Git básico para científicos

Kiko Correoso, 2020-02-13

Las figuras originales son de Pybonacci, que las creó utilizando los iconos de FontAwesome sin modificar.

Git es un sistema de control distribuido para poder hacer seguimiento de cambios en código fuente (y otras cosas). En palabras más mundanas, es un vigilante de las cosas que vamos haciendo (cambios de código, cambios de nombres de ficheros…) en un carpeta de ficheros en un ordenador.

1. CREAR ENTORNO DE ANACONDA E INSTALAR GIT

Hay varias formas de instalar Git. El que aquí se presenta funciona en todos los sistemas operativos comunes (Windows, Linux, Mac)

```
conda create -n tools
conda activate tools
conda install -c conda-forge git
```

2. IDENTIFICACIÓN Y LOS REPOSITORIOS

2.1. Identificarse en mi repositorio local (origin)

En primer lugar, es necesario configurar git para que aparezcamos como los autores cuando estamos interactuando con el repositorio de código.

```
git config --global user.name "KikoCorreoso"
git config --global user.email KikoCorreoso@example.com
```

Con estos dos comandos se ha definido el nombre de usuario y el correo de forma global y será el que se use desde la máquina en la que me encuentro.

2.2. Definir un repositorio remoto (master)

```
git remote add origin https://github.com/kikocorreoso/pyboqt.git
```

Para verificar que hemos añadido correctamente el repositorio remoto podemos hacer uso de:

```
git remote -v
```

2.3. Crear un repositorio local (origin)

```
mkdir mi_repo
cd mi_repo
git init
```

Con esto hemos creado una carpeta llamada "mi_repo" y desde dentro de la misma he inicializado un repositorio indicándole a git que quiero que siga los cambios que se produzcan en esa carpeta.

2.4. 'Clonar' un repositorio ya existente

```
git clone https://github.com/kikocorreoso/pyboqt.git
cd pyboqt
```

Con lo anterior hemos clonado un repositorio ya existente que se encuentra en github.com, en la cuenta de kikocorreoso y con nombre pyboqt. La carpeta descargada y su contenido tiene seguimiento por parte de git.

2.5. Marcar archivo para seguimiento (añadir al índice)

```
git add nombre_del_fichero
```

2.6. Marcar todos los archivos para que git le haga el seguimiento

```
git add *
```

2.7. Desmarcar archivo para que cese el seguimiento

```
git rm nombre_del_fichero
```

2.8. Informar de los cambios con un mensaje descriptivo

```
git commit -m "Cambios en la función x para que reciba el parámetro j"
```

2.9. Enviar cambios del repositorio local al remoto (master)

```
git push origin master
```

3. SEGUIMIENTO

3.1. Mostrar el estado del repositorio (estado actual de las cosas)

```
git status
```

3.2. Mostrar el seguimiento de tu actividad con git (log)

```
git log
```

3.3. Descarga toda la información del repositorio remoto

descarga la información del repositorio remoto, pero no hace nada con ello. Solo la tenemos ahí para inspeccionarla

```
git fetch
```

4. RAMAS DE DESARROLLO

Imagina ahora que quieres añadir una nueva rama de código para comprobar la viabilidad de añadir una nueva funcionalidad o para trabajar en un reformateo de código o para trabajar en un bug.

4.1. Crear rama

```
git branch nombre_de_la_rama
```

4.2. Ver la lista de ramas

```
git branch
```

4.3. Seleccionar la rama que acabamos de crear

```
git checkout nombre_de_la_rama
```

4.4. Enviar rama al repositorio remoto para continuar más tarde

```
git push origin nombre_de_la_rama
```

4.5. Mandar todas las ramas locales (origin) al remoto (master)

```
git push --all origin
```

4.6. Eliminar una rama en el repositorio local (origin)

```
git branch -d nombre_de_la_rama
```

4.7. Eliminar una rama en el repositorio remoto (master)

```
git push origin --delete nombre_de_la_rama
```

4.8. Mostrar las diferencias entre fuentes de datos (ej. Ramas)

```
git diff
```

4.9. Añadir funcionalidad completada a la rama principal

```
git merge nombre_de_la_rama
```

4.10. Deshacer cambios

```
git checkout -- nombre_del_fichero
```

5. Actualizar el repositorio local con el contenido del remoto

```
git pull
```

git pull puede ser un comando destructivo puesto que hace dos cosas en una, git fetch y git merge, es decir, descarga datos del repositorio remoto y los intenta unir a los del local. Si al hacer la unión aparecen problemas pueden complicarse las cosas. En general es más seguro usar primero usar esos dos comandos por separado.

6. ETIQUETAS

6.1. Etiquetar punto en el desarrollo en el repositorio local

```
git tag 1.0.0 id_del_commit
```

Lo último, id_del_commit, no es estrictamente necesario y usará el último "commit". Podemos añadir un mensaje también usando:

```
git tag -a 1.0.0 -m "Final 1.0.0 version"
```

6.2. Etiquetar punto en el desarrollo en el repositorio remoto

```
git push --tags origin
```

6.3. Borrar etiquetas en el repositorio local

```
git tag -d nombre_del_tag
```

6.4. Borrar etiquetas en el repositorio remoto

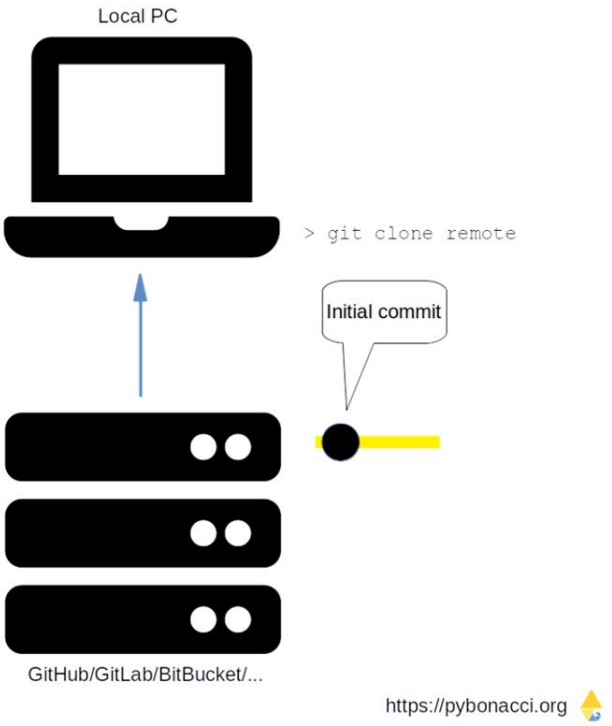
```
git push --delete origin nombre_del_tag
```

7. PRÁCTICA

7.1. Clonar, seguimiento y añadir módulo al repositorio remoto

Vamos a hacer uso de algunos de los comandos anteriores con un repositorio nuevo. El repositorio, en muchas ocasiones lo creamos primero en el repositorio remoto donde centralizaremos el desarrollo. El que yo he creado para este ejemplo se llama pybogit y está en github. Para traernos el repositorio a nuestro PC local uso:

```
git clone https://github.com/kikocorreoso/pybogit.git
```



Ahora que ya lo tenemos en nuestro PC nos movemos a esa carpeta:

```
cd pybogit
```

Y ahí podemos crear una carpeta llamada src y dentro de esa carpeta podemos crear un fichero que se llame mi_modulo.py:

```
mkdir src
touch src/mi_modulo.py
```

Lo anterior habrá creado una carpeta y un fichero dentro de ella. Lo podéis hacer de otras formas. En windows, por ejemplo, podéis usar el notepad. Si ahora le preguntamos a git por el estátus del repositorio nos dirá:

```
git status

En la rama master
Tu rama está actualizada con 'origin/master'.
Archivos sin seguimiento:
  (usa "git add ..." para incluirlo a lo que se será confirmado)
    src/
no hay nada agregado al commit pero hay archivos sin seguimiento presentes (usa "git add" para hacerles seguimiento)
```

El anterior mensaje me dice que me encuentro en la rama master y que tengo cosas en la carpeta que no están bajo el seguimiento de git. Para decirle a git que tenga en cuenta el nuevo fichero puedo hacer, por ejemplo:

```
git add src/mi_modulo.py
```

Si volvemos a mirar el estátus nos dirá:

```
git status

Tu rama está actualizada con 'origin/master'.
Cambios a ser confirmados:
 (usa "git restore --staged ..." para sacar del área de stage)
   nuevo archivo:  src/mi_modulo.py
```

Me indica que he añadido un nuevo fichero al repositorio, pero todavía no se han confirmado los cambios a git. Para confirmarlos hacemos un commit:

```
git commit -m "commit con mensaje super util"
```

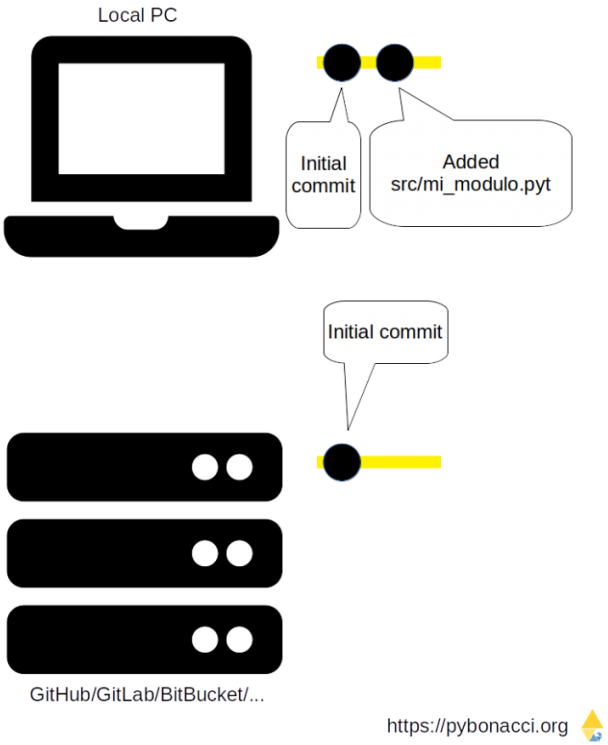
En la consola podré leer un mensaje parecido a:

```
[master d296fbe] commit con mensaje super util
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 src/mi_modulo.py
```

Si miramos de nuevo el estatus, vemos que los cambios han sido integrados:

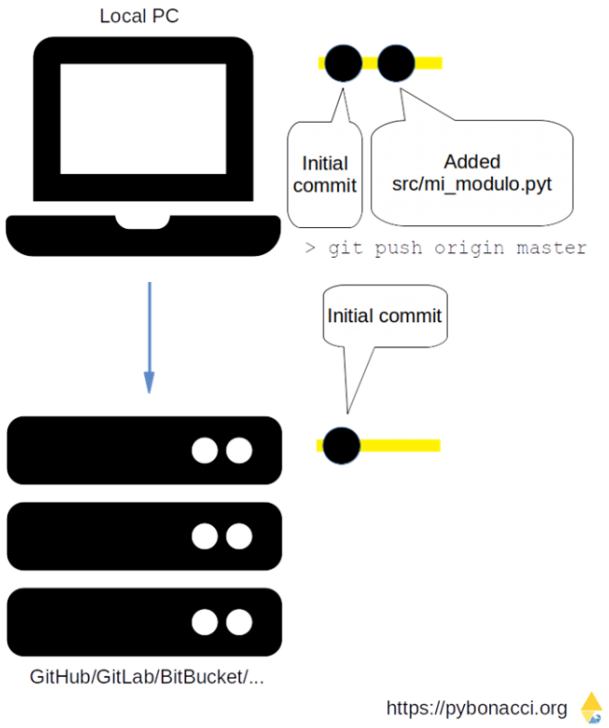
```
git status

En la rama master
Tu rama está adelantada a 'origin/master' por 1 commit.
 (usa "git push" para publicar tus commits locales)
nada para hacer commit, el árbol de trabajo está limpio
```

Pero me dice que están en mi rama local y que la remota está un 'cambio' (commit) por detrás.

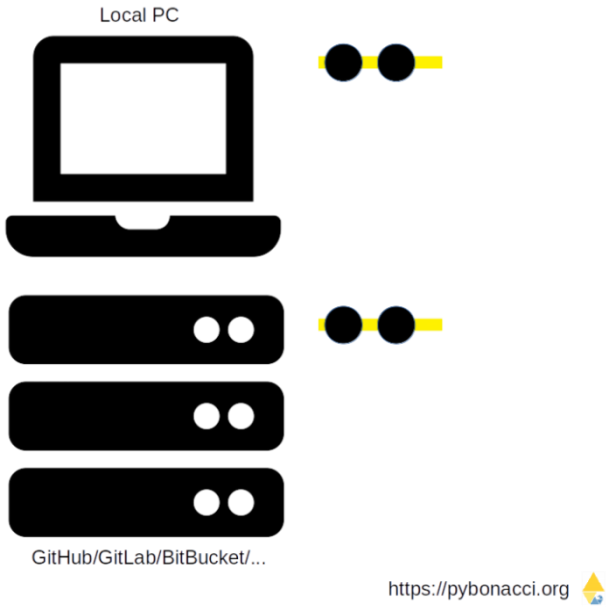


En este momento podríamos mandar los cambios al repositorio remoto (siempre es buena idea hacer esto). Lo podemos hacer con:

```
git push origin master
```



Y ahora estaríamos en esta situación:



Es decir, tenemos la misma información en local y en remoto. Vamos a añadir el siguiente código al fichero que hemos estado usando:

```
def greet():
   return "Hello"
```

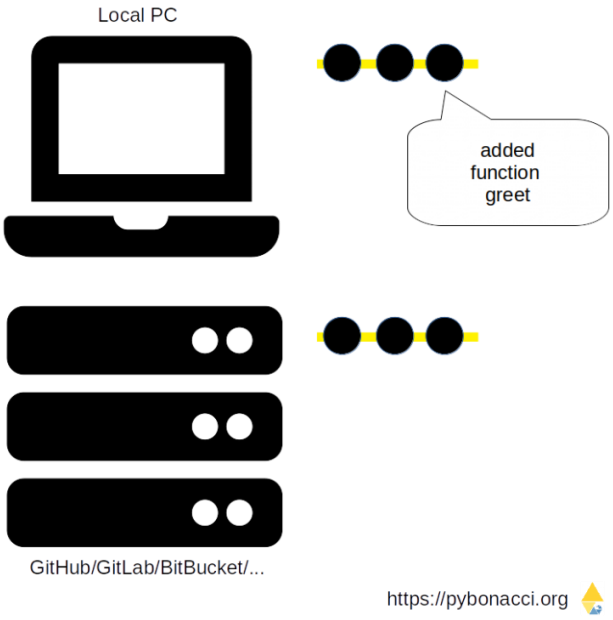
Guardo el fichero y si miro el estátus veré que git me dice que:

```
git status

En la rama master
Tu rama está actualizada con 'origin/master'.
Cambios no rastreados para el commit:
 (usa "git add ..." para actualizar lo que será confirmado)
 (usa "git restore ..." para descartar los cambios en el directorio de trabajo)
   modificado:    src/mi_modulo.py
sin cambios agregados al commit (usa "git add" y/o "git commit -a")
```

Añadimos y 'commiteamos' (sí, la jerga es la que es) y lo mandamos al remoto:

```
git add *
git commit -m "added function greet"
git push origin master
```

La situación actual sería:



7.2. Crear rama para añadir una función

Se me ocurre ahora añadir una función experimental. Voy a crear una rama nueva para meter esta función experimental. La rama master podéis verla como el tronco principal del árbol. Siempre partimos con una sola rama principal o master. Pero al árbol le podemos añadir ramas y vamos a aprovechar este momento para hacer eso mismo. Para crear la rama y cambiarnos a ella en un único comando podemos hacer:

```
git checkout -b nueva_rama
```

Lo anterior es un atajo para los dos siguientes comandos:

```
git branch nueva_rama
git checkout nueva_rama
```

Ahora, git considera que estamos trabajando en una nueva rama. Lo podemos ver preguntando nuevamente el estátus (git status):

```
En la rama nueva_rama
nada para hacer commit, el árbol de trabajo está limpio
```

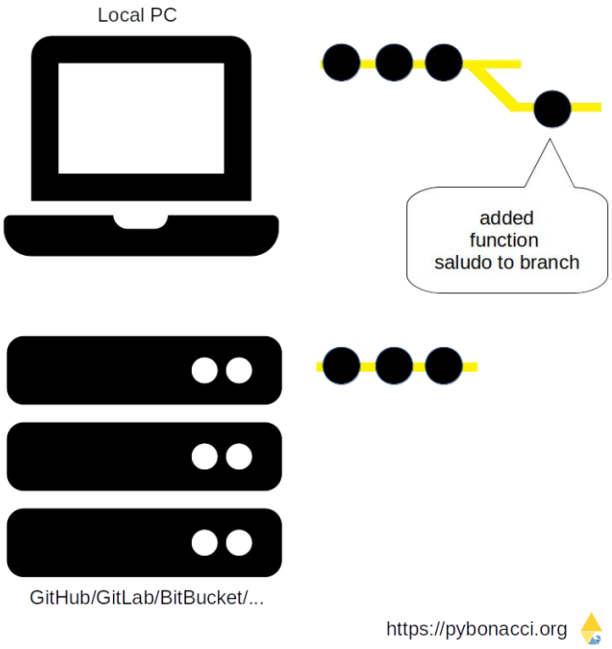
y los cambios que hagamos no irán al tronco (la rama master). Hagamos algún cambio añadiendo nuestra funcionalidad experimental. Modificamos el fichero que hemos creado anteriormente añadiendo el siguiente código:

```
def saludo():
   return "Hola"
```

Añadimos los nuevos cambios a la rama en la que nos encontramos usando:

```
git add *
git commit -m "added experiment to branch"
```

La situación actual sería algo así:



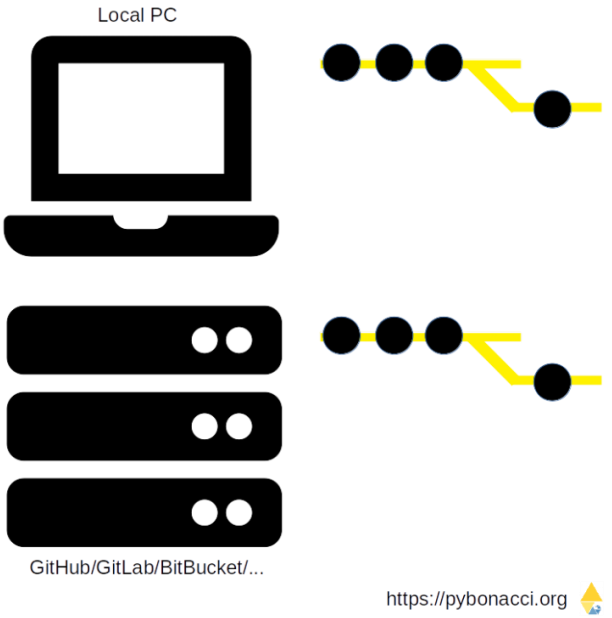
Podemos mandar todas las ramas al escritorio remoto con el siguiente comando (en este caso solo hay una, en otros habría que tener cuidado):

```
git push --all origin
```

Y me sale un mensaje parecido a:

```
Username for 'https://github.com': kikocorreoso
Password for 'https://kikocorreoso@github.com':
Enumerando objetos: 7, listo.
Contando objetos: 100% (7/7), listo.
Compresión delta usando hasta 4 hilos
Comprimiendo objetos: 100% (3/3), listo.
Escribiendo objetos: 100% (4/4), 355 bytes | 355.00 KiB/s, listo.
Total 4 (delta 1), reusado 0 (delta 0)
remote: Resolving deltas: 100% (1/1), completed with 1 local object.
remote:
remote: Create a pull request for 'nueva_rama' on GitHub by visiting:
remote: https://github.com/kikocorreoso/pybogit/pull/new/nueva_rama
remote:
To https://github.com/kikocorreoso/pybogit.git
[new branch]      nueva_rama -> nueva_rama
```

Ahora que la rama local está también en el remoto tendríamos algo así:



Y podríamos pedir a nuestros colaboradores que revisasen esa nueva rama, si creen que es útil, si mejor eliminar ese código por la razón que sea…

Hemos decidido eliminarla. La elimino del local y del remoto. Primero salgo de la rama:

```
git checkout master
```

Podemos ver las ramas que tenemos y en la que nos encontramos usando:

```
git branch
```

Que nos mostrará algo como:

```
* master
  nueva_rama
```

Con el asterisco indicando que estamos ahora en la rama master. Voy a coger la sierra y a cortar la rama nueva_rama:

```
git branch -D nueva_rama
```

La opción -D es importante que esté en mayúscula. Probadlo primero en minúscula. Cojo la sierra y me la llevo hasta el servidor y corto la rama en el remoto:

```
git push origin --delete nueva_rama
```

Ahora podéis mirar las ramas que os quedan y deberíais ver solo la rama master.