Práctica 11. Python para SysAdmins

```
# Ejercicio 1: creación de usuarios
#   Escribe un script que reciba un nombre de grupo (ejemplo: asir1) por
línea de
#   parámetros y realice las siguientes tareas:
#   • Compruebe que el grupo no exista. En ese caso:
#       • Asigne un GID libre desde el 8000 en adelante, si el 8000 y el
8001 están
#       ocupados, deberá asignar el 8002.
#       • Cree el grupo asir1 con el GID asignado utilizando el comando
groupadd.
#   • Cree 100 usuarios (UPG) utilizando el comando useradd, desde asir1-00
hasta asir1-
#   99. Además deberá:
#       • Fijar changeme como contraseña.
#       • Añadir al usuario al grupo asir1.
#       • Crear en el directorio HOME un fichero de bienvenida al sistema
llamado
#       welcome.ASIR1.user.txt.
#
#   Víctor Manuel Andreu Felipe 2025

import os
import subprocess
import sys

# buscamos los gid existentes a partir del 8000 hasta encontrar uno libre
def get_next_free_gid(start_gid=8000):
    existing_gids = {int(line.split(':')[2]) for line in
open('/etc/group')}
    while start_gid in existing_gids:
        start_gid += 1
    return start_gid

# comprobamos que el grupo exista
def group_exists(group_name):
    try:
        subprocess.run(["getent", "group", group_name], check=True,
stdout=subprocess.DEVNULL)
        return True
    except subprocess.CalledProcessError:
        return False

# creamos el grupo con el id correcto
def create_group(group_name, gid):
    subprocess.run(["sudo", "groupadd", "-g", str(gid), group_name],
check=True)
    print(f"Grupo '{group_name}' creado con GID {gid}.")
```

```python
# creación de usuarios con su home y grupo adecuado
def create_user(username, group_name):
    subprocess.run(["sudo", "useradd", "-m", "-G", group_name, username],
check=True)

    # asignacion de password con chpasswd a changeme
    subprocess.run(["sudo", "chpasswd"], input=f"
{username}:changeme\n".encode(), check=True)

    # creación de fichero de bienvenida
    home_dir = f"/home/{username}"
    welcome_file = os.path.join(home_dir, f"welcome.{group_name}.
{username}.txt")
    with open(welcome_file, "w") as f:
        f.write(f"Bienvenido al sistema, {username}!\n")
    subprocess.run(["sudo", "chown", f"{username}:{username}",
welcome_file])
    print(f"Usuario '{username}' creado y añadido a '{group_name}'.")

def main():
    if len(sys.argv) != 2:
        print("Uso: sudo python3 create_users_group.py <nombre_grupo>")
        sys.exit(1)

    group_name = sys.argv[1]

    # comprobación de grupo
    if group_exists(group_name):
        print(f"El grupo '{group_name}' ya existe.")
        sys.exit(1)

    gid = get_next_free_gid()
    create_group(group_name, gid)

    # iteración de usuarios
    for i in range(100):
        username = f"{group_name}-{i:02d}"
        create_user(username, group_name)

    print("Proceso de creación completado.")

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

```python
# Ejercicio 2: hash de ficheros
#   Crea un script que dado un directorio que se reciba por línea de
parámetros, muestre un
#   listado de forma recursiva con la ruta completa de cada uno de los
ficheros y su hash
#   SHA256. No se permite ejecutar ningún comando externo, la salida será
similar a esta:
#
#   Víctor Manuel Andreu Felipe 2025



import os
import hashlib
import sys

# calcula el hash SHA256 de un archivo
def calculate_sha256(file_path):
    sha256 = hashlib.sha256()
    try:
        with open(file_path, 'rb') as f:
            for chunk in iter(lambda: f.read(4096), b""):
                sha256.update(chunk)
        return sha256.hexdigest()
    except PermissionError:
        return "PERMISSION_DENIED"


# recorre los archivos de un directorio de forma recursiva
def list_files_with_hashes(directory):
    print(f"Procesando directorio {directory}:")
    print("=" * 30)
    for root, _, files in os.walk(directory):
        for file in files:
            file_path = os.path.join(root, file)

            # comprobación de que existe
            if not os.path.exists(file_path):
                print(f"SKIPPED (File Not Found): {file_path}")
                continue

            # calculo del sha256
            file_hash = calculate_sha256(file_path)
            print(f"{file_hash} {file_path}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Uso: python ejercicio2.py <directorio>")
        sys.exit(1)

    directory = sys.argv[1]
    if not os.path.isdir(directory):
```

```
        print("Error: El directorio especificado no existe o no es
válido.")
        sys.exit(1)

    list_files_with_hashes(directory)
```

```python
# Ejercicio 3: creación y modificación de ficheros
#   Inotify es una API del kernel de Linux que permite a las aplicaciones monitorizar en
#   tiempo real los cambios que ocurren en el sistema de ficheros. Esto incluye eventos como la
#   creación, modificación, eliminación o movimiento de archivos y directorios.
#   Entre sus características principales destacan:
#       • Monitorización en tiempo real: permite recibir notificaciones inmediatas cuando se
#       produce algún cambio en el sistema de archivos, evitando la necesidad de realizar
#       consultas periódicas (polling).
#       • Eficiencia: al ser una API basada en eventos, inotify reduce el consumo de recursos ya
#       que el sistema notifica solo cuando se detecta un cambio, en lugar de tener que revisar
#       constantemente el estado del sistema de ficheros.
#       • Facilidad de integración: varios lenguajes de programación, incluyendo Python, tienen
#       librerías que facilitan el uso de inotify. En el caso de Python se pueden utilizar módulos
#       como pyinotify o inotify_simple para integrar esta funcionalidad en cualquier
#       script.
#   Escribe un script que muestre un mensaje en pantalla cada vez que se cree o se modifique
#   un fichero en cualquier subdirectorio bajo /etc.
#
#   Víctor Manuel Andreu Felipe


import inotify.adapters


# monitorización del directorio etc
def monitor_directory(path="/etc"):
    notifier = inotify.adapters.InotifyTree(path)
    print(f"Monitoreando cambios en {path}...")

    try:
        for event in notifier.event_gen(yield_nones=False):
            (_, type_names, path, filename) = event

            # avisa cuando se crea o modifica un fichero
            if "IN_CREATE" in type_names:
                print(f"[CREADO] {path}/{filename}")
            elif "IN_MODIFY" in type_names:
                print(f"[MODIFICADO] {path}/{filename}")
    except KeyboardInterrupt:
        print("Monitorización finalizada.")
```

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

```python
#   Ejercicio 4: API
#   Escribe un script que muestre cada hora los siguientes datos sobre la
ciudad en la que
#   vives:
#       • Temperatura actual.
#       • Humedad relativa.
#       • Velocidad del tiempo.
#   Puedes usar para ello cualquier API de acceso gratuito que ofrezcan el
servicio.
#
#   Víctor Manuel Andreu Felipe


import time
import requests

# obtiene la información del tiempo dada una api key y una ubicación
def get_weather():
    api_key = "zsEzXz44X4qycv1"  # Reemplaza con tu clave de API
    location_id = "3489"  # Cambia por el ID de tu localidad
    url = f"https://api.tutiempo.net/json/?lan=es&apid={api_key}&lid=
{location_id}"

    # hace un get y almacena la respuesta en un json
    try:
        response = requests.get(url)
        response.raise_for_status()
        data = response.json()

        # extrae la hora actual, temperatura, humedad y viento de la misma
        current_hour_key = list(data["hour_hour"].keys())[0]
        current_weather = data["hour_hour"][current_hour_key]

        temperature = current_weather["temperature"]
        humidity = current_weather["humidity"]
        wind_speed = current_weather["wind"]

        print(f"Temperatura: {temperature}°C")
        print(f"Humedad: {humidity}%")
        print(f"Velocidad del viento: {wind_speed} km/h")
    # control de excepciones
    except requests.exceptions.RequestException as e:
        print(f"Error al obtener el clima: {e}")
    except KeyError:
        print("Error: No se encontraron datos de clima en la respuesta de
la API.")

if __name__ == "__main__":
    while True:
        print("Obteniendo datos meteorológicos...")
        get_weather()
        print("Esperando una hora para la próxima actualización...")
```

```
        time.sleep(3600)  # espera 1 hora (3600 segundos) antes de la
siguiente ejecución
```

```python
# Ejercicio 5: procesamiento de ficheros de logs
# Escribe un script que reciba como parámetro un fichero de logs de Nginx,
como por
# ejemplo /var/log/nginx/access.log, y muestre una estadística mostrando el
número de
# visitas que el servidor recibe por IP, ordenado de mayor a menor número
de visitas.
#
#   Víctor Manuel Andreu Felipe 2025

import sys
import collections

# procesa un archivo de logs de Nginx y muestra la lista de ip ordenadas
por visitas
def process_nginx_log(log_file):
    ip_counts = collections.Counter()
    try:
        with open(log_file, 'r') as file:
            for line in file:
                parts = line.split()
                if parts:
                    ip = parts[0]  # la ip está en el primer campo del log
                    ip_counts[ip] += 1
    # comprobación de excepciones
    except FileNotFoundError:
        print(f"Error: El archivo {log_file} no existe.")
        sys.exit(1)
    except PermissionError:
        print(f"Error: No tienes permisos para leer {log_file}.")
        sys.exit(1)

    # ordenación de las ips
    sorted_ips = sorted(ip_counts.items(), key=lambda x: x[1],
reverse=True)

    print("IP Address | Visit Count")
    print("-------------------------------")
    for ip, count in sorted_ips:
        print(f"{ip} | {count}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Uso: python nginx_log_stats.py <ruta_del_log>")
        sys.exit(1)

    log_file = sys.argv[1]
    process_nginx_log(log_file)
```

```python
# Ejercicio 6: hosts iniciados
# Escribe un script que reciba una red en notación CIDR y la escanee
buscando máquinas
# que estén encendidas. Para ello NO se podrá llamar a ningún comando
externo como nmap o
# ping, pero sí utilizar cualquier librería que nos proporcione cualquier
funcionalidad útil.
# Para enumerar las máquinas de la red será obligatorio utilizar la
librería ipaddress.
#
#   Víctor Manuel Andreu

# por alguna razón este script falla en la red virtual de libvirt

import ipaddress
import sys
from scapy.all import ICMP, IP, sr, conf

# escanea la red especificada usando ICMP
def scan_network_icmp_fast(network_cidr):

    network = ipaddress.ip_network(network_cidr, strict=False)
    print(f"Escaneando la red {network_cidr} con ICMP...")

    ip_list = [str(ip) for ip in network.hosts()]  # Lista de IPs a
escanear

    # enviamos paquetes ICMP a todas las ips al mismo tiempo
    packets = [IP(dst=ip) / ICMP() for ip in ip_list]
    answered, _ = sr(packets, timeout=1, verbose=False)  # envío masivo

    active_hosts = [rcv.src for snd, rcv in answered]


    # for ip in active_hosts:
    #     print(f"Host activo -> IP: {ip}")

    return active_hosts

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Uso: sudo python3 scan_icmp_fast.py <red/CIDR>")
        sys.exit(1)

    network_cidr = sys.argv[1]
    active_hosts = scan_network_icmp_fast(network_cidr)

    # resultado
    print("\nResumen de Hosts Activos:")
    for host in active_hosts:
        print(host)
```

```python
# Ejercicio 7: SSH
# Escribe un script que reciba una lista de servidores Linux y para cada uno de ellos se
# conecte por SSH y ejecutando comandos del sistema muestre la siguiente información:
#    • Dirección IP.
#    • Nombre de máquina.
#    • Machine-id (se puede obtener tanto del fichero /etc/machine-id como del comando
#      hostnamectl).
#    • Distro y versión.
#    • Procesador.
#    • Tiempo que la máquina lleva encendida.
#    • Carga en el último minuto, últimos cinco minutos y últimos quince.
#    • Memoria total y memoria libre.
#    • Versión de SSH instalada.
#
# Victor Manuel Andreu Felipe


import paramiko
import sys

# conectamos a una lista de servidores por SSH usando autenticación con clave

def get_server_info(host):
    try:
        client = paramiko.SSHClient()
        client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        client.load_system_host_keys()
        client.connect(hostname=host, username='root', timeout=5)

        # definimos la lista de comandos a pasar para la info del server
        commands = {
            "Dirección IP": "hostname -I | awk '{print $1}'",
            "Nombre de máquina": "hostname",
            "Machine-id": "cat /etc/machine-id",
            "Distro y versión": "cat /etc/os-release | grep -E 'PRETTY_NAME' | cut -d '=' -f2 | tr -d '\"'",
            "Procesador": "lscpu | grep 'Nombre del modelo' | awk -F: '{print $2}' | sed 's/^ *//'",
            "Tiempo encendido": "uptime -p",
            "Carga": "uptime | awk -F'load average:' '{print $2}'",
            "Memoria Total": "free -h | grep 'Mem:' | awk '{print $2}'",
            "Memoria Libre": "free -h | grep 'Mem:' | awk '{print $7}'",
            "Versión de SSH": "ssh -V 2>&1 | awk '{print $1, $2}'"
        }

        print(f"\n Conectado a {host}...")
        for key, cmd in commands.items():
            stdin, stdout, stderr = client.exec_command(cmd)
```

```
            output = stdout.read().decode().strip()
            print(f"{key}: {output}")

        client.close()
    except Exception as e:
        print(f"Error al conectar con {host}: {e}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Uso: python ssh_info_gather.py <lista_servidores.txt>")
        sys.exit(1)

    file_path = sys.argv[1]

    with open(file_path, "r") as file:
        servers = [line.strip() for line in file.readlines() if
line.strip()]

    for server in servers:
        get_server_info(server)
```

```python
import paramiko
import sys
import sqlite3
import datetime

# crea la base de datos y la tabla si no existen

def setup_database():
    conn = sqlite3.connect("server_info.db")
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS server_info (
            ip TEXT PRIMARY KEY,
            hostname TEXT,
            machine_id TEXT,
            distro TEXT,
            processor TEXT,
            uptime TEXT,
            load_avg TEXT,
            mem_total TEXT,
            mem_free TEXT,
            ssh_version TEXT,
            last_update TEXT
        )
    ''')
    conn.commit()
    conn.close()


# guarda o actualiza la información del servidor en la base de datos
def save_server_info(data):
    conn = sqlite3.connect("server_info.db")
    cursor = conn.cursor()
    cursor.execute('''
        INSERT INTO server_info (ip, hostname, machine_id, distro,
processor, uptime, load_avg, mem_total, mem_free, ssh_version, last_update)
        VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
        ON CONFLICT(ip) DO UPDATE SET
            hostname=excluded.hostname,
            machine_id=excluded.machine_id,
            distro=excluded.distro,
            processor=excluded.processor,
            uptime=excluded.uptime,
            load_avg=excluded.load_avg,
            mem_total=excluded.mem_total,
            mem_free=excluded.mem_free,
            ssh_version=excluded.ssh_version,
            last_update=excluded.last_update
    ''', data[:11])  # raro, me decía que estaba mandado 12 valores
    conn.commit()
    conn.close()
```

```
def get_server_info(host):
    # se conecta a un servidor por SSH usando autenticación con clave y
obtiene información del sistema
    try:
        client = paramiko.SSHClient()
        client.set_missing_host_key_policy(paramiko.AutoAddPolicy())
        client.load_system_host_keys()
        client.connect(hostname=host, username='root', timeout=5)

        commands = {
            "Dirección IP": "hostname -I | awk '{print $1}'",
            "Nombre de máquina": "hostname",
            "Machine-id": "cat /etc/machine-id",
            "Distro y versión": "cat /etc/os-release | grep -E
'PRETTY_NAME' | cut -d '=' -f2 | tr -d '\"'",
            "Procesador": "lscpu | grep 'Nombre del modelo' | awk -F:
'{print $2}' | sed 's/^ *//'",
            "Tiempo encendido": "uptime -p",
            "Carga": "uptime | awk -F'load average:' '{print $2}'",
            "Memoria Total": "free -h | grep 'Mem:' | awk '{print $2}'",
            "Memoria Libre": "free -h | grep 'Mem:' | awk '{print $7}'",
            "Versión de SSH": "ssh -V 2>&1 | awk '{print $1, $2}'"
        }

        print(f"\n Conectado a {host}...")
        data = [host]
        for key, cmd in commands.items():
            stdin, stdout, stderr = client.exec_command(cmd)
            output = stdout.read().decode().strip()
            print(f"{key}: {output}")
            data.append(output)

        data.append(datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"))
        save_server_info(data)
        client.close()
    except Exception as e:
        print(f"Error al conectar con {host}: {e}")

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print("Uso: python ssh_info_gather.py <lista_servidores.txt>")
        sys.exit(1)

    file_path = sys.argv[1]

    setup_database()

    with open(file_path, "r") as file:
        servers = [line.strip() for line in file.readlines() if
line.strip()]

    for server in servers:
        get_server_info(server)
```

```python
import sqlite3

# muestra toda la información almacenada en la base de datos
def show_server_info():
    conn = sqlite3.connect("server_info.db")
    cursor = conn.cursor()
    cursor.execute("SELECT * FROM server_info")
    rows = cursor.fetchall()

    if not rows:
        print("No hay datos almacenados en la base de datos.")
        return

    print("\nInformación almacenada en la base de datos:")
    for row in rows:
        print("----------------------------------")
        print(f"IP: {row[0]}")
        print(f"Hostname: {row[1]}")
        print(f"Machine ID: {row[2]}")
        print(f"Distro: {row[3]}")
        print(f"Procesador: {row[4]}")
        print(f"Tiempo encendido: {row[5]}")
        print(f"Carga: {row[6]}")
        print(f"Memoria Total: {row[7]}")
        print(f"Memoria Libre: {row[8]}")
        print(f"Versión de SSH: {row[9]}")
        print(f"Última actualización: {row[10]}")

    conn.close()

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