

Git - pequeña introducción práctica

Autor: Carlos Lombardi

Antes que nada, el tutorial que leí para preparar las clases y estas notas.

<https://www.atlassian.com/git/tutorials>

leí varios tutoriales y artículos, estos me parecieron los más claros.

Ahora sí, pasemos rápidamente por lo que hicimos (o deberíamos haber hecho y se me pasó).

Para arrancar

1. Se crea un proyecto en un servidor.
 - ◆ Elegir servidor, crearse una cuenta en el servidor elegido.
El más popular es GitHub, pero todos los repositorios que creas son públicos. Yo usé BitBucket, porque te permite repos privados. Pero es limitado (en la versión gratuita) para la cantidad de usuarios que puedo poner para que compartan mis repos. Otra opción es GitLab, creo que la cuenta gratuita es más potente.
 - ◆ Crear un nuevo repo. Poner algo en README.md, conviene que el repo no esté vacío para que sea más sencillo tener un repo local conectado¹.
 - ◆ Notar que al grabar el README.md, ya hace el primer commit, sobre el branch master. Podemos hacer un dibujo inicial de commits, como los que fuimos haciendo en el pizarrón.
2. Se hace un clon en nuestra máquina.
Recordar que lo que estamos haciendo es un repo local, que está vinculado con un repo remoto. Desde mi máquina, el repo remoto se llama origin.
 - ◆ Claro, antes hay que instalarse un cliente git en la máquina local. Hay varios, de línea de comando y también gráficos. Las capturas de pantalla de este documento son de **SourceTree**. OJO a veces hay que darle F5 al SourceTree para que actualice el gráfico a partir de los cambios que pudo haber en el disco.
 - ◆ Hay dos métodos para especificar el repo remoto: SSH o HTTPS. Usar HTTPS es más fácil, evita una configuración.
 - ◆ La referencia al repo remoto es una URL, ni más ni menos.
 - ◆ Si uno pone `git clone https://miSuperURL`, entonces hace el clon en una subcarpeta repo (o algo así). Para evitar la subcarpeta, ponerle un punto al final, que quiere decir "haceme el clon exactamente donde estoy parado". O sea:
`git clone https://miSuperURL .`
 - ◆ Ahora tenemos master y origin/master, los dos apuntando al único commit.

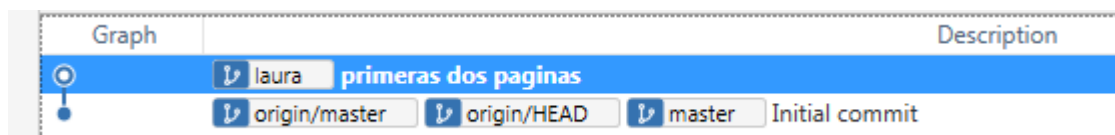
Una persona trabajando

3. Empieza a trabajar una persona (a quien llamaremos Laura), para lo cual lo primero que hace es crear su branch.
Para crear el branch: `git branch laura`.
Para "moverse" al nuevo branch (o sea, que lo que se haga a continuación afecte al branch laura y no al branch master): `git checkout laura`².
Para ver qué branches hay: `git branch`.
 - ◆ Notar que hay dos branches locales: master y laura, más uno remoto: origin/master. Pero commit hay uno solo. Los tres branches apuntan al único commit.
4. Laura agrega dos páginas: index y bibliotecas/centro
 - ◆ En este momento los archivos están "unstaged".
Es decir, a git no le interesan los cambios que se puedan hacer sobre estos archivos.

¹ Hay una razón técnica: git no maneja carpetas, solamente archivos, entonces se lleva mal con las carpetas vacías (las ignora), y por lo tanto no sabe bien cómo manejar un repo sin ningún archivo.

² Como crear un branch y luego moverse es de lo más común, el comando abreviado `git checkout -b` hace ambas cosas. En este ejemplo, `git checkout -b laura` equivale a hacer `git branch laura` e inmediatamente después `git checkout laura`. ¡Gracias Federico Aloí!

- ◆ Esto se ve en el sourcetree, ver abajo a la izquierda que tiene secciones de “unstaged” y “staged”.
 - ◆ Desde línea de comandos se puede hacer git status. Acá las cosas unstaged se muestran como “Untracked changes”. De las carpetas que todavía no maneja, el git status no muestra el contenido. Es así, un poco tontito.
5. Laura le comunica a git la existencia de estos archivos. El comando para esto es git add. Vamos de a uno.
- ◆ git add bibliotecas/centro.html .
 - ◆ Ver ahora, tanto en consola (git status) como en sourcetree. Tenemos un archivo staged y uno unstaged.
 - ◆ ¿Qué quiere decir esto? Que si hiciéramos un nuevo commit en este momento, se incluiría bibliotecas/centro.html, pero no index.html.
Moraleja: por lo general, antes de hacer un commit conviene verificar bien que se está incluyendo todo lo que se quiere commitear.
 - ◆ Vamos con el otro: git add index.html .Ahora no hay nada “unstaged”.
Pero ¡OJO! sigue habiendo un solo commit.
6. Laura hace un commit: git commit -m “primeras dos paginas” . Ahora sí se agrega un nuevo commit, que incluye los dos archivos que subimos a “staged” en el punto anterior. El branch laura está parado en el nuevo commit. El branch master sigue estando en el commit anterior, o sea, uno antes que laura.



Por otro lado, todos los archivos están commiteados en su versión actual. Dicho de otro modo, no hay nada ni “staged” ni “unstaged”.

```
c:\data\carlos\obj2\git-test-repeticion\laura>git status
# On branch laura
nothing to commit, working directory clean
```

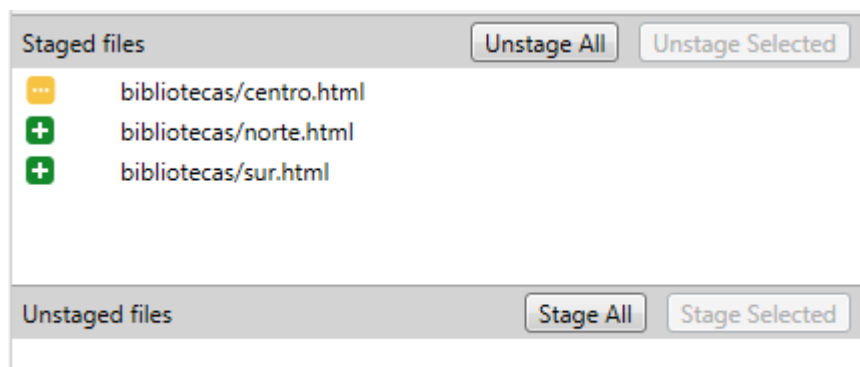
7. Laura modifica bibliotecas/centro.html (para agregar un libro), y las páginas de dos bibliotecas más.
- ◆ Después de tocar los archivos, está todo unstaged.

```
c:\data\carlos\obj2\git-test-repeticion\laura>git status
# On branch laura
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   bibliotecas/centro.html
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
#       bibliotecas/norte.html
#       bibliotecas/sur.html
no changes added to commit (use "git add" and/or "git commit -a")
```

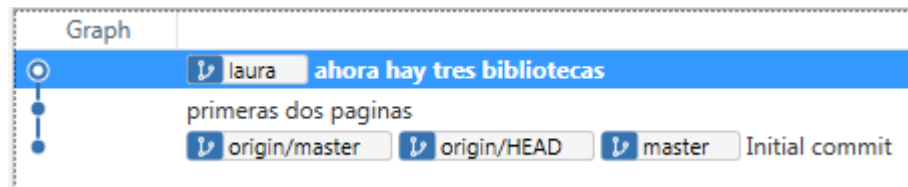


- ◆ Mediante git add . (desde la carpeta raíz de Laura) pasamos todo a staged.

```
c:\data\carlos\obj2\git-test-repeticion\laura>git add .
c:\data\carlos\obj2\git-test-repeticion\laura>git status
# On branch laura
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   bibliotecas/centro.html
#       new file:   bibliotecas/norte.html
#       new file:   bibliotecas/sur.html
#
```



- ◆ Creamos un nuevo commit: git commit -m "ahora hay tres bibliotecas". Ahora el branch laura está dos commits más adelante que master.

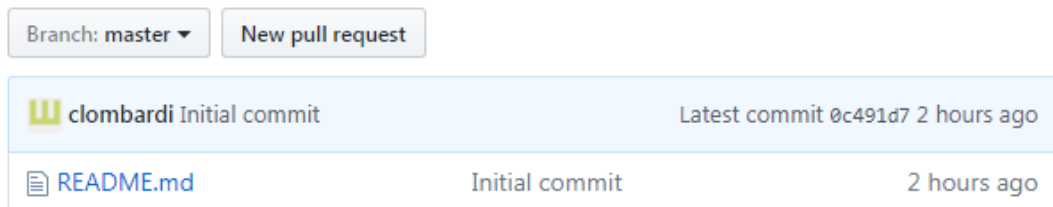


Se suma una segunda persona

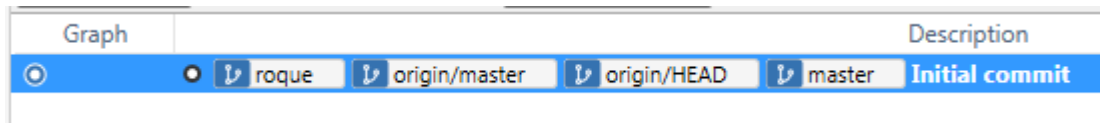
8. Se suma Roque a trabajar, se va a encargar de la biblioteca del oeste.
 Laura y Roque van a trabajar cada uno en su branch.
 Para arrancar, Roque hace un clone en su máquina, y se crea su branch. Repasemos:
`git branch roque`
`git checkout roque`

Pero ... no le aparece ningún contenido. Él en sourcetree ve esto

¿Qué pasó? Que el clone se trae el contenido de origin/master. Los cambios están **solamente** en el *branch local* laura. Esto quiere decir que **nunca se subieron al repo remoto**. Si consultamos github.com, aparece solamente esto



Coherentemente, Roque en sourcetree ve esto



Moraleja: git commit maneja solamente el repo local. Para subir cosas al repo remoto hay que usar git push.

9. Para que Roque pueda trabajar a partir de lo que hizo Laura, hay que hacer varias cosas:
 - a. Laura va a actualizar su copia del branch master para que esté en el mismo lugar que laura.
 - b. Laura va a sincronizar origin/master para que esté en el mismo lugar que (su copia de) master.
 - c. Después de esto, Roque puede traerse los cambios que (ahora sí) están en origin/master. Como ya creó su branch roque, tiene que actualizarlo.

Veamos cómo se hace cada una de estas cosas.

10. Paso 1:

actualización de master en el repo local de Laura. Para hacer esto Laura

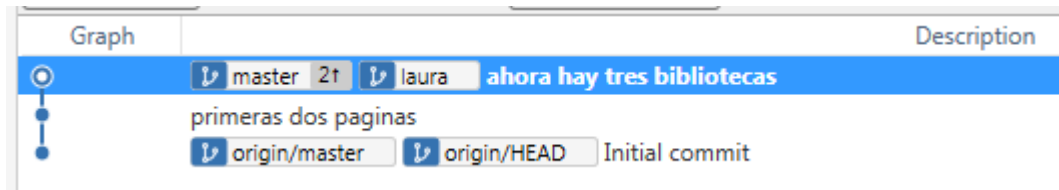
- a. "Salta" de branch: `git checkout master` .

```
c:\data\carlos\obj2\git-test-repeticion\laura>git checkout master
Switched to branch 'master'

c:\data\carlos\obj2\git-test-repeticion\laura>git branch
  laura
* master
```

- b. Actualiza master (donde está parada) con el estado de laura.
 Para eso se usa el comando `git merge` : actualiza el branch **donde estoy parado** con lo que esté en otro branch. En este caso: `git merge laura` . Recordar que Laura está parada en master.

Ahora en sourcetree, Laura ve esto.



Un detalle: el “2↑” al lado de master. Eso quiere decir que el master local (de Laura) está dos commits por encima del branch remoto correspondiente, que es origin/master.

- c. Obviamente, Roque ni se entera. Notar que Laura tampoco ve el branch roque, ese está solamente en el repo local de Roque.

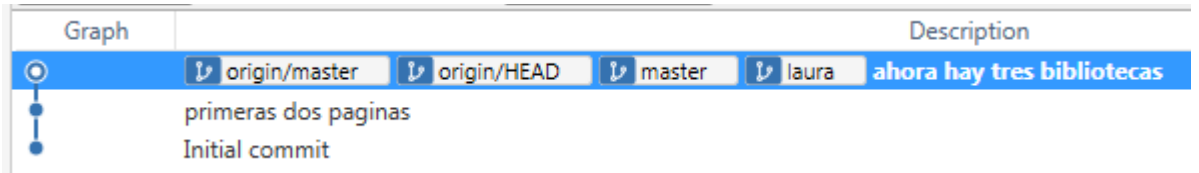
11. Paso 2:

Laura lleva origin/master donde está master.





Esto es exactamente lo que hace git push: actualiza un branch remoto poniéndolo en el mismo lugar en donde está el local. O sea:

`git push origin master3`

A ver en el sourcetree de Laura:



Ahora sí vemos agregados en github.com.

 laura-obj2 ahora hay tres bibliotecas	Latest commit b75d1ca 2 hours ago
 bibliotecas	ahora hay tres bibliotecas 2 hours ago
 README.md	Initial commit 4 hours ago
 index.html	primeras dos paginas 2 hours ago

Super importante

Inmediatamente después de esto, Laura tiene que volver a su branch

`git checkout laura`

para no hacer commits en master.

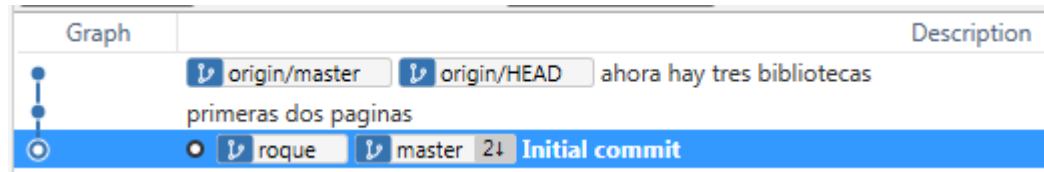
³ ¿Por qué hay que poner “origin”? Porque git permite tener varios repos remotos para el mismo proyecto, entonces hay decir cuál se quiere usar. Para nosotros va a ser siempre “origin”.

12. Paso 3:

Roque sincroniza su branch x con origin/master. Esto requiere de dos pasos obligatorios, más uno opcional pero que yo haría

- Traerse las novedades del repo remoto. Para esto se usa git fetch. Así de fácil:
`git fetch origin`

Ahora Roque ve esto en su sourcetree



O sea, el fetch trae la info de los branches remotos, pero no actualiza los branches locales. Para actualizar un branch local, usamos git merge. (el mismo comando que usó Laura antes⁴).

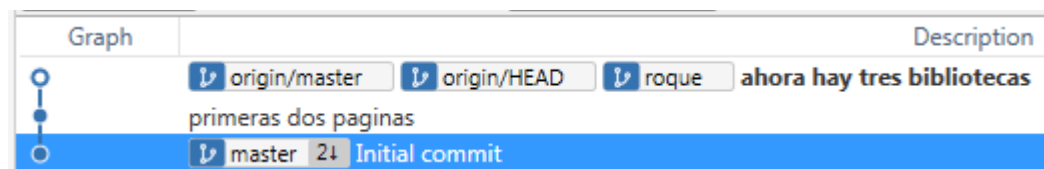
Ahora el master de Roque está dos commits “abajo” que el correspondiente remoto, por eso el “2↓”.

- Vamos con el merge. Vale hacer merge de branches remotos ... si antes me los traje con fetch, OJO ahí. En este caso:

`git merge origin/master`

(recordar que Roque está parado en el branch roque, ese es el que se actualiza).

Ahora tenemos



Roque ya puede trabajar sobre lo que hizo Laura. Por prolijidad, hagamos un paso más ...

- ... que es actualizar el branch master en el repo local de Roque. Para eso:

`git checkout master`

“salto” a master

`git merge origin/master`

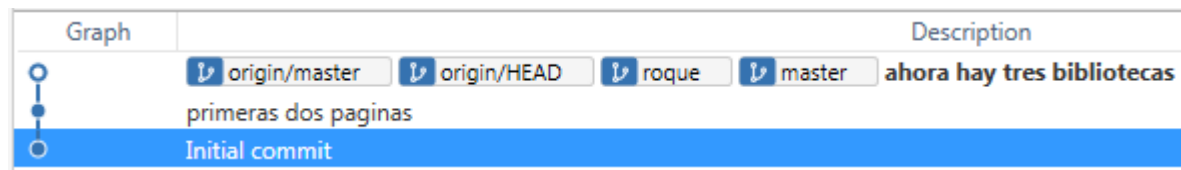
actualiza master local

`git checkout roque`

vuelvo a roque

OJO no olvidar este paso

Ahora el local master de Roque ya no está dos commits abajo:



⁴ Como pasa casi siempre, en Git hay distintas formas de actualizar un repo local. En este caso se puede usar git pull. Pero en la explicación (y también en mi trabajo diario) prefiero usar siempre git merge para actualizar un branch local, me resulta más claro. Por eso lo explico así.

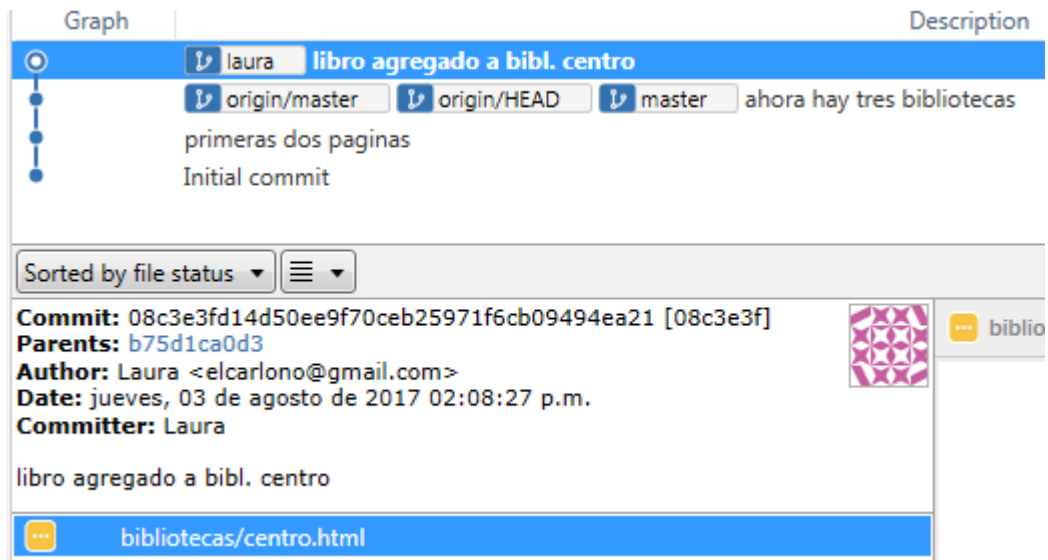
Las dos personas hacen cambios

13. Tanto Roque como Laura hacen cambios. Supongamos que

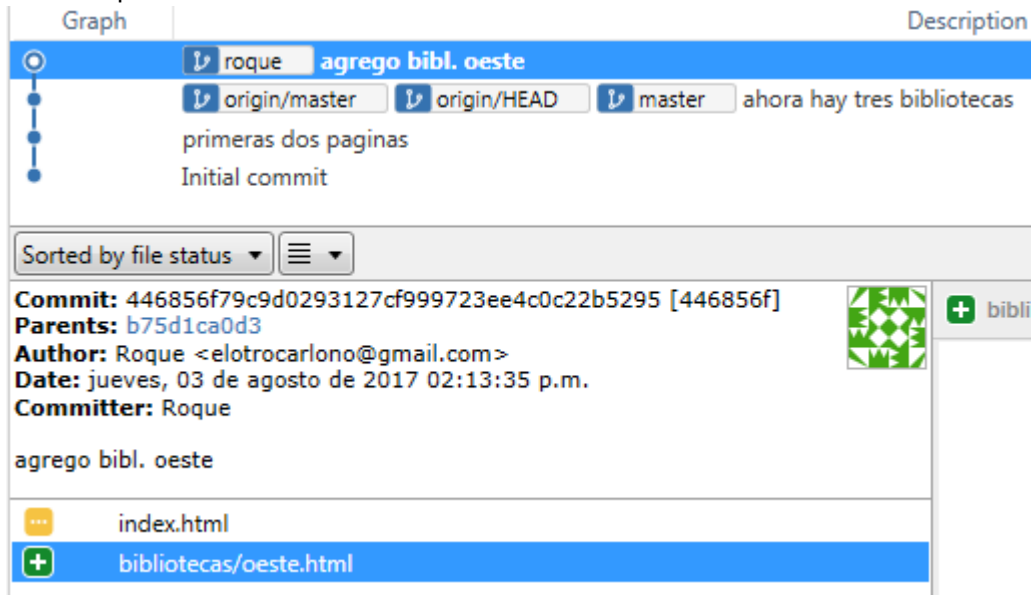
- ◆ Roque toca index.html para agregar la biblioteca del oeste, y algo de contenido para esa biblioteca.
- ◆ Laura agrega contenido a la página de la biblioteca del centro.

Cada uno genera un nuevo commit en su branch, en su repo local.










El sourcetree de Laura queda así:



El de Roque muestra esto



El repo remoto no se entera de nada.

	ahora hay tres bibliotecas laura-obj2 committed 2 hours ago	 b75d1ca	
	primeras dos paginas laura-obj2 committed 2 hours ago	 c708980	
	Initial commit clombardi committed 4 hours ago	 0c491d7	

14. Supongamos que Roque es el primero que sube sus cambios. Son necesarios varios pasos

a. Por las dudas, se trae las novedades del repo remoto: `git fetch origin` .
No muestra nada, el sourcetree no cambió, Roque está en su día de suerte, puede meter sus cambios derecho viejo ...

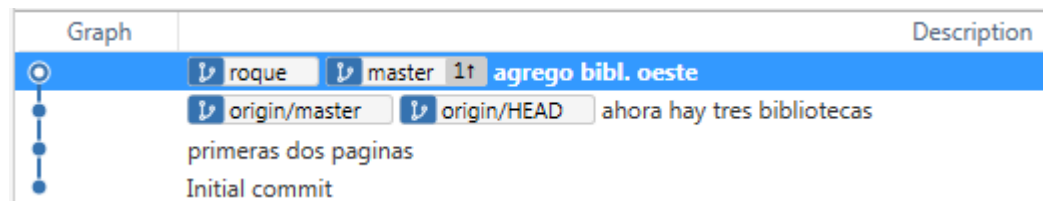
b. ... siguiendo los mismos pasos que hizo Laura antes (ver sección “**Se suma una segunda persona**”).

Primero lleva su master local a donde está el branch roque:

`git checkout master`

`git merge roque`

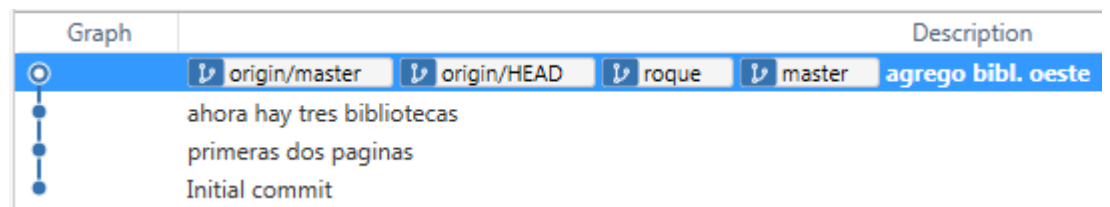
El sourcetree muestra que vamos bien:



Después, se sube origin/master donde está ahora el master local:

`git push origin master`.

Queda así:



c. **OJO no olvidar**: `git checkout roque`.

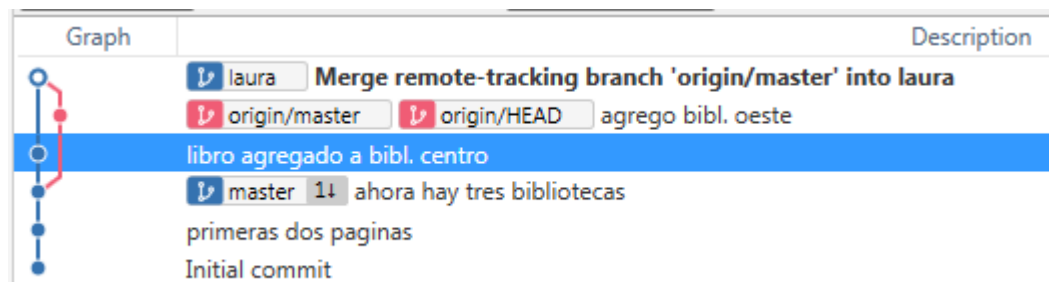
15. El turno de Laura

- Al igual que Roque, arranca con un git fetch origin . Pero ... auch, ella sí se trae novedades, lo que hizo Roque:



El árbol de branches se abrió, esto refleja que Laura y Roque hicieron modificaciones por separado. Ahora Laura va a juntar estas ramas.

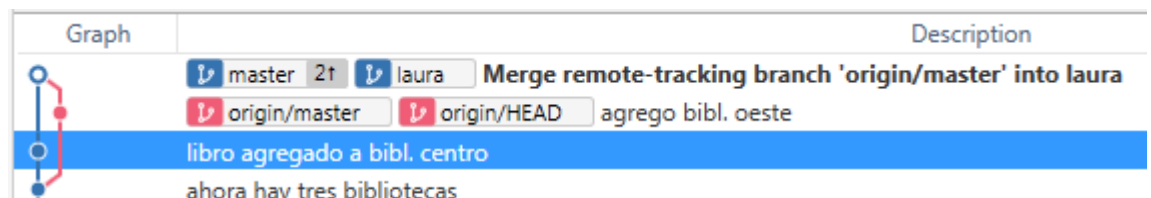
- Laura se trae a su branch las novedades. Notar que todavía está parada en el branch laura, entonces alcanza con: `git merge origin/master` .
ATENCIÓN: en este caso el branch laura no se puede mover a origin/master, porque se perderían los cambios que hizo Laura. Lo que hace git en este caso es *crear un nuevo commit de prepo*, en el que mezcla los cambios que se hicieron en origin/master y en laura, a partir de donde se separaron las ramas (en este caso, el commit “ahora hay tres bibilotecas”). En sourcetree se ve así



IMPORTANTE:

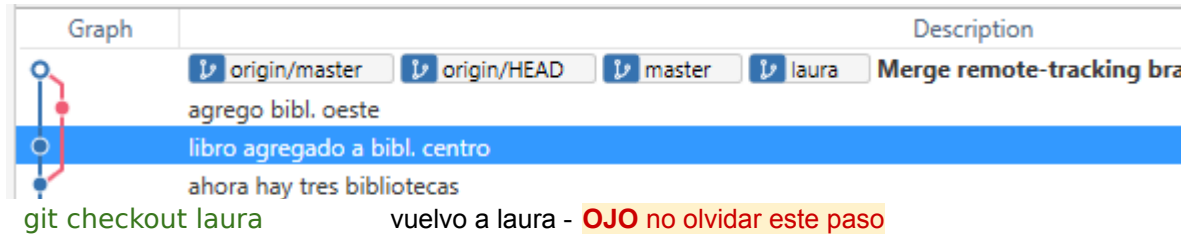
En este punto, Laura tiene el resultado de integrar el código que escribió Roque junto con el de ella. Antes de seguir, **tiene que probar que la integración no haya roto nada**. O sea, que siga andando todo, o al menos lo que ella hizo.

- Después de haber comprobado que la integración no rompió nada, lo que queda es actualizar, primero master, después origin/master, como hizo antes. Repasemos:
`git checkout master` “salto” a master
`git merge laura`
 actualiza master local con el resultado de la integración, que está en laura



`git push origin/master`

actualiza master remoto con lo que (ahora) tiene el local



d. Comentario 1:

Laura tiene todo integrado en su repo local, Roque no, no se cargó las modificaciones de Laura.

Mientras Roque siga trabajando sobre la biblioteca del oeste y Laura sobre las otras, no es imprescindible que ambos se actualicen todo el tiempo las novedades del server. Sí lo pueden hacer, para eso:

- Primero git fetch origin
- Si hay cambios, git merge origin/master
- Antes de seguir, probar que todo siga andando. Si se rompió algo, avisar.

e. Comentario 2:

¿Qué pasa si justo Roque y Laura tocaron **el mismo** archivo? Depende.

Si tocaron secciones distintas, git se da cuenta, y en el merge integra los cambios locales con lo que viene del repo remoto.

Si se tocó p.ej. la misma línea, ahí hay un conflicto, y hay que resolverlo. Esto por favor googléenlo, git merge conflict. Vale preguntar. Si alguien (mejor si no Román) tiene ganas de chusmearlo y contarlo, excelente.

f. Comentario 3:

Hay una alternativa al git merge que se llama git rebase. Si entendí bien, te deja más prolijo el árbol de branches, creo que evita el commit que hace git de prepo en el merge. Esto es un chiche. Otra vez, si alguien (mejor si no Román) tiene ganas de chusmearlo y contarlo, excelente.

Backup personal en el repo remoto

En lo que hicimos recién, ni Laura ni Roque suben nada al servidor hasta que consideran que terminaron algo que vale la pena consolidar en origen/master.

Si p.ej. Laura está trabajando durante varios días en algo que quiere subir al final, puede tener ganas de ir haciendo backups de su trabajo en el repo remoto.

Una forma de hacer esto es replicar el branch de trabajo (en este caso laura) en el repo remoto.

Para esto, parada en su branch laura, Laura puede hacer sencillamente

git push origin laura

La primera vez que Laura hace esto, se crea un nuevo branch en el repo remoto, origin/laura. Los pushes subsiguientes llevan origin/laura a donde esté laura.

En principio, mientras Laura siga trabajando en la misma máquina, no necesita incorporar cambios de este branch. Ahora, si Laura va a otra máquina, y había hecho push con lo último, alcanza con

git clone ...repo...

git branch laura

git fetch origin

git merge origin/laura

y listo, sigue trabajando desde el punto en que había dejado en la otra máquina. Esto puede servir para los que trabajan en una compu en clase y en otra distinta en casa.

Para chusmear cada uno

- Archivo .gitignore, sirve para p.ej. no incluir los .class en el repo remoto.
- git stash ... mírenlo.
- GitHub Flow: es la forma en que GitHub recomienda trabajar en equipo. Por lo que veo, es bastante parecido a lo que propongo acá. La diferencia son los pull requests, que no sé si

tienen sentido en un equipo chico que se puede juntar a discutir.







Ver <https://guides.github.com/introduction/flow/> .

Resumen de comandos

git clone <repo_remoto> .	Crea un repo local conectado con repo_remoto
git branch laura	Crea un branch local laura
git branch	Muestra los branches locales
git branch -r	Muestra los branches remotos
git checkout laura	Me paro en el branch local laura
git status	Muestra qué hay unstaged y staged
git add .	Pasa a staged todo lo que esté unstaged, de la carpeta donde estoy parado “para abajo”
git add <file_1> ... <file_n>	Pasa a staged los archivos que se indican
git commit -m “msg-de-commit”	Crea un nuevo commit, y mueve el branch donde estoy parado al nuevo commit.
git fetch origin	Trae al repo local las actualizaciones que se hayan registrado en el repo remoto. No modifica ningún branch local.
git merge <branch>	<p>Me traigo al branch local donde estoy parado, las actualizaciones que estén registradas en <branch> .</p> <p>Se puede hacer merge con branch locales o remotos, p.ej. si estoy en master, entonces</p> <ul style="list-style-type: none">• git merge roque Trae al master local los cambios que estén en el branch local roque.• Git merge origin/master Trae al master local los cambios que estén registrados en el master del repo remoto. OJO para que ande esto hay que hacer fetch antes. <p>Si hubo cambios simultáneos, el merge genera un commit de prepo, que junta las ramas.</p> <p>Si hubo cambios simultáneos incompatibles, se generan conflictos que hay que resolver.</p>
git push origin <branch>	<p>Lleva a origin/<branch> a donde esté el <branch> local. Lo usamos para dos cosas:</p> <ol style="list-style-type: none">1. Mediante git push origin master Subimos los cambios después de integrarlos y llevarlos al master local.2. Mediante git push origin <mi_branch> hago un backup en el repo remoto de lo que tenga en mi branch local.

Cómo queda el repo al final de todo ...

... en la vista de Laura que lo tiene consolidado

Graph	Description	Date	Author
	  llego otro libro a la bibl. central	3 ago 2017 15:57	Laura <e af54408
	   Merge remote-tracking branch 'orig	3 ago 2017 15:01	Laura <e c758216
	agrego bibl. oeste	3 ago 2017 14:13	Roque < 446856f
	libro agregado a bibl. centro	3 ago 2017 14:08	Laura <e 08c3e3f
	ahora hay tres bibliotecas	3 ago 2017 12:32	Laura <e b75d1ca
	primeras dos paginas	3 ago 2017 12:11	Laura <e c708980
	Initial commit	3 ago 2017 10:29	Carlos Lc 0c491d7

Después de la consolidación, Laura hizo un commit adicional, y después hizo backup en el repo remoto.

Notar que

- Se ve quién hizo cada commit.
- Cada commit tiene un identificador que es un número en hexa, p.ej. el del commit inicial es 0c491d7.. Este identificador sirve para algunas tareas donde uno quiera trabajar con un commit particular.
- Si ahora Roque se hace fetch y después merge desde origin/master, no se trae lo último que hizo Laura. Esto está bien, se tiene que traer solamente hasta donde Laura haya decidido consolidar en master.

¡Ah! el repo lo pueden chusmear, y también hacerle clone. Está publicado como

<https://github.com/clombardi/obj2-pruebasGit>