El comando git show nos muestra los cambios que han existido sobre un archivo y es muy útil para detectar cuándo se produjeron ciertos cambios, qué se rompió y cómo lo podemos solucionar. Pero podemos ser más detallados.

Si queremos ver la diferencia entre una versión y otra, no necesariamente todos los cambios desde la creación del archivo, podemos usar el comando git diff commitA commitB.

Recuerda que puedes obtener el ID de tus commits con el comando git log.Para iniciar un repositorio, o sea, activar el sistema de control de versiones de Git en tu proyecto, solo debes ejecutar el comando git init.

Este comando se encargará de dos cosas: primero, crear una carpeta .git, donde se guardará toda la base de datos con cambios atómicos de nuestro proyecto; y segundo, crear un área que conocemos como Staging, que guardará temporalmente nuestros archivos (cuando ejecutemos un comando especial para eso) y nos permitirá, más adelante, guardar estos cambios en el repositorio (también con un comando especial).

Ciclo de vida o estados de los archivos en Git:

Cuando trabajamos con Git nuestros archivos pueden vivir y moverse entre 4 diferentes estados (cuando trabajamos con repositorios remotos pueden ser más estados, pero lo estudiaremos más adelante):

Archivos Tracked: son los archivos que viven dentro de Git, no tienen cambios pendientes y sus últimas actualizaciones han sido guardadas en el repositorio gracias a los comandos git add y git commit.

Archivos Staged: son archivos en Staging. Viven dentro de Git y hay registro de ellos porque han sido afectados por el comando git add, aunque no sus últimos cambios. Git ya sabe de la existencia de estos últimos cambios, pero todavía no han sido guardados definitivamente en el repositorio porque falta ejecutar el comando git commit.

Archivos Unstaged: entiéndelos como archivos “Tracked pero Unstaged”. Son archivos que viven dentro de Git pero no han sido afectados por el comando git add ni mucho menos por git commit. Git tiene un registro de estos archivos, pero está desactualizado, sus últimas versiones solo están guardadas en el disco duro.

Archivos Untracked: son archivos que NO viven dentro de Git, solo en el disco duro. Nunca han sido afectados por git add, así que Git no tiene registros de su existencia.

Recuerda que hay un caso muy raro donde los archivos tienen dos estados al mismo tiempo: staged y untracked. Esto pasa cuando guardas los cambios de un archivo en el área de Staging (con el comando git add), pero antes de hacer commit para guardar los cambios en el repositorio haces nuevos cambios que todavía no han sido guardados en el área de Staging (en realidad, todo sigue funcionando igual pero es un poco divertido).

Comandos para mover archivos entre los estados de Git:

git status: nos permite ver el estado de todos nuestros archivos y carpetas.

git add: nos ayuda a mover archivos del Untracked o Unstaged al estado Staged. Podemos usar git nombre-del-archivo-o-carpeta para añadir archivos y carpetas individuales o git add -A para mover todos los archivos de nuestro proyecto (tanto Untrackeds como unstageds).

git reset HEAD: nos ayuda a sacar archivos del estado Staged para devolverlos a su estado anterior. Si los archivos venían de Unstaged, vuelven allí. Y lo mismo se venían de Untracked.

git commit: nos ayuda a mover archivos de Unstaged a Tracked. Esta es una ocasión especial, los archivos han sido guardados o actualizados en el repositorio. Git nos pedirá que dejemos un mensaje para recordar los cambios que hicimos y podemos usar el argumento -m para escribirlo (git commit -m "mensaje").

git rm: este comando necesita alguno de los siguientes argumentos para poder ejecutarse correctamente:

- git rm --cached: Mueve los archivos que le indiquemos al estado Untracked.

- git rm --force: Elimina los archivos de Git y del disco duro. Git guarda el registro de la existencia de los archivos, por lo que podremos recuperarlos si es necesario (pero debemos usar comandos más avanzados).

Algunos comandos que pueden ayudar cuando colaboren con proyectos muy grandes de github:

1. git log --oneline - Te muestra el id commit y el título del commit.
2. git log --decorate- Te muestra donde se encuentra el head point en el log.
3. git log --stat - Explica el número de líneas que se cambiaron brevemente.
4. git log -p- Explica el número de líneas que se cambiaron y te muestra que se cambió en el contenido.
5. git shortlog - Indica que commits ha realizado un usuario, mostrando el usuario y el titulo de sus commits.
6. git log --graph --oneline --decorate y
7. git log --pretty=format:"%cn hizo un commit %h el dia %cd" - Muestra mensajes personalizados de los commits.
8. git log -3 - Limitamos el número de commits.
9. git log --after="2018-1-2" ,
10. git log --after="today" y
11. git log --after="2018-1-2" --before="today" - Commits para localizar por fechas.
12. git log --author="Name Author" - Commits realizados por autor que cumplan exactamente con el nombre.
13. git log --grep="INVIE" - Busca los commits que cumplan tal cual está escrito entre las comillas.

14. git log --grep=“INVIE” –i- Busca los commits que cumplan sin importar mayúsculas o minúsculas.
15. git log – index.html- Busca los commits en un archivo en específico.
16. git log -S “Por contenido”- Buscar los commits con el contenido dentro del archivo.
17. git log > log.txt - guardar los logs en un archivo txt

.

Clase de branches:

Git commit –am automaticamente hace el git add de los cambios pero solo de archivos que se le ha hecho el git add previamente… pilas

Git show te muestra donde esta el master