Introducción a git

Control de versiones con git
Repositorios remotos - gitHub

21/08/2020

¿Qué es un CV?

Llamamos **control de versiones** a una forma de gestionar los cambios que se realizan sobre documentos, programas o cualquier tipo de archivo. La idea de un **sistema de control de versiones** (SCV) es poder recuperar la versión de cualquier archivo o documento en un estado anterior en el tiempo.

Algunos ejemplos de la vida real pueden ser:

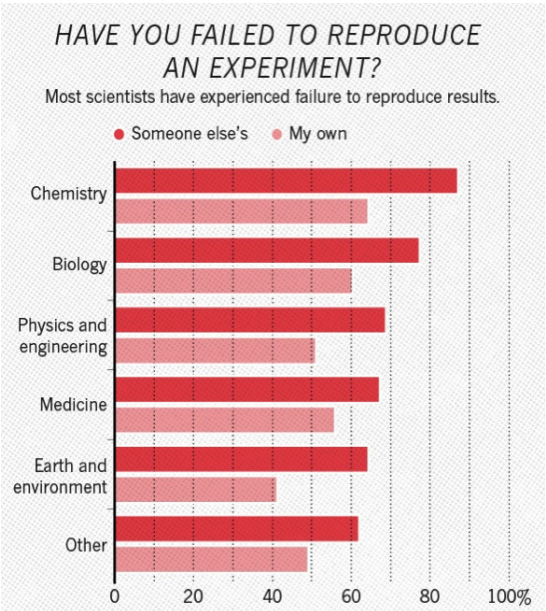
- Cuaderno de laboratorio
- Revisiones de un artículo o tesis
- Desarrollo de un código (*scripting*)

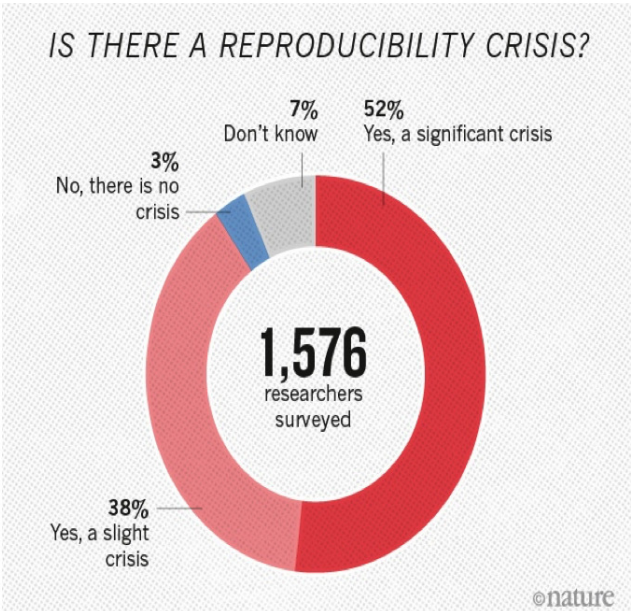
SCV

OK, y ¿para qué?

SCV

¿Para qué queremos realizar un CV?

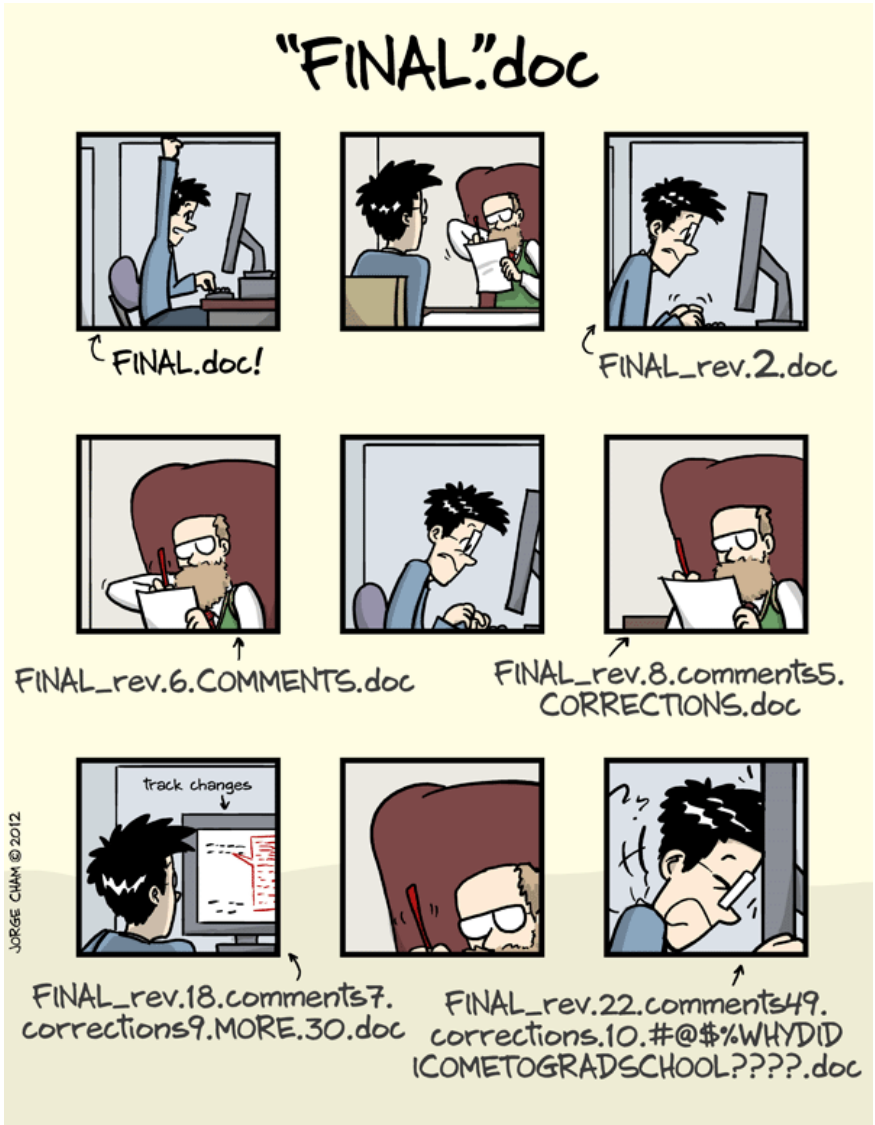




Number of respondents from each discipline:
Biology **703**, Chemistry **106**, Earth and environmental **95**, Medicine **203**, Physics and engineering **236**, Other **233** ©nature

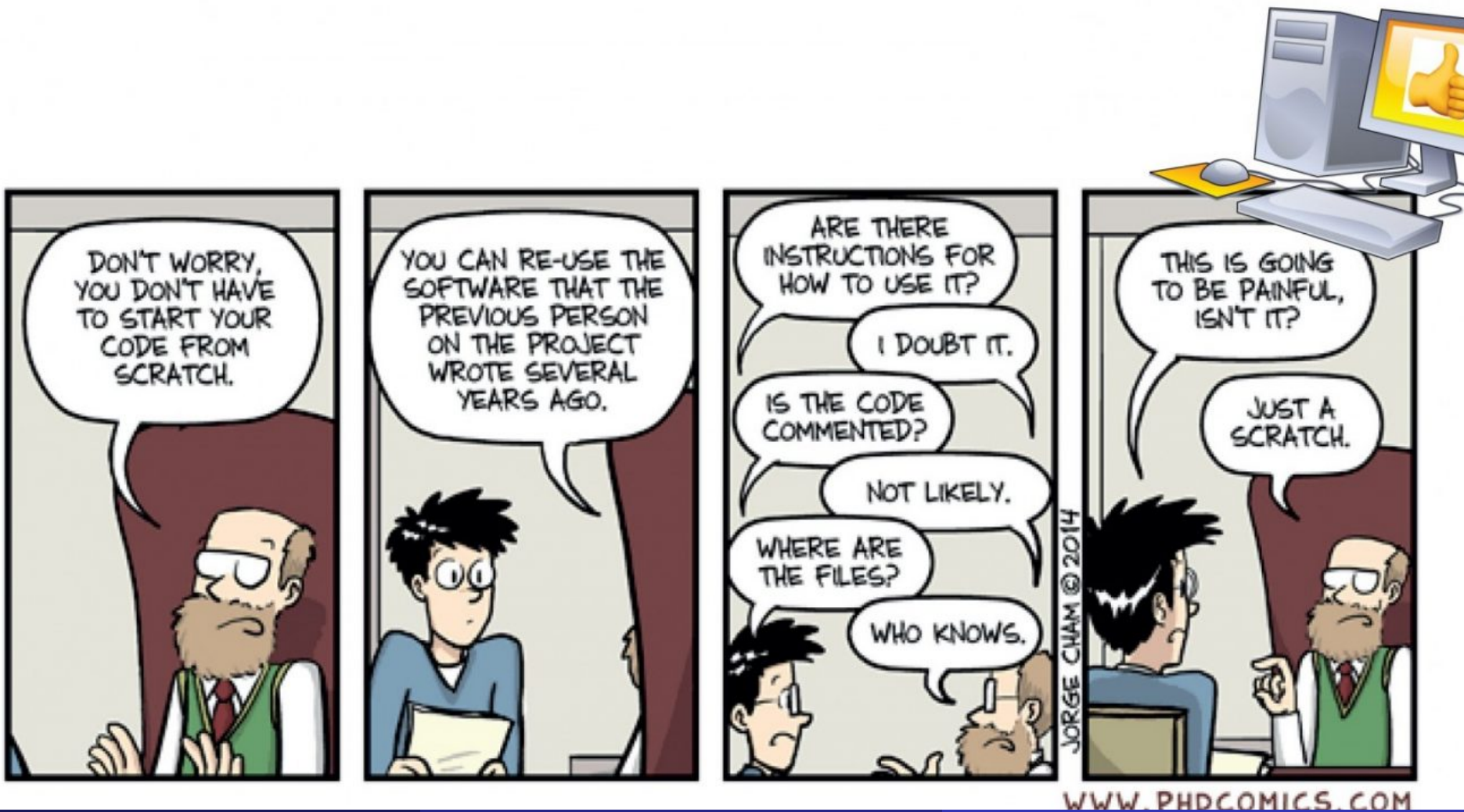
SCV

¿Para qué queremos realizar un CV?



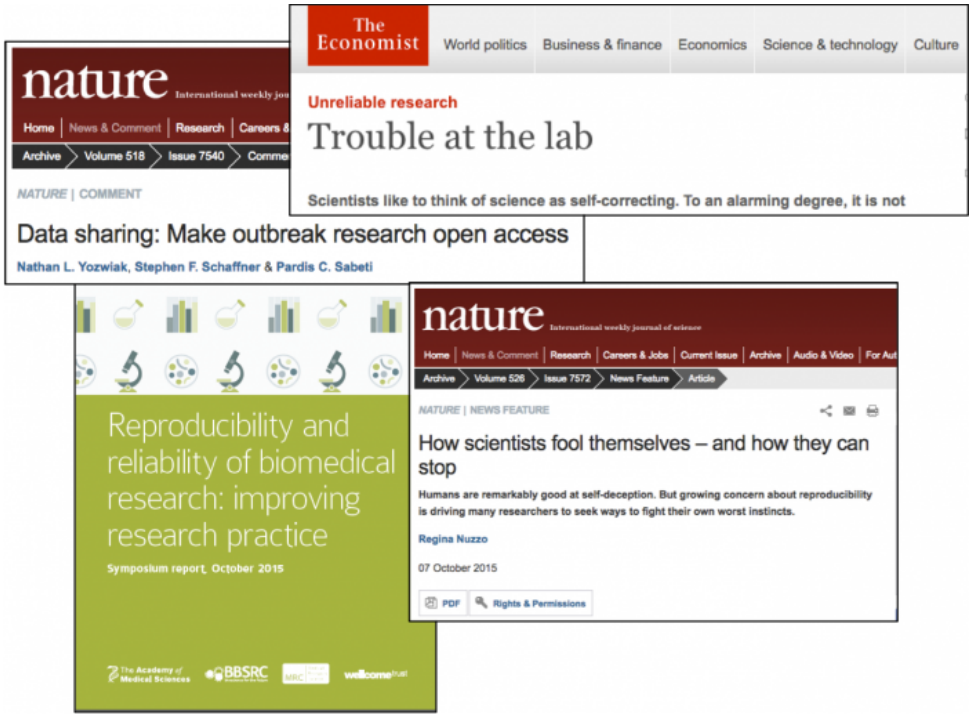
SCV

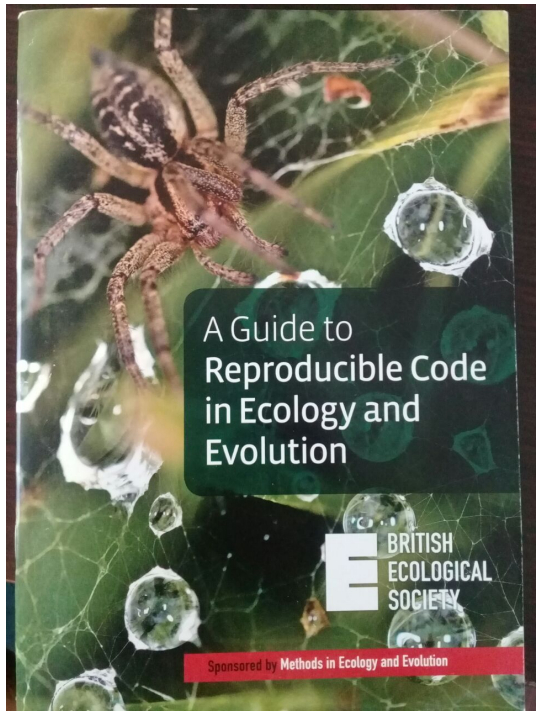
¿Para qué queremos realizar un CV?



SCV

¿Para qué queremos realizar un CV?





git

¿Qué nos permite hacer `git`?

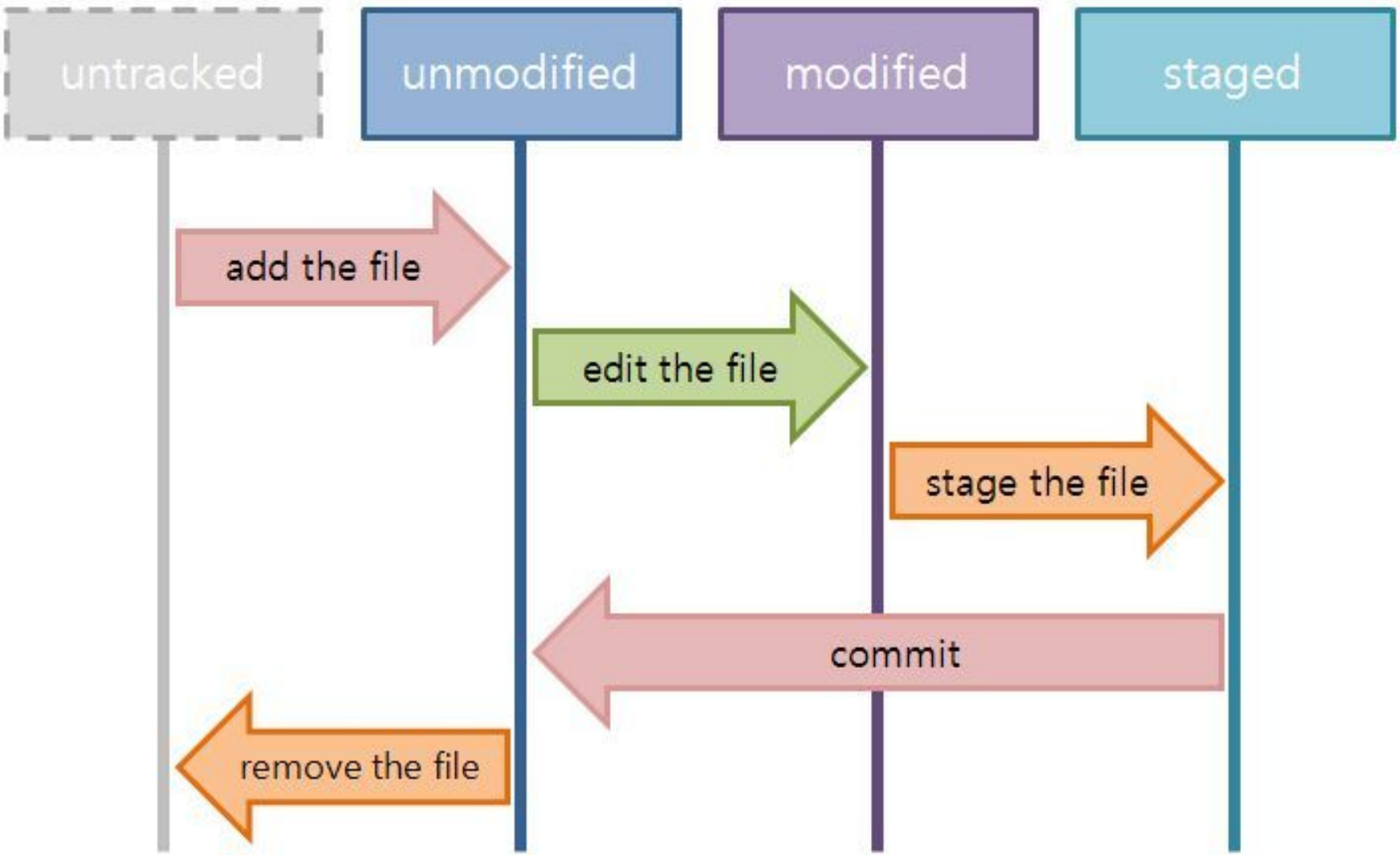
- Tener backups en todos los estadíos de nuestro proyecto
- Documentar cambios
- Compartir cambios
- Distribuir el producto y el desarrollo a la vez a muchas personas

git

Algunos conceptos importantes

- **Repositorios:** análogos a carpetas o conjunto de carpetas que conforman nuestro proyecto.
- **Estados de un repositorio:** Conforma una imagen instantánea del proyecto. Cuando terminamos de trabajar en nuestro repositorio (carpeta), podemos consolidar los cambios realizados para no modificarlo más. Esto es como tomar un *snapshot* del repo. El *snapshot* actual se llama `HEAD`
- **Ciclo de vida de los archivos:** en un repo de `git` cada archivo puede tener uno de estos tres estados:
  - no-modificado
  - modificado
  - actualizado

git


untracked
unmodified
modified
staged
add the file
edit the file
stage the file
commit
remove the file

git

- Un archivo está en estado **no-modificado** cuando es igual al archivo guardado en el último *snapshot*.
- Al modificar cualquier linea o caracter de un archivo, transforma al documento en un archivo **modificado**.
- El hecho de que un archivo esté modificado no implica que `git` haga un seguimiento automático del mismo. Hay que **actualizar** (en inglés, *stage*) los archivos modificados para luego consolidar esos cambios, es decir, tomar un nuevo *snapshot*.
- Al hacer esto, los archivos que estaban actualizados ahora forman parte del nuevo *snapshot*, que pasa a ser el nuevo `HEAD` del repositorio. De esta manera, los archivos que se encontraban en el estado actualizado pasan al estado no-modificado.
- Finalmente, si creamos un archivo nuevo y le queremos dar seguimiento, tenemos que agregarlo al repositorio. De la misma forma, podemos remover un archivo del repo.

git (hands-on)

Primero, vamos a crear un directorio de trabajo:

```
# mkdir ejercicio-git
```

Ahora sí, vamos al directorio de trabajo y creamos un repositorio:

```
# cd ejercicio-git
# git init
```

Esto genera una carpeta oculta dentro del directorio en el que estamos trabajando y lo podemos visualizar con los comandos:

```
# ls -a
```

El estado del repositorio se consulta ejecutando:

```
# git status
```

git (hands-on)

Dado que todavía no agregamos archivos para seguir en el repositorio, este comando nos dirá que el repositorio está vacío.
Ahora, creen/copien un archivo cualquiera en el directorio.

```
# echo 'hola' > hola.txt
```

¿Cómo iniciamos el seguimiento de archivos?

```
# git add .
```

Una vez que empezamos a seguir todos los archivos en la carpeta debemos *consolidar* los cambios:

```
# git commit -m "Agrego un archivo al repositorio"
```

Cada *commit* tiene un *hash*, que es un número hexadecimal con el que se reconoce unívocamente el *snapshot* que estamos tomando. En mi caso:

```
# 0ae96ca09bb0293af61e3e8a6de1a198a8c6459f
```

git (hands-on)

Si ahora consultamos el estado del repositorio:

```
# git status
```

```
# nothing to commit, working tree clean
```

El mensaje nos dice que no hay archivos en estado modificado, ya que el archivo que agregamos antes (y que no volvimos a modificar) es igual al que está guardado en HEAD.

git (hands-on)

Ahora, la tarea para generar un *snapshot* es siempre la misma:

- Modificamos/creamos uno o varios archivos.
- Los agregamos al staging area con `git add .`
- Los consolidamos en un snapshot con `git commit -m "mensaje"`

Entonces ahora agreguen/creen un par de archivos más y hagan otro `commit`

git (hands-on)

Podemos a todo momento consultar el historial de *commits* realizados en el repositorio:

```
# git log
```

```
# commit 0ae96ca09bb0293af61e3e8a6de1a198a8c6459f (HEAD -> master,
origin/master)
# Author: Nico Lois <nico.harry.lois@gmail.com>
# Date: Tue Jul 28 16:59:12 2020 -0300
# agrego archivo
```

Cada commit tendrá un código con el cual uno puede ir a *snapshots* viejos:

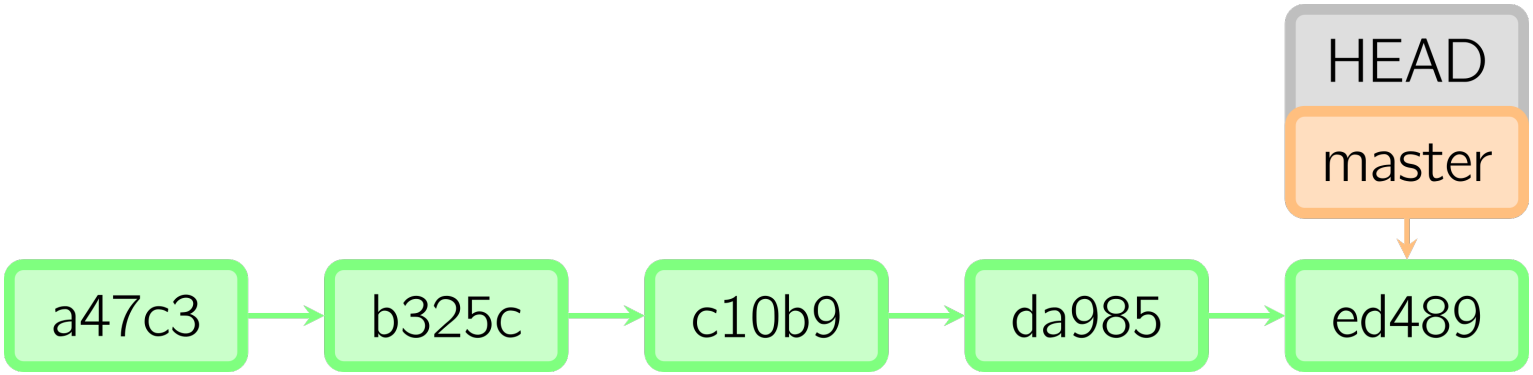
```
# git checkout <commit hash>
```

En mi caso:

```
# git checkout 9f8b
```

git: Ramas (Branches)

Hasta ahora, nuestro trabajo puede resumirse en una línea de tiempo de este estilo:

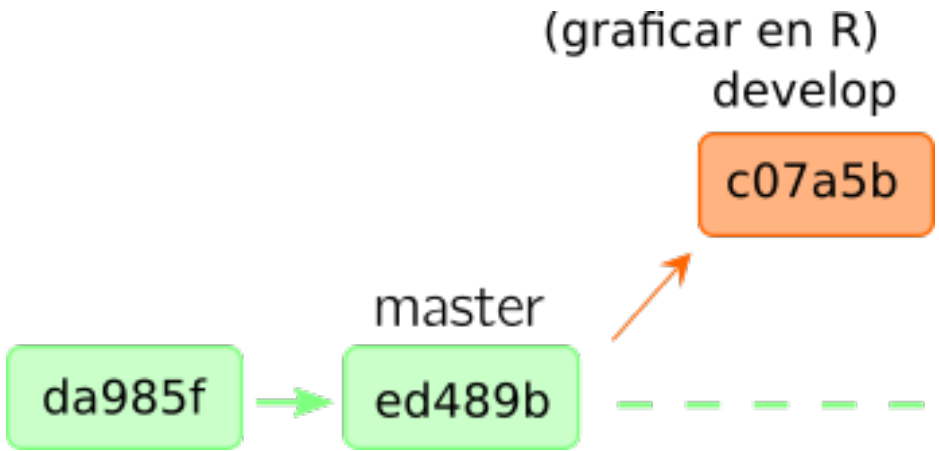


El trabajo se realizó siempre en la rama *master*

git: Ramas (Branches)

Ahora veamos cómo implementar cambios grandes que pueden comprometer la funcionalidad general de todo el repositorio. Por ejemplo, agregar una nueva funcionalidad

- graficar una serie de variables
- agregar un paquete de análisis nuevo.

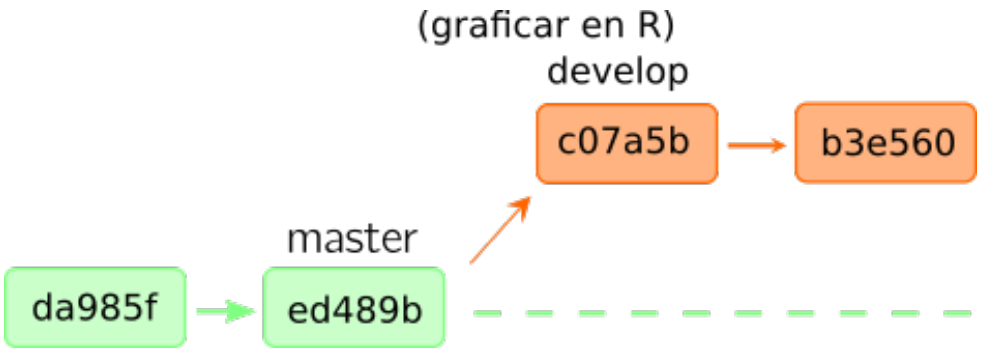


git: Ramas (Branches)

Con `git` podemos dar siguimiento a una nueva rama de la siguiente manera: generamos una rama llamada **develop** y nos posicionamos en esa rama

```
# git branch develop
# git checkout develop
```

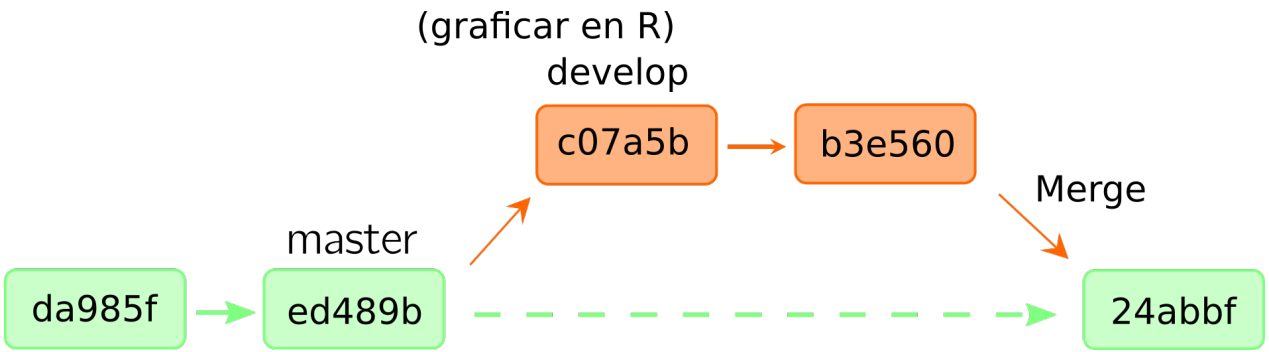
Luego, seguimos trabajando de la misma manera que vimos anteriormente, y los *commits*, se realizarán sobre la rama **develop** y la rama **master** no tendrá *commits*.



git: Ramas (Branches)

El siguiente paso será juntar estar dos implementaciones (al fin y al cabo, es lo que queremos). Para eso, debemos unir las dos ramas.

```
# git merge master develop
```



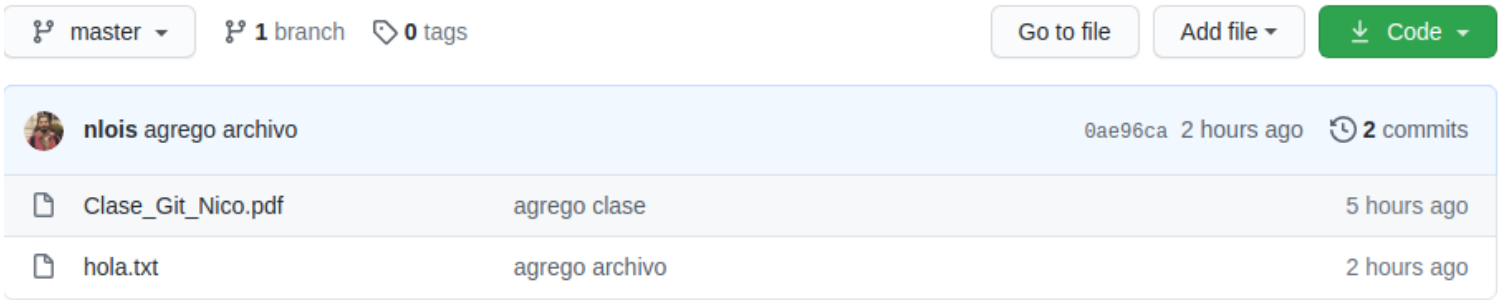
git: repositorios remotos

El último paso es poder comunicarnos con un repositorio remoto

- Crear un usuario en `github.com`
- Crear un repositorio en github, que llamaremos `ejercicio-git`, la dirección del repo será: `https://github.com/<USUARIO>/ejercicio-git.git`
- Nos comunicamos con el repositorio remoto: `git remote add origin https://github.com/dbrisaro/ejercicio-git.git`
- *Pusheamos* los cambios del local al remoto `git push -u origin master`
- Si estamos colaborando, puede el repositorio remoto tener cambios que no están en el local, para eso realizamos `git pull origin master`

git: repositorios remotos

Algo que solemos hacer es querer utilizar repositorios remotos públicos. Hay varias formas para obtener los repositorios:



- Abrir el archivo en el repositorio remoto y copiarlo.
- Descargar el repositorio desde el botón verde **Code**
- Utilizar una línea de comando

```
# git clone https://github.com/nlois/ejercicio-git.git
```