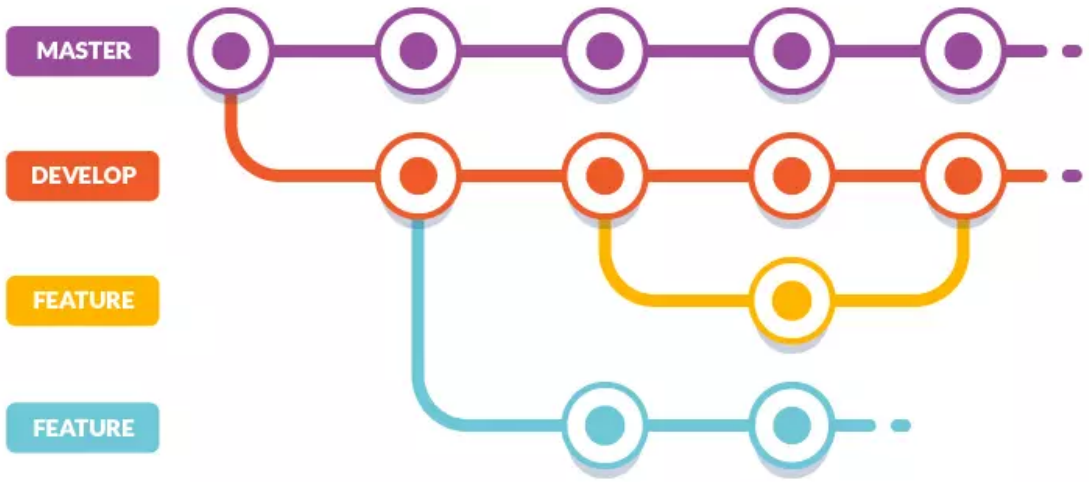
Examen Semana 4 - Fabiola Gómez Montiel

1. Explica los siguientes conceptos:

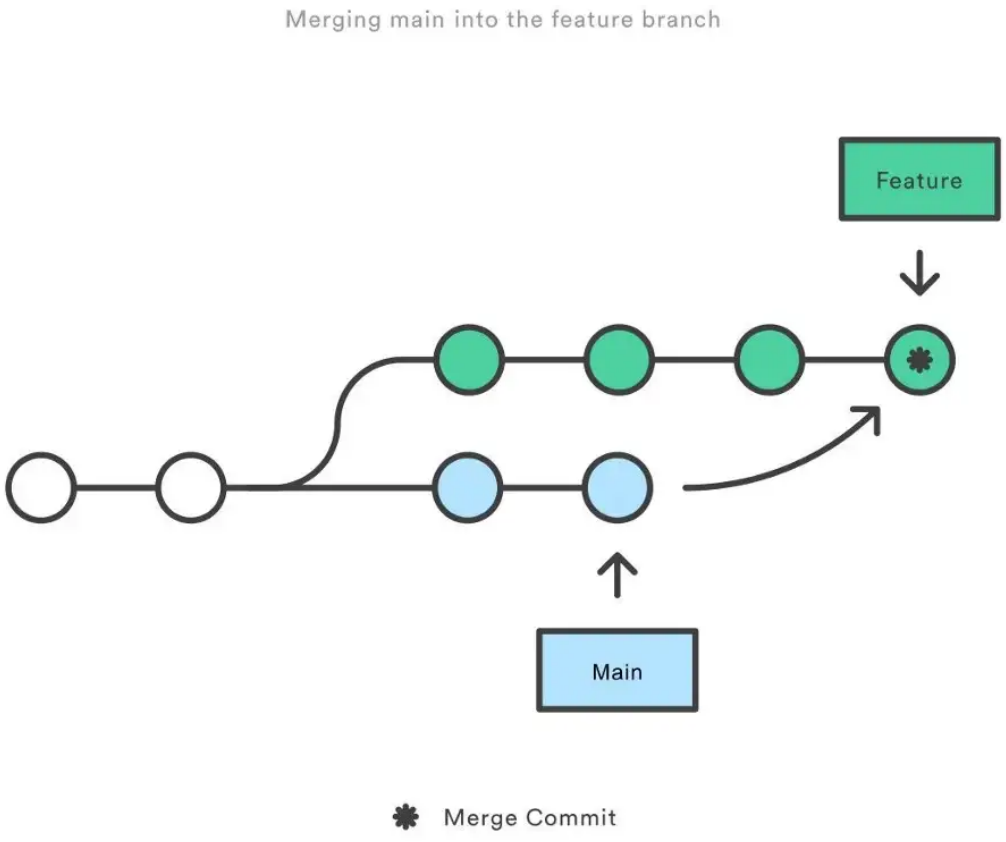
🟣 **Branch**

Las **Branches** son ramificaciones que parten de una rama principal llamada Main o Master y que nos sirven para poder encapsular y realizar cambios independientes sin afectar otras partes del código que se encuentran en la rama principal. Una vez que los cambios de una rama se terminan y se aprueban, estos se unen a la rama principal, o bien, pueden borrarse si es necesario.



🟣 **Merge**

Hacer **Merge** en nuestro proyecto simboliza que el trabajo en alguna de las branches está terminado y listo para formar parte del proyecto principal.



🟣 **Conflicts**

Los **Conflicts** aparecen generalmente cuando en más de una rama se han hecho modificaciones a los mismos archivos o líneas y de código y por lo tanto Git no puede determinar automáticamente qué cambios deberían mantenerse. Es necesario elegir manualmente qué datos deben conservarse para poder detener el conflicto.

lib/compose.js

View file @e71062d

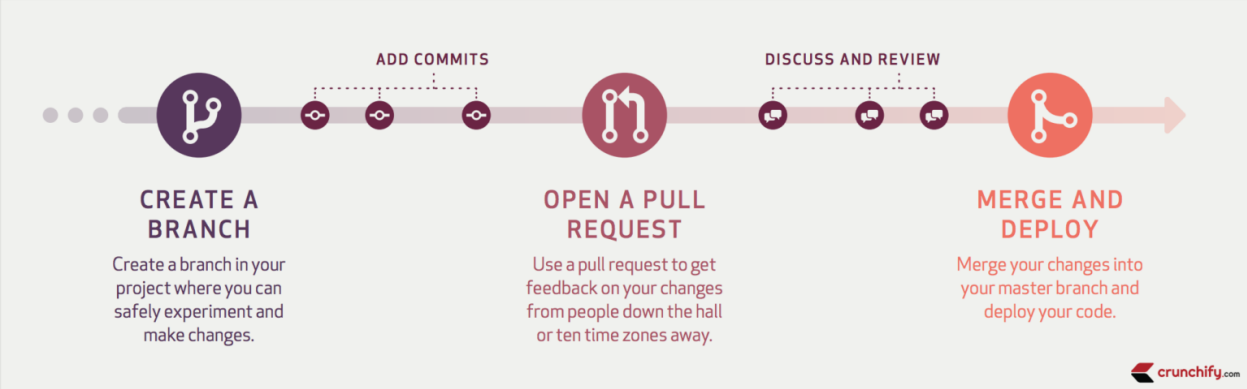
```
@@ -12,19 +12,27 @@ function setWritability(obj, writable) {
  });
}

origin//their changes  Use this
function mixin(base, mixins) {
  base.mixedIn = base.hasOwnProperty('mixedIn') ? base.mixedIn :
[];

  for (var i = 0; i < mixins.length; i++) {
    if (base.mixedIn.indexOf(mixins[i]) == -1) {
module.exports = {
  mixin: mixin
};
```

```
@@ -12,19 +12,27 @@ function setWritability(obj, writable) {
  });
}

HEAD//our changes  Use this
    function mixin(base, mixins) {
      base.mixedIn = Object.prototype.hasOwnProperty.call(base, 'mi
xedIn') ? base.mixedIn : [];

      for (var i = 0; i < mixins.length; i++) {
        if (base.mixedIn.indexOf(mixins[i]) == -1) {
          setWritability(base, false);
          mixins[i].call(base);
          base.mixedIn.push(mixins[i]);
        }
      }

  for (var i = 0; i < mixins.length; i++) {
    if (base.mixedIn.indexOf(mixins[i]) == -1) {
module.exports = {
  mixin: mixin
};
```

Pull Request

Un **Pull Request** (también conocido como **Merge Request**) se da cuando un contribuidor solicita que los cambios que ha realizado en una rama se unan al código de la rama principal.

Cuando esto sucede, el encargado del repositorio deberá revisar el código nuevo y determinar si los cambios que se han hecho pueden ser unidos o si aún necesitan tests o cambios para no afectar al producto final.

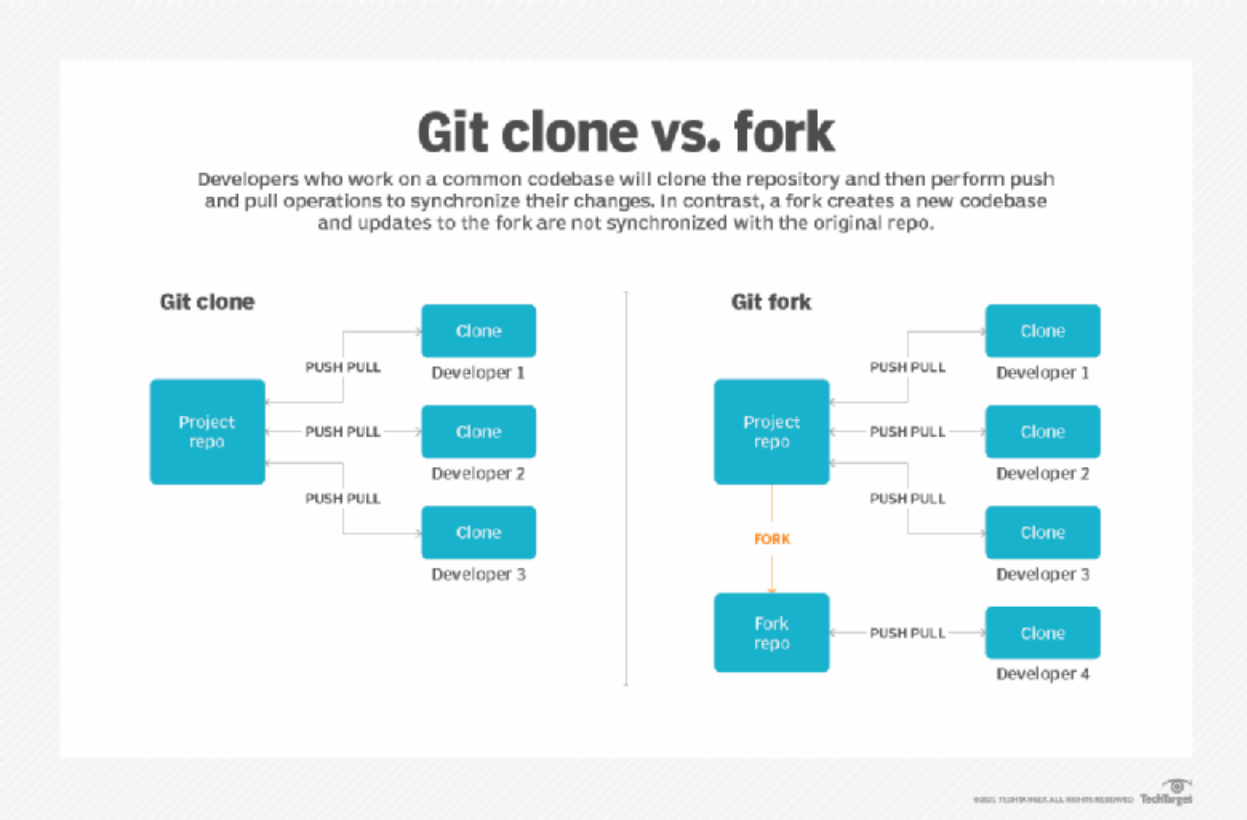


Fork

Un **Fork** es la copia de un repositorio que nos permite hacer cambios en un repositorio nuevo sin afectar de ninguna forma la información que contiene el original.

En proyectos de código abierto, hacer un fork es usado para proponer ideas que puedan implementarse al repositorio

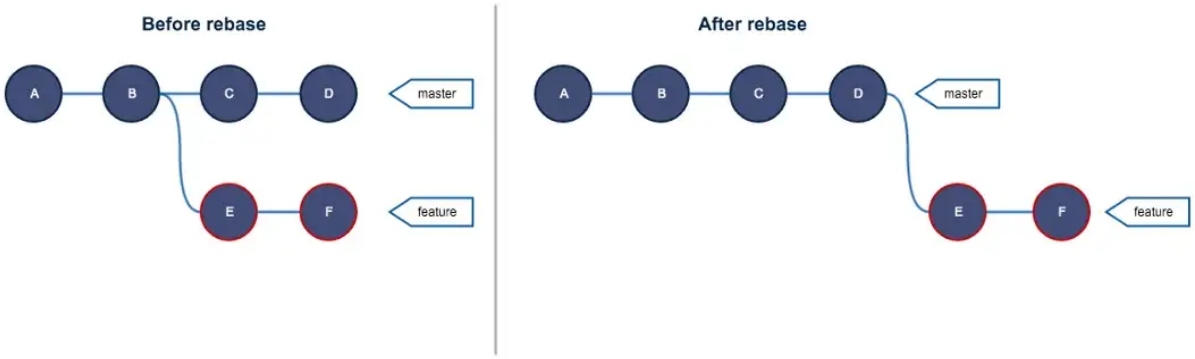
original a través de un pull request que compara el trabajo de ambos repositorios.



🟣 Rebase

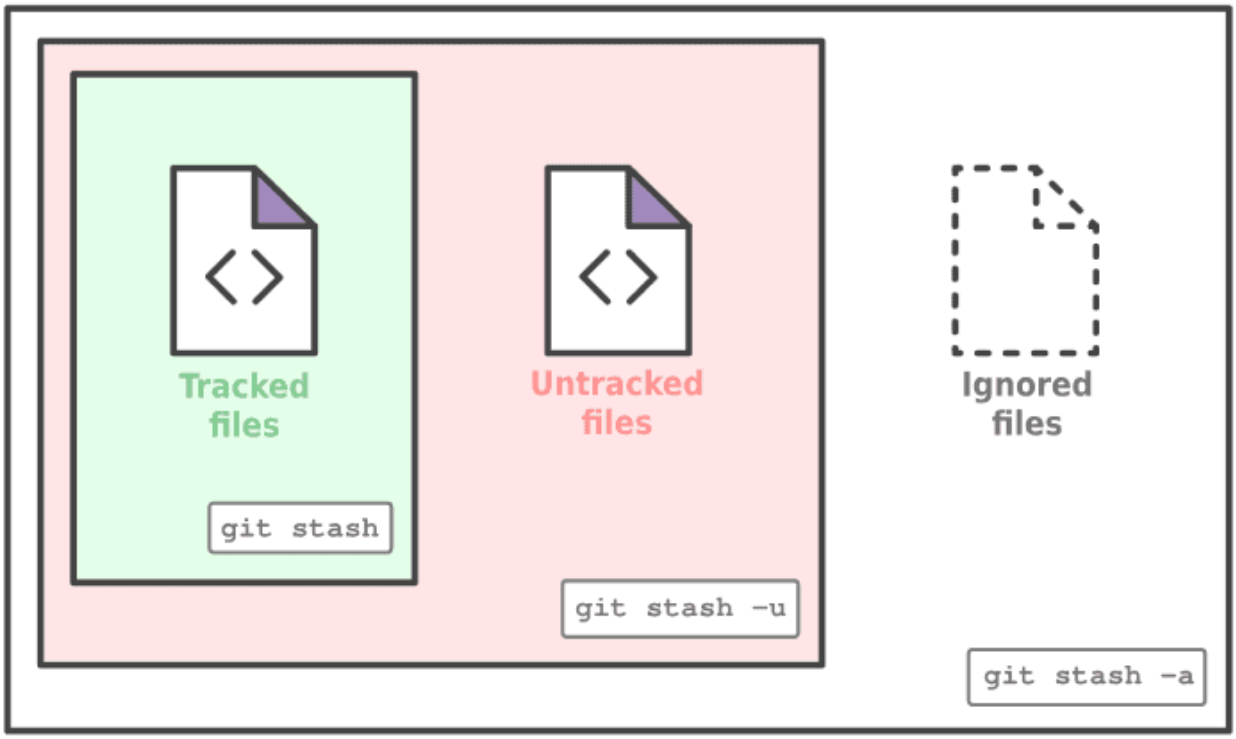
Rebase es un comando que nos ayuda a integrar los cambios mientras que cambiamos el inicio de una rama de un commit a otro.

Esto puede resultar útil cuando se han realizado modificaciones a la rama principal mientras se trabajaba en una rama secundaria y se quieren implementar esos cambios a la rama secundaria, aunque se recomienda usar esta funcionalidad solo en repositorios locales.



🟣 Stash

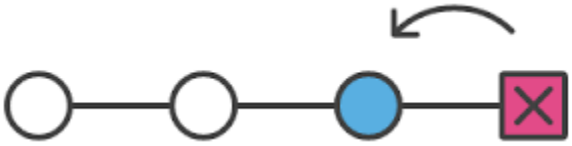
Git tiene un área llamada **Stash** donde se almacenan temporalmente los cambios que no se han enviado al repositorio y es posible “ocultarlos” para trabajar temporalmente en otra cosa. Es posible guardar o descargar los cambios ocultos.



🟣 Clean

Clean es un comando que sirve para eliminar los archivos que se han creado en el directorio de trabajo pero que aún no se

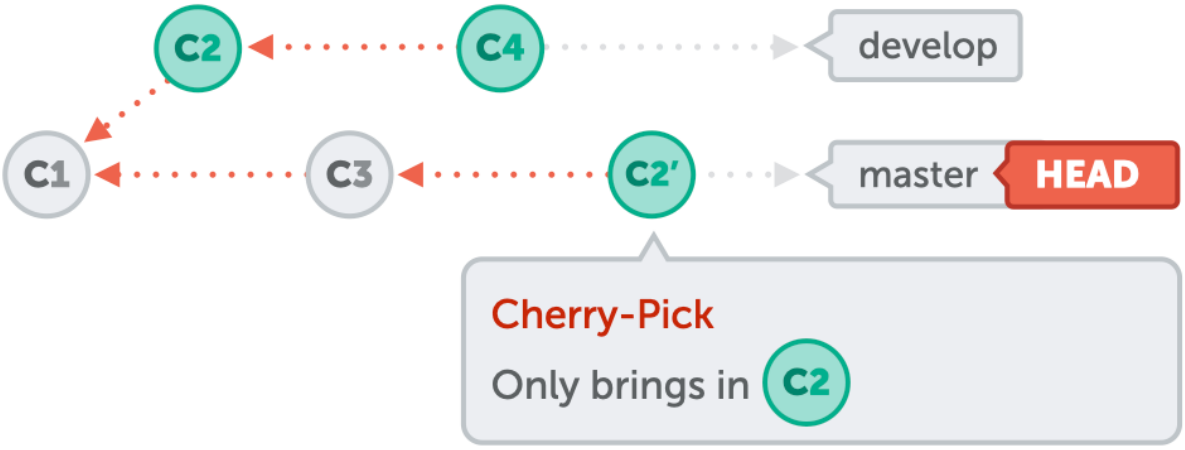
han añadido usando el comando de git add.



🟣 **Cherry Pick**

Cherry-pick nos permite tomar commits específicos de una rama y llevarlos a la rama principal sin tener que hacer un merge completo.

Es muy útil para deshacer cambios (por ejemplo, si un commit se hizo equivocadamente en una rama en la que no debía hacerse), sin embargo hay que usarlo con moderación ya que al hacer un merge con todas las ramas pueden aparecer conflictos debidos al cherry-pick.



2. Explica para qué sirven los verbos HTTP y sus buenas prácticas:

🟩 **POST**

Este tipo de solicitudes se encarga de mandar los datos nuevos al servidor.

🟩 **GET**

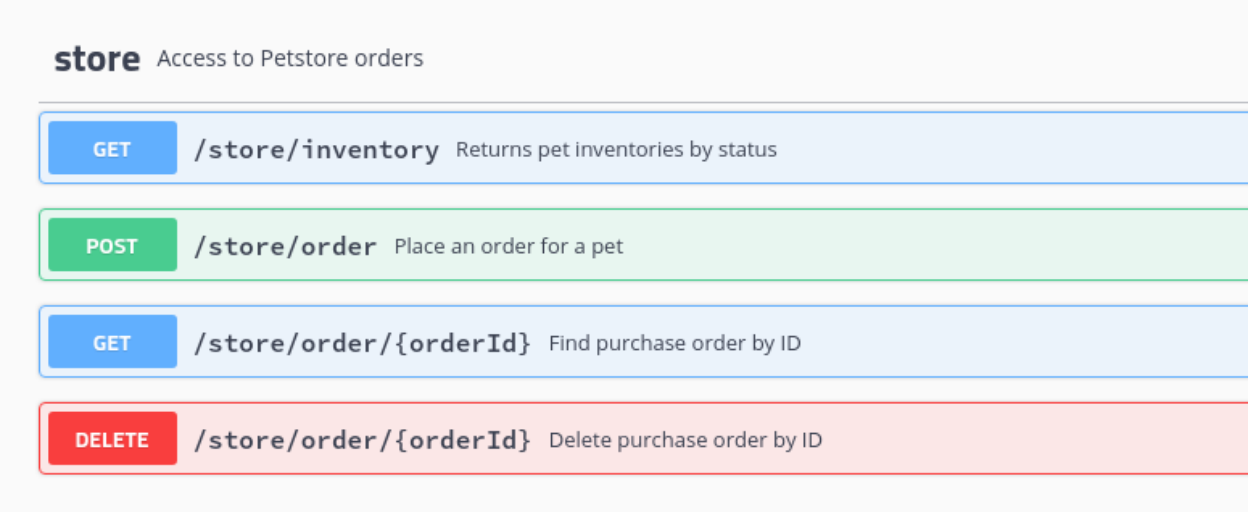
Se utiliza para obtener la información de los datos guardados sin modificarla de ninguna manera.

🟩 **PUT**

Sirve para enviar datos al servidor, esta vez para actualizar un recurso ya existente, lo que es útil cuando ha habido cambios o errores al realizar los registros.

🟩 **DELETE**

Sirve para borrar un registro específico.



💙 La mejor forma de generar la URL de la API es seguir la relación de pertenencia, donde una clase B pertenece a una clase A, ya que es la forma más clara de mostrar una jerarquía o ruta en la estructura para que sea fácil de navegar, incluso por personas con poca experiencia.

En el ejemplo de la imagen, Inventario y Orden pertenecen a Tienda, mientras que Orden cuenta a su vez con un ID.