Git y GitHub

Arturo Rivas Rojas

14/12/18

Contents

Chapter 1

Curso Profesional de Git

1.1 Sistema Control de Versiones

Sistema que registra los cambios realizados sobre un archivo o conjunto de archivos a lo largo del tiempo.

1.1.1 Clasificación

1.1.1.1 Locales

La información, las versiones, solo existe en una sola computadora.

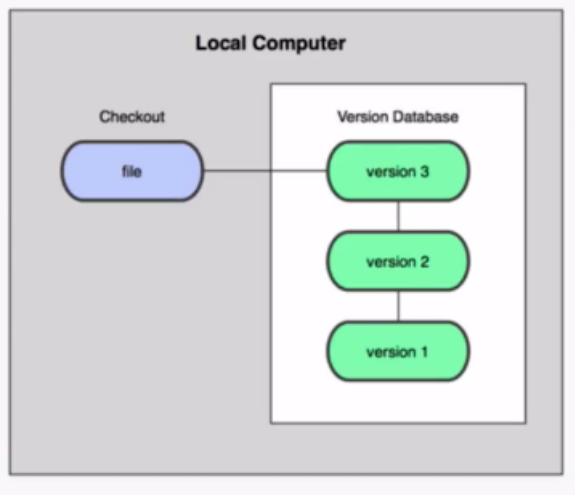

Figure 1.1: Sistema Local

1.1.1.2 Centralizados

Los cambios se guardan en un servidor que comparte con todos sus subscriptores los cambios

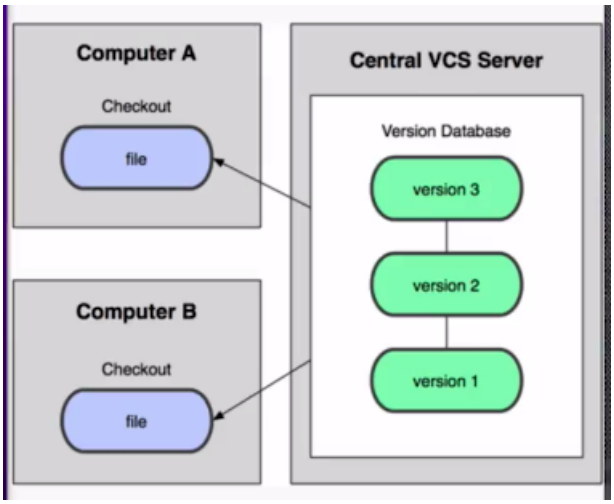

Figure 1.2: Sistema centralizado

1.1.1.3 Distribuidos (Git)

Se guarda una copia del repositorio en la coputadora de cada participante. Que operan de manera independiente y puede mezclarce en cualquier momento.

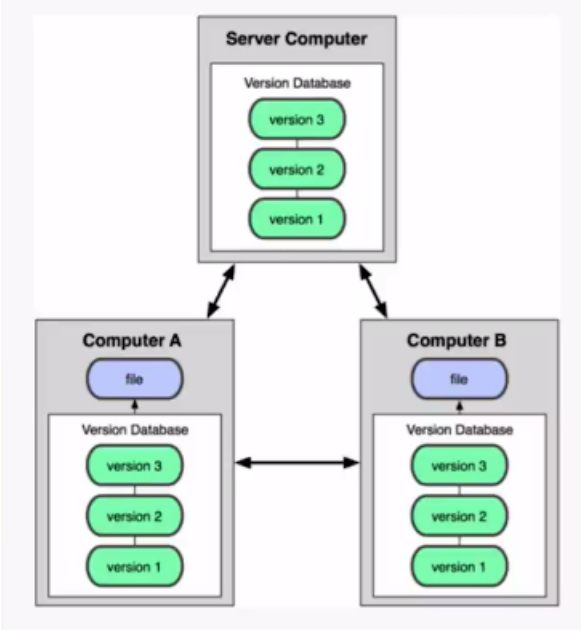

Figure 1.3: Sistema distribuido

1.1.2 Beneficios de usar Git

- Trabajar offline
- Integridad de los datos

1.1.3 Estados de Git

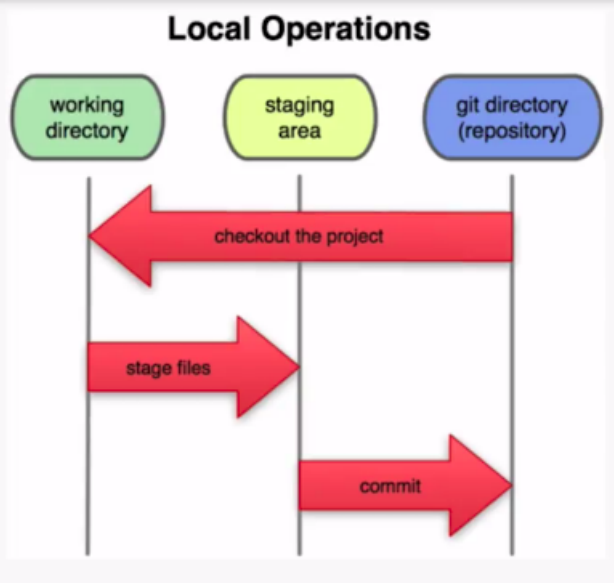

Figure 1.4: Los 3 estados de Git

1.1.3.1 Working directory

Es el directorio local en donde se guarda la copia del repositorio. Se agregan o se hacen cambios a los archivos de manera local.

1.1.3.2 Staging area

Se realiza un add para preparar el archivo para su commit.

1.1.3.3 Git directory (repositorio)

Se realiza el commit y se distribuyen los cambios realizados hacia las copias.

1.1.3.3.1 SHA-1

Secure Hash Algorithm, Algoritmo de Hash Seguro, es un conjunto de numeros cifrados que sirven como id de lso commits.

1.1.4 GitHub

Git y GitHub no son lo mismo.
Git es el software de control de cambios mientras que GitHub es la plataforma host de los repositorios.
Aparte GitHub aparte del control de versiones tiene muchas mas herramientas para la organización.

1.1.5 Manejo de archivos en la terminal

- **cd:** cambiar de directorio.
    - ejemplo: cd Desktop
- **mkdir:** crear directorio.
    - ejemplo: mkdir apuntes
- **ls:** lista de archivos.
    - ejemplo: ls
- **touch:** crear archivo.
    - ejemplo: touch file.txt
- **rm:** eliminar archivos y directorios.
    - ejemplo: rm file.txt, rm -r apuntes

1.2 Comandos de Git

1.2.1 Inicializar repositorios

- git **init** [nombre] : inicializa un repositorio.
- git **init** : dentro de la carpeta que queremos que sea el repositorio.

1.2.2 Trabajo en el workplace

- git **status** : muestra la rama y los cambios sin confirmar.
- git **add** [archivo o directorio] :** pasar los cambios al estado staging.
- git **rm --cached** [archivo o directorio] : borrar los cambios del git.
- git **commit** : confimar los cambios a git.
    - **-m** [“mensaje”] : asociar un mensaje para nuestro commit.
    - **--amend** : concatenar el commit con el anterior.

1.2.3 Nombrar Versiones

- git **tag** [etiqueta] : etiquetar o nombrar version actual del repositorio.
- git **tag** [etiqueta] **sha-1(codigo del commit)** : etiquetar o nombrar version actual del repositorio.
    - **-a:** etiqueta
    - **-m:** mensaje
    - **-f:** renombrar
    - **-d:** eliminar

1.2.4 Bitacora de commits

- git **log** : lista de commits con su respectivo sha-1, autor, fecha y mensaje. Para salir basta con presionar **q**.
  - **--oneline** : infomación reducida a una linea.
  - **--graph** : grafico con respecto a las ramas.
  - **-#** : numero de commits que quieres visualizar comenzando desde el ultimo.
  - **> [archivo]** :respalda el log en el archivo.

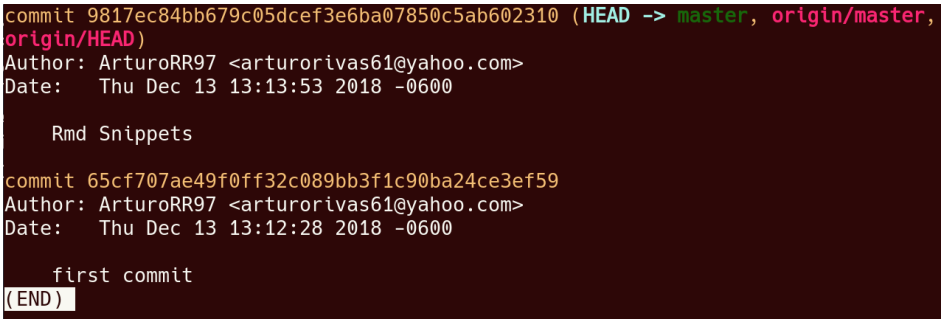

Figure 1.5: imagen de la bitacora

Figure 1.6: imagen de la bitacora con todas las banderas sin banderas

1.2.5 Differencias entre commits

- git **diff** [commit] : permite ver las differencias entre el estado actual y el commit.
- git **diff** [commit1] [commit2]: permite ver las differencias entre los commits que se pasaron como parametros. El orden de los parametros afecta el resultado. Se muestran las acciones que tendrian que pasar para llegar del primer commit al segundo.

1.2.6 Borrar o reescribir

- git **reset** : reescribe la historia del proyecto.
  - **--soft** [commit] : se resetea al commit del sha-1, los commits posteriores a este se borran. Sin borrar archivos y dejando los cambios que ya se habian confirmado en staging.
  - **--mixed** [commit] : a diferencia del **--soft** descarta completamente los commits regresandolos al estado 1, Working directory.
  - **--hard** [commit] : elimina los cambios tando confirmados o en staging. Tambien funciona de manera inversa es decir si se hizo un **reset --hard** al inicio del repositorio y luego se hace un **reset --hard** al ultimo commit, este se recupera.
  - **HEAD** [archivo] : quita de staging al archivo seleccionado.

1.3 Ramas (branches)

Son las versiones alternas que pude tener nuestro proyecto en git

1.3.1 Master

Es la rama principal y se genera por defecto.

1.3.2 Ramas Paralelas

Es la forma correcta de trabajar en git pues los bugs y las nuevas caracteristicas se programan de manera separada al proyecto pirncipal y uan vez que estan completas se agregan a la rama master.

1.3.3 Comandos para Ramas

- git **branch** [nombre de la rama] : crea la rama con el nombre dado.
    - **-l** : muestra la lista de ramas del proyecto.
    - **-d** [rama] : borrar la rama si es que no se han realizado commits
    - **-D** [rama] : borrar la rama sin importar nada.
    - **-m** [rama] [nuevo nombre] : renombrar la rama.
- git **checkout** [rama|commit] : navegar entre ramas o commits.
    - **-b** [nombre] : crear y navegar a una nueva rama.
- git **merge** [rama] : se mezcla la rama indicada con la rama en la que estamos ubicados.
    - **Fast-forward** : ya que la rama a mezclar sigió la linea temporal de la rama en la que estamos solo se agregaron los cambios sin problemas
    - **Auto-merging** : si ahora la rama a mezclar trabajo de manera paralela a la rama en la que estamos ubicados. por lo que se necesitna mezclar con cierta logica.
        * **Conflictos** : archivos que git no supo con que version quedarse y se deben resolver de manera manual.
- git **rebase** [rama] : mezcla la rama dada con la rama en la que esta ubicado pero reescribe la bitacora por lo que no pide commits extras y el grafico es lineal. Al reescribir la historia de los commits se pude alterar el trabajo de los demas por lo que es mas peligroso que el merge.