Git tag

Con el comando git tag (equivalente a git tag -l o git tag --list) veremos los tags que tenemos existentes

Si queremos crearlos, lo hacemos con git tag -a miEtiqueta -m "el mensaje que quedamos que acompañe a la etiqueta.

$ git tag -a v0.1 -m "texto que queremos poner, como por ejemplo version 0.1"  
$ git tag  
  
v0.1

Ese tag que acabamos de crear estará vinculado al git commit que hayamos hecho anteriormente.

Git show

Una vez creado el tag después del commit correspondiente, podremos explorar cualquier tag con git show donde no sólo veremos las etiquetas sino también los cambios a nivel de código que se han producido.

$ git tag -a v0.1 -m "texto que queremos poner, como por ejemplo version 0.1"  
$ git show v0.1  
  
tag kuworking@0.1  
Tagger: kuworking  
Date: Tue Jun 20 08:43:10 2019 +0200  
  
kuworking@0.1  
  
commit 280a654f0e63fec63457ffdee95c4779e917adre (tag: site@0.1, tag: kuworking@0.1)  
Author: kuworking  
Date: Tue Jun 20 08:43:10 2019 +0200  
  
 Publish  
  
 - kuworking@0.1  
 - site@0.1  
  
diff --git a/packages/kuworking/package.json b/packages/kuworking/package.json  
index 6ede0d8..19a93f2 100644  
--- a/packages/kuworking/package.json  
+++ b/packages/kuworking/package.json  
@@ -1,6 +1,6 @@  
 {  
 "name": "kuworking",  
- "version": "0.01",  
+ "version": "0.1",  
 "private": true,  
 "description": "kuworking.com",  
 "author": "kuworking",  
 ...

Git tag (ligero)

$ git tag v0.1  
$ git show v0.1

La diferencia es que el vínculo entre el tag y el commit no será tan permanente, pero igualmente podremos apuntar a ese tag si nos interesa.

Borrar un tag

Borrar un tag por un error o por una rutina que se ha interrumpido inesperadamente es algo que podemos hacer de forma sencilla con.

$ git tag -d v0.1  
  
Deleted tag 'v0.1' (was 9165190)

Esto sólo borra el tag a nivel local, por lo que si antes habíamos sincronizado con push también necesitaremos borrar el tag en remote.

$ git push origin --delete v0.1

Y para ver los tags en remote, haríamos.

$ git ls-remote --tags origin  
e8a0cc653954551c4838966d1c1178110892a802 refs/tags/v0.0.2  
81101b87b2e02dcfefa326cbe35ae1e6904b2bca refs/tags/v0.0.3  
4287f60d08a9ce19e27fc1f67e546f6b1300ad1b refs/tags/v0.0.7

Cuando vale la pena hacer el git tag?

El gran uso de los tags es cambiar el identificador de un commit con un texto tipo 0d52aaab4479697da7686c15f77a3d64d9165190, por un texto tipo draft01, mucho más amigable.

¿Y para qué queremos commits que tengan nombres "humanos"?

Porque podemos movernos a ese commit específico, y a partir de ahí hacer cambios en ese código y crear una rama paralela, o simplemente explorar el código y listos.

Otra aplicación de tener un tag específico es que podremos instalar esa particular versión de forma muy fácil en package.json.

Lo que nos ahorramos aquí es el tener que poner ese hash tan poco amigable de antes (y tan rebuscado de encontrar, por otro lado).

"myPublicPackage": "git+https://github.com/myUser/myPublicPackage.git#v0.0.14",  
"myPrivatePackage": "git+https://miTokenParaAcceder:x-oauth-basic@github.com/myUser/myPrivatePackage.git#v0.0.14",

Esto es muy útil cuando estemos utilizando el git de forma directa (aquí en github), en lugar de publicar los paquetes via npm.

En resumen, con git tenemos un control de nuestro desarrollo, y con git tag podemos dotar de significado a los puntos de control del desarrollo (los commits), es decir, facilitar la identificación de distintos commits para su futura referencia.

Lo suyo no es poner etiquetas a todos los commits, sólo a aquellos que representen estados que merezcan la pena ser remarcados.

Git fork

El fork que nos permite hacer GitHub es muy similar a clonar un repositorio con la ventaja de que en lugar de clonar el repositorio de forma local, podremos crear una copia en remoto del proyecto que queremos y quedara almacenada en nuestra cuenta personal de GitHub.

En la parte superior derecha de la pantalla podemos hacer click en el botón FORK y así de simple se creará un repositorio en nuestra cuenta de GitHub que será una copia del repositorio anterior.

Trabajando en local con un repositorio Fork

El comportamiento de este nuevo repositorio sera el mismo de cualquier otro que hayamos creado, puede ser clonado, se pueden hacer cambios, registrar commits, etc. Ya que contamos con el repositorio remoto podemos crear una carpeta en local, iniciar git y hacer pull de los cambios, recuerda usar la url de tu nuevo proyecto.

$ mkdir presentacion-fork

$ cd presentacion-fork

$ git init

$ git add origin master url\_de\_tu\_repo\_en\_GitHub

$ git pull origin master

De esta forma todos los cambios commits que realicemos podrán ser enviados al repositorio remoto sin ningún problema.

Nota: los cambios enviados serán almacenados en nuestra copia del repositorio y no afectarán de ninguna forma el repositorio original que hemos clonado.

Aclarando diferencias entre Clone y Fork

Con git clone podemos copiar el código de un repositorio de forma local, pero no estará conectado al repositorio de origen.

Con Fork podremos crear una copia en remoto de un repositorio git en nuestra cuenta de GitHub y de esa forma será más fácil clonar de forma local para hacer pull y push de los cambios realizados.

