Introducción a GIT

Icia Carro Barallobre

¿Qué es GIT?

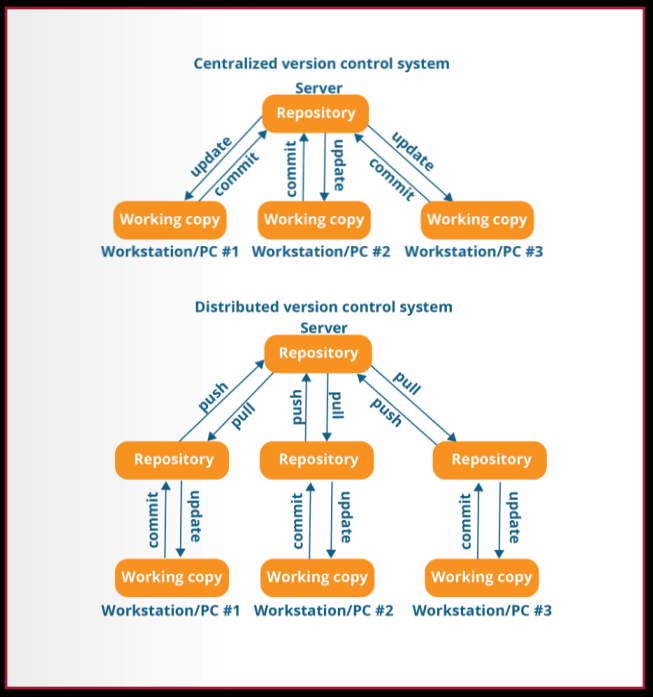
Es una herramienta de Control de Versiones que nos permite mantener un histórico de los cambios realizados en nuestro código.

Instalación & Configuración

Instalación: Enlace

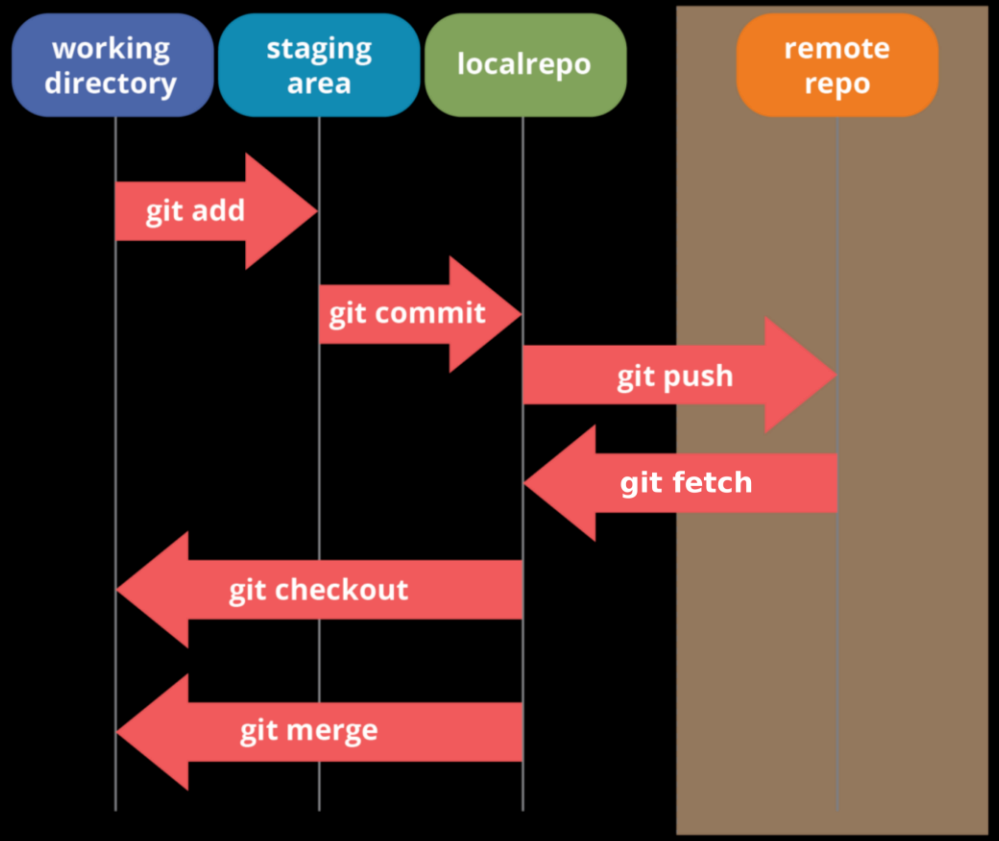
Configuración global - archivo .gitconfig:

```
git config --global user.name "nombre_aqui" # Nombre - no  unico
git config --global user.email "correo_aqui" # Email - es unico
```



Zonas de trabajo

- **Working dir/Área de trabajo**: Zona con el estado actual de los ficheros.
- **Staging Area/Index/Área de ensayo**: Zona intermedia donde vamos almacenando los cambios que van a conformar un commit.
- **El repositorio local**: Zona donde se almacenan los commits. Toda su información se guarda en el *directorio .git*.
- **Los repositorios remotos**: Puede ser uno solo (por defecto se llama origin), o podríamos tener varios. La comunicación entre el repositorio local y los remotos se realiza mediante los comandos *push* y *pull*.
- ***Stash**: Es la zona de guardado rápido, una zona auxiliar para guardar los cambios en los que estamos trabajando cuando por algún motivo nos interrumpen y tenemos que cambiar de rama, pero aún no queremos hacer un commit.



Comandos básicos

Comandos básicos en local:

`> git status` Muestra la rama actual + estado de los archivo

`> git add -A | "archivo"` - Rastrear todos o un archivos (añadido o actualizado)

`> git commit -m "mensaje"` - Pasar al área de staging los archivos rastreados

`> git commit -am "mensaje"` - Rastrear (excepto archivos nuevos) y pasar al área de staging a la vez.

`> git log` - Históricos de commits en la rama actual

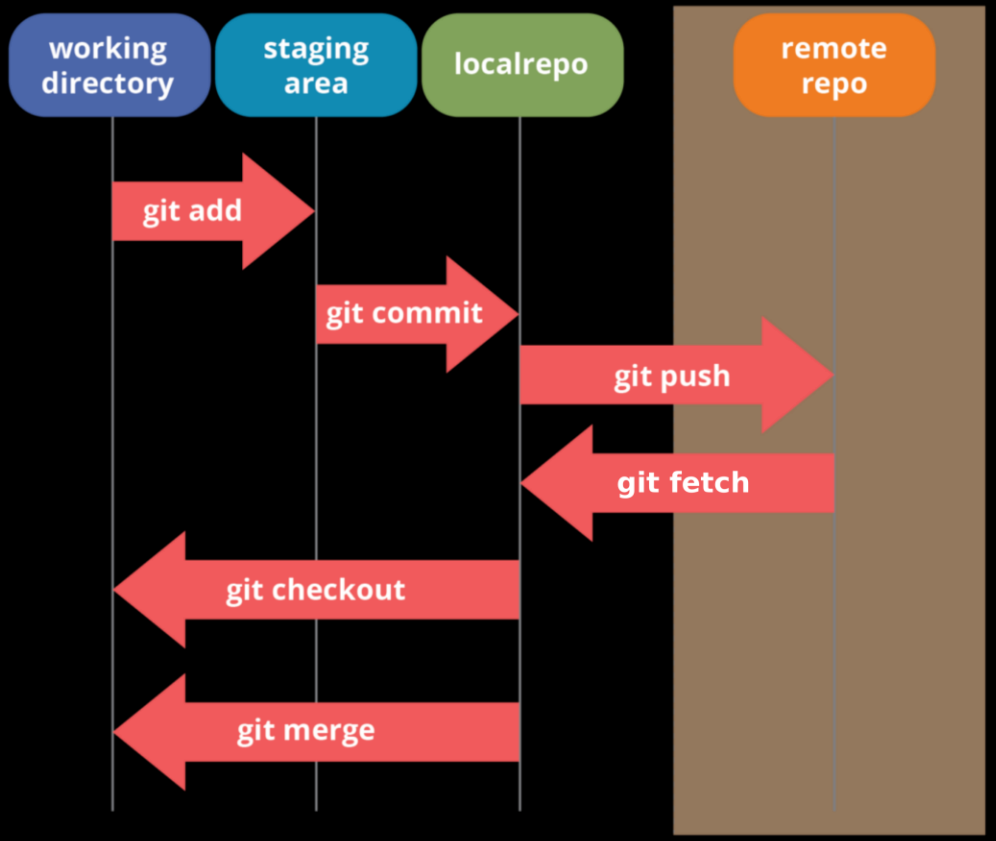
`> git diff` - Muestra las differencias con la version rastreada anterior del archivo

Comandos básicos en remoto

`> git push` - Empujar los cambios a remoto

`> git pull` - Traerse los cambios de remoto

`> git clone` - Clonar a local un repo ya ecistente en remoto



Elementos de git

- Stash
- Commit
- Branch
- Tag
- Revert

Stash

El área de Stash es un espacio en paralelo en el
que podemos guardar los cambios que tengamos sin commitear (es decir, que no hayan llegado a local repo).

Comandos útiles:

```
git stash (push -m)  # Stashear cambios actuales (excepto archivos untracked)
git stash - u        # Stashear  cambios actuales (incluyendo archivos untracked)
git stash -p         # Selección interactiva de las partes que quieres stashear

git stash list       # Listar todos los stashes (incluyendo nombre y rama)

git stash apply (name_or_commit) # Aplicar un stash (por defecto el 0)
git stash pop  (name)            # Aplicar un stash (por defecto el 0) y removerlo
git stash drop (name)            # Borrar un stash
```

Commits

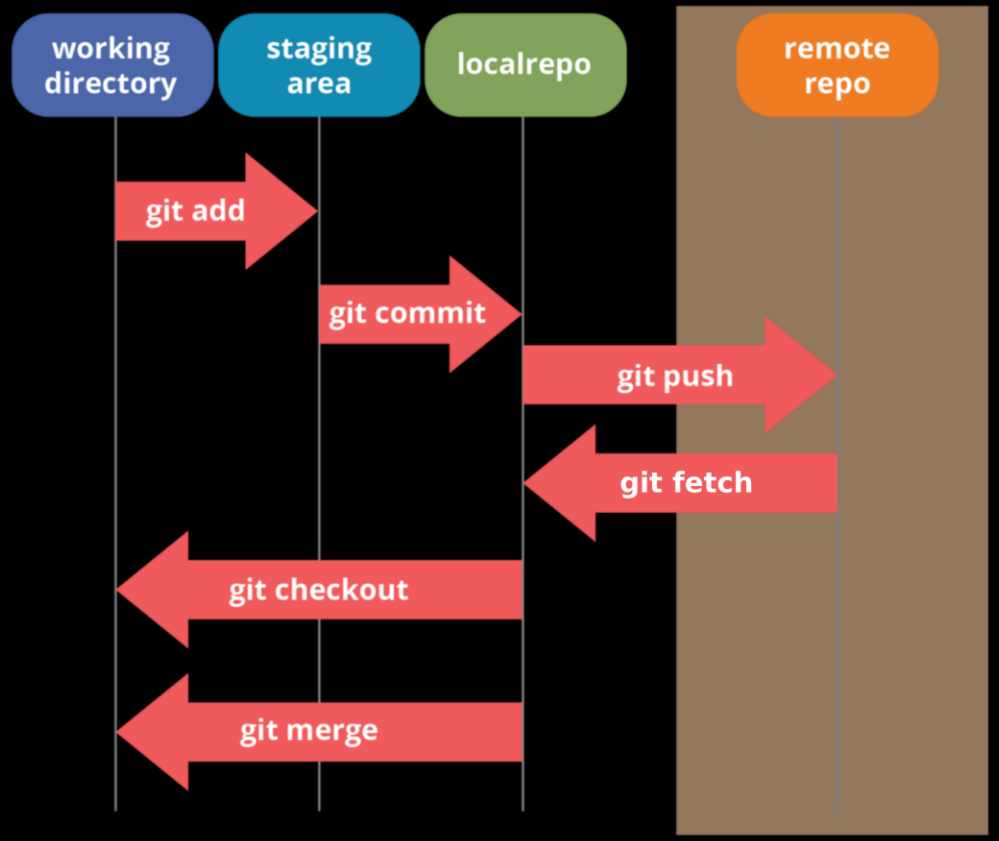
Los commits son únicos y muchas de las operaciones de git trabajando con ellos. Tiene una clave sha que los identifica.

El comando git commit captura una instantánea de los cambios rastreados en ese momento del proyecto.

Podemos mezclar commits con el comando:

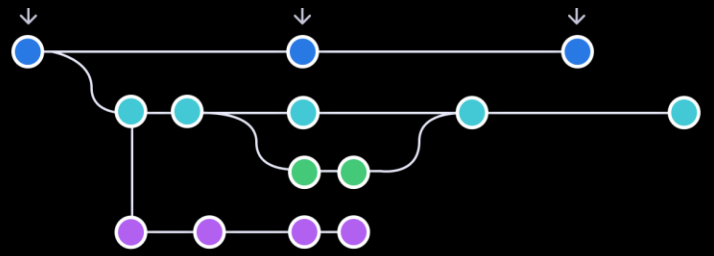
```
git commit –amend –m "..."
```

Podemos omitir el –m si queremos el mismo mensaje



Branch

Las ramas son espacios de trabajo paralelos donde trabajar en diferentes ideas, que después se pueden juntar o bifurcar. Comandos útiles:



```
git branch  # Lista las ramas locales, la actual la marca con asterisco
git branch -a # Lista todas las ramas, locales y remotas
git branch name # Crea una rama, en el commit actual
git branch name commit_hash # Crea una rama en el commit escogido
git branch -m old_name new_name #Renombra una rama
git branch -d branch_name # Borra una rama

## Moverse entre ramas
git checkout branch-name
git checkout –b  branch-name # Crearla y moverse a la vez
```

Tag

Referencia 'fija' del estado de nuestro código (commit) que podemos identificar fácilmente mediante un alias. Se suele usar para definir versiones de una aplicación.

Comandos útiles:

```
git tag                    # Lista todos los tags
git tag name               # Crear un tag con el commit actual
git tag name commit        # Crear un tag de un commit en concreto
git tag name -m msg        # Crear un tag del commit actual con un mensaje
git tag -d name            #  Borrar un tag
git fetch --tags           # Obtener tags actualizados
git tag --contains commit  # Lista tags cuyos antecesores tengan ese commit
```

Revert

A veces puede darse el caso en que hayamos hecho commit de nuestros cambios pero no queremos que suban esos cambios al hacer push.

Lo que nos permite este comando es generar un nuevo commit que es el inverso al que le hayamos indicado, con lo que habremos revertido sus cambios sin tocar el histórico previo.

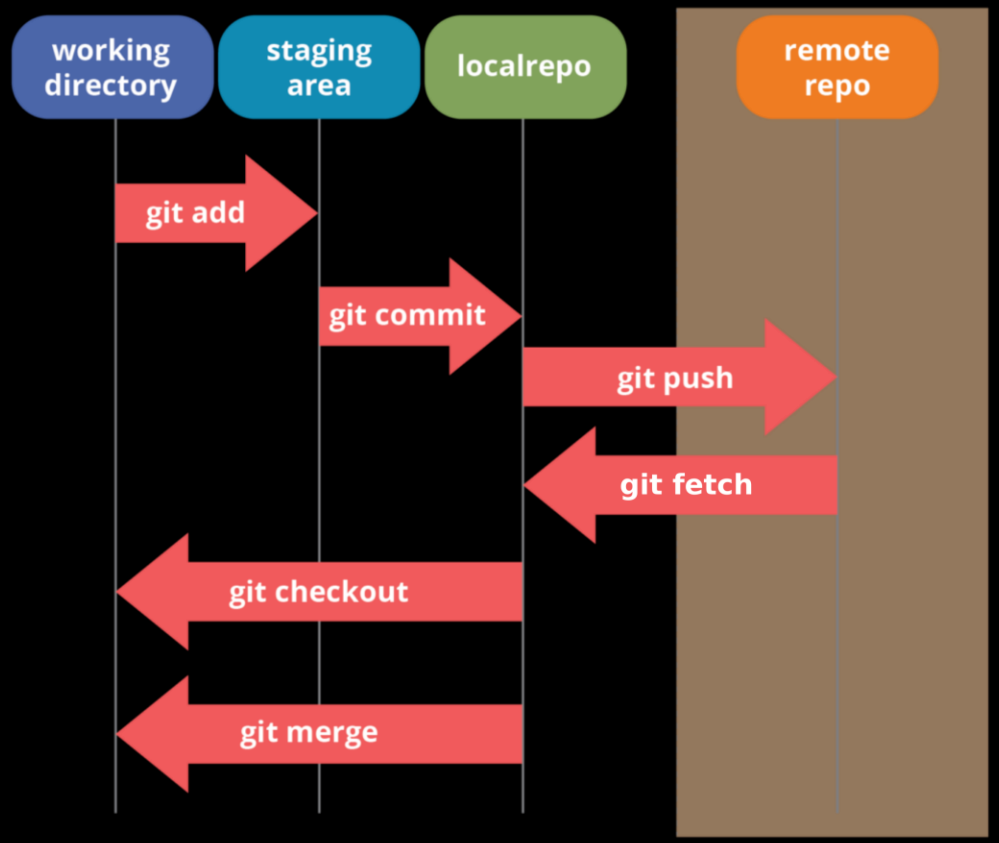
```
git revert id_commit   # Generar un commit inverso
                       #  al que le hayamos indicado
```

Trabajo en remoto

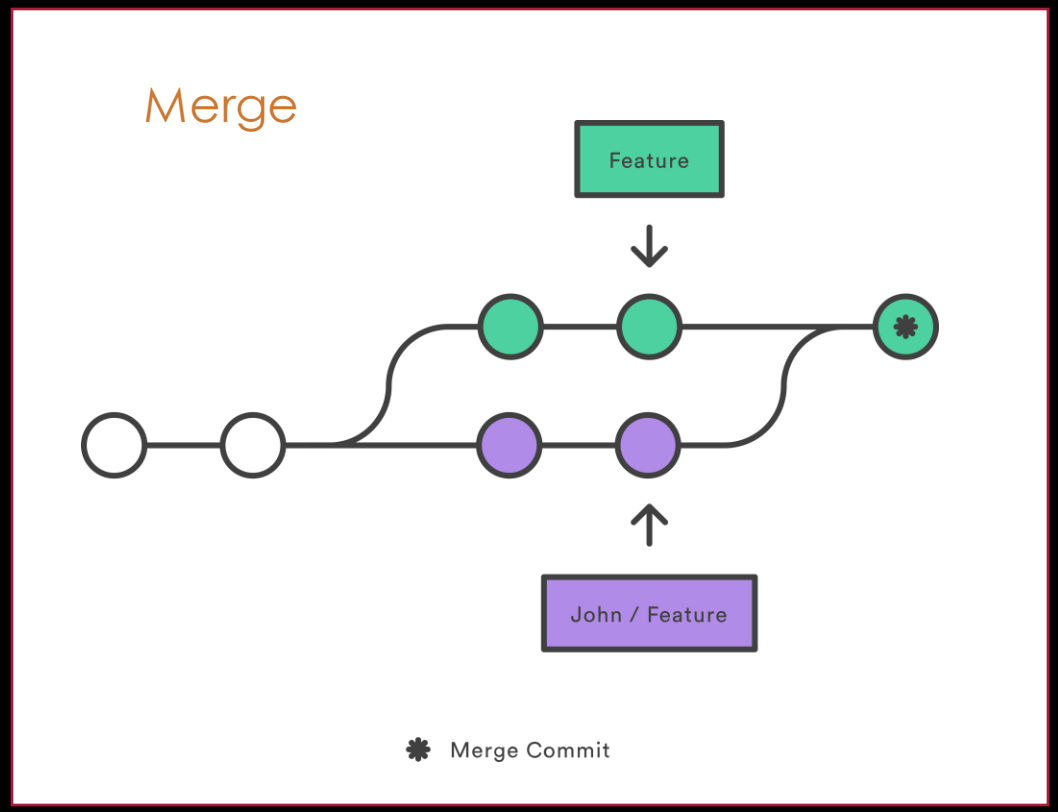
Descargarnos ramas / commits: `git fetch origin`.

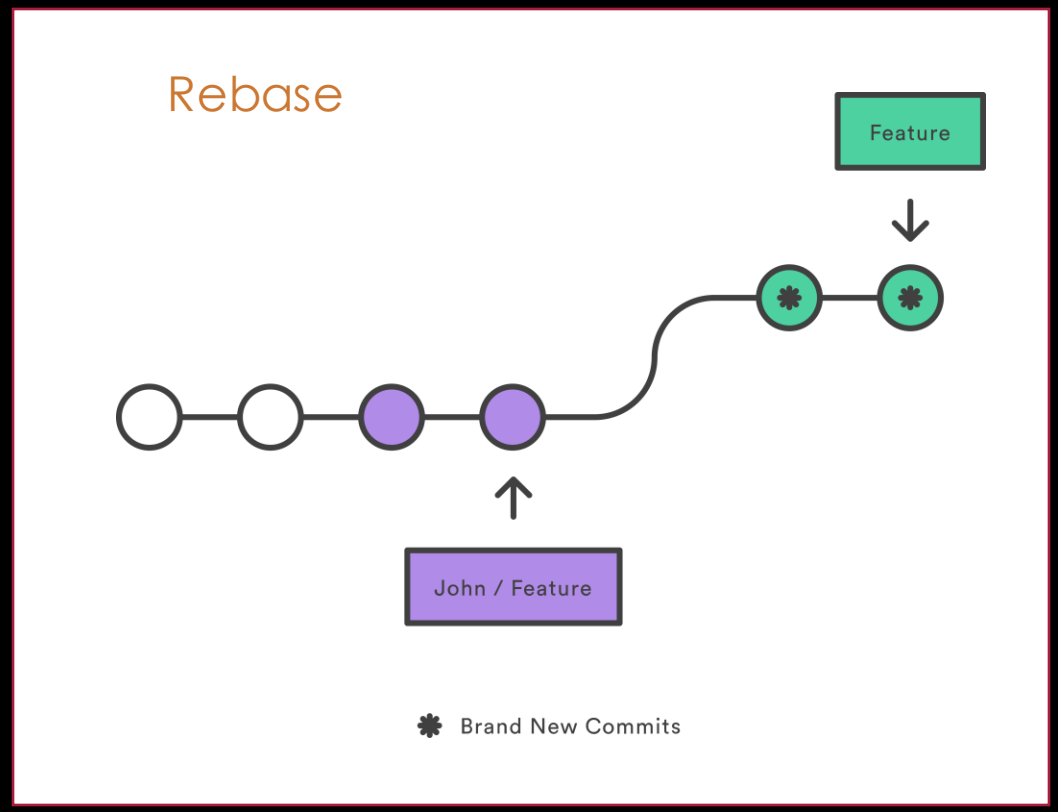
Descargarnos cambios: `git pull`, que es un alias de un fetch y merge.

Limpiar todas las referencias a ramas que se hayan muerto: `git fetch -ap`



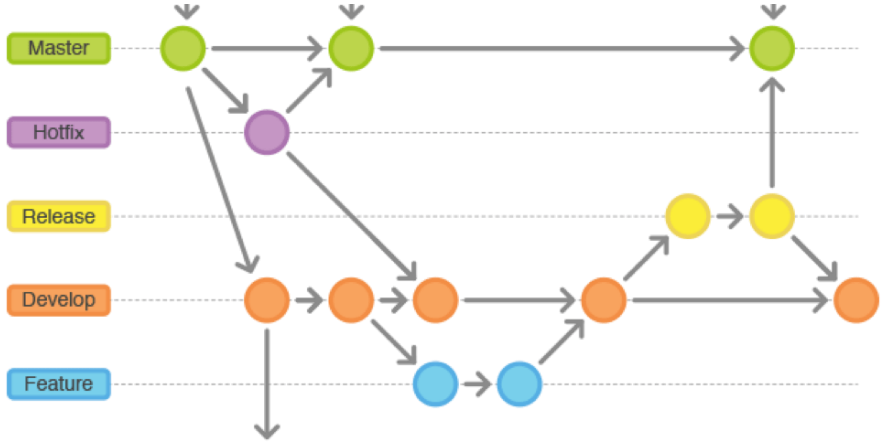
Resolviendo conflictos



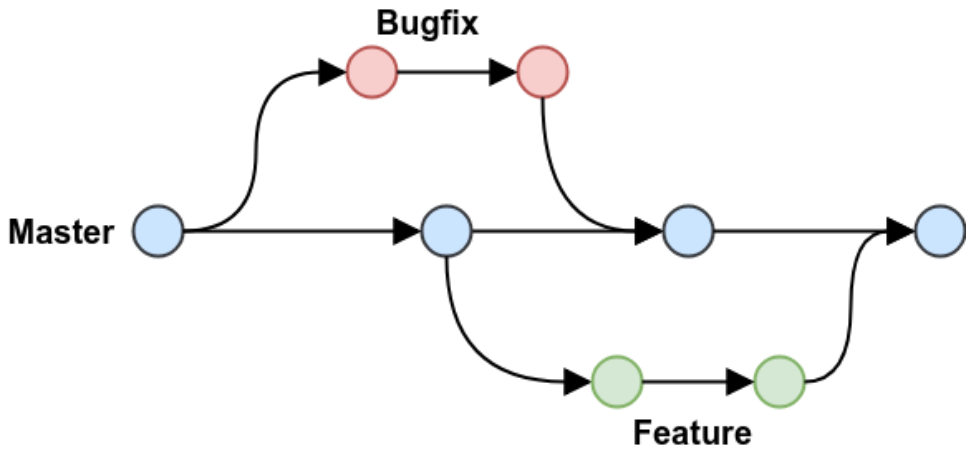


Trabajo en equipo

GitFlow



GithubFlow



Buenas prácticas

Archivos que no queremos rastrear

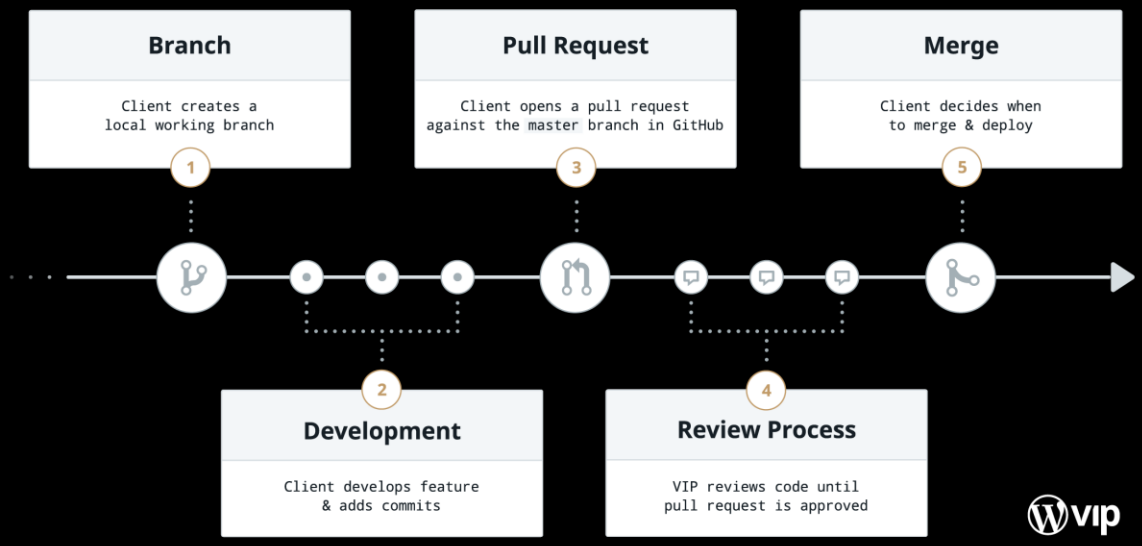
- Para ignorar archivos en un proyecto: `.gitignore`
- Para ignorar localmente (archivos propios – IDE, …):

```
git config –global core.excludesfile ~/.gitignore_global
```

How to Write a Git Commit Message

- Título y Contenido: Git entiende los saltos de línea para separarlos
- Título (< 50 caracteres), cuerpo (<72)
- Primera letra en mayúsculas
- Usar el imperativo
- Usar el cuerpo para poner el porqué no el como

GitHub: Pull Request



Template

Titulo

Porqué

Checklist

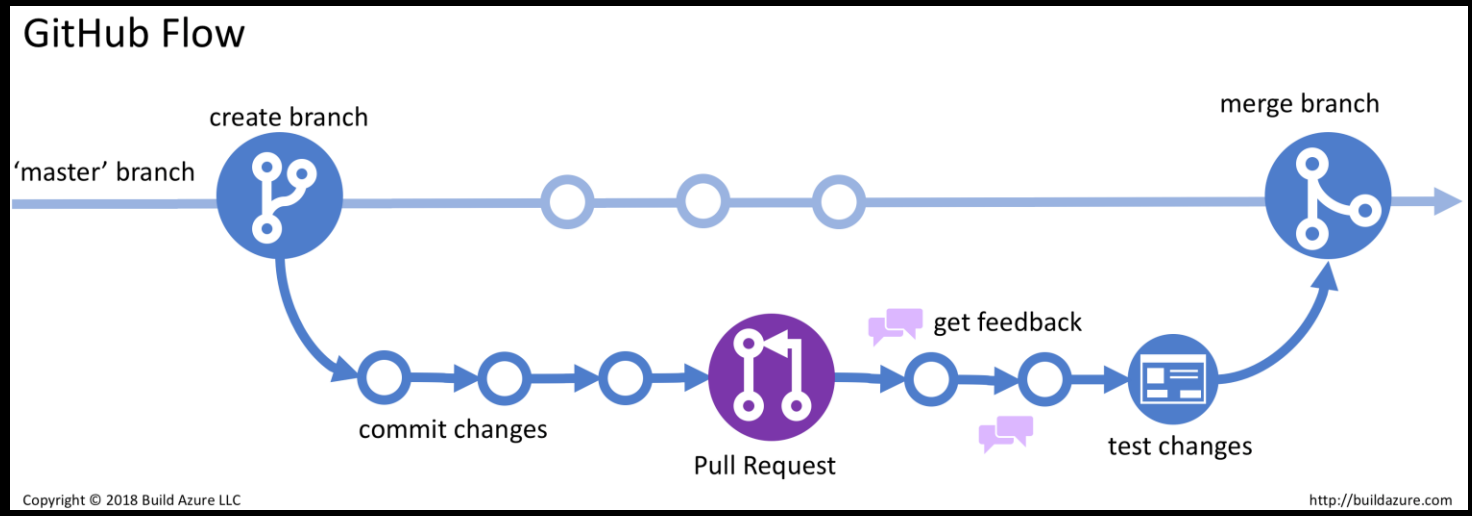
- Actualizar docu
- Actualizar los test

Otros

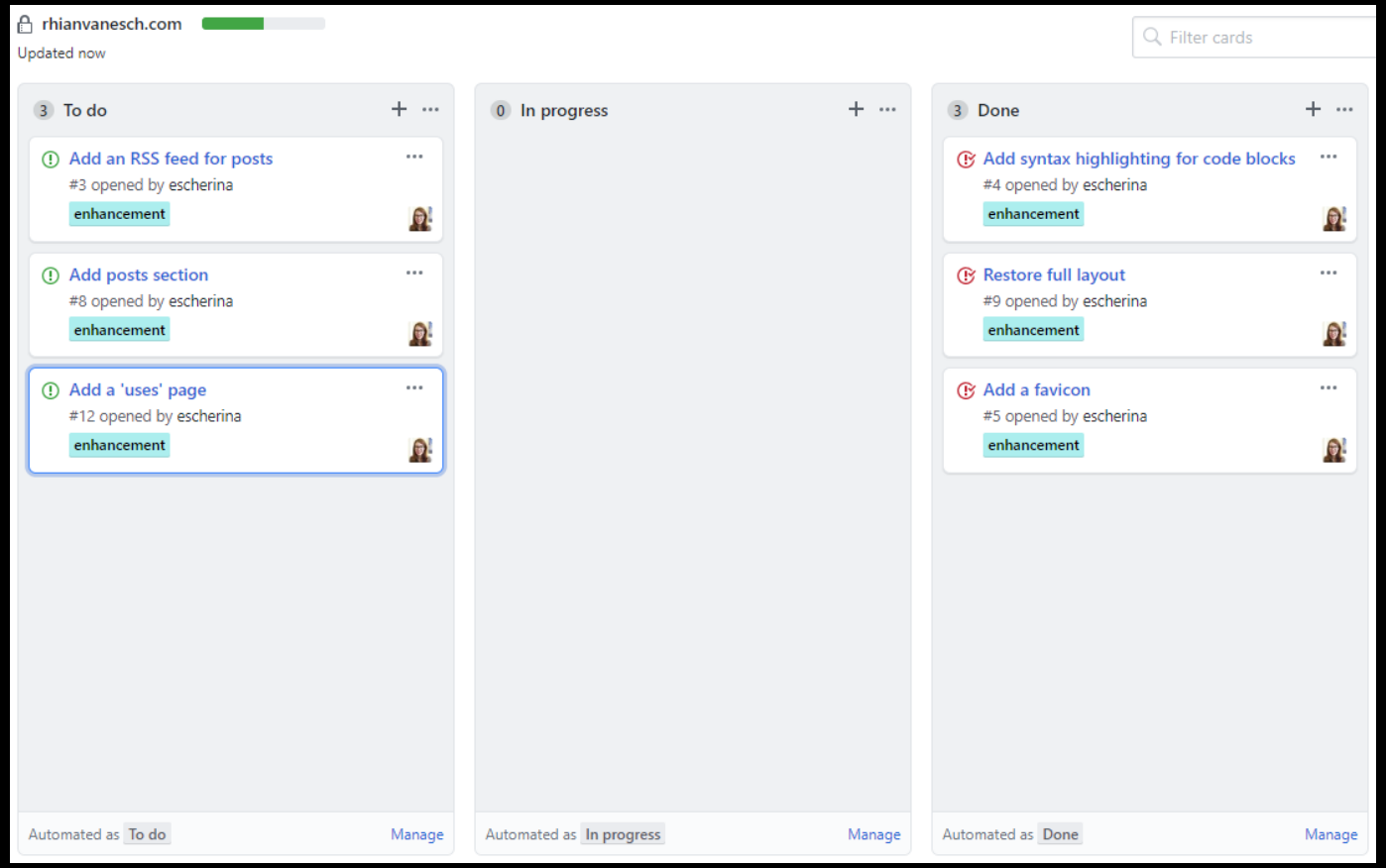
...

GitHub: Actions

Es una plataforma para automatizar el desarrollo de workflows. Uno muy común suele ser ver si el código de la PR pasa los tests. Otro puede ser desplegar cuando haya cambios en la rama main.



GitHub: Projects



Ayudas (I)

Revertir cambios commiteados

```
git revert @  // El mas reciente
```

```
git revert HEAD~4  // Los 5 ultimos commits
```

```
git revert master~2..master~5  // Multiples commits
```

Esta estrategia resulta especialmente útil si ese commit que vamos a revertir ya estuviera subido al repositorio remoto

Añadir cambios al último commit

```
git commit --amend
```

Ayudas (II)

Al ejecutar un git reset reseteáremos los cambios aplicados, pero no añadiendo un nuevo commit, sino modificando el histórico al eliminar el commit o commits de la rama.

Eliminar commits

```
git reset HEAD~1 --hard  // Restea todo, head, workdirectory y indice, perdemos los cambios.
```

Fusionar commits

```
git reset --mixed HEAD~6   // Mixed (por defecto) restablece HEAD e indice pero mantenemos workdirectory (stage)
git reset --soft HEAD~6    // Restablece el HEAD , mantiene indice y workdirectory (modificados pero no stage)
git commit -m 'mensaje de commit'
```