

# Documentación: Proceso de Envío de Correos y Ejecución de Código Batch

## Introducción

Este documento explica el desarrollo y funcionamiento del sistema de envío automático de correos mediante Python junto a un script Bash para la programación semanal de este proceso. El objetivo es enviar saludos automáticos a una lista de destinatarios, utilizando un servidor SMTP para manejar el envío y el Programador de Tareas (cron) para automatizar la ejecución del proceso cada lunes.

## 1. Proceso de Envío de Correos con Python

El script `send_email.py` está escrito en Python y sigue el estándar PEP8, asegurando buenas prácticas de legibilidad y consistencia. Este script se encarga de leer una lista de correos almacenada en un archivo CSV y enviar un correo electrónico de saludo a cada uno de los destinatarios.

### 1.1 Dependencias

El script utiliza las siguientes bibliotecas:

- `csv` : Para leer la lista de destinatarios desde un archivo CSV.
- `smtplib` y `email.mime` : Para manejar el envío de correos a través de SMTP.
- `dotenv` : Para cargar credenciales almacenadas en un archivo `.env`.
- `os` : Para acceder a las variables de entorno.

### 1.2 Configuración de Credenciales

Las credenciales de acceso al servidor SMTP se almacenan en un archivo `.env` para asegurar que no se exponen directamente en el código. El archivo `.env` contiene las siguientes variables:

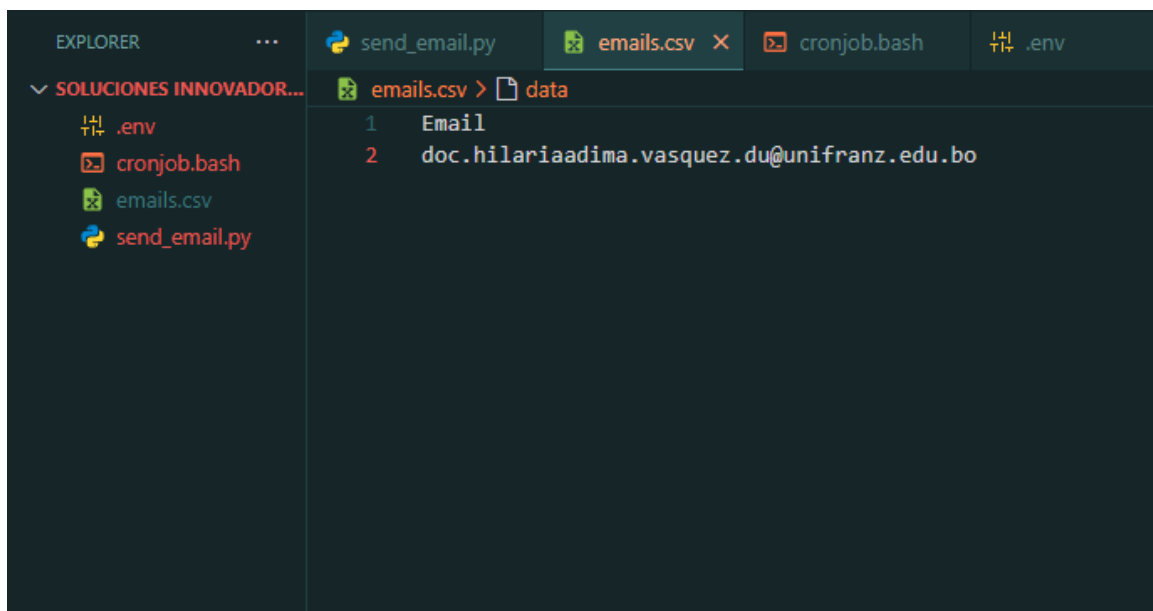
```
EMAIL_USERNAME=tu_correo@gmail.com
```

```
EMAIL_PASSWORD=tu_contraseña_de_aplicacion
```

Estas credenciales se cargan en el script mediante la biblioteca `dotenv` para luego ser usadas en la autenticación del servidor SMTP de Gmail.

## 1.3 Funcionamiento del Script

1. **Lectura del Archivo CSV:** La función `leer_lista_correos(archivo_csv)` lee los correos de un archivo `emails.csv`, ignorando la cabecera.



2. **Envío de Correos:** La función `enviar_saludo(destinatario)` configura y envía un correo electrónico a cada destinatario utilizando `smtplib`. El correo tiene un asunto definido y un cuerpo que contiene un mensaje de saludo.
3. **Manejo de Errores:** Se implementa un bloque `try-except` para manejar posibles errores de autenticación y otras excepciones que puedan ocurrir durante el envío.

```
import csv
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
import os
from dotenv import load_dotenv
```

```

# Cargar variables de entorno desde el archivo .env
load_dotenv()

# Configuración del servidor SMTP
SMTP_SERVER = 'smtp.gmail.com'
SMTP_PORT = 587
USERNAME = os.getenv('EMAIL_USERNAME')
PASSWORD = os.getenv('EMAIL_PASSWORD') # Usar variable
de entorno para la contraseña

# Crear una función para enviar el saludo
def enviar_saludo(destinatario):
    # Configurar el correo electrónico
    mensaje = MIMEMultipart()
    mensaje['From'] = USERNAME
    mensaje['To'] = destinatario
    mensaje['Subject'] = (
        'Saludos desde Soluciones innovadoras en el lenguaje de programación local'
    )

    # Cuerpo del correo
    cuerpo = """
    Hola,

    Espero que estés teniendo un excelente día.

    Saludos cordiales,
    """
    mensaje.attach(MIMEText(cuerpo, 'plain'))

    # Enviar el correo
    try:
        servidor = smtplib.SMTP(SMTP_SERVER, SMTP_PORT)
        servidor.starttls()
        servidor.login(USERNAME, PASSWORD)
        servidor.sendmail(USERNAME, destinatario, mensaje.as_string())

```

```

servidor.quit()
print(f'Correo enviado a {destinatario}')
except smtplib.SMTPAuthenticationError:
    print(
        f'Error de autenticación al enviar correo a
{destinatario}: Verifique las credenciales.'
    )
except Exception as e:
    print(f'Error al enviar correo a {destinatario}:
{e}')

# Leer la lista de correos del archivo CSV
def leer_lista_correos(archivo_csv):
    with open(archivo_csv, mode='r') as file:
        reader = csv.reader(file)
        next(reader) # Saltar la cabecera
        for row in reader:
            if row:
                enviar_saludo(row[0])

# Ruta del archivo CSV con la lista de correos
archivo_csv = 'emails.csv'

# Ejecutar la función para leer la lista y enviar los sa
ludos
leer_lista_correos(archivo_csv)

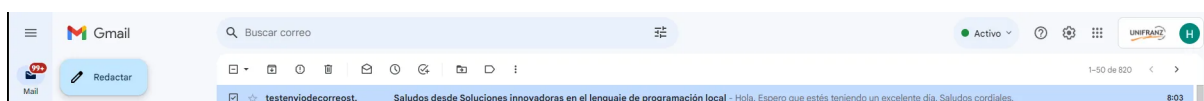
```

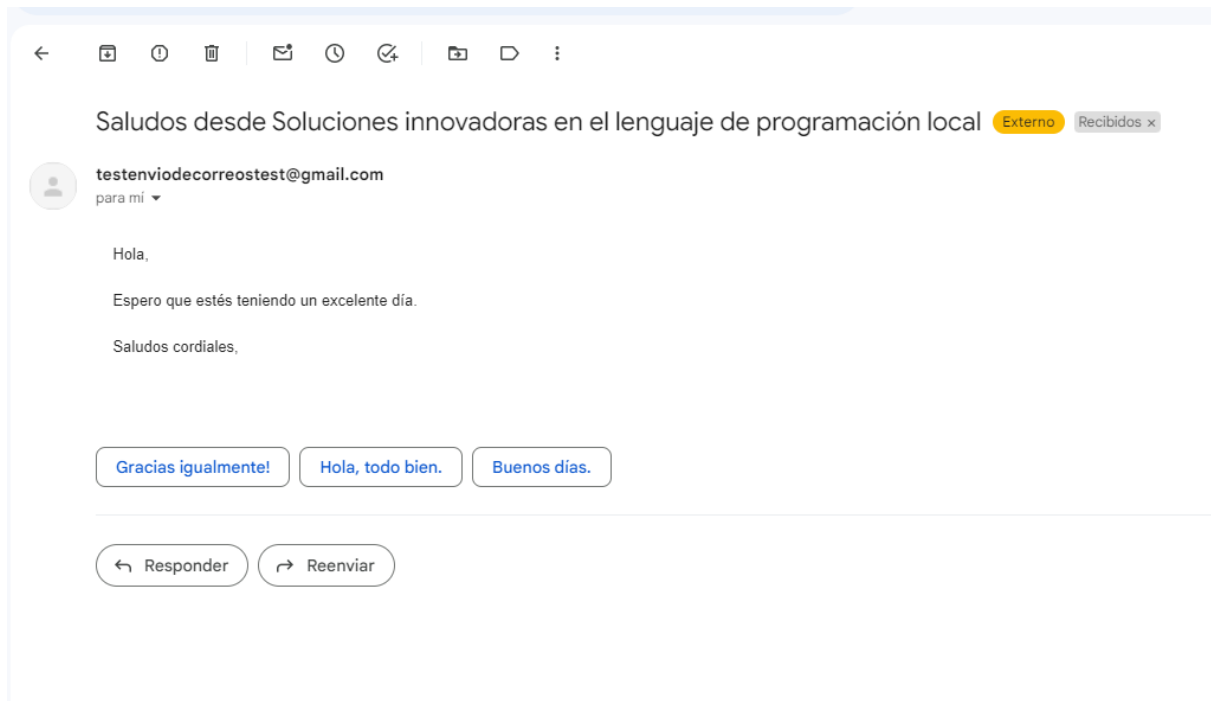
## 1.4 Ejecución

Para ejecutar el script manualmente, se usa el siguiente comando:

```
python send_email.py
```

El script se ejecuta y envía un correo a cada destinatario listado en `emails.csv`.





## 2. Proceso del Código Batch con Bash

El script en Bash, denominado `send_greetings.sh`, se encarga de ejecutar el script de Python en un horario especificado para que se repita cada semana de forma automática.

### 2.1 Estructura del Script Bash

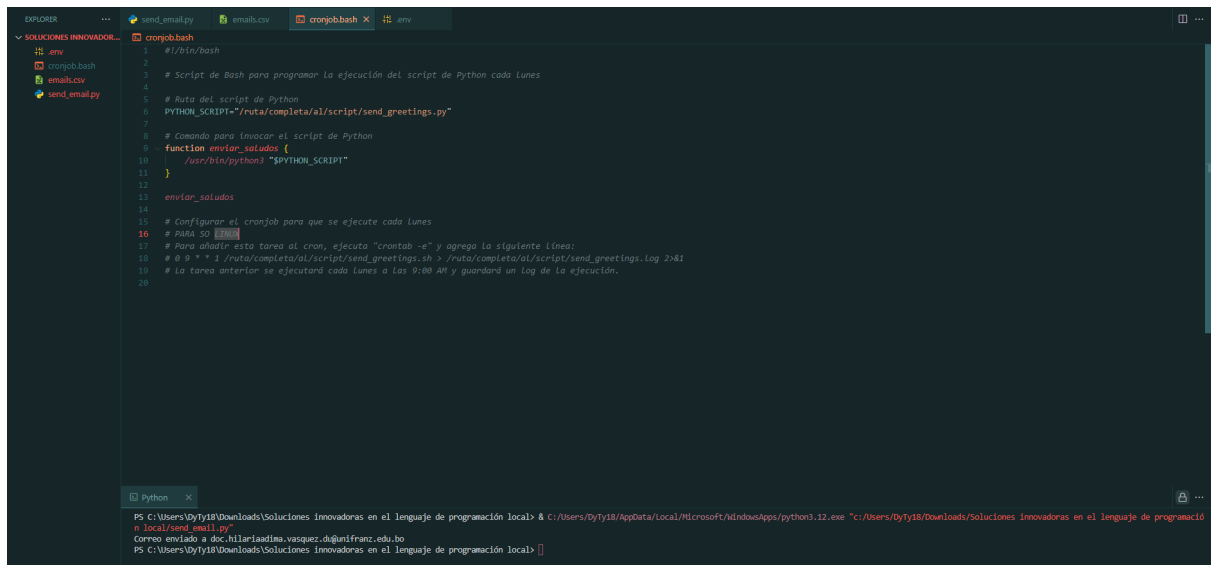
```
#!/bin/bash

# Script de Bash para programar la ejecución del script de
# Python cada lunes

# Ruta del script de Python
PYTHON_SCRIPT="/ruta/completa/al/script/send_greetings.py"

# Comando para invocar el script de Python
function enviar_saludos {
    /usr/bin/python3 "$PYTHON_SCRIPT"
}
```

enviar\_saludos



The screenshot shows a code editor with a file named `enviar_saludos` open. The script is a Bash script that sets up a cronjob to run a Python script every Monday at 9:00 AM. The script includes comments in Spanish explaining each step. Below the code editor, a terminal window shows the execution of the script, which successfully creates the cronjob and logs the execution.

```
1 #!/bin/bash
2
3 # Script de Bash para programar la ejecución del script de Python cada Lunes
4
5 # Ruta del script de Python
6 PYTHON_SCRIPT="/ruta/completa/al/script/send_greetings.py"
7
8 # Comando para invocar el script de Python
9 function enviar_saludos {
10     /usr/bin/python3 "$PYTHON_SCRIPT"
11 }
12
13 enviar_saludos
14
15 # Configuran el cronjob para que se ejecute cada Lunes
16 # Para SO Linux
17 # Para añadir esta tarea al cron, ejecuta "crontab -e" y agrega la siguiente línea:
18 # 0 9 * * 1 /ruta/completa/al/script/send_greetings.sh > /ruta/completa/al/script/send_greetings.log 2>&1
19 # La tarea anterior se ejecutará cada Lunes a las 9:00 AM y guardará un log de la ejecución.
20
```

Terminal output:

```
PS C:\Users\Dy\18\Downloads\Soluciones Innovadoras en el lenguaje de programación local> && C:\Users\Dy\18\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/Dy/18/Downloads/Soluciones Innovadoras en el lenguaje de programación local/send_email.py"
Correo enviado a docilariadefina.vazquez.d@unifrma.edu.bo
PS C:\Users\Dy\18\Downloads\Soluciones Innovadoras en el lenguaje de programación local>
```

## 2.2 Programación del Cronjob en Linux

Para asegurar que el script de Python se ejecute de forma automatizada cada lunes a las 9:00 AM, el script de Bash se programa mediante un cronjob. Los pasos son los siguientes:

### 1. Abrir el archivo crontab:

Ejecutar

`crontab -e` en la terminal para editar las tareas programadas del usuario.

### 2. Agregar el Cronjob:

Agregar la siguiente línea al archivo crontab:

Esto asegura que el script se ejecute cada lunes a las 9:00 AM y almacene un registro (

`send_greetings.log`) de la salida o errores generados durante la ejecución.

```
0 9 * * 1 /ruta/completa/al/script/send_greetings.sh > /
ruta/completa/al/script/send_greetings.log 2>&1
```

## 2.3 Programación de la Tarea en Windows

En Windows, el script se puede programar utilizando el Programador de Tareas de Windows. A continuación, se describe cómo hacerlo:

### 1. Abrir el Programador de Tareas:

- Presiona **Win + S** y escribe "Programador de tareas" para abrirlo.

## 2. Crear una Nueva Tarea:

- Haz clic en "Crear Tarea" en la parte derecha.
- Dale un nombre (por ejemplo, "Enviar correos cada lunes").

## 3. Configurar la Programación:

- Ve a la pestaña "Triggers" (Disparadores).
- Haz clic en "Nuevo...".
- Configura la tarea para que se ejecute "Semanalmente" y elige el lunes.

## 4. Configurar la Acción:

- Ve a la pestaña "Actions" (Acciones).
- Haz clic en "Nuevo...".
- En "Programa/script", coloca la ruta del ejecutable de Python. En tu caso:

```
C:\\Users\\TuUsuario\\AppData\\Local\\Microsoft\\WindowsApps\\python3.12.exe
```

- En "Agregar argumentos (opcional)", coloca la ruta completa de tu script Python, por ejemplo:

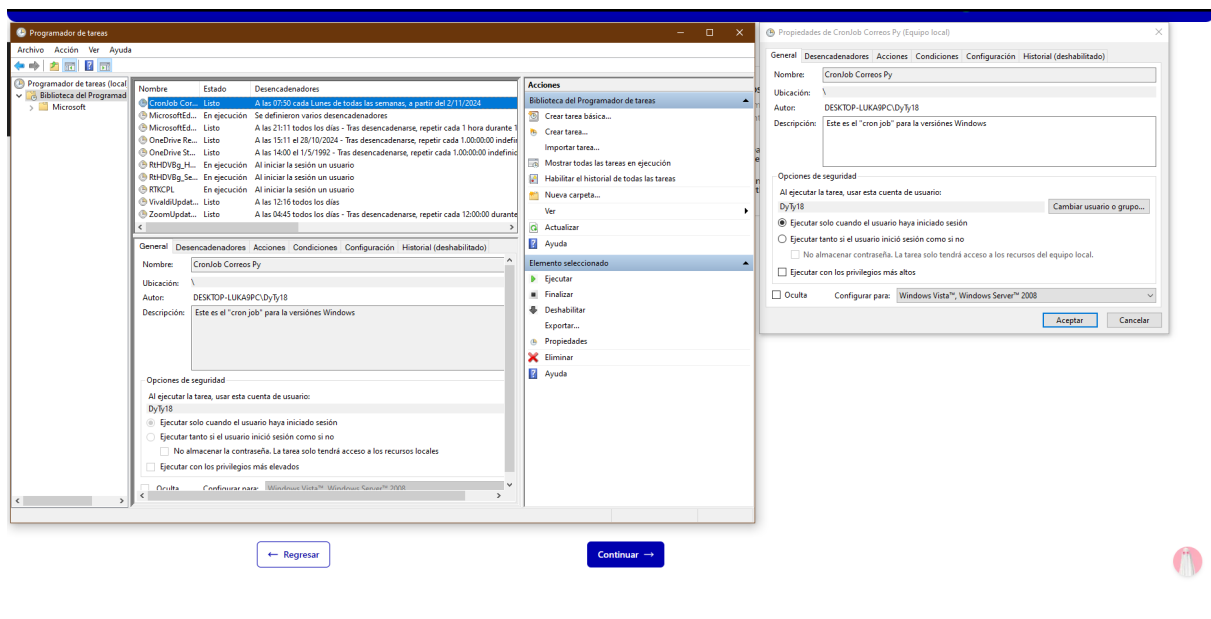
```
"C:\\ruta\\completa\\al\\script\\send_greetings.py"
```

## 5. Configurar las Opciones:

- En la pestaña "Condiciones" y "Configuración", asegúrate de que la tarea se pueda ejecutar incluso si el ordenador no está conectado a la corriente (opcional, dependiendo de tu preferencia).

## 6. Guardar la Tarea:

- Haz clic en "OK" para guardar la tarea.



### 3. Resumen del Flujo de Trabajo

#### 1. Preparación de Archivos:

- Se crea el archivo `.env` con las credenciales de Gmail.
- Se prepara el archivo `emails.csv` con la lista de destinatarios.

#### 2. Ejecución Manual del Script Python: Ejecutar `send_email.py` para probar el envío de correos.

#### 3. Automatización del Proceso:

- Se crea el script Bash `send_greetings.sh`.
- Se programa un cronjob para ejecutar el script de Bash cada lunes a las 9:00 AM en Linux o una tarea programada en Windows.

### 4. Consideraciones Finales

- **Seguridad:** Es importante utilizar una "contraseña de aplicaciones" en lugar de la contraseña principal para Gmail, de modo que se reduzca el riesgo de seguridad.
- **Pruebas:** Antes de programar la tarea automática, se recomienda ejecutar el script manualmente para asegurarse de que las credenciales sean correctas y el envío de correos funcione como se espera.
- **Manejo de Errores:** Tanto el script de Python como el script Bash contienen mecanismos básicos para registrar errores y asegurarse de que cualquier



problema sea documentado para futuras revisiones.

## 5. Conclusión

La combinación de Python y Bash permite una solución flexible y automatizada para el envío de correos electrónicos. Python se utiliza para gestionar la lógica del envío de correos, mientras que Bash y el cronjob permiten automatizar el proceso de forma periódica en sistemas Linux. En Windows, el Programador de Tareas se utiliza para automatizar el mismo proceso. Este enfoque es eficaz y fácilmente adaptable a otros escenarios donde se requiera una programación periódica.