GUÍA RÁPIDA GIT / GITHUB

ÍNDICE

1. COMANDOS DE GIT

`git --version` Versión de GIT.

`git help` Ayuda.

`git help nombreComando` Para saber más sobre un comando.

Con los siguientes comandos, cambiaremos el nombre de usuario y la cuenta de mail, asociadas a la sesión (Windows/MacOs/Linux) :

`git config --global user.name "MiNombre"` Asignar un nombre de usuario.

`git config --global user.email "MiEmail"` Asignar un email.

`git config --global -e` abrimos la configuración en el editor.

`git config –global -l` sacamos un listado de la configuración por consola.

`git config --global core.editor \"C:\\Users\\Antonio\\AppData\\Local\\atom\\app-1.43.0\\atom.exe\" –wait` Ruta dónde tenemos el editor de texto

`git init` Con este comando creamos un repositorio local, para que pueda llevar los cambios de todo lo que vamos haciendo en el repositorio.

`git status` Para ver los archivos que están sincronizados o no, aparecerán todos los archivos que hayan sufrido algún tipo de modificación.

`git status -s` para ver los archivos que están en el stage o no.

Staging area, es un archivo simple, contenido en nuestro directorio GIT, que almacena la información que irá en nuestro siguiente commit. Antes de hacer el commit, podemos seleccionar que archivos de la escena stagin área vamos a subir o no.

`git add .` (significa todos los cambios) o bien `git add NombreArchivo` Añado todos los archivos al stage, posteriormente los tendré que hacer un commit para sincronizar.

`git add *.png` agrega todos los archivos png

`git add ccs/` agrega el directorio ccs

`git add -A` o `--all` añado todos los archivos que se hayan modificado

`git add "*.txt"` agrega todos los txt de TODO el proyecto.

`git add *.txt` agrega todos los txt del directorio actual.

`git add -u` actualizar.

`git commit -m "Nombre Que queramos"` – Guarda una captura del proyecto, en ese mismo instante, es conveniente que en el nombre pongamos algo identificativo.
`-am` agrega todos los archivos que haya al Stage.
`--amend -m "Nombre nuevo mensaje"` - Cambia el nombre del mensaje de commit.

`git checkout -- .` Reconstruye todo como estaba en el último commit, si no pone nada es que lo entendió correctamente.

`git checkout – nombreArchivo` Reconstruye todo como estaba en el último commit dentro de nombre archivo o **`git checkout codigoHash nombreArchivo.`**

`git log` muestra todos los commit hechos, mostrando el autor y la fecha

`git log --oneline` muestra todos los commit hechos, mostrando solo su hash corto y en una línea.

`ls -al` , buscamos en el directorio si existe un archivo .git, para ver si existe un repositorio de git o no.

`git config –global alias.s "status -s -b"` creamos un alias llamado s, dónde se ejecuta status -s -b

`git diff` muestra los cambios que ha habido en los diferentes archivos, es decir entre el último commit y el archivo actual.

`git diff --staged` comprueba las modificaciones de los archivos que están en el stage.

`git diff nombre_rama master` muestra las diferencias entre la rama "nombre_rama" y master.

`git reset *.xml` (o lo que sea) quito del stage todos los archivos que terminen en XML
`--soft HEAD^` si quiero por ejemplo modificar un archivo anterior al último, para posteriormente incluirlo en el último commit, es decir para deshacer un commit.

`git reset --mixed 8923c3` (hash del commit) quita desde ese commit, hacia arriba todo del stage, es decir desde esa fecha temporal asociada al commit a lo más reciente, no borra los cambios de los archivos de esos commit, solo los elimina del stage.

`git reset --hard 71a02b0`

`git reflog` Comando dónde se puede ver todo lo que ocurre en el registro.

`git mv nombre-antiguo.txt nombre-nuevo.txt` Cambiar nombre de un archivo desde git

`git rm salvar-mundo.txt` `Borramos un archivo.`

`git ignore` Se utiliza para especificar que archivos queremos que no se haga un seguimiento, es decir que no se sincronicen. Para ello, creamos este archivo en la raíz de la carpeta que tengamos sincronizada con git, y pondremos dentro del fichero (en cada línea), que archivos queremos excluir. Ejemplo:

*.log (aquí excluiremos todos los archivos que tengan la extensión log)
node_modules/ (aquí excluiremos la carpeta node_modules)

`git branch nombre_rama` Para crear una rama nueva.

`git branch` ver las ramas que hay creadas.

`git branch -d nombre_rama` borra la rama "nombre_rama"

`git checkout nombre_rama` Cambiar a la rama "nombre_rama".

`git checkout -b nombre_rama` creamos una rama y nos movemos hacia ella.

`git merge nombre_rama` une la rama "nombre_rama" a la rama que esté activa.

`git tag nombre_release` creamos un tag o una etiqueta con "nombre_release", esto se utiliza para anotar el control de versiones, ejemplo: App_alpha_0.1

`git tag -d nombre_release` borra un tag.

`git tag` vemos todos los tags que tenemos.

`git tag -a v1.0.0 -m "Versión 1.0.0"` -a "v1.0.0" aparecerá cuando hagamos git log en el log. -m "Versión 1.0.0" será el mensaje del commit

`git tag -a v0.1.0 334d7i – m "Versión alfa"` agregamos al hash de un commit "334d7i" , el nombre de versión v0.1.0

`git show v1.0.0` veo la información del tag "v1.0.0"

Stash (sirve para dejar una especie copia fija en el Stage) es una especie de almacén temporal, dónde se pueden dejar los cambios que hemos hecho y retomar un proyecto previo a un commit, posteriormente, se podrán retomar los cambios que habían guardados en el Stash y volver dónde lo dejamos.

`git stash` o `git stash save` guardamos todo lo que hayamos hecho entre el último commit y el stash

`git stash pop` recuperamos todos los cambios que teníamos en el stash.

`git stash list` lista los stash que hay almacenados.

`git stash apply` restaura el último registro en el stash, si hay varios, hacemos un stash list y ponemos la opción entre llaves que queramos recuperar.

`git stash apply stash@{0}` restaura el registro de stash@{0}.

`git stash drop` borra el último registro que está en el stash, si hay varios, seleccionamos uno el número que queramos para borrar.

`git stash drop stash@{0}` borra el registro de stash@{0}

`git stash save --keep -index` guarda todos los archivos menos lo que están en el stage.

`git stash save --include -untracked` Incluye todos los archivos, junto a los que git no le da seguimiento.

`git stash save “nombre del cambio que queramos guardar”`

`git show stash` no muestra información de los cambios de todo lo que se hizo en ese stash.

`git show stash@{1}` nos muestra la información de cambios del stash@{1}

`git stash clear` borra todas las entradas que hay en el stash.

`git rebase master` se utiliza para poner por delante de “master” los cambios hechos en otra rama y que así queden en la misma línea temporal. Para unir las dos ramas en una, habrá que hacer un `git merge rama2` y luego tendremos que borrar la rama2 (`git branch -d rama2`)

`Squash` se utiliza para fusionar varios commit

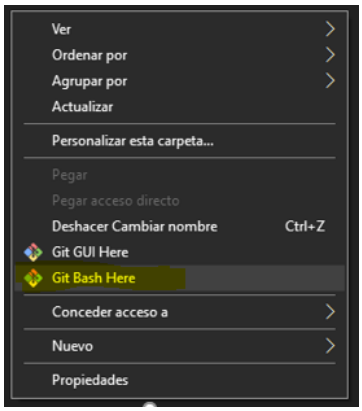
`git rebase -i HEAD~4` une los 4 últimos commit de forma interactiva “-i”

`reword` se usa para reescribir un squash, se utiliza dentro del squash, cuando nos aparece todas las opciones en el editor, pulsamos “a”, escribimos “r” o “reword”, hacemos wq para guardar cambios y nos saldrá en la siguiente pantalla el promp para que podamos hacer la modificación.

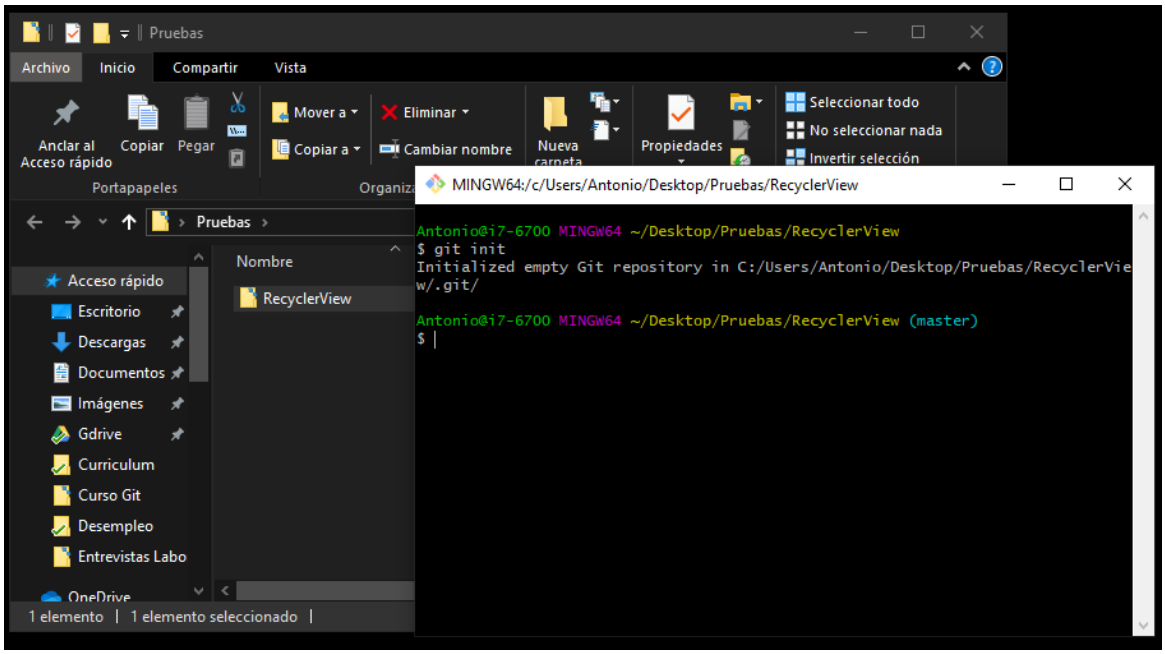
2. EJEMPLOS DE USO EN GIT

3.1. AGREGAR UN PROYECTO A GIT

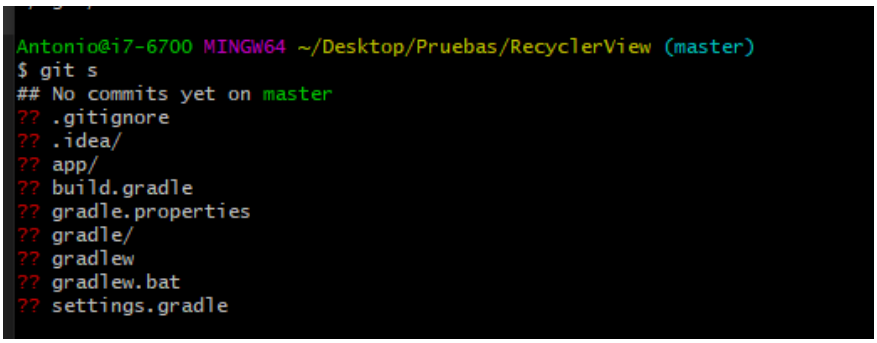
En la carpeta del proyecto que queramos inicializar git, hacemos clic (botón derecho) en el menú contextual "`Git bash here`", esa selección, abrirá la consola en esa ruta.



Posteriormente, hacemos un "`git init`" para inicializar git en ese directorio.



Hacemos un "`git status -s`" o un "`git s`" para ver los archivos que están sincronizados en el stage, como no hemos añadido ninguno, saldrá lo siguiente:



Añadimos todos los archivos mediante "`git add .`" para darle seguimiento de cambios.

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git add .
warning: LF will be replaced by CRLF in app/src/main/res/drawable/icono_borrar.x
ml.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in app/src/main/res/drawable/icono_editar.x
ml.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in app/src/main/res/drawable/icono_guardar.
xml.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in app/src/main/res/drawable/icono_nuevo.xm
l.
The file will have its original line endings in your working directory
warning: LF will be replaced by CRLF in gradlew.
The file will have its original line endings in your working directory
```

La advertencia, es generada por una de las opciones de configuración de git, la opción core.autocrlf.

Lo que hace esa opción, es indicar si quieres que en el repositorio se guarden los saltos de línea de los ficheros en formato Unix (LF), pese a tener tus ficheros en tu entorno local con saltos de línea al formato Windows (CRLF).

Para evitar esto, git tiene esta funcionalidad (core.autocrlf) que se encarga de convertir los saltos de línea a LF en todos los ficheros de texto del repositorio.

Esa opción la puedes configurar de varias formas:

- core.autocrlf = true: Cuando commitees, tus ficheros se transformarán automáticamente a LF, y cuando hagas checkout de un fichero, se convertirá automáticamente a CRLF
- core.autocrlf = input: Cuando comitees, tus ficheros se transformarán automáticamente a LF, pero cuando hagas checkout, recibirás el fichero sin modificación de como esté en el repositorio.
- core.autocrlf = false: No se hará ningún cambio a los finales de línea de los ficheros de texto.

Para configurar una opción, introducimos en consola `git config core.autocrlf true`

Comprobamos con "`git status -s`" que los ficheros se han añadido correctamente.

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git s
## No commits yet on master
A  .gitignore
A  .idea/codeStyles/Project.xml
A  .idea/gradle.xml
A  .idea/misc.xml
A  .idea/render.experimental.xml
A  .idea/runConfigurations.xml
A  app/.gitignore
A  app/build.gradle
A  app/proguard-rules.pro
A  app/src/androidTest/java/com/example/recyclerview/ExampleInstrumentedTest.jav
a
A  app/src/main/AndroidManifest.xml
A  app/src/main/java/com/example/recyclerview/AdaptadorRecyclerView.java
```

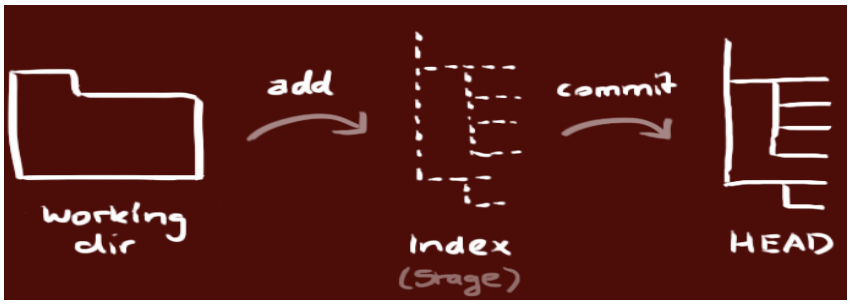
Hacemos un commit para subir todo del stage a nuestro repositorio
`git commit -am “Primer commit”`

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git commit -am "Primer Commit"
[master (root-commit) 2dfbc17] Primer Commit
 64 files changed, 1662 insertions(+)
 create mode 100644 .gitignore
 create mode 100644 .idea/codeStyles/Project.xml
 create mode 100644 .idea/gradle.xml
 create mode 100644 .idea/misc.xml
 create mode 100644 .idea/render.experimental.xml
 create mode 100644 .idea/runConfigurations.xml
 create mode 100644 app/.gitignore
 create mode 100644 app/build.gradle
 create mode 100644 app/proguard-rules.pro
 create mode 100644 app/src/androidTest/java/com/example/recyclerview/ExampleIns
trumentedTest.java
 create mode 100644 app/src/main/AndroidManifest.xml
 create mode 100644 app/src/main/java/com/example/recyclerview/AdaptadorRecycler
View.java
 create mode 100644 app/src/main/java/com/example/recyclerview/MainActivity.java
 create mode 100644 app/src/main/java/com/example/recyclerview/Persona.java
 create mode 100644 app/src/main/java/com/example/recyclerview/PersonaActivity.j
ava
 create mode 100644 app/src/main/java/com/example/recyclerview/PersonaNueva.java
 create mode 100644 app/src/main/res/drawable-v24/foto01.jpg
 create mode 100644 app/src/main/res/drawable-v24/foto02.jpg
 create mode 100644 app/src/main/res/drawable-v24/foto03.jpg
 create mode 100644 app/src/main/res/drawable-v24/foto04.jpg
 create mode 100644 app/src/main/res/drawable-v24/foto05.jpg
 create mode 100644 app/src/main/res/drawable-v24/foto06.jpg
```

Resumen:

1. Inicializamos nuestro proyecto `git init`
2. Comprobamos que archivos están listos para agregarse al proyecto con `git status -s`
3. Agregamos los archivos deseados o todos mediante `git add .`
4. Hacemos un primer commit mediante `git commit -am “Primer commit”`

Nota
En este proceso, estaremos añadiendo todo a la rama master, que es la rama que se crea por defecto.
El Stage es un área temporal que se usa para trabajar, hasta que no se hace un commit, no se “sube” al head del proyecto.



3.2. VER COMMITES ANTERIORES

Hacemos un `git status -s` o git s y vemos que tenemos todo en la rama master y que no hay ningún cambio por guardar. (Que haya algo pendiente en el stage)

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git s
## master
```

Hacemos un `git log` para ver la información de commites.

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git log
commit 2dfbc1722912be25b0fb15036c59e3dcc37d0f89 (HEAD -> master)
Author: Antonio <antoniomy82@gmail.com>
Date:   Tue Apr 14 12:47:22 2020 +0200

    Primer Commit
```

o un `git log --oneline`

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git log -- oneline

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git log --oneline
389c72c (HEAD -> master) Añado archivo .idea/vcs.xml
f1ff139 Cambio de color
2dfbc17 Primer Commit
```

Para volver a un commit concreto hacemos lo siguiente `git checkout hashCommit`

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git checkout 2dfbc17
Note: switching to '2dfbc17'.

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

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

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

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

HEAD is now at 2dfbc17 Primer Commit
```

Para volver otra vez a la rama principal (master), dónde hicimos el último commit y puesto que estamos trabajando con la rama master, `git checkout master`

Nota: Cuando estemos dentro del commit, no podremos hacer modificaciones.

3.3. RESTAURAR UN "COMMIT" ANTERIOR Y DEJARLO COMO "HEAD"

Este supuesto lo usaremos para volver a una versión anterior guardada en un commit y dejarla como último commit, borrando todos los sucesivos (o posteriores) a este. Es decir, el código de dicho commit, será sobre el que trabajaremos, borrando todas las versiones posteriores a este.

Se hace un `git log --oneline` para ver todos los commites.

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git log --oneline
b12ddb6 (HEAD -> master) Colores originales
c0bf9a6 Warning resueltos y celdas en Rojo
2dfbc17 Primer Commit
```

Se quiere recuperar el commit `c0bf9a6` y dejarlo como último.

Para borrar todos los commits posteriores `git reset --mixed c0bf9a6`

Se hace un log para comprobar que se han efectuado los cambios y está como HEAD el commit deseado.

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git log --oneline
c0bf9a6 (HEAD -> master) Warning resueltos y celdas en Rojo
2dfbc17 Primer Commit
```

Con la opción `--mixed`, se resetea el índice, pero no el árbol de trabajo, por tanto el resto de trabajos (commites) no aparecerán, aunque sus archivos no se borran.

Con la opción `--hard`, se resetea el índice y además se borra el trabajo de los commites posteriores.

Más información `git reset --help`

Resumen:

1. Ver la lista de commites `git log –oneline`
2. Se selecciona un commit mediante su hash y se borran todos los posteriores : `git reset --mixed hashCommit`

3.4. RECUPERAR UN "COMMIT" O CUALQUIER OTRO CAMBIO

En este ejemplo, se quiere recuperar un commit que previamente hemos reseteado y no aparece en el log de commits.

También se puede utilizar para deshacer cualquier cambio hecho, por ejemplo recuperar una rama que hemos borrado.

Se ejecuta `git reflog` y posteriormente nos movemos con el cursor (dentro de la consola), hasta encontrar el commit deseado.

Para salir de la consola :q

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git reflog
c0bf9a6 (HEAD -> master) HEAD@{0}: checkout: moving from c0bf9a699af1cef2fe2589db9c1a3d483632178b to master
b12ddb6 HEAD@{24}: commit: Colores originales
c0bf9a6 (HEAD -> master) HEAD@{25}: checkout: moving from 2dfbc1722912be25b0fb15036c59e3dcc37d0f89 to master
2dfbc17 HEAD@{26}: checkout: moving from master to 2dfbc17
c0bf9a6 (HEAD -> master) HEAD@{27}: commit: Warning resueltos y celdas en Rojo
2dfbc17 HEAD@{28}: reset: moving to 2dfbc17
aab375a HEAD@{29}: commit: Ultimo cambio guardado - celdas en rojo
de5c9fd HEAD@{30}: checkout: moving from de5c9fde83baa1354caf916f76c06b54fa243852 to master
de5c9fd HEAD@{31}: rebase: checkout master
e460b2d HEAD@{32}: commit (merge): Union de dos commits resuelta
f957fa9 HEAD@{33}: commit: Colores iniciales con un comentario
69316ac HEAD@{34}: checkout: moving from master to 69316ac
de5c9fd HEAD@{35}: commit: Ponemos en rojo las celdas pares
69316ac HEAD@{36}: checkout: moving from 389c72c319f43e9db895aed0e3d00bf1bab543c6 to master
389c72c HEAD@{37}: checkout: moving from master to 389c72c
69316ac HEAD@{38}: checkout: moving from adbd081cfe34d84f02565c6cd4ca86c784e2b35f to master
adbd081 HEAD@{39}: commit: Vuelvo a poner los colores originales
389c72c HEAD@{40}: checkout: moving from master to 389c72c
69316ac HEAD@{41}: checkout: moving from 389c72c319f43e9db895aed0e3d00bf1bab543c6 to master
389c72c HEAD@{42}: checkout: moving from master to 389c72c
69316ac HEAD@{43}: commit: Cambio el color como estaba inicialmente
389c72c HEAD@{44}: checkout: moving from 2dfbc1722912be25b0fb15036c59e3dcc37d0f89 to master
2dfbc17 HEAD@{45}: checkout: moving from master to 2dfbc17
389c72c HEAD@{46}: checkout: moving from 2dfbc1722912be25b0fb15036c59e3dcc37d0f89 to master
2dfbc17 HEAD@{47}: checkout: moving from master to 2dfbc17
389c72c HEAD@{48}: commit: Añado archivo .idea/vcs.xml
f1ff139 HEAD@{49}: commit: Cambio de color
2dfbc17 HEAD@{50}: commit (initial): Primer Commit
```

o usar `git reflog --date=iso` para verlo en formato fecha, hora.

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git reflog --date=iso
0f48956 (HEAD -> master) HEAD@{2020-04-16 17:15:09 +0200}: commit (amend): Colores originales
047f74b HEAD@{2020-04-16 17:08:41 +0200}: commit (amend): Colores originales
b12ddb6 HEAD@{2020-04-15 19:21:42 +0200}: reset: moving to b12ddb6
c0bf9a6 HEAD@{2020-04-15 18:51:56 +0200}: reset: moving to master2
c0bf9a6 HEAD@{2020-04-15 18:51:15 +0200}: checkout: moving from master2 to master
c0bf9a6 HEAD@{2020-04-15 18:50:13 +0200}: checkout: moving from master to master2
```

Posteriormente, haremos un `git reset --hard hashDondeQueramosVolver`

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git reset --hard b12ddb6
HEAD is now at b12ddb6 Colores originales

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git log --oneline
b12ddb6 (HEAD -> master) Colores originales
c0bf9a6 Warning resueltos y celdas en Rojo
2dfbc17 Primer Commit
```

`git reflog` es un alias de `git log -g --abbrev-commit --pretty=oneline`
Es un comando en el que quedan registrados todos los cambios efectuados en git.

3.5. AÑADIR UN ARCHIVO AL ÚLTIMO COMMIT

Imagínese que se quiere añadir un nuevo archivo, o modificación al último commit.

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git s
## master
 M .idea/misc.xml

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git add .idea/misc.xml

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git s
## master
M  .idea/misc.xml
```

Usamos `git commit --amend --no-edit` para no cambiar el mensaje del `commit.`

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git commit --amend --no-edit
[master 0f48956] Colores originales
 Date: Tue Apr 14 18:15:11 2020 +0200
 1 file changed, 1 insertion(+), 1 deletion(-)

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git status s
On branch master
nothing to commit, working tree clean
```

3.6. CREAR UNA NUEVA RAMA DE UN "COMMIT" Y TRABAJAR EN ELLA

En el supuesto en el que se lleva trabajado en una modificación en concreto, y se quiere dejar para más adelante, y volver a retomar el trabajo en un punto anterior, se deberá crear una nueva rama, y hacer un commit determinado (p.e. el de la nueva modificación) a esa nueva rama. Posteriormente, seguiremos trabajando en la rama previa desde dónde lo dejamos.

Se crea la rama2 `git branch rama2`

Se revisan cuantas ramas hay `git branch`

Nos movemos hacia la rama2 `git checkout rama2`

Se revisa en qué rama estamos (aparece marcado en verde y con asterisco)

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git branch rama2

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git branch
* master
  rama2

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git checkout rama2
Switched to branch 'rama2'

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (rama2)
$ git branch
  master
* rama2
```

Hacemos cambios en nuestro código, o bien hacemos un commit para guardar el stage de ese punto en la rama2

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (rama2)
$ git commit -am "Cambiamos a gridLayaut de 3 celdas en Main y en AdaptadorRecyclerView"
[rama2 a2b12e3] Cambiamos a gridLayaut de 3 celdas en Main y en AdaptadorRecyclerView
 2 files changed, 5 insertions(+), 4 deletions(-)

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (rama2)
$ git log --oneline
a2b12e3 (HEAD -> rama2) Cambiamos a gridLayaut de 3 celdas en Main y en AdaptadorRecyclerView
df8e03b Cambiamos las celdas por azul marino
0f48956 (master) Colores originales
c0bf9a6 Warning resueltos y celdas en Rojo
2dfbc17 Primer Commit
```

Ahora podremos dejar este trabajo aparcado en la rama 2 y volver con lo que estábamos. Para volver a la rama master hacemos `git checkout master`

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (rama2)
$ git checkout master
Switched to branch 'master'

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git branch
* master
  rama2

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git log --oneline
0f48956 (HEAD -> master) Colores originales
c0bf9a6 Warning resueltos y celdas en Rojo
2dfbc17 Primer Commit
```

3.7. UNIR RAMAS "MERGE"

Imaginemos que se llega a un punto, en el que se quieren unir dos ramas, para ello se pueden dar 3 tipos de uniones:

A. Fast-Forward, se dispara cuando git detecta que en la nueva rama, no hay cambios con respecto a la principal y los cambios son integrados de forma transparente. (automáticamente, se puede desactivar esa función si no se quiere).

B. Unión automática, es cuando git detecta que hay cambios entre la rama principal y otra, pero se pueden unir en un punto sin que haya conflictos.

C. Unión con conflictos o manual, es cuando se realizan modificaciones en ambas ramas en las mismas líneas de código, por tanto tendremos que revisar a mano qué queremos dejar para unir ambas ramas.

Antes de hacer una unión o merge, podemos ver las diferencias que hay entre una rama y otra haciendo el siguiente comando: `git diff master rama2`

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git diff master rama2
diff --git a/app/src/main/java/com/example/recyclerview/AdaptadorRecyclerView.java b/app/src/main/java/com/exar
RecyclerView.java
index 4394dbf..e62de2a 100644
--- a/app/src/main/java/com/example/recyclerview/AdaptadorRecyclerView.java
+++ b/app/src/main/java/com/example/recyclerview/AdaptadorRecyclerView.java
@@ -31,7 +31,7 @@ public class AdaptadorRecyclerView extends RecyclerView.Adapter {
     //Aquí es dónde vamos a crear o inflar la vista
     public @NonNull RecyclerView.ViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {

-        View contentView = LayoutInflater.from(context).inflate(R.layout.recyclerview_item_lista, parent, fals
po de vista item_grid o Item_lista
+        View contentView = LayoutInflater.from(context).inflate(R.layout.recyclerview_item_grid, null); //Aqu
a item_grid o Item_lista
         //View contentView = LayoutInflater.from(context).inflate(esLista? R.layout.recyclerview_item_lista :
_grid,null); //Aquí cambiamos el tipo de vista item_grid o Item_lista
         System.out.println("Create View Holder: "+viewType);

diff --git a/app/src/main/java/com/example/recyclerview/MainActivity.java b/app/src/main/java/com/example/recyc
index 54c50a2..0c16ea9 100644
--- a/app/src/main/java/com/example/recyclerview/MainActivity.java
+++ b/app/src/main/java/com/example/recyclerview/MainActivity.java
@@ -8,6 +8,7 @@ import android.view.Menu;
 import android.view.MenuItem;
 import java.util.ArrayList;
 import androidx.appcompat.app.AppCompatActivity;
+import androidx.recyclerview.widget.GridLayoutManager;
 import androidx.recyclerview.widget.LinearLayoutManager;
 import androidx.recyclerview.widget.RecyclerView;

@@ -35,8 +36,8 @@ public class MainActivity extends AppCompatActivity  {


         //Habrá que cambiarlo más abajo, ya que dependiendo del Switch que usemos lo hará automaticamente.
-        manager = new LinearLayoutManager(this); //Mostrar como LinearLayout
-        //manager=new GridLayoutManager(this,3);
+        //manager = new LinearLayoutManager(this); //Mostrar como LinearLayout
+        manager=new GridLayoutManager(this,3);


         recyclerView.setLayoutManager(manager);

Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$
```

Para unir ambas ramas usamos, nos situaremos en la rama desde dónde queremos unir todo y haremos un `git merge rama2` (en el caso de ejemplo la rama2 se unirá a la master).

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git merge rama2
Updating 0f48956..a2b12e3
Fast-forward
 .../main/java/com/example/recyclerview/AdaptadorRecyclerView.java    | 2 +-
 app/src/main/java/com/example/recyclerview/MainActivity.java         | 5 +++--
 2 files changed, 4 insertions(+), 3 deletions(-)
```

Se ve como se han unido ambas ramas.

```
Antonio@i7-6700 MINGW64 ~/Desktop/Pruebas/RecyclerView (master)
$ git log --oneline
a2b12e3 (HEAD -> master, rama2) Cambiamos a gridLayaut de 3 celdas en Main y en AdaptadorRecyclerView
df8e03b Cambiamos las celdas por azul marino
0f48956 Colores originales
c0bf9a6 Warning resueltos y celdas en Rojo
2dfbc17 Primer Commit
```

Para evitar conflictos, borraremos la rama que no vayamos a usar `git branch -d rama2`

3. GITHUB

`git remote add origin https://github.com/miEjemplo.git` agregar un origen remoto, dónde origin es el nombre que le queremos dar a nuestro repositorio, https://.... Es la dirección dónde va a estar nuestro repositorio.

`git remote -v` esto es para ver los origines remotos que tenemos agregadas al repositorio.

`git push -u origin master` establecemos por defecto (-u) la rama (master) y el repositorio (origin) dónde vamos a hacer los push por defecto.

`git push --tags` sube todas las versiones o tags que tenemos a github

`git pull origin master` (git pull si hemos hecho antes -u origin master) actualiza en nuestro PC todos los cambios que se hayan hecho en github

`git clone https://github.com/....miProyecto` copiamos todo el repositorio en el PC, en la ruta desde dónde lancemos el comando.

`git clone https://github.com/....miProyecto CarpetaProyecto` igual que el anterior, pero definimos el nombre de la carpeta dónde se va a clonar todo el contenido del repositorio.

`git fetch` no hace un merge de forma automática, actualiza localmente todos los cambios que haya habido en el repositorio remoto.

`git push` sube todos los cambios locales a github

Markdown lenguaje de maquetación de documentación de Github:
https://www.markdowntutorial.com/
https://guides.github.com/pdfs/markdown-cheatsheet-online.pdf

En github

- Raw es para ver los archivos en texto plano.
- Blame es para ver quién ha hecho modificaciones sobre un archivo.
- History qué commits afectaron a un archivo en particular.

`Fork` es una rama adicional, en la cual podemos trabajar, pero en un momento dado puede ser unida a la rama padre por parte del propietario del repositorio mediante un `Pull Request`

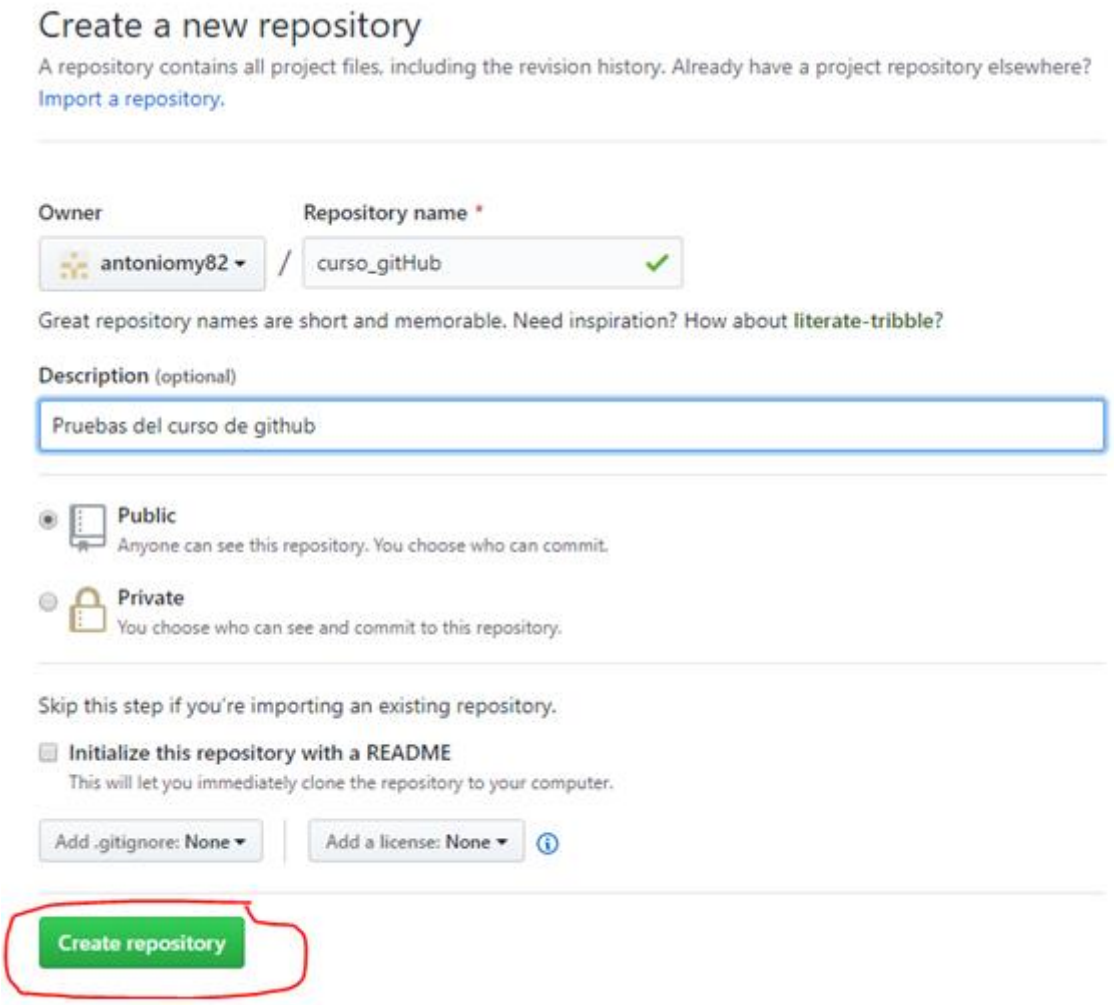
`Feature branch` es una rama de un proyecto principal, en el que cada desarrollador, está trabajando en nuevas características.

- Si queremos unir nuestra rama al proyecto principal:
  - `git checkout master` nos pasamos a la rama master.
  - `git merge miRama` hacemos un merge de miRama a master.
  - `git push` subimos los cambios.

- Otra forma, usando pull request, para que el resto de compañeros vean los cambios que he hecho.
  - `git push origin miRama` y hacemos `un pull request`

4. CREAR UN REPOSITORIO EN GITHUB

Nos vamos a nuestra cuenta de github, en la sección de “Your repositories” , “New”



En la consola:
Inicializamos el repositorio : `git init`
Comprobamos el estado de los archivos del proyecto (stage y branch) `git status -s -b`

Comprobamos que no hay commits `git log`
Agregamos todo `git add .`
Agregamos todo `git commit -m “Primer commit”`

Luego agregamos el origen

```
git remote add origin https://github.com/antoniomy82/curso_gitHub.git
git push -u origin master
```

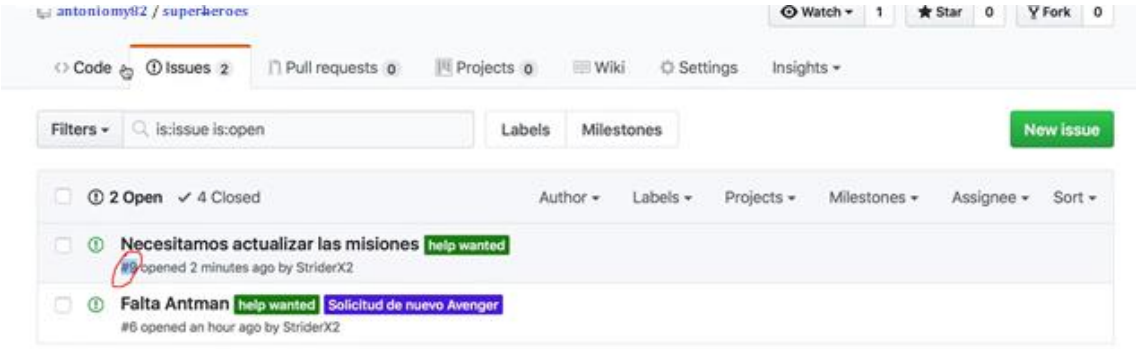
5. GITHUB – EXTRAS

Issues, se usan para resolver problemas, o para detallar tareas.

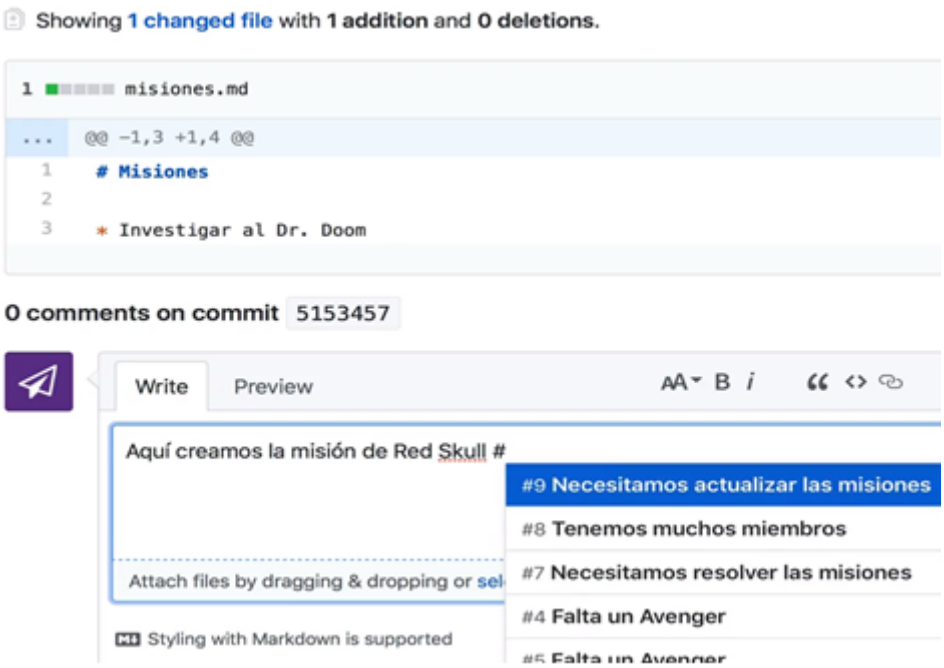
Milestones, son hitos que se ponen dentro del proyecto y están compuesto por varios issues, nos muestra en una barra de progreso el estado del milestone.

Asociar commits a issues:

Creamos un issue



Hacemos un commit referenciando al issue que hemos creado



Wikis Se pueden crear wikis, hay que crear todo en markdown.

Proyectos "projects", se pueden crear listas de notas o tareas al estilo trello. Podemos agarrar issues que tengamos como tareas.

Pages, se pueden hacer páginas web, html, ccs

Dentro de la opción Insight:

- **Pulse,** es una pestaña de github dónde se puede ver toda la información de commit, push, participantes y cómo va el proyecto.

- **Graph,** hace una gráfica de toda la actividad del proyecto, así como qué usuario ha hecho más commits y qué.

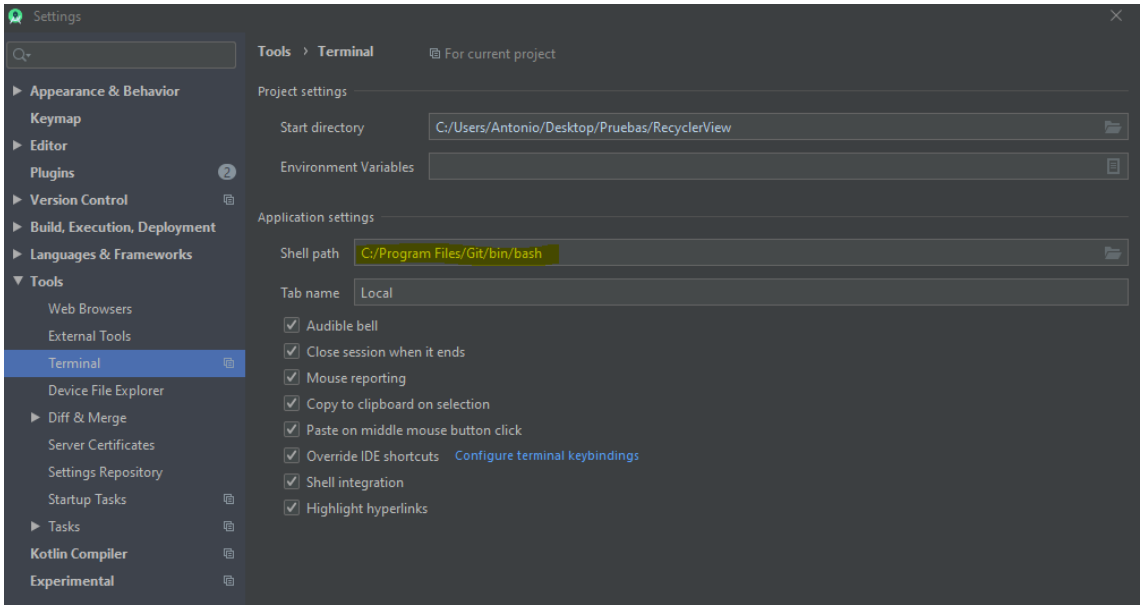
- **Gist**, es para compartir pequeños archivos o notas.

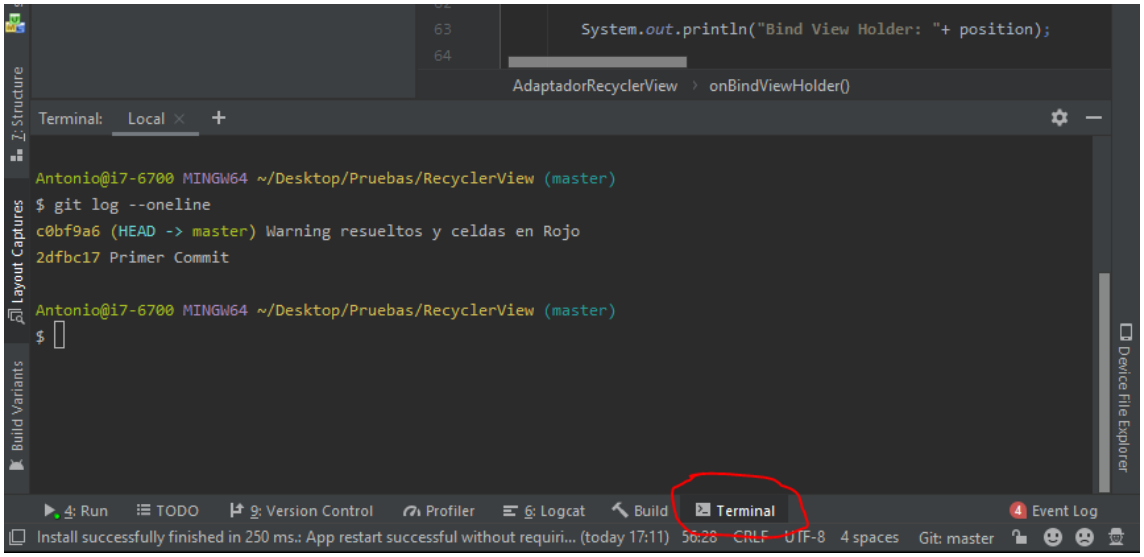
6. ANDROID STUDIO CONFIGURAR TERMINAL BASH

Instalar GIT, si no se ha instalado previamente: https://git.scm.com

En Android Studio ir a **File->Settings->Terminal**

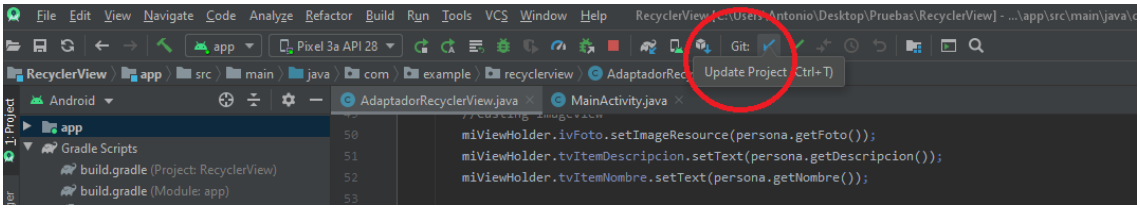
Y poner la ruta dónde se tenga instalado git bash en Shell path (Ojo con las contra barras)



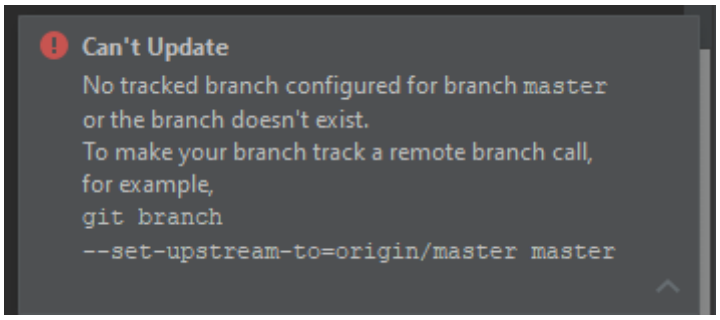


7. ANDROID STUDIO – ERRORES VARIOS

Después de trastear con ramas y merges, cuando actualizo el proyecto en : Git Update



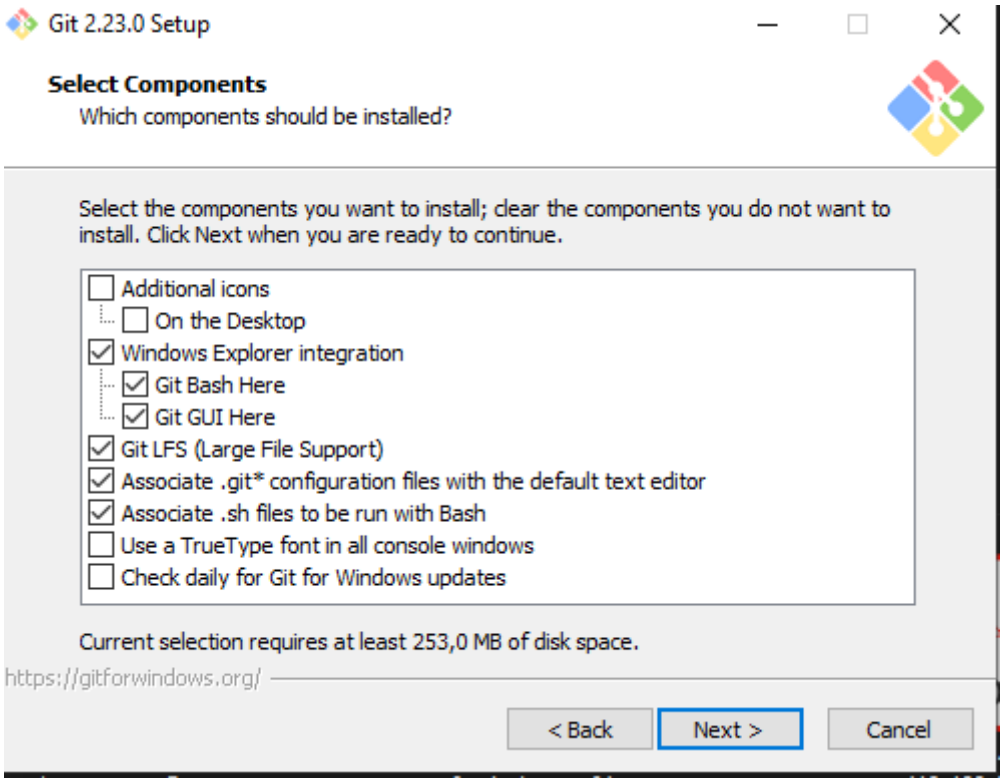
Me sale el siguiente error:

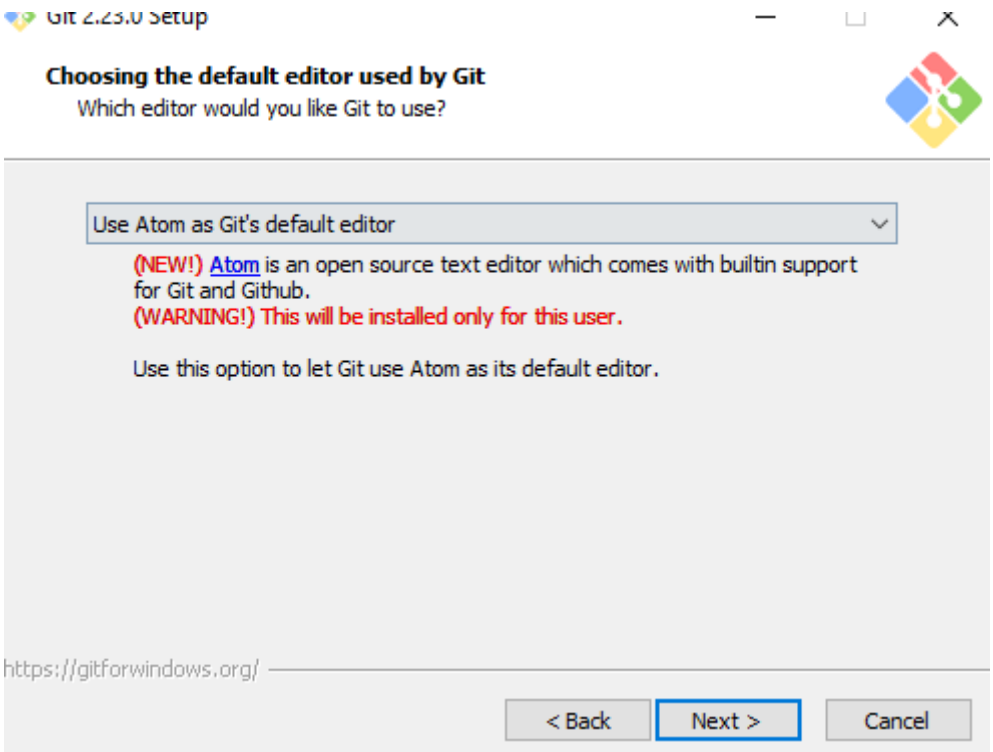


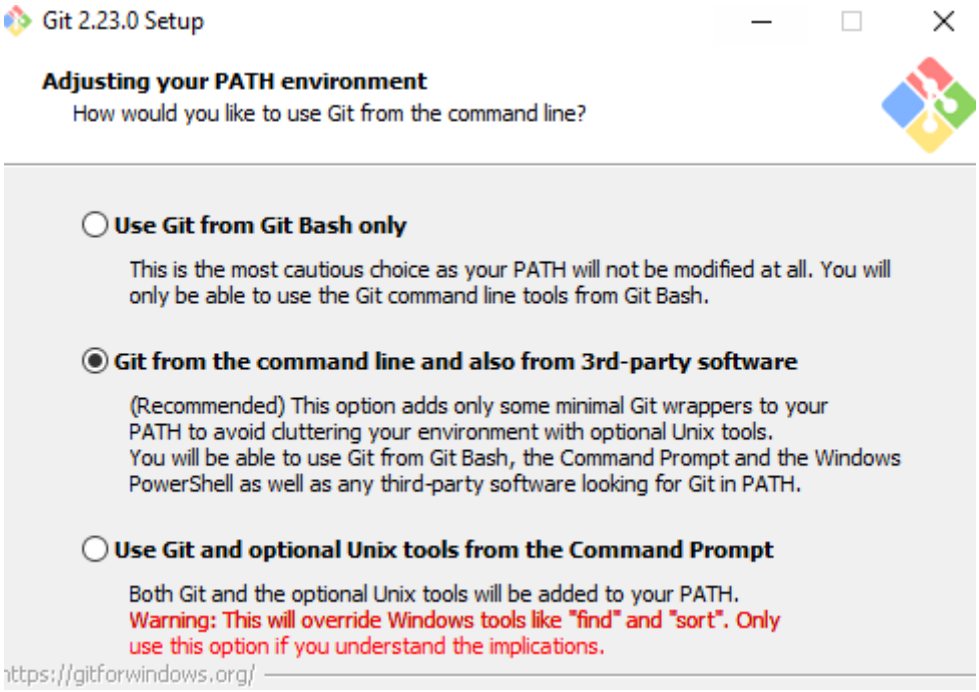
Como arreglar este tipo de bugs
https://stackoverflow.com/questions/10228760/fix-a-git-detached-head

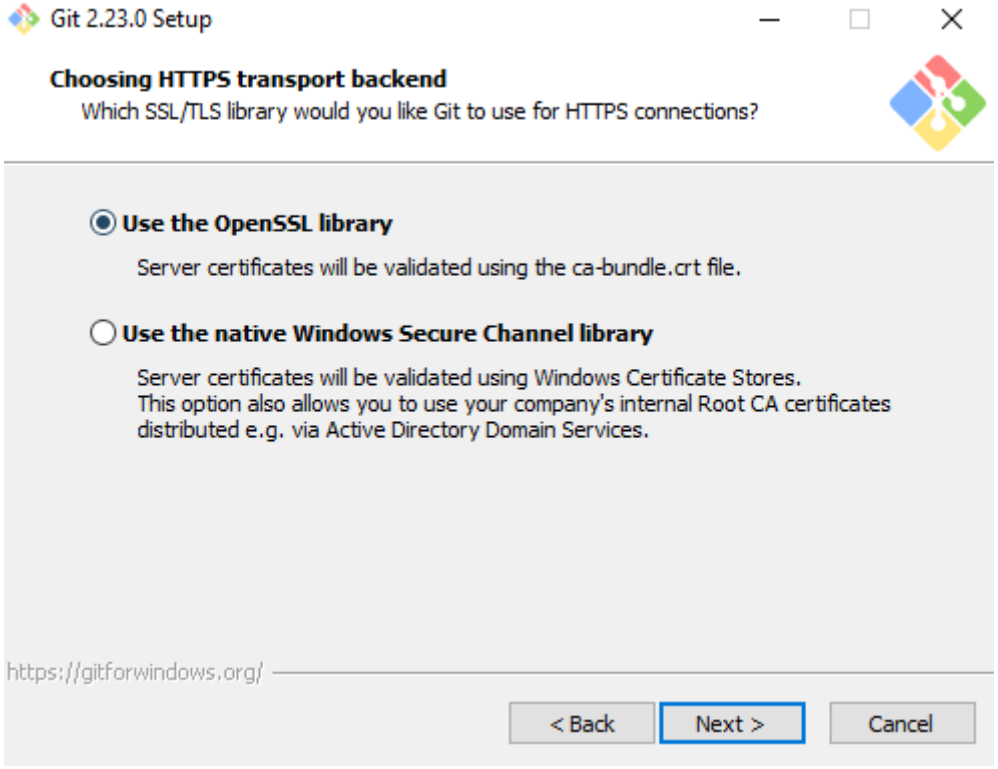
8. INSTALACIÓN DE GIT EN WINDOWS

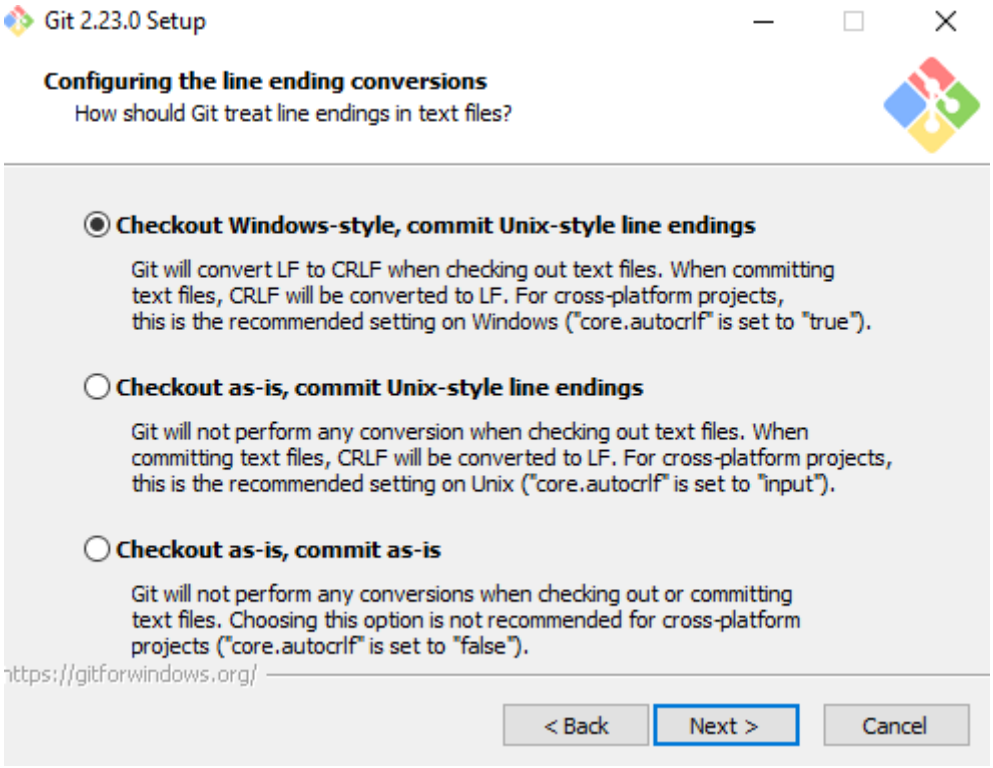
Enlace de descarga https://git-scm.com/

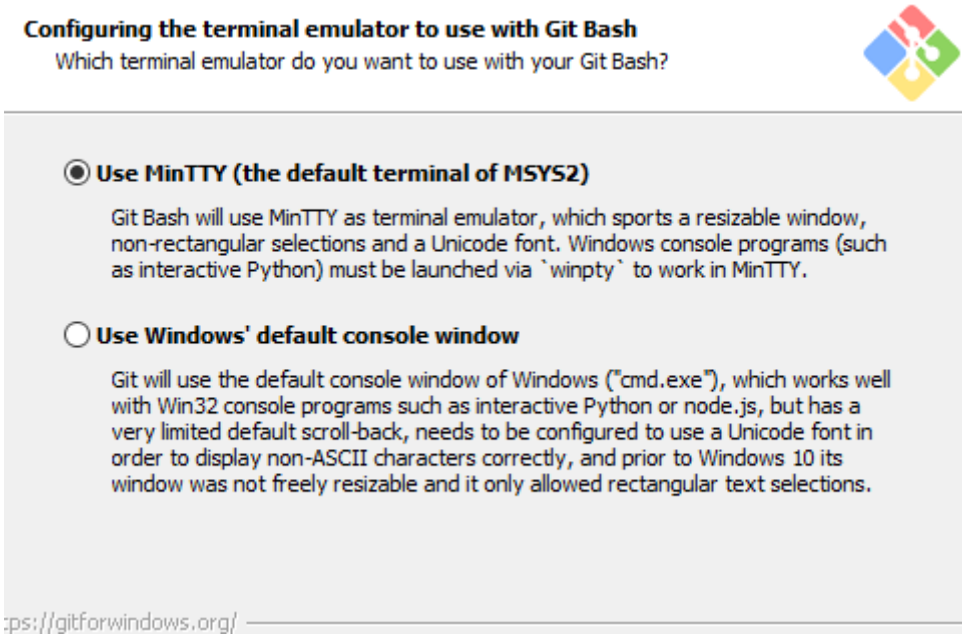


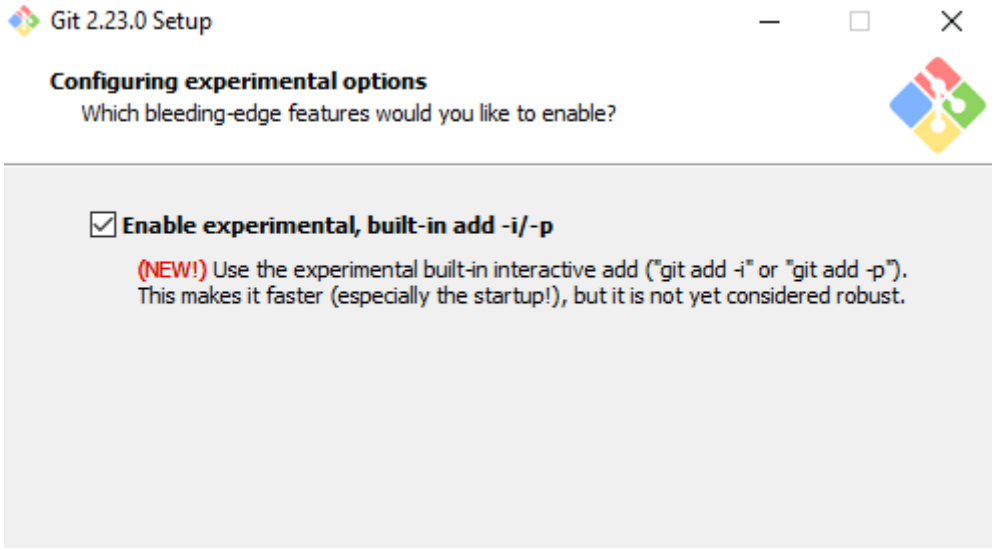



Git 2.23.0 Setup
Adjusting your PATH environment
How would you like to use Git from the command line?
Use Git from Git Bash only
This is the most cautious choice as your PATH will not be modified at all. You will only be able to use the Git command line tools from Git Bash.
Git from the command line and also from 3rd-party software
(Recommended) This option adds only some minimal Git wrappers to your PATH to avoid cluttering your environment with optional Unix tools.
You will be able to use Git from Git Bash, the Command Prompt and the Windows PowerShell as well as any third-party software looking for Git in PATH.
Use Git and optional Unix tools from the Command Prompt
Both Git and the optional Unix tools will be added to your PATH.
Warning: This will override Windows tools like "find" and "sort". Only use this option if you understand the implications.
https://gitforwindows.org/


Git 2.23.0 Setup
Choosing HTTPS transport backend
Which SSL/TLS library would you like Git to use for HTTPS connections?
Use the OpenSSL library
Server certificates will be validated using the ca-bundle.crt file.
Use the native Windows Secure Channel library
Server certificates will be validated using Windows Certificate Stores.
This option also allows you to use your company's internal Root CA certificates distributed e.g. via Active Directory Domain Services.
https://gitforwindows.org/
< Back
Next >
Cancel


Git 2.23.0 Setup
Configuring the line ending conversions
How should Git treat line endings in text files?
Checkout Windows-style, commit Unix-style line endings
Git will convert LF to CRLF when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Windows ("core.autocrlf" is set to "true").
Checkout as-is, commit Unix-style line endings
Git will not perform any conversion when checking out text files. When committing text files, CRLF will be converted to LF. For cross-platform projects, this is the recommended setting on Unix ("core.autocrlf" is set to "input").
Checkout as-is, commit as-is
Git will not perform any conversions when checking out or committing text files. Choosing this option is not recommended for cross-platform projects ("core.autocrlf" is set to "false").
https://gitforwindows.org/
< Back
Next >
Cancel


Configuring the terminal emulator to use with Git Bash
Which terminal emulator do you want to use with your Git Bash?
Use MinTTY (the default terminal of MSYS2)
Git Bash will use MinTTY as terminal emulator, which sports a resizable window, non-rectangular selections and a Unicode font. Windows console programs (such as interactive Python) must be launched via `winpty` to work in MinTTY.
Use Windows' default console window
Git will use the default console window of Windows ("cmd.exe"), which works well with Win32 console programs such as interactive Python or node.js, but has a very limited default scroll-back, needs to be configured to use a Unicode font in order to display non-ASCII characters correctly, and prior to Windows 10 its window was not freely resizable and it only allowed rectangular text selections.
https://gitforwindows.org/



ATOM

https://atom.io

Es un editor (como Notepad++) pero integrado con Git / Github

Saber la versión, abrimos cmd y `atom -- versión`