Git & GitHub

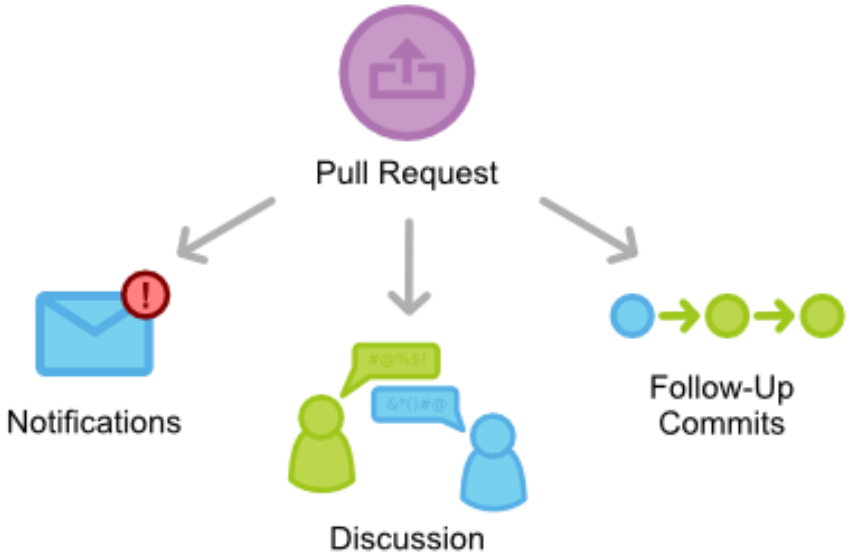
Comandos Avanzados.

Pull Request.

El comando *pull request* es una función comúnmente asociada con sistemas de control de versiones. Permite al desarrollador solicitar que los cambios que ha realizado en una rama de su repositorio sean integrados en otra rama, por lo general es hacia la rama principal (master o main).

Pasos de un Pull Request:

1. Creación de una nueva rama: Se crea una rama con los cambios propuestos.
2. Hacer cambios: Se realizan los cambios necesarios en la nueva rama.
3. Subir los cambios: Se suben los cambios al repositorio remoto.
4. Iniciar pull request: Se solicita fusionar los cambios a través de una solicitud en la plataforma de control de versiones.
5. Describir los cambios: Se proporciona una descripción detallada de los cambios propuestos.
6. Revisión y comentarios: Otros miembros del equipo revisan y comentan los cambios.
7. Aprobación: Se aprueba el pull request si los cambios son aceptados.
8. Integración: Los cambios se fusionan en la rama de destino.

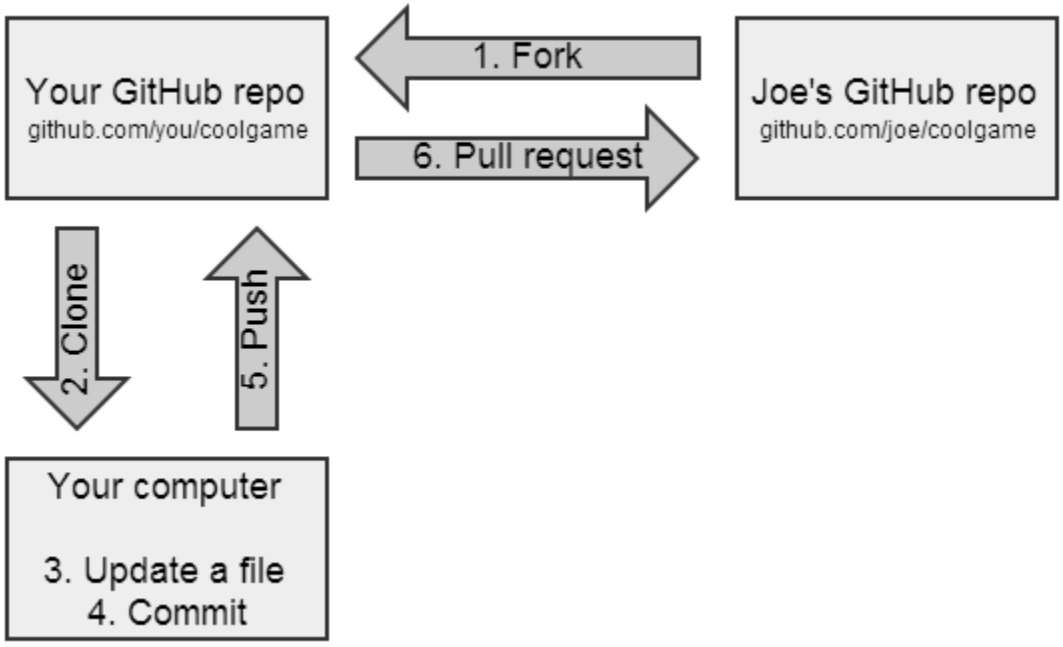


Fork.

Es una copia independiente de un repositorio de código en un sistema de control de versiones. Cuando se realiza un fork de un repositorio, se esta creando una copia del repositorio original en la cuenta personal, esto permite trabajar de manera independiente sin afectar el repositorio original.

Proceso de fork:

1. Crear un fork: Haces una copia independiente de un repositorio original en tu cuenta del sistema de control de versiones.
2. Trabajar con el fork: Realizar cambios en la copia del repositorio original, de la misma manera que se trabajaría con el repositorio original.
3. Mantener el fork actualizado: Si el repositorio original recibe actualizaciones, se puede sincronizar el fork realizado con el repositorio original para incluir esas actualizaciones en la copia.
4. Contribuir de vuelta: Si realizas cambios significativos en tu fork y quieres que estos sean agregados al repositorio original (de donde se realizó el fork), se puede enviar una solicitud de pull request al repositorio original para proponer los cambios realizados y contribuir al proyecto original.

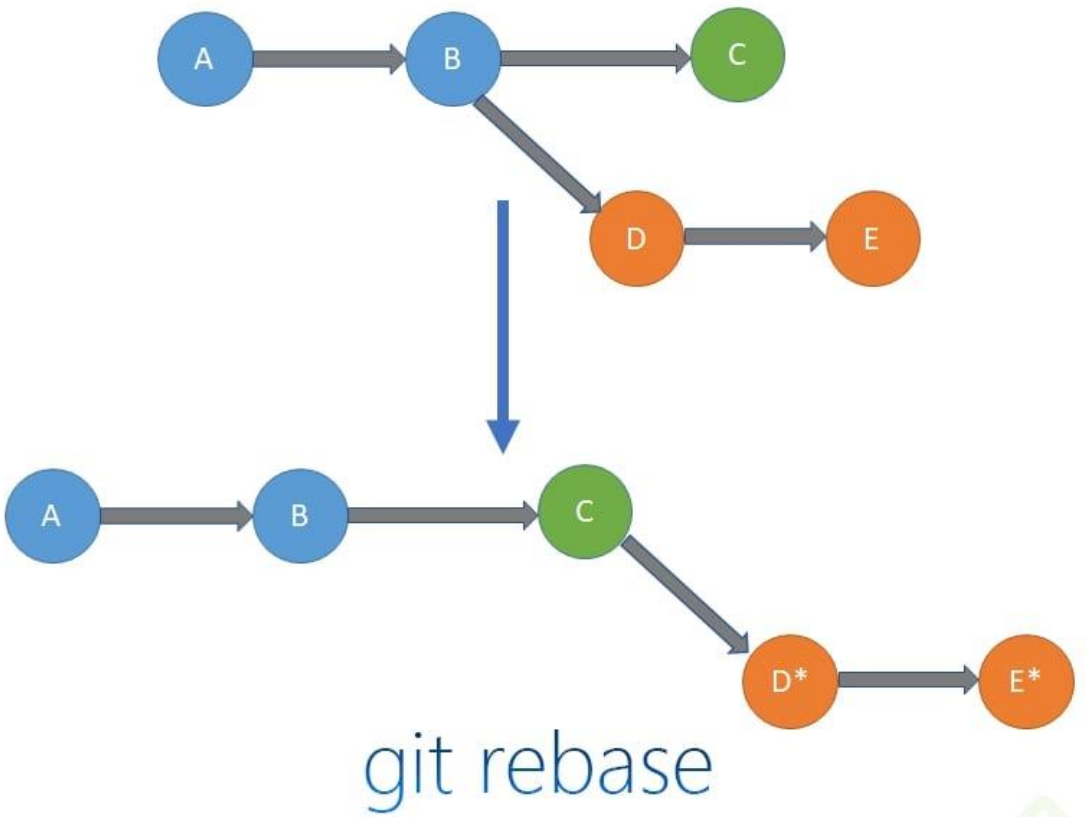


Rebase.

Reorganiza la historia de los commits en una rama. Básicamente, permite cambiar la base de una rama, reescribiendo la historia de los commits para que parezca que se basa en una rama diferente.

Proceso de rebase

1. Seleccionar una rama base: Escoger la rama en la que quieres basar el trabajo actual.
2. Iniciar el rebase: Utilizar el comando git rebase para comenzar el proceso.
3. Aplicar los cambios: Git deshace los commits de tu rama desde el punto de bifurcación y los aplica sobre la punta de la rama base.
4. Resolver conflictos: Si hay conflictos entre los cambios realizados y la rama base, se deben de resolver manualmente.
5. Finalizar el rebase: Una vez que se resuelvan los conflictos, se puede continuar con el rebase o abortarlo.
6. Actualizar la rama remota: Si se rebasa una rama compartida, se debe de actualizar la rama remota con git push –force.

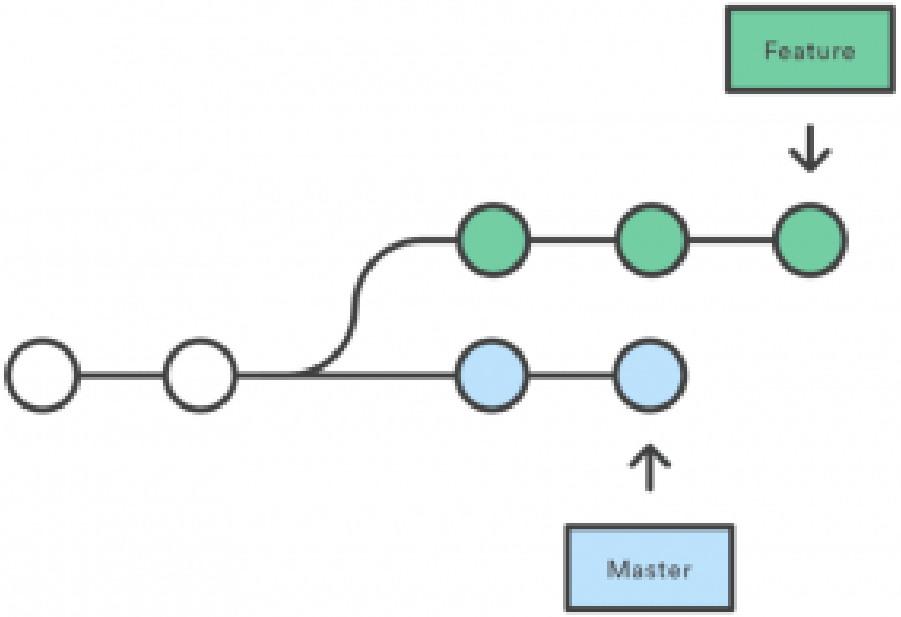


Stash.

Se utiliza para almacenar temporalmente cambios no comprometidos en un área de almacenamiento temporal llamada stash, para que se pueda cambiar de rama o realizar otras operaciones sin necesidad de comprometer esos cambios.

Función del comando stash:

1. Guardar cambios: Se utiliza git stash para almacenar temporalmente cambios no comprometidos
2. Continuar con el trabajo: Se puede seguir trabajando en otras tareas o cambiar de rama sin esos cambios en tu directorio de trabajo.
3. Aplicar cambios almacenados: Se utiliza el comando git stash apply o git stash pop para retomar los cambios guardados.
4. Resolver conflictos: Si hay conflictos al aplicar los cambios, se resolverían como si se tratara de una fusión.
5. Limpiar el stash: Cuando ya no necesites los cambios guardados, se utiliza git stash drop, para eliminar los cambios en stash.

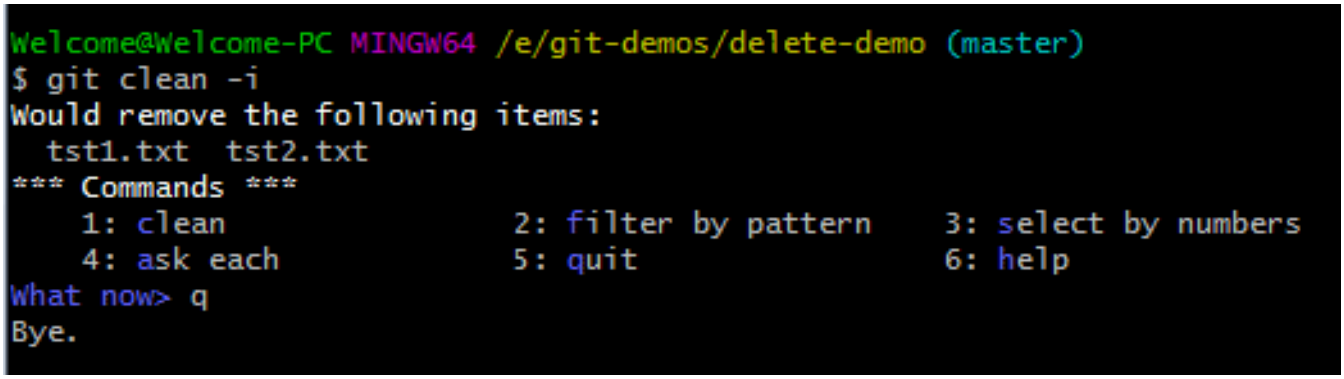


Clean.

Se utiliza para eliminar archivos que no están siendo seguidos por el sistema de control de versiones. Estos archivos no se encuentran en el índice de Git (staged) ni están siendo rastreados por Git en el directorio de trabajo.

Función del comando clean:

1. Verificar archivos no rastreados: Utilizando el comando git status para revisar que archivos no están siendo rastreados por Git en el directorio de trabajo.
2. Eliminar los archivos no rastreados: Se emplea el comando git clean para eliminar estos archivos no rastreados. Se usa -f o –force para confirmar la eliminación.
3. Opciones adicionales: Se puede personalizar la eliminación de archivos no rastreados con opciones como -n para previsualizar la acción o -d para eliminar también directorios no rastreados.



Cherry-Pick.

Se utiliza para aplicar cambios específicos de un commit a otra rama. Esto significa que puedes seleccionar un solo commit de una rama y aplicarlo en otra rama sin necesidad de fusionar toda la rama.

Funcionalidad del comando Cherry-pick:

1. Identificar el commit: Encontrar el commit especifico que se desea aplicar en otra rama.
2. Cambiar a la rama destino: Se debe dirigir a la rama que se desea aplicar el commit.
3. Ejecutar el comando cherry-pick: Utilizar git cherry-pick <commit> para aplicar el commit seleccionado en la rama actual.
4. Resolver conflictos (si los hay): Si surgen conflictos con los cambios existentes, se resuelven manualmente.
5. Confirmar los cambios: Después de resolver los conflictos, confirma los cambios resultantes con git commit.