Ejercicio Práctico: Seguridad y Manejo Avanzado de Repositorios en GitHub

Este ejercicio te guiará a través de la configuración de **claves SSH y autenticación segura**, **control de acceso a repositorios**, y **técnicas avanzadas de gestión de ramas y uso de rebase**. Cada sección incluye los códigos necesarios y explicaciones para asegurar un flujo de trabajo seguro y eficiente en GitHub.

Parte 1: Gestión de Claves SSH y Autenticación Segura

En esta parte, configuraremos una clave SSH exclusiva para un repositorio en GitHub, limitando el acceso a nivel de carpeta.

Paso 1.1: Crear una Clave SSH para el Proyecto

En la terminal, genera una nueva clave SSH para el repositorio:
bash

```
ssh-keygen -t ed25519 -C "tu_email@example.com" -f
~/.ssh/github_repo_key
```

1. Esto crea dos archivos:
    - `github_repo_key`: Clave privada (nunca debe compartirse).
    - `github_repo_key.pub`: Clave pública (la utilizaremos en GitHub).

Paso 1.2: Agregar la Clave Pública a GitHub

1. Abre GitHub y navega a **Settings > SSH and GPG keys > New SSH key**.
2. Copia el contenido del archivo `~/.ssh/github_repo_key.pub`.
3. Pega el contenido copiado en el campo de clave SSH de GitHub.
4. Asigna un nombre descriptivo a la clave, por ejemplo, "Clave SSH para Ejercicio GitHub".

Paso 1.3: Configurar Git para Usar la Clave en un Repositorio Específico

Abre (o crea) el archivo de configuración SSH en `~/.ssh/config` y añade lo siguiente:
plaintext

```
Host github-ejercicio
  HostName github.com
  User git
  IdentityFile ~/.ssh/github_repo_key
  IdentitiesOnly yes
```

1. **Explicación**:
    - `Host github-ejercicio`: Define un alias para conectarse a GitHub con esta clave específica.
    - `IdentityFile ~/.ssh/github_repo_key`: Indica a SSH que use esta clave al conectarse con el alias `github-ejercicio`.

Paso 1.4: Clonar el Repositorio Usando la Nueva Configuración
Clona el repositorio empleando el alias configurado:
bash

```
git clone git@github-ejercicio:usuario/nombre-repositorio.git
```

Resultado: La clave SSH exclusiva se ha configurado, asegurando una autenticación aislada y segura para el repositorio.

Parte 2: Control de Acceso a Repositorios

Configurar permisos para colaboradores y aplicarás reglas de protección en la rama `main` para garantizar la seguridad y control de calidad.

Paso 2.1: Asignar Permisos de Acceso en GitHub

1. Ve a la página del repositorio en GitHub, accede a **Settings > Manage access** y haz clic en **Invite a collaborator**.
2. Ingresa el nombre del colaborador y selecciona el nivel de acceso adecuado, por ejemplo, "Write" (permite modificar el contenido sin permisos administrativos).

Paso 2.2: Configurar Reglas de Protección en la Rama `main`

1. En la página del repositorio, ve a **Settings > Branches > Branch protection rules**.
2. Haz clic en **Add rule** y selecciona la rama `main`.
3. Configura las siguientes reglas para proteger la rama:
    - **Requerir revisión de solicitudes de extracción**: Al menos un revisor debe aprobar antes de fusionar cambios en `main`.
    - **Requerir que las pruebas de CI/CD pasen**: Solo permite fusiones si las pruebas han pasado.
    - **Prohibir fusiones directas a `main`**: Limita los cambios a través de solicitudes de extracción.

Resultado: El control de acceso está configurado, y la rama `main` está protegida con reglas que mejoran la calidad y seguridad del código.

Parte 3: Técnicas Avanzadas de Gestión de Ramas y Uso de Rebase

En esta parte, crearás una rama de característica, aplicarás un rebase para mantener un historial limpio y resolverás posibles conflictos.

Paso 3.1: Crear y Organizar las Ramas del Proyecto

Cambia a la rama `main` y crea una rama `develop` para desarrollo continuo:
bash

```
git checkout main
git checkout -b develop
```

Desde `develop`, crea una rama de característica para implementar una nueva función:
bash

```
git checkout develop
git checkout -b feature/nueva-funcionalidad
```

Paso 3.2: Realizar Cambios y Commits en la Rama de Característica

1. Agrega y modifica archivos en `feature/nueva-funcionalidad` según los cambios requeridos.

Realiza algunos commits:
bash

```
git add .
git commit -m "Implementa parte 1 de la nueva funcionalidad"
git commit -m "Corrige errores en la nueva funcionalidad"
```

Paso 3.3: Rebase de `main` en la Rama de Característica

Asegúrate de que tu rama `main` esté actualizada:
bash

```
git checkout main
git pull origin main
```

Cambia a `feature/nueva-funcionalidad` y haz un rebase para integrar los cambios recientes de `main`:
bash

```
git checkout feature/nueva-funcionalidad
git rebase main
```

Si se detectan conflictos, Git pausará el rebase y solicitará resolverlos. Abre los archivos en conflicto, resuélvelos y luego ejecuta:
bash

```
git add archivo_con_conflicto
git rebase --continue
```

Paso 3.4: Rebase Interactivo para Limpiar el Historial de Commits
Usa un rebase interactivo para consolidar y mejorar la legibilidad de los commits antes de fusionar la rama en `develop`:
bash

```
git rebase -i HEAD~2  # Reemplaza 2 por el número de commits a revisar
```

En el editor que se abre, utiliza `squash` para combinar commits o `reword` para editar mensajes, según sea necesario.

Paso 3.5: Fusionar la Rama de Característica en `develop`
Cambia a la rama `develop` y realiza la fusión de `feature/nueva-funcionalidad`:
bash

```
git checkout develop
git merge feature/nueva-funcionalidad
```

Elimina la rama de característica ahora que se ha fusionado:
bash

```
git branch -d feature/nueva-funcionalidad
```

Resultado: La rama `feature/nueva-funcionalidad` ha sido integrada de forma lineal y limpia en `develop`, manteniendo un historial de commits claro y bien organizado.

Resumen

Este ejercicio práctico ha cubierto:

1. **Gestión de Claves SSH y Autenticación Segura**: Configuraste una clave SSH exclusiva para un repositorio, asegurando autenticación aislada.
2. **Control de Acceso a Repositorios**: Asignamos permisos de acceso en GitHub y aplicaste reglas de protección en la rama `main`.
3. **Técnicas Avanzadas de Gestión de Ramas y Uso de Rebase**: Creaste una rama de característica, realizaste un rebase para mantener un historial limpio y resolviste conflictos antes de fusionar los cambios en `develop`.