Práctica Git.

José Antonio Arredondo Escoto.

1. Dos comando no vistos en clase.

Comadon: git log.

Muestra todas las commits en el historial del repositorio. Por defecto el comando muestra los datos de cada commit: secure hash algorithm, autor, fecha, y el mensaje del commit.

Se puede personalizar la información presentada por git utilizando las siguientes banderas:

git log --oneline: muestra un commit por línea, los primeros 7 caracteres del SHA, y el mensaje del commit.

git log --stat: muestra los archivos que se modificaron en cada confirmación, el número de líneas añadidas o eliminadas, una línea de resumen con el número total de archivos y líneas.

git --patch o --p: muestra los archivos que has modificado, ubicación de las líneas añadidas o eliminadas, y los cambios específicos que se realizaron.

Comando: git reset.

Permite restablecer tu estado actual a un estado específico. Puedes restablecer el estado de archivos específicos, así como el de toda una rama. Es útil si no has subido el commit a un repositorio remoto.

git reset (--patch | -p) [tree-ish][--][ruta]: permite elegir selectivamente trozo de contenido y revertirlos o unstage los archivos.

git reset HEAD archivo: si tenemos un archivo en stage pero ya no queremos que forme parte del commit podemos usar git reset para deshacer la acción sobre ese archivo.

git reset mode commit: permite restablecer una rama a un commit anterior. Rebobina el estado de la rama, y los siguientes commit se escribirán sobre todo lo que pasó después del reinicio.

Las opciones para mode son:

--soft: no restablece el fichero índice o el árbol de trabajo, pero restablece HEAD para commit. Cambia todos los archivos a cambios a ser commited.

--mixed: restablece el índice, pero no el árbol de trabajo e informa lo que no se ha actualizado.

--hard: restablece el índice y el árbol de trabajo. Cualquier cambio en los archivos rastreados en el árbol de trabajo desde el commit son descartados.

--merge: restablece el índice y actualiza los archivos en el árbol de trabajo que son diferentes entre el commit y HEAD, pero mantiene los que son diferentes entre el índice y el árbol de trabajo.

--keep: restablece las entradas del índice, actualiza los archivos en el árbol de trabajo que son diferentes entre commit y HEAD. Sin un archivo que es diferente entre commit y HEAD tiene cambios locales, el reinicio se aborta.

2. Git Fork qué es y cómo funciona.

Sirve para hacer una copia exacta del repositorio original que podemos utilizar como un repositorio git cualquiera. Tendremos dos repositorios git idénticos pero con distintos URL. Tiene la misma historia al momento de crearlos. Son dos repositorios independientes que pueden cada uno evolucionar de forma autónoma. Las modificaciones que se hagan en ellos no interferirán con la edición del otro. La diferencia con clone.

Su uso más común es para permitir a los desarrolladores contribuir a un proyecto de forma segura haciéndolo de la siguiente manera:

1. Un desarrollador hace un fork a un repositorio, para lo que sólo se necesita darle permiso de lectura.
2. El desarrollador trabaja en su copia que es el fork. Como es suya, puede hacer lo que quiera, es su copia no afectará la real.
3. Cuando el desarrollador termine de hacer su trabajo, avisa al autor original sobre la rama en la que estuvo trabajando para que la verifique.
4. El autor va al repositorio, mira lo que ha hecho y si es algo que el autor original considera correcto y lo valida, lo incorpora haciendo merge con el repositorio original e integrando los cambios.

Las ventajas más significativas son:

- En ningún momento se tiene acceso no deseado al repositorio original.
- El proceso de incorporación es muy sencillo, si no hay conflicto en los ficheros, un par de comandos git incorporará los cambios.
- Cuesta muy poco contribuir a proyectos y administrar las contribuciones a tus proyectos por parte de otros desarrolladores.