

Universidad Nacional de Guillermo Brown  
Estructuras de Datos  
Documentación del Proyecto Final

Integrantes:  
Villalva Franco  
Casas Oriana

Comisión 2 – Dr. Ambrossio Diego Agustín

# 1. Introducción y Objetivos.

## **1-Objetivos**

El presente proyecto tiene como objetivo el diseño e implementación de un Cliente de Correo Electrónico basado en el paradigma de Orientación a Objetos, integrando estructuras de datos no lineales para la gestión eficiente de la información.

## **2-Pautas**

Las entregas se harán cada, aproximadamente, 3 semanas, de una manera secuencial donde por cada entrega se van a ir añadiendo nuevas condiciones/funcionalidades al proyecto para que vaya evolucionando.

# 2. Arquitectura y diseño del Sistema (UML)

La arquitectura del proyecto se definió por medio de un UML (Lenguaje unificado modelado), el cual se basó en un diagrama de clases que nos ejemplifica de una manera más fácil como son las relaciones y los métodos que va a tener cada parte del proyecto.

### **Descripción de Clases:**

**ServidorCorreo:** La clase central del proyecto. Se encarga de la gestión de usuarios, la autenticación y el enrutamiento de mensajes. Contiene un diccionario de usuarios.

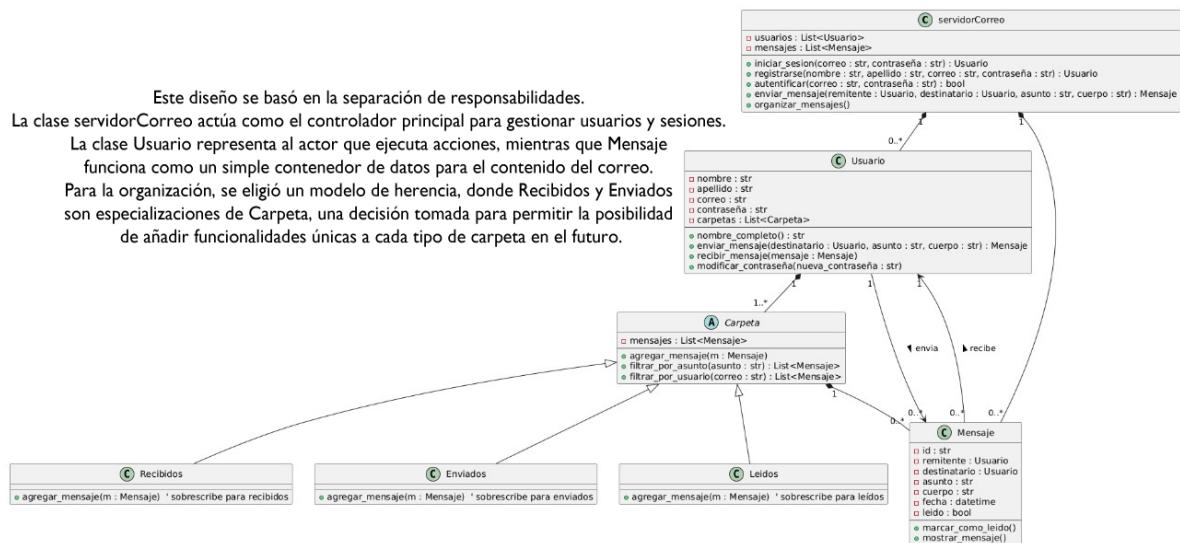
**Métodos principales:** registrarse(), iniciar\_sesion(), enviar\_mensaje().

**Usuario:** Representa a cada usuario del sistema. Contiene información personal y gestiona sus propias carpetas de correo (bandeja\_entrada, bandeja\_salida).

**Mensaje:** Un objeto simple que almacena toda la información de un correo electrónico, como remitente, destinatario, asunto, y cuerpo del mensaje.

**Carpeta:** Una clase que actúa como un contenedor de objetos Mensaje. Proporciona métodos para agregar, listar y filtrar mensajes. Las carpetas Recibidos, Enviados y Leidos son especializaciones de esta clase.

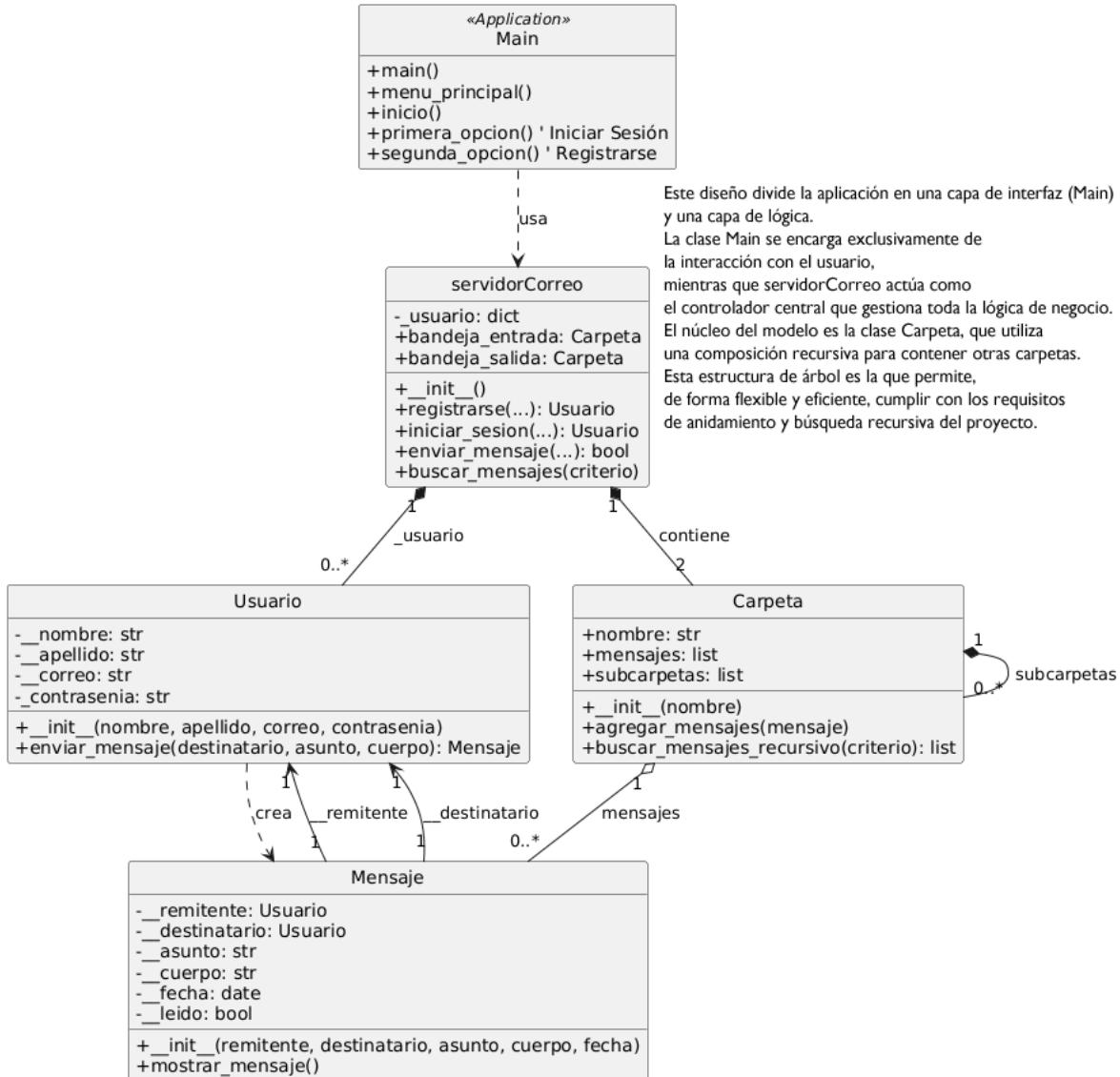
## Primer Diseño del UML (Primera Entrega):



## Segunda Entrega

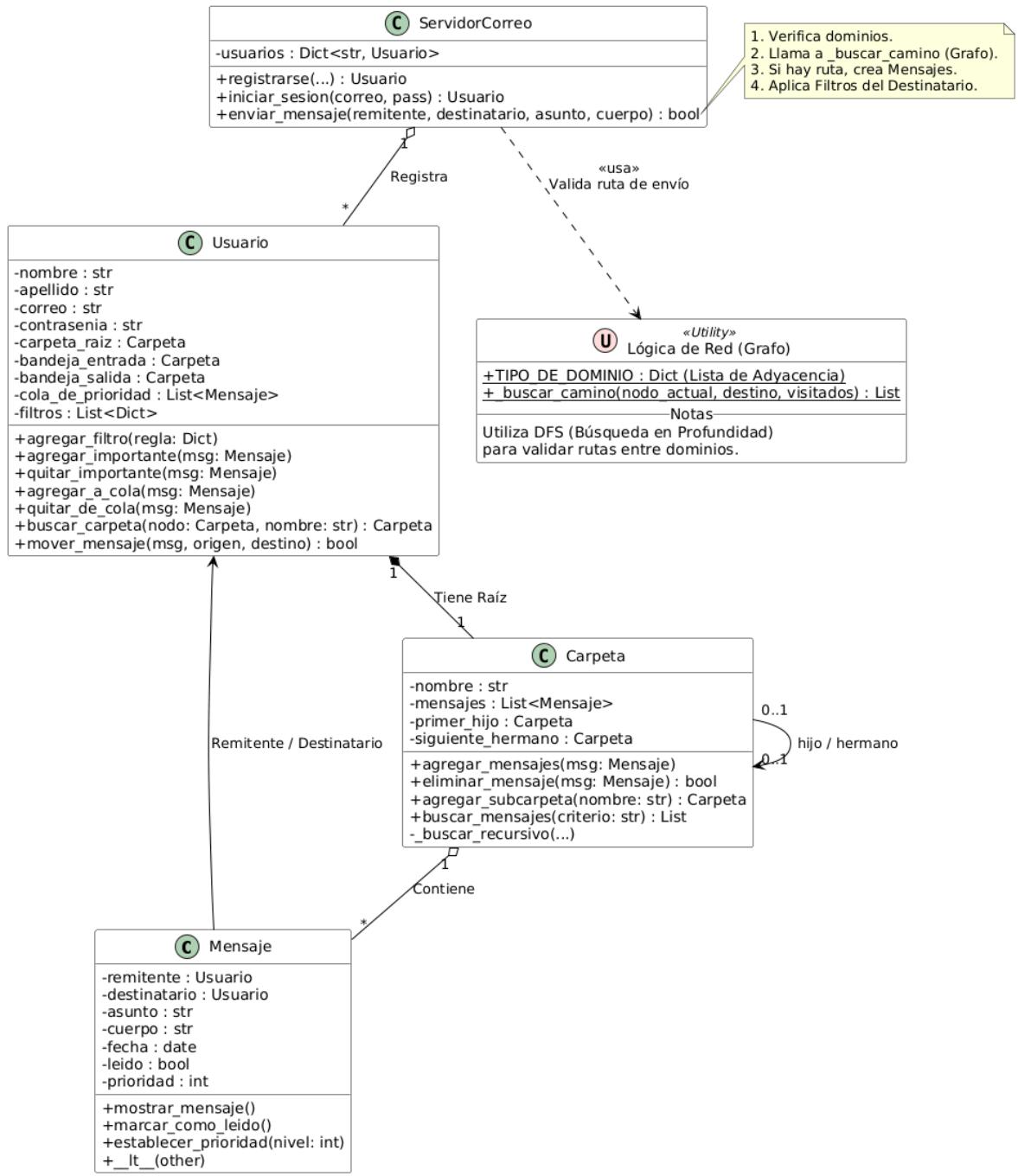
Se diseñó un nuevo diagrama UML, agregando main (que maneja la aplicación principal) y modificando Carpeta para que el buscador que funciona por asunto o usuario tenga una sola función

de filtrado que funcione de forma recursiva.



## Tercera Entrega

Para la entrega final, se consolidó la arquitectura en un diagrama definitivo que integra la Lógica de Red del proyecto y que se ajustara a lo que buscábamos.



### 3. Estructuras de Datos Implementadas

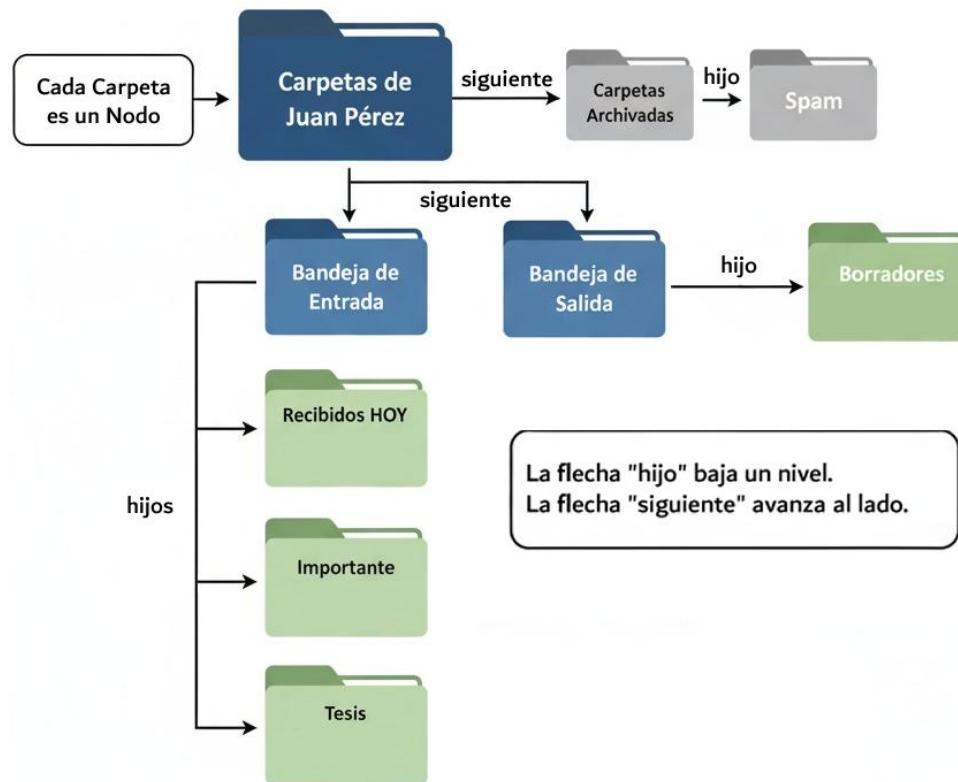
Para resolver los desafíos de gestión de información, no nos limitamos a listas simples. Seleccionamos estructuras específicas para cada problema:

- **Gestión de Carpetas (Árboles Generales):** Dado que un usuario puede tener carpetas dentro de carpetas con profundidad infinita, utilizamos una estructura de Árbol General.
  - *Decisión Técnica:* En lugar de una lista de hijos, implementamos la lógica "Left Child - Right Sibling" (primer\_hijo, siguiente\_hermano). Esto optimiza la memoria, ya que cada nodo solo necesita dos punteros, independientemente de cuántas subcarpetas tenga.
- **Sistema de Prioridad (Min-Heap):** Para la funcionalidad de "Mensajes Importantes", necesitábamos que el sistema siempre devolviera el mensaje más urgente, no el último que llegó.
  - *Decisión Técnica:* Implementamos una Cola de Prioridad utilizando un Heap Binario (vía heapq). Esto garantiza que la extracción del mensaje más importante sea siempre eficiente ( $O(\log n)$ ), reordenando la cola automáticamente.
- **Simulación de Red (Grafos y DFS):** Para simular el envío de correos entre dominios (ej: de yahoo a google), modelamos la red como un Grafo donde los nodos son los dominios.
  - *Decisión Técnica:* Utilizamos un algoritmo de Búsqueda en Profundidad (DFS) recursivo para encontrar si existe un camino viable entre el remitente y el destinatario, simulando saltos de red reales.
- Para la implementación de árboles generales, se estableció una estructura donde, para crear las subcarpetas, se crearía una carpeta principal para cada usuario, de la cual surgirían tres subcarpetas "Bandeja de Entrada", "Bandeja de Salida" y "Mensajes Archivados". Dentro de cada una, los usuarios podrán crear otras subcarpetas con nombres personalizables.

- Figura 3: Diagrama esquemático de la estructura de Árbol General (Primer Hijo, Siguiente Hermano).

## ESTRUCTURA DE ÁRBOL DE CARPETAS CLIENTE DE CORREO

Implementación "primer hijo, siguiente hermano"



## 4. Análisis de Eficiencia Algorítmica (Big-O)

Para garantizar el rendimiento del sistema, se analizaron las complejidades temporales de las operaciones críticas:

- Gestión de Carpetas (Árbol General):

- **Operación:** Búsqueda de una carpeta específica.

- **Complejidad:**  $O(n)$ , donde n es el número total de carpetas. En el peor caso (árbol degenerado o búsqueda profunda), se deben recorrer todos los nodos. La estructura de "Primer Hijo, Siguiente Hermano" optimiza el uso de memoria frente a listas de punteros vacíos.

- Cola de Prioridad (Heap Binario):

- *Operación:* Inserción y Extracción del mensaje más importante.
- *Complejidad:* **O(logn)**. Al utilizar un Heap, evitamos el costo de reordenar una lista completa ( $O(n\log n)$ ) cada vez que llega un mensaje urgente. Esto hace que el sistema sea escalable incluso con miles de correos en cola.
- **Enrutamiento de Red (DFS en Grafo):**
- *Operación:* Búsqueda de ruta entre dominios.
- *Complejidad:* **O(V+E)**, donde V son los servidores (nodos) y E las conexiones. Dado que el grafo de dominios es disperso, DFS es eficiente para encontrar conectividad sin consumir memoria excesiva.

## 5. Estrategia de Testing y QA

A diferencia de una validación manual, se implementó una suite de pruebas unitarias y de integración utilizando la librería Pytest. Esto nos permite asegurar la estabilidad del sistema ante cambios.

### Escenarios Cubiertos:

- **Test de Estructura de Datos (test\_folder.py):** Valida que al agregar una subcarpeta, los punteros del árbol no se rompan y la jerarquía se mantenga intacta.
- **Test de Algoritmos Avanzados (test\_advanced.py):**
  - *Grafo:* Verifica que el algoritmo DFS encuentre rutas complejas y falle controladamente si el dominio no existe.
  - *Heap:* Inserta mensajes en orden aleatorio y verifica matemáticamente que al extraerlos, salgan en estricto orden de prioridad (Urgente -> Importante -> Normal).

### Evidencia de Ejecución:

```
PS C:\Users\samue\Desktop\Estudios-Ori\UNAB Ori\VSC\Cliente-de-Correo-Electr-nico-Email-Client\código> python -m pytest
=====
platform win32 -- Python 3.11.9, pluggy-1.6.0
rootdir: C:\Users\samue\Desktop\Estudios-Ori\UNAB Ori\VSC\Cliente-de-Correo-Electr-nico-Email-Client\código
plugins: anyio-4.9.0
collected 11 items

tests\test_advanced.py ....
tests\test_folder.py ....
tests\test_user.py ....

11 passed in 0.10s
PS C:\Users\samue\Desktop\Estudios-Ori\UNAB Ori\VSC\Cliente-de-Correo-Electr-nico-Email-Client\código>
```

*Resultado: 100% de los casos de prueba superados (Passed), garantizando la robustez de la lógica central.*

## 6. Material Adicional

Debido al peso del video, GitHub no permitió su subida directa al repositorio, así que fue solucionado creando una carpeta Drive que contenga los archivos de material extra más pesados. [\[Link a la carpeta con archivos adicionales\]](#)

## 7. Conclusiones

El desarrollo del "Cliente de Correo Electrónico" ha permitido integrar con éxito los conceptos teóricos de Estructuras de Datos en una aplicación funcional y robusta. A través de la implementación, se validó que la elección correcta de la estructura de datos es determinante para el rendimiento del software:

- **Escalabilidad:** La implementación de **Árboles Generales** demostró ser superior a las estructuras lineales tradicionales para manejar jerarquías complejas de carpetas, permitiendo una anidación ilimitada sin comprometer la organización lógica.
- **Eficiencia Crítica:** La incorporación de **Heaps Binarios** para la gestión de prioridades y algoritmos de grafos (**DFS**) para el enrutamiento transformó operaciones potencialmente costosas en procesos eficientes ( $O(\log n)$  y  $O(V+E)$  respectivamente), cumpliendo con los requisitos de optimización computacional.
- **Fiabilidad del Software:** La adopción de una metodología de **QA automatizado con Pytest** no solo garantizó la correctitud de los algoritmos entregados, sino que estableció una base sólida para el mantenimiento futuro del código, minimizando la deuda técnica.

En resumen, el sistema resultante no es solo un conjunto de clases interconectadas, sino una solución de software optimizada, validada y documentada, capaz de simular operaciones de correo electrónico con precisión algorítmica.

## 8. Anexo: Manual de Instalación - Requerimientos del sistema

Para la ejecución del entorno de desarrollo, se requiere la instalación de las siguientes dependencias:

- Python 3.13.7 o superior
- Visual Studio Code instalado
- Tener clonado el repositorio de Git Hub y crear una rama aparte

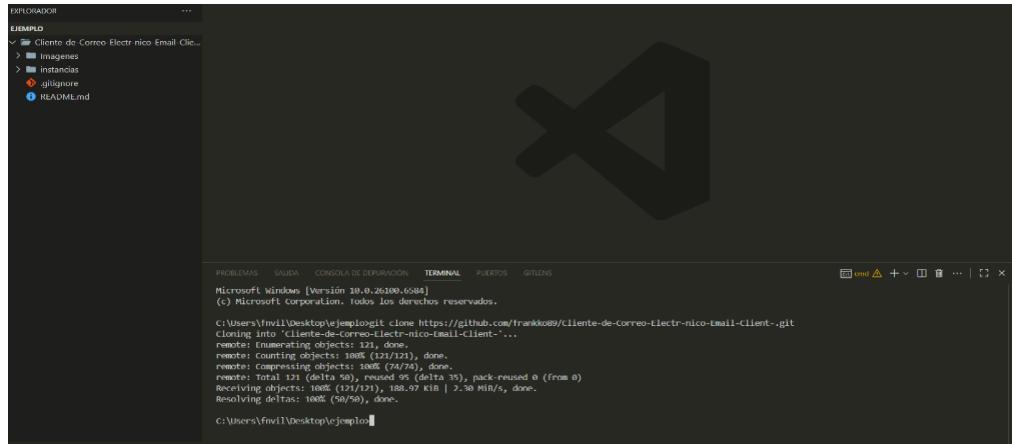
### Pasos para clonar el repositorio y crear una rama

#### 8.1. [Entrar al repositorio de Git Hub](#)

8.2. Ejecutar el siguiente comando en la terminal para clonar el repositorio:

```
C:\Users\fnvil\Desktop\ejemplo>git clone https://github.com/frankko89/Cliente-de-Correo-Electr-nico-Email-Client-.git
```

8.3. Luego de ejecutar esa linea va a tener clonado el repositorio y se vera de esta forma



Al ser colaboradores, se puede crear una rama para poder trabajar y hacer pruebas sin afectar a la main.

```
C:\Users\fnvil\Desktop\ejemplo\Cliente-de-Correo-Electr-nico-Email-Client->git checkout -b pruebas
Switched to a new branch 'pruebas'

C:\Users\fnvil\Desktop\ejemplo\Cliente-de-Correo-Electr-nico-Email-Client->git branch
  main
* pruebas
```

Se crearon ramas para que cada uno trabajara en una rama aparte y con el comando \* git Branch \* podemos ver todas las ramas creadas y en cual estamos ubicados para trabajar.