**SCV O SISTEMA DE CONTROL DE VERSIONES**

Tomado del siguiente [Manual GIT](https://git-scm.com/book/es/v2) en <https://git-scm.com/book/es/v2>:

*Un control de versiones es un sistema que registra los cambios realizados en un archivo o conjunto de archivos a lo largo del tiempo, de modo que puedas recuperar versiones específicas más adelante.*

## SCV CENTRALIZADOS

* El enfoque tradicional de los SCV es el de un sistema centralizado.
* Es el caso de los muy utilizados CVS O Subversión.
* Esto es fácil de administrar, pero obliga a que el servidor siempre esté disponible.

## SCV DISTRIBUIDOS

* Git, como Mercurial, Bazaar o Darcs, es un sistema distribuido.
* Los clientes mantienen una copia completa de todo el repositorio.
* Esto permite hacer cambios sin estar conectados.
* En algún momento deben sincronizarse los clientes y el servidor o servidores.

## GIT

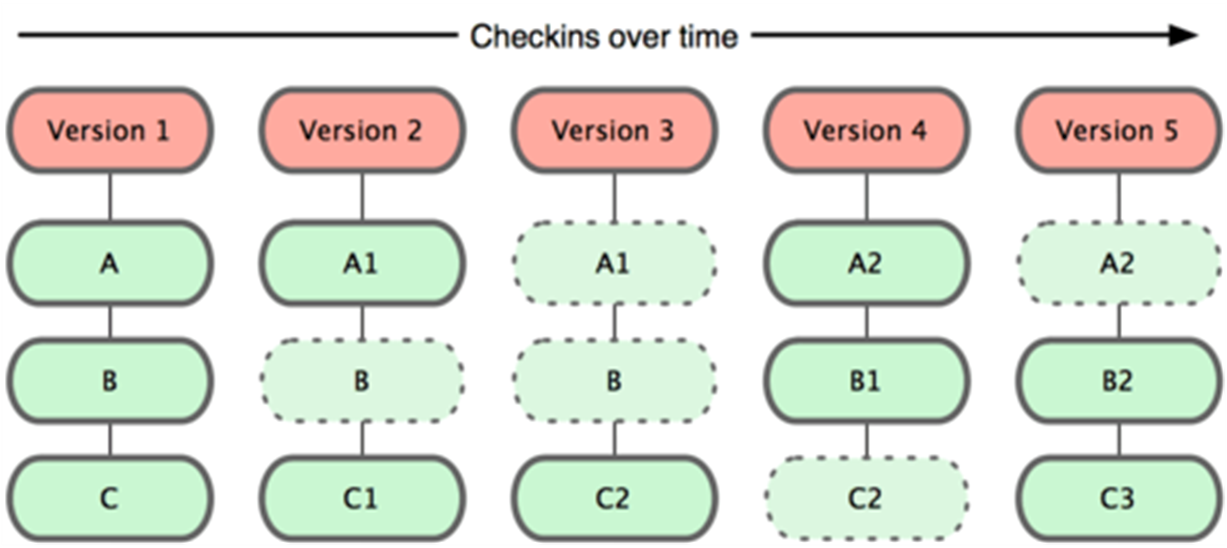
* Linux usaba BitKeeper para su desarrollo, pero en 2005 dejó de ofrecer una licencia gratis para el proyecto GNU/Linux.
* Git se creó para sustituir a BitKeeper
* Linus Torvaldas, creador de Linux, desarrolló Git para dar soporte al desarrollo de este SO.

### LAS CARACTERÍSTICAS DE GIT:

* Es distribuido
* Sencillo y rápido
* Permite grandes proyectos con multitud de ramas
* Usa copias completas. Hace copia de los ficheros modificados y enlaces a los no modificados.
* Asegura la integridad. Realiza un hash SHA-1 a cada fichero. Si un fichero es alterado git lo detecta.
* La mayoría de cambios se hacen localmente.

**ESQUEMA DE ALMACENAMIENTO**

* Cada instantánea se identifica por un *hash* SHA1
* Cada instantánea o *commit* añade los ficheros modificados. Y guarda enlaces al resto.



**INSTALACIÓN**

* Vamos a usar Ubuntu o alguna de sus variantes (Mint, Lubuntu, ...)
* Instalación de git:
* sudo apt-get update
* sudo apt-get install git
* En Windows, descargamos desde [aquí](https://git-scm.com/download/win) y ejecutamos.

**CONFIGURACIÓN INICIAL**

* Configuración necesaria para cada commit que haga:
* git config --global user.name "Your Name"
* git config --global user.email "youremail@domain.com"
* Opcionalmente el editor (si no me gusta el que hay por defecto):
* git config --global core.editor vi

**GITHUB (O BITBUCKET)**

* Git puede usarse localmente y sin conectarse a ningún otro equipo.
* No obstante, el uso de un servidor remoto:
  + Facilita la compartición de código
  + Facilita el trabajo en equipo
  + Aumenta la seguridad de nuestro código.

## INSTALACIÓN

* Vamos a usar Ubuntu o alguna de sus variantes (Mint, Lubuntu, ...)
* Instalación de git:

sudo apt-get update

sudo apt-get install git

* En Windows, descargamos desde [aquí](https://git-scm.com/download/win) y ejecutamos.

## CONFIGURACIÓN INICIAL

* Configuración necesaria para cada commit que haga:
* git config --global user.name "Your Name"
* git config --global user.email "youremail@domain.com"
* Opcionalmente el editor (si no me gusta el que hay por defecto):

git config --global core.editor vi

## NIVELES DE CONFIGURACIÓN DE GIT

Todo lo aquí contado puede verse con más detalle en el citado manual.

* Git tiene 3 niveles de configuración, cada nivel sobreescribe el anterior:
  + Para todos los usuarios: /etc/gitconfig
  + Para un usuario: ~/.gitconfig (opción --global)
  + Para un repositorio: .git/config
* Para ver los parámetros configurados:

git config –list

* Viene bien tener una [chuleta de comandos de Git](https://services.github.com/kit/downloads/github-git-cheat-sheet.pdf)

### GITHUB (O BITBUCKET)

* Git puede usarse localmente y sin conectarse a ningún otro equipo.
* No obstante, el uso de un servidor remoto:
  + Facilita la compartición de código
  + Facilita el trabajo en equipo
  + Aumenta la seguridad de nuestro código.

### USO DE GITHUB O BITBUCKET

* Nos registramos en [Github](https://github.com/) o [Bitbucket](https://bitbucket.org/)
* La comunicación entre un repositorio local y uno remoto puede ser mediante https o ssh.
* Por cuestiones prácticas consideramos mejor usar ssh. Veamos que configuración requiere.

#### CONFIGURACIÓN DE SSH

* Evitaremos introducir usuario/contraseña en nuestra consola
* Accedemos a nuestra cuenta
* Vamos a los settings del **USUARIO** y asociamos una ssh-key
* Como creamos nuestra ssh-key:

ssh-keygen

* Copiaremos el contenido de ~/.ssh/id\\_rsa.pub a una nueva clave ssh en GitHub

# INICIAR REPOSITORIO

## PARA EMPEZAR.

* Podemos iniciar el repositorio en GitHub o Bitbucket
* Podemos iniciarlo localmente.
* Supongamos que hemos iniciado nuestro proyecto y tomamos la segunda opción.

## OBJETIVO

* Tomar una carpeta como nuestro repositorio.
* Puede ser la carpeta de ejercicios de programación.
* Puede ser la carpeta de un proyecto concreto.

.

### ¿QUÉ HAGO LOCALMENTE?

* Voy a git bash y me muevo a mi directorio
* Tecleo lo siguiente:
  + git init
  + echo "# Repositorio de ejemplo" >> README.md
  + git add README.md
  + git commit -m "Commit inicial"

### ¿QUÉ HAGO EN GITHUB/BITBUCKET?

* Creo repositorio
* Sigo las instrucciones para subir mi repositorio local:

## OPCIÓN B. INICIAR EL REPOSITORIO EN GITHUB/BITBUCKET

* Al crear el repositorio se da la posibilidad de añadir un fichero README.md y crear un commit inicial.
* Por otra parte, podemos partir de un repositorio nuestro o ajeno ya creado.
* En tal caso basta con hacer:
* git clone <url repo> #habitual
* git clone <url repo> <carpeta> #permite definir destino

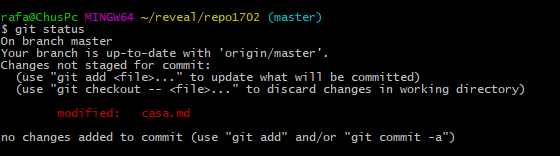
## USO DE GIT: GUARDAR CAMBIOS

Todas estas operaciones son locales:

* No es necesario salir a la red
* Velocidad muy alta de las operaciones
* Lee de tu base de datos local
* Calcula diferencias entre ficheros en local
* No limita el trabajo sin conexión

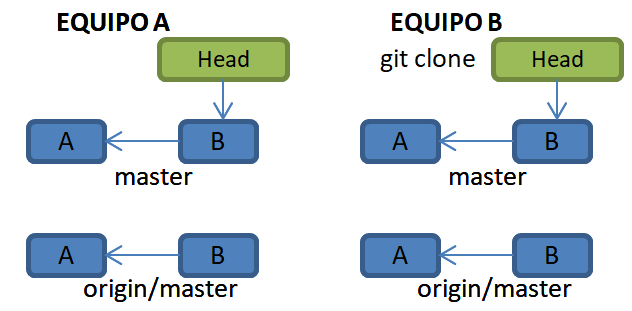
### VER EL ESTADO

* Consultar el estado del repositorio: git status
* Nos dirá qué ficheros hay nuevos o modificados
* La rama que estamos, por defecto master
* Si estamos por detrás/delante del origin, el repositorio remoto



### GUARDAR EN REMOTO.

* En local por defecto trabajamos en la rama master.
* Si hay un remoto, además se guarda oculta la rama origin/master.
* Sean los equipos A y B, y mismo repositorio:



### ¿QUÉ HAGO LOCALMENTE?

* Voy a git bash y me muevo a mi directorio
* Tecleo lo siguiente:
  + git init
  + echo "# Repositorio de ejemplo" >> README.md
  + git add README.md
  + git commit -m "Commit inicial"

### ¿QUÉ HAGO EN GITHUB/BITBUCKET?¿QUÉ HAGO EN GITHUB/BITBUCKET?

* Creo repositorio
* Sigo las instrucciones para subir mi repositorio local:

## UOPCIÓN B. INICIAR EL REPOSITORIO EN GITHUB/BITBUCKET

* Al crear el repositorio se da la posibilidad de añadir un fichero README.md y crear un commit inicial.
* Por otra parte podemos partir de un repositorio nuestro o ajeno ya creado.
* En tal caso basta con hacer:
* git clone <url repo> #habitual

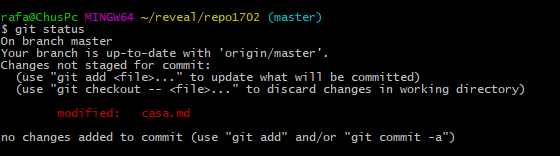
git clone <url repo> <carpeta> #permite definir destino

## SO DE GIT: GUARDAR CAMBIOS

Todas estas operaciones son locales:

* No es necesario salir a la red
* Velocidad muy alta de las operaciones
* Lee de tu base de datos local
* Calcula diferencias entre ficheros en local
* No limita el trabajo sin conexión

### VER EL ESTADO

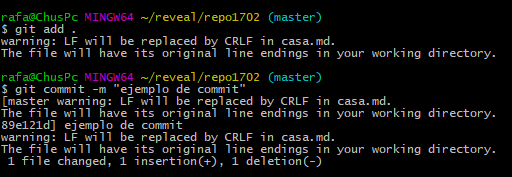
* Consultar el estado del repositorio: git status
* Nos dirá qué ficheros hay nuevos o modificados
* La rama que estamos, por defecto master
* Si estamos por detrás/delante del origin, el repositorio remoto
* 

### PREPARAR Y COMPROMETER CAMBIOS:

* Con git add ... pasamos los ficheros modificados/nuevos a preparados.
* Tras hacer esto se guarda temporalmente los cambios realizados.
* Con `git commit -m "...." comprometemos los cambios.

git add .

git commit -m "comentario explicativo"



#### EJEMPLOS DE git add

git add <nombre fichero> #un sólo fichero

git add img/logo.jpg

git add <nombre directorio> #todo lo de un directorio

git add . #caso particular, todo, todo...

git add public/js

#### RECORRIDO COMPLETO:

### git status

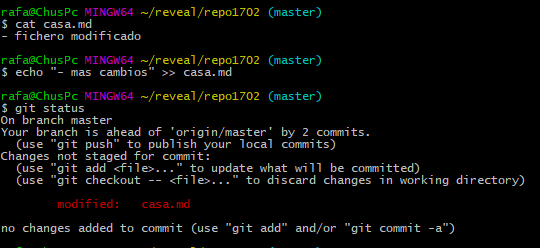
### git status

### git status

### git status

#### MARCHA ATRÁS DE UN FICHERO MODIFICADO

* Hemos modificado un fichero y queremos recuperar la anterior versión:



git checkout <nombrefichero>

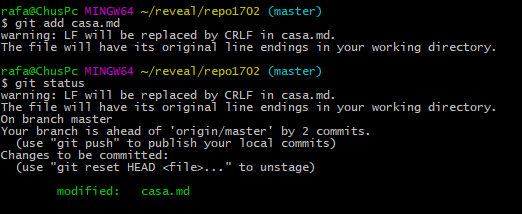
git checkout casa.md

git checkout <nombre directorio>

git checkout .

#### MARCHA ATRÁS DE UN FICHERO PREPARADO

* Queremos sacar del estado preparado después de git add:



git reset HEAD <nombrefichero>

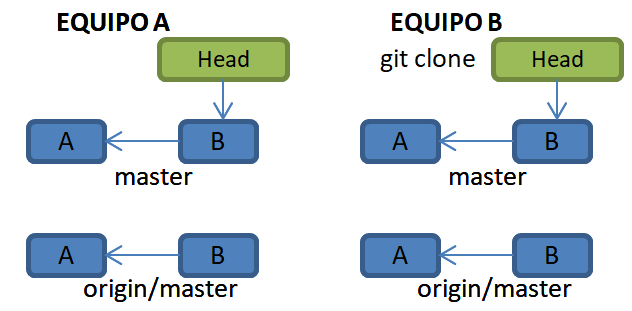
git reset HEAD casa.md

git reset HEAD <nombre directorio>

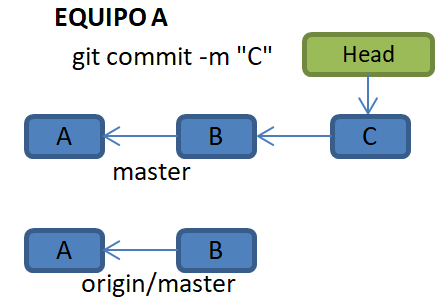
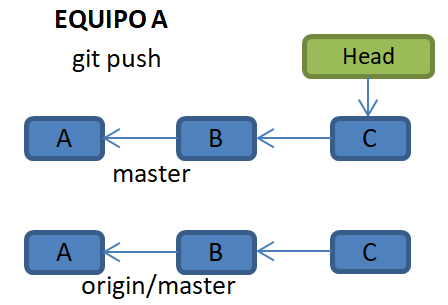
git reset HEAD .

### GUARDAR EN REMOTO.

* En local por defecto trabajamos en la rama master.
* Si hay un remoto, además se guarda oculta la rama origin/master.
* Sean los equipos A y B, y mismo repositorio:



#### COMMIT + PUSH, EQUIPO A

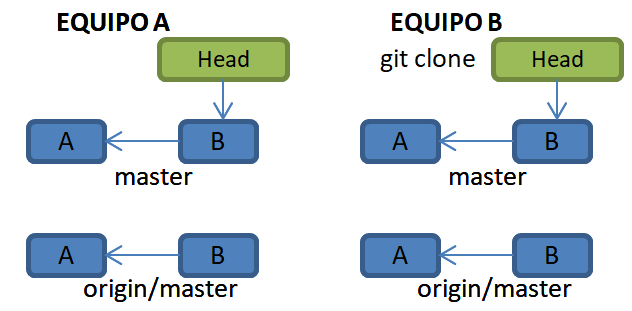
 

## RAMAS (BRANCH)

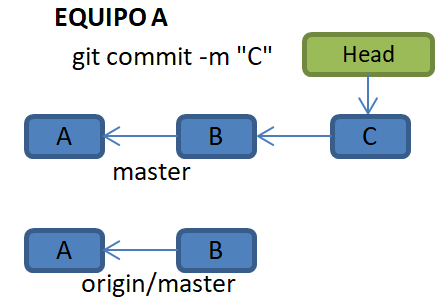
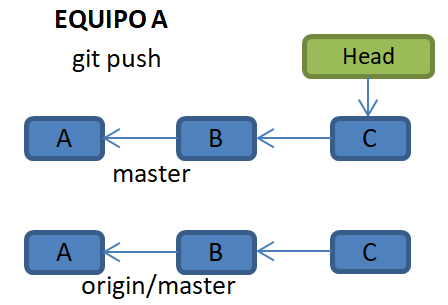
* Por defecto trabajamos en la rama master
* Las ramas se usan fundamentalmente para separar tareas sin modificar la rama master
* Una vez terminada la tarea se funde (merge) la rama de tarea con la rama master.
* Esto permite cambiar de rama/tarea y dejarla incompleta sin dejar el proyecto en versiones incompletas o inestables.

### GUARDAR EN REMOTO.

* En local por defecto trabajamos en la rama master.
* Si hay un remoto, además se guarda oculta la rama origin/master.
* Sean los equipos A y B, y mismo repositorio:



#### COMMIT + PUSH, EQUIPO A

#### GIT PUSH

* Push es el comando usado para subir código a origin, es decir el repositorio remoto.
* Origin es el nombre usado por defecto para la copia remota.
* Ejemplos de uso

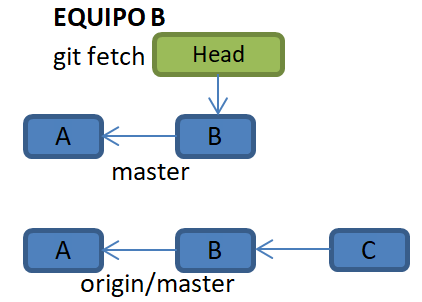
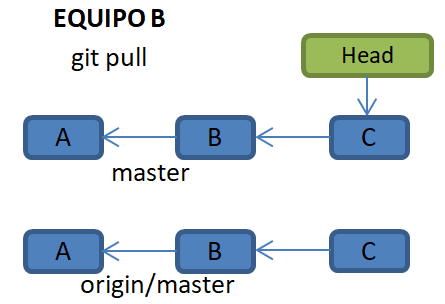
git push # sube rama "preferida"

# comando usado en la subida inicial tras git init. Lo explicaremos

git push -u origin master

#### FETCH + PULL, EQUIPO B

* Se recomienda hacer git status entre ambos.
* En realidad pull es la suma de dos comandos: fetch + merge

### ¿CÓMO USAR DOS EQUIPOS: AULA Y CASA?

* Al empezar a trabajar, ejecutar
* git fetch
* git status
* git pull
* Al acabar la sesión de trabajo
* git commit
* git status
* git push

## RAMAS (BRANCH)

* Por defecto trabajamos en la rama master
* Las ramas se usan fundamentalmente para separar tareas sin modificar la rama master
* Una vez terminada la tarea se funde (merge) la rama de tarea con la rama master.
* Esto permite cambiar de rama/tarea y dejarla incompleta sin dejar el proyecto en versiones incompletas o inestables.

### COMANDOS CON RAMAS

git branch // lista de ramas

git branch <rama> // crear rama

git checkout <rama> // cambiar de rama

git checkout -b <rama> // crear y cambiar rama 2 en 1

git branch -d <rama> // borrar rama

### FUNDIR RAMAS

git checkout master // nos ponemos en rama master

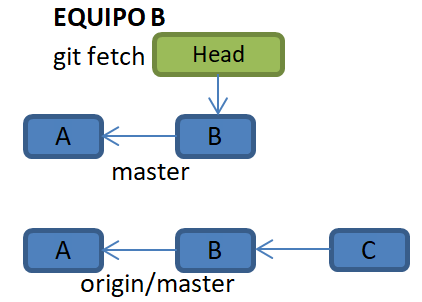
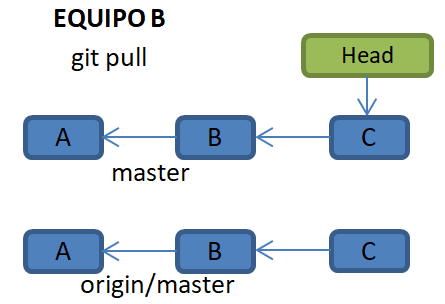
git merge <rama> //fundimos con la rama deseada

## REVISANDO CÓDIGO

* Los comandos básicos son log y diff
* git log nos muestra información histórica
* git diff compara el contenido de los ficheros

#### FETCH + PULL, EQUIPO B

* Se recomienda hacer git status entre ambos.
* En realidad pull es la suma de dos comandos: fetch + merge

### ¿CÓMO USAR DOS EQUIPOS: AULA Y CASA?

* Al empezar a trabajar, ejecutar
* git fetch
* git status
* git pull
* Al acabar la sesión de trabajo
* git commit
* git status
* git push

### FORK & PULL REQUEST

#### FORK

* Clon de un repositorio ajeno en nuestro espacio
* Es una funcionalidad de GitHub/Bitbucket
* P.ej. el repositorio repo1702
* El clon es de mi propiedad y lo puedo clonar y editar

#### PULL REQUEST

* Es una solicitud de que el dueño original incorpore mis cambios.
* Se hace en la interfaz web
* El dueño original lo acepta o no...

### COMMIT --AMEND

* Consiste en rectificar el commit anterior
* Los cambios añadidos sobreescriben el último commit
* OJO! Puede ser peligroso.

## RAMAS (BRANCH)

* Por defecto trabajamos en la rama master
* Las ramas se usan fundamentalmente para separar tareas sin modificar la rama master
* Una vez terminada la tarea se funde (merge) la rama de tarea con la rama master.
* Esto permite cambiar de rama/tarea y dejarla incompleta sin dejar el proyecto en versiones incompletas o inestables.

### COMANDOS CON RAMAS

git branch // lista de ramas

git branch <rama> // crear rama

git checkout <rama> // cambiar de rama

git checkout -b <rama> // crear y cambiar rama 2 en 1

git branch -d <rama> // borrar rama

### FUNDIR RAMAS

git checkout master // nos ponemos en rama master

git merge <rama> //fundimos con la rama deseada

## REVISANDO CÓDIGO

* Los comandos básicos son log y diff
* git log nos muestra información histórica
* git diff compara el contenido de los ficheros

### FORK & PULL REQUEST

#### FORK

* Clon de un repositorio ajeno en nuestro espacio
* Es una funcionalidad de GitHub/Bitbucket
* P.ej. el repositorio repo1702
* El clon es de mi propiedad y lo puedo clonar y editar

#### PULL REQUEST

* Es una solicitud de que el dueño original incorpore mis cambios.
* Se hace en la interfaz web
* El dueño original lo acepta o no...

### COMMIT --AMEND

* Consiste en rectificar el commit anterior
* Los cambios añadidos sobreescriben el último commit
* OJO! Puede ser peligroso.

**RAMAS (BRANCH)**

* Por defecto trabajamos en la rama master
* Las ramas se usan fundamentalmente para separar tareas sin modificar la rama master
* Una vez terminada la tarea se funde (merge) la rama de tarea con la rama master.
* Esto permite cambiar de rama/tarea y dejarla incompleta sin dejar el proyecto en versiones incompletas o inestables.

### COMANDOS CON RAMAS

git branch // lista de ramas

git branch <rama> // crear rama

git checkout <rama> // cambiar de rama

git checkout -b <rama> // crear y cambiar rama 2 en 1

git branch -d <rama> // borrar rama

### FUNDIR RAMAS

git checkout master // nos ponemos en rama master

git merge <rama> //fundimos con la rama deseada

### REVISEMOS EL USO DE PUSH

* Push sube código a un repositorio remoto
* Origin es el alias usado pra el repositorio remoto
* Podríamos usar más de un remoto
* Master es el nombre de la rama por defecto
* git push # sube rama "preferida"
* git push <repo remoto> <rama> # sube una rama concreta
* git push origin dev # Ej. sube rama dev
* git push -u <repo remoto> <rama># sube y predetermina rama

git push -u origin master # Ej. sube y pred. master

### RAMAS REMOTAS

* Si queremos compartir una rama de trabajo
* Subir una rama:
* git push origin <rama>
* Bajar la rama tiene dos partes:
* git fetch #crea la rama oculta origin/<rama>
* Las siguientes ordenes son idénticas.
* Ambas crear la rama local asociada a origin/
* La primera nos permite alterar el nombre .
* git checkout -b <rama> origin/<rama>

git checkout --track origin/<rama>

## REVISANDO CÓDIGO

## REVISANDO CÓDIGO

* Los comandos básicos son log y diff
* git log nos muestra información histórica
* git diff compara el contenido de los ficheros

### LOG

### LOG

* Log soporta una gran cantidad de parámetros:
* Veámoslo con ejemplos:
* git log #uso base
* git log -<n> #log de los últimos n commits
* git log --oneline -5 #lista de commits breve
* git log --follow [file] #lista de commit con cambios para "file"
* git show <commit> # información completa de un commit concreto

### DIFF

* Git diff sirve para ver las diferencias entre el estado actual y otro:

git diff # diferencia entre estado actual y el preparado o comprometido

git diff --cached #diferencia entre el preparado y el último commit

git diff --stat #idem al primero pero con información resumida de cambios

### DIFF DESDE UN COMMIT CONCRETO

 Podemos hacer diff sobre todo el repo o sobre un fichero.

 Podemos referirnos a un commit por su hash o contando hacia atrás desde el HEAD o desde una rama.

git diff <commit> # cambios desde un commit hasta la actualidad

git diff <commit> <file># idem fichero concreto

git diff master~2 README.md #Cambios en los dos últimos commit de master

git diff HEAD~2 README.md #Cambios en los dos últimos commit HEAD

git diff cd598e4 README.md #Cambios desde un commit concreto por hash

## TAGS)

### DIFF ENTRE DOS COMMIT

git diff <commit>:<file> <commit2>:<file> #diff de un fichero entre dos commits

git diff cd598e4:README.md 31422ac:README.md

git diff master~20:README.md master~1:README.md

## GITIGNORANDO COSAS: .GITIGNORE

* Podemos decir a git que no tenga en consideración algunos ficheros/directorios
* Para hacerlo debemos crear un fichero .gitignore en el directorio raiz
* Para ver algunos ejemplos puedes visitar <https://www.atlassian.com/git/tutorials/gitignorenuevo>

## ETIQUETAS (TAGS)

**GITIGNORANDO COSAS: .GITIGNORE**

* Podemos decir a git que no tenga en consideración algunos ficheros/directorios
* Para hacerlo debemos crear un fichero .gitignore en el directorio raiz
* Para ver algunos ejemplos puedes visitar <https://www.atlassian.com/git/tutorials/gitignorenuevo>

**ETIQUETAS (TAGS)**

**pendiente**