# Tutorial básico de GIT – Parte 2

**Introducción**

En este tercer post vamos a trabajar la comparación con un repositorio remoto e introduciremos el concepto merge. Como servicio de repositorio Git remoto usaremos GitLab por algunas razones. Principalmente porqué es un servicio liviano, sin añadidos, simple, con gestión de reportes de errores, wiki y que nos permite realizar repositorios privados. Es simple (en comparación con otros servicios mas potentes como GitHub) y nos permite realizar repositorios privados, es decir perfecto para empezar a hacer pruebas y que estas no sean visible a toda la comunidad (pues es innecesario)

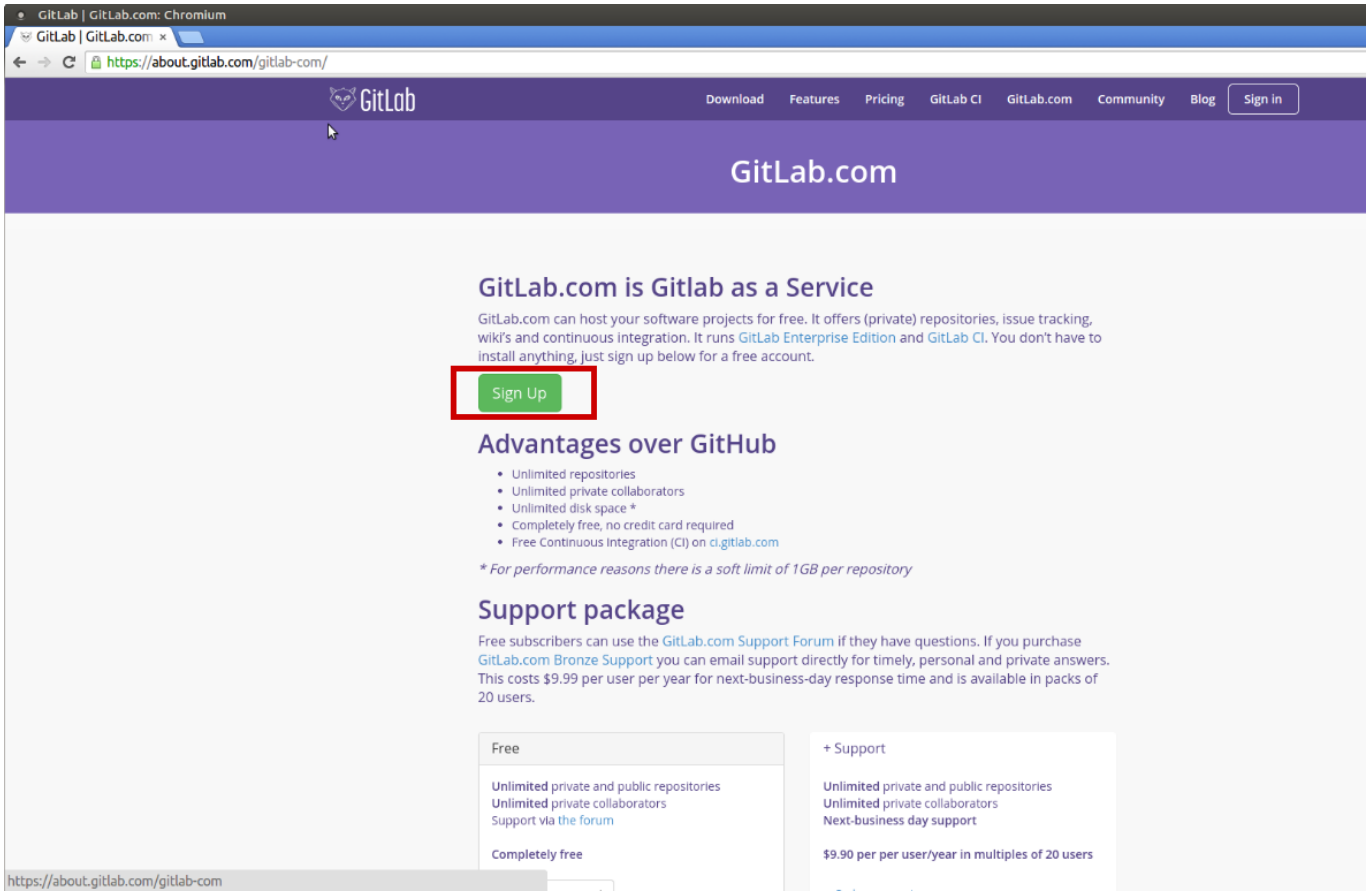
Vamos a tratar:

- Compartir un repositorio
  - Crear un repositorio remoto en GitLab
  - Hacer push a un repositorio remoto
  - Clonar un repositorio remoto
  - Realizar un push des de un repositorio clonado
  - Hacer pull des de un repositorio
- Merge de histórico
  - Hacer merge de un histórico
  - Resolver un conflicto

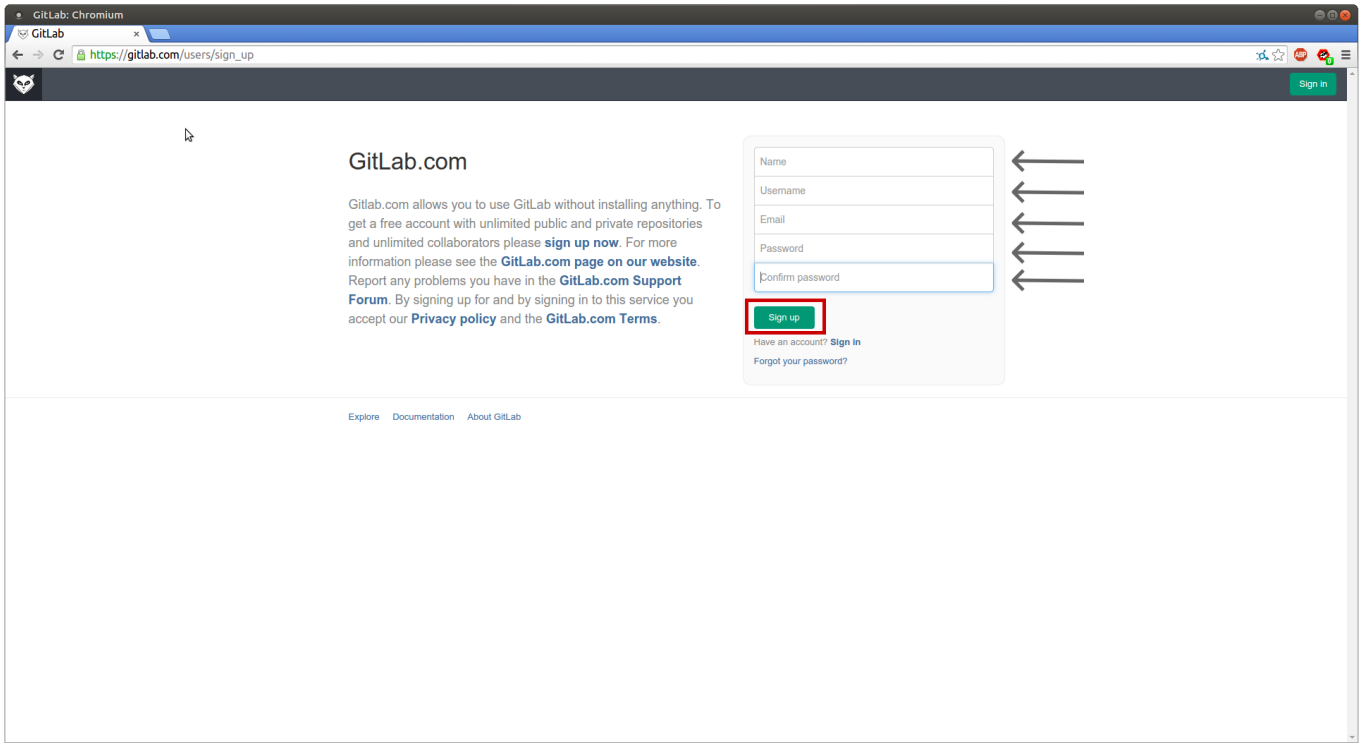
## Compartir un repositorio

### *Crear un repositorio remoto en GitLab*

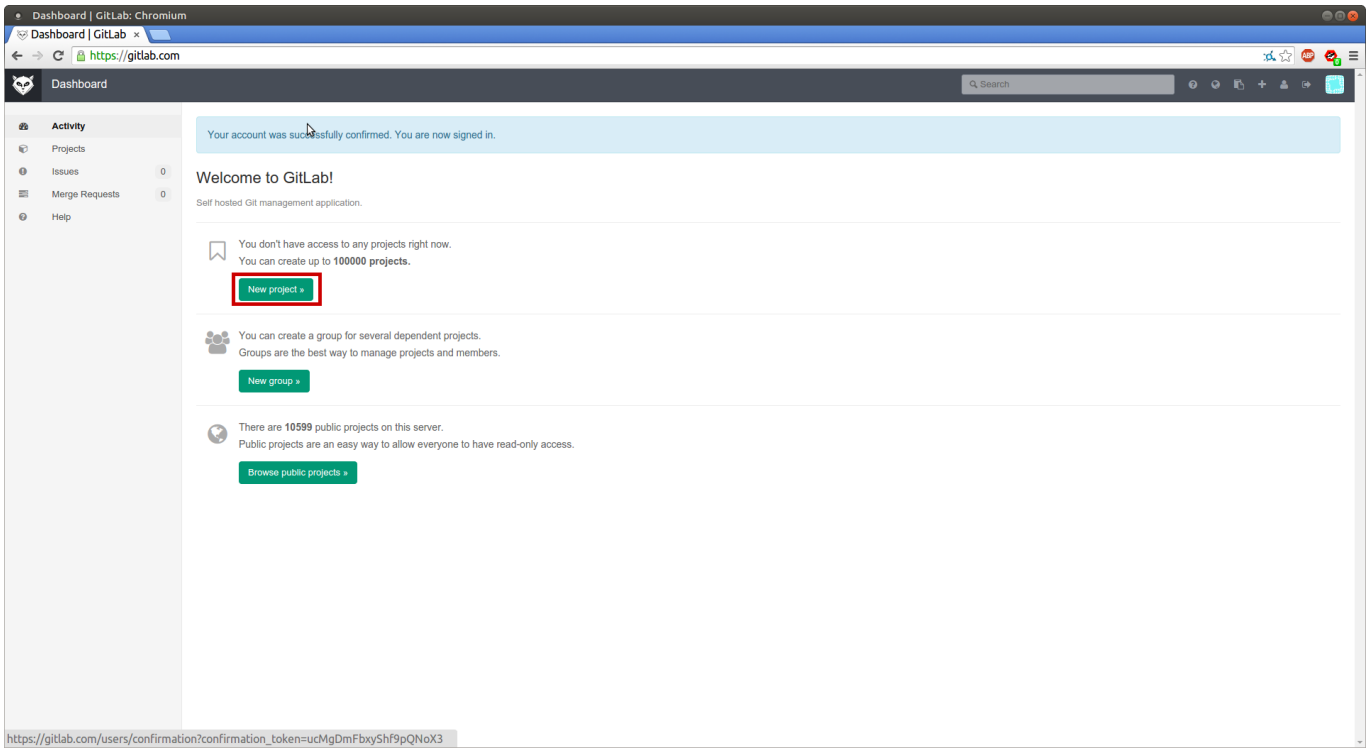
Como hemos comentado antes crearemos el repositorio en GitLab. Primera mente accederemos a la web del servicio y crearemos un usuario desde la siguiente url: https://about.gitlab.com/gitlab-com/



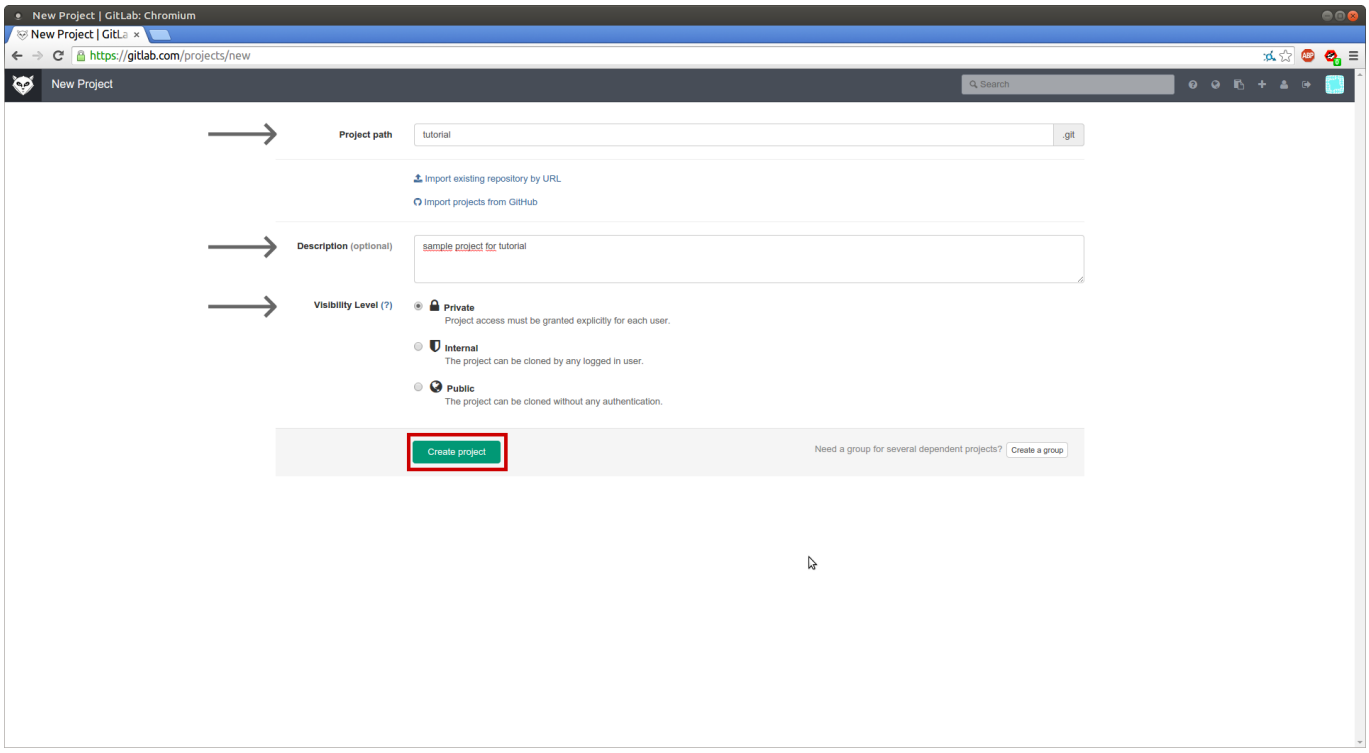

Rellenamos los campos necesarios para crear nuestra cuenta:



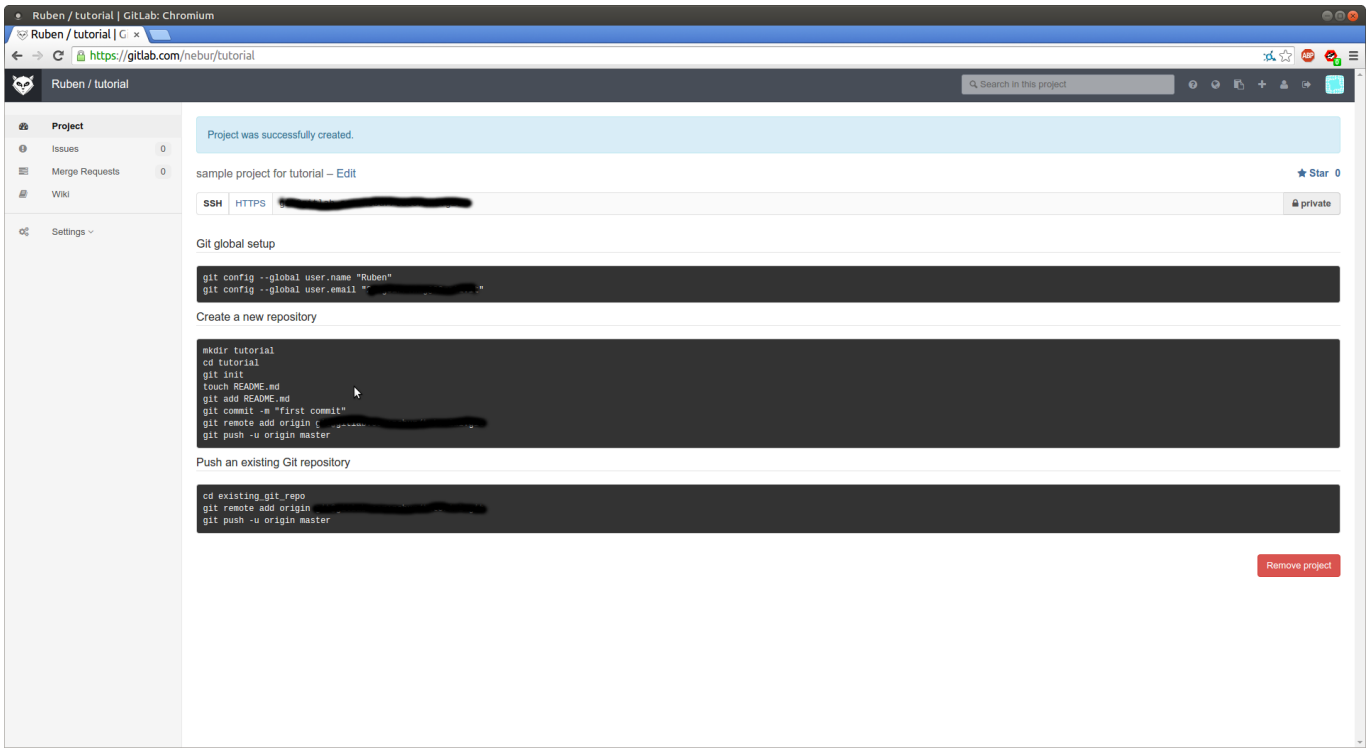

Una vez creada la cuenta entraremos con nuestro usuario y crearemos un proyecto nuevo. El proyecto será privado, puesto que no tienen ningún interés para la comunidad un repositorio donde se hacen pruebas:




Rellenamos los campos necesarios para crear nuestra nuestro repositorio como se muestra a continuación:




Finalmente podemos observar que nuestro repositorio se ha creado correctamente:




## *Hacer push a un repositorio remoto*

Haremos un push del repositorio local «tutorial» que hemos

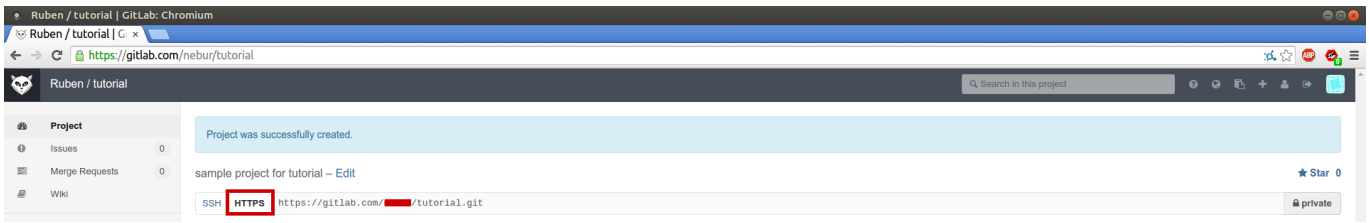
creado anteriormente en la primera parte del tutorial básico de Git.

Podemos utilizar un alias o apodo para un repositorio remoto. Esto es útil ya que no necesitamos recordar la larga dirección del repositorio remoto cada vez que tenemos la intención de hacer un push. En este tutorial, vamos a registrar un nombre de repositorio remoto como «origin».

Para añadir un repositorio remoto, utilizaremos el comando «remote». <name> se utiliza como un alias de un repositorio remoto, seguido de <url> con la URL del repositorio remoto.

```
$ git remote add
```

Ejecutamos el comando utilizando la url del repositorio repoto que hemos creados anteriormente. El nuevo repositorio remoto tendrá el alias «origin»




```
$ git remote add origin https://gitlab.com/[user_name]/tutorial.git
```

El repositorio remoto llamado «origin» se utiliza por defecto si se omite el nombre remoto al hacer push/pull. Esto se debe a «origin» es comúnmente usado como un nombre remoto por convención.

Para hacer un push de nuestros cambios al repositorio remoto, utilizaremos el comando «push». Asignaremos la dirección en <repository> y el nombre del branch en <refspec>, al cual queremos hacer el push. Hablaremos de lo branchs en Git en el tutorial avanzado de Git más adelante.

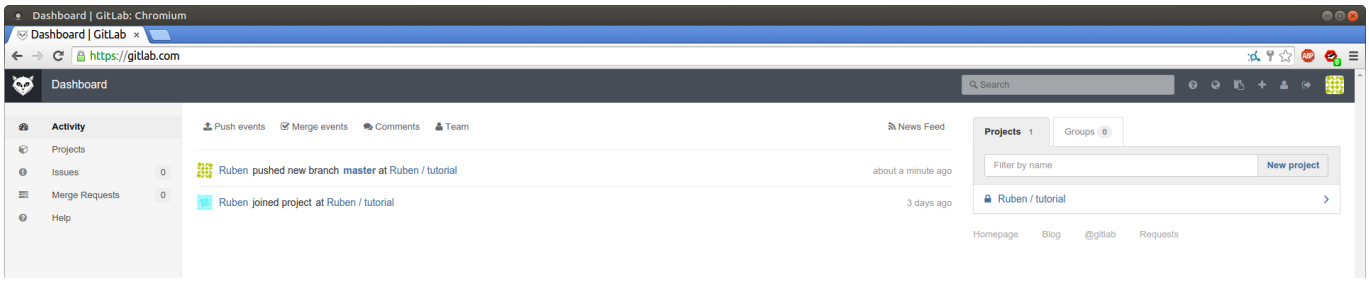
```
$ git push  ...
```

Ejecutaremos el siguiente comando para insertar un commit al repositorio remoto «origin». Si especifica la opción -u al ejecutar el comando, se puede omitir el nombre del branch la próxima vez que se hagamos un push al repositorio remoto. Cuando empujas a un remoto vacante sin embargo, se debe especificar el repositorio remoto y nombre de la rama.

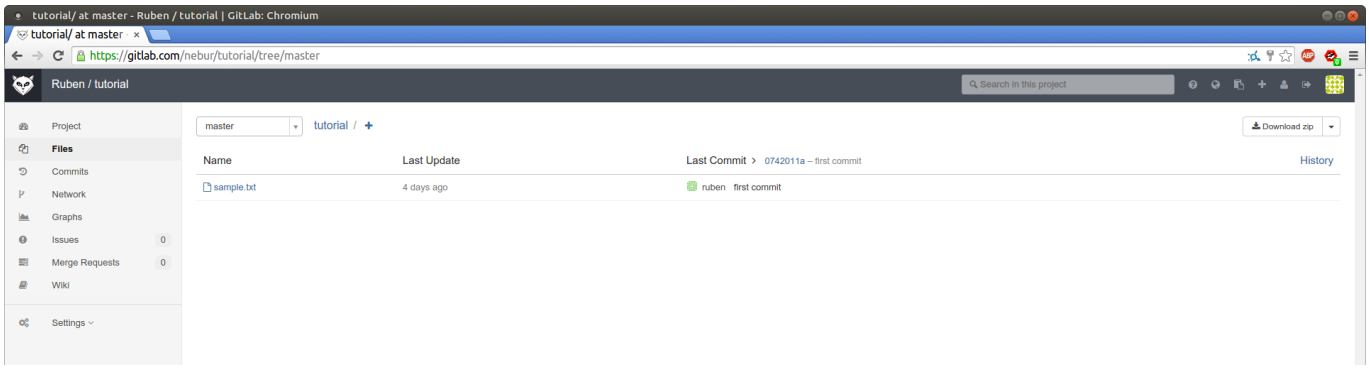
Cuando se nos pregunte por el nombre de usuario y contraseña, introduzca los creados en GitLab.

```
$ git push -u origin master
Username:
Password:
Counting objects: 3, done.
Writing objects: 100% (3/3), 245 bytes, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://monkey.backlogtool.com/git/BLGGIT/tutorial.git
 * [new branch]      master -> master
```

Abra la página de Git en Cartera. Usted encontrará que una nueva actualización que corresponde a su empuje al repositorio remoto se ha incluido en las actualizaciones recientes.




El archivo que hemos subido mediante push también se ha añadido en la lista de archivos del repositorio remoto y podemos observar el comentario.



*Clonar un repositorio remoto*

Realizaremos una copia de un repositorio remoto para poder empezar a trabajar con él en el equipo local.

Ahora, vamos a asumir el papel de otro miembro en el equipo y clonar el repositorio remoto existente en otro directorio llamado «tutorial 2».

Para ello utilizaremos el comando «clon» para copiar un repositorio remoto como se muestra en el siguiente ejemplo. Substitiremos <repository> con la URL del repositorio remoto y <directory> con el nombre del nuevo directorio en el que se descargaran los contenidos remotos.

```
$ git clone <repository> <directory>
```

Ejecutando el siguiente comando, el repositorio remoto se copiará en el directorio tutorial2.

```
$ git clone  https://gitlab.com/<user_name>/tutorial.git tutorial2
Cloning into 'tutorial2'...
Username:
Password:
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
Checking connectivity... done.
```

Para comprobar que la clonación se ha ejecutado correctamente, ejecutaremos el siguiente comando para comprobar el contenido de sample.txt del directorio clonado «tutorial2».

```
$ cat sample.txt
After three days without programming, life becomes meaningless.
```

## *Realizar un push des de un repositorio clonado*

Vamos a realizar un push des de un reporitorio clonado. Para ello trabajaremos en el repositorio clonado **tutorial2**

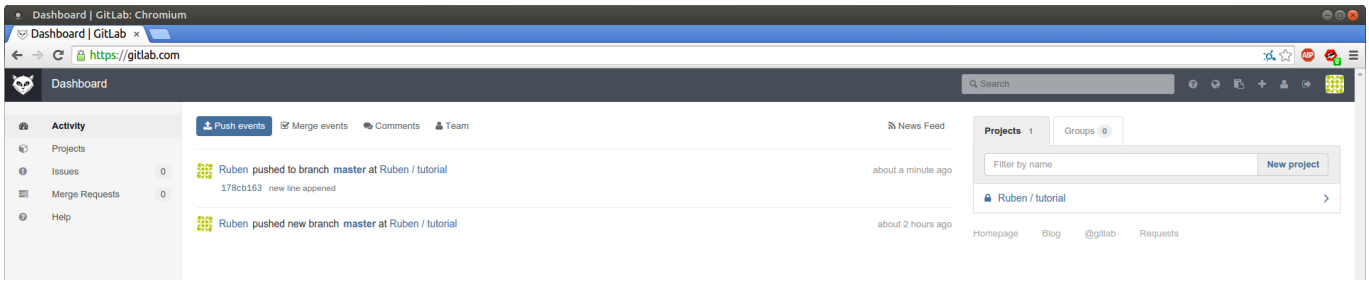
Primeramente añadiremos texto al fichero sample.txt

```
echo "new line text" >> sample.txt
$ git add sample.txt
$ git commit -m "new line appened"
[master 178cb16] new line appened
 1 file changed, 1 insertion(+)
```

Ahora realizaremos el push de este nuevo commit al repositorio remoto. Podemos omitir el repositorio y el branch cuando hacemos un push en el directorio de un repositorio clonado.

```
$git push
Username:
Password:
Counting objects: 3, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 282 bytes | 0 bytes/s, done.
Total 3 (delta 1), reused 0 (delta 0)
To https://gitlab.com/nebur/tutorial.git
   0742011..178cb16  master -> master
```

Podemos observar el nuevo push en GitLab. Podremos verlo en el Dashboard.




## *Hacer pull des de un repositorio*

En esta sección, vamos a hacer un pull del último cambio del

repositorio remoto a nuestro repositorio local (**en el directorio tutorial**).

Ahora que nuestro repositorio remoto está actualizado con los cambios de «tutorial2», vamos a hacer un pull para obtener el cambio y sincronizar nuestro repositorio inicial en el directorio «tutorial».

Para ejecutar un pull, utilizaremos el comando «pull». Si no se incluye el nombre del repositorio, el pull se hará en el repositorio con alias «origin».

```
$ git pull  ...
```

Ejecutaremos el siguiente comando

```
$ git pull origin master
Username:
Password:
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://gitlab.com/nebur/tutorial
 * branch            master     -> FETCH_HEAD
   0742011..178cb16  master     -> origin/master
Updating 0742011..178cb16
Fast-forward
 sample.txt | 1 +
 1 file changed, 1 insertion(+)
```

Ahora vamos a comprobar que el historico esta actualizado con el comando log

```
$ git log
commit 178cb1635855ea757a3bf6ed748f0096cdc1f6de
Author: ruben <ruben11set@hotmail.com>
Date:   Tue Feb 3 21:13:52 2015 +0100

    new line appened

commit 0742011aaf70e0e3a611fb22500c73d633f755c1
```

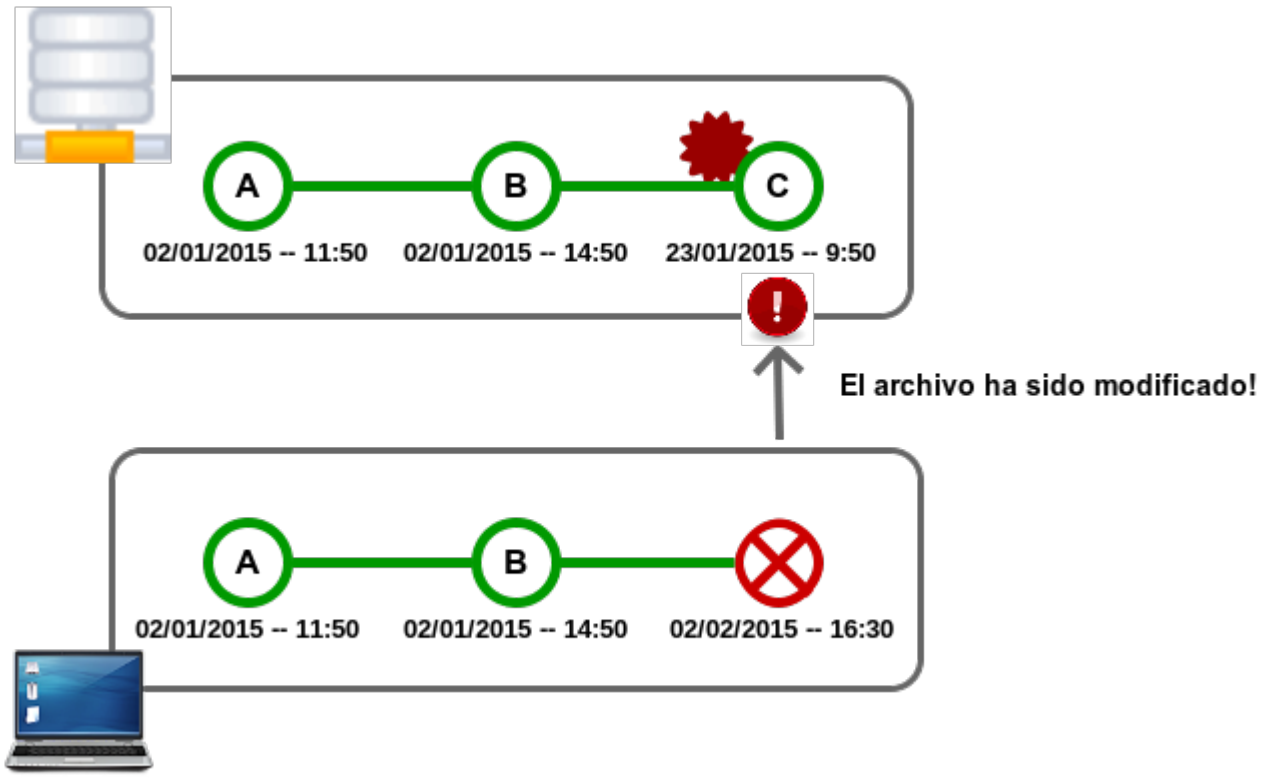
```
Author: ruben <ruben11set@hotmail.com>
Date:   Fri Jan 30 18:08:42 2015 +0100

    first commit
```

El nuevo commit que hemos añadido en «tutorial2» ahora aparece en la lista de registro de la histórico del repositorio «tutorial». Con el siguiente comando comprobaremos el contenido del fichero sample.txt

```
$ cat sample.txt
After three days without programming, life becomes meaningless.
new line text
```

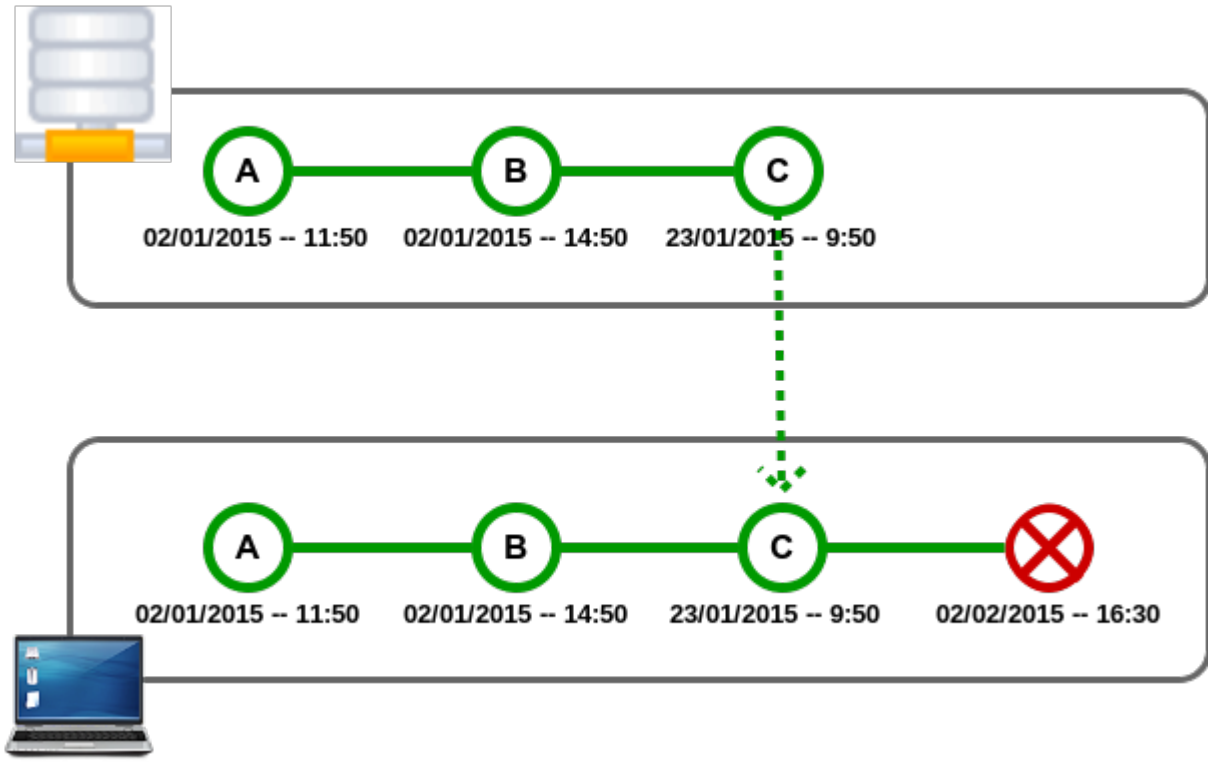
## Merge de histórico

### *Hacer merge de un histórico*




Un push puede ser rechazado si nuestro repositorio local no

está actualizado, posiblemente porque hay algunos cambios añadidos por otros en el repositorio remoto que aún no tenemos en nuestro repositorio local todavía.




Si ese es el caso, hacer un «merge» deberemos obtener el último cambio del repositorio remoto antes de hacer un push. Git de esta manera esto para asegurarse de que los cambios realizados por otros miembros quedan retenidos en el histórico (Commit C en la figura anterior).

Durante un «merge», Git intentará aplicar automáticamente los cambios de la historia y los combina con la rama actual. Sin embargo, si hay un conflicto debido a los cambios, Git retornará un error. En este caso se nos indicará que debemos resolver el conflicto de forma manual.

*Resolver un conflicto*

Como se describe en el apartado anterior, Git intentará aplicar automáticamente los cambios que enlazará con un

historial de cambios existente cuando se ejecuta «merge».

Aveces un «merge» puede fallar y puede suceder cuando hay un conflicto. Si dos o más miembros hacen cambios en la misma parte de un archivo en las dos branchs (remota y local en este caso) que está siendo fusionadas, Git no será capaz de hacerlo automáticamente y retornará un conflicto de combinación.

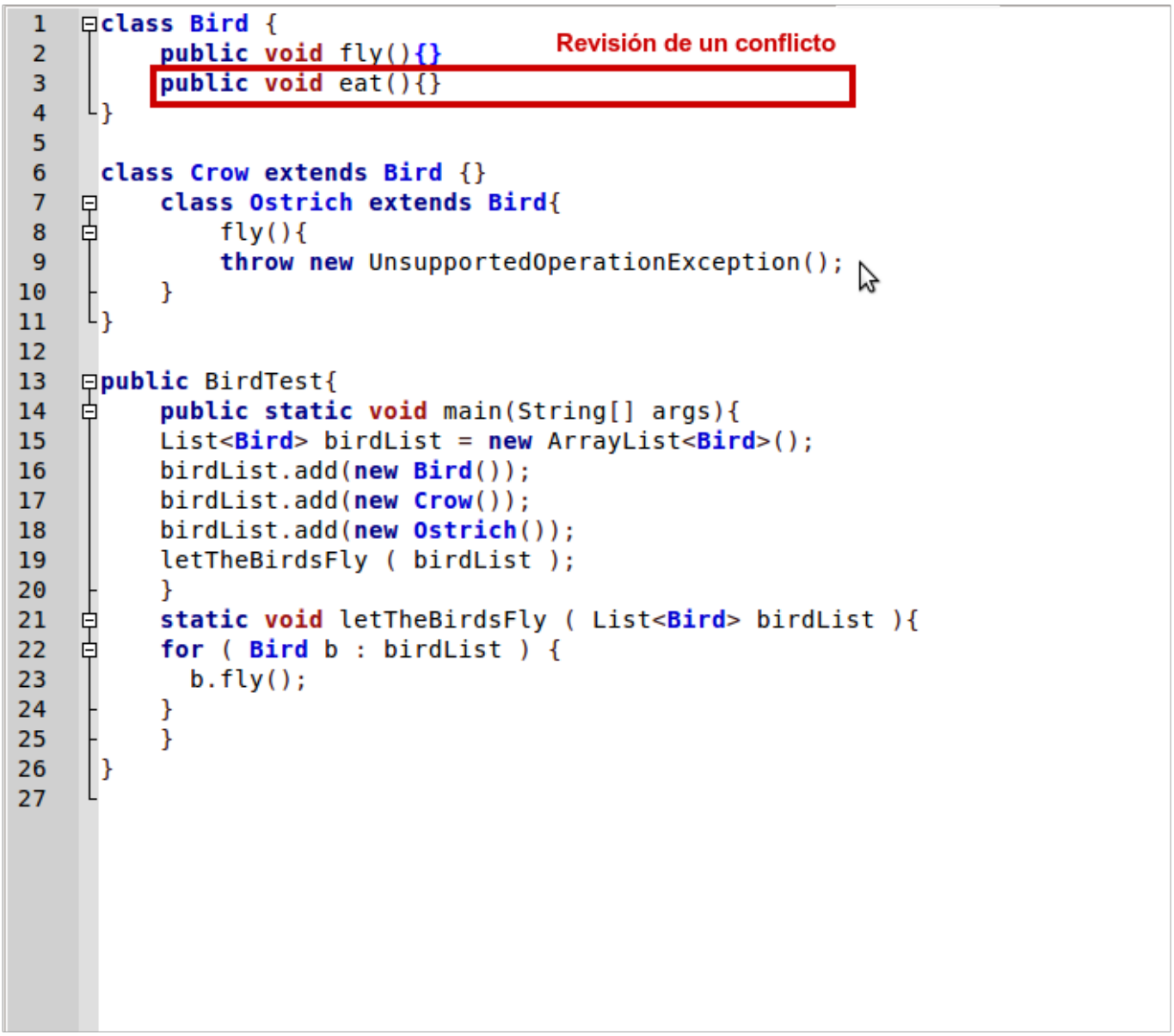
Cuando esto suceda, Git agregará algunos marcadores de resolución de conflictos al archivo. Los marcadores actúan como un indicador para ayudarnos a entender las secciones en el contenido del archivo en conflicto que debemos resolver manualmente.

Ejemplo de un conflicto

```
1  class Bird {
2      public void fly(){}
3      <<<<<<< HEAD
4      eatOnce
5      =======
6      public void eat(){}
7      >>>>>>> 12ih76093675093h78fjn76575a76556f8h9
8  }
9
10 class Crow extends Bird {}
11     class Ostrich extends Bird{
12         fly(){
13         throw new UnsupportedOperationException();
14     }
15 }
16
17 public BirdTest{
18     public static void main(String[] args){
19     List<Bird> birdList = new ArrayList<Bird>();
20     birdList.add(new Bird());
21     birdList.add(new Crow());
22     birdList.add(new Ostrich());
23     letTheBirdsFly ( birdList );
24     }
25     static void letTheBirdsFly ( List<Bird> birdList ){
26     for ( Bird b : birdList ) {
27       b.fly();
28     }
29     }
30 }
31
```

Todo lo anterior «=====» es su contenido local, y todo lo que se encuentra a continuación se trata de la rama remota.

Podemos resolver las partes en conflicto tal y como se muestra a continuación. Ahora ya está listo para proceder con la creación de un «merge commit».




**Observaciones**

En este tercer post hemos visto como trabajar con un repositorio remoto y hemos introducido el concepto merge. En

el próximo, y último  post de esta serie, mostraremos paso a paso como trabajar con merge.

Ruben.