

## Git and GitHub

First video: <https://www.youtube.com/watch?v=RGQj5yH7evk&feature=youtu.be>

Themes:

- ✓ Importance of Git and GitHub

Git es un sistema de control de versiones, es decir, una plataforma donde, principalmente programadores, tienen seguimiento en los cambios que hacen en el código.

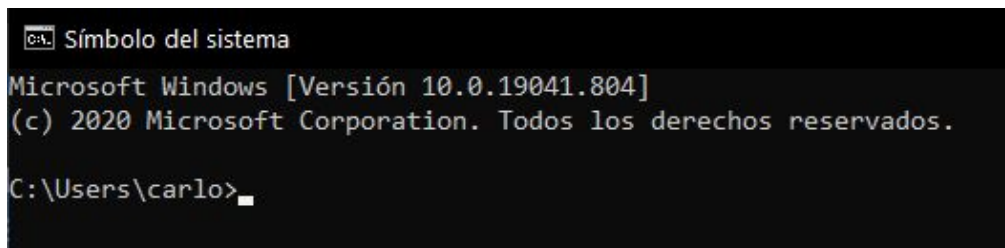
Su importancia radica en que podemos rastrear todos los cambios, desde que subimos el primer código, lo modificamos-cargamos, hasta que terminamos el proyecto. Esto implica que se pueden rastrear errores por la fecha en que se “subió-cargó” el código, volver a hacer cambios y demás.

Terms (términos que permiten entender el “rollo” de Git):

- Directory -> Folder

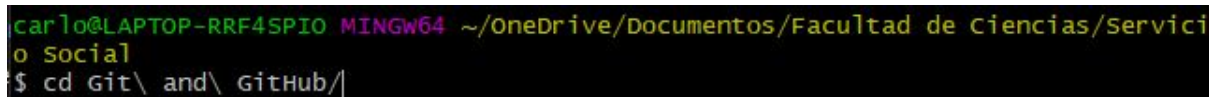


- Terminal or Command Line -> Interface for text commands



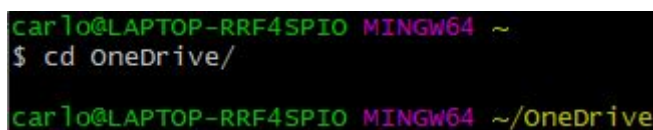
Donde se escribirán los comandos de texto (en vez de seleccionar “a mano”)

- CLI -> Command Line Interface



Es la línea de código que vas a escribir para realizar algo.

- cd -> Change Directory (como dar doble click en un directorio *sin* darlo)



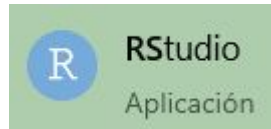
En este caso accedí al folder ‘OneDrive’

Es importante aprender a usar la “Terminal” y sus comandos porque a partir de ellos se hace más fácil la instalación de programas o su “lectura-corrimiento-carga” (run

programs) . Esta forma de trabajar es para tener experiencia como “desarrollador” de algún proyecto.

**ADVERTENCIA SOBRE LA TERMINAL:** la función ctrl+c, ctrl+v no sirven para pegar o copiar en la terminal. Así que se debe hacer la selección de lo que se quiera copiar con el click derecho para copiar y así mismo para pegar.

- Code Editors -> Word Processor for Writing Code



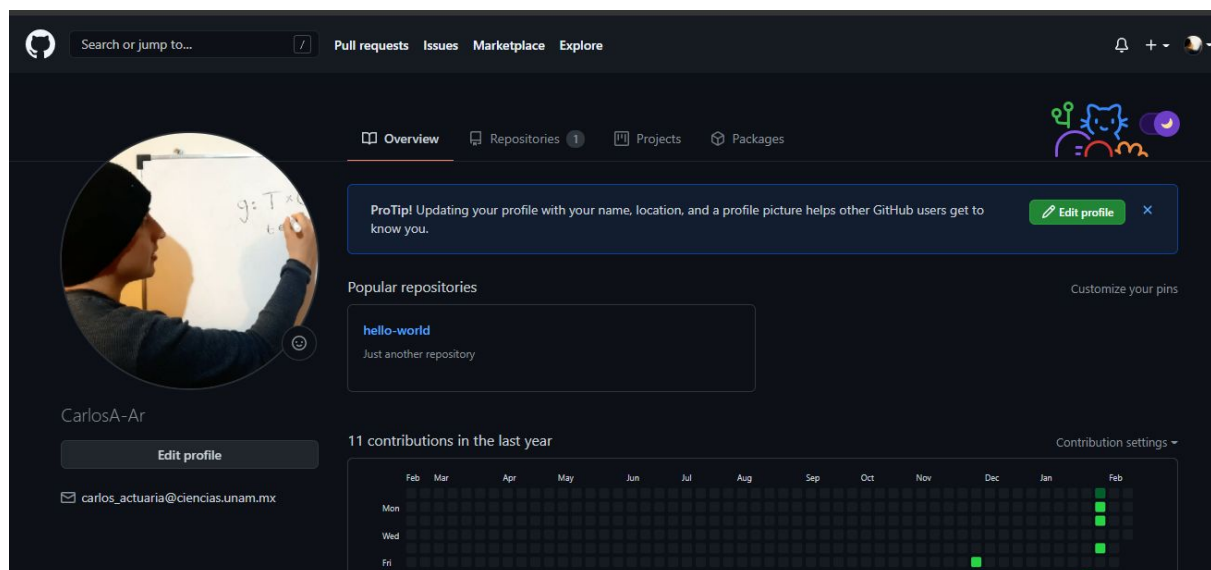
Por ejemplo: Java, RStudio, Spyder, etc.

- Repository -> *Git Repository*. Folder o lugar donde guardas tus proyectos.

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de Ciencias/Estadística/Análisis de Supervivencia/Proyecto/Estadística3 (master)
$
```

- GitHub

Es un sitio web para tener tus proyectos/repositories en línea. Git rastrea los cambios en el código a lo largo del tiempo mientras GitHub “almacena-aloja” tus *repositories*. GitHub permite que sea más fácil trabajar en grupo con otras personas y organizar tus folders en un portafolio para que veas los “empleados” potenciales.



#### ✓ How to use Git and GitHub?

Commands: todos se hacen en **minúsculas** sobre la consola.

- **Clone**. Sirve para “traer” (en tu computadora) archivos que están en algún repositorio de GitHub (por ejemplo) y no están en tu computadora. Es importante “traer” los archivos para que puedas modificarlos. No uso

“descargar” porque una vez que se usó el comando “clone” en un archivo, no es necesario hacerlo en cada ocasión.

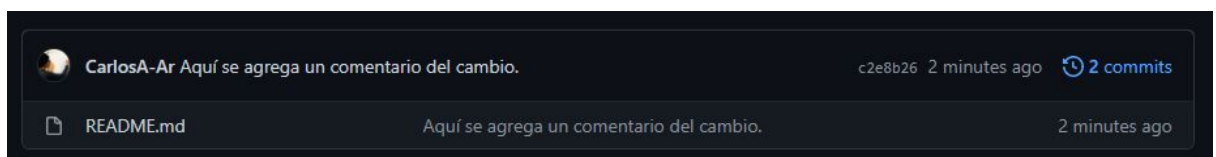
- **add.** Cuando se quieren agregar, borrar o modificar “archivos” (por ejemplo un archivo.R) en un repositorio hay que “avisar” a Git sobre esos cambios para que los *agregue* al historial de cambios en el código.
- **commit.** Una vez que se añadieron los cambios, hay que confirmarlos con este comando.
- **push.** Para subir los cambios y notificarlos a Git. Estos cambios se suben en el repositorio, ya sea GitHub u otros.
- **pull.** Actualizar los cambios en la computadora local. Descargar los cambios que se han hecho hasta el momento en el repositorio, incluidos los tuyos. Si hay tres personas trabajando en el repositorio y luego suben sus cambios, al momento de usar pull, entonces se descargarán los cambios que esas personas realizaron antes de darle pull.
- **status.** Permite conocer los archivos que se cambiaron, borraron y demás antes de que confirmaras (usaras commit) el cambio que tu hiciste en el archivo.
- **clear.** Funciona para eliminar las líneas de código que se han escrito en la terminal hasta el momento. Esto solo es visual, no quita las instrucciones previas, solo “limpia” la terminal.

**NOTA:** Cada una de las funciones anteriores se escriben en la terminal precedidas de la palabra git. Por ejemplo: git push.

✓ Basic git repositories.

Para esto es necesario crear una cuenta de GitHub y un repositorio.

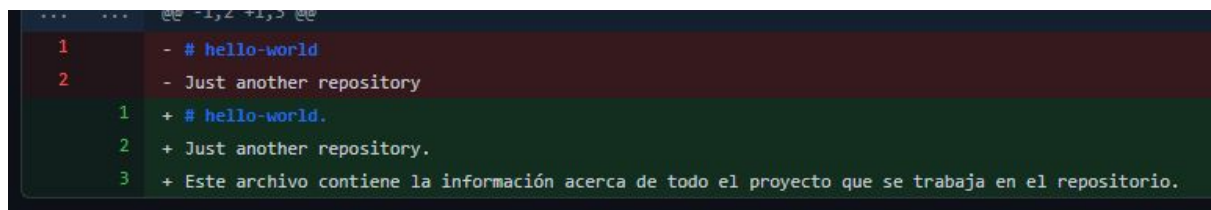
Creó un repositorio y luego un archivo ‘.md’ que es: Markdown, además podemos ver los cambios que hemos hecho, así como los comentarios en cada parte de los “commit”:



Si hacemos cambios y les damos “commit” entonces aparecerán los cambios por vez.

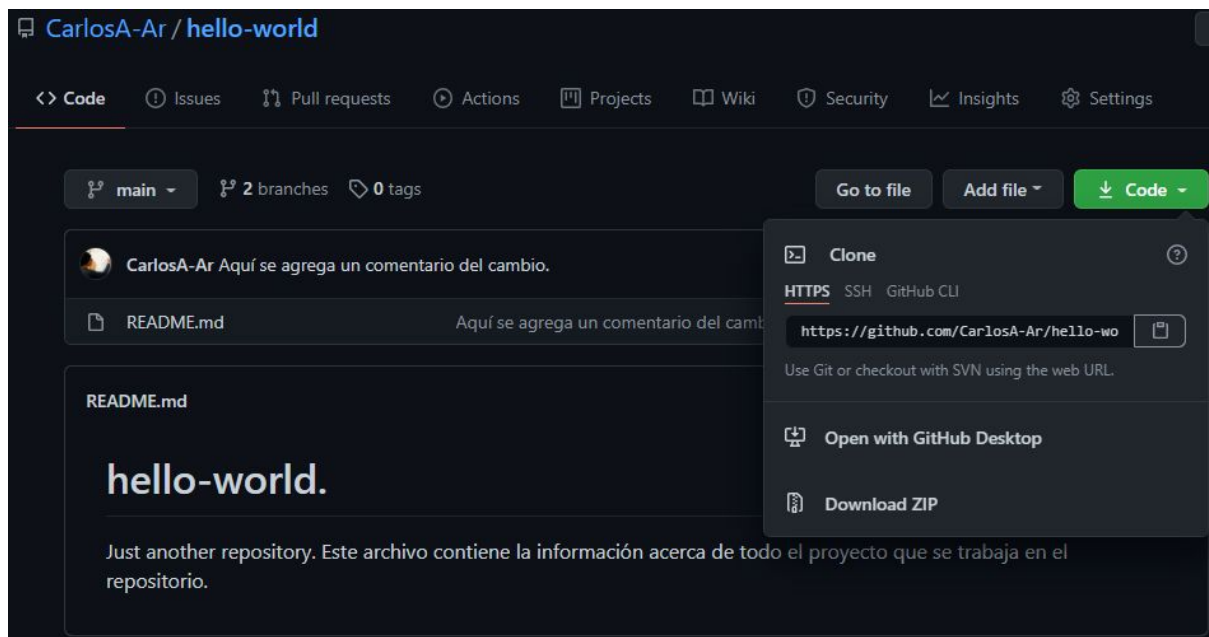


Si vamos a la “actualización” del archivo Markdown donde se agregó un cambio, podemos notar lo siguiente:

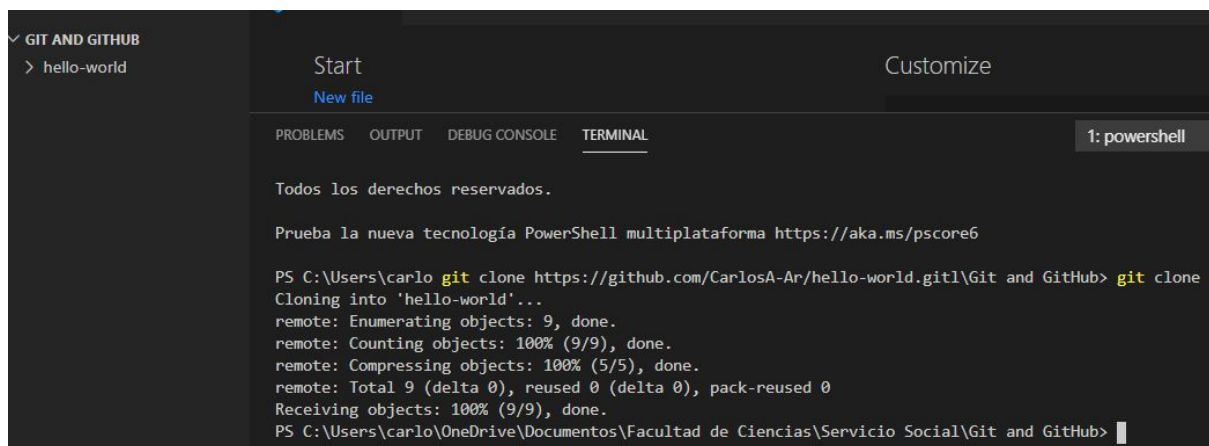


Las líneas en rojo (o con un signo de - a la izquierda) significan una modificación de la línea, lo verde es la modificación hecha que se subió al final y si no hay sombreado entonces esas líneas de código no fueron modificadas.

Usamos Visual Studio code <https://code.visualstudio.com/download> para editar el Markdown que hemos creado. Primero copiamos la liga de descarga



Luego esa liga la pegaremos en la consola correspondiente con el comando git clone



Logrando así que el repositorio que se creó en GitHub ahora esté en nuestra carpeta.

Podemos ahora añadir un archivo particular en Visual Studio Code. En ese momento no se sube al repositorio de Git, por ende si lo creamos tenemos que subirlo al repositorio.



Para subir el archivo HTML que creamos usamos 'git add .' el punto es muy importante porque se indica que todos los archivos se suban al repositorio aunque se puede especificar el archivo que se desea subir:

```
GitHub\hello-world> git add .  
GitHub\hello-world> git add index.html
```

Podemos checar el status de los archivos que hemos creado (para verificar que ya se han rastreado).

```
itHub\hello-world> git status  
On branch main  
Your branch is up to date with 'origin/main'.  
  
Changes to be committed:  
  (use "git restore --staged <file>..." to unstage)  
    new file:   index.html
```

Posteriormente podemos confirmar que se reconozcan los archivos o modificaciones que hicimos al repositorio. Esto se hace con la función commit.

```
GitHub\hello-world> git commit -m "Añadimos el index.html"
```

Es importante que se haga la sintaxis 'git commit -m ""' puesto que la m significa message (mensaje) que debe ser de preferencia descriptivo de las modificaciones realizadas.

```
[main 71ac010] Añadimos el index.html  
1 file changed, 0 insertions(+), 0 deletions(-)  
create mode 100644 index.html
```

Hasta este momento solo hemos guardado los cambios de forma local, es decir, en nuestro equipo, por lo cual es importante ponerlos en GitHub:

git push

Antes de poner el archivo en GitHub, es importante que la computadora se enlace con el correo electrónico que pusiste en tu cuenta de GitHub que se creó al inicio.

**ESTO NO ENTENDÍ CÓMO SE HACE PERO YA LO HABÍA HECHO XD**

Al final tuve que usar el Git Bash donde pude subir el archivo:

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de Ciencias Exactas y Naturales/Social/Git and GitHub/hello-world (main)  
$ git push  
Enumerating objects: 5, done.  
Counting objects: 100% (5/5), done.  
Delta compression using up to 8 threads  
Compressing objects: 100% (3/3), done.  
writing objects: 100% (3/3), 526 bytes | 263.00 KiB/s, done.  
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0  
remote: Resolving deltas: 100% (1/1), completed with 1 local object.  
To https://github.com/CarlosA-Ar/hello-world.git  
d07c0c8..bac566e main -> main
```



Pero sí se pudo.

Ahora si quisiéramos añadir una de las carpetas que ya tenemos creadas entonces basta con hacer lo siguiente:

1. Crear la carpeta.
2. Entrar a la carpeta y añadir algún archivo que se desea subir.
3. Cambiar en la terminal la carpeta con la que queremos trabajar ahora con el comando

```
PS C:\Users\carlo\OneDrive\Documentos\Facultad
GitHub\hello-world> cd ../Hello-World2
PS C:\Users\carlo\OneDrive\Documentos\Facultad
GitHub\Hello-World2> █
```

`cd ../nombre_de_la_carpeta`

4. **git init** será el comando en la terminal que funja como forma de hacer que la carpeta que creamos sea parte del Git

```
PS C:\Users\carlo\OneDrive\Documentos\Facultad de Ciencias\Servicio Social\Git and
GitHub\Hello-World2> git init
Initialized empty Git repository in C:/Users/carlo/OneDrive/Documentos/Facultad de
Ciencias/Servicio Social/Git and GitHub/Hello-World2/.git/
PS C:\Users\carlo\OneDrive\Documentos\Facultad de Ciencias\Servicio Social\Git and
GitHub\Hello-World2> █
```

5. Hay que conectar la carpeta con Git Hub, para ello creamos un repositorio vacío en GitHub y posteriormente copiaremos el enlace donde se encuentra dicho repositorio para tener el comando:

‘git remote add origin [link](#)’ El remote se refiere que usaremos la referencia de un git no local (que es el que creamos previamente) así aparecerá lo siguiente:

```
+ demo-repo2 git:(master) git remote add origin git@github.com:gwenf/dem
o-repo2.git
+ demo-repo2 git:(master) git remote -v
origin git@github.com:gwenf/demo-repo2.git (fetch)
origin git@github.com:gwenf/demo-repo2.git (push)
```

6. Podemos ahora sí cargar los archivos que creamos:  
usando git push (simplemente) o especificando dónde: git push origin master. Si no queremos especificar todas las veces dónde va el archivo, podemos poner el comando: ‘git push -u origin master’ así solo será necesario escribir posteriormente git push y por defecto (default) se subirán las actualizaciones en la rama principal original sin necesidad de especificar.

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de Ciencias/Servicio Social/Git and GitHub/Hello-world2 (master)
$ git push origin master
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 809 bytes | 404.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
To https://github.com/CarlosA-Ar/Hello-world2.git
   95e08d4..0313a39  master -> master

carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de Ciencias/Servicio Social/Git and GitHub/Hello-world2 (master)
$ git push -u origin master
Everything up-to-date
Branch 'master' set up to track remote branch 'master' from 'origin'.
```

- ✓ Branching (*ramificación*), Merging (*fusión*), forking (*bifurcación*) and other intermediate terms.

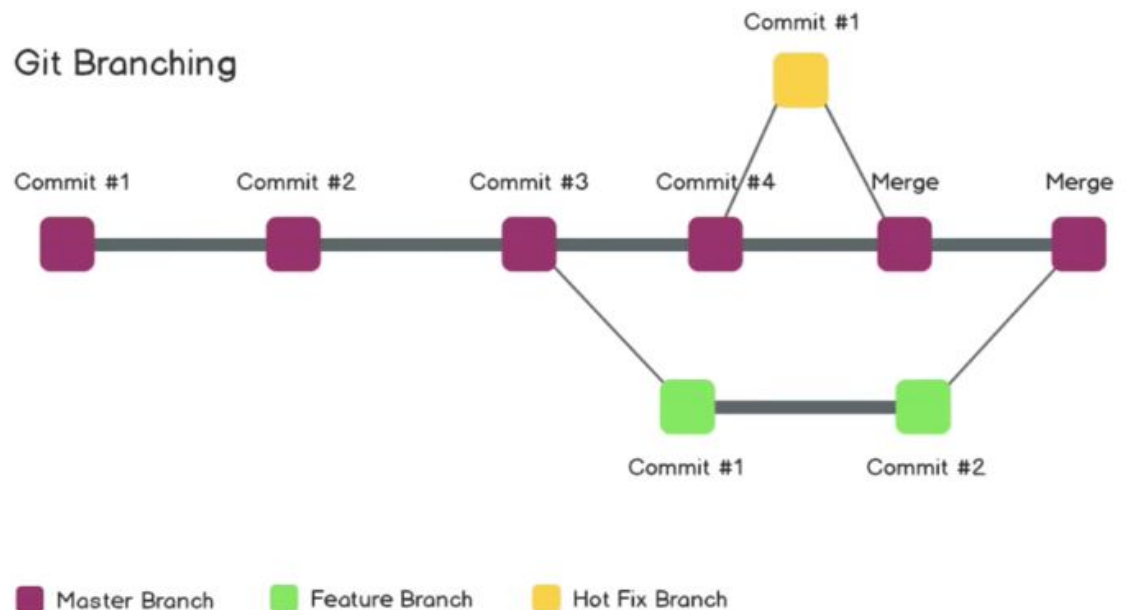
**Git Branching:** para entender la ramificación es importante conocer que existe la rama principal, o predeterminada llamada “**master branch**”, pero no sólo existe la rama principal donde se hacen los “commits” (confirmaciones de cambio) correspondientes, pueden existir otras “ramas características” (*feature branch*).

La gran diferencia es que en las ramas características (*feature branches*) se pueden hacer commits pero estos no se verán reflejados en la rama principal (*master branch*) además de que cada rama solo podrá ver los cambios de código que ahí suceden. Si estoy dentro de una **feature branch** particular, entonces no podré ver los cambios de código que se hagan en otras *features branches* y viceversa.

El uso de estas *feature branch* radica en la construcción de código que aun tiene errores o no está completo por lo que no quiere actualizarse la rama principal dado que podría causar errores de compilación u otros. (Es como tener tu respaldo personal que posteriormente se unirá a la rama principal).

Si se trabaja en resolver un problema particular de código se pueden crear otras ramas de revisión “**fixhot branch**”.





Veamos un ejemplo:

Si escribimos en la terminal el comando: 'git branch' veremos cuáles son las ramas que

tenemos:

```
o Social/Git and GitHub/hello-world (main)
$ git branch
* main
```

donde podemos notar que no

tenemos más que una rama principal.

Para crear otra rama (create a branch) usamos el comando 'git checkout -b mi-primer-branch' donde checkout nos permite ver la opción sobre las branch, y -b es para crear una nueva rama mientras que mi-primer-branch será el nombre de la rama a la que te vas a dedicar a trabajar. En el caso "cotidiano" estas branch tienen nombres especiales como "featue/ or feature or feature-11" para referir a alguna característica particular del código.

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Docu
o Social/Git and GitHub/hello-world (main)
$ git checkout -b mi-primer-rama
Switched to a new branch 'mi-primer-rama'
```

Posteriormente aparecerá que ahora estás trabajando en la nueva rama:

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Fa
o Social/Git and GitHub/hello-world (mi-primer-rama)
```

Si usamos de nuevo el código 'git branch' podemos notar en qué rama nos encontramos, ésta

```
o Social/Git and GitHub/hello-world (mi-primer-rama)
$ git branch
main
* mi-primer-rama
```

se encontrará resaltada:

pero si queremos regresar a alguna otra rama podremos simplemente usar el comando 'git checkout nombre\_de\_la\_rama' y así por ejemplo podemos cambiar a la rama master:

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facu
o Social/Git and GitHub/hello-world (mi-primer-rama)
$ git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.

carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facu
o Social/Git and GitHub/hello-world (main)
$
```

**Tip:** si no quieres escribir demasiado en la computadora, puedes fácilmente poner la tecla tabulador luego de escribir al menos dos letras del nombre de la rama (no necesariamente las primeras dos) o de un directorio.

Si cambiamos algo en el archivo README.md entonces podemos poner dicho cambio en la rama particular que se llama "mi-primer-rama" teniendo lo siguiente

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facu
o Social/Git and GitHub/hello-world (mi-primer-rama)
$ git status
On branch mi-primer-rama
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:   README.md

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

carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facu
o Social/Git and GitHub/hello-world (mi-primer-rama)
$ git add README.md

carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facu
o Social/Git and GitHub/hello-world (mi-primer-rama)
$ git commit -m "README Modificado"
[mi-primer-rama 9517e0b] README Modificado
1 file changed, 5 insertions(+)
```

AWESOME: si nos cambiamos a la rama principal (que en mi caso se llama main) entonces no veremos en el código los cambios que hicimos dentro de la rama secundaria o característica "feature branch".

Ya que hicimos el cambio en una rama y queremos unir ese cambio a la rama principal es conveniente que primero entendamos cuáles son las diferencias entre la rama principal y la otra rama con el comando 'git diff mi-primer-rama'. Lo que se muestra a continuación:

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de ciencias/Servicio Social/Git and GitHub/hello-world (main)
$ git diff mi-primer-rama
diff --git a/README.md b/README.md
index cd33fe1..3e08976 100644
--- a/README.md
+++ b/README.md
@@ -13,8 +13,3 @@ Pero bueno, ya lo sabr

Al parecer todo salio bien, no puedo poner acentos pero al menos ya se pudo subir desde la terminal de _Git Bash_

-
-## Aqui hacemos algo para la rama secundaria.
-
-1. Abre index.html en tu browser
-
```

Y lo que se encuentra en rojo representa las cosas que la rama principal no tiene y sí tiene la rama secundaria.

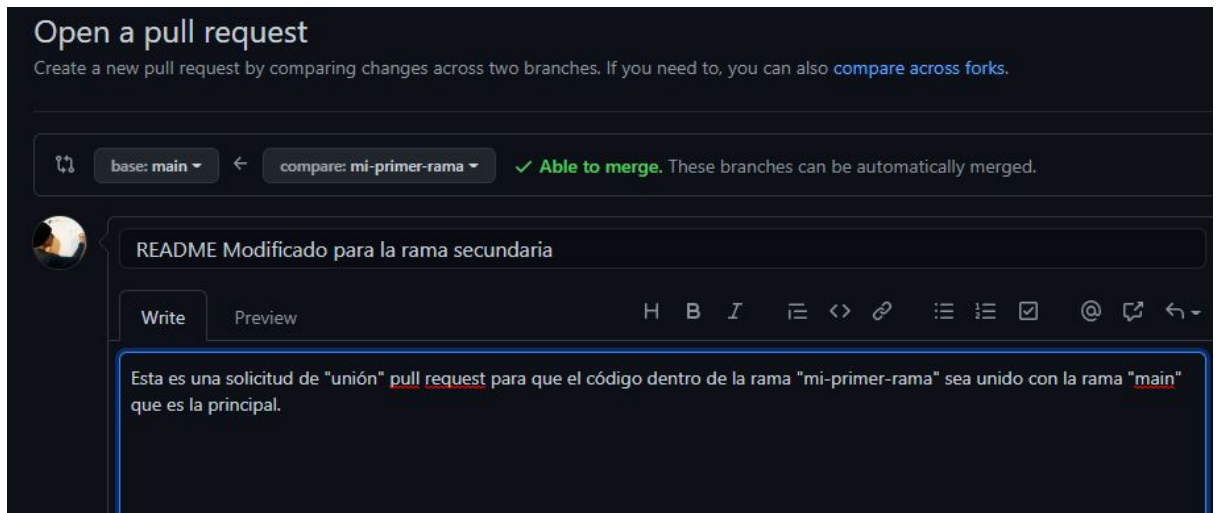
Ahora podríamos ya unir los cambios con el comando merge, sin embargo lo que es más común es subir en GitHub los cambios en la rama secundaria como siempre:

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de ciencias/Servicio Social/Git and GitHub/hello-world (mi-primer-rama)
$ git push --set-upstream origin mi-primer-rama
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (3/3), 429 bytes | 429.00 KiB/s, done.
Total 3 (delta 1), reused 0 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'mi-primer-rama' on GitHub by visiting:
remote:   https://github.com/CarlosA-Ar/hello-world/pull/new/mi-primer-rama
remote:
To https://github.com/CarlosA-Ar/hello-world.git
 * [new branch]      mi-primer-rama -> mi-primer-rama
Branch 'mi-primer-rama' set up to track remote branch 'mi-primer-rama' from 'origin'.
```

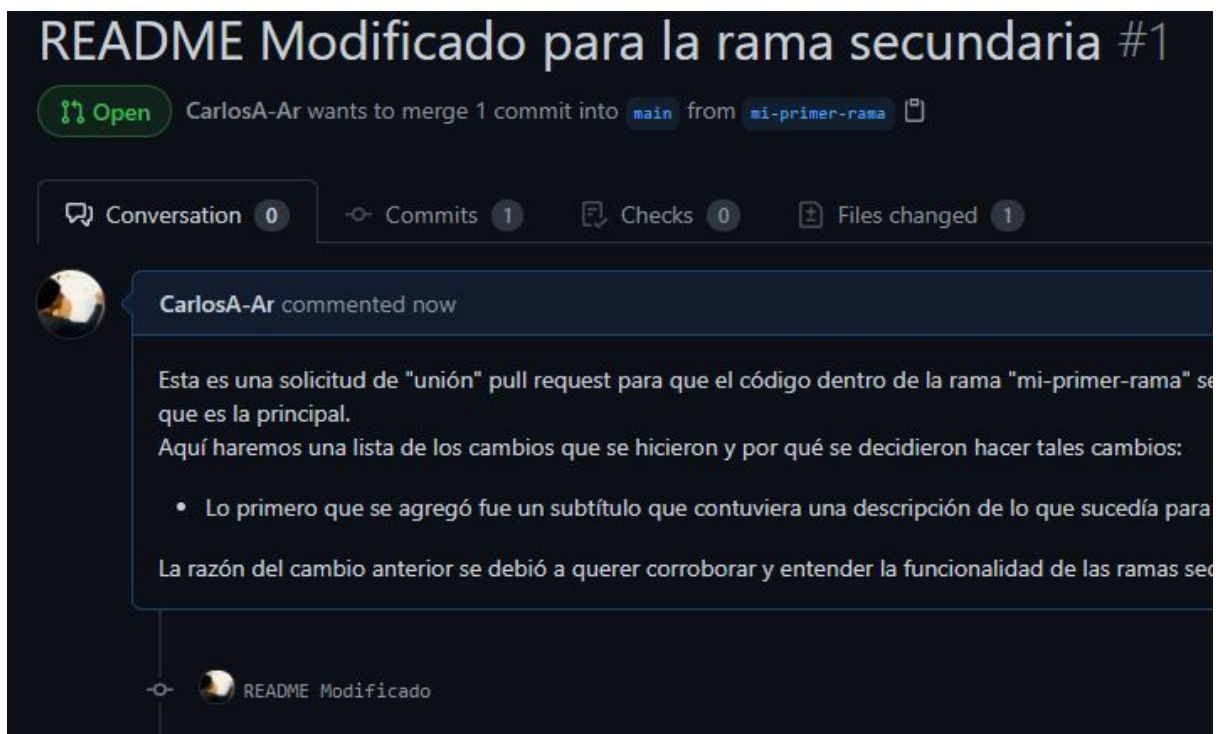
Aquí a diferencia de la vez pasada podemos notar que hay muchas cosas que notar:

1. Tenemos una sugerencia sobre una “solicitud de extracción” (pull request- PR) con la página que tiene el link <https://github.com/CarlosA-Ar/hello-world/pull/new/mi-primer-rama>
2. ¿Qué es una PR *Solicitud de extracción*? Es una solicitud para que tu código se lleve a otra rama. En este caso nosotros tenemos una rama secundaria y queremos solicitar que el código de la rama secundaria se extraiga a la rama principal.
3. Una vez que se tiene la PR, las demás personas podrán ver las modificaciones que has hecho en la rama principal, comentar en tu código y modificarlo, así que ya no sería necesario tener la branch que creaste y “juntaste” a la rama principal.
4. Si es necesario hacer otro cambio, entonces creamos otra branch específica.

5. ¿Cómo se hace una PR? Podemos copiar el enlace de la terminal en el buscador o ir a [github](#) directamente.

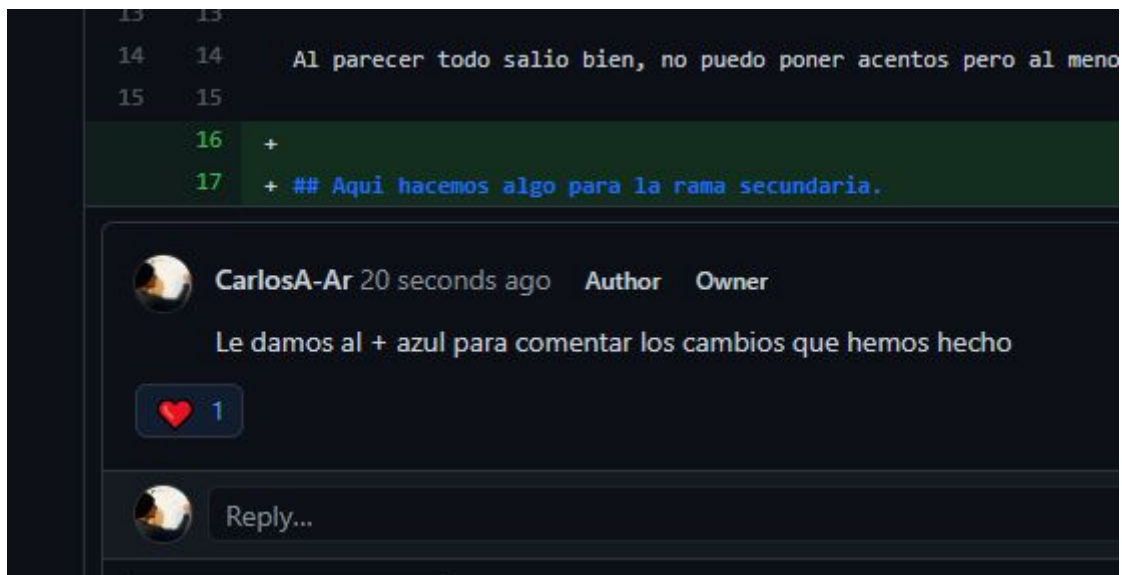


6. Lo que aparece es que pueden hacerse conversaciones sobre los cambios que se han hecho y [agregado](#):

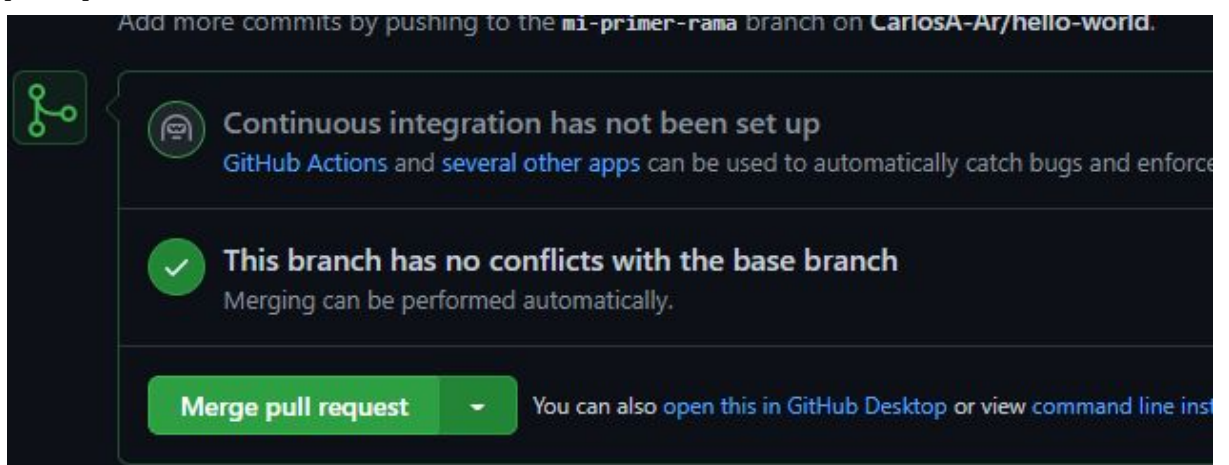


Podemos checar los cambios que se hicieron así como los comentarios:

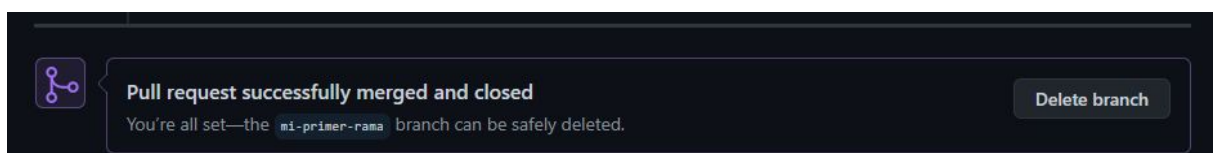




Si no tuviéramos los permisos necesarios para acceder a la rama principal, tendríamos que esperar a que se nos dé el permiso, para este ejemplo no es necesario y entonces aprobaremos la solicitud de la unión (entre la rama secundaria y la principal)



dando click en el boton verde y confirmando de nuevo



Notemos que ya estuvo el cambio:





Ahora que ya estuvo la juntazón, podemos usar pull para que se actualice el “juntado” en nuestra máquina local:

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de Ciencias/Servicio Social/Git and GitHub/hello-world (main)
$ git pull
remote: Enumerating objects: 1, done.
remote: Counting objects: 100% (1/1), done.
remote: Total 1 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (1/1), 657 bytes | 59.00 KiB/s, done.
From https://github.com/CarlosA-Ar/hello-world
   bac566e..f770331  main       -> origin/main
Updating bac566e..f770331
Fast-forward
 README.md | 5 +++++
 1 file changed, 5 insertions(+)
```

Como ya no se usan las ramas que se unieron a la principal, entonces podemos borrarla con el comando ‘git branch -d mi-primer-rama’

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de Ciencias/Servicio Social/Git and GitHub/hello-world (main)
$ git branch -d mi-primer-rama
Deleted branch mi-primer-rama (was 9517e0b).

carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de Ciencias/Servicio Social/Git and GitHub/hello-world (main)
$ git branch
* main
```

**No todo es miel sobre hojuelas.** En los proyectos reales muchas personas hacemos cambios en el código de un mismo archivo y git no sabe bien si hay redundancia, o

errores de unión, etc. por lo cual debemos hacer lo siguiente para juntar sin tanto lío los cambios. Es un rollo que no me entendí bien. **Minuto 51.**

```
o Social/Git and GitHub/hello-world (main)
$ git checkout checar-la-union
error: Your local changes to the following files would be overwritten by checkout:
    README.md
Please commit your changes or stash them before you switch branches.
Aborting
```

Si se escribe en la misma línea de código va a aparecer un error, entonces hay que hacer cosas como “stashing” pero eso no es parte del video.

Si tratamos de unir las cosas que llevamos entonces nos aparecerá un conflicto porque estaríamos sobrescribiendo en una línea igual del código:

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de Ciencias Exactas y Naturales/Social/Git and GitHub/hello-world (checar-la-union)
$ git merge main
Auto-merging README.md
CONFLICT (content): Merge conflict in README.md
Automatic merge failed; fix conflicts and then commit the result.
```

Hay muchas formas de resolver el conflicto que en el código se ven así

```
## Aquí hacemos algo para la rama secundaria.

1. Abre index.html en tu browser

<<<<<<< HEAD
##Para checar la unión

Es importante verificar la forma de juntar las cosas, entonces hacemos otra rama
checadora.

=====
## Ahora sobrescribiré

para que sala un error y entonces no se pueda subir
>>>>>> main
|
```

Lo que aparece antes del main significa que ese cambio lo hicimos en la rama principal a diferencia de los otros.

Además los signos de igualdad implican que sobrescribimos el código y entonces podemos simplemente borrar esas cosas estorbosas:

```
1. Abre index.html en tu browser

##Para checar la unión

Es importante verificar la forma de juntar las cosas, entonces hacemos otra rama
checadora.

## Ahora sobrescribiré

para que sala un error y entonces no se pueda subir
```

**Tip:** si tenemos un archivo creado, no es necesario usar add y luego commit, si solo es un cambio en el archivo podemos usar -am

¿Cómo borramos los errores? undoing git.

Si por ejemplo añadimos al README.md la línea

```
1. Abre index.html en tu browser
2. Diviértete
```

que dice diviértete. Y por mensos usamos el add, pues podemos simplemente usar el comando 'git reset' para no guardar lo que habíamos agregado:

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de Cien
o Social/Git and GitHub/hello-world (checar-la-union)
$ git reset
Unstaged changes after reset:
M      README.md

carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de Cien
o Social/Git and GitHub/hello-world (checar-la-union)
$ git status
on branch checar-la-union
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directo
      modified:   README.md

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

Ahora qué pasa si vamos más allá y hacemos un commit, pues hacemos lo mismo pero ahora le añadimos la palabra HEAD~1 para que se borre el último commit hecho:

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Fac
o Social/Git and GitHub/hello-world (checar-la-union)
$ git reset HEAD~1
Unstaged changes after reset:
M      README.md
```

Posteriormente podemos checar ladiferencia de código para ver lo que le añadimos mal:

```
o Social/Git and GitHub/hello-world (chechar-la-union)
$ git diff
diff --git a/README.md b/README.md
index 500fb26..896
--- a/README.md
+++ b/README.md
@@ -17,6 +17,7 @@
os ya se pudo subir
## Aquí hacemos a

1. Abre index.html
+2. Divi<E9>rtete
```

y entonces borrarlo.

Si queremos borrar algún commit en articular no es tan sencillo como HEAD~1 pero sí que se puede mejorar el asunto con el comando 'git log' que te permite observar todos los commits que has hecho por orden cronológico.

Lo cual nos queda como

```
commit 71ac01031178dd61771a97913c129a26bd6ffb83
Author: CarlosA-Ar <carlos_actuaria@ciencias.unam.mx>
Date: Wed Feb 17 15:06:01 2021 -0600

    Añadimos el index.html

commit c2e8b2679617d349e4b262d5b7118eab95b348a4
Author: CarlosA-Ar <carlos_actuaria@ciencias.unam.mx>
Date: Wed Feb 17 14:18:25 2021 -0600

    Aquí se agrega un comentario del cambio.

commit 8cccc6df95efb27d68b900fc1058cb1ac45502c2
Author: CarlosA-Ar <75510715+CarlosA-Ar@users.noreply.github.com>
Date: Fri Dec 4 20:39:24 2020 -0600

    Initial commit
```

a lado de cada commit hay un código especial que nos ayuda a identificar lo que ha cambiado. Si queremos que ese cambio sea totalmente borrado usamos 'git --hard código\_único' por ejemplo:

```
carlo@LAPTOP-RRF4SPIO MINGW64 ~/OneDrive/Documentos/Facultad de Ciencias Exactas y Físicas/Social/Git and GitHub/hello-world (chechar-la-union)
$ git reset --hard b18f332097320710bdbb9cd239070a23dca1b231
HEAD is now at b18f332 Hicimos un cambio desde todos lados
```

Y ahora en el código tales cambios ya no aparecen:



```
la terminal de _Git Bash_  
  
## Aqui hacemos algo para la rama secundaria.  
  
1. Abre index.html en tu browser  
|  
##Para checar la unión  
  
Es importante verificar la forma de juntar las cosas, entonces hacemos otra rama  
checadora.
```

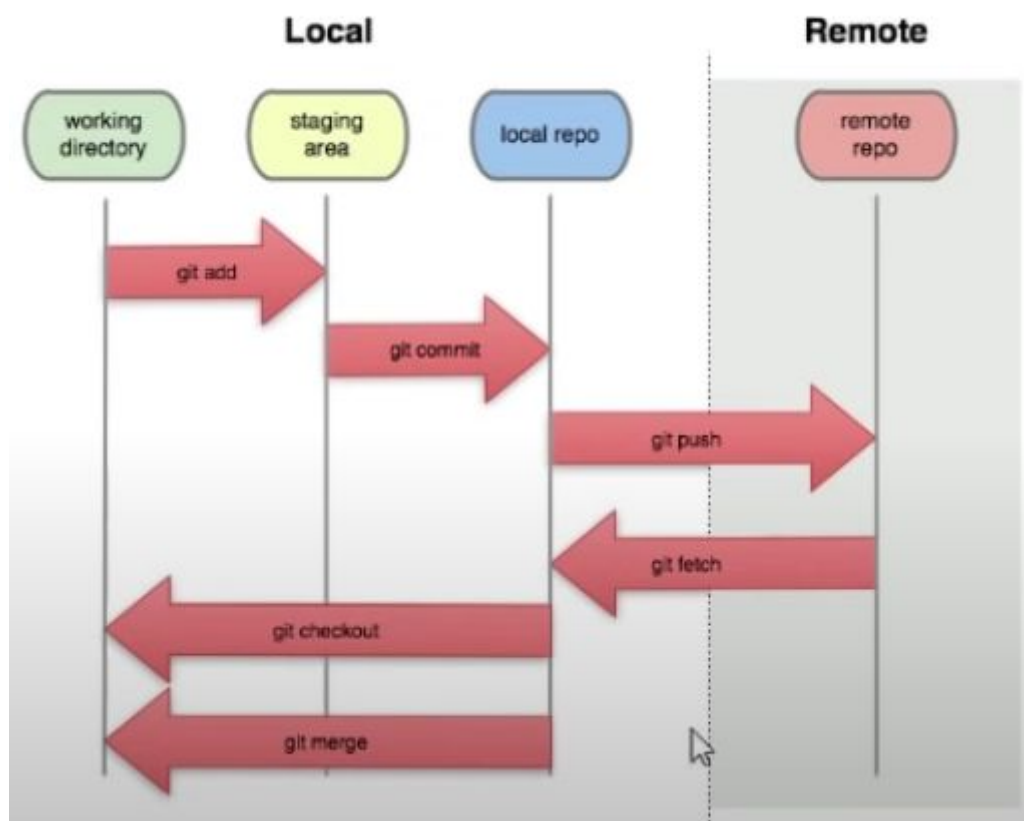
Puedo hacer un fork al respecto de los repositorios y “copiar” y editar como quiera un proyecto que ya ha estado trabajandose.

Second video: <https://youtu.be/hWglK8nWh60>

Themes:

✓ Lo mismo que el primer video más:

## Flujo de trabajo de GIT



Vienen en resumen las funciones:

git init (Empezamos a trabajar con git)

git add ("compilar" los cambios)



git commit (guardar los cambios en la parte temporal)

git log --oneline (historial de cambios)

git reset --hard (regresar/borrar)

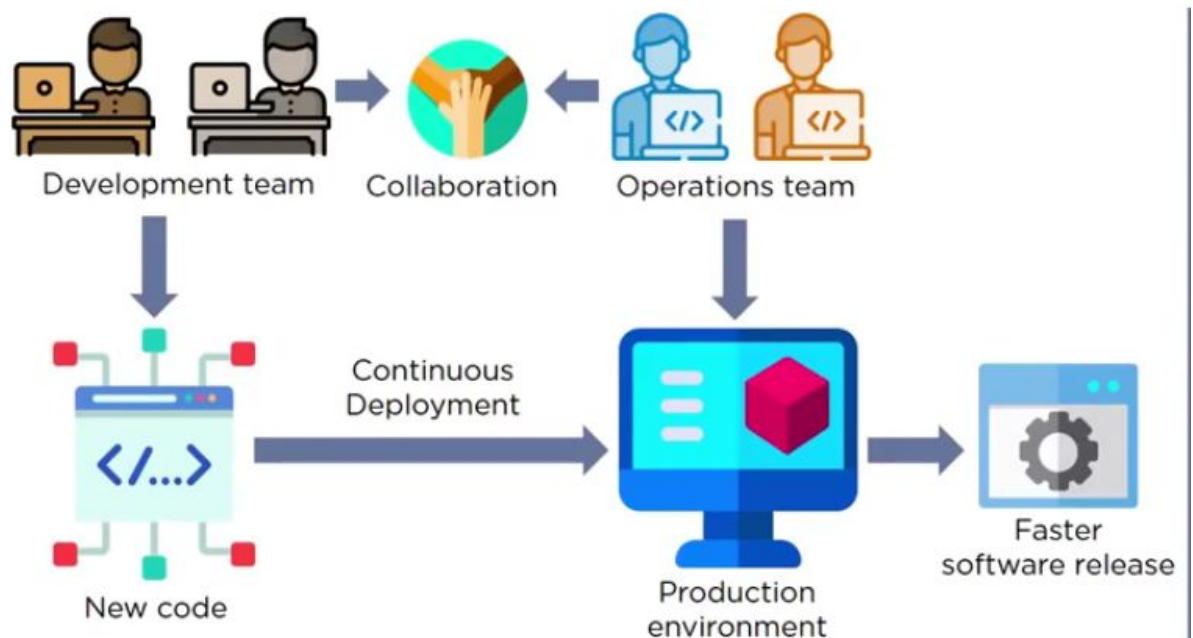
git status -s (Ver la modificación que se ha hecho en los archivos)

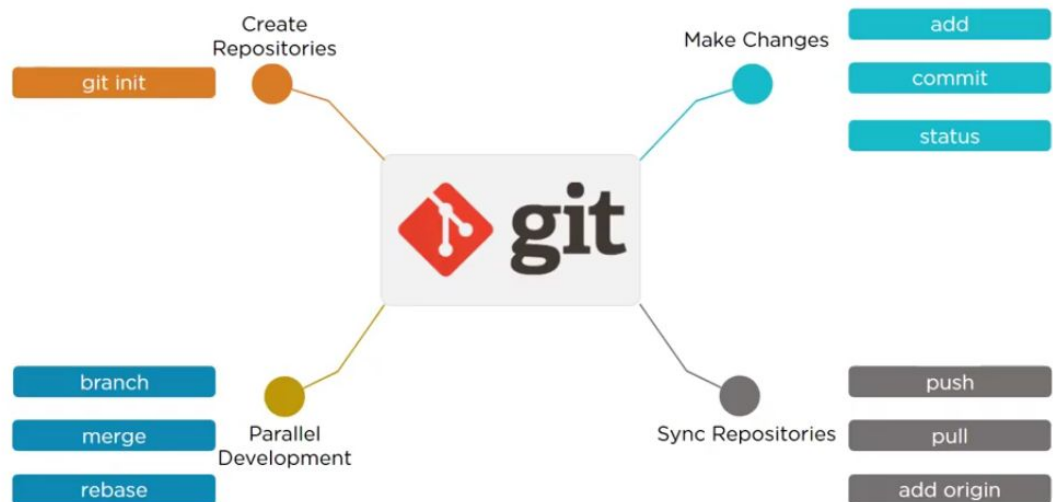
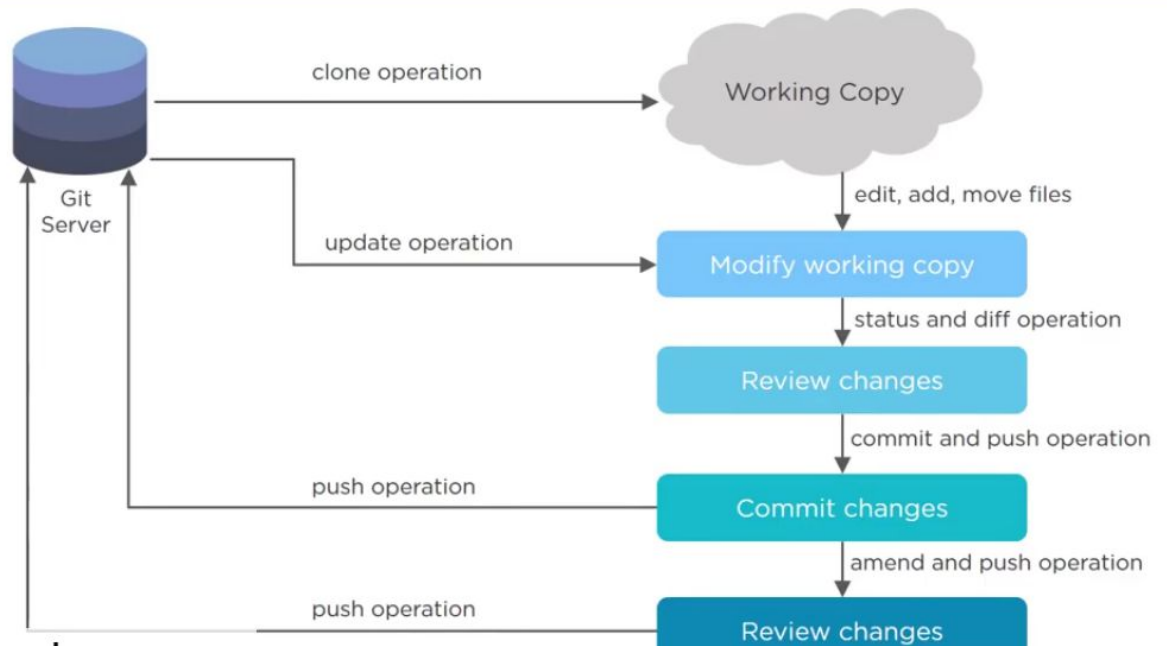
Da una idea de la diferencia real entre git y github. Todo lo de git se puede hacer inicialmente en la consola y luego ya compartirlo en github.

**Third video:** <https://youtu.be/liwv7Hi68aI>

Themes:

- ✓ Lo mismo que el primero en el sentido de la teoría, instalación, diferencias entre git y github y los comandos útiles.





`git help config` para saber las funciones de los comandos.

```
git config --global user.username simp
SPL-LP-DNS-YTO+Simplilearn@SSPL-LP-DNS-
git remote add origin https://github.c
```

El primero es para conectar con nuestro nombre de usuario en github. En vez de sim... va nuestro nombre de usuario en gitHub. Y en veez de https... iría el link de nuestro repositorio.

1. *Check the version of Git*

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~  
$ git --version  
git version 2.18.0.windows.1
```

2. *Set up some global config variables* -----> If you are working with other developers, you need to know who is checking the code in and out and making the changes

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~  
$ git config --global user.name "Sandeep.D"
```

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~  
$ git config --global user.email "sandeep.d@simplilearn.net"
```

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~  
$ git config --list
```

4. *Create a "test" repository in the local system*

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~  
$ mkdir test
```

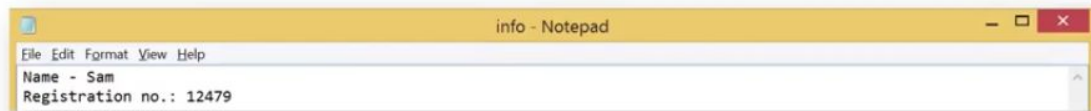
5. *Move to the test repository*

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~  
$ cd test
```

6. *Create a new git instance for a project*

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test  
$ git init  
Initialized empty Git repository in C:/Users/Simplilearn/test/.git/
```

7. Create a text file called `info.txt` in the test folder, write something and save it



8. Check the status of the repository

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ git status
On branch master

No commits yet

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

        info.txt

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

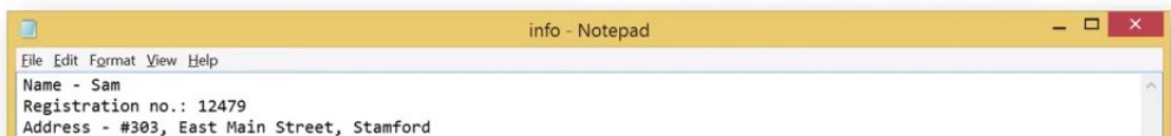
9. Add the file you created to make a commit

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ git add info.txt
```

10. Commit those changes to the repository's history with a short message

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ git commit -m "committing a text file"
[master (root-commit) 1c0673b] committing a text file
1 file changed, 2 insertions(+)
create mode 100644 info.txt
```

11. Make some changes to the file and save



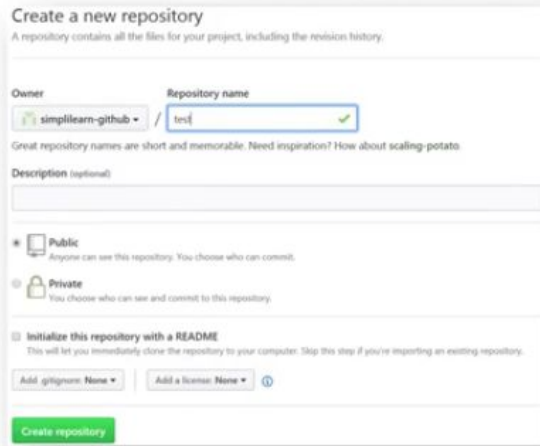
12. View the difference in file now and how it was at your last commit

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ git diff
diff --git a/info.txt b/info.txt
index Odd82a6..706570c 100644
--- a/info.txt
+++ b/info.txt
@@ -1,2 +1,6 @@
  Name - Sam
  Registration no.: 12479
+Address - #303, East Main Street, Stamford
```

### 13. Add GitHub username to Git Configuration

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ git config --global user.username simplilearn-github
```

### 14. Create a remote repository



Create a new repository

A repository contains all the files for your project, including the revision history.

Owner: simplilearn-github / Repository name: test

Great repository names are short and memorable. Need inspiration? How about scaling-potato

Description (optional):

☐ Public  
Anyone can see this repository. You choose who can control it.

☐ Private  
You choose who can see and control this repository.

☐ Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add a gitignore: None Add a license: None

Create repository

simplilearn All rights reserved

### 15. Connect the local repository to your remote repository

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ git remote add origin https://github.com/simplilearn-github/test.git
```

### 16. Push the file to the remote repository

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master)
$ git push origin master
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 270 bytes | 135.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0)
To https://github.com/simplilearn-github/test.git
* [new branch]      master -> master
```

### 17. Create 3 more text files in the local repository - "info1.txt", "info2.txt", "info3.txt"





18. Create a branch "first\_branch" and merge it to the main (master) branch

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master) -----> creating a branch
$ git branch first_branch
```

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (master) -----> switching to the new branch from
$ git checkout first_branch master branch
Switched to branch 'first_branch'
M      info.txt
```

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (first_branch) -----> create and add "info3.txt" to
$ git add info3.txt first_branch
```

18. Create a branch "first\_branch" and merge it to the main (master) branch

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (first_branch)
$ git commit -m "make some changes to first_branch"
[first_branch b14d609] make some changes to first_branch
Committer: Simplilearn <Simplilearn@SSPL-LP-DNS-YT01.b1rsimplilearn.com>
Your name and email address were configured automatically based
on your username and hostname. Please check that they are accurate.
You can suppress this message by setting them explicitly:

    git config --global user.name "Your Name"
    git config --global user.email you@example.com -----> make a commit to the first_branch
```

After doing this, you may fix the identity used for this commit with:

```
git commit --amend --reset-author
```

```
1 file changed, 1 insertion(+)
create mode 100644 info3.txt
```

```
SSPL-LP-DNS-YT0+Simplilearn@SSPL-LP-DNS-YT01 MINGW64 ~/test (first_branch) -----> new branch has access to
$ ls all the files
info.txt info1.txt info2.txt info3.txt
```

El comando 'touch nombredelarchivo.rmd' sirve para crear archivos.

LOS COMANDOS

```
git init
```

```
git add .
```

```
git commit -m "Message"
```

```
git status
```

```
git log
```

```
git add remote origin Server_Repo_URL
```

```
git push -u origin master
```

```
git clone Server_Repo_URL
```

**DUDA:**

Cuál es la diferencia entre trabajar con un repositorio HTML y uno SSH.

¿Qué hacer con archivos commits que no se quieren?

```
git push origin --delete Blr_Branch
```

Este comando sirve para borrar la rama que se llama *Blr\_Branch* del github. Porque uno la puede borrar localmente y existir apun en el servidor de github,

```
git branch New_Branch
```

```
git checkout New_Branch
```

```
git push -u origin New_Branch
```

```
git branch -av
```

```
git merge New_Branch
```

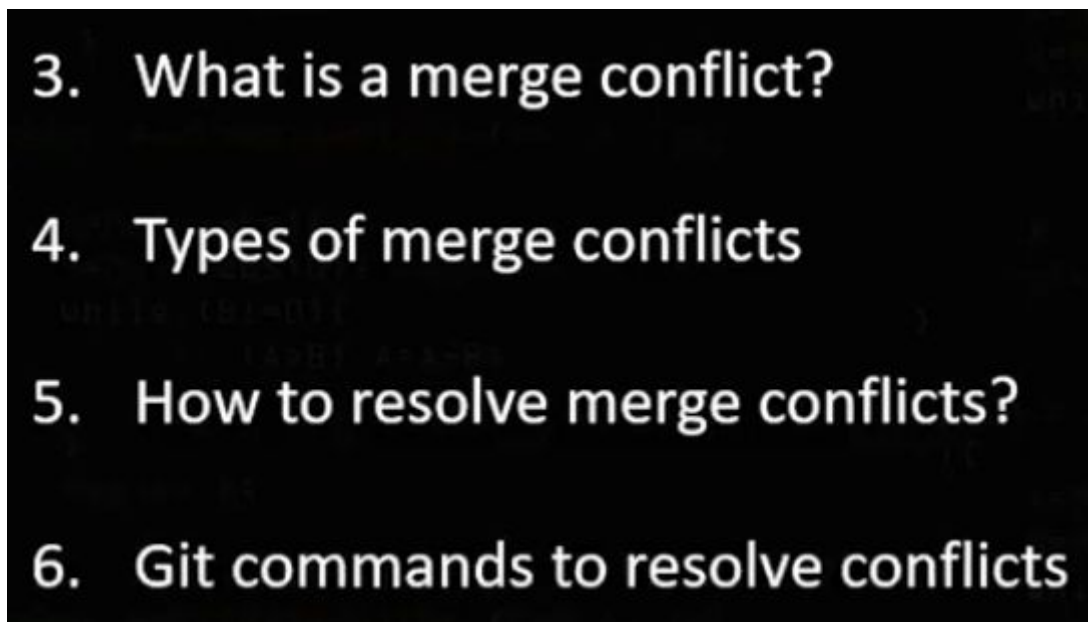
```
git branch -D New_Branch
```

```
git push origin -delete New_Branch
```

Git config	Configure the username and email address
Git init	Initialize a local Git repository
Git add	Add one or more files to staging area
Git diff	View the changes made to the file
Git commit	Commit changes to head but not to the remote repository

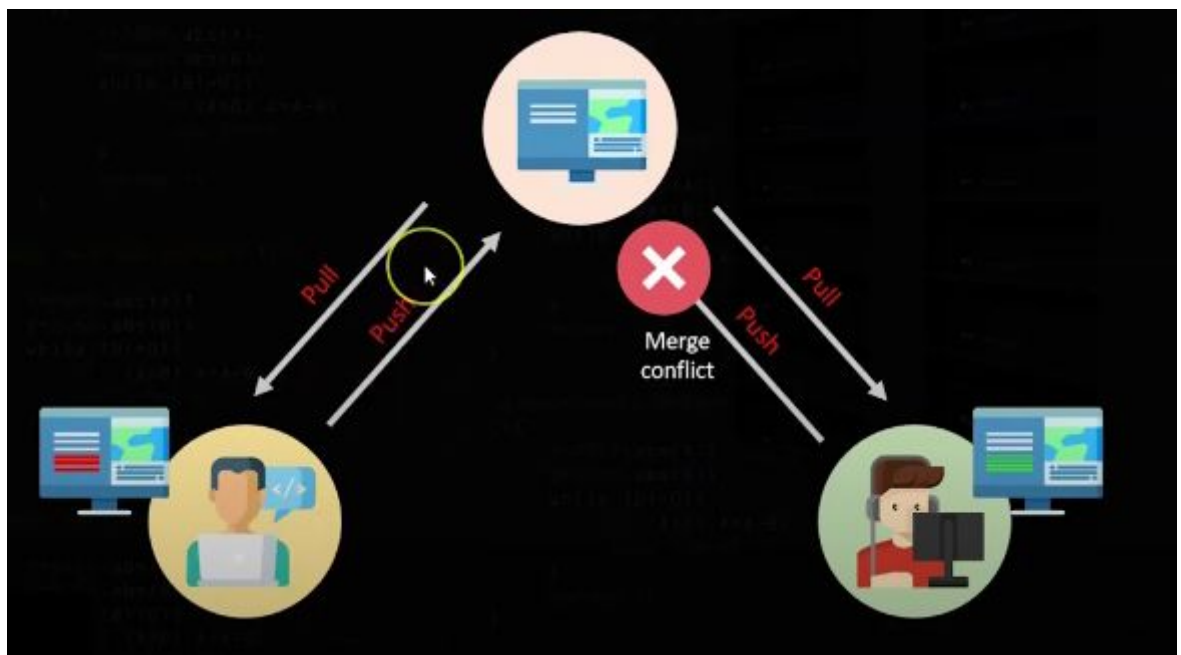
Git reset	Undo local changes to the state of a Git repo
Git status	Displays the state of the working directory and staging area
Git merge	Merge a branch into an active branch
Git push	Upload content from local repository to a remote repository
Git pull	Fetch and download content from a remote repository

✓ Qué hacer para subir versiones del trabajo sin conflictos.



1. ¿Qué es un error de “fusión” (merge)?

Se genera cuando hacemos alguna actividad relacionada con la fusión entre los archivos que se han actualizado (y que se han subido o descargado a github, por ejemplo). Estas acciones son pull, push (jalar las nuevas versiones que pueden sobre escribir en las líneas de código existentes o subir versiones que no empatan con las que ya estaban en la plataforma).



Pero la definición es que se da un error de fusión cuando se quiere unir una de las ramas secundarias a la rama principal y las secundarias tienen código que no compila con la rama principal.

2. Tipos de conflictos que ocurren en la “fusión”.



- a. Al iniciar el proceso de fusión (merge process)  
Si hay cambios en el área de trabajo (directory) en un proyecto, la fusión no podrá comenzar.  
Los conflictos se dan cuando los cambios no se pueden actualizar.
- b. Durante el proceso de fusión.  
Hay un conflicto entre la rama local y la que se quiere fusionar.  
No todo lo puede resolver git.
3. ¿Cómo resolver los problemas de “fusión”?
  - a. Abrir el archivo y ver línea por línea los problemas posibles y hacer los cambios necesarios para que funcione. Luego añadir los cambios con **git add** y **git commit**. *No es lo más óptimo.*
  - b. Usar comandos de git para la fusión.
4. Comandos de Git para resolver conflictos de “fusión”.
  - a. **git log --merge**. Hace una lista de los “commits” que causan el conflicto de fusión.
  - b. **git diff**. Permite encontrar las diferencias entre las ramas y sus archivos en un repositorio.
  - c. **git checkout**. Para deshacer los cambios al archivo o cambiar de rama.
  - d. **git reset --mixed**. Deshacer los cambios en la carpeta y en el área de trabajo.
  - e. **git merge --abort**. Para salir del proceso de fusión y regresar al área de trabajo que se tenía antes de iniciar la fusión.
  - f. **git reset**. Eliminar las cosas que causan los conflictos.

**Ejemplo (demostrativo):**

`git pull --rebase origin master`

`git rebase --continue`

**NO ENTIENDO:** cómo manejar los errores de merge. En las 2 horas y min me hice bolas.

*There are two processes through which you can revert a commit:*

1

Remove or fix the bad file in a new commit and push it to the remote repository. Then commit it to the remote repository using:

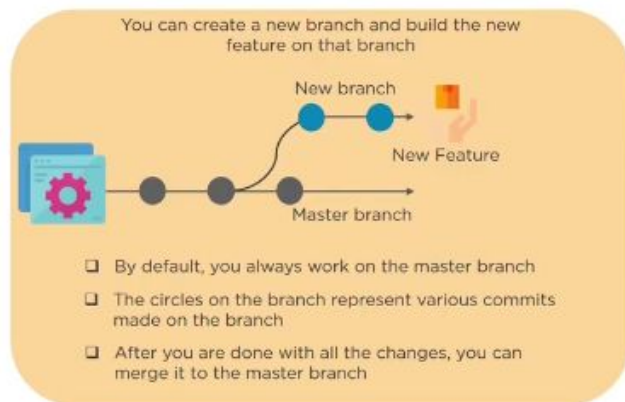
```
git commit -m "commit message"
```

2

Create a new commit that undoes all the changes that were made in the bad commit. Use the following command:

```
git revert <commit id>
```

*Example: git revert 56de0938f*

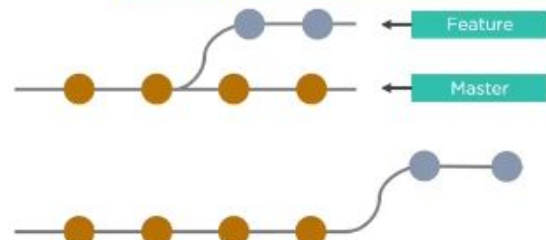


Suppose you are working on a new feature in a dedicated branch and another team member updates the master branch with new commits



- Creates an extra merge commit every time you need to incorporate changes
- Pollutes your feature branch history

As an alternative to merging, you can rebase the feature branch onto master




- Incorporates all the new commits in master branch
- Re-writes the project history by creating brand new commits for each commit in the original branch







***git diff-tree -r {commit hash}***

***Example: git diff-tree -r 87e673f21b***

- ***-r*** flag allows the command to list individual files
- ***commit hash*** will list all the files that were changed or added in that commit

### Resolving a merge conflict using conflict editor

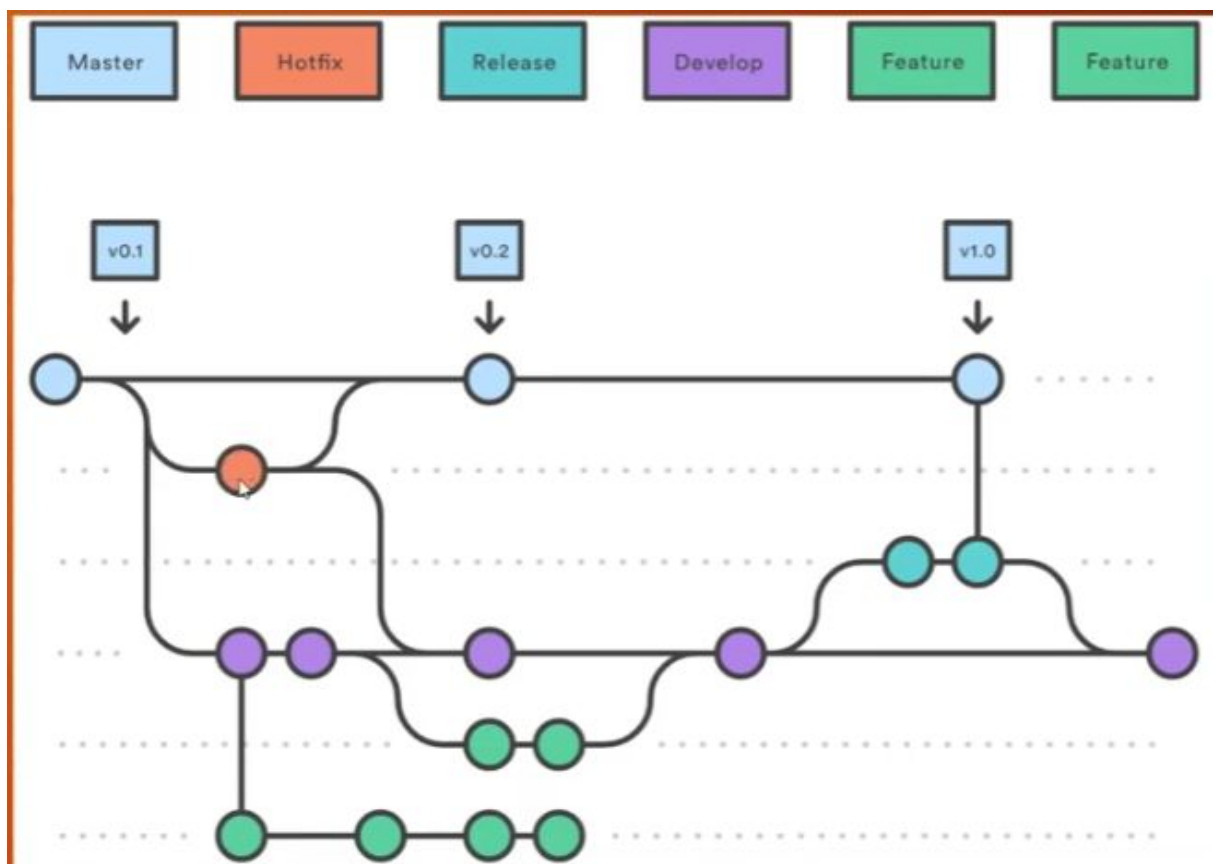
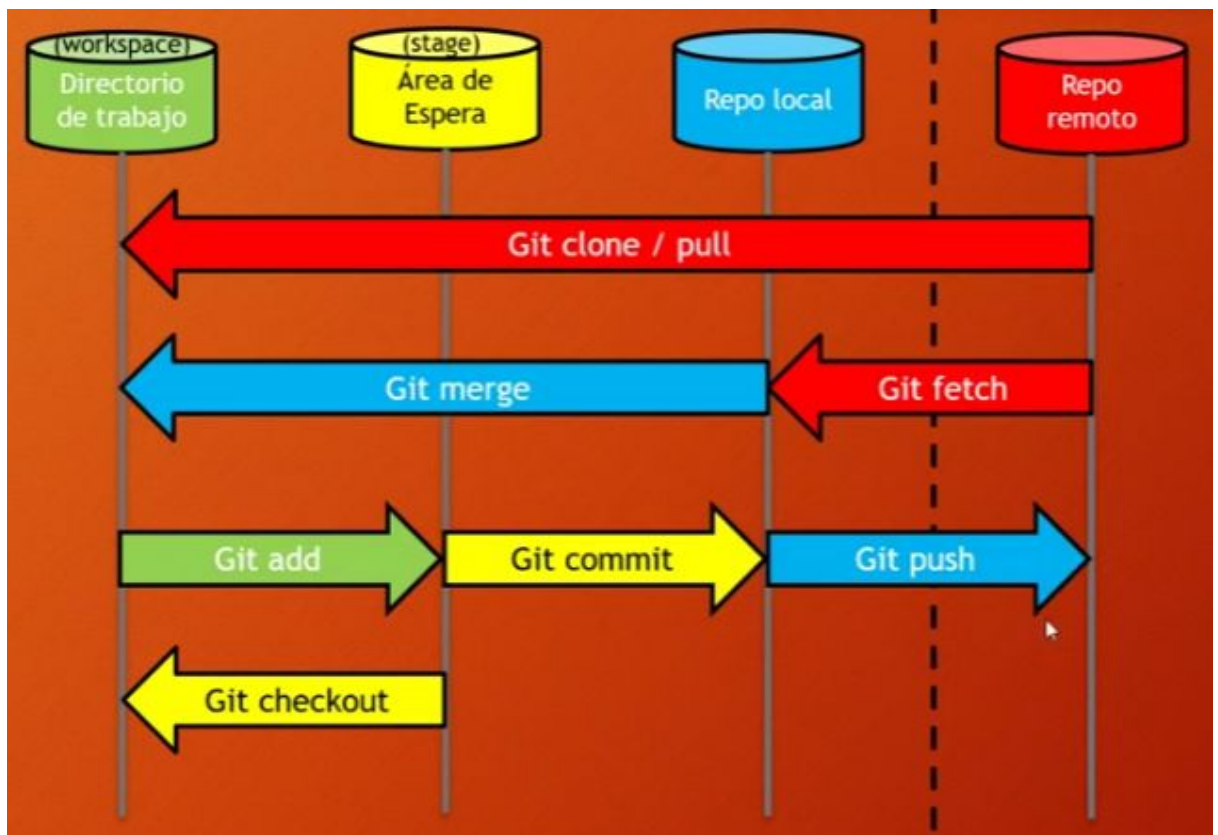
- 1 Under your repository name, click  Pull requests.
- 2 In "Pull Requests" list, click the pull request with a merge conflict that you'd like to resolve
- 3 Near the bottom of your pull request, click Resolve conflicts
- 4 Decide if you want to keep only your branch's changes, keep only the other branch's changes or make a brand new change, which may incorporate changes from both branches
- 5 Delete the conflict markers `<<<<<<<<`, `=====`, `>>>>>>>` and make changes you want in the final merge
- 6 If you have more than one merge conflict in your file, scroll down to the next set of conflict markers and repeat steps four and five to resolve your merge conflict
- 7 Once you have resolved all the conflicts in the file, click Mark as resolved
- 8 If you have more than one file with a conflict, select the next file you want to edit on the left side of the page under "conflicting files" and repeat steps 4 to 7 until you've resolved all of your pull request's merge conflicts
- 9 Once you've resolved all your merge conflicts, click **Commit merge**. This merges the entire base branch into your head branch
- 10 To merge your pull request, click Merge pull request



Fourth video: <https://youtu.be/97mY9uWL-Kc>

Themes:

- ✓ Controlar los cambios que se hacen a los proyectos de versiones.



- ✓ GitLab. Como GitHub
- ✓ GitKraken. Para ver los cambios y no solo todo en consola.

**Fifth video:** <https://youtu.be/z1CpcG-vs8U>

Themes:

- ✓ GitKraken: es una herramienta más completa que gitHub. Nos permite ver líneas de tiempo para gestionar las ramas locales y remotas.  
Permite visualizar mucho mejor las distintas ramas que se han creado a lo largo del proyecto de forma que no sea todo en consola.

**Sixth video:** <https://youtu.be/EscDe0jG6XM>

Themes:

- ✓ Sobre las diferencias “medio bien explicadas” entre gitHub y GitLab

### **HACER UN RMarkdown que resuelva:**

¿Qué es?

¿Cómo se instala?

¿Cómo hacer una carpeta para que sea de git?

¿Qué es un repositorio?

¿Cómo se inicializa un repositorio nuevo?

¿Cuáles son las funciones principales: add, commit, status, push?

¿Qué es?

¿Cómo hacer github?

¿Cómo hacer un clone (descargar de github a compu?)

¿Cómo se cambia la cuenta o la contraseña para entrarle a GitHub y a Git solito?

En un proyecto trabajan mucha gente...

¿Cuál es la rama principal?

¿Qué es la rama secundaria?

¿Cómo se arreglan los errores?

¿Cómo se forkea un repo? Diferencia entre forkear y clonar.

Usar pull, push y pull request.



Avila Argüello Carlos

Para pullear y pushear desde R: <https://happygitwithr.com/> ver secciones **9 - 18**  
<https://youtu.be/rwryI4c7NMY>

¿Qué tendríamos que hacer para que todo se ponga en una página?

¿Cuáles son los pasos a seguir para hacer una página?