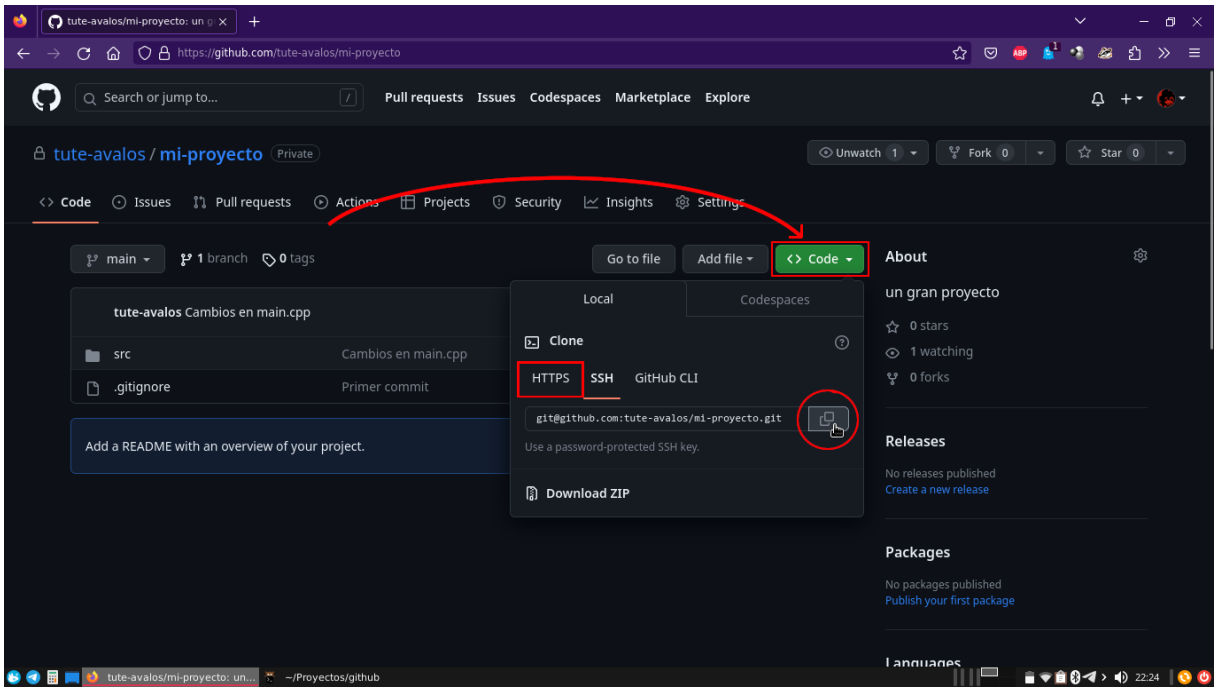
Una parte importante de un sistema de control de versiones es el trabajo en equipo. Para ello se debe organizar para trabajar en el mismo proyecto minimizando los posibles conflictos.

1. Clonar un repositorio

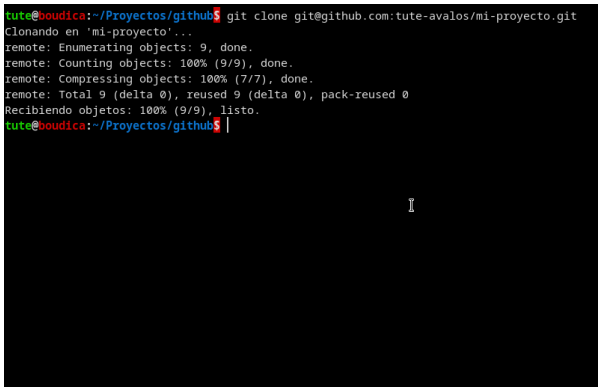
Antes de empezar, será necesario obtener el repositorio ya creado y publicado en algún servidor. Primero, debemos tener acceso al repositorio para poder copiar el link correspondiente. Habitualmente tiene 2 opciones: utilizar una **clave SSH** o utilizar el protocolo `HTTPS`. Por el momento se utilizará `HTTPS` y se copiará el link correspondiente.



Luego desde la terminal en la carpeta donde usualmente se encuentran nuestros proyectos, se utiliza el sub-comando `clone` con el link correspondiente.

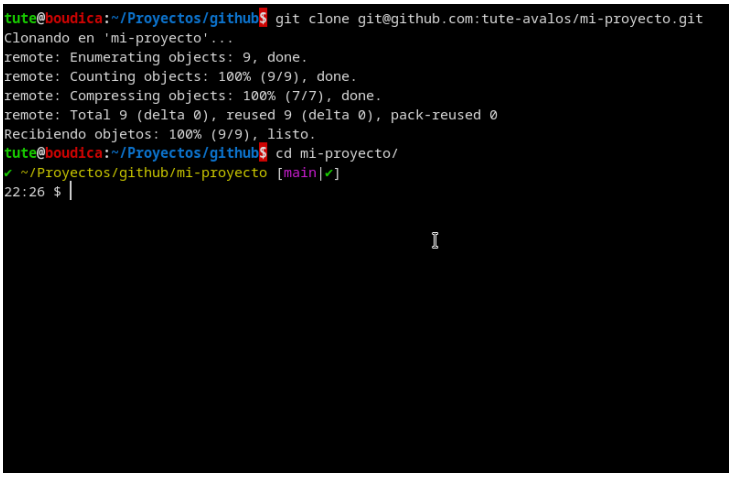
```
$ git clone <url> (carpeta-destigno)
```

Podemos especificar o no de forma opcional como se llamará la carpeta destino donde estarán los archivos que componen el proyecto.



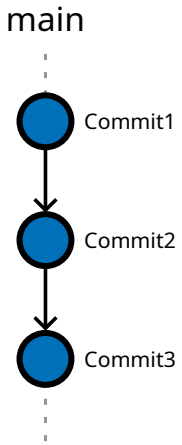
Luego con el comando `cd` (*change directory*) entramos a la carpeta del repositorio clonado.

```
$ cd <carpeta-del-proyecto>
```



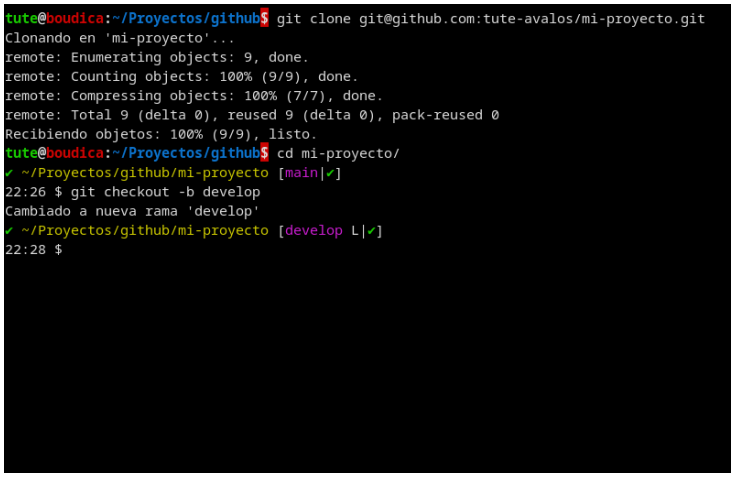
2. Crear una rama

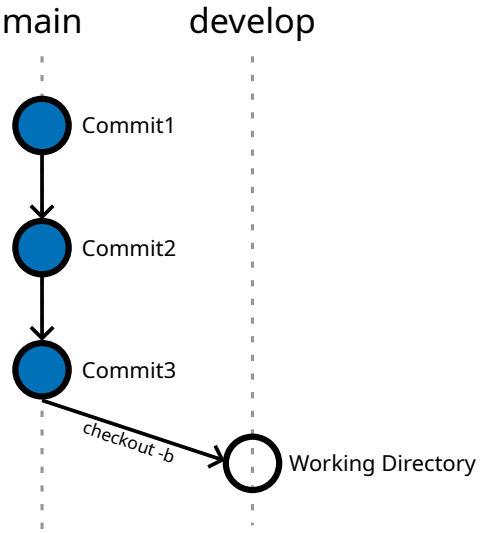
Hasta el momento, se ha trabajado en una única rama (`main`), la cual contiene los diversos *commits* que se fueron realizando. En todo caso, esto no es lo recomendable, ya que habitualmente en ésta rama se espera encontrar las versiones definitivas o estables del software en cuestión. Es por ello que se suelen utilizar otras ramas para el desarrollo, dejando la rama `main` para el propósito ya mencionado.



Para poder crear una nueva rama basada en el último *commit* realizado se utilizará el sub-comando `checkout` en combinación con el parámetro `-b`:

```
$ git checkout -b <nueva-rama>
```





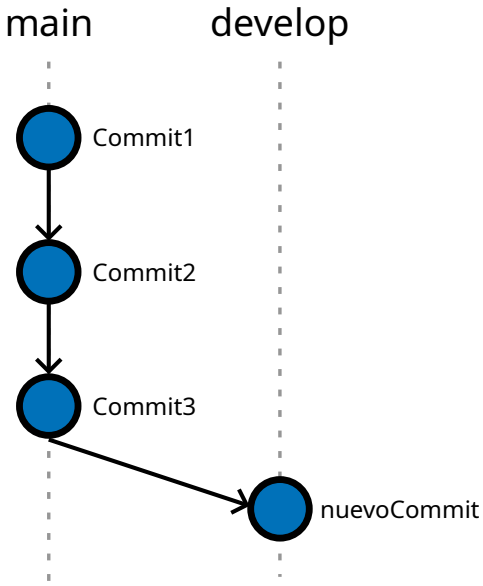
Es importante notar que lo único que cambio es que estamos en otra rama en nuestro *espacio de trabajo*. Es decir que no hubo un nuevo *commit* todavía. Para ello, haremos un cambio en algún archivo del proyecto y luego lo confirmaremos, generando una *diferencia real* entre una rama y la otra.

```
Clonando en 'mi-proyecto'...
remote: Enumerating objects: 9, done.
remote: Counting objects: 100% (9/9), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 9 (delta 0), reused 9 (delta 0), pack-reused 0
Recibiendo objetos: 100% (9/9), listo.
tute@boudica:~/Proyectos/github$ cd mi-proyecto/
✔ ~/Proyectos/github/mi-proyecto [main|✔]
22:26 $ git checkout -b develop
Cambiado a nueva rama 'develop'
✔ ~/Proyectos/github/mi-proyecto [develop L|✔]
22:28 $ vim src/main.cpp
✔ ~/Proyectos/github/mi-proyecto [develop L|+ 1]
22:30 $ git status -u
En la rama develop
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo
)
        modificados:     src/main.cpp

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
✔ ~/Proyectos/github/mi-proyecto [develop L|+ 1]
23:22 $
```

Una vez hechos los cambios, los confirmaremos con el sub-comando `commit`:

```
$ git commit -am <mensaje del nuevo commit>
```

```
remote: Total 9 (delta 0), reused 9 (delta 0), pack-reused 0
Recibiendo objetos: 100% (9/9), listo.
tute@boudica:~/Proyectos/github$ cd mi-proyecto/
✔ ~/Proyectos/github/mi-proyecto [main|✔]
22:26 $ git checkout -b develop
Cambiado a nueva rama 'develop'
✔ ~/Proyectos/github/mi-proyecto [develop L|✔]
22:28 $ vim src/main.cpp
✔ ~/Proyectos/github/mi-proyecto [develop L|+ 1]
22:30 $ git status -u
En la rama develop
Cambios no rastreados para el commit:
  (usa "git add <archivo>..." para actualizar lo que será confirmado)
  (usa "git restore <archivo>..." para descartar los cambios en el directorio de trabajo
)
        modificados:     src/main.cpp

sin cambios agregados al commit (usa "git add" y/o "git commit -a")
✔ ~/Proyectos/github/mi-proyecto [develop L|+ 1]
23:22 $ git commit -am "Mensaje agregado al main"
[develop 1e0d404] Mensaje agregado al main
 1 file changed, 1 insertion(+)
✔ ~/Proyectos/github/mi-proyecto [develop L|✔]
23:31 $
```



2.1. Cambiar de rama (ya existente)

Podemos volver a la rama `main` en cualquier momento utilizando el mismo sub-comando `checkout` sin el parámetro `-b`:

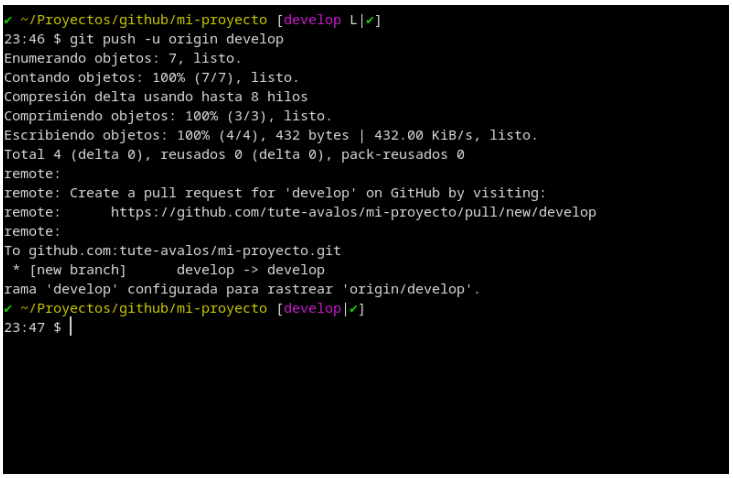
```
$ git checkout <rama-existente>
```

2.2. Publicar la rama nueva

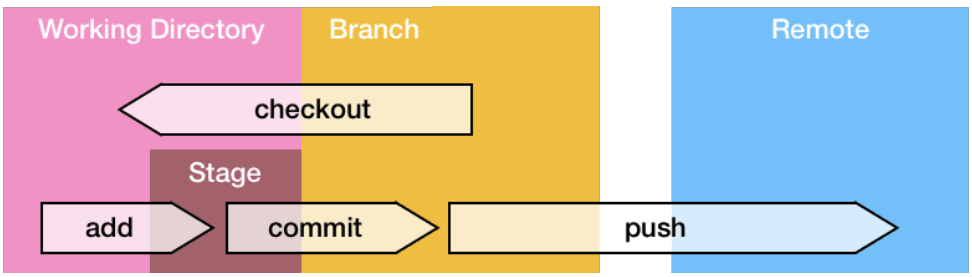
Nótese que todo cambio que se haga en esta nueva rama `develop` no se verá reflejada en el servidor de GitHub, ya que es una **rama local** en nuestra computadora.

Para que los cambios, si así lo deseamos, se vean reflejados en el servidor, debemos *publicar* la rama, de la misma manera que se hizo con `main`, utilizando el sub-comando `push`.

```
$ git push -u origin <rama-local>
```



Hasta aquí podemos decir que con `checkout` podemos crear o cambiar la rama en la que nos encontramos y luego seguir trabajando como se indicó anteriormente con el flujo `add`→`commit`→`push`.



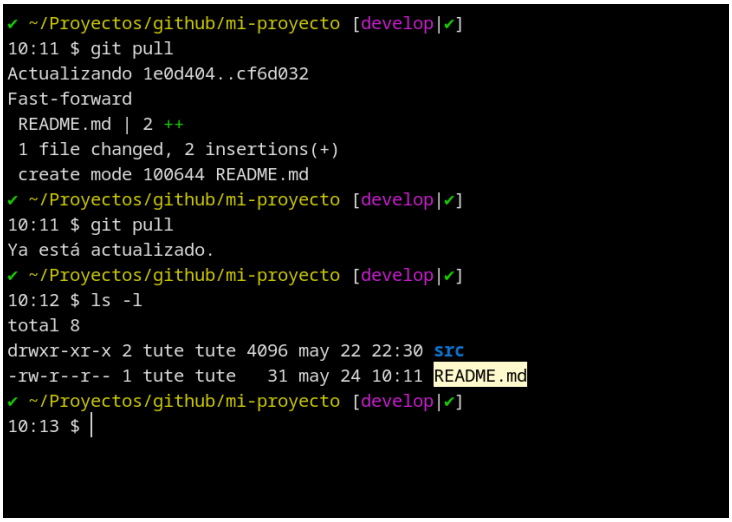
3. Obtener cambios hechos en el servidor

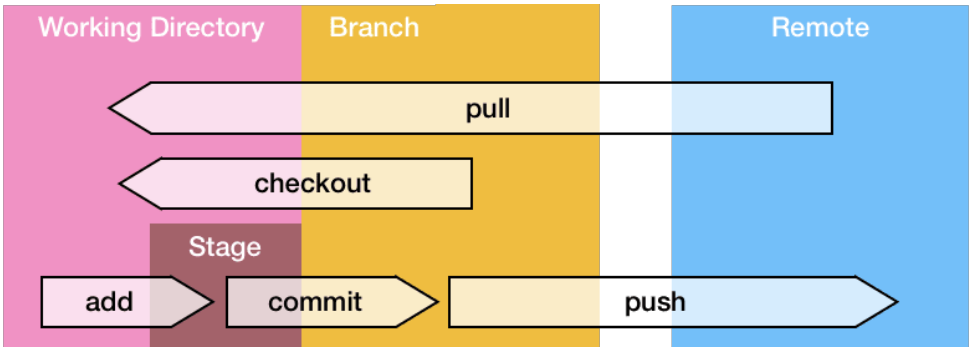
Si hubo cambios en el servidor hay dos formas de obtenerlos. La más habitual es utilizando el sub-comando `pull` que toma los cambios del servidor y los lleva hasta nuestro

espacio de trabajo (*Working Directory*). Esto nos traerá los cambios hechos en la rama donde estemos posicionados.

```
$ git pull
```

En el caso de que hayan cambios (sin conflictos), entonces se cargaran directamente en nuestro espacio de trabajo y asentadas en la rama correspondiente. En caso de que no haya cambios (el *commit* en el servidor es el mismo que el local), entonces nada se hará.





La otra manera de verificar que hay cambios en el repositorio remoto es con el subcomando `fetch`, el cual trae los cambios solo hasta la rama almacenada y no al espacio de trabajo. Aquí optaremos por utilizar siempre `pull`.

4. Unir/Fucionar una rama con otra

En este punto la rama `develop` tiene 2 *commits* de diferencia con respecto a la rama `main`. Si los cambios están listos para *producción*, es decir, una versión estable, sería deseable

traer esos cambios a la rama principal.

Para lograr este objetivo, debemos seguir los siguientes pasos:

1. Posicionarnos en la rama a la cual que deseamos traer los cambios.
2. Hacer un **`pull`** para traer cambios que hayan surgido desde el repositorio remoto (en caso de que haya).
3. Utilizar el sub-comando **`merge`** indicando de cual rama traeremos los cambios.
4. Finalmente, subir la rama *fusionada* al repositorio remoto.

En la linea de comandos ser vería de la siguiente manera:

```
$ git checkout <rama-destino>
$ git pull
$ git merge <rama-donde-estan-los-cambios>
$ git push
```

```
✔ ~/Proyectos/github/mi-proyecto [develop|✔]
10:28 $ git checkout main
Cambiado a rama 'main'
Tu rama está actualizada con 'origin/main'.
✔ ~/Proyectos/github/mi-proyecto [main|✔]
10:29 $ git merge develop
Actualizando 7b8ff94..cf6d032
Fast-forward
 README.md    | 2 ++
 src/main.cpp | 1 +
 2 files changed, 3 insertions(+)
 create mode 100644 README.md
✔ ~/Proyectos/github/mi-proyecto [main ↑·2|✔]
10:41 $
```

```
✔ ~/Proyectos/github/mi-proyecto [develop|✔]
10:28 $ git checkout main
Cambiado a rama 'main'
Tu rama está actualizada con 'origin/main'.
✔ ~/Proyectos/github/mi-proyecto [main|✔]
10:29 $ git merge develop
Actualizando 7b8ff94..cf6d032
Fast-forward
 README.md    | 2 ++
 src/main.cpp | 1 +
 2 files changed, 3 insertions(+)
 create mode 100644 README.md
✔ ~/Proyectos/github/mi-proyecto [main ↑·2|✔]
10:41 $ git push
Total 0 (delta 0), reusados 0 (delta 0), pack-reusados 0
To github.com:tute-avalos/mi-proyecto.git
   7b8ff94..cf6d032  main -> main
✔ ~/Proyectos/github/mi-proyecto [main|✔]
10:42 $
```