Actividad opcional 1. Comandos Git

Índice

1. Comando git tag

El comando git tag se utiliza para crear, listar y gestionar las etiquetas (tags) en un repositorio git. Las etiquetas son punteros inmutables a confirmaciones específicas (commit) en la historia del repositorio.

Al añadir una etiqueta se añade en refs/tags/, a menos que se utilice el comando eliminar (-d), listar (-l) o verificar (-v) etiquetas.

Hay dos tipos de etiquetas:

- Etiquetas ligeras que son simplemente un nombre para un commit.

- Etiquetas anotadas, que contienen una fecha de creación, un nombre de la persona que la crea, un correo electrónico y de manera opcional, una firma GnuGP.

Las etiquetas anotadas tienen la intención de utilizarse para las versiones entregables, mientras que las ligeras se utilizan para etiquetas de objetos temporales. Por esa razón, algunos comandos para nomenclatura de objetos ignoran por defecto las etiquetas ligeras.

2. Opciones más útiles del comando git tag

-a
--annotate
Realiza una etiqueta sin firmar y anotada.

-d
--delete
Elimina las etiquetas que existen con un nombre dado.

-f
--force
Remplaza una etiqueta con el nombre dado en lugar de dar un error. Esta opción se utiliza para mover una etiqueta ya creada a un nuevo commit.

-m <msg>
--message=<msg>
Permite añadir el mensaje sin la necesidad que lo solicite cuando se crea una etiqueta anotada.

-n<num>
<num> especifica cuanta lineas del la anotación, si hay alguna, se muestran cuando se usa el comando -l (implica –list). Por defecto no muestra ninguna línea de anotación. Si ningún numero se añade a la opción -n, sólo muestra la primera línea. Si la etiqueta no es anotada, en su lugar se muestra el mensaje del commit.

-l
--list
Lista las etiquetas del repositorio en orden alfabético. Con el opcional <patrón> se listan únicamente las etiquetas que cumplen el patrón. Se pueden pasar varios patrones, y se mostraran todas las etiquetas que cumplan alguno de los patrones.

3. Ejemplos más útiles de uso del comando git tag

- **Crear etiqueta:**
  - Crear una etiqueta ligera

    ```
    git tag etiqueta
    ```

  - Crear una etiqueta anotada (-a) con un mensaje (-m)

    ```
    git tag –a etiqueta –m “Mi etiqueta”
    ```

  - Crear una etiqueta tardía mucho tiempo después de haber creado el commit

    ```
    git tag –a etiqueta <commit>
    ```

- **Listar etiquetas:**

  El comando lista las etiquetas en orden alfabético y también tenemos la opción de listar las etiquetas que siguen un determinado patrón.

  - Listar todas las etiquetas en el repositorio

    ```
    git tag | git tag –l
    ```

  - Listar etiquetas que coinciden con un patrón

    ```
    git tag –l “patrón” | git tag –l “v1.*”
    ```

- **Mostrar información de etiquetas:**
  - Mostrar información detallada de una etiqueta

    ```
    git show etiqueta
    ```

- **Eliminar una etiqueta:**
  - Eliminar una etiqueta local

    ```
    git tag –d etiqueta
    ```

  - Eliminar una etiqueta remota

    ```
    git push remote –d etiqueta
    ```

- **Compartir etiquetas:**

Las etiquetas no se comparten con el servidor remoto de manera automática sino que se tiene que hacer de manera explícita, por etiqueta o todas a la vez.

- Compartir una etiqueta local con el repositorio remoto después de haberla creado

  ```
  git push remote etiqueta
  ```

- Compartir todas las etiquetas que todavía no existen en el repositorio remoto

  ```
  git push remote --tags
  ```

- **Utilizar los tags creados:**
  - Moverse al estado del repositorio en un tag

    ```
    git checkout etiqueta
    ```

    Este comando pone el repositorio en un estado separado del HEAD. Esto significa que cualquier cambio hecho no actualizará la etiqueta. Creará un nuevo commit separado, que no será parte de ninguna rama y sólo será accesible directamente por el hash SHA del commit. Por lo que ser recomienda crear una rama cuando se realicen cambios de un estado HEAD separado.

  - Crear una nueva rama a partir de un tag

    ```
    git checkout tags/etiqueta -b etiqueta-rama
    ```

4. Casos de uso

Las etiquetas de git son utilizados para marcar puntos específicos en la historia del repositorio.

Aquí se describen algunos casos de uso:

- Marcar las versiones que pasan a producción, como por ejemplo con v0.0.1, v1.0.0..
- Marcar puntos estables en el proceso de desarrollo, que indican commits estables y se conoce que no hay errores.
- Colaboración con otros desarrolladores, para que puedan revisar
- Testing, marcar puntos de testeo en procesos de integración continua.

5. Ejemplo práctico

Imaginamos un repositorio de código en Git donde se desarrolla una aplicación donde se han realizado varias confirmaciones y se han etiquetado las versiones de código que se han publicado en el entorno de producción a lo largo del tiempo. En la aplicación únicamente trabaja un único desarrollador por lo que no se pueden dar conflictos, salvo descuido del propio desarrollador. Se descubre un error de código (bug) en la versión más reciente (v1.3.0) y se desea recuperar la versión de código anterior donde el bug no existía para que se pueda seguir trabajando normalmente, mientras se repara ese error de código. Una vez el bug se ha reparado se actualiza de nuevo la versión de producción con la nueva versión de código libre de errores.

Supongamos que tenemos un repositorio con las siguientes etiquetas versiones en producción:

```
v1.0.0 – v1.1.0 – v1.2.0 – v1.3.0 (bug)
```

Podemos ver un listado de los últimos commits y conocer el estado del repositorio:

```
$ git log –oneline
995762c (HEAD -> master, origin/master, origin/dev, dev) Merge branch ‘dev’ into ‘master’
bb9c654 (tag: v1.3.0) Feature 10
8b1b7d9 Merge branch 'dev' into ‘master’
2123735 (tag: v1.2.0) Feature 9
e18c8de Merge branch 'dev' into 'master'
4ced42d (tag: v1.1.0) Feature 9
87eeef6 Merge branch 'dev' into 'master'
cabb3f4 (tag: v1.0.0) Feature 8
c7bd6da Feature 7
ac244b7 Feature 6
e49fcac Feature 5
613792c Feature 4
99c7d6b Feature 3
dd28083 Feature 2
325b0d8 Feature 1
460bfdd ...
```

1. Primero debemos hacer checkout a la etiqueta v1.2.0 para poder recuperar el código libre de errores

   ```
   $ git checkout tags/v1.2.0
   ```

   Hay que recordar que este comando hace que HEAD quede separado por lo que los cambios realizados aquí sólo serían accesibles con el hash del commit.

2. Con el HEAD apuntando a la etiqueta v1.2.0 se puede generar la versión para subir a producción libre de errores.

3. Ahora volvemos a la rama master o a la etiqueta v1.3.0 donde se encuentra el error a corregir, que están alineadas.

   ```
   $ git switch master
   ```

4. Se recomienda crear una rama para arreglar el error

   ```
   $ git checkout –b hotfix
   ```

5. Detectamos el error y lo arreglamos el error con un nuevo commit en la rama hotfix

   ```
   $ git add filewitherror
   ```

   ```
   $ git commit –m “Fixed error v1.3.0”
   ```

6. Podemos etiquetar este commit con la nueva versión que subirá a producción v1.3.1

   ```
   $ git tab –a v1.3.1 –m “Fixed error de producción v1.3.0”
   ```

7. Cambiamos de nuevo a la rama master

   ```
   $ git switch master
   ```

8. Realizamos el merge de la rama hotfix con la master

   ```
   $ git merge hotfix
   ```

9. Podemos realizar la subida de la nueva versión a producción, habiendo solucionado el problema.