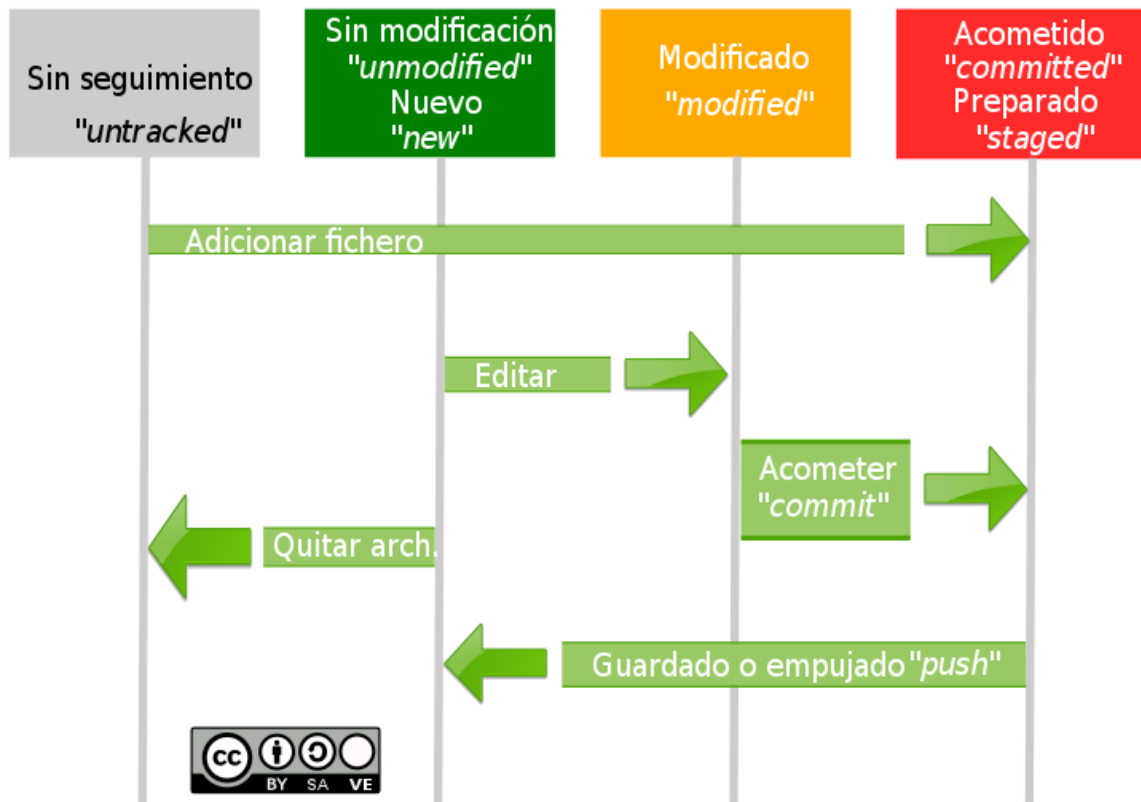


# Ciclo de vida de los archivos mediante Git



## Los comandos básicos

Como cualquier tipo de lenguaje usa códigos básicos y complicados estos son los códigos que se utiliza para las personas que se están iniciando en git es:

### git help

Muestra una lista con los comandos más utilizados en GIT.

### git init

Podemos ejecutar ese comando para crear localmente un repositorio con GIT y así utilizar todo el funcionamiento que GIT ofrece. Basta con estar ubicados dentro de la carpeta donde tenemos nuestro proyecto y ejecutar el comando. Cuando agreguemos archivos y un commit, se va a crear el branch master por defecto.

- `git fetch`:

Descarga los cambios realizados en el repositorio remoto.

- `git merge <nombre_rama>`:

Impacta en la rama en la que te encuentras parado, los cambios realizados en la rama “nombre\_rama”.

- `git pull`:

Unifica los comandos *fetch* y *merge* en un único comando.

- `git commit -am "<mensaje>"`:

Confirma los cambios realizados. El “mensaje” generalmente se usa para asociar al *commit* una breve descripción de los cambios realizados.

- `git push origin <nombre_rama>`:

Sube la rama “nombre\_rama” al servidor remoto.

- `git status`:

Muestra el estado actual de la rama, como los cambios que hay sin *commit*ear.

- `git add <nombre_archivo>`:

Comienza a trackear el archivo “nombre\_archivo”.

- `git checkout -b <nombre_rama_nueva>`:

Crea una rama a partir de la que te encuentres parado con el nombre “nombre\_rama\_nueva”, y luego salta sobre la rama nueva, por lo que quedas parado en esta última.

- `git checkout -t origin/<nombre_rama>`:

Si existe una rama remota de nombre “nombre\_rama”, al ejecutar este comando se crea una rama local con el nombre “nombre\_rama” para hacer un seguimiento de la rama remota con el mismo nombre.

- `git branch`:

Lista todas las ramas locales.

- `git branch -a`:

Lista todas las ramas locales y remotas.

- `git branch -d <nombre_rama>`:

Elimina la rama local con el nombre “nombre\_rama”.

- `git push origin <nombre_rama>`:

Commitea los cambios desde el branch local origin al branch “nombre\_rama”.

- `git remote prune origin`:

Actualiza tu repositorio remoto en caso que algún otro desarrollador haya eliminado alguna rama remota.

- `git reset --hard HEAD`:

Elimina los cambios realizados que aún no se hayan hecho *commit*.

- `git revert <hash_commit>`

Revierte el *commit* realizado, identificado por el “hash\_commit”.

Estos códigos de git son básicos mas no son los principales de configuración, de saber cómo está configurado tu repositorio y el manejo tuyo entre las ramas de tu repositorio

Iniciar configuración inicial: **Lo primero para empezar a utiliza Git es iniciar la configuración inicial con nuestro nombre de usuario y nuestro correo**

```
$ git config --global user.name "miUsuario"
```

```
$ git config --global user.email aprendiendoingenieria@ejemplo.com
```

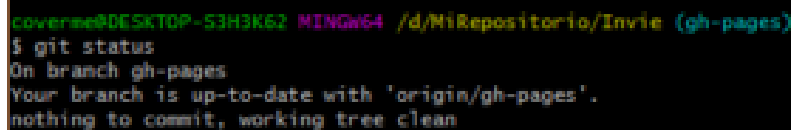
**Lista parámetros de configuración:** Si deseamos ver que parámetros de configuración estamos utilizando podemos usar los siguientes comandos:

1. \$ git config --global user.name

2. \$ git config --global user.email

## Comandos para commit

- **Conocer estado del repositorio:** Con el siguiente comando podemos conocer que esta actualizado en el repositorio y que nos falta por añadir.  
1 \$ git status



```
coverme@DESKTOP-53H3K62 MINGW64 /d/MiRepositorio/Invie (gh-pages)
$ git status
On branch gh-pages
Your branch is up-to-date with 'origin/gh-pages'.
nothing to commit, working tree clean
```

**Añadir archivos al stage:** Si queremos añadir todos los archivos que hemos creado nuevos o que hemos modificado al stage para posteriormente hacer un commit podemos utilizar 2 comandos

1 \$ git add .

1 \$ git add -a

**Añadir un archivo al stage:** Si en vez de añadir todos los archivos solo queremos añadir uno, podemos utilizar:

1 \$ git add archivo.extension

**Quitar archivo del stage:** Si hemos añadido un archivo por error al stage, podemos eliminarlo antes de hacer commit utilizando:

1 \$ git reset HEAD ficheroAñadido.extension

**Hacer commit:** Para añadir un nuevo commit con los archivos que se han añadido al stage:

- 1 \$ git commit -m "mensaje"
- 1 \$ git add ficheroOlvidado
- 2 \$ git commit --amend -m "Puedo cambiar el mensaje"

**Listar Commit:** Para listar todos los commit que posee el repositorio:

- 1 \$ git log

**Viajar entre commit:** Para moverse entre commit:

- 1 \$git checkout sha

**Volver al commit master:** Para volver al último commit:

- 1 \$ git checkout master

**Borra commit:** Si queremos borrar el ultimo commit que hemos realizado podemos utilizar el siguiente comando: (Es muy importante utilizar este comando solo **si no se ha compartido el commit con nadie más**)

- 1 \$ git reset --soft sha

Elimina el commit pero **no cambia los archivos** que se han modificado

**1 \$ git reset --hard sha**

**Elimina el commit y restaura y cambia el código a como estaba en el anterior commit.**

**Deshacer cambios desde el ultimo commit:** Para deshacer el código escrito desde el último commit utilizamos:

- 1 \$ git checkout -- fichero1.extension

## Comandos para ramas

```
coverme@DESKTOP-53H1K62 MINGW64 /d/MiRepositorio/Invie (gh-pages)
$ git branch desarrollo

coverme@DESKTOP-53H1K62 MINGW64 /d/MiRepositorio/Invie (gh-pages)
$ git branch
* desarrollo
  gh-pages
  master

coverme@DESKTOP-53H1K62 MINGW64 /d/MiRepositorio/Invie (gh-pages)
$ git branch -D desarrollo
Deleted branch desarrollo (was c8db358).

coverme@DESKTOP-53H1K62 MINGW64 /d/MiRepositorio/Invie (gh-pages)
$ git branch
* gh-pages
  master

coverme@DESKTOP-53H1K62 MINGW64 /d/MiRepositorio/Invie (gh-pages)
$ |
```

Crear una rama: **Para crear una nueva rama:**

```
1 $ git branch nombreRama
```

**Listas ramas:** Para ver las ramas de nuestro repositorio:

```
1 $ git branch
```

**Moverse entre ramas:**

```
1 $ git checkout 'nombreRama'
```

**Eliminar una rama:**

```
1 $ git branch -D nombreRama
```

**Fusionar ramas:** Para fusionar ramas primero nos vamos a la rama donde deseamos integrar otra rama y utilizamos:

```
1 $ git merge nombreDeRama
```

**Donde nombreDeRama es el nombre de la rama que queremos integrar en la rama que nos encontramos.**

**Como se derivan las ramas en git?**

En git se derivan 4 tipos de distintas ramas como:

### ***Master***

Es la rama principal. Contiene el repositorio que se encuentra publicado en producción, por lo que debe estar siempre estable.

### ***Development***

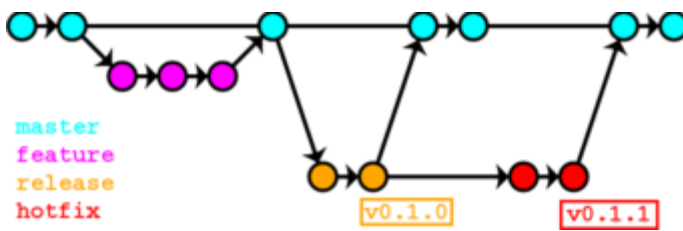
Es una rama sacada de *Master*. Es la rama de integración, todas las nuevas funcionalidades se deben integrar en esta rama. Luego que se realice la integración y se corrijan los errores (en caso de haber alguno), es decir que la rama se encuentre estable, se puede hacer un merge de development sobre la rama *Master*.

### ***Features***

Cada nueva funcionalidad se debe realizar en una rama nueva, específica para esa funcionalidad. Estas se deben sacar de *Development*. Una vez que la funcionalidad esté desarrollada, se hace un merge de la rama sobre *Development*, donde se integrará con las demás funcionalidades.

### ***Hotfix***

Son errores de software que surgen en producción, por lo que se deben arreglar y publicar de forma urgente. Es por ello, que son ramas sacadas de *Master*. Una vez corregido el error, se debe hacer una unificación de la rama sobre *Master*. Al final, para que no quede desactualizada, se debe realizar la unificación de *Master* sobre *Development*.



**Ejemplo sencillo, una sola vía, sin incluir fase de desarrollo *development***

## Referencias

Extraído del canal de YouTube de Google . Consultado el 20 de julio de 2014.

<https://www.youtube.com/watch?v=4XpnKHJAok8&t=1m30s>

Comandos git. Recuperado de

<http://aprendiendoingenieria.es/resumen-de-comandos-de-git/>

Linus Torvalds (8 de abril de 2005). Recuperado de

<https://marc.info/?l=linux-kernel&m=111293537202443>