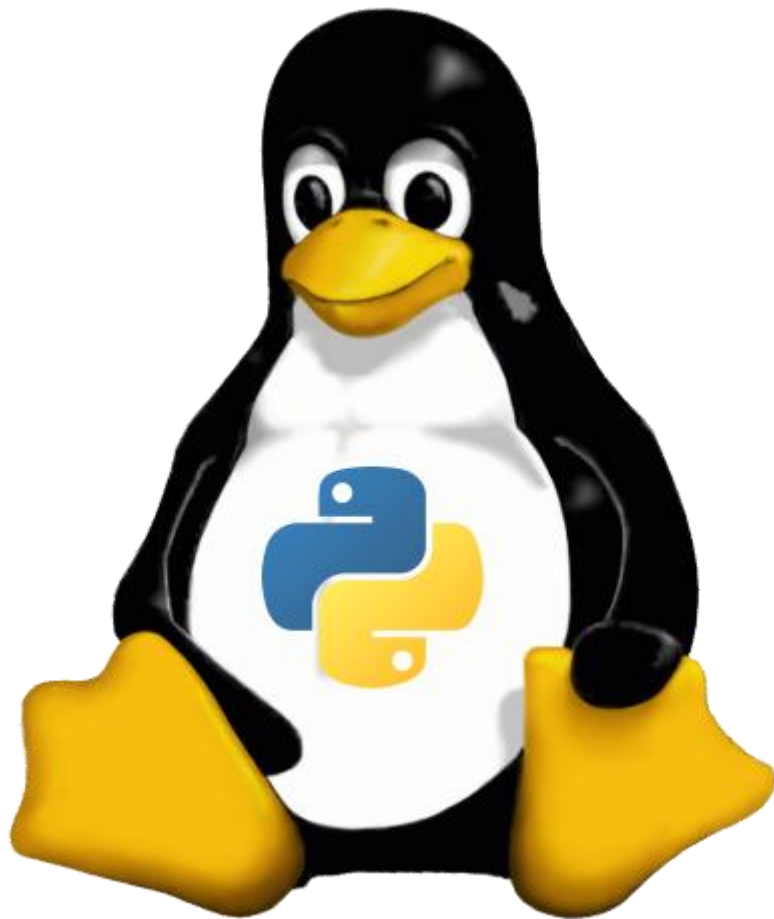


Scripts en Python para Linux



1	Nomenclatura aplicable en Python	4
2	Limpiar la pantalla del terminal	5
3	Mostrar un menú de selección de comandos	6
4	Mostrar los argumentos del script	8
4.1	Ejemplo genérico. Versión 1	8
4.2	Ejemplo genérico. Versión 2	8
5	Emplear los argumentos del script	10
5.1	Mostrar un calendario. Versión 1	10
5.2	Mostrar un calendario. Versión 2	11
6	Librerías específicas	13
6.1	Listar el directorio actual	13
6.2	Clasificar un directorio	13
7	Capturar el resultado de un comando	17
7.1	Mostrar usuario actual con whoami. Versión 1	17
7.2	Mostrar usuario actual con whoami. Versión 2	17
7.3	Crear un directorio con mkdir	18
8	Capturar el resultado de un comando	19
8.1	Comunicación con ping. Versión 1	19
8.2	Comunicación con ping. Versión 2	19
8.3	Comunicación con una subred	22
9	Capturar el resultado de un comando	24
9.1	listar directorios	24
9.2	Enumerar el contenido de la variable de entorno PATH	24
10	Librerías para tareas específicas	26
10.1	Librería datetime para fechas	26
10.2	Librería os para directorios y ficheros	27
10.3	Librería os para directorios y ficheros	27
10.4	Librería os para directorios y ficheros	28
11	Aplicación crear_directorios con refactoring	31
11.1	Versión 1	31
11.2	Versión 2	31
11.3	Versión 3	31
11.4	Versión 4	32
11.5	Versión 5	32
11.6	Versión 6	33
11.7	Versión 7	34
12	Ejercicios varios	36
12.1	Clase auxiliar de colores	36
12.2	Crear fichero en directorio	36
12.3	Ping a un único equipo	38
12.4	Ping a todos los equipos de una subred	40
12.5	Crear 20 directorios	41
12.6	Clasificar ficheros por la longitud de la extensión	42
12.7	Comprobar años bisiestos	45
12.8	Crear varios usuarios	46
12.9	Obtener usuarios reales del sistema	48
12.10	Obtener la dirección IP	49
12.11	Mostrar el calendario de un mes y año	50
12.12	Mostrar las rutas del PATH	52
12.13	Consumo de memoria por los procesos	54
12.14	Comprobar la existencia de usuarios	56
12.15	Creación de múltiples usuarios	58

13	Scripts para redes	63
13.1	Conversión entre decimal y binario	63
13.2	Traducción de la máscara entre formato abreviado y largo	64
13.2.1	Versión 1	64
13.2.2	Versión 2	66
13.3	Información de la configuración de red	68
14	IBM. Encontrar ficheros y mostrar sus permisos	71
15	IBM. Mostrar credenciales cuyo usuario o contraseña incumplen las normas	72

1 Nomenclatura aplicable en Python

3.16.4 Guidelines derived from Guido's Recommendations

Type	Public	Internal
Packages	<code>lower_with_under</code>	
Modules	<code>lower_with_under</code>	<code>_lower_with_under</code>
Classes	<code>CapWords</code>	<code>_CapWords</code>
Exceptions	<code>CapWords</code>	
Functions	<code>lower_with_under()</code>	<code>_lower_with_under()</code>
Global/Class Constants	<code>CAPS_WITH_UNDER</code>	<code>_CAPS_WITH_UNDER</code>
Global/Class Variables	<code>lower_with_under</code>	<code>_lower_with_under</code>
Instance Variables	<code>lower_with_under</code>	<code>_lower_with_under</code> (protected)
Method Names	<code>lower_with_under()</code>	<code>_lower_with_under()</code> (protected)
Function/Method Parameters	<code>lower_with_under</code>	
Local Variables	<code>lower_with_under</code>	

2 Limpiar la pantalla del terminal

El siguiente script limpia la pantalla del terminal.

La librería ***subprocess*** permite ejecutar comandos del sistema operativo. Entre los métodos disponibles nos interesa ***run***, que ejecutará el comando pasado como argumento. Los argumentos se suministran como un arreglo en el que cada elemento del comando es un elemento del arreglo.

```
#!/usr/bin/python3

import subprocess

#-----
def main() -> None:
    clear_screen()

#-----
def clear_screen() -> None:
    subprocess.run(['clear'])

#-----
if __name__ == '__main__':
    main()
```

3 Mostrar un menú de selección de comandos

El siguiente script muestra un menú en el que el usuario selecciona el comando que desea ejecutar.

Desarrolla un script denominado `seleccionar_comando_de_menu.py`. Al ejecutarlo mostrará un menú con seis opciones para ejecutar las siguientes seis operaciones: mostrar la fecha actual, mostrar quién soy, mostrar quiénes han iniciado sesión en el equipo, mostrar el calendario actual, cambiar la contraseña y salir.

```
#!/usr/bin/python3

import subprocess
import sys

def main():
    limpiar_pantalla()
    mostrar_menu()
    pedir_y_ejecutar_opciones()

def limpiar_pantalla():
    subprocess.run(['clear'])

def mostrar_menu():
    print('\n-----MENÚ-----')
    print('\t1) Mostrar fecha.')
    print('\t2) Mostrar quién soy.')
    print('\t3) Mostrar quién está conectado.')
    print('\t4) Mostrar el calendario.')
    print('\t5) Cambiar la contraseña.')
    print('\t6) Cerrar la aplicación.')

def pedir_y_ejecutar_opciones():
    while True:
        opcion: int = pedir_opcion()
        ejecutar_opcion(opcion)

def pedir_opcion() -> int:
    while True:
        respuesta = input('\nSeleccione una opción del menú: ')

        if not respuesta.isnumeric():
            print('Debe introducir un número entre 1 y 6.')
            continue
```

```

        opcion: int = int(respuesta)

        if opcion < 1 or opcion > 6:
            print('Debe introducir un número entre 1 y 6.')
            continue

        return opcion

def ejecutar_opcion(opcion: int):
    if opcion == 1:
        mostrar_fecha()
    elif opcion == 2:
        mostrar_quien_soy()
    elif opcion == 3:
        mostrar_usuarios_conectados()
    elif opcion == 4:
        mostrar_calendario()
    elif opcion == 5:
        cambiar_contraseña()
    elif opcion == 6:
        cerrar_aplicacion()

def mostrar_fecha():
    subprocess.run(['date'])

def mostrar_quien_soy():
    subprocess.run(['whoami'])

def mostrar_usuarios_conectados():
    subprocess.run(["who"])

def mostrar_calendario():
    subprocess.run(["cal"])

def cambiar_contraseña():
    subprocess.run(["passwd"])

def cerrar_aplicacion():
    sys.exit(0)

if __name__ == "__main__":
    main()

```

4 Mostrar los argumentos del script

La librería **sys** ofrece utilidades relacionadas con el sistema operativo.

La propiedad **argv** de este objeto permite conocer los elementos que permitieron la ejecución del script, es decir, lo que se escribió en el terminal para ejecutar el script. Es un arreglo cuya posición 0 corresponde al nombre del script (el fichero), y las siguientes posiciones (1, 2, ...) los argumentos suministrados al script.

El método **exit** finaliza totalmente el script actual, devolviendo al terminal un código numérico que indica si el script finalizó sin errores (código 0) o con errores (cualquier número distinto de 0 debe indicar un tipo distinto de error).

4.1 Ejemplo genérico. Versión 1

Desarrolla un script denominado `mostrar_argumentos_v1.py`. El script muestra por separado el nombre del script y los argumentos.

```
import sys

print('Nombre del script:', sys.argv[0])
print(f'Argumentos del script:', sys.argv[1:])
```

4.2 Ejemplo genérico. Versión 2

Desarrolla un script denominado `mostrar_argumentos_v2.py`. El script muestra cada argumento en una línea y numerando las líneas.

```
#!/usr/bin/python3

import sys

def main() -> None:
    show_arguments()

def show_arguments() -> None:
    script = sys.argv[0]
    arguments = sys.argv[1:]

    print(f'El script ejecutado es {script}')

    if len(arguments) == 0:
        print('El script no ha recibido argumentos.')
        sys.exit()

    print('Los argumentos del script son:')
    for i in range(0, len(arguments)):
        position = i+1
        if position == 1:
```



```
        print(f'\t[1\u1d49\u02b3 argumento] {arguments[i]}')
    elif position == 3:
        print(f'\t[3\u1d49\u02b3 argumento] {arguments[i]}')
    else:
        print(f'\t[ {position}\u1d52 argumento] {arguments[i]}')

if __name__ == '__main__':
    main()
```

5 Emplear los argumentos del script

La librería **sys** ofrece utilidades relacionadas con el sistema operativo.

La propiedad **argv** de este objeto permite conocer los elementos que permitieron la ejecución del script, es decir, lo que se escribió en el terminal para ejecutar el script. Es un arreglo cuya posición 0 corresponde al nombre del script (el fichero), y las siguientes posiciones (1, 2, ...) los argumentos suministrados al script.

5.1 Mostrar un calendario. Versión 1

Desarrolla un script llamado `mostrar_calendario_v1.py`. El script recibe dos argumentos, el mes y el año, ambos expresados como número, mostrando el calendario correspondiente a dicho mes y año.

```
#!/usr/bin/python3

import subprocess
import sys

def main():
    check_arguments()

    month = sys.argv[1]
    year = sys.argv[2]
    show_calendar(month, year)

def check_arguments():
    quantity = len(sys.argv) - 1
    if quantity != 2:
        print('ERROR: El número de argumentos no es correcto.')
        sys.exit()

    try:
        month = int(sys.argv[1])
    except:
        print('ERROR: El mes es incorrecto')
        sys.exit()

    if month < 1 or month > 12:
        print('ERROR: El mes es incorrecto')
        sys.exit()

    try:
        year = int(sys.argv[2])
    except:
        print('ERROR: El año es incorrecto')
        sys.exit()
```

```
def show_calendar(month, year):
    subprocess.run(['cal', month, year])

if __name__ == '__main__':
    main()
```

5.2 Mostrar un calendario. Versión 2

Desarrolla un script llamado `mostrar_calendario_v2.py`. El script recibe dos argumentos, el mes y el año, expresado el año como número y el mes como texto (enero, febrero, ...), mostrando el calendario correspondiente a dicho mes y año.

```
#!/usr/bin/python3

import sys
import subprocess

def main():
    if not son_argumentos_correctos():
        sys.exit()

    mes, año = obtener_argumentos()

    mostrar_calendario(mes, año)

def mostrar_calendario(mes: str, año: int) -> None:
    mes = mes.lower()

    meses = ('enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio', \
            'julio', 'agosto', 'septiembre', 'octubre', 'noviembre', 'diciembre')
    mes_en_numero = meses.index(mes) + 1

    subprocess.run(['cal', str(mes_en_numero), str(año)])

def obtener_argumentos():
    argumentos = sys.argv[1:]
    mes = argumentos[0]
    año = int(argumentos[1])

    return mes, año

def son_argumentos_correctos() -> bool:
    argumentos = sys.argv[1:]

    if len(argumentos) != 2:
        print('Error: debes poner dos argumentos que son el mes y el año.')
```

```

        return False

    mes = argumentos[0]
    año = argumentos[1]

    if not es_mes(mes):
        print('ERROR: el primer argumento debe ser el mes con nombre. ')
        return False

    if not año.isnumeric():
        print('ERROR: el segundo argumento del año debe ser un entero.')
        return False

    return True

def es_mes(mes) -> bool:
    mes = mes.lower()

    meses = ('enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio', \
            'julio', 'agosto', 'septiembre', 'octubre', 'noviembre', 'diciembre')

    if mes in meses:
        return True
    else:
        return False

    # if mes != 'enero' \
    #     and mes != 'febrero' \
    #     and mes != 'marzo' \
    #     and mes != 'abril' \
    #     and mes != 'mayo' \
    #     and mes != 'junio' \
    #     and mes != 'julio' \
    #     and mes != 'agosto' \
    #     and mes != 'septiembre' \
    #     and mes != 'octubre' \
    #     and mes != 'noviembre' \
    #     and mes != 'diciembre':
    #     return False
    # else:
    #     return True

if __name__ == "__main__":
    main()

```

6 Librerías específicas

La librería **os** incluye métodos para obtener información del sistema de ficheros. Con ella podemos ver el contenido de un directorio, consultar si un fichero es un fichero o un directorio, etc.

6.1 Listar el directorio actual

El método **listdir** devuelve una lista con los nombres de los ficheros (en Linux los directorios también son ficheros) del directorio suministrado como argumento. Si no se suministra un argumento, devuelve los ficheros del directorio actual.

Desarrolla un script denominado `listar_directorio.py` que muestre los ficheros que hay en el directorio actual. Debe emplear una librería diseñada para ello.

```
#!/usr/bin/python3

import subprocess
import os

def main() -> None:
    clear_screen()
    list_files()

def clear_screen() -> None:
    subprocess.run(['clear'])

def list_files() -> None:
    files: list[str] = os.listdir()

    print('Los elementos del directorio actual son:')

    i = 0
    for file in files:
        i += 1
        print(f'\t[{i}] {file}')

if __name__ == '__main__':
    main()
```

6.2 Clasificar un directorio

Desarrolla un script denominado `clasificar_directorio.py`. El script muestra los ficheros que hay en el directorio actual, clasificándolos en directorios y ficheros. Además, muestra la información en color.

```
#!/usr/bin/python3
```

```

import subprocess
import os
import sys

from terminal_colors import TerminalColors

# BLUE = '\033[1;34;40m'
# CYAN = '\033[1;36;40m'
# WHITE = '\033[1;37;40m'

NUMBER_OF_ARGUMENTS_WRONG = 1
ARGUMENT_IS_NOT_A_DIRECTORY = 2

def main() -> None:
    clear_screen()

    if not is_right_number_arguments():
        sys.exit(NUMBER_OF_ARGUMENTS_WRONG)

    directory = get_argument()

    if not is_a_directory(directory):
        sys.exit(ARGUMENT_IS_NOT_A_DIRECTORY)

    list_directories_and_files(directory)

def clear_screen() -> None:
    subprocess.run(['clear'])

def is_right_number_arguments() -> bool:
    arguments = sys.argv[1:]
    if len(arguments) > 1:
        show_error(
            NUMBER_OF_ARGUMENTS_WRONG,
            'El número de argumentos es erróneo.')
        return False

    return True

def get_argument():
    arguments = sys.argv[1:]
    if len(arguments) == 1:
        return arguments[0]
    else:
        return os.getcwd()

def is_a_directory(file, path = None):
    if path:
        if os.path.isdir(os.path.join(path, file)):
            return True
        else:
            return False

```

```

    else:
        if os.path.isdir(file):
            return True
        else:
            return False

def is_a_file(file, path = None):
    if path:
        if os.path.isfile(os.path.join(path, file)):
            return True
        else:
            return False
    else:
        if os.path.isfile(file):
            return True
        else:
            return False

def show_error(type, message):
    print(f'ERROR ({type}): {message}')
    print('USO: comando [directorio]')

def list_directories_and_files(directory) -> None:
    # print(BLUE)
    print(TerminalColors.WHITE_BLUE)
    list_directories(directory)
    # print(CYAN)
    print(TerminalColors.WHITE_CYAN)
    list_files(directory)

def list_directories(directory) -> None:
    files: list[str] = os.listdir(directory)

    print('Los directorios son:')

    for file in files:
        if is_a_directory(file, directory):
            print(f'\t{file}')

def list_files(directory) -> None:
    files: list[str] = os.listdir(directory)

    print('Los ficheros son:')

    for file in files:
        if is_a_file(file, directory):
            print(f'\t{file}')

#-----
if __name__ == '__main__':
    main()

```

El fichero importado con los códigos de los colores es:

```
# terminal_colors.py
class TerminalColors:
    BLACK_GRAY    = '\033[1;30;40m'
    BLACK_RED     = '\033[1;31;40m'
    BLACK_GREEN   = '\033[1;32;40m'
    BLACK_YELLOW  = '\033[1;33;40m'
    BLACK_BLUE    = '\033[1;34;40m'
    BLACK_MAGENTA = '\033[1;35;40m'
    BLACK_CYAN    = '\033[1;36;40m'
    BLACK_WHITE   = '\033[1;37;40m'

    WHITE_GRAY    = '\033[0;30;47m'
    WHITE_RED     = '\033[0;31;47m'
    WHITE_GREEN   = '\033[0;32;47m'
    WHITE_YELLOW  = '\033[0;33;47m'
    WHITE_BLUE    = '\033[0;34;47m'
    WHITE_MAGENTA = '\033[0;35;47m'
    WHITE_CYAN    = '\033[0;36;47m'
    WHITE_WHITE   = '\033[0;37;47m'
```


7 Capturar el resultado de un comando

Dentro de un script es posible capturar el resultado de la ejecución de un comando.

Un comando normalmente muestra por su salida estándar (la pantalla del terminal) los mensajes generados durante su ejecución.

El método **run** permite modificar este comportamiento con los parámetros **capture_output** y **text**. Cuando ambos son True significa que la salida es en formato texto y no queremos que aparezca por pantalla.

El método **run** siempre devuelve un objeto que permite gestionar adecuadamente su salida. Las propiedades más importantes de dicho objeto son:

- **returncode**: almacena el código de error devuelto por la ejecución directa del comando. Está estandarizado que el 0 indique que el comando se ejecuto con éxito y cualquier valor distinto de 0 que hubo un problema.
- **stdout**: aquí se encuentra almacenada la salida de ejecución correcta como una cadena de texto.
- **stderr**: aquí se encuentra almacenada la salida de error como una cadena de texto.

7.1 Mostrar usuario actual con whoami. Versión 1

Es siguiente script se denomina `mostrar_quien_soy_v1.py`.

El script muestra el usuario actual sin realizar ninguna captura del resultado.

```
#!/usr/bin/python3
import subprocess

def main() -> None:
    subprocess.run(['whoami'])

if __name__ == '__main__':
    main()
```

7.2 Mostrar usuario actual con whoami. Versión 2

Es siguiente script se denomina `mostrar_quien_soy_v2.py`.

El script primero no muestra los resultados del comando en la pantalla, los captura en la variable `resultado` que almacena un objeto. Con `returncode` se comprueba que todo ha ido bien y con `stdout` se ve la gestiona la información que mostraría la salida correcta y que ha sido capturada.

```
#!/usr/bin/python3

import subprocess

def main() -> None:
    resultado = subprocess.run(
        ['whoami'],
        text = True,
        capture_output = True
    )

    if resultado.returncode == 0:
        print(f'El comando "{resultado.args[0]}" ha sido ejecutado con éxito')
        print(f'El usuario actual es "{resultado.stdout.strip()}"')

if __name__ == '__main__':
    main()
```

7.3 Crear un directorio con mkdir

Desarrolla un script llamado crear_varios_directorios.py. El script no recibirá argumentos. En su interior habrá una lista de nombres de usuario y para cada uno de ellos se creará un directorio en /home.

```
#!/usr/bin/python3
import subprocess

def main():
    directorios = ('juan', 'cristina', 'maria', 'nicolas', 'fernando')
    try:
        crear_directorios(directorios)
    except Exception as error:
        print(error)

def crear_directorios(directorios):
    for directorio in directorios:
        resultado = subprocess.run(
            ['mkdir', f'./pruebas/{directorio}'],
            text=True,
            capture_output=True
        )

        if resultado.returncode != 0:
            raise Exception(f'\tNo se ha podido crear el directorio {directorio}.\n\tEl motivo es "{resultado.stderr}")

if __name__ == '__main__':
    main()
```

8 Capturar el resultado de un comando

8.1 Comunicación con ping. Versión 1

Dentro de un script es posible capturar el resultado de la ejecución de un comando.

En redes el término host significa equipo de comunicaciones, es decir, cualquier elemento con capacidad de comunicarse.

Desarrolla un script denominado `hacer_ping_a_un_host.py`. Este script recibe como único argumento la dirección IP de un equipo (por ejemplo 192.168.1.2) e informa si existe comunicación con ese equipo.

```
#!/usr/bin/python3

import sys
import subprocess

def main() -> None:
    ip = get_ip()
    test_comunication(ip)

def get_ip() -> str:
    if len(sys.argv[1:]) != 1:
        print('ERROR: Hay que suministrar la dirección IP')
        sys.exit(1)

    ip = sys.argv[1]
    return ip

def test_comunication(ip: str) -> None:
    result = subprocess.run(
        ['ping', '-c', '1', ip],
        text = True,
        capture_output = True
    )

    if result.returncode == 0:
        print('Hay comunicación con el destino')
    else:
        print('No hay comunicación con el destino')

if __name__ == '__main__':
    main()
```

8.2 Comunicación con ping. Versión 2

Otra versión más elaborada del mismo programa.

Dentro de un script es posible capturar el resultado de la ejecución de un comando.

En redes el término host significa equipo de comunicaciones, es decir, cualquier elemento con capacidad de comunicarse.

Desarrolla un script denominado `hacer_ping_a_un_host.py`. Este script recibe como único argumento la dirección IP de un equipo (por ejemplo 192.168.1.2) e informa si existe comunicación con ese equipo.

```
#!/usr/bin/python3

import subprocess
import sys

ERROR_NUMERO_ARGUMENTOS_ERRONEO = 1
ERROR_DIRECCION_NO_VALIDA = 2

def main():
    limpiar_pantalla()
    direccion = obtener_direccion()
    testear_conexion(direccion)

def testear_conexion(direccion: str) -> None:
    resultado = subprocess.run(
        ['ping', '-c', '1', direccion],
        text=True,
        capture_output=True
    )

    if resultado.returncode != 0:
        print(f'No se ha podido establecer comunicación con {direccion}')
    else:
        print(f'Hay comunicación con la dirección {direccion}')

def obtener_direccion() -> str:
    argumentos = sys.argv[1:]
    if len(argumentos) > 1:
        print('ERROR: Solo puedes poner cero o un argumentos.')
        sys.exit(ERROR_NUMERO_ARGUMENTOS_ERRONEO)

    if len(argumentos) == 0:
        direccion = pedir_direccion_por_teclado()
    else:
        direccion = extraer_direccion(argumentos)

    if not es_direccion_correcta(direccion):
        print('ERROR: El elemento introducido no es una dirección.')
```

```

        sys.exit(ERROR_DIRECCION_NO_VALIDA)

    return direccion

def es_direccion_correcta(direccion: str) -> bool:
    numeros = direccion.split('.')

    if len(numeros) != 4:
        return False

    for numero in numeros:
        if not numero.isnumeric():
            return False

        numero = int(numero)
        if numero < 0 or numero > 255:
            return False

    return True

def pedir_direccion_por_teclado() -> str:
    direccion = input('Dime la dirección con la que quieres comunicarte: ')
    return direccion

def extraer_direccion(argumentos: list[str]) -> str:
    return argumentos[0]

def limpiar_pantalla():
    subprocess.run(['clear'])

if __name__ == "__main__":
    main()

```

8.3 Comunicación con una subred

Dentro de un script es posible capturar el resultado de la ejecución de un comando.

En redes el término subred significa varios equipos capaces de comunicarse directamente, es decir, sin emplear intermediarios como una puerta de enlace.

Desarrolla un script denominado `hacer_ping_a_una_subred.py`. Este script recibe como argumento la dirección de una subnet (por ejemplo 192.168.1) e informa de con que equipos de la subred existe comunicación.

```
#!/usr/bin/python3

import sys
import subprocess

def main() -> None:
    try:
        subnet: str = get_subnet()
    except Exception as error:
        print(f'ERROR: {error}')
        sys.exit(1)

    test_comunication_with_subnet(subnet)

def get_subnet() -> str:
    if len(sys.argv[1:]) != 1:
        raise Exception('Problema con el número de argumentos')

    subnet = sys.argv[1]

    numbers = subnet.split('.')
    if len(numbers) != 3:
        raise Exception('El argumento no es una subnet')

    try:
        first_number = int(numbers[0])
        second_number = int(numbers[1])
        third_number = int(numbers[2])
    except:
        raise Exception('El argumento no es una subnet')

    if first_number not in [10, 172, 192]:
        raise Exception('El argumento no es una subnet')

    return subnet

def test_comunication_with_subnet(subnet: str) -> None:
    for i in range(1, 255):
        ip = f'{subnet}.{i}'

        if test_comunication_with_host(ip):
```

```
        print(f'Hay comunicación con el destino {ip}')
    else:
        print(f'No hay comunicación con el destino {ip}')

def test_comunication_with_host(ip: str) -> bool:
    result = subprocess.run(
        ['ping', '-c', '1', ip],
        text=True,
        capture_output=True
    )

    if result.returncode == 0:
        return True
    else:
        return False

if __name__ == '__main__':
    main()
```

9 Capturar el resultado de un comando

9.1 listar directorios

Desarrolla un script denominado `listar_directorios.py`. El script listará el contenido del directorio raíz del sistema de ficheros.

```
#!/usr/bin/python3

from typing import List
import subprocess

def main():
    try:
        directorio = '/'
        contenido = listar_directorio(directorio)
        print(f'El contenido del directorio "{directorio}" es:')
        for elemento in contenido:
            print(f'\t{elemento}')
    except Exception as error:
        print(error)

def listar_directorio(directorio: str) -> List[str]:
    resultado_ls = subprocess.run(
        ['ls', directorio],
        text=True,
        capture_output=True,
    )

    if resultado_ls.returncode != 0:
        raise Exception(f'ERROR al listar el directorio {directorio}')

    return resultado_ls.stdout.split('\n')

if __name__ == '__main__':
    main()
```

9.2 Enumerar el contenido de la variable de entorno PATH

Crea un script llamado `mostrar_path.py` que enumere los directorios de la variable de entorno PATH.

Versión 1

```
#!/usr/bin/python3

import subprocess

def main():
    limpiar_pantalla()
```



```

valor_path = extraer_valor_linea_path()
enumerar_directorios_path(valor_path)

def enumerar_directorios_path(path):
    directorios = path.split(':')

    print('Los directorios incluidos en el PATH son:')
    for i, directorio in enumerate(directorios):
        print(f'\t{i+1}) {directorio}')

def extraer_valor_linea_path():
    resultado = subprocess.run(
        ['env'],
        capture_output=True,
        text=True
    )

    if resultado.returncode != 0:
        print('No se han podido recuperar las variables de entorno')
        print(resultado.stderr)
    else:
        lineas = resultado.stdout.split('\n')

        for linea in lineas:
            linea_como_lista = linea.split('=')
            if linea_como_lista[0] == "PATH":
                return linea_como_lista[1]

def limpiar_pantalla():
    subprocess.run(['clear'])

if __name__ == "__main__":
    main()

```

10 Librerías para tareas específicas

10.1 Librería datetime para fechas

El siguiente script importa una librería nativa de Python que permite trabajar con fechas y no tener que recurrir al comando propio del sistema operativo.

```
from datetime import date

date_today = date.today()
print('La fecha de hoy sin procesar es', date_today)
print("La fecha de hoy procesada es: %d/%d/%d" % (date_today.day,
date_today.month, date_today.year))

custom_date = date(2021, 4, 5)
print("La fecha particular es:", custom_date)
print("La fecha particular procesada es: %d/%d/%d" % (custom_date.day,
custom_date.month, custom_date.year))
```

Este script funciona bien pero el estilo de programación es mejorable. El siguiente script organiza mejor el código y añade alguna funcionalidad.

```
#!/usr/bin/python3

from datetime import date

def main():
    date_today = date.today()
    print('La fecha de hoy sin procesar es', date_today)
    print('La fecha de hoy procesada es:', get_date_as_numbers(date_today))
    print('La fecha de hoy procesada es:', get_date_as_text(date_today))

    custom_date = date(2021, 11, 5)
    print("La fecha particular sin procesar es:", custom_date)
    print('La fecha particular procesada es:',
        get_date_as_numbers(custom_date))
    print('La fecha particular procesada es:', get_date_as_text(custom_date))

def get_date_as_numbers(date):
    # if date.month < 10:
    #     month = f'0{date.month}'
    # else:
    #     month = date.month
    month = f'0{date.month}' if date.month < 10 else date.month
    return f'{date.day}/{month}/{date.year}'

def get_date_as_text(date):
    months = ['enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio',
        'julio', 'agosto', 'septiembre', 'octubre', 'noviembre', 'diciembre']
    month = months[date.month - 1]
    return f'{date.day} de {month} de {date.year}'

if __name__ == '__main__':
    main()
```

10.2 Librería os para directorios y ficheros

La librería os ofrece herramientas para acceder al sistema de ficheros. Entre las operaciones que podemos realizar destacamos:

- Recuperar el contenido de un directorio, empleando el método listdir.
- Averiguar si un fichero es un fichero ordinario, empleando el método isfile.
- Averiguar si un fichero es un directorio, empleando el método isdir.

```
import os

print(os.listdir('/'))
print(os.listdir('/lib'))
print(os.path.isdir('/dev'))
print(os.path.isfile('/dev'))
print(os.path.isdir('/etc/passwd'))
print(os.path.isfile('/etc/passwd'))
```

10.3 Librería os para directorios y ficheros

El siguiente script clasifica los ficheros que hay situados dentro de un directorio que hay que pasar como argumento en la línea de comandos en ficheros ordinarios, directorios y ficheros especiales.

```
#!/usr/bin/python3

import os
import subprocess
import sys

# Limpiamos el terminal
subprocess.run(['clear'])

# Comprobamos que el script recibe un argumento
if len(sys.argv[1:]) != 1:
    print(f'ERROR: Número de argumentos erróneo')
    print('USO: python <script> <directorio>')
    sys.exit(1)

# Comprobamos que el argumento es un directorio
directorio = sys.argv[1]
if not os.path.isdir(directorio):
    print(f'ERROR: El termino "{directorio}" no es un directorio')
    print('USO: python <script> <directorio>')
    sys.exit(1)

# Clasificamos los ficheros en las categorías:
# 1) Ficheros ordinarios
# 2) Directorios
# 3) Ficheros especiales
ficheros = os.listdir(directorio)
```

```

print(f'Los directorios situados en {directorio} son:')
for fichero in ficheros:
    if os.path.isdir(f'{directorio}/{fichero}'):
        print(fichero, end=', ')

print(f'\nLos ficheros ordinarios situados en {directorio} son:')
for fichero in ficheros:
    if os.path.isfile(f'{directorio}/{fichero}'):
        print(fichero, end=', ')

print(f'\nLos ficheros especiales situados en {directorio} son:')
for fichero in ficheros:
    if not os.path.isdir(f'{directorio}/{fichero}') \
        and not os.path.isfile(f'{directorio}/{fichero}'):
        print(fichero, end=', ')

```

Se ejecuta el script desde el terminal.

```
$ python clasificar_ficheros.py /dev
```

10.4 Librería os para directorios y ficheros

El siguiente script clasifica los ficheros que hay situados dentro de un directorio que hay que pasar como argumento en la línea de comandos en ficheros ordinarios, directorios y ficheros especiales.

```

#!/usr/bin/python3

import os
import subprocess
import sys

EXITO = 0
ERROR_NUMERO_ARGUMENTOS_ERRONEO = 1
ERROR_ARGUMENTO_NO_ES_DIRECTORIO = 2

def limpiar_pantalla():
    subprocess.run(['clear'])

def mostrar_error(mensaje):
    print(f'ERROR: {mensaje}')
    print('USO: comando <directorio>')

def tiene_un_argumento_el_script(argumentos):
    if len(argumentos) != 1:
        mostrar_error('Número de argumentos erróneo')
        return False

    return True

```

```

def es_un_fichero(argumento):
    if not os.path.isfile(argumento):
        return False

    return True

def es_un_directorio(argumento):
    if not os.path.isdir(argumento):
        return False

    return True

def es_argumento_del_script_un_directorio(argumento):
    if not es_un_directorio(argumento):
        mostrar_error(f'El término "{argumento}" no es un directorio')
        return False

    return True

def listar_ficheros_clasificados(directorio):
    ficheros = os.listdir(directorio)

    print(f'Los directorios situados en {directorio} son:')
    for fichero in ficheros:
        if es_un_directorio(f'{directorio}/{fichero}'):
            print(fichero, end=', ')

    print(f'\nLos ficheros ordinarios en {directorio} son:')
    for fichero in ficheros:
        if es_un_fichero(f'{directorio}/{fichero}'):
            print(fichero, end=', ')

    print(f'\nLos ficheros especiales en {directorio} son:')
    for fichero in ficheros:
        if not es_un_directorio(f'{directorio}/{fichero}')
           and not es_un_fichero(f'{directorio}/{fichero}'):
            print(fichero, end=', ')

def principal():
    limpiar_pantalla()

    argumentos = sys.argv[1:]

    if not tiene_un_argumento_el_script(argumentos):
        sys.exit(ERROR_NUMERO_ARGUMENTOS_ERRONEO)

    directorio = argumentos[0]
    if not es_argumento_del_script_un_directorio(directorio):
        sys.exit(ERROR_ARGUMENTO_NO_ES_DIRECTORIO)

    listar_ficheros_clasificados(directorio)

```

```
sys.exit(EXITO)

if __name__ == '__main__':
    principal()
```

Se ejecuta el script desde el terminal.

```
$ python clasificar_ficheros.py /dev
```

11 Aplicación crear_directorios con refactoring

Los siguientes scripts realizan todos la misma operación: reciben por medio de la línea de comandos una lista de nombres y se crear un directorio por cada uno de los nombres.

Por ejemplo,

```
$ python3 crear_directorios.py pera fresa manzana sandia naranja
```

crearía en el directorio actual cinco directorios con los nombres indicados.

11.1 Versión 1

La primera versión hace simplemente lo que se pide: crear los directorios.

```
import subprocess, sys

for i in range(1, len(sys.argv)):
    subprocess.run(['mkdir', sys.argv[i]])
```

Podemos observar que no hay ningún control de errores, ni se aplican ninguna de las normas recomendadas en Python.

11.2 Versión 2

Esta versión añade el código recomendado para cualquier script Python.

```
#!/usr/bin/python3

import subprocess, sys

#-----
def main():
    for i in range(1, len(sys.argv)):
        subprocess.run(['mkdir', sys.argv[i]])

#-----
if __name__ == '__main__':
    main()
```

En la primera línea indicamos el intérprete Python para el que hemos desarrollado el script. Al final indicamos la función que queremos que sea el inicio del programa.

11.3 Versión 3

En esta versión añadimos algo de tratamiento de errores.

```
#!/usr/bin/python3

import subprocess, sys
```

```
#-----
def main():
    for i in range(1, len(sys.argv)):
        resultado = subprocess.run(['mkdir', sys.argv[i]])
        if resultado.returncode != 0:
            print('ERROR: No se ha podido crear un directorio')
            sys.exit(1)

#-----
if __name__ == '__main__':
    main()
```

El método run devuelve un objeto con información sobre la ejecución del comando. Empleamos la propiedad returncode para saber si hubo problemas con la ejecución del comando de Linux, y si es así informamos adecuadamente.

11.4 Versión 4

La siguiente mejora informa también de cuando no ha habido problemas.

```
#!/usr/bin/python3

import subprocess, sys

#-----
def main():
    for i in range(1, len(sys.argv)):
        resultado = subprocess.run(['mkdir', sys.argv[i]])
        if resultado.returncode != 0:
            print(f'ERROR: No se ha podido crear directorio {sys.argv[i]}')
            continue

        print(f'El directorio {sys.argv[i]} ha sido creado')

#-----
if __name__ == '__main__':
    main()
```

11.5 Versión 5

Una aplicación grande no puede estar compuesta de una única función. Hay que organizar el código en funciones que hagan una única tarea.

```
#!/usr/bin/python3

import subprocess, sys

#-----
def main():
    for i in range(1, len(sys.argv)):
        try:
            crear_directorio(sys.argv[i])
        except Exception as error:
            print(f'ERROR: {error}')

#-----
def crear_directorio(directorio):
```



```

    resultado = subprocess.run(
        ['mkdir', directorio],
        text=True,
        capture_output=True
    )

    if resultado.returncode != 0:
        raise Exception(f'Problema en la creación del directorio
{directorio}')

    print(f'El directorio {directorio} ha sido creado')

#-----
if __name__ == '__main__':
    main()

```

La función `crear_directorio` tiene como tarea lo expresado en su nombre y esa es su única responsabilidad. Ahora el código es más sencillo de entender.

La función gestiona los errores lanzando una excepción cuando se produce un error en el comando `mkdir`. Esta excepción debe tratarse por las funciones que empleen esta función, es decir, deben capturarla y actuar de acuerdo a sus necesidades.

Además se ha eliminado un mensaje de error emitido por el comando `mkdir`, es decir, los únicos mensajes que deben aparecer son los mostrados por nuestro código, no por otros elementos. Los dos parámetros añadidos al método `run` tienen la función de no mostrar los mensajes generados por `mkdir` y almacenarlos en el objeto `resultado`.

11.6 Versión 6

Todo script debe asegurarse de que los argumentos recibidos son correctos. Añadimos dicha comprobación.

```

#!/usr/bin/python3

import subprocess, sys

#-----
def main():
    if not son_argumentos_correctos():
        print('ERROR: No has suministrado la lista de directorios')
        sys.exit(1)

    crear_directorios()

#-----
def son_argumentos_correctos() -> bool:
    if len(sys.argv) == 1:
        return False
    else:
        return True

#-----
def crear_directorios() -> None:
    for i in range(1, len(sys.argv)):
        try:

```

```

        crear_directorio(sys.argv[i])
    except Exception as error:
        print(f'ERROR: {error}')

#-----
def crear_directorio(directorio: str) -> None:
    resultado = subprocess.run(
        ['mkdir', directorio],
        text=True,
        capture_output=True
    )

    if resultado.returncode != 0:
        raise Exception(f'Problema en la creación del directorio
{directorio}')

    print(f'El directorio {directorio} ha sido creado')

#-----
if __name__ == '__main__':
    main()

```

La función `son_argumentos_correctos` tiene como única responsabilidad esa tarea, revisar que recibimos la lista de nombres para los directorios.

Además se ha añadido una función llamada `crear_directorios` para asumir dicha responsabilidad, y que no recaiga en la función `main`. Así el código es más entendible.

Por último hemos añadido los tipos de datos, algo que no es obligatorio en Python pero sí deseable, casi imprescindible en programas grandes.

11.7 Versión 7

Últimos ajustes.

```

#!/usr/bin/python3

import subprocess, sys
from typing import List

#-----
def main():
    directorios = sys.argv[1:]

    if not son_argumentos_correctos(directorios):
        print('ERROR: No has suministrado la lista de directorios')
        sys.exit(1)

    crear_directorios(directorios)

#-----
def son_argumentos_correctos(argumentos: List[str]) -> bool:
    if len(argumentos) == 0:
        return False
    else:
        return True

```

```

#-----
def crear_directorios(directorios: List[str]) -> None:
    for i in range(0, len(directorios)):
        try:
            crear_directorio(directorios[i])
        except Exception as error:
            print(f'ERROR: {error}')

#-----
def crear_directorio(directorio: str) -> None:
    resultado = subprocess.run(
        ['mkdir', directorio],
        text=True,
        capture_output=True
    )

    if resultado.returncode != 0:
        raise Exception(f'Problema en la creación del directorio
{directorio}')

    print(f'El directorio {directorio} ha sido creado')

#-----
if __name__ == '__main__':
    main()

```

Terminamos afinando el código. Hasta ahora en los argumentos teníamos incluido el nombre del script, y en esta versión quitamos ese elemento ya que no es un argumento que tenga sentido para este script realmente.

Además hemos añadido el tipo de datos List que debe ser importado, y detallamos que nuestras listas son de cadenas de texto.

12 Ejercicios varios

12.1 Clase auxiliar de colores

En un fichero llamado `terminal_colors.py` crea una clase auxiliar para poder imprimir mensajes de colores. La clase se llamará `TerminalColors` y su contenido serán propiedades con los códigos que establecen el color con el que se escribirá en el terminal.

```
class TerminalColors:
    BLACK_GRAY      = '\033[1;30;40m'
    BLACK_RED       = '\033[1;31;40m'
    BLACK_GREEN     = '\033[1;32;40m'
    BLACK_YELLOW    = '\033[1;33;40m'
    BLACK_BLUE      = '\033[1;34;40m'
    BLACK_MAGENTA   = '\033[1;35;40m'
    BLACK_CYAN      = '\033[1;36;40m'
    BLACK_WHITE     = '\033[1;37;40m'
    WHITE_GRAY      = '\033[0;30;47m'
    WHITE_RED       = '\033[0;31;47m'
    WHITE_GREEN     = '\033[0;32;47m'
    WHITE_YELLOW    = '\033[0;33;47m'
    WHITE_BLUE      = '\033[0;34;47m'
    WHITE_MAGENTA   = '\033[0;35;47m'
    WHITE_CYAN      = '\033[0;36;47m'
    WHITE_WHITE     = '\033[0;37;47m'
```

12.2 Crear fichero en directorio

Desarrolla un script llamado `crear_fichero_en_directorio.py`.

Todas las operaciones deben hacerse con comandos de Linux (Unix, MacOS, Powershell) y `subprocess.run`, no se permite emplear otras librerías.

El script recibirá argumentos por la línea de comandos:

- Si recibe un argumento y es `-h` debe mostrar la ayuda del comando. Cuidado, para mostrar esta ayuda no se emplea el comando `man`, simplemente se imprime desde dentro del script.
- Si recibe dos argumentos, el primero es el nombre del fichero y el segundo es el nombre del directorio.

Para crear el directorio emplea el comando **`mkdir`**. Por ejemplo, `mkdir directorio1`

Para crear el fichero emplea el comando **`touch`**. Como el fichero ha de crearse dentro del directorio emplea una ruta relativa. Por ejemplo, `touch directorio1/fichero1`

El mensaje de ayuda es:

NOMBRE

`crear_fichero_en_directorio.py` Crea un directorio y dentro crea un fichero

SINTAXIS

crear_fichero_en_directorio.py -h

crear_fichero_en_directorio.py FICHERO DIRECTORIO

OPCIONES

-h Muestra la ayuda

FICHERO Nombre del fichero a crear

DIRECTORIO Nombre del directorio a crear

```
#!/usr/bin/python3

import subprocess
import sys
from typing import List, Tuple

EXITO = 0
PROBLEMA_CON_LOS_ARGUMENTOS = 1
PROBLEMA_CON_LA_CREACION = 2

def main():
    argumentos = sys.argv[1:]

    if es_mostrar_ayuda(argumentos):
        mostrar_ayuda()
        sys.exit(EXITO)

    try:
        fichero, directorio = obtener_argumentos(argumentos)
    except Exception as error:
        print(f'ERROR recuperando los argumentos:\n{error}')
        sys.exit(PROBLEMA_CON_LOS_ARGUMENTOS)

    try:
        crear_fichero_directorio(fichero, directorio)
    except Exception as error:
        print(f'ERROR creando los elementos:\n{error}')
        sys.exit(PROBLEMA_CON_LA_CREACION)

    sys.exit(EXITO)

def es_mostrar_ayuda(argumentos: List[str]) -> bool:
    if len(argumentos) == 1 and argumentos[0] == '-h':
        return True
    else:
        return False

def mostrar_ayuda():
    mensaje = '''
    NOMBRE
        crear_fichero_en_directorio.py Crea un directorio y dentro crea un fichero
    SINTAXIS
        crear_fichero_en_directorio.py -h
        crear_fichero_en_directorio.py FICHERO DIRECTORIO
    OPCIONES
        -h Muestra la ayuda
        FICHERO Nombre del fichero a crear
        DIRECTORIO Nombre del directorio a crear
    ...
    '''
    print(mensaje)
```

```

def obtener_argumentos(argumentos: List[str]) -> Tuple[str]:
    if len(argumentos) != 2:
        raise Exception('Número de argumentos erróneo')

    fichero = argumentos[0]
    directorio = argumentos[1]
    return fichero, directorio

def crear_fichero_directorio(fichero: str, directorio: str):
    crear_directorio(directorio)
    crear_fichero(fichero, directorio)

def crear_directorio(directorio):
    resultado = subprocess.run(
        ['mkdir', directorio],
        text=True,
        capture_output=True
    )
    if resultado.returncode != 0:
        raise Exception(
            f'''
            No se ha podido crear el directorio.
            Información técnica: {resultado.stderr}
            '''
        )

def crear_fichero(fichero, directorio):
    resultado = subprocess.run(
        ['touch', f'{directorio}/{fichero}'],
        text=True,
        capture_output=True
    )
    if resultado.returncode != 0:
        raise Exception(
            f'''
            No se ha podido crear el fichero.
            Información técnica: {resultado.stderr}
            '''
        )

if __name__ == '__main__':
    main()

```

12.3 Ping a un único equipo

En un fichero llamado `test_comunicacion_equipo.py` desarrolla un script que reciba como argumento una dirección IP y compruebe si hay o no hay comunicación con dicho equipo.

`python3 test_comunicacion_equipo.py 8.8.8.8`

Emplea el comando “ping” con la opción “-c 1” que envía un único mensaje al destinatario. Por ejemplo, “ping -c 1 8.8.8.8”.

```
#!/usr/bin/python3
```

```

import sys
import subprocess
from typing import List
from terminal_colors import TerminalColors

# -----
def main():
    argumentos = sys.argv[1:]
    if not es_argumento_correcto(argumentos):
        print('El argumento recibido no es correcto. Debe poner una
dirección IP')
        sys.exit(1)

    ip = argumentos[0]
    try:
        enviar_paquete_a_equipo(ip)
        print(TerminalColors.BLACK_GREEN)
        print(f'Comunicación con éxito con el host {ip}')
        print(TerminalColors.BLACK_WHITE)
    except Exception as error:
        print(TerminalColors.BLACK_RED)
        print(f'ERROR: {error}')
        print(TerminalColors.BLACK_WHITE)

# -----
def enviar_paquete_a_equipo(ip: str):
    resultado_ping = subprocess.run(
        ['ping', '-c', '1', ip],
        text=True,
        capture_output=True
    )

    if resultado_ping.returncode != 0:
        raise Exception(f'No he podido hacer el ping al host {ip}')

# -----
def es_argumento_correcto(argumentos: List[str]) -> bool:
    if len(argumentos) != 1:
        return False

    ip = argumentos[0]

    ip_troceada = ip.split('.')
    if len(ip_troceada) != 4:
        return False

    for cadena in ip_troceada:
        try:
            numero = int(cadena)
            if numero < 0 or numero > 255:
                return False
        except:
            return False

    return True

# -----
if __name__ == '__main__':
    main()

```

12.4 Ping a todos los equipos de una subred

En un fichero llamado `test_comunicacion_subred.py` desarrolla un script que reciba como argumento una dirección de subred y compruebe con qué equipos de la subred hay o no hay comunicación. La llamada será del tipo:

```
python3 test_comunicacion_subred.py 192.168.1
```

Emplea el comando “ping” con la opción “-c 1” que envía un único mensaje al destinatario. Por ejemplo, “ping -c 1 8.8.8.8”.

```
#!/usr/bin/python3

import sys
import subprocess
from typing import List
from terminal_colors import TerminalColors

# -----
def main():
    argumentos = sys.argv[1:]
    if not es_argumento_correcto(argumentos):
        print('El argumento recibido no es correcto. Debe poner una
dirección de subred')
        sys.exit(1)

    subred = argumentos[0]
    enviar_paquetes_a_equipos(subred)

# -----
def enviar_paquetes_a_equipos(subred: str):
    for i in range(100, 255):
        try:
            ip = f'{subred}.{i}'
            enviar_paquete_a_equipo(ip)
            print(TerminalColors.BLACK_GREEN, f'Comunicación con éxito con
el host {ip}', TerminalColors.BLACK_WHITE)
        except Exception as error:
            print(TerminalColors.BLACK_RED, f'ERROR: {error}',
TerminalColors.BLACK_WHITE)

# -----
def enviar_paquete_a_equipo(ip: str):
    resultado_ping = subprocess.run(
        ['ping', '-c', '1', '-W', '1', ip],
        text=True,
        capture_output=True
    )

    if resultado_ping.returncode != 0:
        raise Exception(f'No he podido hacer el ping al host {ip}')

# -----
def es_argumento_correcto(argumentos: List[str]) -> bool:
    if len(argumentos) != 1:
        return False

    ip = argumentos[0]
```



```

ip_troceada = ip.split('.')
if len(ip_troceada) != 3:
    return False

for cadena in ip_troceada:
    try:
        numero = int(cadena)
        if numero < 0 or numero > 255:
            return False
    except:
        return False

return True

# -----
if __name__ == '__main__':
    main()

```

12.5 Crear 20 directorios

Crea un fichero llamado crear_20_directorios.py que reciba como argumento una palabra, y empleando un bucle cree los 20 directorios empleando la palabra como base y a continuación la numeración 01, 02, ..., 20.

El comando de Linux mkdir permite crear un directorio. Al ejecutarlo devuelve los siguientes valores:

- 0 si no hay problemas.
- 1 si no se puede crear el directorio porque ya existe.
- 2 si el usuario no puede crear el directorio porque no tiene permiso para crearlo.

Por ejemplo, "python3 crear_20_directorios.py daw" tras ejecutarse crearía los directorios daw01, daw02, daw03, ..., daw19, daw20.

```

#!/usr/bin/python3

import subprocess
import sys
import re

COLOR_ROJO = '\033[91m'
COLOR_BLANCO = '\033[0m'
COLOR_VERDE = '\033[92m'
ARGUMENTOS_NO_CORRECTOS = 1

def main():
    if not es_argumento_correcto():
        print(COLOR_ROJO, '\tERROR: Los argumentos no son correctos.',
COLOR_BLANCO)
        sys.exit(ARGUMENTOS_NO_CORRECTOS)

```

```

nombre_base = sys.argv[1]
crear_20_directorios(nombre_base)

def es_argumento_correcto() -> bool:
    argumentos = sys.argv[1:]

    if len(argumentos) != 1:
        return False

    argumento = argumentos[0]
    if not hay_solo_letras(argumento):
        return False

    return True

def hay_solo_letras(palabra):
    patron = re.compile(r'[A-Za-z]+')
    resultado = patron.fullmatch(palabra)
    if resultado == None:
        return False

    return True

def crear_20_directorios(nombre_base: str) -> None:
    for i in range(1, 21):
        if i < 10:
            nombre_directorio = f'{nombre_base}0{i}'
        else:
            nombre_directorio = f'{nombre_base}{i}'

        resultado = subprocess.run(
            ['mkdir', nombre_directorio],
            text=True,
            capture_output=True
        )

        if resultado.returncode != 0:
            raise Exception(f'''No se han podido crear los
directorios.
                Motivo {resultado.stderr}''')

if __name__ == '__main__':
    main()

```

12.6 Clasificar ficheros por la longitud de la extensión

El comando **ls** muestra el contenido del directorio actual. Crea un script llamado `clasificar_por_extensión.py` que muestre primero los ficheros sin extensión (incluidos directorios), luego los ficheros en los que la extensión es un carácter, luego los que la extensión son dos caracteres, luego los de tres caracteres, y así sucesivamente.

Hay que emplear obligatoriamente `subprocess.run()`.

```
#!/usr/bin/python3

import subprocess
from typing import Dict, List

def main():
    try:
        clasificacion: List[Dict] = clasificar_por_extension()
        imprimir_clasificacion(clasificacion)
    except Exception as error:
        print(f'\tERROR: {error}')

def clasificar_por_extension() -> List[Dict]:
    def obtener_lista_ficheros() -> List[str]:
        resultado_ls = subprocess.run(
            ['ls'],
            text=True,
            capture_output=True
        )
        if resultado_ls.returncode != 0:
            raise Exception(
                'Se ha producido un error: ',
                f'Información técnica: "{resultado_ls.stderr}"')

        ficheros_en_lista = resultado_ls.stdout[0:-1].split('\n')
        return ficheros_en_lista

    def inicializar_lista_clasificacion(limite: int) -> List[List]:
        clasificacion = []

        for i in range(0, limite):
            clasificacion.append([])

        return clasificacion

    def completar_lista_clasificacion(
        ficheros: List[str],
        clasificacion: List[List]
    ) -> List[List[str]]:

        for fichero in ficheros:
            fichero_lista = fichero.split('.')
            extension = fichero_lista[len(fichero_lista) - 1]
            clasificacion[len(extension)].append(fichero_lista[0])

        return clasificacion

    def convertir_a_diccionario(
        clasificacion: List[List]) -> List[Dict]:
        clasificacion_diccionario = []
        for i in range(0, 10):
```

```

        clasificacion_diccionario.append(
            {'longitud': i, 'ficheros': clasificacion[i]})
    return clasificacion_diccionario

ficheros_en_lista = obtener_lista_ficheros()
clasificacion = inicializar_lista_clasificacion(10)
clasificacion = completar_lista_clasificacion(
    ficheros_en_lista, clasificacion)
clasificacion_diccionario =
    convertir_a_diccionario(clasificacion)
return clasificacion_diccionario

def imprimir_clasificacion(clasificacion: List[Dict]) -> None:
    for elemento in clasificacion:
        print(f'\tExtensión con longitud {elemento["longitud"]}: ')
        for fichero in elemento["ficheros"]:
            print(f'\t\t{fichero}')

if __name__ == '__main__':
    main()

```

Solución alternativa, ya que el mismo problema se puede solucionar de variadas formas:

```

#!/usr/bin/python3

import subprocess

def main():
    print()

    ficheros = obtener_lista_de_ficheros()
    extension_mas_larga = obtener_longitud_extension_mas_larga(ficheros)
    clasificar_por_longitud(ficheros, extension_mas_larga)

def clasificar_por_longitud(ficheros, extension_mas_larga):
    for i in range(0, extension_mas_larga+1):
        print(f'Los ficheros con extensión de longitud {i} son:')

        for fichero in ficheros:
            fichero_en_partes = fichero.split('.')

            if i == 0:
                if len(fichero_en_partes) == 1:
                    print('\t', fichero)
            else:
                if len(fichero_en_partes) > 1:
                    if i == len(fichero_en_partes)-1:
                        print('\t', fichero)

def obtener_lista_de_ficheros():
    resultado = subprocess.run(['ls'], text=True, capture_output=True)
    ficheros = resultado.stdout.split('\n')[0:-1]
    return ficheros

def obtener_longitud_extension_mas_larga(ficheros):

```

```

extension_mas_larga = 0
for fichero in ficheros:
    fichero_en_partes = fichero.split('.')
    if len(fichero_en_partes) == 0:
        continue
    if len(fichero_en_partes) > 1:
        if len(fichero_en_partes[1]) > extension_mas_larga:
            extension_mas_larga = len(fichero_en_partes[1])

return extension_mas_larga

if __name__ == '__main__':
    main()

```

12.7 Comprobar años bisiestos

El comando "date" permite trabajar con fechas. Por medio de sus múltiples opciones podemos averiguar mucha información sobre fechas.

En este ejercicio necesitamos saber si un año es o no es bisiesto. Tras realizar pruebas con el comando date nos damos cuenta de que:

date --date=2004-02-29 devuelve 29 de febrero de 2004

date --date=2005-02-29 devuelve 1 de marzo de 2005

así que si el año no es bisiesto al suministrar el día 29 avanza al día 1 y no da error.

Desarrolla un script llamado comprobar_bisiesto.py que reciba como argumento un número indeterminado de años y nos indique si cada uno de ellos es o no es un año bisiesto. Por ejemplo:

python3 comprobar_bisiesto.py 2005, 1993, 1938, 2018

Todo debe hacerse con subprocess.run, no se permite usar otras librerías.

```

#!/usr/bin/python3

import subprocess
import sys

def main():
    argumentos = sys.argv[1:]
    comprobar_si_los_años_del_argumento_son_bisiestos_e_imprimir(argumentos)

def comprobar_si_los_años_del_argumento_son_bisiestos_e_imprimir(argumentos):
    for argumento in argumentos:
        try:
            año: int = int(argumento)
            if año < 0:
                print(f'ERROR: El año {argumento} no puede ser menor de cero')
                continue
            if año > 3000:

```

```

        print(f'ERROR: El año {argumento} no puede ser mayor de 3000')
        continue
    except Exception as error:
        print(f'ERROR: El argumento {argumento} no es un año')
        continue

    if es_año_bisiesto(año):
        print(f'El año {año} es bisiesto')
    else:
        print(f'El año {año} no es bisiesto')

def es_año_bisiesto(año: int) -> bool:
    resultado = subprocess.run(
        ['date', f'--date={año}-02-29'],
        text=True,
        capture_output=True
    )

    if resultado.stdout.split(' ')[2] == '29':
        return True
    else:
        return False

if __name__ == '__main__':
    main()

```

12.8 Crear varios usuarios

Este ejercicio necesita un sistema operativo completo, por ello tiene que ser ejecutado con VirtualBox o VMware, no puede probarse en Docker.

Desarrolla un script en Linux (en Unix y MacOS debe de funcionar también) llamado **crear_usuarios.py**. Recibirá como argumento un número indeterminado de nombres de usuario que no existen en el sistema. Por ejemplo:

```
python3 crear_usuarios.py jimena juan hugo raquel oscar sara
```

Todo debe hacerse con `subprocess.run`, no se permite usar otras librerías.

El comando para crear usuarios en Linux es **useradd**. Tendrás que ver el manual con **man useradd** y buscar las opciones necesarias para lo que se pide a continuación.

Para cada usuario:

- debe crear un directorio con el mismo nombre del usuario dentro del directorio `/home` (directorio que almacena los directorios personales de los usuarios),
- crear el usuario asociándolo a su directorio personal (buscar en la ayuda del comando `useradd` la opción adecuada),
- averiguar el año en el que estamos con el comando `date`,
- indicar que la cuenta estará bloqueada el 30 de junio del año en que se ejecute el script (buscar en la ayuda del comando `useradd` la opción adecuada).

```
#!/usr/bin/python3
```

```

import sys
import subprocess
from typing import List

def comprobar_argumentos():
    ...

def obtener_año_actual() -> str:
    resultado = subprocess.run(["date", "+%Y"], text=True, capture_output=True)
    if resultado.returncode != 0:
        raise Exception(f"No se ha podido obtener el año: {resultado.stderr}")

    return resultado.stdout.strip()

def obtener_fecha_bloqueo(año_actual: str) -> str:
    fecha_bloqueo = f"{año_actual}-06-30"
    return fecha_bloqueo

def crear_usuario(usuario: str) -> None:
    try:
        fecha = obtener_fecha_bloqueo(obtener_año_actual())
    except Exception as error:
        print(error)

    resultado = subprocess.run(
        ["useradd", "-m", "-e", fecha, usuario],
        text=True,
        capture_output=True
    )
    if resultado.returncode == 0:
        return
    elif resultado.returncode == 1:
        raise Exception(
            "Fichero de usuarios bloqueado, ejecuta el script con sudo")
    elif resultado.returncode == 3:
        raise Exception(f"Uno de los argumentos no es válido")
    elif resultado.returncode == 4:
        raise Exception(f"ID de usuario en uso")
    elif resultado.returncode == 9:
        raise Exception(f"Nombre de usuario {usuario} en uso")
    elif resultado.returncode == 12:
        raise Exception(f"No se puede crear directorio home")
    elif resultado.returncode == 14:
        raise Exception(f"No se puede actualizar SELinux user mapping")

def crear_usuarios(usuarios: List[str]) -> None:
    for usuario in usuarios:
        try:
            crear_usuario(usuario)
        except Exception as error:
            print(error)

def main():
    usuarios = sys.argv[1:]
    crear_usuarios(usuarios)

```

```
if __name__ == "__main__":  
    main()
```

12.9 Obtener usuarios reales del sistema

El fichero **/etc/passwd** es un fichero de configuración de Unix que almacena los usuarios creados en el sistema más alguna otra información.

El comando **cat** se usa habitualmente para ver el contenido de un fichero.

Ejecutamos el comando **cat /etc/passwd**. Si vemos el contenido de este fichero observaremos que cada línea del fichero es un usuario. Para cada usuario, hay varios campos de información separados por dos puntos (:).

El tercer campo es un número que almacena el identificador único de usuario. Si el número es 1000 o mayor es un usuario normal (un usuario que puede iniciar una sesión), mientras que los números menores de 1000 son usuarios de tipo administrativo.

El quinto campo es el directorio personal del usuario.

El sexto campo es el intérprete de comandos usado por el usuario.

Desarrolla un script llamado **mostrar_usuarios.py** que muestre los usuarios normales, su identificador de usuario, su directorio personal y el intérprete de comandos que emplea.

```
#!/usr/bin/python3  
  
import subprocess  
import sys  
from typing import List  
  
def main():  
    mostrar_usuarios_reales()  
  
def mostrar_usuarios_reales():  
    CAMPO_LOGIN = 0  
    CAMPO_IDENTIFICADOR = 2  
    CAMPO_DIRECTORIO_PERSONAL = 5  
    CAMPO_INTERPRETE_COMANDOS = 6  
  
    try:  
        usuarios = obtener_lista_usuarios()  
    except Exception as error:  
        print(error)  
        sys.exit(1)  
  
    for linea in usuarios:  
        campos: str = linea.split(':')
```



```

        codigo: int = int(campos[CAMPO_IDENTIFICADOR])
        if codigo >= 1000:
            print(
                campos[CAMPO_LOGIN],
                campos[CAMPO_DIRECTORIO_PERSONAL],
                campos[CAMPO_INTERPRETE_COMANDOS]
            )

def obtener_lista_usuarios() -> List[str]:
    resultado = subprocess.run(
        ['cat', '/etc/passwd'],
        text=True,
        capture_output=True
    )

    if resultado.returncode != 0:
        raise Exception('No ha funcionado el comando cat /etc/passwd')

    lineas_usuarios = resultado.stdout.split('\n')
    lineas_usuarios = lineas_usuarios[:-1]

    return lineas_usuarios

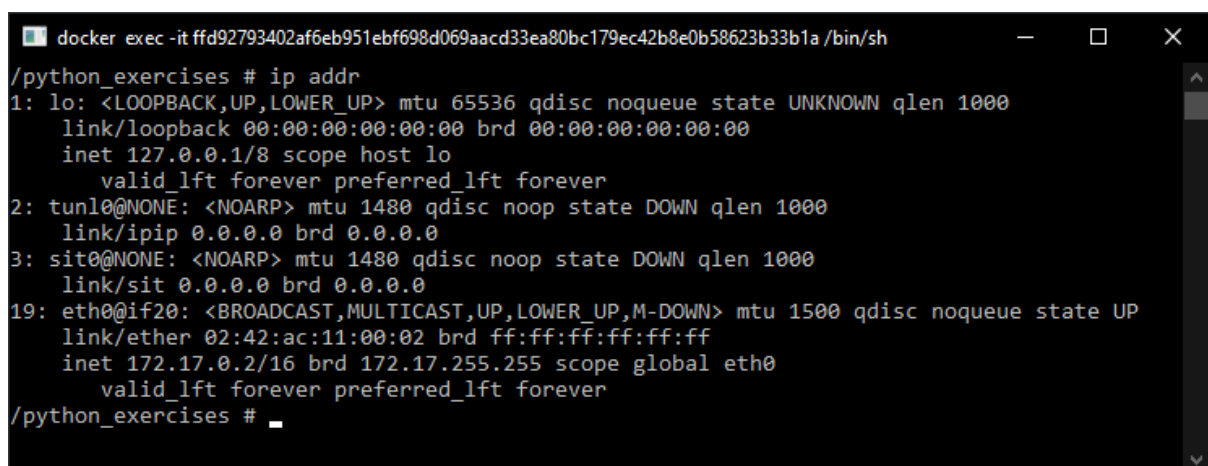
if __name__ == '__main__':
    main()

```

12.10 Obtener la dirección IP

Desarrolla un script denominado `dime_mi_direccion.py`.

El comando `ip` con la opción `addr` muestra información sobre todas las interfaces de red del equipo (tarjetas de red).



```

docker exec -it ffd92793402af6eb951ebf698d069aacd33ea80bc179ec42b8e0b58623b33b1a /bin/sh
/python_exercises # ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
2: tunl0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN qlen 1000
    link/ipip 0.0.0.0 brd 0.0.0.0
3: sit0@NONE: <NOARP> mtu 1480 qdisc noop state DOWN qlen 1000
    link/sit 0.0.0.0 brd 0.0.0.0
19: eth0@if20: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1500 qdisc noqueue state UP
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
        valid_lft forever preferred_lft forever
/python_exercises #

```

Este es el comando a ejecutar con `subprocess.run`.

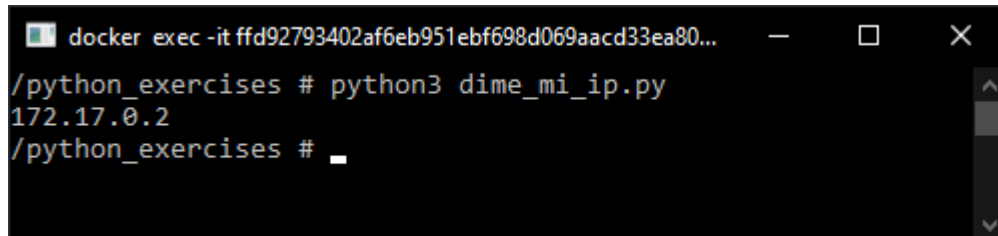
Vemos que devuelve mucha información. Yo solo quiero la dirección IP.

Si busco un patrón que me sirva de ayuda, veo que la dirección IP (172.17.0.2) está en una línea que incluye el texto "scope global eth0", así que vamos a forzar el resultado.

Apoyándote en las funciones para cadenas de texto:

- split
- find
- strip (ésta yo ya la he usado pero no es necesaria)

desarrolla un script que muestre solo la dirección IP.



```
docker exec -it ffd92793402af6eb951ebf698d069aacd33ea80...  
/python_exercises # python3 dime_mi_ip.py  
172.17.0.2  
/python_exercises # _
```

```
#!/usr/bin/python3  
  
import subprocess  
  
def main() -> None:  
    try:  
        print(extraer_direccion_ip())  
    except Exception as error:  
        print(error)  
  
def extraer_direccion_ip() -> str:  
    resultado = subprocess.run(  
        ['ip', 'addr'],  
        text=True,  
        capture_output=True  
    )  
    if resultado.returncode != 0:  
        raise Exception(  
            'No se ha podido recuperar información de los interfaces de red')  
  
    lineas = resultado.stdout.split('\n')  
  
    for linea in lineas:  
        posicion = linea.find('scope global eth0')  
        if posicion >= 0:  
            direccion_con_mascara = linea.strip().split(' ')[1]  
            direccion = direccion_con_mascara.split('/')[0]  
            return direccion  
  
if __name__ == '__main__':  
    main()
```

12.11 Mostrar el calendario de un mes y año

El comando **cal** muestra un calendario. Por ejemplo,

cal 1999 muestra el calendario de todo el año 1999.

cal 12 1999 muestra el calendario de diciembre del año 1999.

Desarrolla un script llamado *mostrar_calendario.py*. Recibirá distintos argumentos por la línea de comandos.

- Si recibe un único argumento debe ser un año y mostrar el calendario de dicho año.
- Si recibe dos argumentos el primero debe ser un mes expresado en español (enero, febrero, marzo, ...) y el segundo un año.

Hay que hacer por lo menos las siguientes comprobaciones de errores:

- Hay que comprobar que el año esté entre 1 y 9999.
- Hay que comprobar que el mes sea correcto.

```
#!/usr/local/bin/python3

import subprocess
import sys

def main():
    try:
        mes, año = obtener_argumentos()
        mostrar_calendario(mes, año)
    except Exception as error:
        print(error)

def mostrar_calendario(mes: int, año: int) -> None:
    if mes == None:
        resultado = subprocess.run(
            ['cal', str(año)]
        )
    else:
        resultado = subprocess.run(
            ['cal', str(mes), str(año)]
        )

    if resultado.returncode != 0:
        raise Exception('No se ha podido obtener el calendario')

def obtener_argumentos() -> tuple[int, int]:
    argumentos = sys.argv[1:]
    if len(argumentos) < 1:
        raise Exception('Pocos argumentos')

    if len(argumentos) > 2:
        raise Exception('Demasiados argumentos')

    mes_en_numero = None
    if len(argumentos) == 2:
        mes_en_letra = argumentos[0]
        try:
            mes_en_numero = convertir_mes_a_numero(mes_en_letra)
        except Exception as error:
            raise Exception('El argumento no es un mes')

    try:
        if len(argumentos) == 1:
            año = int(argumentos[0])
        else:
```

```

        año = int(argumentos[1])
    except:
        raise Exception('El argumento no es un año')

    if año < 1 or año > 9999:
        raise Exception('El año está fuera del rango permitido')

    return (mes_en_numero, año)

def convertir_mes_a_numero(mes: str) -> int:
    meses = (
        '', 'enero', 'febrero', 'marzo', 'abril', 'mayo', 'junio',
        'julio', 'agosto', 'septiembre', 'octubre', 'noviembre', 'diciembre'
    )

    for i in range(1, len(meses)):
        if meses[i] == mes:
            return i

    raise Exception('El argumento no es un mes')

if __name__ == '__main__':
    main()

```

12.12 Mostrar las rutas del PATH

Crea un script llamado `mostrar_path.py` que enumere los directorios incluidos en el PATH.

```

#!/usr/bin/python3

import subprocess

def main():
    limpiar_pantalla()
    valor_path = extraer_valor_linea_path()
    enumerar_directorios_path(valor_path)

def enumerar_directorios_path(path):
    directorios = path.split(':')

    print('Los directorios incluidos en el PATH son:')
    for i, directorio in enumerate(directorios):
        print(f'\t{i+1}) {directorio}')

def extraer_valor_linea_path():
    resultado = subprocess.run(
        ['env'],
        capture_output=True,
        text=True
    )

```

```

)

if resultado.returncode != 0:
    print('No se han podido recuperar las variables de entorno')
    print(resultado.stderr)
else:
    lineas = resultado.stdout.split('\n')

    for linea in lineas:
        linea_como_lista = linea.split('=')
        if linea_como_lista[0] == "PATH":
            return linea_como_lista[1]

def limpiar_pantalla():
    subprocess.run(['clear'])

if __name__ == "__main__":
    main()

```

12.13 Consumo de memoria por los procesos

El comando de Unix (Linux, MacOS o Powershell) **ps** muestra todos los procesos activos (programas actualmente en ejecución).

Este script hay que probarlo en la máquina virtual, en Docker el comando ps no funciona como necesito. O se trabaja directamente en una máquina virtual con entorno gráfico o hay que realizar la configuración del último día en clase con SSH.

Vamos a emplear el comando acompañado de las siguientes tres opciones que he visto en Google y que no se lo que hacen, pero estaría bien abrir el manual (man ps) y ver lo que hacen.

```
ps -a -u -x (también se puede poner px -aux)
```

Al ejecutar este comando para probarlo vemos que aparecen tantas líneas que solo vemos las últimas. Para ver las anteriores usamos:

```
ps -a -u -a | less
```

la barra vertical es el operador tubería, que convierte la salida del primer comando en la entrada del segundo comando. El comando less permite ver el contenido de un fichero y moverse por él usando los cursores. En este caso, al unir los tres elementos: ps, less y la tubería, lo que tenemos es que podemos ver todos los procesos usando los cursores para subir y bajar.

Dentro del script no es necesario usar la tubería y less, es solo para ver lo que sale.

Si subimos hasta la cabecera vemos que hay una columna llamada VSZ. Esta columna muestra la cantidad de memoria principal asignada al proceso. Si nos movemos por la columna vemos que no todos los procesos tienen valor en dicha columna.

Hay que crear un script llamado mostrar_uso_memoria.py que muestre solo los procesos que tienen memoria principal asignada, ordenados de mayor a menor uso de la memoria, mostrando solo el comando, la memoria asignada y el propietario del proceso. Al final hay que mostrar también un resumen con el proceso que consume más memoria, menos memoria y la media de todos los procesos.

No tendría que contaros esto, ya que tendríais que encontrar vosotros la solución:

- al igual que el día del examen, al hacer subprocess.run devuelve lo deseado y una línea en blanco al final que hay que eliminar con un trim, que en Python se hace con el método strip().
- para hacer el split cuando puede haber un número indeterminado de espacios se pone solo split() sin argumentos.
- la primera fila es la cabecera informativa, hay que ignorarla.

El script no debe usar librerías de terceros para la gestión de procesos o para la ordenación de los procesos, es decir, tienes que usar subprocess.run y ordenar los procesos por tus propios medios usando bucles.

Si nos vemos con fuerzas también mostramos la memoria total consumida por cada usuario.

```
#!/usr/bin/python3

import subprocess

def main() -> None:
    show_memory_usage()

def show_memory_usage() -> None:
    result_ps = subprocess.run(
        ['ps', '-a', '-u', '-x'],
        text=True,
        capture_output=True
    )

    if result_ps.returncode != 0:
        raise Exception(f'Te command ps has not work properly: {result_ps.stderr}')

    processes_as_string = result_ps.stdout.strip()
    processes_that_use_memory = get_processes_that_use_memory(processes_as_string)
    show_processes_pretty(processes_that_use_memory)

def get_processes_that_use_memory(processes_as_string: str) -> list[list[str]]:
    processes_as_list = processes_as_string.split('\n')

    processes_that_use_memory = []
    USER_FIELD = 0
    MEMORY_FIELD = 4
    PROCCES_FIELD = 10

    for process in processes_as_list[1:]:
        process_as_list = process.split()
        used_memory = int(process_as_list[MEMORY_FIELD])
        if used_memory == 0:
            continue
        else:
            item_process = [
                process_as_list[PROCCES_FIELD],
                process_as_list[USER_FIELD],
                process_as_list[MEMORY_FIELD],
            ]
```

```

        processes_that_use_memory.append(item_process)

    return processes_that_use_memory

def show_processes_pretty(processes: list[list[str]]):
    PROCESS_FIELD = 0
    USER_FIELD = 1
    MEMORY_FIELD = 2

    print('PROCESS', 'OWNER', 'MEMORY')
    for process in processes:
        print('{:20} {:40} {:10}'.format(
            process[PROCESS_FIELD],
            process[USER_FIELD],
            process[MEMORY_FIELD]
        ))

if __name__ == '__main__':
    main()

```

12.14 Comprobar la existencia de usuarios

Desarrolla un script llamado `comprobar_usuarios.py`:

- que reciba por la línea de comandos un número indeterminado de nombres de usuario, pero siempre al menos un nombre de usuario.
- que para cada nombre informe si existe o no existe un usuario con dicho nombre.

Hay que usar la máquina virtual. En ella, el comando a emplear es:

```
cat /etc/passwd
```

Verás que aparece un montón de contenido:

- Cada línea del fichero corresponde a un usuario.
- Cada línea tiene varios campos separados por dos puntos `:`.
- El primer campo de cada línea es el nombre de usuario donde tienes que hacer la búsqueda.

En el desarrollo del programa:

- Verifica que se recibe al menos un nombre como argumento.
- Intenta gestionar como constantes varios tipos de error que tengan sentido en este ejercicio.
- Verifica que el comando se ejecuta correctamente.
- Si falla intenta lanzar una excepción.
- Tendrás que procesar la salida con `split` de la forma adecuada.

```
#!/usr/bin/python3
```



```

import subprocess
import sys

def main():
    limpiar_pantalla()

    argumentos = recuperar_argumentos()
    usuarios_del_sistema = extraer_usuarios_del_sistema()
    informar_si_los_usuarios_pertenecen_al_sistema(argumentos,
usuarios_del_sistema)

def informar_si_los_usuarios_pertenecen_al_sistema(usuarios,
usuarios_del_sistema):
    for usuario in usuarios:
        if usuario in usuarios_del_sistema:
            print(f'El usuario {usuario} existe en el sistema')
        else:
            print(f'El usuario {usuario} no existe en el sistema')

def extraer_usuarios_del_sistema() -> list[str]:
    resultado = subprocess.run(
        ['cat', '/etc/passwd'],
        capture_output=True,
        text=True
    )

    if resultado.returncode != 0:
        print('ERROR')
        sys.exit(2)

    lineas = resultado.stdout.split('\n')

    usuarios = []
    for linea in lineas:
        usuario = linea.split(':')[0]
        usuarios.append(usuario)

    return usuarios[0:len(usuarios) - 1]

def recuperar_argumentos():
    argumentos = sys.argv[1:]

```

```

    if len(argumentos) == 0:
        print('ERROR')
        sys.exit(100)
    else:
        return argumentos

def limpiar_pantalla():
    subprocess.run(['clear'])

if __name__ == '__main__':
    main()

```

12.15 Creación de múltiples usuarios

Desarrolla un script llamado `crear_usuarios.py`, cuyo objetivo es crear varios usuarios y su directorio personal.

Como hay tareas administrativas el script se ejecuta con permisos de administrador, por ejemplo:

```
"sudo python3 crear_usuarios.py lucia david jaime"
```

Es importante que el código esté organizado en funciones. Las funciones facilitan la programación al descomponer una tarea compleja en varias tareas más sencillas.

Todas las operaciones deben incorporar un adecuado tratamiento que informe de modo claro al usuario lo que está ocurriendo, si un recurso se crea, y hay un error, ...

El comando para crear un usuario es "useradd".

En la línea de comandos debe recibir al menos el nombre de un usuario.

Hay que crear un directorio en "/home" con el mismo nombre del usuario. El comando para crear directorios es "mkdir".

Todos los usuarios deben tener la contraseña "Hola1234". El problema es que hay que suministrarla encriptada. El comando para ello es "mkpasswd".

Averigua con la ayuda del comando "useradd" que opciones emplear para indicarle el directorio personal y la contraseña.

Tras crear un usuario indicando su directorio personal, resulta que el usuario no puede emplear dicho directorio porque no tiene la propiedad del directorio. Emplea el comando "chown" para indicar que el usuario es el propietario de su directorio.

Además de la propiedad, hay que modificar los permisos de acceso al directorio. Emplea el comando "chmod" para indicar que el propietario puede leer, escribir y ejecutar el directorio.

```
#!/usr/bin/python3
```

```

import subprocess
import sys

ERROR_HAY_UN_PROBLEMA_CON_LOS_ARGUMENTOS = 1
ERROR_NO_SE_HAN_PODIDO_CREAR_LOS_USUARIOS = 2

def main():
    limpiar_pantalla()

    # if not son_argumentos_correctos(sys.argv[1:]):
    #     print('ERROR')
    #     sys.exit(ERROR_HAY_UN_PROBLEMA_CON_LOS_ARGUMENTOS)

    try:
        nombres_usuarios = extraer_usuarios()
    except Exception as error:
        print(f'ERROR: {error}')
        sys.exit(ERROR_HAY_UN_PROBLEMA_CON_LOS_ARGUMENTOS)

    try:
        crear_usuarios(nombres_usuarios)
    except Exception as error:
        print(f'ERROR: {error}')
        sys.exit(ERROR_NO_SE_HAN_PODIDO_CREAR_LOS_USUARIOS)

# def son_argumentos_correctos(argumentos) -> bool:
#     if len(argumentos) == 0:
#         return False
#     else:
#         return True

def extraer_usuarios() -> list[str]:
    argumentos = sys.argv[1:]

    if len(argumentos) < 2:
        # return []
        # return None
        raise Exception('Hay que suministrar al menos un usuario')

    try:
        cantidad = int(argumentos[0])
    except Exception as error:

```

```

        raise Exception('El primer argumento no es un número')

# if not argumentos[0].isnumeric():
#     raise Exception('El primer argumento no es un número')

if cantidad != len(argumentos) - 1:
    raise Exception('No coinciden la cantidad y el número de nombres')

def limpiar_pantalla():
    subprocess.run(['clear'])

def crear_usuarios(usuarios: list[str]):
    for usuario in usuarios:
        try:
            crear_directorio(usuario)
            contraseña_encriptada = encriptar_contraseña('Hola1234')
            crear_usuario(usuario, contraseña_encriptada,
f'/home/{usuario}')
            cambiar_propietario(usuario, f'/home/{usuario}')
            cambiar_permisos('700', f'/home/{usuario}')
        except Exception as error:
            print (error)

def crear_directorio(directorio):
    resultado = subprocess.run([
        'mkdir', directorio],
        capture_output=True,
        text=True,
        cwd='/home'
    )

    if resultado.returncode != 0:
        raise Exception(f'No se ha podido crear el directorio:
[resultado.returncode] {resultado.stderr}')

def encriptar_contraseña(contraseña):
    resultado = subprocess.run([
        'mkpasswd', contraseña],
        capture_output=True,
        text=True,
    )

```

```

        if resultado.returncode != 0:
            raise Exception(f'No se ha podido encriptar la contraseña:
[{{resultado.returncode}}] {{resultado.stderr}}')

        return resultado.stdout.strip()

def crear_usuario(usuario, contraseña, directorio):
    resultado = subprocess.run(
        ['useradd', '-p', contraseña, '-d', directorio, usuario],
        capture_output=True,
        text=True,
    )

    if resultado.returncode != 0:
        raise Exception(f'No se ha podido crear el usuario:
[{{resultado.returncode}}] {{resultado.stderr}}')

def cambiar_propietario(usuario, directorio):
    resultado = subprocess.run(
        ['chown', usuario, directorio],
        capture_output=True,
        text=True,
    )

    if resultado.returncode != 0:
        raise Exception(f'No se ha podido cambiar el propietario del
directorio {{directorio}} a {{usuario}}: [{{resultado.returncode}}]
{{resultado.stderr}}')

def cambiar_permisos(permisos, directorio):
    resultado = subprocess.run(
        ['chmod', permisos, directorio],
        capture_output=True,
        text=True,
    )

    if resultado.returncode != 0:
        raise Exception(f'No se han podido cambiar los permisos:
[{{resultado.returncode}}] {{resultado.stderr}}')

```

```
if __name__ == '__main__':  
    main()
```

13 Scripts para redes

13.1 Conversión entre decimal y binario

Esta librería contiene funciones que realizan la conversión de decimal a binario y viceversa.

La conversión de decimal a binario se desarrolla con el método de divisiones sucesivas. La conversión de binario a decimal se desarrolla con el método de sumas de potencias.

```
# conversion.py
def convertir_entero_decimal_a_binario(numero_decimal: int) -> str:
    binario = ''
    cociente = numero_decimal

    while (cociente >= 2):
        numero = cociente
        cociente = numero // 2
        resto = numero % 2
        binario = str(resto) + binario

    binario = str(cociente) + binario
    return binario

def convertir_entero_binario_a_decimal(numero_binario: str) -> int:
    numero_decimal = 0
    numero_decimal2 = 0

    for i in range(len(numero_binario) - 1, -1, -1):
        bit = numero_binario[len(numero_binario) - 1 - i]
        if bit == '1':
            peso = 2 ** i
            numero_decimal += peso

    return numero_decimal
```

Los test unitarios los realizamos con el siguiente código.

```
# test.py
import unittest
from conversion import convertir_entero_decimal_a_binario, convertir_entero_binario_a_decimal

class TestConversion(unittest.TestCase):

    def test_decimal_a_binario(self):
        self.assertEqual(convertir_entero_decimal_a_binario(0), '0')
        self.assertEqual(convertir_entero_decimal_a_binario(1), '1')
        self.assertEqual(convertir_entero_decimal_a_binario(255), '11111111')
        self.assertEqual(convertir_entero_decimal_a_binario(254), '11111110')
        self.assertEqual(convertir_entero_decimal_a_binario(8), '1000')
        self.assertEqual(convertir_entero_decimal_a_binario(9), '1001')

        self.assertEqual(convertir_entero_binario_a_decimal('0'), 0)
        self.assertEqual(convertir_entero_binario_a_decimal('1'), 1)
        self.assertEqual(convertir_entero_binario_a_decimal('11111111'), 255)
        self.assertEqual(convertir_entero_binario_a_decimal('11111110'), 254)
        self.assertEqual(convertir_entero_binario_a_decimal('1000'), 8)
        self.assertEqual(convertir_entero_binario_a_decimal('1001'), 9)

if __name__ == '__main__':
    unittest.main()
```

13.2 Traducción de la máscara entre formato abreviado y largo

13.2.1 Versión 1

Este script traduce una máscara en formato largo a su representación en formato abreviado, y viceversa.

El script se llama `traducir_mascara.py`. Recibe un único argumento que es la máscara en uno de los dos posibles formatos de representación, por ejemplo, `255.254.0.0`, o por ejemplo `15`, averigua en que formato está y lo traduce al otro formato.

```
#!/usr/bin/python3

from ast import arg
import sys

def principal() -> None:
    try:
        argumentos = sys.argv[1:]
        posible_mascara = extraer_argumento(argumentos)

        if not es_mascara(posible_mascara):
            print('ERROR: El argumento no es una máscara')

        if es_mascara_abreviada(posible_mascara):
            mascara_larga = traducir_mascara_abreviada_a_larga(posible_mascara)
            print(
                f'La máscara abreviada {posible_mascara} se corresponde con la máscara larga {mascara_larga}')
        else:
            mascara_abreviada = traducir_mascara_larga_a_abreviada(
                posible_mascara)
            print(
                f'La máscara larga {posible_mascara} se corresponde con la máscara abreviada {mascara_abreviada}')

    except ArgumentosError as error:
        print(error)

def extraer_argumento(argumentos: list[str]) -> str:
    if len(argumentos) != 1:
        raise ArgumentosError('Número de argumentos erróneo')

    return argumentos[0]

def es_mascara(mascara: str) -> bool:
    if es_mascara_abreviada(mascara):
        return True

    if es_mascara_larga(mascara):
        return True

    return False

def es_mascara_abreviada(mascara: str) -> bool:
    try:
        mascara_numero = int(mascara)
        if mascara_numero < 1 or mascara_numero > 30:
            return False

        return True
    except ValueError:
        return False

def es_mascara_larga(mascara: str) -> bool:
    mascara_troceada = mascara.split('.')
    if len(mascara_troceada) != 4:
        return False
    for troceado in mascara_troceada:
        if not troceado.isdigit():
            return False
    return True

def traducir_mascara_abreviada_a_larga(mascara: str) -> str:
    mascara_numero = int(mascara)
    mascara_troceada = []
    while mascara_numero > 0:
        troceado = mascara_numero % 256
        mascara_troceada.append(troceado)
        mascara_numero = mascara_numero // 256
    mascara_troceada.reverse()
    mascara_larga = '.'.join(str(troceado) for troceado in mascara_troceada)
    return mascara_larga

def traducir_mascara_larga_a_abreviada(mascara: str) -> str:
    mascara_troceada = mascara.split('.')
    mascara_numero = 0
    for troceado in mascara_troceada:
        mascara_numero = (mascara_numero * 256) + int(troceado)
    return str(mascara_numero)

if __name__ == '__main__':
    principal()
```



```

    if len(mascara_troceada) != 4:
        return False

    for trozo in mascara_troceada:
        try:
            numero = int(trozo)

            if numero < 0 or numero > 255:
                return False
        except ValueError:
            return False

    return True

def traducir_mascara_abreviada_a_larga(mascara_abreviada: str) -> str:
    numero = int(mascara_abreviada)

    octetos_hechos = 0
    mascara_larga = ''

    while (numero >= 8):
        mascara_larga += '255.'
        octetos_hechos += 1
        numero -= 8

    octeto = 0
    for p in range(7, -1, -1):
        if numero > 0:
            octeto += 2 ** p
            numero -= 1
    mascara_larga += str(octeto)
    octetos_hechos += 1
    if octetos_hechos < 4:
        mascara_larga += '.'

    for i in range(octetos_hechos + 1, 5):
        mascara_larga += '0.' if i < 4 else '0'

    return mascara_larga

def traducir_mascara_larga_a_abreviada(mascara_larga: str) -> str:
    mascara_abreviada = 0

    octetos_mascara = mascara_larga.split('.')
    for octeto in octetos_mascara:
        binario: str = convertir_entero_decimal_a_binario(int(octeto))
        for i in range(0, len(binario)):
            if binario[i] == '1':
                mascara_abreviada += 1

    return str(mascara_abreviada)

def convertir_entero_decimal_a_binario(numero_decimal: int) -> str:
    binario = ''
    cociente = numero_decimal

    while (cociente >= 2):
        numero = cociente
        cociente = numero // 2
        resto = numero % 2
        binario = str(resto) + binario

    binario = str(cociente) + binario
    return binario

class ArgumentosError(Exception):
    pass

class MascaraError(Exception):
    pass

```

```
if __name__ == '__main__':  
    principal()
```

13.2.2 Versión 2

Este script traduce una máscara en formato largo a su representación en formato abreviado, y viceversa.

El script se llama traducir_mascara.py. Recibe un único argumento que es la máscara en uno de los dos posibles formatos de representación, por ejemplo, 255.254.0.0, o por ejemplo 15, averigua en que formato está y lo traduce al otro formato.

```
#!/usr/bin/python3  
  
import sys  
  
def principal() -> None:  
    try:  
        argumentos = sys.argv[1:]  
        posible_mascara = extraer_argumento(argumentos)  
  
        if not es_mascara(posible_mascara):  
            print('ERROR: El argumento no es una máscara')  
  
        if es_mascara_abreviada(posible_mascara):  
            mascara_larga = traducir_mascara_abreviada_a_larga(posible_mascara)  
            print(  
                f'La máscara abreviada {posible_mascara} se corresponde con la máscara larga  
{mascara_larga}')  
        else:  
            mascara_abreviada = traducir_mascara_larga_a_abreviada(  
                posible_mascara)  
            print(  
                f'La máscara larga {posible_mascara} se corresponde con la máscara abreviada  
{mascara_abreviada}')  
  
    except ArgumentosError as error:  
        print(error)  
  
def extraer_argumento(argumentos: list[str]) -> str:  
    if len(argumentos) != 1:  
        raise ArgumentosError('Número de argumentos erróneo')  
  
    return argumentos[0]  
  
def es_mascara(mascara: str) -> bool:  
    if es_mascara_abreviada(mascara):  
        return True  
  
    if es_mascara_larga(mascara):  
        return True  
  
    return False  
  
def es_mascara_abreviada(mascara: str) -> bool:  
    try:  
        mascara_numero = int(mascara)  
        if mascara_numero < 1 or mascara_numero > 30:  
            return False  
  
        return True  
    except ValueError:
```

```

        return False

def es_mascara_larga(mascara: str) -> bool:
    mascara_troceada = mascara.split('.')
    if len(mascara_troceada) != 4:
        return False

    for trozo in mascara_troceada:
        try:
            numero = int(trozo)

            if numero < 0 or numero > 255:
                return False
        except ValueError:
            return False

    return True

def traducir_mascara_abreviada_a_larga(mascara_abreviada: str) -> str:
    numero = int(mascara_abreviada)

    primer_byte = 0
    segundo_byte = 0
    tercer_byte = 0
    cuarto_byte = 0

    if numero >= 8:
        primer_byte = 255
    if numero >= 16:
        segundo_byte = 255
    if numero >= 24:
        tercer_byte = 255

    resto = numero % 8
    byte = 0
    for i in range(0, resto):
        byte += 2 ** (7 - i)

    if primer_byte == 0:
        primer_byte = byte
    elif segundo_byte == 0:
        segundo_byte = byte
    elif tercer_byte == 0:
        tercer_byte = byte
    else:
        cuarto_byte = byte

    return f'{primer_byte}.{segundo_byte}.{tercer_byte}.{cuarto_byte}'

def traducir_mascara_larga_a_abreviada(mascara_larga: str) -> str:
    bytes_a_abreviada = {'255': 8, '254': 7, '252': 6,
                        '248': 5, '240': 4, '224': 3, '192': 2, '128': 1, '0': 0}
    mascara_abreviada = 0

    bytes_mascara = mascara_larga.split('.')
    for byte in bytes_mascara:
        mascara_abreviada += bytes_a_abreviada[byte]

    return str(mascara_abreviada)

class ArgumentosError(Exception):
    pass

class MascaraError(Exception):
    pass

if __name__ == '__main__':
    principal()

```

13.3 Información de la configuración de red

Este script muestra la configuración de la tarjeta de red. No recibe argumentos.

```
#!/usr/bin/python3

import subprocess
from dataclasses import dataclass
import re

def main() -> None:
    subprocess.run(['clear'])

    try:
        physical_address = get_physical_address()
        print(f'Dirección física: {physical_address}')

        mask = get_mask_address()
        print(f'Máscara de red: {mask}')

        router_address = get_router_address()
        print(f'Puerta de enlace: {router_address}')

        broadcast_address = get_broadcast_address()
        print(f'Dirección de difusión: {broadcast_address}')

        dns_server_addresses = get_dns_servers()
        print(f'Servidores de nombres de dominio DNS: {dns_server_addresses}')
    except Exception as error:
        print(f'ERROR: {error}')

def get_physical_address() -> str:
    try:
        lines = execute_command_and_get_result_as_list('ip addr')
        line = find_line_with_physical_address(lines)
        address = extract_physical_address_from_line(line)
        return address
    except Exception as error:
        raise Exception(f'Cannot get physical address: {error}')

def extract_physical_address_from_line(line: str) -> str:
    address_and_mask = line.split(' ')[1]
    address = address_and_mask.split('/')[0]
    return address

def find_line_with_physical_address(lines: list[str]) -> str:
    # list comprehension
    def contains_physical_address(line: str) -> bool:
        # pattern = re.compile('global eth0')
        # is_there_pattern = pattern.match(line)
        search_result = re.search('global eth0', line)
        return False if search_result is None else True

    filtered = [line for line in lines if contains_physical_address(line)]
    return filtered[0].strip()
```

```

def get_mask_address() -> str:
    try:
        lines = execute_command_and_get_result_as_list('ip addr')
        line = find_line_with_physical_address(lines)
        short_mask = extract_short_mask_from_line(line)
        long_mask = convert_short_mask_to_long(short_mask)

        return long_mask
    except Exception as error:
        raise Exception(f'Cannot get physical address: {error}')

def convert_short_mask_to_long(short_mask: str) -> str:
    if short_mask == '8':
        return '255.0.0.0'
    elif short_mask == '16':
        return '255.255.0.0'
    elif short_mask == '24':
        return '255.255.255.0'
    else:
        return short_mask

def extract_short_mask_from_line(line: str) -> str:
    address_and_mask = line.split(' ')[1]
    mask = address_and_mask.split('/')[1]
    return mask

def get_router_address() -> str:
    try:
        lines = execute_command_and_get_result_as_list('ip route')
        line = find_line_with_router_address(lines)
        address = extract_router_address_from_line(line)
        return address
    except Exception as error:
        raise Exception(f'Cannot get router address: {error}')

def find_line_with_router_address(lines: list[str]) -> str:
    # list comprehension
    def contains_router_address(line: str) -> bool:
        search_result = re.search('default via', line)
        return False if search_result is None else True

    filtered = [line for line in lines if contains_router_address(line)]
    return filtered[0].strip()

def extract_router_address_from_line(line: str) -> str:
    address = line.split(' ')[2]
    return address

def get_broadcast_address() -> str:
    try:
        lines = execute_command_and_get_result_as_list('ip addr')
        line = find_line_with_physical_address(lines)
        address = extract_broadcast_address_from_line(line)
        return address
    except Exception as error:
        raise Exception(f'Cannot get router address: {error}')

def extract_broadcast_address_from_line(line: str) -> str:
    address = line.split(' ')[3]
    return address

```

```

def get_dns_servers() -> list[str]:
    try:
        lines = execute_command_and_get_result_as_list(
            'grep nameserver /etc/resolv.conf')
        addresses = extract_dns_server_addresses_from_lines(lines)
        return addresses
    except Exception as error:
        raise Exception(f'Cannot get physical address: {error}')

def extract_dns_server_addresses_from_lines(lines: str) -> str:
    addresses = list(map(lambda line: line.split(" ")[1], lines))
    return addresses

def execute_command_and_get_result_as_list(command: str) -> str:
    command_as_list: list[str] = command.split(' ')

    result = subprocess.run(command_as_list, text=True, capture_output=True)
    if result.returncode != 0:
        raise Exception(f'Cannot execute command: {command}')

    lines: list[str] = result.stdout.strip().split('\n')

    return lines

if __name__ == '__main__':
    main()

```

14 IBM. Encontrar ficheros y mostrar sus permisos

El siguiente script pide un patrón de búsqueda para ficheros, los busca en el directorio actual y muestra sus permisos.

```
import stat, sys, os, string, subprocess
from terminal_colors import TerminalColors
try:
    pattern = input('Enter the file pattern to search for: ')
    command = f'find {pattern}'
    output = subprocess.run(
        command,
        shell=True,
        text=True,
        capture_output=True
    )
    print('\n', output)

    find_results = output.stdout.splitlines()
    print('\n', find_results)

    print('Files:')
    print('=====')
    for file in find_results:
        print(f'{TerminalColors.BLACK_BLUE}\nPermissions for file {file}:')
        mode = stat.S_IMODE(os.lstat(file)[stat.ST_MODE])
        for level in 'USR', 'GRP', 'OTH':
            for perm in 'R', 'W', 'X':
                if mode & getattr(stat, 'S_-' + perm + level):
                    print(TerminalColors.BLACK_GREEN,
                        level, ' has ', perm, ' permission')
            else:
                print(TerminalColors.BLACK_RED,
                    level, ' does NOT have ', perm, ' permission')
except Exception as exception:
    print('There was a problem - check the message.')
    print(f'[Type] {type(exception)}')
    print(f'[Arguments]: {exception.args}')
    print(f'[Exception] {exception}')
```

15 IBM. Mostrar credenciales cuyo usuario o contraseña incumplen las normas

El siguiente script obtenido de IBM revisa las credenciales de acceso de los distintos usuarios y muestra los nombre de usuario inválidos y las contraseñas inválidas.

El módulo **pwd** provee acceso a la base de datos de usuarios y contraseñas de Unix. En este script empleamos el método **getpwall** para obtener una lista con toda la información de los usuarios. Cada elemento de esta lista es un objeto en el que emplearemos las propiedades que ocupan las posiciones 0 y 1 del objeto: **pw_name** y **pw_passwd**.

```
root@ubuntu: ~/Escritorio/python3 /bin/python3 /home/usuario/Escritorio/python3/check_users_passwords.py
[pwd.struct_passwd(pw_name='root', pw_passwd='x', pw_uid=0, pw_gid=0, pw_gecos='root', pw_dir='/root', pw_shell='/bin/bash'),
pwd.struct_passwd(pw_name='daemon', pw_passwd='x', pw_uid=1, pw_gid=1, pw_gecos='daemon', pw_dir='/usr/sbin', pw_shell='/usr
/sbin/nologin'), pwd.struct_passwd(pw_name='bin', pw_passwd='x', pw_uid=2, pw_gid=2, pw_gecos='bin', pw_dir='/bin', pw_shell=
```

```
import pwd

#initialize counters
erroruser = []
errorpass = []

#get password database
passwd_db = pwd.getpwall()

try:
    #check each user and password for validity
    for entry in passwd_db:
        username = entry[0]
        password = entry[1]
        if len(username) < 6:
            erroruser.append(username)
        if len(password) < 8:
            errorpass.append(username)

    #print results to screen
    print("The following users have an invalid userid (less than six
characters):")
    for item in erroruser:
        print(item)
    print("\nThe following users have invalid password(less than eight
characters):")
    for item in errorpass:
        print(item)
except:
    print("There was a problem running the script.")
```

Código organizado en funciones

Organizamos el código en funciones para una mejor comprensión.

```
#!/usr/bin/python3
```



```

import pwd

MIN_LENGTH_LOGIN = 6
MIN_LENGTH_PASSWORD = 8

def main():
    try:
        users_passwords = get_users_and_passwords()

        invalid_users = get_invalid_users(users_passwords)
        show_invalid_users(invalid_users)

        invalid_passwords = get_invalid_passwords(users_passwords)
        show_invalid_passwords(invalid_passwords)

    except:
        print('There was a problem running the script.')

def get_users_and_passwords():
    passwd_db = pwd.getpwall()
    return passwd_db

def get_invalid_users(users_passwords):
    invalid_users = []

    for entry in users_passwords:
        if len(entry.pw_name) < MIN_LENGTH_LOGIN:
            invalid_users.append(entry.pw_name)

    return invalid_users

def get_invalid_passwords(users_passwords):
    invalid_passwords = []
    for entry in users_passwords:
        if len(entry.pw_passwd) < MIN_LENGTH_PASSWORD:
            invalid_passwords.append(entry.pw_name)

    return invalid_passwords

def show_invalid_users(invalid_users):
    print(f'\nThe following users have an invalid userid (less than {MIN_LENGTH_LOGIN} characters):')
    for item in invalid_users:
        print(f'\t{item}')

def show_invalid_passwords(invalid_passwords):
    print(f'\nThe following users have invalid password (less than {MIN_LENGTH_PASSWORD} characters):')
    for item in invalid_passwords:
        print(f'\t{item}')

```

```
if __name__ == '__main__':  
    main()
```