hello World!"
```

El archivo new_script contiene ahora el mismo comando que antes.

Cómo ejecutar un script

Vamos a demostrar algunos de los pasos necesarios para que este archivo se ejecute de la forma en que esperamos, el primer pensamiento de un usuario podría ser simplemente escribir el nombre del script. La forma en que ellos podrían escribir el nombre de cualquier otro comando:

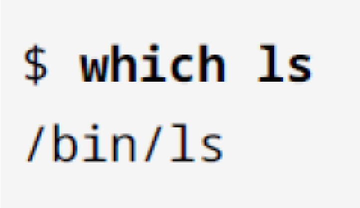


```
$ new_script
/bin/bash: new_script: command not found
```

Podemos asumir con seguridad que new_script existe en nuestra ubicación actual, pero note que el mensaje de error no nos está diciendo que el archivo no existe, sino que nos está diciendo que el comando no existe.

Comandos y PATH

Cuando escribimos el comando **ls** en la shell, por ejemplo, estamos ejecutando un archivo llamado ls que existe en nuestro sistema de archivos:



```
$ which ls
/bin/ls
```

Rápidamente se volvería tedioso escribir la ruta absoluta de ls cada vez que deseemos ver el contenido de un directorio, así que Bash tiene una variable de entorno (environment) que contiene todos los directorios en los que podemos encontrar los comandos que deseamos ejecutar.

El comando which en Linux se utiliza para encontrar la ubicación del archivo ejecutable de un comando o programa. En otras palabras, te dice dónde se encuentra en el sistema de archivos el comando que estás intentando ejecutar.

Comandos y PATH

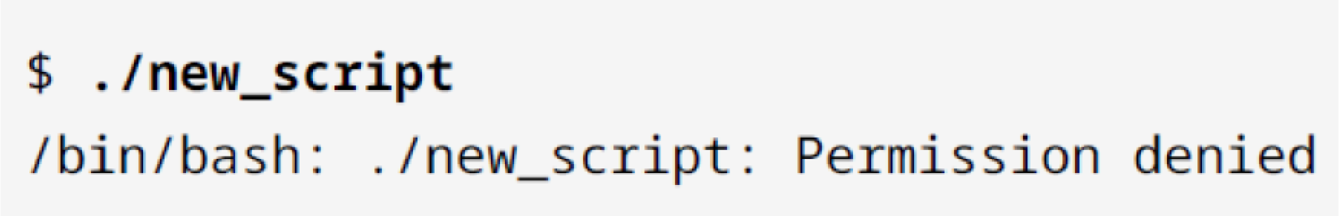
Cada una de estas ubicaciones es donde el shell espera encontrar un comando, delimitado con dos puntos lo está.

```
$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/games:/usr/local/games:/sn
ap/bin
```

La shell buscará new_script en cada uno de estos directorios, pero no lo encontrará y por lo tanto lanzará el error que vimos antes.

Comandos y PATH

Hay tres soluciones para este problema: podemos **mover** new_script a uno de los directorios PATH, podemos **añadir** nuestro directorio actual a PATH, o podemos **cambiar** la forma en que intentamos llamar al script, esta última solución es la más sencilla, simplemente requiere que especifiquemos la ubicación actual cuando llamemos al script usando la barra de puntos (./).

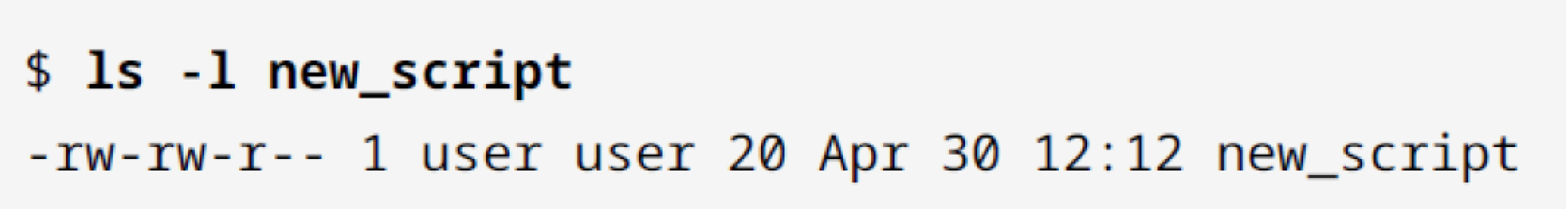


```
$ ./new_script
/bin/bash: ./new_script: Permission denied
```

El mensaje de error ha cambiado, lo que indica que hemos hecho algunos progresos.

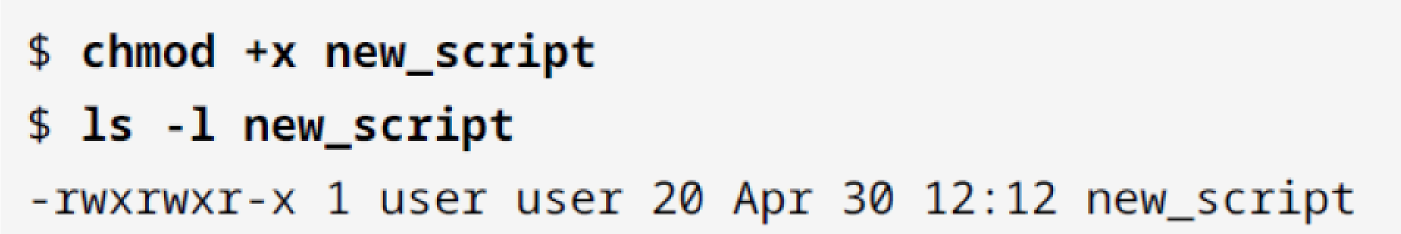
Ejecutar Permisos

La primera indagación que un usuario debe hacer en este caso es usar ls -l para mirar el archivo:



```
$ ls -l new_script
-rw-rw-r-- 1 user user 20 Apr 30 12:12 new_script
```

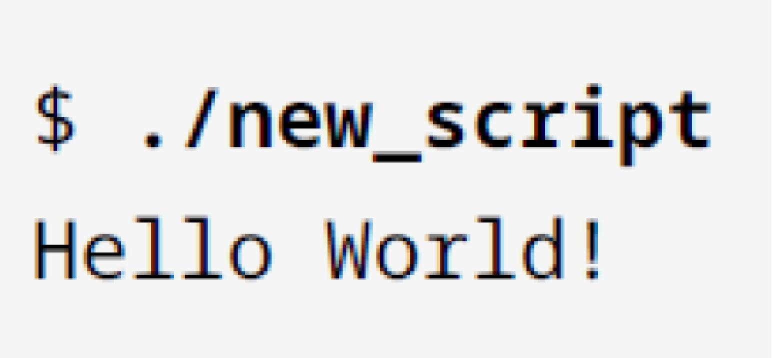
Podemos ver que los permisos para este archivo están configurados para que ningún usuario tenga permisos de ejecución. Por lo tanto:



```
$ chmod +x new_script
$ ls -l new_script
-rwxrwxr-x 1 user user 20 Apr 30 12:12 new_script
```

Ejecutar Permisos

Este comando ha dado permisos de ejecución a todos los usuarios, tenga en cuenta que esto puede ser un riesgo para la seguridad, pero por ahora es un nivel aceptable de permisos.



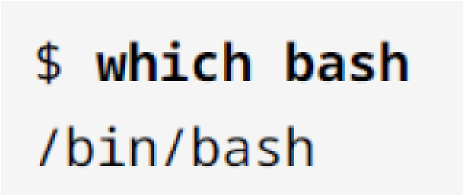
```
$ ./new_script
Hello World!
```

Ahora podemos ejecutar nuestro script.

Definición del intérprete

Como hemos demostrado, hemos podido simplemente introducir texto en un archivo, configurarlo como ejecutable y ejecutarlo. El new_script es funcionalmente un archivo de texto normal, pero logramos que sea interpretado por Bash, pero ¿qué pasa si está escrito en Python?

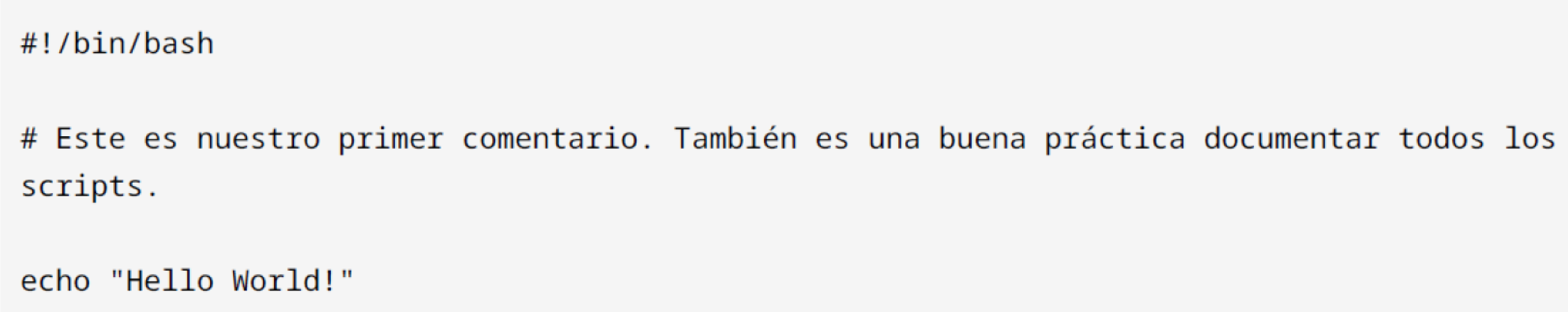
Es muy recomendable especificar el tipo de intérprete que queremos usar en la primera línea de un script, este se llama **bang line** o más conocido como **shebang**; e indica al sistema cómo queremos que se ejecute este archivo. Como estamos aprendiendo Bash, estaremos usando la ruta absoluta a nuestro ejecutable Bash, una vez más usando **which**:



```
$ which bash
/bin/bash
```

Definición del intérprete

Nuestro **shebang** comienza con un signo de hash (#) y un signo de exclamación de cierre (!), seguido de la ruta absoluta que vimos antes (/bin/bash). Ahora abrimos **new_script** en un editor de texto e insertemos el **shebang**, y aprovechemos la oportunidad para insertar un comentario en nuestro script, ya que los comentarios son ignorados por el intérprete y están escritos para el beneficio de otros usuarios que deseen entender su script.



```
#!/bin/bash

# Este es nuestro primer comentario. También es una buena práctica documentar todos los
scripts.

echo "Hello World!"
```

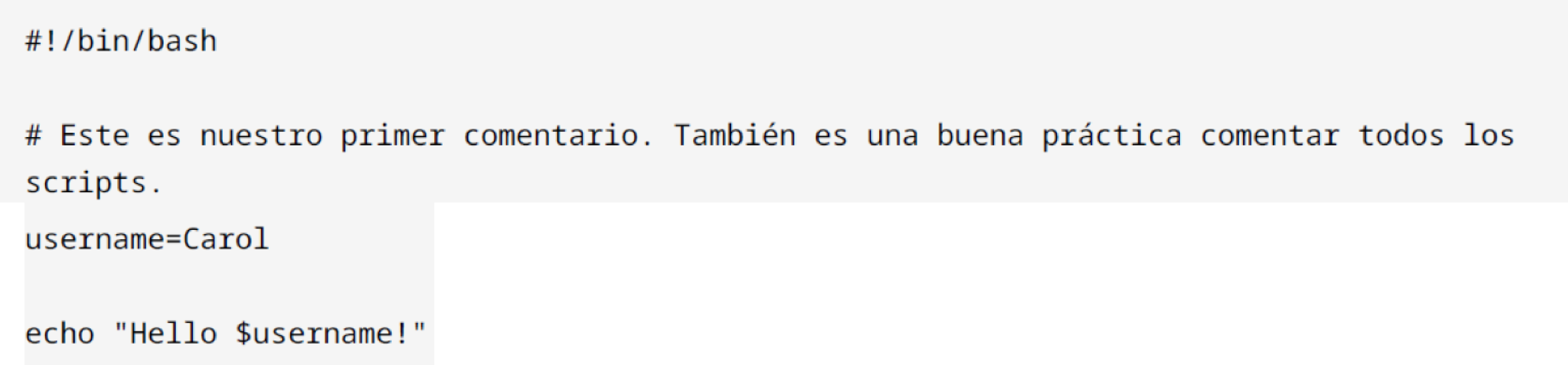
Definición del intérprete

También haremos un cambio adicional al nombre del archivo: guardaremos este archivo como new_script.sh. El sufijo del archivo **.sh** no cambia la ejecución del archivo de ninguna manera, es una convención que los scripts bash sean etiquetados con **.sh** o **.bash** para identificarlos fácilmente, así como los scripts Python son usualmente identificados con el sufijo **.py**.

Variables

Las variables son una parte importante de cualquier lenguaje de programación y **Bash** no es diferente, cuando se inicia una nueva sesión desde la terminal, la shell ya establece algunas variables, como por ejemplo la variable PATH, a la que llamamos **variables de entorno**, ya que suelen definir las características de nuestro entorno de shell, por lo que se pueden modificar y añadir variables de entorno, pero por ahora vamos a centrarnos en la configuración de variables dentro de nuestro script.

Modificaremos nuestro script para que se vea así:



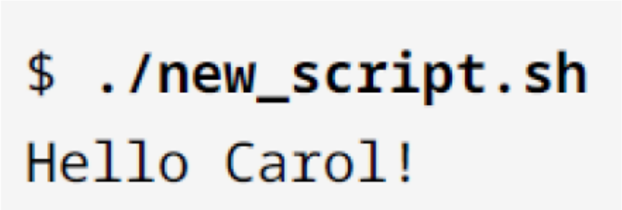
```
#!/bin/bash

# Este es nuestro primer comentario. También es una buena práctica comentar todos los scripts.
username=Carol

echo "Hello $username!"
```

En este caso, hemos creado una variable llamada username y le hemos asignado el valor de Carol. Tenga en cuenta que no hay espacios entre el nombre de la variable, el signo igual o el valor asignado.

Modificaremos nuestro script para que se vea así:

En la siguiente línea, hemos utilizado el comando echo con la variable, pero hay un signo de dólar ($) adelante del nombre de la variable, esto es importante, ya que indica a la shell que queremos tratar el nombre de usuario como una variable y no sólo como una palabra normal. Al introducir $username en nuestro comando, indicamos que queremos realizar una sustitución, reemplazando el nombre de una variable por el valor asignado a esa variable.

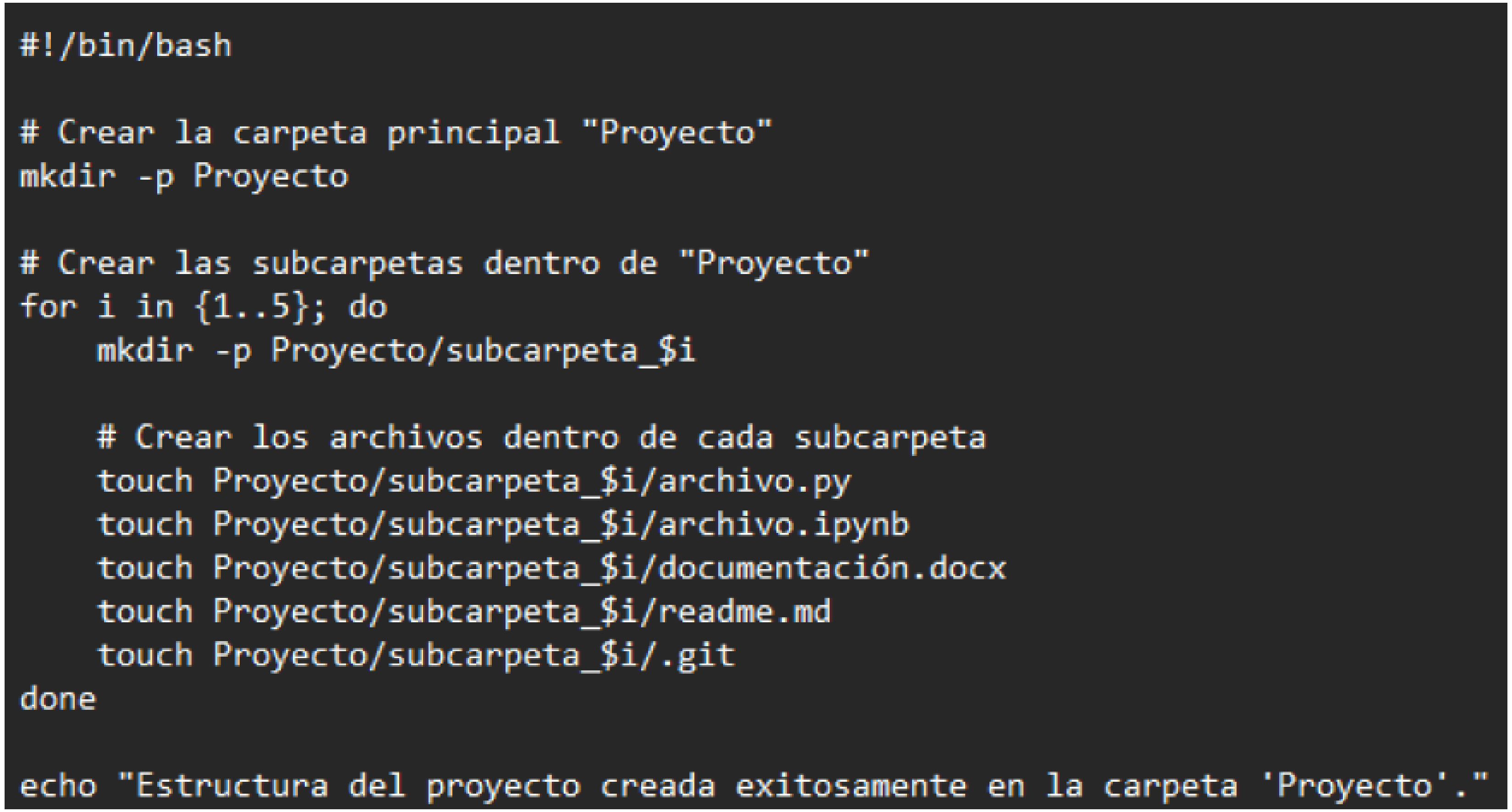
Ejecutando el nuevo script, obtenemos esta salida:



```
$ ./new_script.sh
Hello Carol!
```

Ejemplo básico de un script para crear un directorio

Vamos a crear una carpeta (desde la terminal) dentro del directorio practica que se llame: ej_script. Con ayuda del editor nano crear un script que contenga este código:

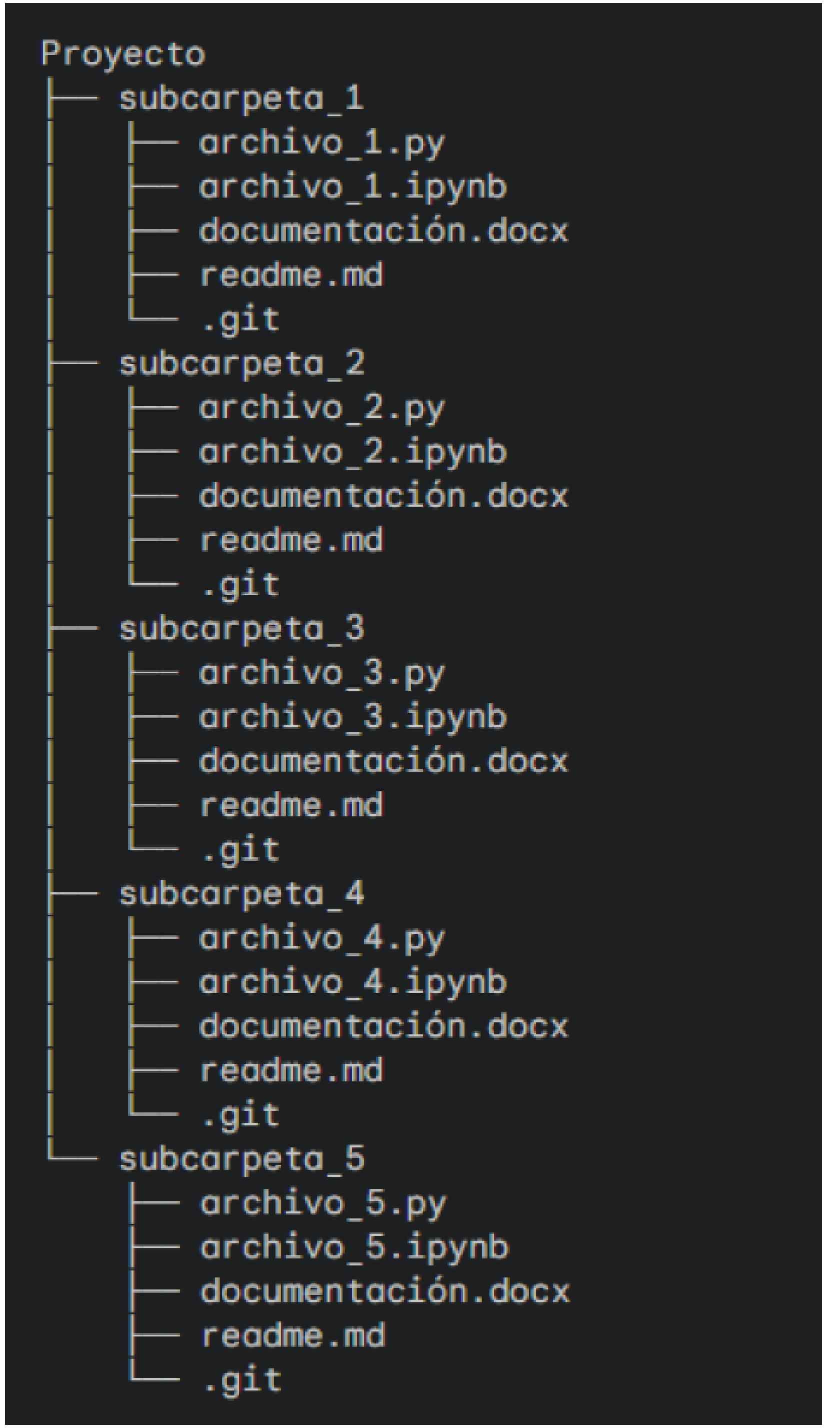


```
#!/bin/bash

# Crear la carpeta principal "Proyecto"
mkdir -p Proyecto

# Crear las subcarpetas dentro de "Proyecto"
for i in {1..5}; do
    mkdir -p Proyecto/subcarpeta_$i

    # Crear los archivos dentro de cada subcarpeta
    touch Proyecto/subcarpeta_$i/archivo.py
    touch Proyecto/subcarpeta_$i/archivo.ipynb
    touch Proyecto/subcarpeta_$i/documentación.docx
    touch Proyecto/subcarpeta_$i/readme.md
    touch Proyecto/subcarpeta_$i/.git
done

echo "Estructura del proyecto creada exitosamente en la carpeta 'Proyecto'."
```

Ejemplo básico de un script para crear un directorio

- Guardar el script con nombre carpetas.sh y salir del editor nano.
- Una vez en el terminal, verificar los permisos del archivo carpetas.sh
- Dar los permisos de ejecución con chmod
- Ejecutar el script escribiendo: ./carpetas.sh

La estructura de ese sistema de ficheros es esta:



```
Proyecto
├── subcarpeta_1
│   ├── archivo_1.py
│   ├── archivo_1.ipynb
│   ├── documentación.docx
│   ├── readme.md
│   └── .git
├── subcarpeta_2
│   ├── archivo_2.py
│   ├── archivo_2.ipynb
│   ├── documentación.docx
│   ├── readme.md
│   └── .git
├── subcarpeta_3
│   ├── archivo_3.py
│   ├── archivo_3.ipynb
│   ├── documentación.docx
│   ├── readme.md
│   └── .git
├── subcarpeta_4
│   ├── archivo_4.py
│   ├── archivo_4.ipynb
│   ├── documentación.docx
│   ├── readme.md
│   └── .git
└── subcarpeta_5
    ├── archivo_5.py
    ├── archivo_5.ipynb
    ├── documentación.docx
    ├── readme.md
    └── .git
```

Para ver la estructura se puede escribir en el terminal el comando: **tree -a.**

Actividad 12.

Esta actividad contiene dos ejercicios prácticos. Realizar la entrega en un fichero comprimido.zip.

Ejercicio práctico 1.

Abrir el fichero **estructura.txt.** Crear un script en bash que permita la creación automatizada de todo el directorio del proyecto mostrado incluyendo los ficheros (estos deben estar vacíos)

Ejercicio práctico 2

Caso de Uso: "Automatización en DataCorp“.

DataCorp es una empresa que gestiona datos provenientes de distintas fuentes para su análisis y almacenamiento en un entorno de Big Data. Para mejorar la eficiencia, necesitan scripts en Bash que automaticen las tareas diarias de administración de archivos y gestión de usuarios. Se deberán crear scripts que cubran tareas como la creación de directorios, copias de seguridad, configuración de permisos y gestión de usuarios, entre otros.

2.1: Creación de Directorios para la Ingesta de Datos

DataCorp recibe datos diariamente desde distintas fuentes. Crea un script que automatice la creación de directorios llamados ingesta/<fecha> con la fecha actual en formato YYYY-MM-DD, donde se almacenarán los datos del día. El script debe ejecutarse cada día antes de la llegada de los datos.

2.2: Copia de Seguridad Automática

Los archivos de datos procesados deben ser respaldados antes de ser archivados. Crea un script que copie todos los archivos de la carpeta procesados/ a una carpeta llamada backup/, y añade la fecha actual al nombre del archivo para evitar sobreescrituras.

El script debe usar el comando cp para copiar los archivos desde procesados/ a backup/, y añadir la fecha al nombre del archivo.

2.3: Mover Archivos por Tamaño

Los archivos que superan los 100 MB deben ser movidos automáticamente a un directorio llamado grandes/, y los menores de 100 MB deben ser movidos a pequeños/. Crea un script que automatice esta tarea.

Para mover los archivos mayores a 100 MB a grandes/ y los menores a pequeños/, el script puede utilizar find junto con mv.

2.4 : Creación de Usuarios del Equipo de Análisis

Cada miembro del equipo de análisis necesita una cuenta en el sistema. Crea un script que lea una lista de nombres de usuarios desde un archivo de texto (usuarios.txt) y cree esos usuarios en el sistema con sus respectivos directorios de inicio.

El script debe leer cada nombre de usuario del archivo usuarios.txt y usar el comando useradd para crear la cuenta:

2.5. Configuración de Permisos para Grupos de Usuarios en DataCorp

DataCorp tiene seis grupos de usuarios:

Analistas
Ingenieros
Administradores
Científicos de Datos
Ingenieros de Machine Learning (ML)
Ingenieros de AI

Cada grupo tiene diferentes niveles de acceso a los datos y scripts en el sistema. El ejercicio consiste en crear un script que configure los permisos para cada grupo de la siguiente manera:

Grupos de usuarios

- Grupo "Analistas":

Solo debe tener permisos de lectura en el directorio ingesta/.No debe tener acceso al directorio scripts/ ni a los datos procesados.

- Grupo "Ingenieros":

Debe tener permisos de lectura y escritura en el directorio procesados/ para poder trabajar con los datos transformados.

También debe tener permisos de lectura en scripts/ para ver los scripts que ejecutan los procesos de ETL.

Grupos de usuarios

- Grupo "Administradores":

Debe tener todos los permisos (lectura, escritura, ejecución) en todos los directorios (ingesta/, procesados/, scripts/, etc.)

- Grupo "Científicos de Datos":

Debe tener permisos de lectura y escritura en el directorio datos_procesados/ para acceder a los datos limpios.

Permisos de lectura y escritura en notebooks/ para desarrollar análisis y modelos.

Permisos de lectura en logs/ para revisar errores o mensajes importantes

Grupos de usuarios

- Grupo "Ingenieros de ML":

Debe tener permisos de lectura y escritura en el directorio machine_learning/ para el desarrollo y ajuste de modelos.

Permisos de ejecución en scripts/ para ejecutar scripts de entrenamiento.

Permisos de lectura y escritura en modelos/ para almacenar y modificar los modelos.

- Grupo "Ingenieros de AI":

Debe tener permisos de lectura, escritura y ejecución en machine_learning/generative_ai/ para desarrollar soluciones de IA generativa.

Permisos de lectura y escritura en visualizations/ para colaborar con la visualización de resultados de modelos generativos.

Permisos de lectura en docs/crisp_dm/ para revisar documentación de las fases del proyecto.

Tarea

Crea un script en Bash que configure los permisos indicados para cada grupo. El script debe crear los grupos si no existen y aplicar los permisos recursivamente a los directorios especificados.

El script debe crear los grupos si no existen y luego asignar los permisos especificados con chmod y chown

2.6. Compresión Automática de Archivos Antiguos

Para optimizar el almacenamiento, todos los archivos de la carpeta archivos/ que no han sido modificados en los últimos 90 días deben ser comprimidos automáticamente en un archivo .tar.gz. Crea un script para realizar esta tarea.

2.7. Resumen Automático de Archivos en un Directorio

Crea un script que genere un informe con la cantidad total de archivos, el tamaño acumulado y el número de archivos de cada tipo (por extensión) en el directorio datos/.

2.8. Búsqueda de Palabras Clave en Archivos de Log

Crea un script que busque una palabra clave dada en los archivos de log del directorio logs/ y devuelva las líneas que contienen dicha palabra. El script debe aceptar la palabra clave como un argumento.

El script debe aceptar un argumento y buscar en los archivos de log