Laboratorio de Git y GitHub

Tabla de contenido

Ejercicio 1: Configuración Inicial de Git

Objetivo: Configurar Git en el entorno local.
Instrucciones:

1. **Instalación de Git:**
   - Verifica si Git está instalado ejecutando git --version en la terminal.
   - Si no está instalado, descárgalo e instálalo desde git-scm.com.

2. **Configuración global:**
   - Configura tu nombre de usuario:

     >_ git config --global user.name "Tu Nombre"

   - Configura tu correo electrónico:

     >_ git config --global user.email tu.email@example.com

3. **Verificación de la configuración:**
   - Ejecuta git config --list para verificar que la configuración se haya aplicado correctamente.

Resultados Esperados:

- Git instalado y configurado con el nombre y correo electrónico del usuario.

Ejercicio 2: Creación y Gestión de un Repositorio Local

Objetivo: Crear un repositorio local y gestionar archivos básicos.

Instrucciones:

1. **Crear un nuevo directorio para el proyecto:**
   >_ mkdir mi_proyecto
   >_ cd mi_proyecto

2. **Inicializar el repositorio Git:**
   >_ git init

3. **Crear un archivo README.md:**
   - Usa un editor de texto para crear README.md con una breve descripción del proyecto.

4. **Agregar y confirmar cambios:**
   >_ git add README.md
   >_ git commit -m "Agregar archivo README.md inicial"

5. **Crear un archivo .gitignore:**
   - Añade al menos dos tipos de archivos o directorios que no deseas que Git rastree (por ejemplo, node_modules/, *.log).

6. **Verificar el estado del repositorio:**
   >_git status

Resultados Esperados:

- Un repositorio Git inicializado con un README.md y un .gitignore correctamente configurado.

Ejercicio 3: Trabajando con Ramas

Objetivo: Crear y gestionar ramas para el desarrollo paralelo.

Instrucciones:

1. **Crear una nueva rama llamada desarrollo:**
   >_ git branch desarrollo

2. **Cambiar a la rama desarrollo:**
   >_git checkout desarrollo

3. **Realizar cambios en un archivo existente o crear uno nuevo (por ejemplo, funcionalidad.py):**
   - Agrega código o contenido al archivo.

4. **Agregar y confirmar los cambios:**
   >_ git add funcionalidad.py
   >_ git commit -m "Agregar funcionalidad inicial en la rama desarrollo"

5. **Volver a la rama main y fusionar los cambios de desarrollo:**
   >_ git checkout main
   >_ git merge Desarrollo

6. **Eliminar la rama desarrollo si ya no es necesaria:**
   >_ git branch -d desarrollo

Resultados Esperados:

- La rama desarrollo creada, los cambios fusionados en main, y la rama eliminada correctamente.

Ejercicio 4: Resolución de Conflictos de Fusión

Objetivo: Identificar y resolver conflictos al fusionar ramas.

Instrucciones:

1. **Crear una nueva rama feature1 desde main:**
   >_ git checkout -b feature1

2. **Modificar un archivo existente (por ejemplo, README.md) y confirmar los cambios:**
   >_ echo "Cambios desde feature1" >> README.md
   >_ git add README.md
   >_ git commit -m "Modificar README.md en feature1"

3. **Volver a main y modificar el mismo archivo de manera diferente:**
   >_ git checkout main
   >_ echo "Cambios desde main" >> README.md
   >_ git add README.md
   >_ git commit -m "Modificar README.md en main"

4. **Intentar fusionar feature1 en main:**
   >_ git merge feature1

5. **Resolver el conflicto manualmente:**
   - Abre README.md y edita las secciones conflictivas.
   - Marca los cambios resueltos.

6. **Agregar y confirmar la fusión:**
   >_ git add README.md
   >_ git commit -m "Resolver conflictos al fusionar feature1 en main"

Resultados Esperados:

- Conflicto detectado durante la fusión y resuelto correctamente por el usuario.

Ejercicio 5: Trabajo con Repositorios Remotos en GitHub

Objetivo: Conectar el repositorio local con GitHub y realizar operaciones básicas de push y pull.

Instrucciones:

1. **Crear un nuevo repositorio en GitHub:**
   - Ve a GitHub y crea un repositorio vacío llamado mi_proyecto.

2. **Agregar el repositorio remoto a tu repositorio local:**
   >_ git remote add origin https://github.com/tu_usuario/mi_proyecto.git

3. **Enviar (push) los cambios locales al repositorio remoto:**
   >_ git push -u origin main

4. **Clonar el repositorio remoto en una nueva ubicación:**
   >_ cd ..
   >_ git clone https://github.com/tu_usuario/mi_proyecto.git mi_proyecto_clonado
   >_ cd mi_proyecto_clonado

5. **Realizar un cambio en el repositorio clonado y enviar los cambios:**
   >_ echo "Nuevo cambio desde el clon" >> README.md
   >_ git add README.md
   >_ git commit -m "Agregar nuevo cambio desde el clon"
   >_ git push origin main

6. **Actualizar el repositorio original con los cambios remotos:**
   >_ cd ../mi_proyecto
   >_ git pull origin main

Resultados Esperados:

- El repositorio local y el remoto en GitHub sincronizados correctamente.
- Cambios reflejados en ambas ubicaciones.

Ejercicio 6: Colaboración mediante Pull Requests

Objetivo: Utilizar pull requests para colaborar en proyectos.

Instrucciones:

1. **Fork del repositorio original:**
   - En GitHub, realiza un fork del repositorio mi_proyecto a tu cuenta personal.

2. **Clonar el fork en tu máquina local:**
   >_ git clone https://github.com/tu_usuario_fork/mi_proyecto.git
   >_ cd mi_proyecto

3. **Configurar el repositorio original como remoto upstream:**
   >_ git remote add upstream https://github.com/tu_usuario/mi_proyecto.git

4. **Crear una nueva rama para una funcionalidad o mejora:**
   >_ git checkout -b mejora_readme

5. **Realizar cambios en README.md y confirmar los cambios:**
   >_ echo "Mejora en la documentación" >> README.md
   >_ git add README.md
   >_ git commit -m "Mejorar README.md con nueva información"

6. **Enviar los cambios a tu fork en GitHub:**
   >_ git push origin mejora_readme

7. **Crear un pull request desde GitHub:**
   - Ve a tu repositorio fork en GitHub.
   - Haz clic en "Compare & pull request".
   - Añade una descripción detallada de los cambios y crea el pull request hacia el repositorio original.
   -

8. **Revisar y fusionar el pull request en el repositorio original:**
   - En el repositorio original, revisa el pull request.
   - Si todo está correcto, fusiona el pull request.

Resultados Esperados:

- Pull request creado y fusionado correctamente, integrando las mejoras al repositorio original.

Ejercicio 7: Uso de Etiquetas (Tags) y Versionado

Objetivo: Crear y gestionar etiquetas para el versionado del proyecto.

Instrucciones:

1. **Crear una etiqueta anotada para una versión específica:**
   >_ git tag -a v1.0 -m "Versión 1.0 - Lanzamiento inicial"

2. **Listar todas las etiquetas existentes:**
   >_ git tag

3. **Enviar las etiquetas al repositorio remoto:**
   >_ git push origin v1.0

4. **Crear una nueva versión con cambios adicionales y etiquetar:**
   - Realiza algunos cambios en el proyecto.
   - Agrega y confirma los cambios:

   >_ git add .
   >_ git commit -m "Añadir nuevas funcionalidades para la versión 1.1"

   - Crear y enviar la etiqueta:

   >_ git tag -a v1.1 -m "Versión 1.1 - Nuevas funcionalidades"
   >_ git push origin v1.1

Resultados Esperados:

- Etiquetas v1.0 y v1.1 creadas y enviadas al repositorio remoto, reflejando diferentes versiones del proyecto.

Ejercicio 8: Revertir Cambios y Usar git revert

Objetivo: Aprender a deshacer cambios de manera segura utilizando git revert.

Instrucciones:

1. **Identificar el ID del commit que deseas revertir:**
   >_ git log –oneline

2. **Revertir un commit específico:**
   >_ git revert <commit_id>

3. **Confirmar el mensaje del commit de reversión y finalizar:**
   - Se abrirá el editor para confirmar el mensaje. Guarda y cierra el editor.

4. **Verificar que el commit ha sido revertido:**
   >_ git log –oneline

5. **Enviar los cambios al repositorio remoto:**
   >_ git push origin main

Resultados Esperados:

- El commit seleccionado ha sido revertido correctamente, y los cambios han sido reflejados en el repositorio remoto.

Ejercicio 9: Uso de Stash para Guardar Cambios Temporales

Objetivo: Utilizar git stash para guardar cambios sin comprometerlos.

Instrucciones:

1. **Realizar cambios en uno o más archivos sin agregarlos al área de preparación:**
   - Edita funcionalidad.py pero no hagas git add.

2. **Guardar los cambios en el stash:**
   >_ git stash

3. **Verificar que el directorio de trabajo está limpio:**
   >_ git status

4. **Crear una nueva rama y cambiar a ella:**
   >_ git checkout -b nueva_rama

5. **Aplicar los cambios del stash en la nueva rama:**
   >_ git stash apply

6. **Confirmar que los cambios han sido aplicados:**
   >_ git status

Resultados Esperados:

- Los cambios guardados en el stash han sido aplicados correctamente en la nueva rama, sin afectar el estado del repositorio original.

Ejercicio 10: Revisión y Optimización del Historial con git rebase

Objetivo: Entender y utilizar git rebase para mantener un historial de commits limpio.

Instrucciones:

1. **Crear una rama de características feature_rebase desde main:**
   >_ git checkout -b feature_rebase

2. **Realizar múltiples commits en feature_rebase:**
   - Edita archivos y realiza al menos tres commits distintos.

3. **Volver a main y realizar un commit adicional:**
   >_ git checkout main
   >_ echo "Cambios en main antes del rebase" >> README.md
   >_ git add README.md
   >_ git commit -m "Actualizar README.md en main"

4. **Volver a la rama feature_rebase y realizar el rebase sobre main:**
   >_ git checkout feature_rebase
   >_ git rebase main

5. **Resolver cualquier conflicto que pueda surgir durante el rebase:**
   - Edita los archivos conflictivos, añade los cambios y continúa el rebase:
   >_ git add archivo_conflictivo
   >_ git rebase –continue

6. **Enviar los cambios rebased al repositorio remoto (puede requerir fuerza si ya fueron enviados previamente):**
   >_ git push origin feature_rebase –force

Resultados Esperados:

- La rama feature_rebase se ha actualizado con los últimos cambios de main, manteniendo un historial lineal y limpio.