Introducción a Git y GitHub

Luis Miguel de la Cruz Salas

Instituto de Geofísica

Universidad Nacional Autónoma de México

Contenido

Git

(https://git-scm.com/)

Todos hemos utilizado estos nombres alguna vez, al llevar control de los cambios realizados a algún archivo importante. Sin embargo, es fácil darse cuenta de lo ineficiente que es este "método" y lo inmanejable que se vuelve con el paso del tiempo.

Git proporciona una mejor manera de gestionar los archivos.

De acuerdo con la Wikipedia:

> "Git es un software de control de versiones diseñado por Linus Torvalds, pensando en la eficiencia, la confiabilidad y compatibilidad del mantenimiento de versiones de aplicaciones cuando estas tienen un gran número de archivos de código fuente. Su propósito es llevar registro de los cambios en archivos de computadora incluyendo coordinar el trabajo que varias personas realizan sobre archivos compartidos en un repositorio de código."
>
> Git. (2023, 19 de noviembre). Wikipedia, La enciclopedia libre. Fecha de consulta: 15:43, noviembre 19, 2023 desde https://es.wikipedia.org/w/index.php?title=Git&oldid=155483121.

- El mantenimiento de Git actualmente está a cargo de Junio Hamano.
- Git es un sistema de control de versiones de software o de documentos en general.
- Git monitorea los archivos de un proyecto el cual es considerado como un repositorio de información.
- Git obtiene sus repositorios de dos maneras:
  - Convertir un directorio local con los archivos de un proyecto en un repositorio Git.
  - Clonar un repositorio de algún servicio en la nube (GitHub, GitLab, etc.).
- Git se instala localmente en tu computadora personal.
- Git es abierto y gratuito.

Git como un sistema de archivos.

Git gestiona los archivos de un repositorio como un sistema de archivos miniatura en donde se van almacenando sus estados en ciertos instantes (*snapshots*). Cada vez que se hace una confirmación de los archivos (*commit*) se toma una foto de su estado y se almacena en la base de datos. En la figura 1 se da una breve descripción de esto.

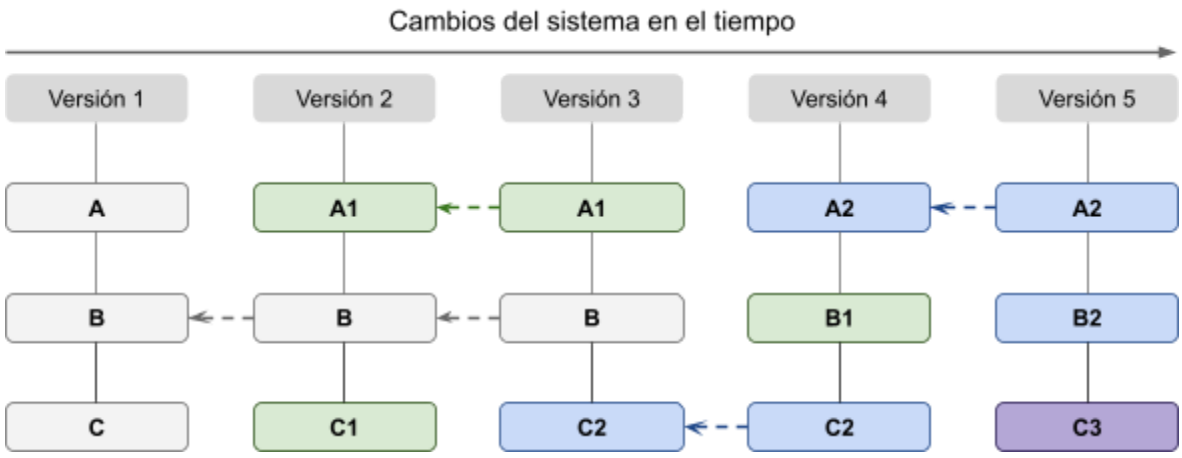


Figura 1. Se muestra un "mini sistema" de archivos con los archivos originales A, B, y C en la versión 1. Para la versión 2 se realizaron cambios en los archivos A y C, dando lugar a las versiones A1 y C1; el archivo B no cambió por lo que solo se hace referencia a la versión 1. Para la versión 3, cambió el archivo C1 para obtener la versión C2; en este caso A1 y B se mantuvieron sin cambios y esto es indicado haciendo referencia a las versiones anteriores. Las versiones 4 y 5 indican algunos cambios que se pueden identificar por los colores y las flechas en los archivos. Diagrama basado en la figura 5 de (Chacon, 2014).

Todos los cambios son registrados por Git mediante una cadena de caracteres creada usando el algoritmo SHA-1 de tal manera que es imposible perder el registro de estos cambios. Esta cadena se compone de 40 caracteres hexadecimales y son calculados con base en el contenido del archivo y/o de la estructura de los directorios y es algo similar a lo siguiente:

7861433d5eb8ada9b64e805a429d019aec778317

Git almacena estos valores en su base de datos para llevar el control de todos los cambios.

Los tres estados de los archivos en Git.

En Git, los archivos tienen tres estados principales (véase figura 2):

- Modificado (*modified*).
  Se ha modificado el archivo pero este cambio aún no ha sido confirmado en la base de datos.
- Preparado (*staged*).
  El archivo, en su versión actual, se ha marcado como modificado para que se agregue a la base de datos en la próxima confirmación.
- Confirmado (*committed*).
  El archivo se ha almacenado en la base de datos local de manera segura.

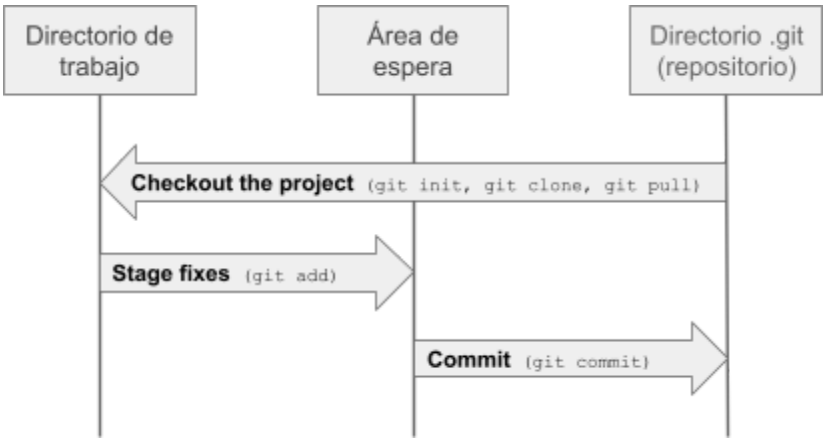


Figura 2. Tres diferentes áreas y etapas por las que pasan los archivos de un repositorio. Para obtener una versión del repositorio se pueden usar los comandos **`git init`** (inicializa un repositorio local), **`git clone`** (clona los archivos de un repositorio remoto) o **`git pull`** (obtienen una nueva versión de un repositorio); el repositorio así obtenido es el directorio de trabajo donde se modifican los archivos. Con **`git add`** los archivos se preparan y se envían al área de espera (*staging area*) y con **`git commit`** se confirman los cambios en la base de datos (directorio **`.git`**). Con base en la figura 6 de (Chacon, 2014).

Para tener claros estos tres estados, en lo que sigue vamos a crear un repositorio local con algunos archivos, los cuales modificaremos para identificar su flujo por cada uno de los diferentes estados.

Configuración inicial.

Para que Git identifique el autor de cada cambio en los archivos de un repositorio se debe configurar el nombre de usuario y el correo electrónico[1] usando los comandos que siguen:

```
git config --global user.name "luiggix"
git config --global user.email "luiggix@gmail.com"
```

Para revisar que todo haya salido correctamente se usa el siguiente comando:

```
git config --global --list
user.name=luiggix
user.email=luiggix@gmail.com
```

Para obtener ayuda sobre lo que se puede hacer con **`git config`** se puede usar:

```
git config -h
usage: git config [<options>]

Config file location
    --global              use global config file
    --system              use system config file
    --local               use repository config file
    --worktree            use per-worktree config file

...
```

[1] Tanto el usuario como el correo electrónico deben coincidir con los que se usen en un servicio de alojamiento remoto como GitHub, GitLab, entre otros, para poder hacer sincronizaciones entre el repositorio local y el remoto.

Observación. En que sigue, los comandos se deben teclear en una Terminal de Jupyter Lab. Los comandos se muestran en recuadros grises con font **`courier resaltado`**. Algunos de estos comandos proveen salida y otros no. La salida se muestra en font `courier simple`.

Creación de un repositorio local.

Crear un directorio para almacenar los archivos con el siguiente comando:

```
mkdir curso
```

Cambiarse al nuevo directorio :

```
cd curso
```

Inicializar Git para el repositorio que estará almacenado en el directorio **`curso`**:

```
git init
Initialized empty Git repository in /home/jovyan/GIT/curso/.git/
```

El comando **`git init`** genera un repositorio local vacío de Git y básicamente crea un directorio dentro de la carpeta **`curso`** con el nombre **`.git`** en donde pone toda la información del repositorio para darle seguimiento a las diferentes versiones de los archivos.

Para listar los archivos actuales del directorio se utiliza el siguiente comando:

```
ls -la
total 0
drwxr-xr-x 3 jovyan users  26 Dec 22 00:27 .
drwxr-xr-x 3 jovyan users  27 Dec 22 00:27 ..
drwxr-xr-x 7 jovyan users 155 Dec 22 00:27 .git
```

También puedes usar el comando **`ls -laR`**, el cual lista los archivos de manera recursiva dentro del directorio actual.

Observación. Todos los comandos de Git que siguen se deben teclear dentro de un directorio que haya sido inicializado con **`git init`**. En este caso se asume que todo se realiza dentro del directorio **`curso`**.

Creación de un archivo en el repositorio.

El comando **`git init`** generó todo lo necesario para dar seguimiento a los archivos que se encuentren en el directorio **`curso`**. Es posible checar el estado actual del repositorio con el comando **`git status`**:

```
git status
On branch main

No commits yet

nothing to commit (create/copy files and use "git add" to track)
```

Observa los mensajes que proporciona el comando anterior:

- primero indica en qué rama del proyecto nos encontramos (`On branch main`);
- después nos dice que no se ha realizado ninguna confirmación (`No commits yet`), es decir el proyecto está vacío actualmente;
- finalmente agrega que no hay nada que confirmar (`nothing to commit`) y nos da una sugerencia, entre paréntesis, para agregar archivos al proyecto. Casi siempre Git nos va a dar recomendaciones que estarán entre paréntesis.

Vamos a crear un archivo de texto con el siguiente comando:

```
echo "Simplifiquemos la gramática antes de que la gramática termine por simplificarnos a nosotros" > hola.txt
```

El comando anterior crea un archivo con el nombre **`hola.txt`**, para ver su contenido usamos el siguiente comando:

```
cat hola.txt
Simplifiquemos la gramática antes de que la gramática termine
por simplificarnos a nosotros
```

Y también podemos listar los archivos que existen en el directorio:

```
ls -l
total 4
-rw-r--r-- 1 jovyan users 94 Dec 22 01:45 hola.txt
```

Ahora veamos el estado actual del repositorio usando el comando **`git status`**:

```
git status

On branch main

No commits yet

Untracked files:
  (use "git add <file>..." to include in what will be committed)
        hola.txt

nothing added to commit but untracked files present (use "git add"
to track)
```

Observa que Git acaba de identificar el nuevo archivo **`hola.txt`** dentro del directorio **`curso`**, al cual no le está dando seguimiento (`Untracked files`). En general, este comando pone en color rojo la lista de archivos que no están siendo monitoreados.

Flujo del estado de los archivos.

Como se explicó en la figura 2, los archivos pueden estar en tres estados: modificado, preparado (*staged*) y confirmado (*committed*). Para entender el flujo entre estos estados usaremos la figura 3.

Figura 3. El comando **`git status`** nos indicó que el archivo **`hola.txt`** no está siendo monitoreado (*untracked*). Todos los archivos van a estar siempre en el directorio de trabajo en alguno de los estados: *untracked*, *unmodified*, *modified* o *staged*.

Una vez creado el archivo **`hola.txt`** lo que sigue es decirle a Git que este archivo es parte del repositorio, para ello usamos el comando **`git add`** para darle seguimiento.

```
git add hola.txt
```

Con este comando Git pone al archivo **`hola.txt`** en el estado preparado (*staged*) para que posteriormente sea parte de la primera versión, figura 4.

Figura 4. El comando **`git add hola.txt`** prepara el archivo y lo pone en la zona de espera (*staged*).

Si ahora checamos el estado del proyecto obtenemos:

```
git status
On branch main

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)
        new file:   hola.txt
```

Observamos que Git identifica al archivo **`hola.txt`** como un nuevo archivo que se va a monitorear a partir de ahora. En este momento el

archivo está en estado preparado (*staged*). También Git indica que este archivo será tomado en cuenta cuando se haga una actualización del proyecto (`Changes to be committed`). Se observa que la lista de archivos en el área de espera (archivos preparados) están en color verde.

Para actualizar los cambios realizados a nuestro repositorio, que en este caso fue la creación del archivo **`hola.txt`**, se usa el siguiente comando:

```
git commit -m "Inicio del repositorio del curso"
[main (root-commit) 1d709c8] Inicio del repositorio del curso
 1 file changed, 1 insertion(+)
 create mode 100644 hola.txt
```

La opción **`-m`** permite poner entre comillas un mensaje corto acerca de lo que se realizó en esta versión del repositorio. Se recomienda siempre poner un mensaje para saber qué cambios se realizaron. Si solo se usa el comando **`git commit`** sin ningún mensaje, automáticamente se abrirá un editor de texto para teclear el mensaje (el editor de texto que se abre por omisión es **`vi`**). Este comando guarda localmente los cambios. En este momento el archivo ha sido confirmado (*committed*), se ha marcado este evento en la base de datos del repositorio y el archivo ya forma parte del mismo, figura 5. Es un evento histórico dentro del repositorio al cual se podrá regresar en un futuro.

Figura 5. El comando **`git commit -m "..."`** confirma los cambios y en este caso etiqueta al archivo **`hola.txt`** como *unmodified*, es decir es la última versión del mismo.

Si ahora revisamos el estado del proyecto obtenemos:

```
git status
On branch main
nothing to commit, working tree clean
```

Git nos dice que no hay cambios que realizar al proyecto por lo que todo está en orden.

Ahora vamos a modificar el archivo hola.txt agregando texto al final con el siguiente comando:

```
echo "Jubilemos la ortografía, terror del ser humano desde la cuna" >> hola.txt
```

Verificamos el contenido del archivo:

```
cat hola.txt
Simplifiquemos la gramática antes de que la gramática termine por simplificarnos a nosotros
Jubilemos la ortografía, terror del ser humano desde la cuna
```

Y checamos el estado del repositorio:

```
git status
On branch main
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   hola.txt

no changes added to commit (use "git add" and/or "git commit -a")
```

Observamos que ahora Git nos indica que el archivo **`hola.txt`** ha sido modificado. La figura 6 muestra el estado del archivo.

Figura 6. Cuando se modifica un archivo en el directorio de trabajo, pasa a estado *modified*. Para llevarlo al área de espera y posteriormente confirmar los cambios, se usan los comandos **`git add`** y **`git commit`** respectivamente. También es posible hacerlo en un solo paso con **`git commit -am "un mensaje"`**.

Para confirmar estos cambios en la base de datos del repositorio, debemos usar el comando **`git add hola.txt`**, seguido de **`git commit -m "un mensaje"`**. En este caso es posible realizar estas dos acciones en un solo paso con el siguiente comando:

```
git commit -am "Nueva versión del archivo hola.txt"
[main 4172f8f] Nueva versión del archivo hola.txt
 1 file changed, 1 insertion(+)
```

Finalmente, es posible revisar la bitácora de cambios de los archivo usando el comando **`git log`**:

```
git log
commit 4172f8f6f59b379614d83aa526cd5fcc9bc51ecf (HEAD -> main)
Author: luiggix <luiggix@gmail.com>
Date:   Fri Dec 22 22:50:54 2023 +0000

    Nueva versión del archivo hola.txt

commit 1d709c8c3018097d91293fff226ff42c27c6bd88
Author: luiggix <luiggix@gmail.com>
Date:   Fri Dec 22 01:53:00 2023 +0000

    Inicio del repositorio del curso
```

Observa lo siguiente:

- La bitácora registra todos las confirmaciones (**`git commit`**) que se hayan realizado hasta el momento.
- Cada confirmación:
    - se registra con un identificador de 40 caracteres que se construye usando el algoritmo SHA-1 con base en los cambios de los archivos del repositorio;
    - contiene el mensaje que se agrega al hacer **`git commit -m "mensaje"`**;
    - registra el autor de los cambios (usuario y correo electrónico) y la fecha de la confirmación;

- la última confirmación está marcada con `HEAD -> main` lo que indica que es el lugar donde se encuentra el puntero de la versión actual del repositorio y la rama a la que pertenece (main).

Esto lo podemos visualizar como se muestra en la figura 7.

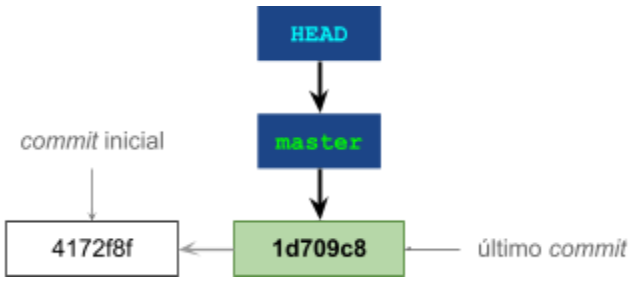


Figura 7. Confirmaciones del repositorio hasta ahora.

En resumen, para gestionar los cambios de los archivos de un repositorio, de manera local, se realizan los siguientes pasos:

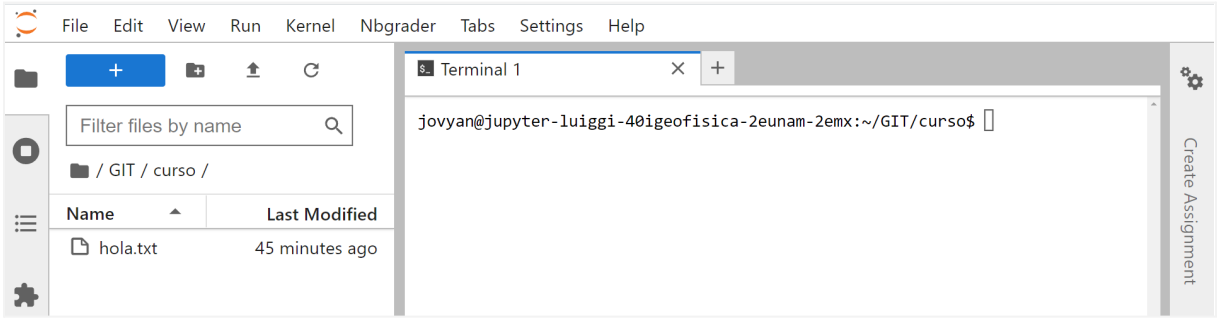
	Comando	Explicación
1	**`mkdir`** *dir_repo*	Crear un directorio en donde pondremos los archivos del repositorio. También se puede clonar el repositorio de un sitio remoto, esto se verá más adelante.
2	**`cd`** *dir_repo*	Cambiarse al directorio antes creado.
3	**`git init`**	Inicializar el repositorio. Esto se realiza una sola vez.
4	Crear o modificar los archivos que componen el repositorio. Esto se realiza en el directorio de trabajo.	
5	**`git add`** *lista_de_archivos*	Agregar la *lista_de_archivos* modificados al área de espera. Se pueden usar *wildcards* para mover varios archivos a la vez, por ejemplo: • **`git add .`** (agrega todos los archivos del directorio actual al área de espera).

		Usar **`git add --help`** para obtener ayuda con respecto a este comando.
6	**`git commit -m "mensaje"`**	Confirmar los cambios que están en el área de espera.
7	Repetir el flujo de trabajo a partir del paso 4 tantas veces como sea necesario.	

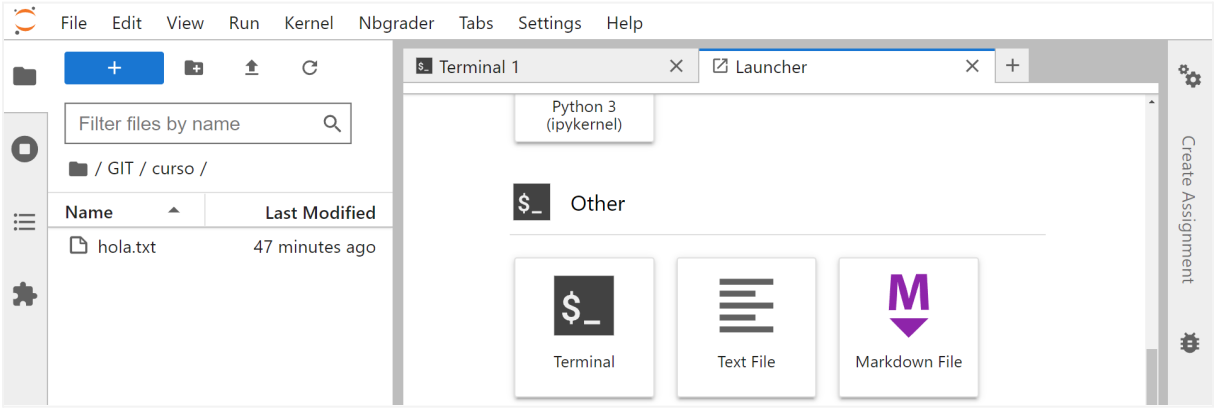
Ignorando archivos.

Algunas veces es necesario tener archivos dentro del directorio de trabajo pero que no sean monitoreados por Git. Por ejemplo, archivos ejecutables, imágenes o datos de gran tamaño. Si bien es posible no agregarlos al repositorio (es decir no usar **`git add`** con esos archivos), cada vez que veamos el estado del repositorio (**`git status`**) aparecerá esa lista de archivos *untracked* lo cual puede ser confuso. Veamos un ejemplo.
Agregaremos un nuevo archivo al repositorio, pero ahora usaremos el editor de Text File de Jupyter lab como sigue:

1. Haz clic en el ícono de folder en el menú vertical de la izquierda de Jupyter Lab. Asegúrate de estar en el directorio correcto del repositorio (en la imagen es /GIT/curso/).

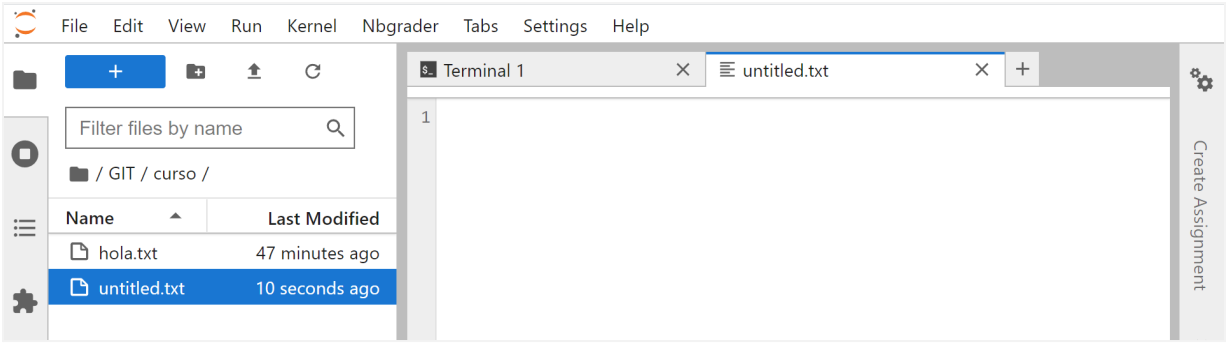


2. Haz clic en el signo + que se encuentra a la derecha de la pestaña **Terminal** para abrir un nuevo **Launcher**. En la pestaña de **Launcher** baja hasta al final y luego haz clic en **Text File**.



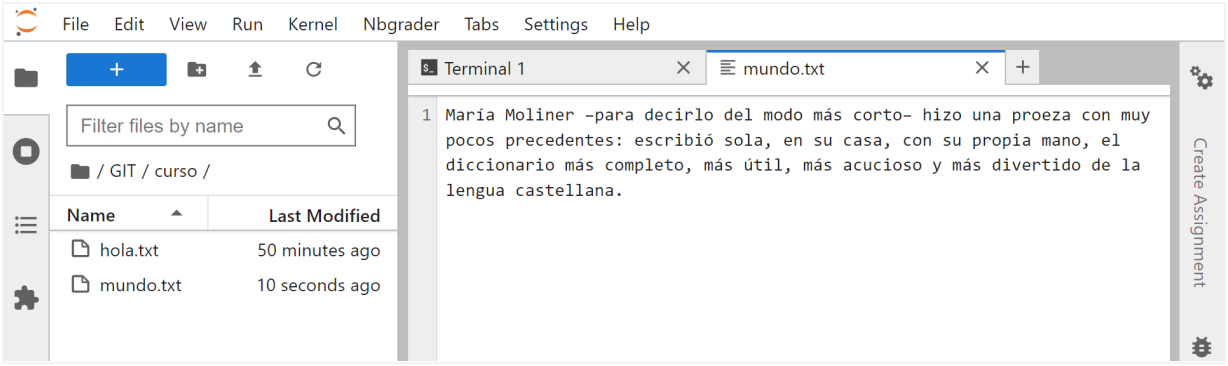
3. En el editor (**untitled.txt**) teclea el siguiente texto:

```
María Moliner –para decirlo del modo más corto– hizo una proeza con muy pocos precedentes: escribió sola, en su casa, con su propia mano, el diccionario más completo, más útil, más acucioso y más divertido de la lengua castellana.
```



4. Guarda el archivo tecleando **Ctrl+S** o usando el menú **File** > **Save Text**. Ingresa el nombre **mundo.txt** en el recuadro que aparece.

5. El resultado final es como se muestra en la siguiente figura.



Una vez creado el archivo **`mundo.txt`**, regresa a la Terminal y teclea el comando **`ls -la`** para ver la lista de archivos:

```
ls -la
total 8
drwxr-xr-x 4 jovyan users  97 Dec 22 23:34 .
drwxr-xr-x 3 jovyan users  41 Dec 22 22:43 ..
drwxr-xr-x 8 jovyan users 214 Dec 22 22:50 .git
-rw-r--r-- 1 jovyan users 156 Dec 22 22:44 hola.txt
drwxr-xr-x 2 jovyan users  42 Dec 22 23:34 .ipynb_checkpoints
-rw-r--r-- 1 jovyan users 242 Dec 22 23:34 mundo.txt
```

Observa que además de haberse agregado el archivo **`mundo.txt`**, también se agregó el directorio **`.ipynb_checkpoints/`**. Este directorio lo agrega Jupyter Lab automáticamente para guardar estados de los notebooks. Si checamos el estado del repositorio obtendremos lo siguiente:

```
git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .ipynb_checkpoints/
        mundo.txt

nothing added to commit but untracked files present (use "git add"
to track)
```

Observa que además del archivo **`mundo.txt`**, también Git detecta el directorio **`.ipynb_checkpoints/`**.

En este ejercicio agregaremos el archivo **mundo.txt** al repositorio e ignoraremos el directorio **.ipynb_checkpoints/**. Para ello necesitamos crear el archivo **.gitignore** en donde se pondrán los archivos que Git va a ignorar. Creamos el archivo como sigue:

```
echo -e "# Jupyter Notebook \n.ipynb_checkpoints/" > .gitignore
```

Para verificar el contenido del archivo hacemos lo siguiente:

```
cat .gitignore
# Jupyter Notebook
.ipynb_checkpoints/
```

La primera línea que comienza con # es un comentario. La segunda línea es el nombre del archivo o directorio que deseamos que Git ignore.

Si ahora verificamos el estado del repositorio obtendremos lo siguiente:

```
git status
On branch main
Untracked files:
  (use "git add <file>..." to include in what will be committed)
        .gitignore
        mundo.txt

nothing added to commit but untracked files present (use "git add"
to track)
```

Ahora Git ya no toma en cuenta el directorio **.ipynb_checkpoints/**. Es importante agregar al repositorio el archivo **.gitignore**. pues puede cambiar en el futuro. Entonces ahora hacemos lo siguiente:

```
git add .
git status
On branch main
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   .gitignore
```

```
        new file:   mundo.txt
```

El comando **`git add .`** agrega todos los archivos del repositorio que estén en el directorio donde se ejecuta el comando y de manera recursiva sobre cualquier subdirectorio. El segundo comando muestra el estado del repositorio.

Ahora confirmamos los cambios:

```
git commit -m "Agregué los archivos mundo.txt y .gitignore"
[main 6089315] Agregué los archivos mundo.txt y .gitignore
 2 files changed, 4 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 mundo.txt
```

Finalmente checamos la bitácora de cambios:

```
curso$ git log
commit 60893154da439b01e8f18b472fff1235104e9dd6 (HEAD -> main)
Author: luiggix <luiggix@gmail.com>
Date:   Sat Dec 23 01:36:52 2023 +0000

    Agregué los archivos mundo.txt y .gitignore

commit 4172f8f6f59b379614d83aa526cd5fcc9bc51ecf
Author: luiggix <luiggix@gmail.com>
Date:   Fri Dec 22 22:50:54 2023 +0000

    Nueva versión del archivo hola.txt

commit 1d709c8c3018097d91293fff226ff42c27c6bd88
Author: luiggix <luiggix@gmail.com>
Date:   Fri Dec 22 01:53:00 2023 +0000

    Inicio del repositorio del curso
```

Vemos que se tienen tres confirmaciones del repositorio, que gráficamente se ven como en la figura 8.

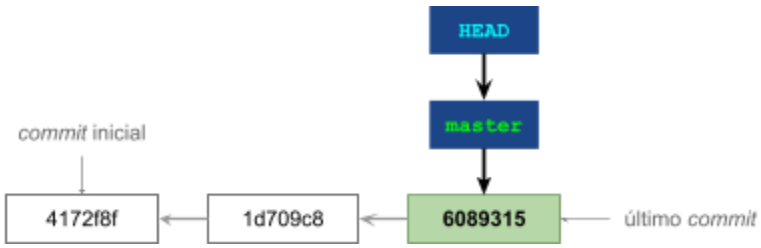


Figura 8. Confirmaciones del repositorio hasta ahora.

Regresando a confirmaciones anteriores.

En este punto tenemos tres estados del repositorio (tres confirmaciones). Podemos regresar a cualquier estado y ver cómo estaba el proyecto en ese momento. Para lograr esto necesitamos conocer el identificador de la confirmación (cadena de 40 caracteres) que nos proporciona **`git log`** o solo un extracto (7 caracteres) que obtenemos con **`git log --oneline`**:

```
git log --oneline
6089315 (HEAD, main) Agregué los archivos mundo.txt y .gitignore
4172f8f Nueva versión del archivo hola.txt
1d709c8 Inicio del repositorio del curso
```

Por ejemplo regresamos al primer *commit* usando **`git checkout`** y la cadena completa:

```
git checkout 1d709c8c3018097d91293fff226ff42c27c6bd88
Note: switching to '1d709c8c3018097d91293fff226ff42c27c6bd88'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by switching back to a branch.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -c with the switch command. Example:

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -
```

```
Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at 1d709c8 Inicio del repositorio del curso
```

La salida del comando incluye mucha información, la parte más importante es la última línea que nos dice dónde nos encontramos ahora.

Una vez que regresamos al commit inicial, veamos que contiene el repositorio:

```
ls -la
total 4
drwxr-xr-x 4 jovyan users  76 Dec 23 01:43 .
drwxr-xr-x 4 jovyan users  79 Dec 23 01:23 ..
drwxr-xr-x 8 jovyan users 214 Dec 23 01:43 .git
-rw-r--r-- 1 jovyan users  94 Dec 23 01:43 hola.txt
drwxr-xr-x 2 jovyan users  77 Dec 23 01:06 .ipynb_checkpoints

git log
commit 1d709c8c3018097d91293fff226ff42c27c6bd88 (HEAD)
Author: luiggix <luiggix@gmail.com>
Date:   Fri Dec 22 01:53:00 2023 +0000

    Inicio del repositorio del curso

cat hola.txt
Simplifiquemos la gramática antes de que la gramática termine por
simplificarnos a nosotros
```

Observamos que efectivamente regresamos a la primera confirmación.

Ahora regresemos al estado actual:

```
git checkout 6089315
Previous HEAD position was 1d709c8 Inicio del repositorio del curso
HEAD is now at 6089315 Agregué los archivos mundo.txt y .gitignore

ls -la
total 12
drwxr-xr-x 4 jovyan users 119 Dec 23 01:49 .
drwxr-xr-x 4 jovyan users  79 Dec 23 01:23 ..
drwxr-xr-x 8 jovyan users 214 Dec 23 01:49 .git
-rw-r--r-- 1 jovyan users  40 Dec 23 01:49 .gitignore
-rw-r--r-- 1 jovyan users 156 Dec 23 01:49 hola.txt
drwxr-xr-x 2 jovyan users  77 Dec 23 01:06 .ipynb_checkpoints
```

```
-rw-r--r-- 1 jovyan users 242 Dec 23 01:49 mundo.txt

cat hola.txt
Simplifiquemos la gramática antes de que la gramática termine por
simplificarnos a nosotros
Jubilemos la ortografía, terror del ser humano desde la cuna
```

Observa que usamos solo los primeros 7 caracteres del identificador de la confirmación.

Podemos seguri agregando y modificando archivos dentro del repositorio, es importante seguir los pasos que se marcan en resumen de la sección Flujo del estado de los archivos.

Para saber más sobre Git revisa el libro (Chacon, 2014).

GitHub

(https://github.com/)

GitHub es una plataforma de desarrollo colaborativo en web en donde se alojan y se da seguimiento a proyectos usando el software Git, con opciones gratuitas y de paga, administrado por Microsoft.

Creación de una cuenta en GitHub.

GitHub es uno de los servicios de alojamiento de proyectos más populares en la actualidad. Primero se debe abrir una cuenta en https://github.com, como se muestra en la figura 9.

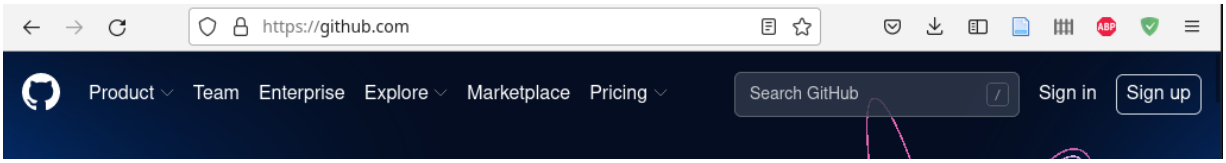


Figura 9. En el sitio de Github hacer clic en **Sign up** y seguir las instrucciones para crear tu cuenta. Asegúrate de usar el mismo usuario y correo electrónico que usaste al configurar Git localmente en la sección Configuración inicial. Una vez creada tu cuenta puedes acceder a ella haciendo clic en **Sign in** y proporcionando los datos correspondientes (user y password).

Creación de la llave pública y privada.

Una vez que se tenga la cuenta, procedemos a crear el canal de comunicación seguro entre la computadora local y nuestra cuenta en GitHub mediante el protocolo SSH. Este protocolo provee la seguridad necesaria para autenticarse en el servidor mediante una llave pública y una privada. La llave privada permanece sólo en el equipo del cliente, mientras que la pública se usa en el servidor remoto.

Primero abrimos una Terminal en Jupyter Lab, en el directorio raíz (para ir al directorio raíz teclea **`cd`** seguido de **`enter`** en la terminal).

Para generar las llaves, tecleamos el siguiente comando (teclear **enter** cuando solicite algún dato):

```
ssh-keygen -t ed25519
Generating public/private ed25519 key pair.
Enter file in which to save the key (/home/jovyan/.ssh/id_ed25519):
Created directory '/home/jovyan/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/jovyan/.ssh/id_ed25519
Your public key has been saved in /home/jovyan/.ssh/id_ed25519.pub
The key fingerprint is:
SHA256:zDqQmLBWdZ2bgjbvjDf4CU16jLnrGml0TOprhyfusdA
jovyan@jupyter-luiggix-40gmail-2ecom
The key's randomart image is:
+--[ED25519 256]--+
|     . .. .      |
|    . .  o       |
|.  . ..   o      |
| o.o+= + o       |
|..oo+oo.S        |
|. + o.Bo         |
| . E.=B+         |
|  o+==+*.        |
|  +**++o.        |
+----[SHA256]-----+
```

Si todo sale correctamente se generará la llave privada **id_ed25519** y la llave pública **id_ed25519.pub** en el directorio **.ssh** que se puede ver como sigue:

```
ls -l .ssh/
total 8
-rw------- 1 jovyan users 432 Dec 23 19:12 id_ed25519
-rw-r--r-- 1 jovyan users 118 Dec 23 19:12 id_ed25519.pub

cat .ssh/id_ed25519.pub
ssh-ed25519 AAAAC3Nza ... jovyan@jupyter ... 2ecom
```

En el recuadro anterior observamos que con el comando **ls -l .ssh/** se muestran los archivos que hay en el directorio **.ssh** y con el comando

`cat .ssh/id_ed25519.pub` se muestra el contenido de la llave pública (aquí solo se muestra un extracto).

Configuración de llaves SSH en GitHub.

Accede a tu cuenta en GitHub haciendo clic en Sign in, vease figura 9. Una vez dentro de tu cuenta, dirígete al menú del perfil personal y haz clic en *Settings*, luego ir a la sección de *SSH and GPG Keys*, y luego hacer clic en *New SSH Key*, véase figura 10.

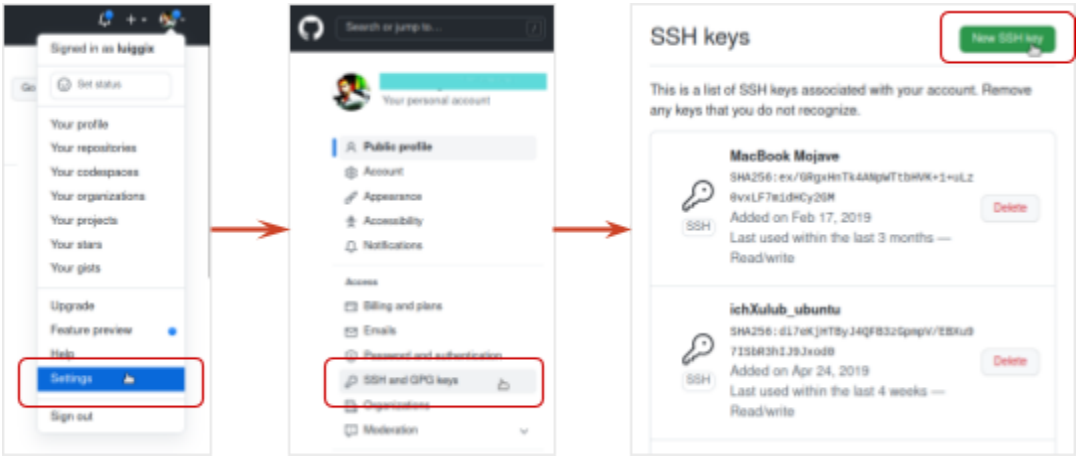


Figura 10. Menú del perfil personal en GitHub. Seleccionar *SSH and GPG Keys* en el menú de la izquierda. En esta sección deberá hacer clic en *New SSH Key*.

Una vez que hagas clic en el botón verde *New SSH Key* deberás realizar lo que se indica en la figura 11.

Figura 11. En esta sección agrega un título para la llave pública y luego copia el texto del archivo **`id_ed25519.pub`** en la sección Key. Finalmente haz clic en *Add SSH key*.

Para probar que todo se hizo correctamente, ve a la Terminal en Jupyter Lab y teclea el siguiente comando (teclear **yes** cuando pregunte si estás seguro) :

```
ssh -T git@github.com
The authenticity of host 'github.com (140.82.113.4)' can't be established.
ED25519 key fingerprint is SHA256:+DiY3wv ... dkr4UvCOqU.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519) to the list of known hosts.
Hi luiggix! You've successfully authenticated, but GitHub does not provide
shell access.
```

Si volvemos a teclear el mismo comando obtenemos lo siguiente:

```
ssh -T git@github.com
Hi luiggix! You've successfully authenticated, but GitHub does not
provide shell access.
```

Creación de un repositorio en GitHub.

Creamos un nuevo repositorio en GitHub haciendo clic en el menú del perfil personal, luego en *Your repositories* y después en *New*, véase la figura 12.

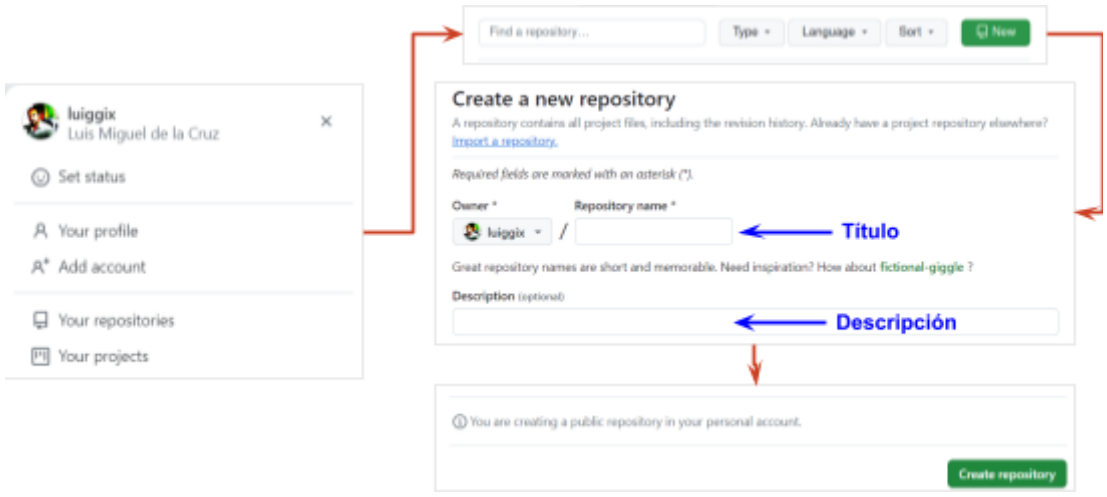
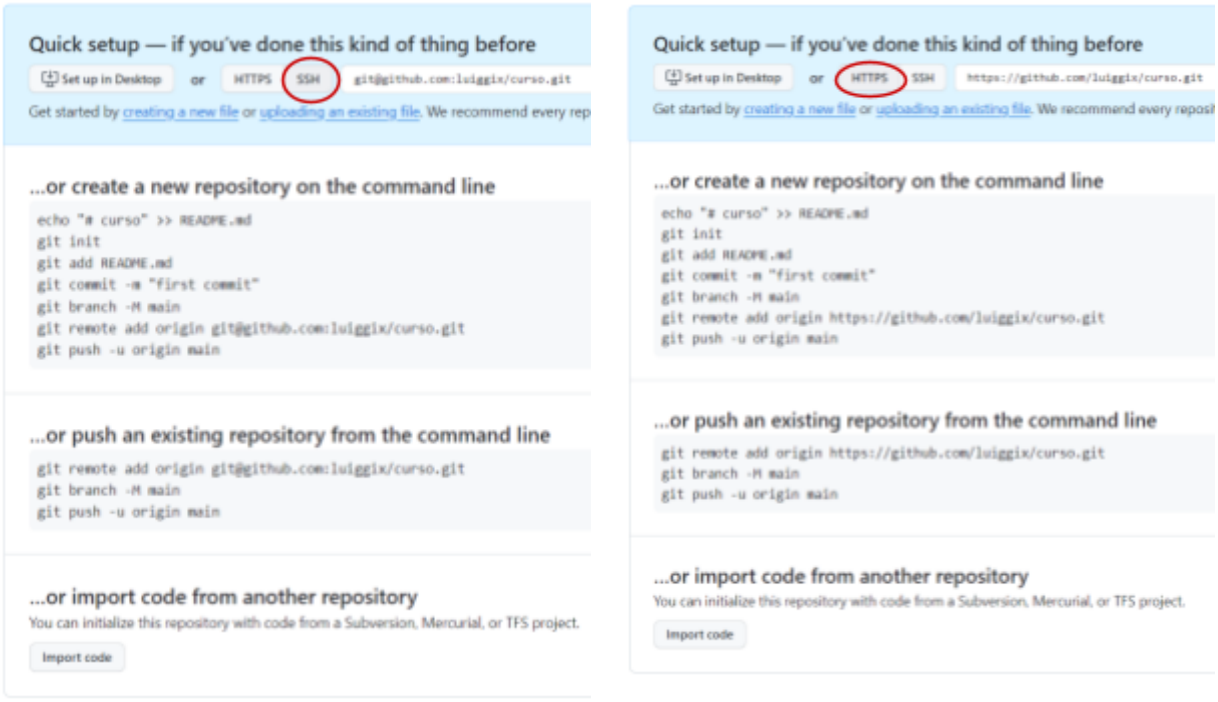


Figura 12. En el formato de creación de un nuevo repositorio (*Create a new repository*) deberás al menos poner un título. Se recomienda también agregar una breve descripción. Por ahora todo lo demás se deja como está y se hace clic en *Create repository* (botón verde al final).

Una vez que se hace clic en *Create repository* se obtiene lo siguiente:



(a) (b)

Figura 13. Instrucciones para sincronizar el repositorio en GitHub con un repositorio local: (a) usando el protocolo SSH, (b) usando el protocolo HTTPS.

Sincronización del repositorio local con GitHub.

Vamos a usar el protocolo SSH para sincronizar el repositorio en GitHub con el repositorio local. En la Terminal de Jupyter Lab, en el directorio de nuestro repositorio local, usamos el comando **`git remote add`** como sigue.

```
git remote add origin git@github.com:luiggix/curso.git
git remote -v
origin  git@github.com:luiggix/curso.git (fetch)
origin  git@github.com:luiggix/curso.git (push)
```

Con el comando **`git remote add origin git@github.com:luiggix/curso.git`** estamos agregando la dirección donde está el repositorio remoto y un

alias del mismo (**origin**). Con el comando **git remote -v** vemos los repositorios remotos que se han definido. Si hubo algún error es posible eliminar esta definición con el comando **git remote rm origin** y volver a definirlo como se mostró antes.

En algunos casos, Git nombra a la rama principal como **master**, pero GitHub usa el nombre **main**. Para que todo sea compatible, es conveniente cambiar el nombre de la rama principal de nuestro repositorio local a **main** con el siguiente comando:

```
git branch -M main
git branch -a
* main
```

El primer comando cambia el nombre de la rama principal y el segundo muestra las ramas del repositorio, que en este caso solo es la principal y ahora se llama **main**.

Ahora, sincronizamos todos los archivos que ya tenemos en el repositorio local con el repositorio remoto:

```
git push -u origin main
Enumerating objects: 10, done.
Counting objects: 100% (10/10), done.
Delta compression using up to 32 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (10/10), 1.06 KiB | 1.06 MiB/s, done.
Total 10 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), done.
To github.com:luiggix/curso.git
 * [new branch]      main -> main
Branch 'main' set up to track remote branch 'main' from 'origin'.
```

Si miramos ahora a nuestro repositorio curso en GitHub veremos que los archivos ya están en el repositorio remoto, véase la figura 14.

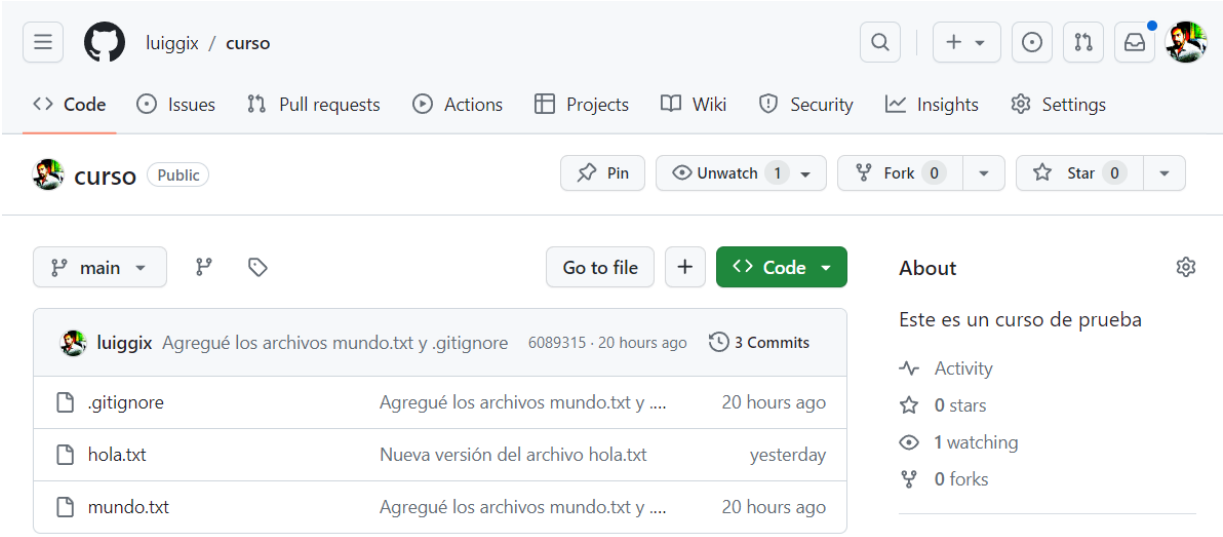
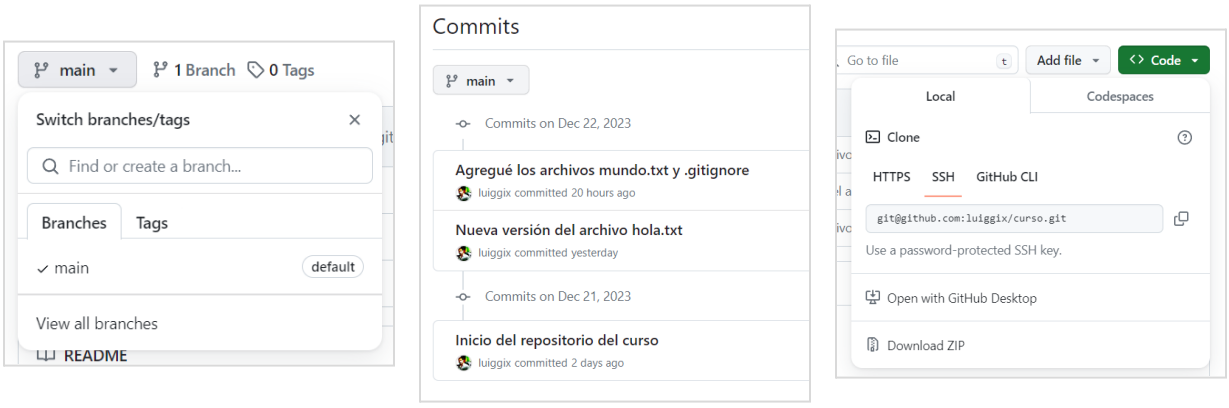


Figura 14. Archivos en GitHub en el repositorio curso.

Observa que se tienen todos los archivos del repositorio local con todos los cambios realizados y las confirmaciones (**3 Commits**) que se hicieron. Podemos ver las ramas que contiene el repositorio en el menú **main**, también se puede ver información de cómo bajar el repositorio (en otras computadoras o por otros usuarios) en el menú **<> Code** y también se puede ver información de las confirmaciones, véase figura 15.



(a) (b) (c)

Figura 15. (a) Menú main en donde se pueden ver las ramas y etiquetas del repositorio (en este ejemplo solo se tiene la rama main), (b) confirmaciones del repositorio, (c) información de cómo obtener el repositorio a través de diferentes protocolos e incluso se puede bajar comprimido en formato ZIP.

Clonación de un repositorio de GitHub.

Para clonar un repositorio de desde GitHub es necesario conocer su dirección. Por ejemplo vamos a clonar el repositorio **macti.git** del usuario **repomatic** en GitHub usando el siguiente comando:

```
git clone https://github.com/repomacti/macti.git
Cloning into 'macti'...
remote: Enumerating objects: 32, done.
remote: Counting objects: 100% (32/32), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 32 (delta 8), reused 26 (delta 5), pack-reused 0
Receiving objects: 100% (32/32), 131.26 KiB | 973.00 KiB/s, done.
Resolving deltas: 100% (8/8), done.
```

Como el repositorio es público no hay ningún problema para obtenerlo. En mi computadora local debo tener ahora los archivos de ese repositorio. Para verificarlo listo los archivos desde donde ejecuté el comando de clonación:

```
ls -l
total 0
drwxr-xr-x 3 jovyan users 89 Dec 24 00:48 curso
drwxr-xr-x 4 jovyan users 90 Dec 24 00:41 macti
```

Observamos que tenemos el directorio **macti** que corresponde al repositorio **repomacti/macti.git** que corresponde en GitHub. Si nos cambiamos al directorio macti, listamos los archivos, checamos el estado del repositorio y su bitácora obtenemos lo siguiente:

```
cd macti/
ls -l
total 4
drwxr-xr-x 3 jovyan users  39 Dec 24 00:41 notebooks
-rw-r--r-- 1 jovyan users 310 Dec 24 00:41 README.md

git status
On branch main
Your branch is up to date with 'origin/main'.

nothing to commit, working tree clean
```

```
git log
commit 034117ad867cc024882d6451bf312b43f167a809 (HEAD -> main, origin/zeus, origin/main, origin/HEAD)
Author: repomacti <macti.global@gmail.com>
Date:   Wed Nov 8 14:29:54 2023 -0600

    del ipynb_checkpoints y __pycache__

commit 0b2ab4fb5fcc1e78f89a2dce5489934e6107f7f5
Author: repomacti <macti.global@gmail.com>
Date:   Wed Nov 8 14:27:52 2023 -0600

    gitignore

commit 37b268104b06d72efad3cd02659a183104f0608d
Author: repomacti <macti.global@gmail.com>
Date:   Wed Nov 8 14:22:06 2023 -0600

    Algebra Lineal 01

commit d4b0a4fa25e54f26f7609bbf472a7015e4319d0e
Author: Luis Miguel de la Cruz <luiggix@gmail.com>
Date:   Wed Nov 8 14:08:13 2023 -0600

    Create README.md

    Nuevo readme

commit 343f11bd4b0c3ae2d591cb37c9714b3392cb5ac9
Author: luiggix <luiggix@gmail.com>
Date:   Wed Nov 8 13:39:21 2023 -0600

    Inicio de macti
```

Observa que este repositorio tiene varias confirmaciones y varios usuarios han contribuido en su desarrollo.

Se pueden agregar y modificar archivos en este repositorio, pero no será posible subir los cambios hasta que el administrador del repositorio (el usuario **repomacti**) te agregue como colaborador en este repositorio en GitHub.

Si posteriomente deseamos obtener nuevas versiones de este repositorio, (que pueden ser actualizaciones de otros usuarios), podemos hacer uso del comando git pull:

```
git pull
Already up to date.
```

En este caso, no hay nuevas actualizaciones.

Resumen de comandos.

En resumen, para usar Git y GitHub para gestionar los archivos de un proyecto/repositorio, los comandos básicos son los siguientes:

	Comando	Explicación
	Creación de un repositorio local.	
1	**`mkdir`** *dir_repo*	Crear un directorio en donde estarán los archivos del repositorio.
2	**`cd`** *dir_repo*	Cambiarse al directorio antes creado.
3	**`git init`**	Inicializar el repositorio. Esto se realiza una sola vez.
4	Crear y modificar los archivos del repositorio.	
5	**`git add`** *lista de archivos*	Preparar los archivos modificados
6	**`git commit -m "..."`**	Confirmar los cambios

	Sincronizar un repositorio local con el remoto en GitHub.	
1	**`cd`** *dir_repo*	Cambiarse al directorio del repositorio.
2	**`git remote add origin`** *remote*	Agregar la dirección del repositorio remoto. En este caso *remote* puede ser en formato HTTPS o en formato SSH. Solo se hace una vez.
3	**`git push -u origin main`**	Subir los archivos locales al repositorio remoto.
	Clonar un repositorio de GitHub	
1	**`git clone`** *dir_repo_github*	Clonar el repositorio de la dirección *dir_repo_github*. Puede ser usando el protocolo HTTPS o el SSH. La dirección del repositorio se obtiene del sitio en GitHub.
2	**`cd`** *dir_repo*	Cambiarse al directorio del repositorio recién clonado y comenzar a trabajar en él.

Crear o modificar los archivos que componen el repositorio.		
1	**`cd`** *dir_repo*	Cambiarse al directorio del repositorio.
2	**`git pull`**	Obtener la última versión del repositorio. Al iniciar el día, este debe ser el primer comando que se realice.
3	Crear y modificar los archivos del repositorio.	
4	**`git add`** *lista_de_archivos*	Agregar la *lista_de_archivos* modificados al área de espera.
5	**`git commit -m "mensaje"`**	Confirmar los cambios que están en el área de espera.
6	**`git push -u origin main`**	Enviar los cambios al repositorio remoto en GitHub. Al finalizar el día de trabajo, este debe ser el último comando que se realice.

Referencias.

Chacon, Scott, and Straub, Ben. *Pro Git*. Apress, 2014. Disponible en línea https://git-scm.com/book/en/v2.

Git, git-scm, 2023. [En línea]. Disponible en https://git-scm.com/. [Consultado en diciembre 21, 2023].