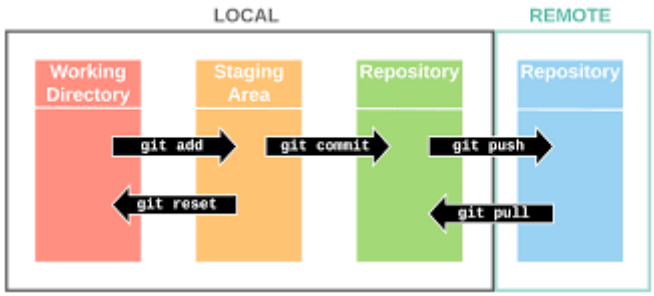
Sumario

1. Introducción



Git es un sistema de control de versiones distribuido que se utiliza para rastrear cambios en el código fuente durante el desarrollo de software. Aquí tienes una breve visión general:

1. **¿Qué es Git?** Git es un sistema de control de versiones que te permite realizar un seguimiento de los cambios en tus archivos a lo largo del tiempo. Facilita la colaboración entre varias personas en un proyecto.
2. **Repositorios:** En Git, tu proyecto se almacena en un repositorio. Puede ser local en tu máquina o remoto en un servidor.
3. **Instantáneas (Commits):** En lugar de rastrear cambios en archivos, Git realiza "commits" que son instantáneas de tu proyecto en un momento específico. Cada commit tiene un identificador único.

4. **Áreas de trabajo (Working Directory, Staging Area, Repository):**
    - **Working Directory:** Es donde trabajas en tus archivos.
    - **Staging Area (Área de preparación):** Aquí eliges qué cambios se incluirán en el próximo commit.
    - **Repository (Repositorio):** Es la base de datos donde se almacenan los commits y la historia del proyecto.
5. **Comandos Básicos:**
    - `git init`: Inicia un nuevo repositorio.
    - `git add`: Añade cambios al área de preparación.
    - `git commit`: Crea un nuevo commit.
    - `git status`: Muestra el estado de los archivos en tu repositorio.
    - `git log`: Muestra la lista de commits.

2. Comandos básicos Linux

Aquí tienes la **definición y un ejemplo de uso** para cada uno de los comandos que mencionaste. Todos son comandos básicos de Linux usados para gestionar archivos y directorios.

1. `mkdir` – *Make Directory*

Definición:

Crea uno o varios directorios nuevos.

Ejemplo:

```
mkdir proyectos
```

Crea un directorio llamado `proyectos`.

2. `cd` – Change Directory

Definición:

Cambia el directorio actual al que se indique.

Ejemplo:

```
cd proyectos
```

Entra en el directorio `proyectos`.

3. `cp` – *Copy*

Definición:

Copia archivos o directorios de una ubicación a otra.

Ejemplo:

```
cp archivo.txt copia.txt
```

Copia `archivo.txt` y crea una copia llamada `copia.txt` en el mismo directorio.

Para copiar directorios completos se usa: `cp -r carpeta1 carpeta2`

4. `mv` – *Move*

Definición:

Mueve o renombra archivos y directorios.

Ejemplo:

```
mv archivo.txt documentos/
```

Mueve `archivo.txt` al directorio `documentos`.

Ejemplo para renombrar:

```
mv archivo.txt nuevo_nombre.txt
```

5. `rm` – *Remove*

Definición:

Elimina archivos o directorios.

Ejemplo:

```
rm archivo.txt
```

Elimina el archivo `archivo.txt`.

Para eliminar directorios con contenido: ***`rm -r nombre_directorio`***

6. `rmdir` – *Remove Directory*

Definición:

Elimina directorios vacíos.

Ejemplo:

```
rmdir carpeta_vacia
```

Elimina la carpeta `carpeta_vacia` si no tiene contenido.

7. `ls` – *List*

Definición:

Lista el contenido de un directorio.

Ejemplo:

```
ls
```

Muestra los archivos y carpetas del directorio actual.

Con más detalle: `ls -l`

Para ver archivos ocultos: `ls -a`

Para ver directorios, subdirectorios y su contenido:

ls -lR nombre_del_directorio

Se va a definir la siguiente estructura de directorios o carpetas:

documentosPrueba/

├── entornos/

│ └── documentosGit/

├── programacion/

└── baseDatos/

2.1 Actividad

Crea el directorio principal.

- ***mkdir documentosPrueba***

Entras en él para crear los subdirectorios.

- ***cd documentosPrueba***

Creas los subdirectorios dentro de documentosPrueba.

- ***mkdir entornos, mkdir programacion, mkdir baseDatos***

Entras en entornos para crear su subdirectorio.

- ***cd entornos***

Crea el subdirectorio dentro de entornos.

- ***mkdir documentosGit***

1. Con ayuda del editor nano vas a crear un documento denominado doc1.txt con el texto: “Ejercicios con comandos Linux”.
2. Guarda este documento con el nombre de comandosLinux.txt.
3. Copia este fichero dentro de la carpeta entornos
4. Copia este fichero dentro de la carpeta programacion
5. Copia este documentos dos veces dentro de la carpeta baseDatos: la primera vez con el nombre bd1.txt, la segunda vez con el nombre de bd2.txt
6. Copia bd1.txt en la carpeta documentosGit con el nombre final de git1.txt
7. Copia el documento bd2.txt en la carptea documentosGit con el nombre de git2.txt
8. En la carpeta documentosPrueba define un nuevo documento con el nano de nombre prueba.txt.
9. Copia el documento prueba.txt en cada una de las carpetas creadas anteriormente.

10. Visualliza el contenido de la carpeta documentosPrueba.
11. Borra el documento existente en ella denominado prueba.txt.
12. En la carpeta entornos guarda un documento creado con nano de nombre entornos.txt.
13. Copia el fichero entornos.txt en cada una de las carpetas.
14. Duplica el fichero git1.txt que se encuentra en la carpeta documentosGit para que exista en ella otro fichero con el nombre git3.txt.
15. Duplica el fichero git2.txt que se encuentra en la carpeta documentosGit para que exista en ella otro fichero con el nombre git4.txt.
16. Mueve el fichero git3.txt a la carpeta programacion.
17. Mueve el fichero git4.txt a la carpeta documentosGit
18. Visualiza el contenido de la carpeta documentosGit.
19. Borra el fichero bd2.txt existente en la carpeta baseDatos
20. Visualiza el contenido de todas las carpetas creadas en este ejercicio.

2.1.2 Solución actividad

Crea el directorio principal.

- ***mkdir documentosPrueba***

Entras en él para crear los subdirectorios.

- ***cd documentosPrueba***

`Creas los subdirectorios dentro de documentosPrueba.`

- ***mkdir entornos, mkdir programacion, mkdir baseDatos***

Entras en `entornos` para crear su subdirectorio.

- ***cd entornos***

`Crea el subdirectorio dentro de entornos.`

- ***mkdir documentosGit***

1. Con ayuda del editor nano vas a crear un documento denominado doc1.txt con el texto: “Ejercicios con comandos Linux”.

   nano doc1.txt

2. Guarda este documento con el nombre de comandosLinux.txt.

   cp doc1.txt comandosLinux.txt

3. Copia este fichero dentro de la carpeta entornos

   cp comandosLinux.txt entornos

4. Copia este fichero dentro de la carpeta programacion

   cp comandosLinux.txt programacion

5. Copia este documento dos veces dentro de la carpeta baseDatos: la primera vez con el nombre bd1.txt, la segunda vez con el nombre de bd2.txt

   cp comandosLinux.txt baseDatos/bd1.txt

   cp comandosLinux.txt baseDatos/bd2.txt

6. Copia bd1.txt en la carpeta documentosGit con el nombre final de git1.txt

   cd baseDatos

   cp bd1.txt /home/usuario/documentosGit/git1.txt

7. Copia el documento bd2.txt en la carptea documentosGit con el nombre de git2.txt

   cp bd2.txt /home/usuario/documentosGit/git2.txt

8. En la carpeta documentosPrueba define un nuevo documento con el nano de nombre prueba.txt.

   cd documentosPrueba

   nano prueba.txt

9. Copia el documento prueba.txt en cada una de las carpetas creadas anteriormente.

   cp prueba.txt /home/usuario/documentosGit/entornos

10. Visualliza el contenido de la carpeta documentosPrueba.

    cd /home/usuario/documentosGit/documentosPrueba

    ls

11. Borra el documento existente en ella denominado prueba.txt.

    rm prueba.txt

12. En la carpeta entornos guarda un documento creado con nano de nombre entornos.txt.

    cd /home/usuario/documentosGit/entornos

    nano entornos.txt

13. Copia el fichero entornos.txt en cada una de las carpetas.

    cp entornos.txt /home/usuario/documentosGit

    ...

14. Duplica el fichero git1.txt que se encuentra en la carpeta documentosGit para que exista en ella otro fichero con el nombre git3.txt.

    cd /home/usuario/documentosGit

    cp git1.txt git3.txt

15. Duplica el fichero git2.txt que se encuentra en la carpeta documentosGit para que exista en ella otro fichero con el nombre git4.txt.

    cp git2.txt git4.txt

16. Mueve el fichero git3.txt a la carpeta programacion.

    mv git3.txt /home/usuario/documentosGit/programacion

17. Mueve el fichero git4.txt a la carpeta programacion.

    mv git4.txt /home/usuario/documentosGit/programacion

18. Visualiza el contenido de la carpeta documentosGit.

    ls -l

19. Borra el fichero bd2.txt existente en la carpeta baseDatos

    cd baseDatos

    rm bd2.txt

20. Visualiza el contenido de todas las carpetas creadas en este ejercicio.

    cd /home/usuario

    ls -l -R

3- Git – Instalación

Para instalar Git sobre un sistema operativo Ubuntu deberás seguir los pasos siguientes:

1. Actualizar el índice de paquetes

   sudo apt update

2. Intalar Git

   sudo apt install git

3. Verificar que la instalación se realizó correctamente

   git --version

4. Inicialización de un repositorio

inicializar un repositorio en Git es el primer paso para comenzar a rastrear los cambios en tu proyecto! Aquí tienes los pasos básicos:

1. Navega al directorio de tu proyecto

cd /ruta/del/tu/proyecto

2. Inicializa el repositorio:

Una vez que estés en el directorio de tu proyecto, ejecuta el siguiente comando para inicializar un repositorio Git:

git init

3. **Añade archivos al repositorio:**

Después de inicializar el repositorio, puedes comenzar a rastrear archivos. Puedes agregar todos los archivos al repositorio utilizando:

git add .

4. **Realiza el primer commit:**

Después de agregar archivos al área de preparación, realiza tu primer commit para guardar esos cambios en el historial del repositorio. Utiliza el siguiente comando:

git commit -m "Primer commit"

Puedes personalizar el mensaje entre las comillas para describir brevemente los cambios que realizaste en este commit.

5. Incluir ficheros en el repositorio local o área de intercambio.

Agregar archivos a tu repositorio local en Git es una parte esencial del proceso de control de versiones.

1. **Navegar al directorio del proyecto:**

   cd /ruta/del/tu/proyecto

2. **Añadir archivos al área de preparación (staging area):**

   Utiliza el comando `git add` para agregar archivos al área de preparación. Puedes agregar archivos específicos o todos los archivos en el directorio. Por ejemplo, para agregar todos los archivos, ejecuta:

   git add .

   Si solo quieres agregar un archivo específico, reemplaza `.` con el nombre del archivo.

3. **Verificar el estado de los archivos:**

   Puedes verificar el estado de los archivos en el área de preparación utilizando:

   git status

   Esto te mostrará los archivos que se han modificado y están listos para ser confirmados (para hacer un commit).

4. **Realizar un commit:**

   Después de agregar los archivos al área de preparación, realiza un commit para guardar esos cambios en la historia del repositorio. Utiliza:

   git commit -m "Mensaje del commit"

   Personaliza "Mensaje del commit" con una descripción breve de los cambios que estás realizando.

5.1 Actividades

Ejercicio 1: Inicialización y Primer Commit

1. Crea un nuevo directorio para tu proyecto Git.
2. Inicializa un repositorio Git en ese directorio.
3. Crea un documento de Writer, denominado ***documento1.odt*** con el texto "Hola buenos días".
4. Añade el documento al área de preparación.
5. Realiza el primer commit con el mensaje "Versión inicial del documento".

Ejercicio 2: Modificación y Segundo Commit

1. Abre el documento de Writer y modifica el texto. Por ejemplo, cambia "Hola buenos días" a "Hola, ¿cómo estás?".
2. Guarda los cambios en el documento.
3. Utiliza **git status** para verificar los cambios.
4. Añade el documento modificado al área de preparación.
5. Realiza el segundo commit con el mensaje "Se modificó el saludo".

Ejercicio 3: Creación del tercer commit

1. Abre el documento de Writer y realiza otra modificación en el texto, diferente a la realizada en el ejercicio 2, por ejemplo: “Hola, buenas noches”
2. Guarda los cambios y añade el documento modificado al área de preparación.
3. Realiza el tercer commit en la rama principal con el mensaje "Se realizó otro cambio en el saludo".

6. Visualizar los cambios en el repositorio

El comando **git status** en Git te proporciona información sobre el estado actual de tu repositorio. Al ejecutar **git status**, obtienes detalles sobre los cambios que has realizado en tus archivos y cómo se encuentran en relación con el repositorio. Aquí hay algunas de las cosas que **git status** puede mostrarte:

1. **Archivos Modificados:**
    - Te mostrará una lista de archivos que han sido modificados desde el último commit.
2. **Archivos en el Área de Preparación (Staging Area):**
    - Mostrará archivos que han sido modificados y están listos para ser incluidos en el próximo commit.
3. **Archivos sin Seguir:**
    - Te indicará si hay nuevos archivos en tu directorio de trabajo que Git no está rastreando actualmente.
4. **Rama Actual:**
    - Muestra la rama en la que te encuentras actualmente trabajando.
5. **Información sobre Commits Pendientes:**
    - Si has realizado commits pero no has empujado esos cambios a un repositorio remoto, te lo indicará.
6. **Sugerencias y Consejos:**
    - Proporciona información adicional, como comandos sugeridos para realizar acciones específicas.

7. Visualización de todos los commits existentes en un proyecto.

Hay varias posibilidades de visualizar los commits de un proyecto:

Ver solo los mensajes de los commits en una sola línea:

```
git log --oneline
```

- Esto es útil para obtener una vista rápida y resumida de los commits.

Mostrar el historial en formato gráfico (ideal para proyectos con ramas):

```
git log --oneline --graph --all
```

- Esto mostrará un gráfico de las ramas y los merges, facilitando la comprensión de la estructura del proyecto.

Incluir solo los commits de un archivo específico:

```
git log -- <archivo>
```

- Muestra los commits que afectaron solo a un archivo en particular.

8. Restauración a un commit de una forma definitiva.

Podemos regresar a un commit anterior de una forma definitiva, perdiendo todos los commits posteriores de nuestro proyecto. El comando git que lo permite es el siguientes:

```
git reset --hard <commit_id>
```

9. checkout

Git checkout se puede utilizar para cambiar a otra rama de nuestro proyecto (veremos ejemplos del mismo una vez hallamos utilizado las ramas), también se puede utilizar el comando git checkout para cambiar a un commit existente en la rama en la que nos encontramos:

1. **Historia del Repositorio:**

A --- B --- C (rama principal)

2. **Cambiar a un Commit Anterior:**

git checkout B

Ahora estás en el commit B con un estado "detached HEAD".

En este momento podrás acceder/ver el contenido de los archivos de tu proyecto, afectados en este commit. Debes de tener en cuenta que como te encuentras en el estado "detached HEAD", si realizas alguna modificación en alguno de los archivos pertenecientes a este commit, aunque guardes los cambios, después de saltar a otro commit mediante el comando git checkout, por ejemplo:

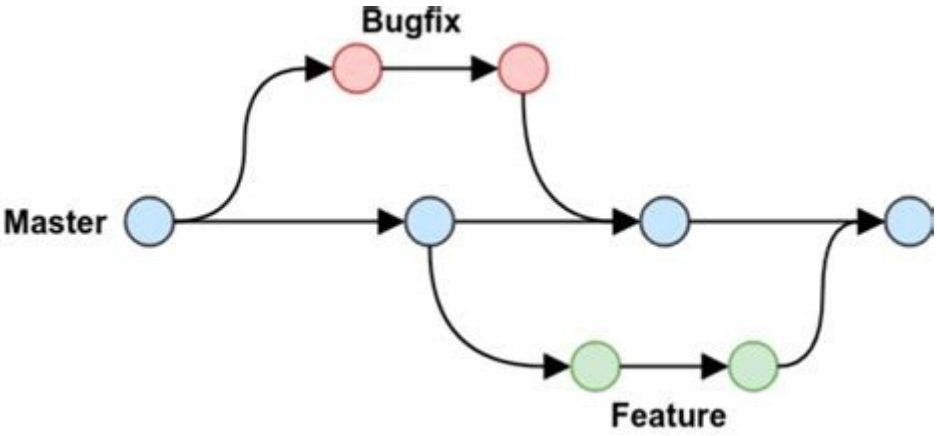
git checkout C

Los últimos cambios prácticados sobre algunos de los archivos del proyecto se perderán.

9.1 Actividades

1. A partir de los commits realizados en el apartado 4.1 Actividades de este temario. Utiliza el comando git checkout para acceder a cada una de las versiones o commits creados. Comprueba que después de acceder a un determinado commit el contenido del documento writer documento1.odt aparecerá modificado.

10. Ramas



En Git, las ramas son una característica fundamental que permite trabajar en paralelo en diferentes líneas de desarrollo. Aquí hay una introducción básica a cómo trabajar con ramas en Git:

Crear una nueva rama:

git branch nombre_de_rama

Cambiar a una rama existente:

git checkout nombre_de_rama

O, en versiones más recientes de Git:

git switch nombre_de_rama

Crear una nueva rama y cambiar a ella en un solo comando:

git checkout -b nombre_de_rama

O, en versiones más recientes de Git:

git switch -c nombre_de_rama

Ver todas las ramas en tu repositorio local:

git branch

Fusionar una rama a la rama actual:

Primero, asegúrate de estar en la rama a la que quieres fusionar cambios:

git checkout rama_destino

Luego, realiza la fusión:

git merge rama_fuente

Eliminar una rama:

git branch -d nombre_de_rama

Este comando elimina la rama si todos los cambios de la rama han sido fusionados. Si hay cambios no fusionados, puedes forzar la eliminación con -D:

git branch -D nombre_de_rama

10.1 Actividades

Aquí tienes un ejercicio práctico para practicar la creación de ramas, la creación de commits en una rama, la fusión de dos ramas y la eliminación de la rama fusionada. Este ejercicio asume que ya tienes un repositorio de Git configurado:

1. **Crea un nuevo repositorio de Git (si no tienes uno):**

   git init

2. **Crea un archivo y realiza tu primer commit en la rama principal (master):**

   echo "Contenido inicial" > archivo.txt

   git add archivo.txt

   git commit -m "Primer commit en la rama principal"

3. **Crea una nueva rama llamada "nueva-funcionalidad" y cámbiate a ella:**

   git checkout -b nueva-funcionalidad

   # O, en versiones más recientes de Git:

   git switch -c nueva-funcionalidad

4. **Realiza algunos cambios en la nueva rama y realiza commits:**

   echo "Nueva funcionalidad" >> archivo.txt

   git add archivo.txt

   git commit -m "Agrega nueva funcionalidad"

5. **Regresa a la rama principal (master):**

   git checkout master

   # O, en versiones más recientes de Git:

   git switch master

6. **Realiza cambios en la rama principal:**

   echo "Cambios en la rama principal" >> archivo.txt

   git add archivo.txt

   git commit -m "Cambios en la rama principal"

7. **Fusiona la rama "nueva-funcionalidad" en la rama principal:**

   git merge nueva-funcionalidad

8. **Elimina la rama "nueva-funcionalidad" (ya que ya ha sido fusionada):**

   git branch -d nueva-funcionalidad

   Si hay problemas al eliminar la rama debido a cambios no fusionados, puedes utilizar -D para forzar la eliminación:

   git branch -D nueva-funcionalidad

   Este ejercicio simula un flujo de trabajo típico donde se crea una rama para desarrollar una nueva funcionalidad, se realizan cambios en ambas ramas y luego se fusiona la nueva funcionalidad en la rama principal. Después, la rama de la nueva funcionalidad se elimina, ya que ya no es necesaria.

11. Diferencia entre ramas – Diff

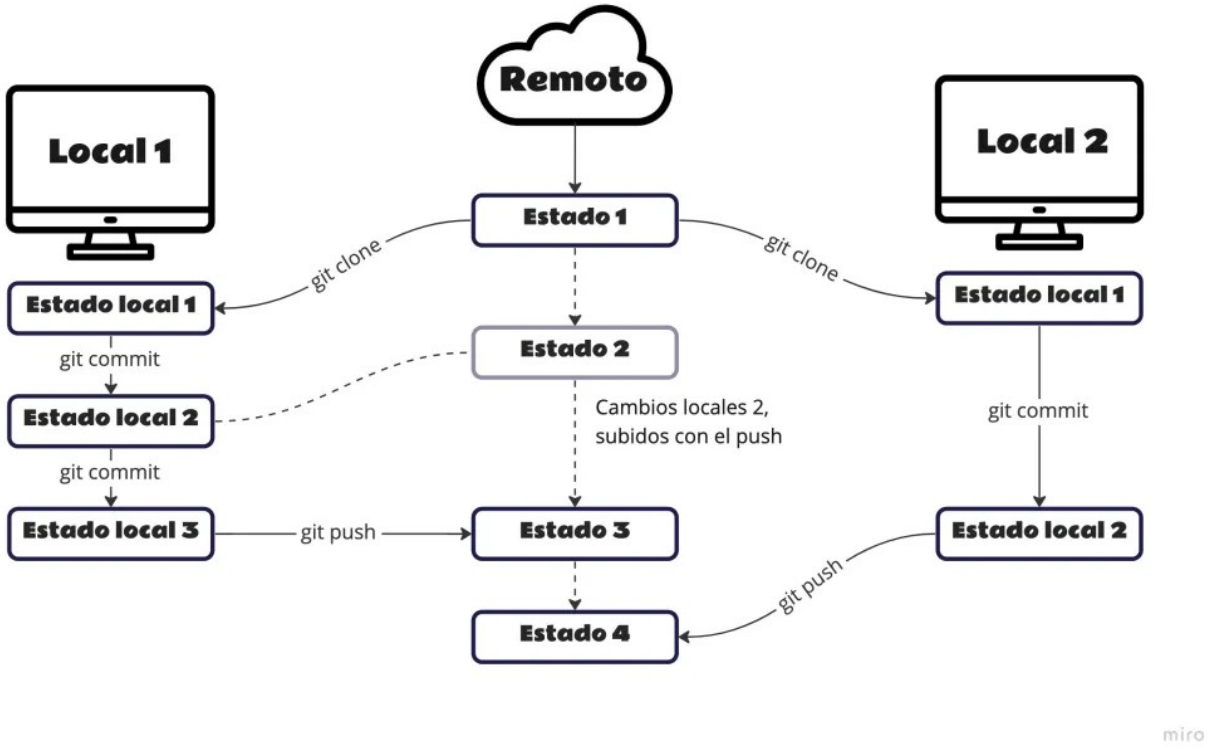
Sitúate en una rama determinada, por ejemplo la rama Master y a continuación escribes el comando que representa las diferencias de la rama en la que estás situado (por ejemplo la rama Master) con otra rama cualquiera (rama2) de tu repositorio:

git diff rama2

Si lo que deseas es ver las diferencias en un archivo entre dos ramas:

git rama1 rama2 – fichero.txt

11.1 Repositorios remotos y locales



12. GitHub – Subir repositorios al repositorio remoto

Un repositorio remoto en GitHub es un repositorio que no está almacenado en tu máquina local, sino en los servidores de GitHub. Puedes utilizar GitHub para alojar proyectos de software, colaborar con otros desarrolladores y gestionar versiones de código mediante el control de versiones Git. Aquí hay algunos pasos básicos para trabajar con un repositorio remoto en GitHub:

Crear un Repositorio en GitHub:

1. **Inicia Sesión en GitHub:**
    - Accede a GitHub y inicia sesión en tu cuenta.
2. **Crea un Nuevo Repositorio:**
    - En la esquina superior derecha, haz clic en el botón + y selecciona "Nuevo repositorio".
    - Completa los detalles del nuevo repositorio, como el nombre, descripción y opciones adicionales.
    - Haz clic en "Crear repositorio".

Conectar un Repositorio Local a GitHub:

3. **Inicia un Repositorio Local:**
    - Si aún no tienes un repositorio local, puedes iniciar uno ejecutando `git init` en el directorio del proyecto.
4. **Agrega un Remoto:**

- Copia la URL de tu repositorio en GitHub.
- En tu terminal, navega al directorio de tu proyecto y ejecuta:

- `git remote add origin URL_del_repositorio_en_GitHub`

  Sustituye "URL_del_repositorio_en_GitHub" con la URL que copiaste.

- **Verifica el Remoto:**
- Puedes verificar que el remoto está configurado correctamente con:
  - ***`git remote -v`***

Trabajar con el Repositorio Remoto:

6. **Empuja Cambios al Repositorio Remoto:**
   - Después de hacer cambios y confirmar localmente, puedes enviar tus cambios al repositorio remoto con:

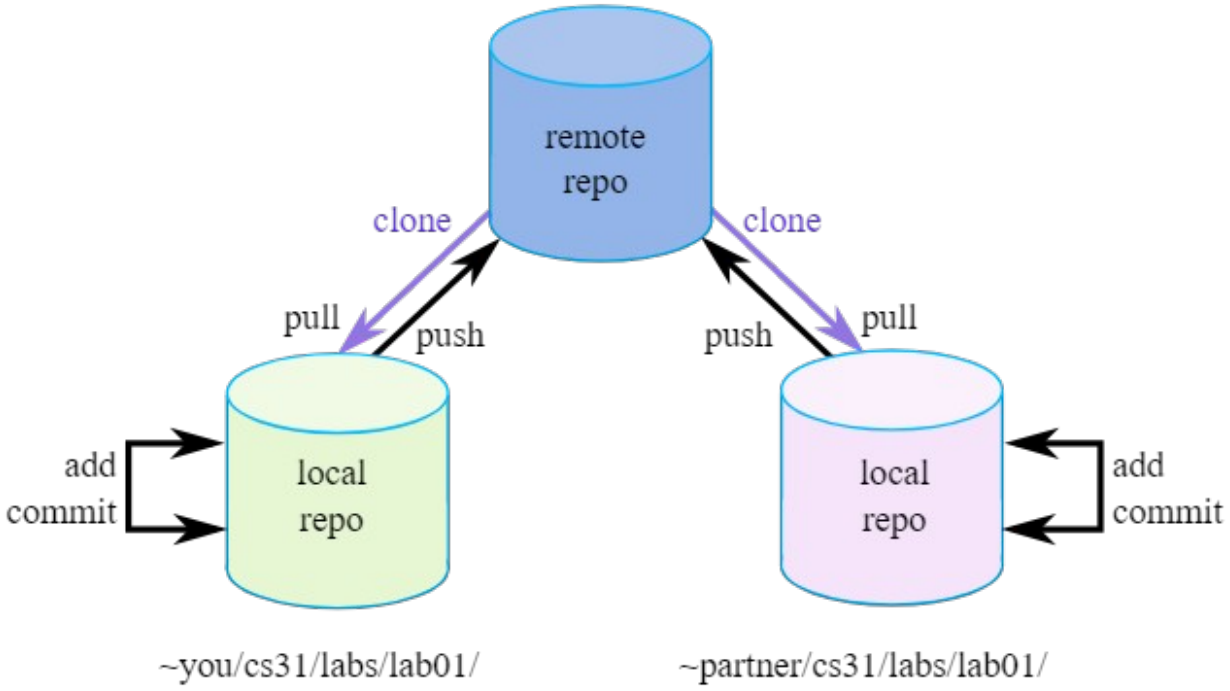
- ***`git push -u origin nombre-de-la-rama`***

  Esto establece la rama remota como rama por defecto para futuros `git push`.

Para la realización de esta transferencia de repositorio a un ordenador remoto, será preciso validarse, especificando por un lado el nombre de un usuario válido en el repositorio remoto (gitHub) y por otro la contraseña, en este caso se deberá introducir el código correspondiente a un token generado en gitHub, para ello se deberá acceder a la dirección siguiente:

https://github.com/settings/tokens

13. GitHub – Clonación de un repositorio para uso local



Tenemos un repositorio en gitHub, o bien por que se ha creado desde esta plataforma o bien por en que su momento se subió desde un equipo local. En cualquier caso podría interesar descargar dicho repositorio para su uso en local. Para ello hará falta realizar una clonación del mismo y que dicho repositorio clonado sea guardado en local.

Para clonar un repositorio de GitHub en tu ordenador, sigue estos pasos:

1. **Obtén la URL del repositorio:**
    - Ve al repositorio en GitHub.
    - Haz clic en el botón verde que dice "Code" y copia la URL que se muestra. Puedes elegir entre HTTPS o SSH, dependiendo de tus preferencias y configuración.

2. **Abre la terminal o línea de comandos:**
    - En Windows, puedes usar Command Prompt o PowerShell.
    - En Linux o macOS, utiliza la Terminal.
3. **Navega al directorio donde deseas clonar el repositorio:**

    cd ruta/del/directorio

4. **Clona el repositorio:**

    git clone <URL_del_repositorio>

    Sustituye `<URL_del_repositorio>` con la URL que copiaste de GitHub.

    Si estás usando HTTPS, el comando se verá así:

    git clone https://github.com/tu_usuario/tu_repositorio.git

13.1 Actividades

Ejercicio 1: Crear un Repositorio en GitHub

1. Crea una cuenta en GitHub si aún no tienes una.
2. Inicia sesión en GitHub.
3. Crea un nuevo repositorio. Puedes llamarlo, por ejemplo, "**MiPrimerRepositorio**".

Ejercicio 2: Crear un Documento de Texto

1. Dentro de tu nuevo repositorio, crea un nuevo archivo de texto llamado **`documento.txt`**.
2. Añade algunas líneas de texto al documento para introducir contenido.

Ejercicio 3: Realizar Commits

1. Realiza un commit inicial con el mensaje "Commit inicial: añadido documento.txt".
2. Modifica el contenido de **`documento.txt`**.
3. Realiza un nuevo commit con el mensaje "Modificación: contenido actualizado".
4. Repite el paso anterior una vez más.

Ejercicio 4: Clonar el Repositorio Localmente

1. Abre la terminal o línea de comandos en tu equipo local.
2. Navega al directorio donde deseas clonar el repositorio.
3. Utiliza el comando `git clone` para clonar tu repositorio desde GitHub:

```
git clone https://github.com/tu_usuario/MiPrimerRepositorio.git
```

Asegúrate de reemplazar `tu_usuario` con tu nombre de usuario en GitHub.

Notas Adicionales:

- Puedes verificar el historial de commits utilizando `git log` tanto en GitHub como en tu máquina local.

14. Clonación, modificación y subida de una rama de github

1. A partir de una repositorio github, repo6, que contiene una rama denominada r2. Se va a realizar una clonación en local de todo el repositorio:

   git clone -b r2 --single-branch https://github.com/jalbertmagro/repo6

2. En local modificamos un archivo existente, por ejemplo texto2.txt
3. Una vez modificado lo añadimos al área de ensayo:

   git add texto2.txt

4. A continuación creamos un commit que contenga esta última modificación:

   git commit -m "version2"

5. Finalmente subimos el contenido de la rama r2 del repositorio repo6 de github

   git push -u origin

15. git pull, descarga de ficheros de remoto a local

Git clone se utiliza para crear un repositorio nuevo a partir de un repositorio ya existente, por ejemplo a partir de un repositorio remoto ubicado en github.

Por otra parte, **git pull** se utilizará para actualizar la información existente en un repositorio remoto en un repositorio local que contiene parte de la información existente en el repositorio remoto pero que requiere de una última actualización.

Para actualizar el repositorio local con las últimas modificaciones realizadas en el repositorio remoto habrá que entrar en el directorio local, asociado al repositorio a actualizar, y escribir el comando siguiente:

git pull origin nombre-rama

Donde nombre-rama será el nombre de la rama del repositorio remoto que queremos copiar sus actualizaciones en el repositorio local.

15.1 Actividades

Ejercicio 1: Actualización Simple

1. Clona un repositorio desde GitHub a tu máquina local utilizando `git clone`.

   ```
   git clone https://github.com/tu_usuario/tu_repositorio.git
   ```

2. Realiza cambios en el repositorio en GitHub (por ejemplo, agrega un nuevo archivo o modifica uno existente).

3. En tu máquina local, navega al directorio del repositorio y utiliza `git pull` para traer los cambios desde el repositorio remoto a tu máquina local.

   ```
   git pull origin nombre-rama
   ```

   Asegúrate de reemplazar `nombre-rama` con la rama actual del repositorio.

4. Observa cómo los cambios realizados en el repositorio remoto se reflejan en tu máquina local.