Administrar un repositorio en Github

Ahora que tenemos listo el repositorio local en nuestra computadora, podemos empezar a trabajar. Seguramente ahora te estarás preguntando, si varias personas participan en un mismo repositorio, ¿entonces cómo cada persona puede realizar su propio trabajo sin afectar los avances de los demás? A diferencia de herramientas como Google Docs o Presentaciones de Google, donde todos trabajan sobre un mismo archivo, en un repositorio, cada persona crea un espacio propio dentro del mismo (llamado **rama**) donde realiza y sube su parte correspondiente. Una vez que alguien o todos hayan terminado, se combinan sus avances para integrar el trabajo en una rama principal. A continuación, aprenderemos todos lo necesario para crear ramas y unir el contenido de varias ramas.

I. Crear y eliminar ramas

Crear ramas es una tarea esencial para el control de versiones de cualquier proyecto de software. La recomendación general es que se cree una rama por cada función de un sistema, pero también se puede crear una rama por cada integrante de equipo para que realice sus respectivos avances. Una vez completada la actividad en cada rama individual, se fusionan para ir integrando el trabajo. En cambio, sólo se elimina una rama cuando deja de ser útil, ya sea porque su contenido se trasladó a otra rama o está desactualizada. Si bien lo ideal es conservar todas las ramas para tener copias de seguridad en caso de algún error, puede ser beneficioso eliminar las innecesarias para gestionar mejor el repositorio. A continuación, veremos cómo realizar ambas tareas, crear y eliminar ramas.

1) En la terminal de Windows, escribe el siguiente comando para crear una carpeta de trabajo nueva: ***git checkout -b [Nombre de rama nueva]*.** La carpeta se llamará “develop”.

```
git checkout -b develop
```

NOTA: La bandera -b te cambia a la carpeta que acabas de crear. Para cambiar entre ramas que ya existen, no hace falta ponerlo, como se muestra en la imagen.

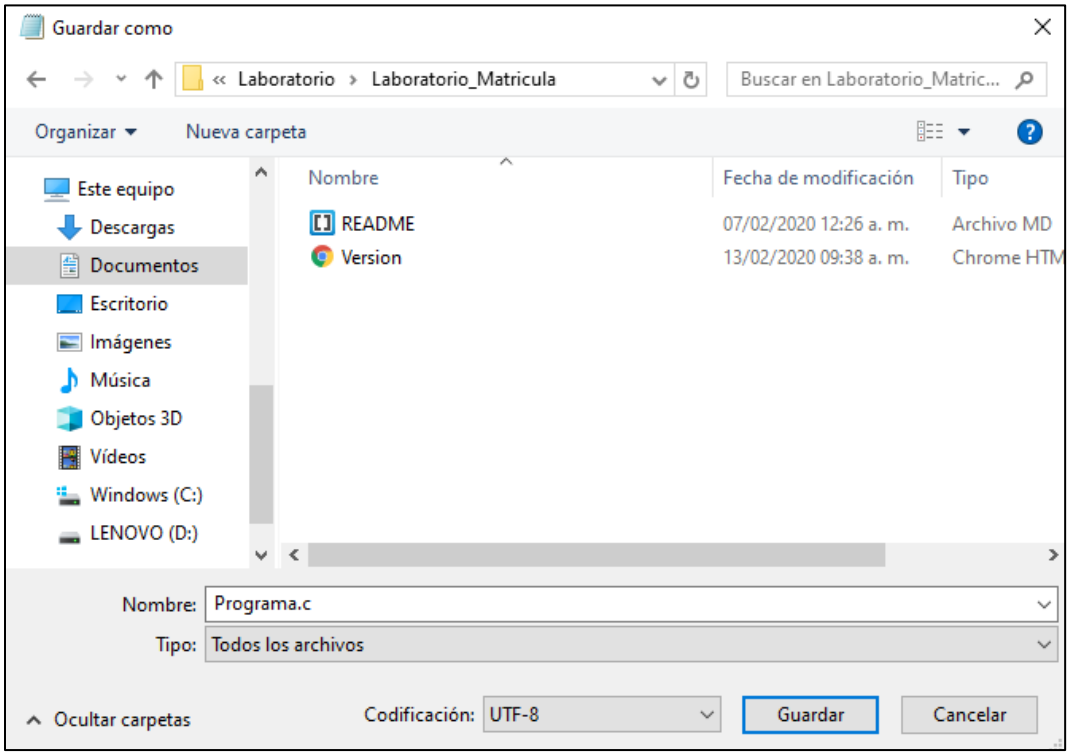
```
git checkout develop
```

2) Abre un archivo de texto nuevo en el Bloc de Notas

3) Copia el siguiente código

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(){
   int length;
   scanf("%i", &length);

   int office = pow(2,length);
   printf("%i",office);
   return 0;
}
```

4) Guarda el archivo con el nombre “Programa.c” en la dirección de tu repositorio local



5) Realiza un commit en la rama “develop”. Ponle un mensaje cualquiera al commit.

`git add -A` → Sube los cambios realizados a los archivos a una zona de pruebas de Git

`git commit -m "Versión 2"` → Sube los archivos al repositorio local

`git push origin develop` → Sube los archivos de tu repositorio local al remoto

6) Cámbiate a la rama “master”

```
git checkout master
```

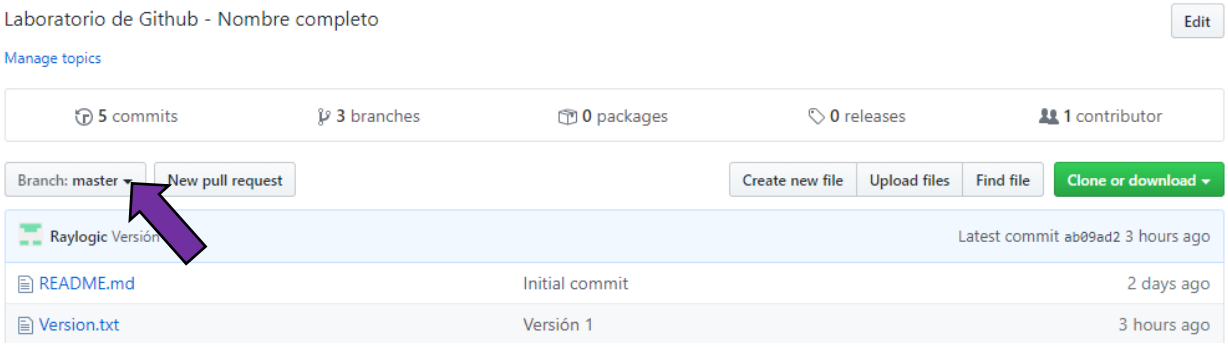
7) Crea otra rama nueva, ahora con el nombre “discard”

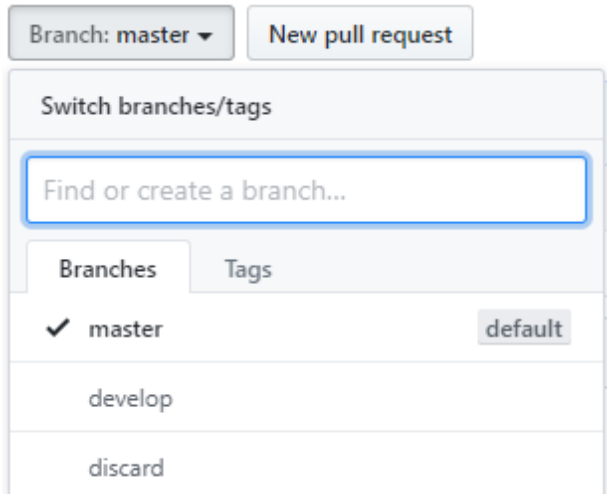
```
git checkout -b discard
```

8) Repite los pasos 2-5 sobre la rama “discard”
9) Escribe el comando ***git branch*** para ver todas las ramas que hay por ahora en el repositorio. La que está en color verde es aquella en la que estás posicionada de momento.

```
  develop
* discard
  master
```

10) Comprueba que las dos ramas se crearon correctamente. Abre Github y presiona el botón “Branch” en el lado medio izquierdo de la pantalla. Las tres ramas deberían aparecer.





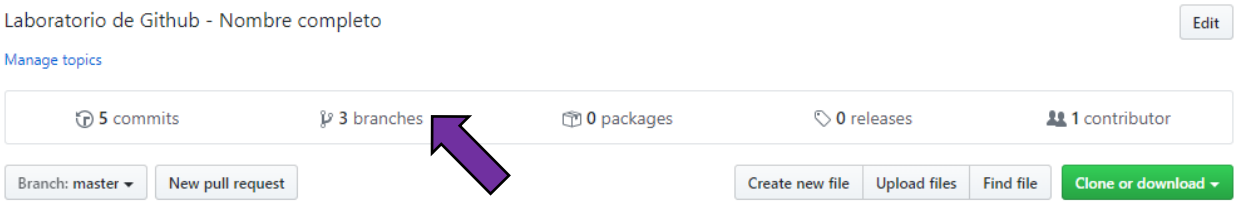
11) Cámbiate a la rama “develop”

```
git checkout develop
```

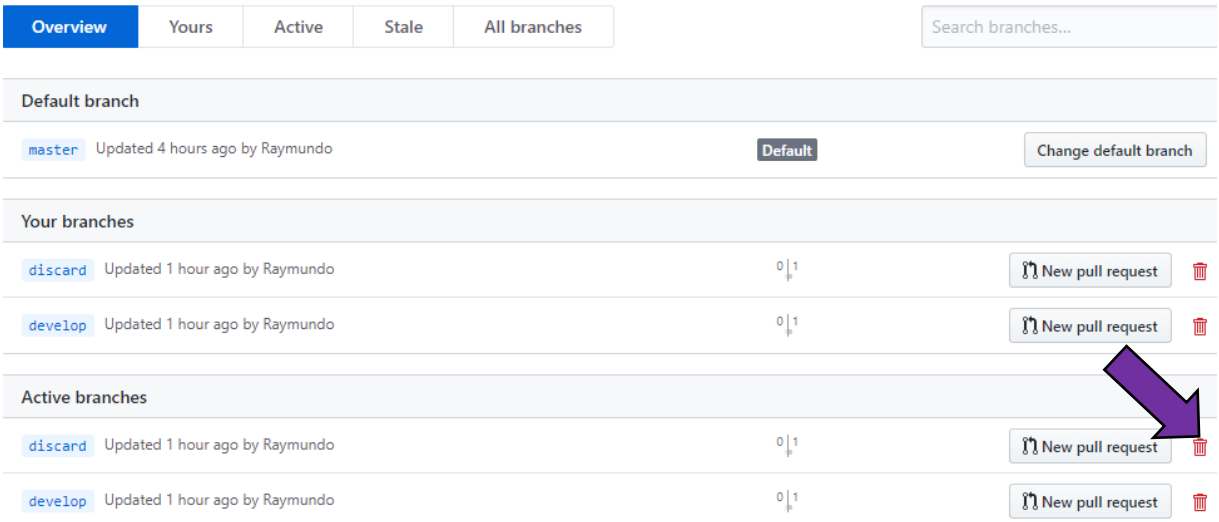
12) Escribe el comando ***git branch -d [Nombre de rama]*** para eliminar la rama “discard”

```
git branch -d discard
```

13) El comando anterior elimina la rama del repositorio local, pero falta hacerlo en el remoto. Abre Github y presiona el botón “Branches” en la barra superior del repositorio



14) Presiona el ícono “Eliminar” de la rama “discard” para eliminarla del repositorio remoto.



II. Combinar ramas

Ahora que cada integrante del equipo ha realizado su trabajo en su respectiva rama, ¿cómo unimos todo en un solo lugar? Aquí entra el concepto de **Merge**, la acción de combinar las ramas entre sí, asegurándose de que el contenido de estas no se sobreponga o choque entre sí. Aquí es donde entra la importancia de la constancia en el control de versiones.

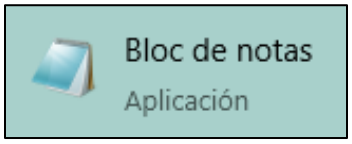
Supongamos que dos personas están trabajando en una función de un sistema. Cada una hace sus propias adiciones al código, pero no le avisa al otro de estos cambios. Un día, uno de ellos modifica sin querer la parte de su amigo en su propia rama. Cuando ambos intentan combinar sus avances, el repositorio les dice que hay inconsistencias entre los códigos (a esto le llamamos un **conflicto),** por lo que ambos tendrán que decidir cuál de los dos es el que se queda.

En un trabajo en equipo, es necesario que los integrantes se comuniquen entre ellos constantemente para informar de lo que han cambiado en el código. También deben subir sus avances de manera constante para asegurarse de que no haya cambios importantes en la estructura del propio sistema. De esta manera, se evita que haya conflictos durante el desarrollo del proyecto que tomen mucho tiempo en arreglar. A continuación, veremos cómo realizar un merge entre dos ramas. Cabe aclarar que este ejercicio está realizado en un ambiente controlado, por lo que no veremos cómo resolver conflictos de código por el momento.

1) Cámbiate a la rama “develop”

```
git checkout develop
```

2) Abre el archivo “Version.html” en el Bloc de Notas

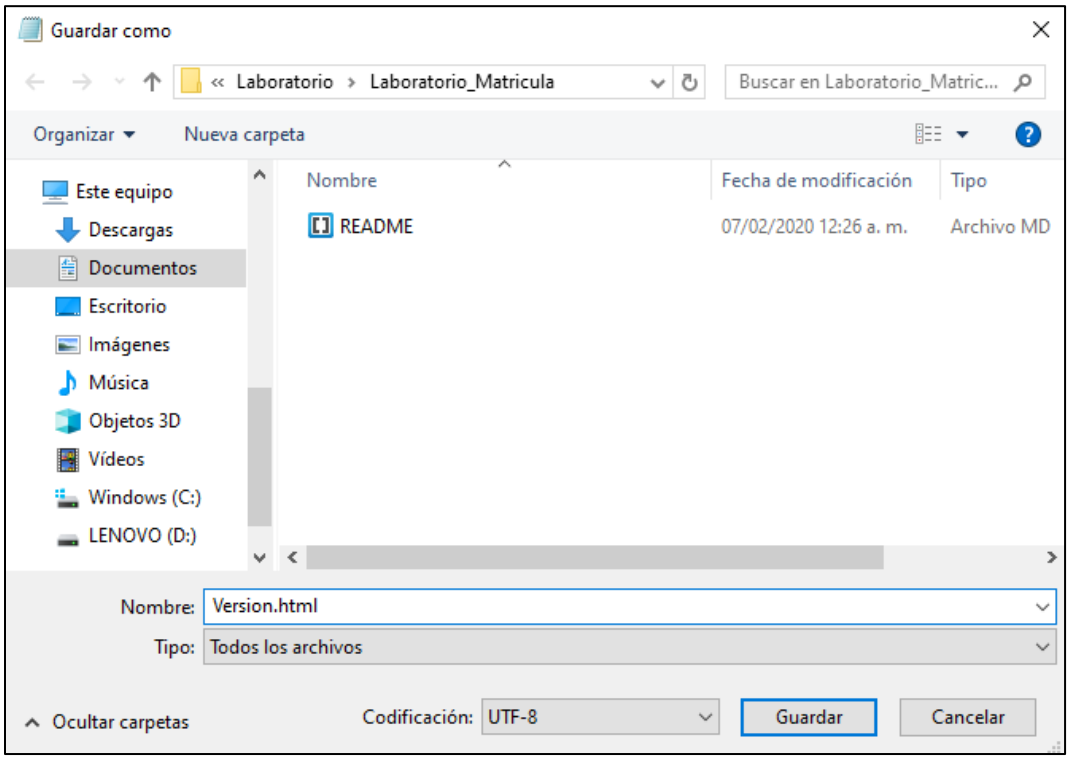


3) Elimina todo el contenido del archivo y copia el siguiente:

```
<!DOCTYPE HTML>
<html>
<head>
   <title>Versión mejorada</title>
   <meta charset="UTF-8">
</head>
<body>
   <main>
```

```
        <h1>Laboratorio de Github</h1>
        <div class="card-panel">
            <span class="blue-text">Tarjeta</span>
        </div>
    </main>
</body>
</html>
```

4) Guarda el archivo con el mismo nombre



5) Sube el contenido modificado del repositorio local a la rama “develop”

```
git add -A
git commit -m "Versión 2"
git push origin develop
```

Como podemos ver, la rama “develop” ahora tiene más avances del trabajo que la rama “master”, por lo que debemos juntar los archivos de ambas ramas para que cualquiera que use el repositorio tenga la versión más actualizada de estos. Estos son los comandos necesarios para unir dos ramas:

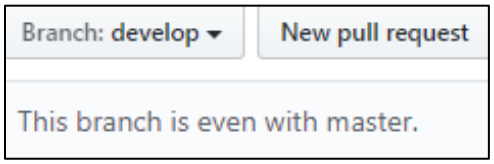
- ✓ ***git checkout [Rama destino]*** = Cambia a la rama donde se combinarán los archivos
- ✓ ***git pull origin [Rama destino]*** = Carga los archivos de la rama donde se hará el merge

- ✓ ***git merge [Rama origen]*** = Combina el contenido de la rama seleccionada con la actual
- ✓ ***git push origin [Rama destino]*** = Sube el contenido combinado a la rama seleccionada del repositorio remoto

6) Siguiendo los comandos en orden, realiza un merge de la rama "develop" con "master"

```
C:\Users\rayro\Documents\TAREAS\TEC\Profesional\Laboratorio\Laboratorio_Matricula> git checkout master
Switched to branch 'master'
Your branch is up to date with 'origin/master'.

C:\Users\rayro\Documents\TAREAS\TEC\Profesional\Laboratorio\Laboratorio_Matricula> git pull origin master
From https://github.com/Raylogic/Laboratorio_Matricula
 * branch            master     -> FETCH_HEAD
Already up to date.

C:\Users\rayro\Documents\TAREAS\TEC\Profesional\Laboratorio\Laboratorio_Matricula> git merge develop
Updating ab09ad2..fab7f15
Fast-forward
 Programa.txt | 12 ++++++++++++
 Version.txt  |  6 ++++--
 2 files changed, 16 insertions(+), 2 deletions(-)
 create mode 100644 Programa.txt

C:\Users\rayro\Documents\TAREAS\TEC\Profesional\Laboratorio\Laboratorio_Matricula> git push origin master
Total 0 (delta 0), reused 0 (delta 0)
To https://github.com/Raylogic/Laboratorio_Matricula.git
   ab09ad2..fab7f15  master -> master
```

7) Para comprobar si el merge fue exitoso, revisa la rama "develop" en Github. Si aparece el siguiente mensaje, entonces ambas ramas tienen el mismo contenido.



III. Trabajar en un repositorio con un equipo

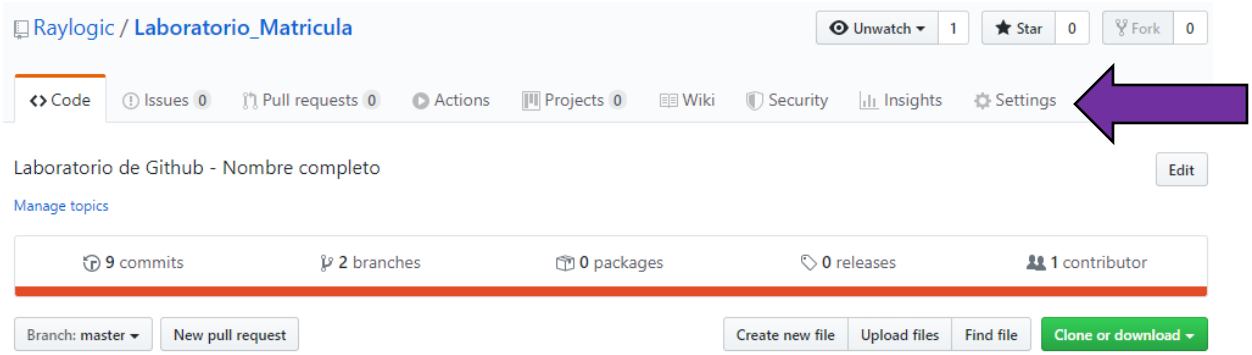
En los ejercicios anteriores hemos visto cómo crear, eliminar y combinar ramas en un repositorio. Sin embargo, todo lo has hecho tú solo en tu propio repositorio. ¿Qué pasa cuando debes trabajar con otras personas? A continuación, trabajaremos en un nuevo repositorio para poner a prueba lo que has aprendido en estos laboratorios con un equipo.

1) **Forma equipo con otras dos personas del salón.**

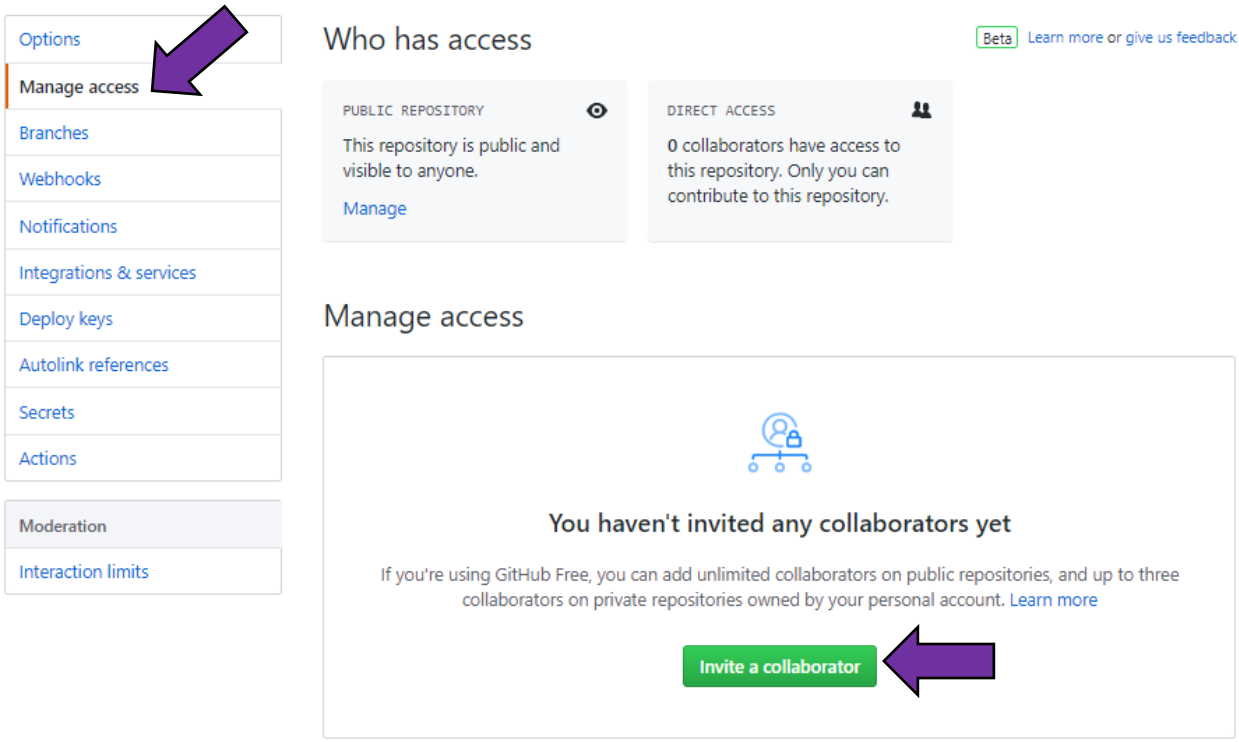
- ➢ Cada uno tendrá un rol a lo largo de este ejercicio.
- ➢ La persona a la cual pertenece el repositorio se llamará **'Líder'**
- ➢ La persona que realizará los avances en el repositorio se llamará **'Participante'**
- ➢ La persona que eliminará los avances en el repositorio se llamará **'Supervisor'**

2) Decidan en cuál repositorio quieren trabajar.

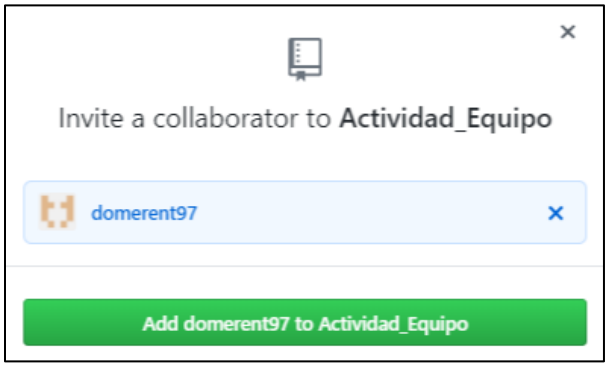
3) **<u>Líder</u>** → En tu repositorio, presiona el botón Settings



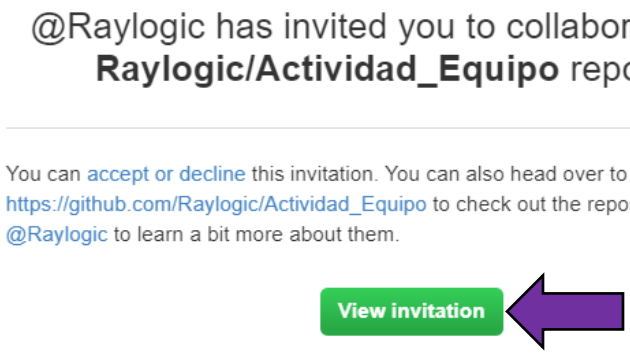
4) En la barra lateral izquierda, presiona el botón Manage Access. En la parte inferior te dirá que tu repositorio que no tienes colaboradores. Presiona el botón Invitar a un colaborador.



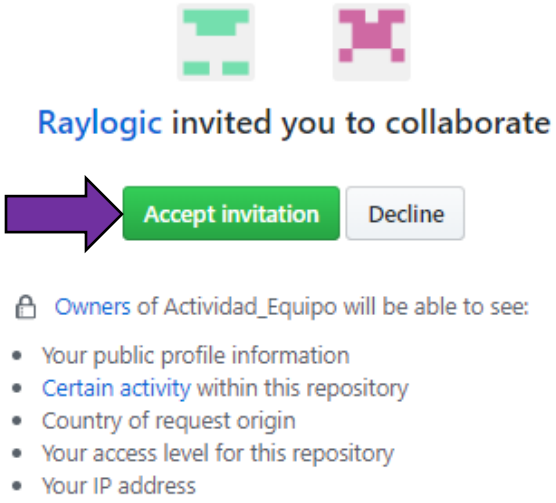
5) Escribe el nombre de usuario o mail de uno de tus compañeros. Deberá aparecer en la lista desplegable. Selecciónalo y presiona el botón de invitación.



6) Repite los pasos 5-6 para invitar al otro compañero
7) **Participante** y **Supervisor** → Recibirán un mail con la invitación al repositorio. Presionen el botón 'View Invitation'

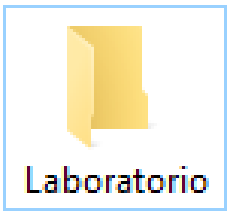


8) Se desplegará el repositorio con la invitación. Presiona el botón 'Accept Invitation'.



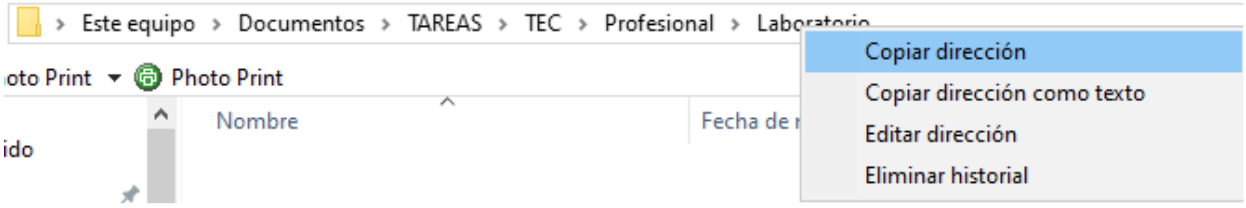
9) Se desplegará el menú principal del repositorio. Ahora puedes empezar a trabajar en él.

10) Crea una carpeta en tu computadora (ponle con un nombre que tenga una sola palabra).

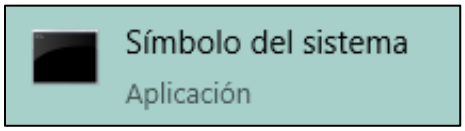


11) Desde la terminal de Windows, accede a la dirección de la carpeta que acabas de crear

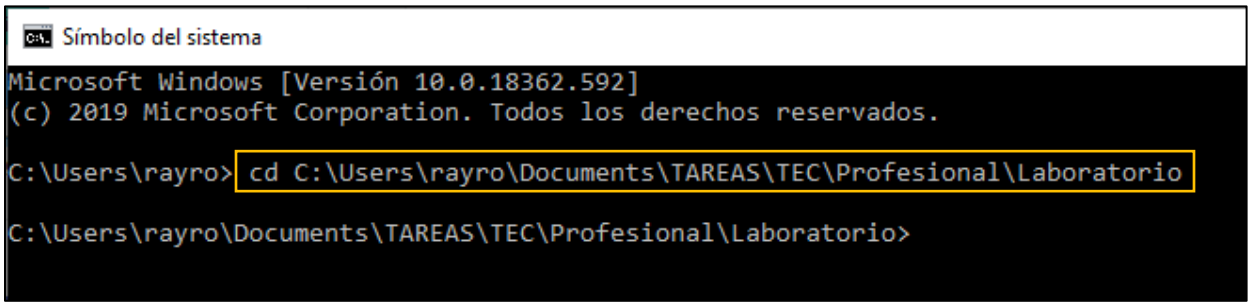
- En el explorador de archivos entra a la carpeta. Una vez dentro, haz clic derecho sobre el nombre en la barra de dirección. Selecciona la opción “Copiar dirección”.



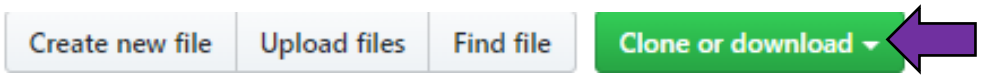
- Abre la terminal de Windows



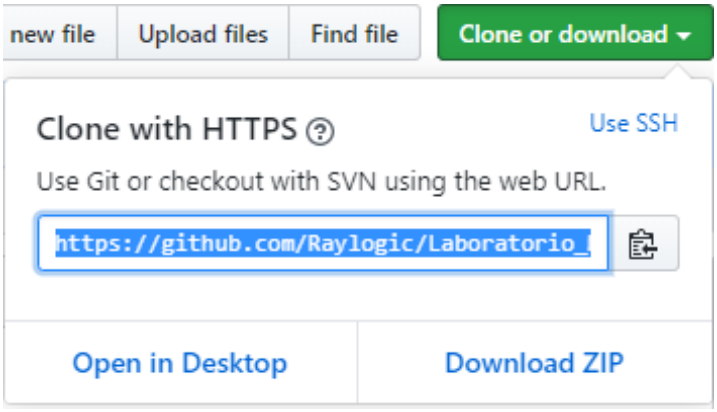
- Escribe el comando ***cd (Dirección de la carpeta)*** y estarás dentro de la carpeta



12) En Github, en la parte derecha del repositorio, presiona el botón verde “Clonar o descargar”



13) Aparecerá un cuadro de diálogo con una dirección URL. Cópialo con el método que desees.



14) Escribe el comando ***git clone***. Luego, pega el URL que copiaste del repositorio

```
git clone https://github.com/Raylogic/Laboratorio_Matricula.git
```

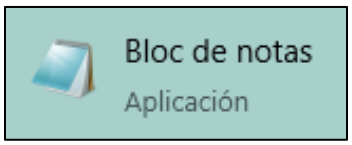
15) Se clonará tu repositorio remoto en la dirección seleccionada, creando tu repositorio local

```
Cloning into 'Laboratorio_Matricula'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

16) **Todos** → Ahora que todos tienen su repositorio local, podemos empezar a trabajar. Escribe el comando ***git checkout -b [Nombre de nueva rama]*** para crear una carpeta de trabajo que tenga de nombre tu matrícula.

```
git checkout -b A00570654
```

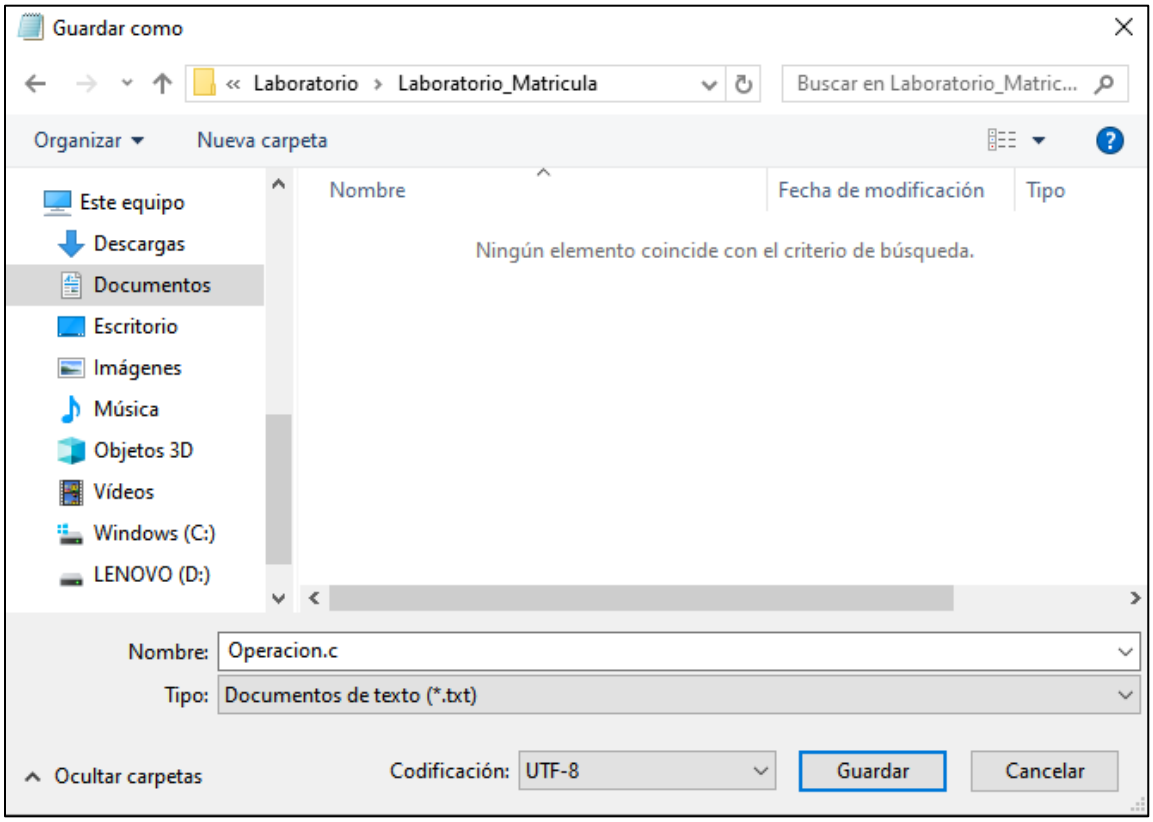
17) **Líder** → Abre el Bloc de Notas de Windows



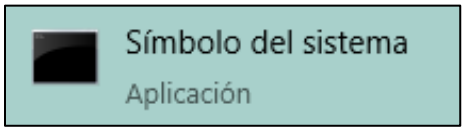
18) Copia el siguiente código en un archivo de texto nuevo

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main(){
int num1 = 12;
int num2 = 3;

int op1 = suma(num1,num2);
printf("%i",op1);
int op2 = resta(num1,num2);
printf("%i",op2);
int op3 = multi(num1,num2);
printf("%i",op3);

return 0;
}
```

19) Guarda el archivo bajo el nombre “Operacion.c” en la dirección de tu repositorio local.

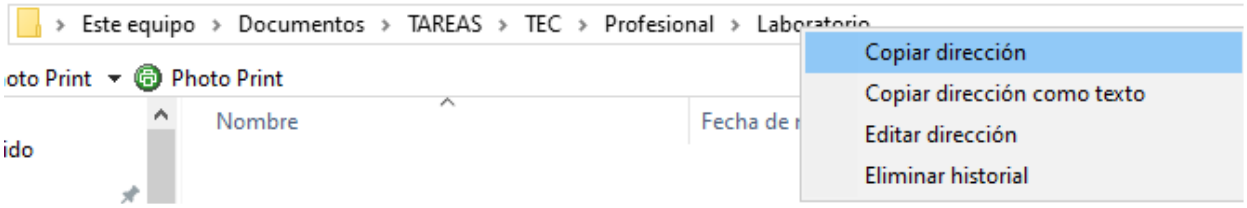


20) Abre la terminal de Windows

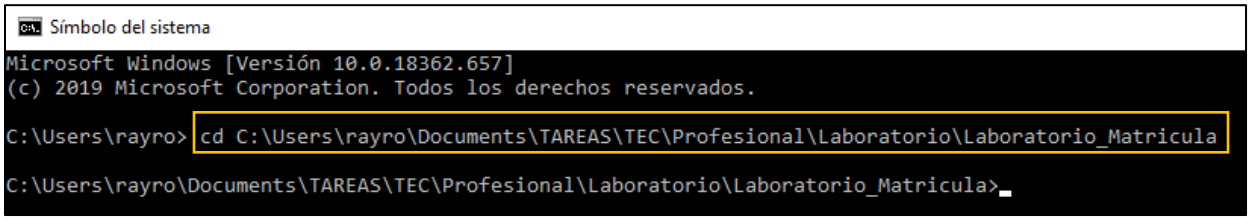


21) Accede al directorio en el que está tu repositorio local

- En el explorador de archivos entra a la carpeta del repositorio local. Una vez dentro, haz clic derecho en el nombre en la barra de dirección. Selecciona la opción "Copiar dirección"



- Escribe el comando ***cd (Dirección de la carpeta)*** para acceder a ella



22) Escribe el comando ***git checkout -b [Nombre de rama nueva]*** para crear una nueva carpeta. Ponle de nombre tu matrícula

```
git checkout -b A00570654
```

23) Sube el archivo 'Operación.c' al repositorio remoto

```
git add -A
git commit -m "Base del programa"
git push origin A00570654
```

24) **Participante** y **Supervisor** → Abre la terminal de Windows

25) Accede al directorio en el que está tu repositorio local (Consulta el Paso 21)

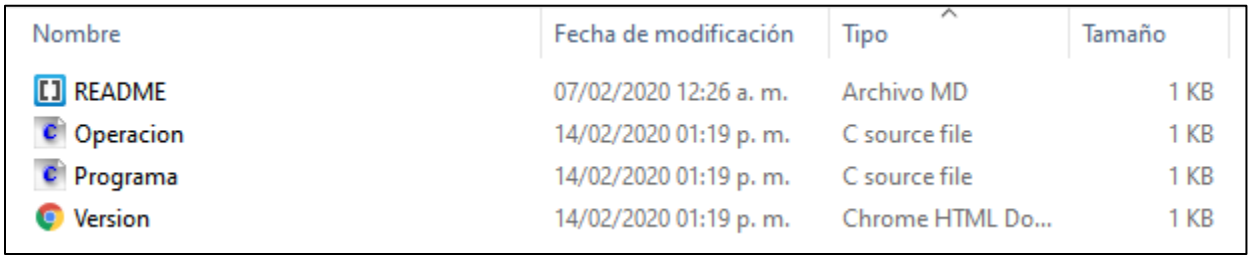
26) Escribe el comando ***git pull.*** Esto actualizará el repositorio local con los últimos cambios que se han realizado al repositorio remoto

27) Escribe el comando ***git checkout [Rama]*** para posicionarte en la rama del Líder

```
git checkout A00570654
```

28) Escribe el comando ***git pull origin [Rama del Líder]*** para descargar en tu computadora el contenido de la rama del Líder. Puedes comprobarlo entrando a la carpeta en tu explorador.

```
git pull origin A00570654
```



Nombre	Fecha de modificación	Tipo	Tamaño
README	07/02/2020 12:26 a. m.	Archivo MD	1 KB
Operacion	14/02/2020 01:19 p. m.	C source file	1 KB
Programa	14/02/2020 01:19 p. m.	C source file	1 KB
Version	14/02/2020 01:19 p. m.	Chrome HTML Do...	1 KB

29) Escribe el comando ***git checkout -b [Nombre de rama nueva]*** para crear una nueva carpeta. Ponle de nombre tu matrícula

```
git checkout -b A01702948
```

30) Abre el archivo 'Operacion.c'

31) **Participante** → Inserta en el archivo el siguiente código entre los #include y el main()

```
int suma(int a, int b){
   return a+b;
}
int resta(int a, int b){
   return a-b;
}
```

Archivo Edición Formato Ver Ayuda

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>


int main(){
int num1 = 12;
int num2 = 3;

int op1 = suma(num1,num2);
printf("%i",op1);
int op2 = resta(num1,num2);
printf("%i",op2);
int op3 = multi(num1,num2);
printf("%i",op3);

return 0;
}
```

32) **Supervisor** → Elimina del archivo las siguientes líneas

*Funciones: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(){
int num1 = 12;
int num2 = 3;

int op1 = suma(num1,num2);
printf("%i",op1);
int op2 = resta(num1,num2);
printf("%i",op2);
```

```
return 0;
}
```

33) **Participante** y **Supervisor** → Guarden el archivo con el mismo nombre: 'Operacion.c'

34) Sube el archivo 'Operación.c' modificado al repositorio remoto

```
git add -A
git commit -m "Funciones añadidas"
git push origin A01702948
```

Participante

```
git add -A
git commit -m "Código depurado"
git push origin A01703675
```

Supervisor

Ahora que los tres integrantes del equipo tienen sus ramas con sus respectivos avances, es tiempo de unir los cambios en una sola rama.

35) **Participante** → Posiciónate sobre la rama del Líder

```
git checkout A00570654
```

36) Escribe el comando ***git pull origin [Rama destino]*** para cargar el contenido de la rama del Líder en tu computadora

```
git pull origin A00570654
```

37) Escribe el comando ***git merge [Rama origen]*** para combinar el contenido de la rama del Líder con el de la tuya

```
git merge A01702948
```

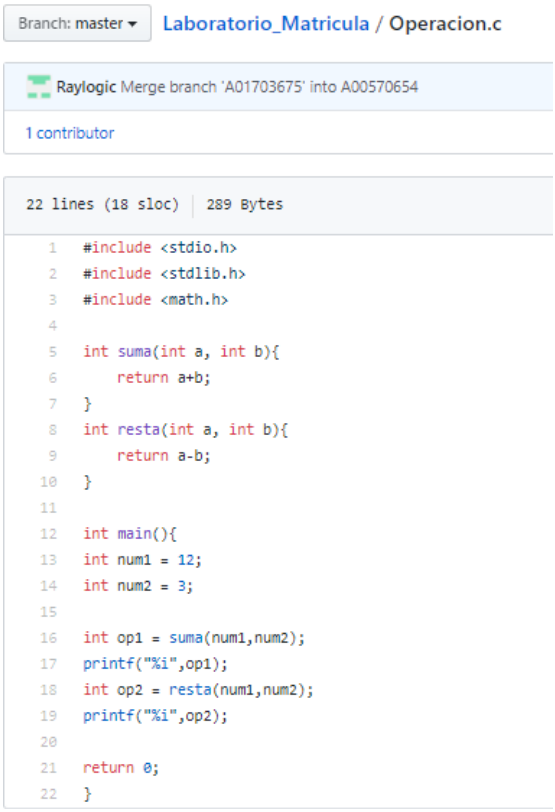
38) Escribe el comando ***git push origin [Rama destino]*** para subir los contenidos combinados en la rama del líder en el repositorio remoto

```
git push origin A00570654
```

39) **Supervisor** → Repite los pasos 35-38 para combinar tu rama con la del Líder

40) **Líder** → Repite los pasos 35-38 para combinar tu rama con todos los cambios de los demás con la rama 'master'

41) **Todos** → Accedan a la rama 'master' en Github. Deberían aparecer todos los cambios que cada uno realizó en el archivo si hacen clic sobre él



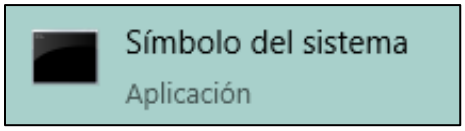
III. Banderas de comandos de Git

A lo largo de estos laboratorios hemos visto la mayoría de los comandos de Git que utilizarás para administrar tus repositorios de proyectos. Por si no lo notaste, algunos de estos como ***git checkout***, ***git commit*** o ***git add*** usan elementos adicionales llamados **banderas**.

Las banderas son argumentos booleanos/opciones que cambian el comportamiento de un comando sin modificar su función base. Por ejemplo, el ***git add*** base sólo permite subir un archivo a la zona de pruebas a la vez. Sería tedioso tener que especificar de uno a uno qué archivos queremos subir, ¿verdad? No obstante, la bandera ***-A*** permite actualizar todos en una sola subida. De ahí la razón por la que la hemos usado a lo largo de los ejercicios.

Cada comando tiene sus propias banderas, y son muchas como para poder enlistarlas aquí mismo. Aunque solo usaremos las que hemos visto en los laboratorios, si estás interesado en conocer toda la lista de banderas que tiene un comando, entonces a continuación veremos cómo buscarlas.

1) Abre la terminal de Windows



1) Escribe el comando ***git help***, seguido del comando del cual quieres consultar

```
git help add
```

2) Se te abrirá la documentación de Git del comando que insertaste

Desafortunadamente, a diferencia de UNIX, Git no puede desplegar el manual de un comando en la terminal, sino que te redirecciona a la documentación que tiene en su carpeta (la que instalaste). Aún si la buscas desde la página de Git, la información será la misma, por lo que no hay diferencia.

3) Para ver las banderas que tiene el comando, simplemente revisa su documentación hasta que encuentres el apartado Options, en el cual se enlista y explica lo que hace cada bandera

OPTIONS

<pathspec>...

Files to add content from. Fileglobs (e.g. `*.c`) can be given to add all matching files. Also a leading directory name (e.g. `dir` to add `dir/file1` and `dir/file2`) can be given to update the index to match the current state of the directory as a whole (e.g. specifying `dir` will record not just a file `dir/file1` modified in the working tree, a file `dir/file2` added to the working tree, but also a file `dir/file3` removed from the working tree). Note that older versions of Git used to ignore removed files; use `--no-all` option if you want to add modified or new files but ignore removed ones.

For more details about the <pathspec> syntax, see the *pathspec* entry in gitglossary(7).

-n

--dry-run

Don't actually add the file(s), just show if they exist and/or will be ignored.

-v

--verbose

Be verbose.