

# Git practico

lunes, 22 de enero de 2018 0:21

- Operaciones CRUD sobre los commits
  - C -> git add / git commit
  - R -> git status / git log
  - U -> git commit --amend
  - D -> git reset
- Operaciones de organización de código
  - Stashing
  - Reset
  - Revert
  - Repositorios remotos
- Gestión de ramas y etiquetas
  - Ramas
  - Merge
  - Resolución de conflictos
  - Tags
  - Rebase
- Workflow

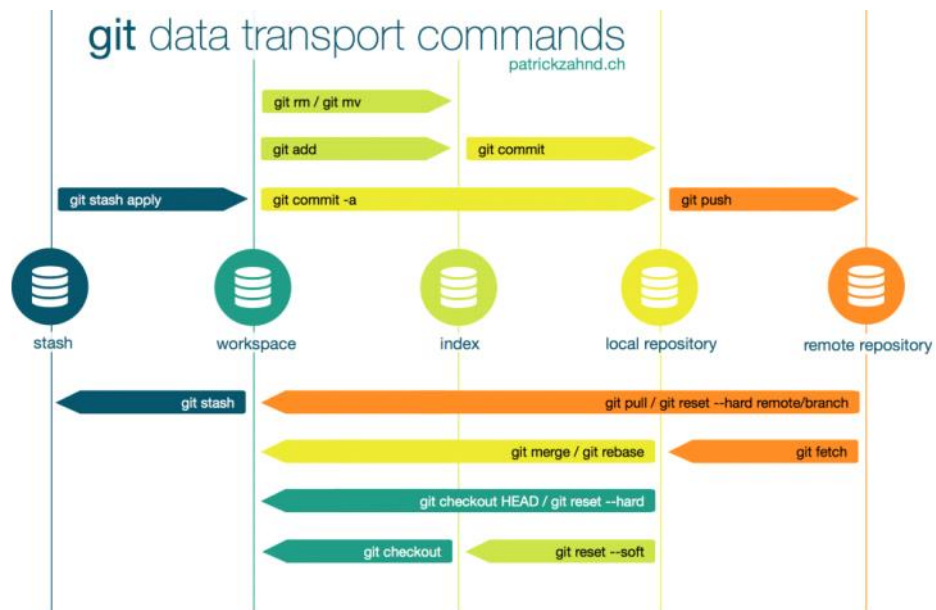
# Operaciones básicas en Git

sábado, 28 de octubre de 2017 13:41

Por analogía con las bases de datos se puede hablar de CRUD

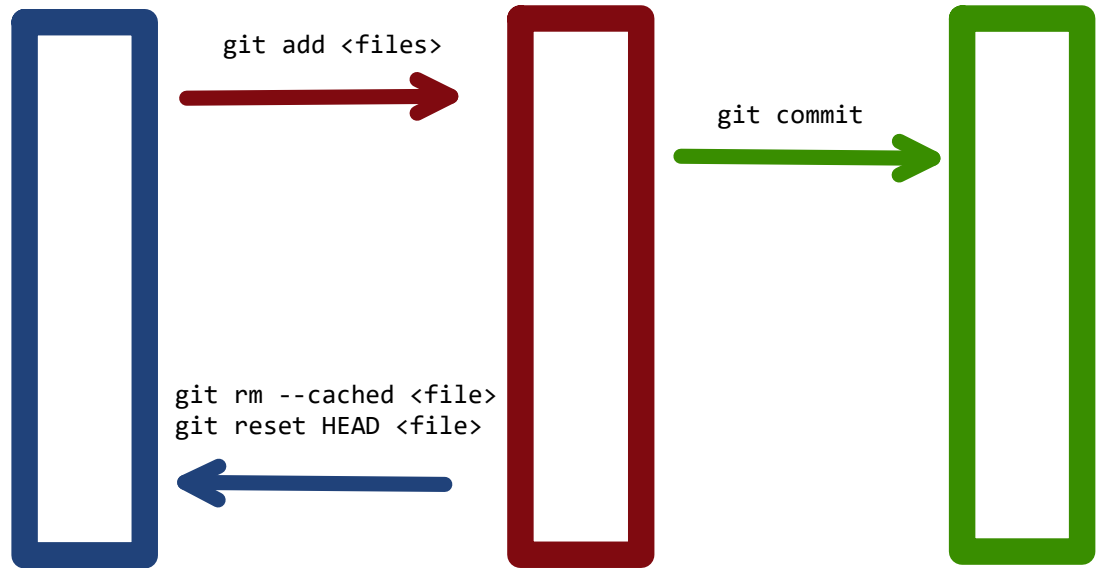
- **Create** -> *git add / git commit.*
- **Read** -> *git status / git log.*
- **Update** -> *git commit --amend.*
- **Delete** -> *git reset.*

```
git add
git status
git reset HEAD
git commit
git log
git reset
```



# Comandos

lunes, 8 de enero de 2018 23:17



# Actualización del repositorio (Create)

sábado, 28 de octubre de 2017 13:44

El proceso de añadir un archivo al repositorio (*commit*) suele realizarse en 2 etapas

1. Añadir elementos a la zona de almacenamiento o indexación temporal (*staging area*) como paso previo para añadirlos al repositorio

- un fichero  
\$ git add <filename>  
\$ git add \*.c  
\$ git add index.html
- un directorio  
\$ git add <directory>
- todo el contenido de la *working area*  
\$ git add <directory>

```
D:\Desarrollo\Tools\GitRepo>git add index.html

D:\Desarrollo\Tools\GitRepo>git status
On branch master

No commits yet

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   index.html
```

2. Añadir al repositorio (*commit*) el contenido de la *staging area*, incorporando un mensaje que permita identificar cada actualización

```
$ git commit -m "message"
```

```
D:\Desarrollo\Tools\GitRepo>git commit -m "Creado index.html"
[master (root-commit) 2fb236e] Creado index.html
 1 file changed, 12 insertions(+)
 create mode 100644 index.html

D:\Desarrollo\Tools\GitRepo>git status
On branch master
nothing to commit, working tree clean

D:\Desarrollo\Tools\GitRepo>git log
commit 2fb236ec95010156687719187031031c94fd338f (HEAD -> master)
Author: alce65 <alce65@hotmail.es>
Date:   Sat Oct 28 14:56:57 2017 +0200

    Creado index.html
```

Información de todos los commits del repositorio, con su identificador único, los datos del usuario que hizo el commit ...

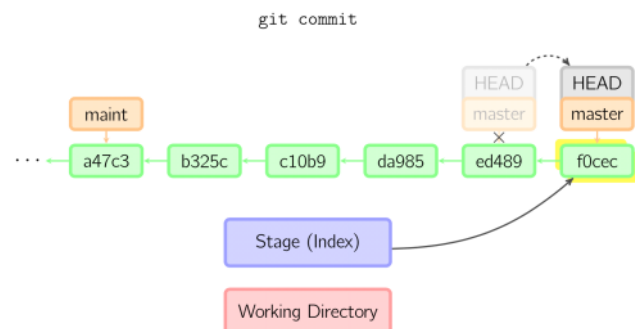
Más conciso mediante  
git log --oneline

```
$ git commit -m "Message"

$ git commit -am "Message"

$ git commit -m "Message" <file>

$ git commit --amend
```



## Ejemplo

lunes, 8 de enero de 2018 22:08

```
D:\desarrollo\Gits\Git_Basisc [master]> md mapas

Directorio: D:\desarrollo\Gits\Git_Basisc

Mode                LastWriteTime         Length Name
----                -
d-----          04/01/2018     18:45             mapas

D:\desarrollo\Gits\Git_Basisc [master]> cd .\mapas\
D:\desarrollo\Gits\Git_Basisc\mapas [master]> echo mapa1 > mapa01.map
D:\desarrollo\Gits\Git_Basisc\mapas [master +1 ~0 -0 !]> echo mapa2 > mapa02.map
D:\desarrollo\Gits\Git_Basisc\mapas [master +1 ~0 -0 !]> dir

Directorio: D:\desarrollo\Gits\Git_Basisc\mapas

Mode                LastWriteTime         Length Name
----                -
-a----          04/01/2018     18:46             16 mapa01.map
-a----          04/01/2018     18:46             16 mapa02.map
```

Se crea una nueva carpeta

```
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 !]> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    mapas/

nothing added to commit but untracked files present (use "git add" to track)
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 !]> git add mapas
D:\desarrollo\Gits\Git_Basisc [master +2 ~0 -0 ~]> git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   mapas/mapa01.map
    new file:   mapas/mapa02.map
```

git status

git add

Ventajas de que exista esta zona:  
poder seleccionar solo una parte del  
contenido de la *workarea* para incorporarlo  
a un *commit*

```
D:\desarrollo\Gits\Git_Basisc [master +2 ~0 -0 ~]> git commit -m "Primeros mapas"
[master 5ee8efb] Primeros mapas
2 files changed, 0 insertions(+), 0 deletions(-)
create mode 100644 mapas/mapa01.map
create mode 100644 mapas/mapa02.map
D:\desarrollo\Gits\Git_Basisc [master]> git log
commit 5ee8efb895f14abff8264e60e4cca392d99e7f4f (HEAD -> master)
Author: alce65 <alce65@hotmail.es>
Date: Thu Jan 4 18:50:46 2018 +0100

    Primeros mapas

commit e4b35526d2e26ce5dff076a6c1233b7c015087e4
Author: alce65 <alce65@hotmail.es>
Date: Thu Jan 4 18:42:07 2018 +0100

    Readme inicial
D:\desarrollo\Gits\Git_Basisc [master]> git log --online
5ee8efb (HEAD -> master) Primeros mapas
e4b3552 Readme inicial
D:\desarrollo\Gits\Git_Basisc [master]>
```

git commit

git log

## Eliminación de temporales

martes, 26 de diciembre de 2017 21:25

Eliminar elementos de la zona de almacenamiento o indexación temporal (*staging area*), i.e. revertir *add*:

```
# Antes del primer commit  
$ git rm --cached <file>
```

(antes del primer *commit* aún no existe el puntero HEAD y por tanto no puede hacerse referencia a él)

```
# Después del primer commit, cuando ya existe HEAD  
$ git reset HEAD <files>
```

```
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 !]> echo funcionalidad > index.js  
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 !]> git add .  
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 ~]> git status  
On branch master  
Changes to be committed:  
  (use "git reset HEAD <file>..." to unstage)  
  
    new file:   index.js  
  
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 ~]> git reset HEAD .  
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 !]> git status  
On branch master  
Untracked files:  
  (use "git add <file>..." to include in what will be committed)  
  
    index.js  
  
nothing added to commit but untracked files present (use "git add" to track)  
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 !]> _
```

git add .

git reset HEAD .

Elimina un fichero de la *staging area*

## Commit & Add

lunes, 8 de enero de 2018 23:24

Cuando un fichero ya está *track* (trazado), i.e. el repositorio ya conoce su existencia, pueden combinarse *add* y *commit* en una sola operación

```
$ git commit -am "message"
```

Staging & Commit unidos en caso de ficheros "trazados" (tracking)

```
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 !]> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        index.js

nothing added to commit but untracked files present (use "git add" to track)
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 !]> git commit -am "Index.js"
On branch master
Untracked files:
  index.js

nothing added to commit but untracked files present
```

Un archivo nuevo "untracked" no puede ser objeto del comando *commit -am*

```
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 !]> git add .
D:\desarrollo\Gits\Git_Basisc [master +1 ~0 -0 ~]> git commit -m "Index.js"
[master 834a7b3] Index.js
 1 file changed, 0 insertions(+), 0 deletions(-)
 create mode 100644 index.js
```

En este caso es necesario ejecutar los dos comandos por separado

```
D:\desarrollo\Gits\Git_Basisc [master]> code .
D:\desarrollo\Gits\Git_Basisc [master]> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.js

no changes added to commit (use "git add" and/or "git commit -a")
D:\desarrollo\Gits\Git_Basisc [master +0 ~1 -0 !]> git commit -am "Modificando index.js"
[master 2c24170] Modificando index.js
 1 file changed, 0 insertions(+), 0 deletions(-)
```

Un fichero modificado, previamente trazado (*tracked*) en el repositorio SI puede ser objeto del comando *commit -am*

*git commit -am*

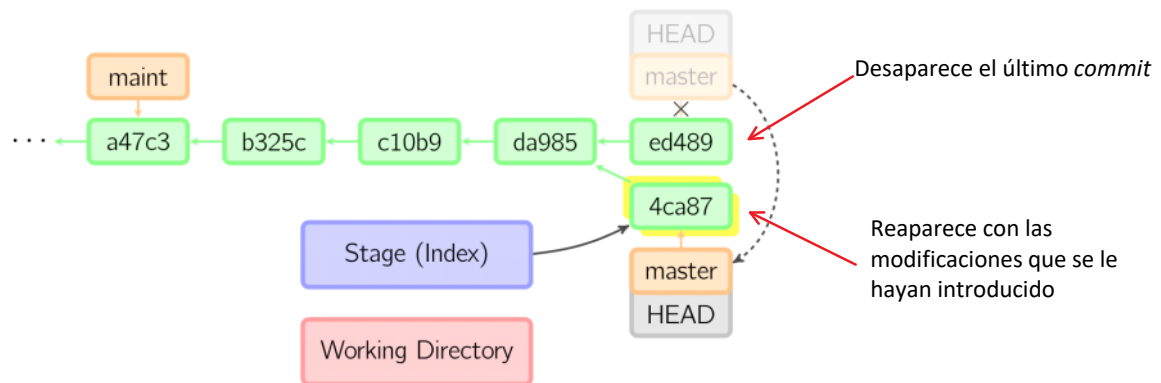
*git checkout -- <file>*  
elimina las modificaciones de un fichero en la *working area*

## Modificación del último *commit*

miércoles, 24 de enero de 2018 23:18

`git commit -amend`

`git commit --amend`





## Ejemplo

viernes, 5 de enero de 2018 7:43

```
git commit -amend
```

```
D:\desarrollo\Gits\Git_Basisc [master +0 ~1 -0 !]> git log --oneline
2c24170 (HEAD -> master) Modificando index.js
834a7b3 Index.js
5ee8efb Primeros mapas
e4b3552 Readme inicial
D:\desarrollo\Gits\Git_Basisc [master +0 ~1 -0 !]> git show
commit 2c24170a890dace9900cb6225a7625d4d29e1103 (HEAD -> master)
Author: alce65 <alce65@hotmail.es>
Date:   Fri Jan 5 07:57:21 2018 +0100

    Modificando index.js

diff --git a/index.js b/index.js
index 08b0b6c..2e2ff1c 100644
Binary files a/index.js and b/index.js differ
```

El último *commit* puede ser modificado, incorporándole nuevo contenido

En realidad esto supone su sustitución por un nuevo *commit* con su propio identificador

```
D:\desarrollo\Gits\Git_Basisc [master +0 ~1 -0 !]> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   index.js

no changes added to commit (use "git add" and/or "git commit -a")
D:\desarrollo\Gits\Git_Basisc [master +0 ~1 -0 !]> git commit --amend
unix2dos: converting file D:\desarrollo\Gits\Git_Basisc/.git/COMMIT_EDITMSG to DOS format...
```

```
COMMIT_EDITMSG: Bloc de notas
Archivo Edición Formato Ver Ayuda
Completando index.js

# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
#
# Date:      Fri Jan 5 07:57:21 2018 +0100
#
# On branch master
# Changes to be committed:
#       modified:   index.js
#
# Changes not staged for commit:
#       modified:   index.js
#
```

El editor de textos configurado para Git permite modificar el mensaje de *commit*

```
dos2unix: converting file D:/desarrollo/Gits/Git_Basisc/.git/COMMIT_EDITMSG to Unix format..
.
[master aabf624] Completando index.js
Date: Fri Jan 5 07:57:21 2018 +0100
1 file changed, 0 insertions(+), 0 deletions(-)
D:\desarrollo\Gits\Git_Basisc [master +0 ~1 -0 !]> git log --oneline
aabf624 (HEAD -> master) Completando index.js
834a7b3 Index.js
5ee8efb Primeros mapas
e4b3552 Readme inicial
D:\desarrollo\Gits\Git_Basisc [master +0 ~1 -0 !]> git show
commit aabf62474ae968b89084b2bc6d4497165351938e (HEAD -> master)
Author: alce65 <alce65@hotmail.es>
Date:   Fri Jan 5 07:57:21 2018 +0100

    Completando index.js

diff --git a/index.js b/index.js
index 08b0b6c..2e2ff1c 100644
Binary files a/index.js and b/index.js differ
```

El resultado es en realidad un nuevo *commit* con su nuevo identificador

# Renombrado

miércoles, 27 de diciembre de 2017 19:38

Si se renombra un fichero desde fuera del repositorio, Git lo interpreta como dos acciones

```
PS D:\Desarrollo\Git_Basic> ren fichero.txt nuevo_nombre.txt
PS D:\Desarrollo\Git_Basic> git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        deleted:    fichero.txt

Untracked files:
  (use "git add <file>..." to include in what will be committed)

        nuevo_nombre.txt

no changes added to commit (use "git add" and/or "git commit -a")
PS D:\Desarrollo\Git_Basic>
```

El borrado de un fichero

La creación de un fichero

En versiones previas

La primera operación se recoge mediante un `git add`

La segunda, para transmitir la eliminación al repositorio, mediante un `git rm`

En la actualidad, es suficiente un `git add`

```
PS D:\Desarrollo\Git_Basic> git add .
PS D:\Desarrollo\Git_Basic> git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    fichero.txt -> nuevo_nombre.txt

PS D:\Desarrollo\Git_Basic>
```

Otra alternativa es indicarle a Git que lleve a cabo el renombrado, usando `git mv`

```
PS D:\Desarrollo\Git_Basic> git mv fichero.txt nuevo_nombre.txt
PS D:\Desarrollo\Git_Basic> git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        renamed:    fichero.txt -> nuevo_nombre.txt

PS D:\Desarrollo\Git_Basic>
```

# Acceso a los commits

martes, 2 de enero de 2018 18:09

Acceder a un de tres maneras diferentes:

1. Referencias absolutas
  - a. A través de su ID (1234567)
2. Referencias relativas
  - A través del nombre de la rama.
  - Sabiendo que por defecto la rama está en el último *commit*
  - a. A través del puntero HEAD

Cada *commit* en Git tienen un ID único correspondiente a un número hexadecimal basado en el algoritmo SHA1.

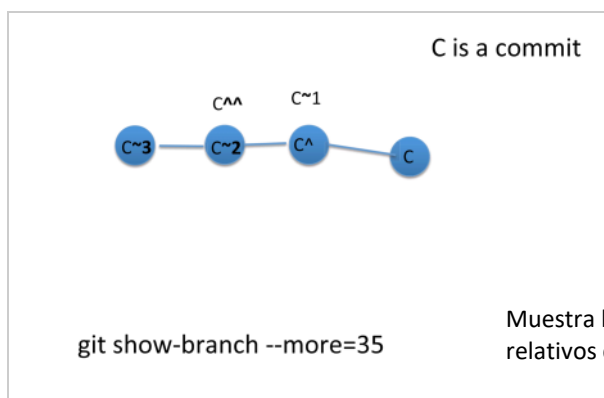
Cada ID es globalmente único, no sólo en un repositorio, sino a nivel de todos los existentes

Git proporciona mecanismos para identificar un a *commit* con respecto a alguna de las referencias (punteros) a otro *commit*, como son HEAD o el nombre de rama:

^: Un *commit* antes (master^, master^^, HEAD^, etc.)

~: N *commits* antes (master~2, HEAD~4)

~ -> altGr 4  
-> alt 126



## Contenido de un commit

martes, 9 de enero de 2018 0:16

Para mostrar un *commit* se utiliza `git show <referencia>`

```
D:\desarrollo\Gits\timetravel [master]> git show master
commit 95431a3ffce7e8208293877a36e3ba63bb89f637 (HEAD -> master)
Author: alce65 <alce65@hotmail.es>
Date:   Fri Jan 5 14:06:41 2018 +0100

    Dia 1

diff --git a/prueba.txt b/prueba.txt
new file mode 100644
index 0000000..b474fba
Binary files /dev/null and b/prueba.txt differ
```

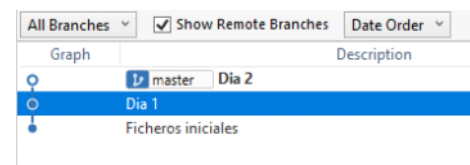
*commit* que recoge la creación de un fichero

```
D:\desarrollo\Gits\timetravel [master]> code .
D:\desarrollo\Gits\timetravel [master]> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   prueba.txt

no changes added to commit (use "git add" and/or "git commit -a")
D:\desarrollo\Gits\timetravel [master +0 ~1 -0 !]> git commit -am "Dia 2"
[master aada1bf] Dia 2
1 file changed, 0 insertions(+), 0 deletions(-)
```

modificamos el fichero y  
creamos un nuevo *commit*



```
D:\desarrollo\Gits\timetravel [master]> git show master
commit aada1bf137c68e92cf5bf80ff6a44a88e6224cee (HEAD -> master)
Author: alce65 <alce65@hotmail.es>
Date:   Fri Jan 5 16:44:12 2018 +0100

    Dia 2

diff --git a/prueba.txt b/prueba.txt
index b474fba..e4ce670 100644
Binary files a/prueba.txt and b/prueba.txt differ
```

*commit* que recoge la modificación de un fichero

## Histórico: log

martes, 2 de enero de 2018 18:11

### Opciones avanzadas de git log

Se puede ver la sucesión histórica (*history*) de los *commits*:

```
git log (igual que git log HEAD)
```

Se puede indicar a partir de que *commit* debe de empezar la serie

```
git log <commit-name>  
e.g. git log master~2
```

Se puede especificar un rango (desde .. hasta)

```
$ git log master~12..master~10
```

Se pueden mostrar los commits que afectan a un determinado path (carpeta, fichero...)

```
$ git log -- <path>
```

E.g. `git log -- README3.txt`

Se pueden mostrar los commits que coincidan con una expresión regular

```
$ git log --grep='reg-exp'
```

Se pueden excluir del listado los commits resultantes de un merge

```
git log --no-merges
```

Se pueden mostrar los cambios producidos durante un tiempo

```
$ git log --since={2010-04-18}
```

```
$ git log --before={2010-04-18}
```

```
$ git log --after={2010-04-18}
```

Se puede mostrar la salida en formato gráfico (dentro de las limitaciones de la consola)

```
git log --decorate --graph --oneline
```



```
* 17e4c5f (HEAD, develop) Merge branch 'hotfix-1.0.1' into develop  
/\   
/ * f43bb33 (hotfix-1.0.1) Hotfix2  
/ * 50a102d Hotfix1  
/ * d2fe7d6 Version 1.0.1  
/ * d534111 (tag: 1.0) Merge branch 'release-1.0'  
/\   
* / \ 77e7eac Merge branch 'release-1.0' into develop  
/\ \ \   
/ / /   
/ / /   
/ * / 9261fad (release-1.0) Release10Fix2  
/ * / 2562cb3 Release10Fix1  
/ * / c51b802 Version 1.0  
* / / fbdd638 work6  
* / / 7353285 work5  
/ /   
* / 538bb9a work4  
* / cdef254 Merge branch 'feature1' into develop  
/\
```

Sugerencia: creación de alias

# Git Blame

martes, 9 de enero de 2018 0:37

Permite conocer el autor de la última modificación de cada línea de un fichero y en que *commit* se incluyó el cambio

`git blame <archivo>`

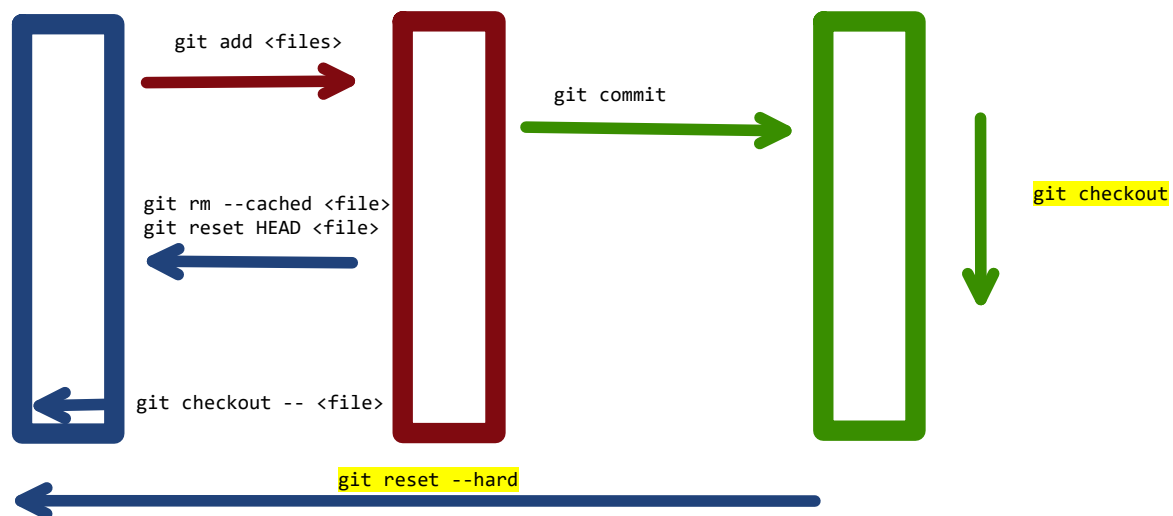
Esta también disponible cuando se accede al repositorio en GitHub

The screenshot shows the GitHub interface for a repository named 'spring-boot-angular4-boilerplate'. The file 'README.md' is selected. The top navigation bar includes tabs for 'Raw', 'Blame', and 'History'. A red arrow points from the 'Blame' tab to the 'Blame' view. The 'Blame' view displays a list of commits on the left and the corresponding code lines on the right. The commits are listed with their authors, commit hashes, and the time since they were made. The code lines are numbered 1 through 17. The first line of code is '# Spring Boot + Angular 4 Boilerplate'.

Commit	Line	Code
Upgrade to Angular 4.0.0-RELEASE and Angular CLI 1.0.0-RE...	1	# Spring Boot + Angular 4 Boilerplate
Add initial folder structure and boilerplate files	2	
docs: Update README	3	!{spring-boot-angular4-boilerplate}(https://raw.githubusercontent.com/Saka7/spring-boot-angular4-bo
	4	
Fix codeclimate badges	5	!{Code Climate}(https://codeclimate.com/github/Saka7/spring-boot-angular4-boilerplate/badges/gpa.s
	6	!{Issue Count}(https://codeclimate.com/github/Saka7/spring-boot-angular4-boilerplate/badges/issue_
Add codecoverage badges	7	
Fix codeclimate badges	8	Quick start for Spring Boot + Angular 4 projects with JWT auth
Modify sh and bat scripts	9	
Add initial folder structure and boilerplate files	10	## Includes:
	11	
	12	Front-end:
	13	
Modify README	14	- angular-cli boilerplate files
docs: Update README	15	- JWT authentication service
Add initial folder structure and boilerplate files	16	
	17	Back-end:

# Desplazamiento en el histórico

domingo, 21 de enero de 2018 10:31



```
git checkout <commit>
```

```
git reset --hard  
git checkout --<file>
```

**Checkout** : Posicionar HEAD en un *commit*  
`git checkout <id-commit al que quiero ir>`

Como consecuencia del desplazamiento del puntero HEAD pueden obtenerse distintos resultados:

- Posicionarme en un *commit* concreto en el pasado
- Cambiar de rama

Recordar

## El doble guion --

Una de sus dos funcionalidades:

Separar un nombre de archivo explícitamente identificado

```
# Checkout the tag named "main.c"  
$ git checkout main.c
```

```
#Checkout the file named "main.c"  
$ git checkout -- main.c
```

## Ejercicio: timeTravel

viernes, 5 de enero de 2018 13:58

Adelantamos el uso de `checkout` y `reset`  
Proyecto timeTravel

```
D:\desarrollo\Gits\timetravel [master +1 ~0 -0 !]> echo Dia1 > prueba.txt
D:\desarrollo\Gits\timetravel [master +1 ~0 -0 !]> git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

        prueba.txt

nothing added to commit but untracked files present (use "git add" to track)
D:\desarrollo\Gits\timetravel [master +1 ~0 -0 !]> git add .
D:\desarrollo\Gits\timetravel [master +1 ~0 -0 ~]> git commit -m "Dia 1"
[master 95431a3] Dia 1
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 prueba.txt
```

Commit 1

Contenido del fichero prueba.txt:  
Dia1

```
D:\desarrollo\Gits\timetravel [master]> code .
D:\desarrollo\Gits\timetravel [master]> git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

        modified:   prueba.txt

no changes added to commit (use "git add" and/or "git commit -a")
D:\desarrollo\Gits\timetravel [master +0 ~1 -0 !]> git commit -am "Dia 2"
[master aa20ad4] Dia 2
1 file changed, 0 insertions(+), 0 deletions(-)
D:\desarrollo\Gits\timetravel [master]> git log --oneline
aa20ad4 (HEAD -> master) Dia 2
95431a3 Dia 1
e3d80a7 Ficheros iniciales
```

Commit 2

```
D:\desarrollo\Gits\timetravel [master]> type prueba.txt
Dia1
Dia2
```

Contenido del fichero prueba.txt:  
Dia1  
Dia2

Retorno al estado del commit 1:

```
D:\desarrollo\Gits\timetravel [master]> git checkout 95431a3
Note: checking out '95431a3'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

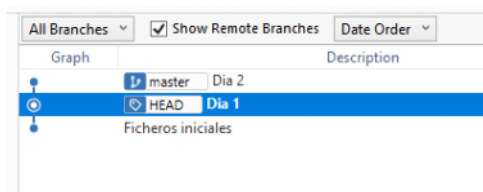
    git checkout -b <new-branch-name>

HEAD is now at 95431a3... Dia 1
D:\desarrollo\Gits\timetravel [(95431a3...)]>
```

Desde este punto, sería incorrecto añadir nuevos *commits* sin crear una nueva rama, tal como indica el mensaje de Git

```
D:\desarrollo\Gits\timetravel [(95431a3...)]> type prueba.txt
Dia1
```

Editando el ficho, su contenido sería

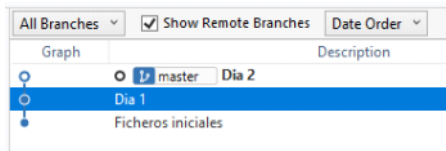


El puntero HEAD ha retrocedido, y no está en el último *commit* de la rama activa o master (está *detached*)

```
D:\desarrollo\Gits\timetravel [(95431a3...)]> git checkout master
Previous HEAD position was 95431a3... Dia 1
Switched to branch 'master'
```

Se puede volver nuevamente al último *commit* de la rama activa indicando *checkout* y su nombre (master)





El puntero HEAD vuelve a su posición "normal" en la rama master

```
D:\desarrollo\Gits\timetravel [master]> type prueba.txt
Día1
Día2
```

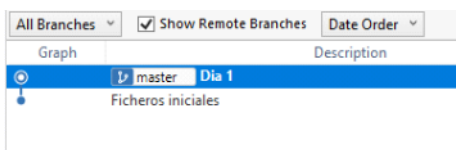
El contenido del fichero vuelve a estar completo

```
D:\desarrollo\Gits\timetravel [master]> git log --oneline
aa20ad4 (HEAD -> master) Día 2
95431a3 Día 1
e3d80a7 Ficheros iniciales
```

Otra operación que modifica más radicalmente el histórico es la eliminación de un *commit*

```
D:\desarrollo\Gits\timetravel [master]> git reset --hard master^
HEAD is now at 95431a3 Día 1
D:\desarrollo\Gits\timetravel [master]> git log --oneline
95431a3 (HEAD -> master) Día 1
e3d80a7 Ficheros iniciales
```

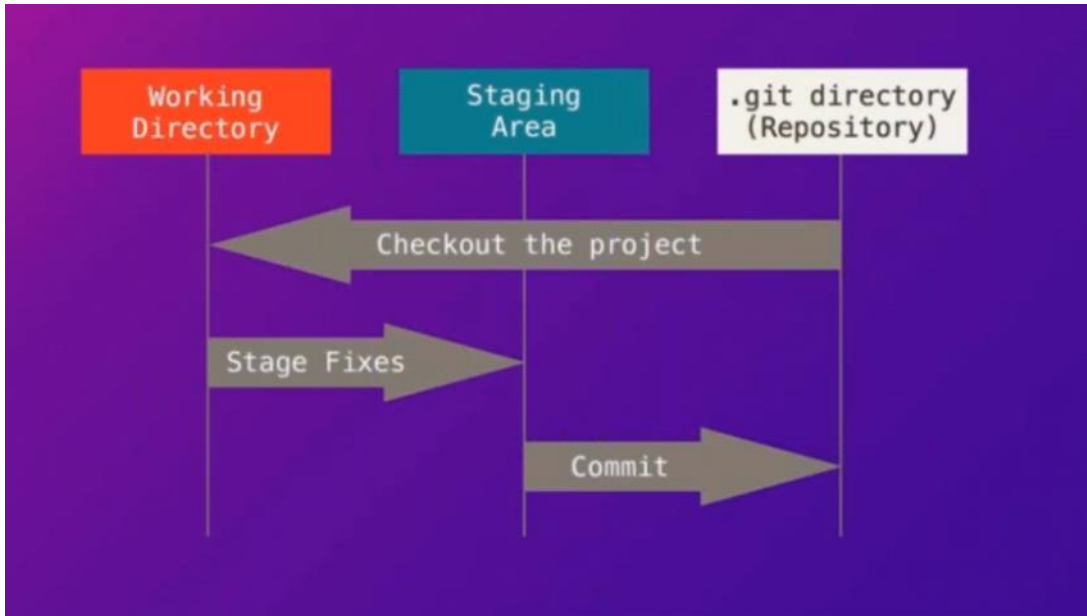
En este caso el *commit* es completamente irrecuperable



```
D:\desarrollo\Gits\timetravel [master]> type prueba.txt
Día1
```

# Resumen Git básico

miércoles, 3 de enero de 2018 19:20



## Ejemplos.

- configuración de git:
  - o user / email
  - o editor asociado
  - o terminal
- creación de un fichero
  - o git add
  - o git status
  - o git commit
- clonación desde github
- el fichero .gitignore: configuraciones - passwords - dependencias - dist
  - o global gitignore
- borrado de archivos. evitar hacerlo desde el S.O. seguido de add
  - o git rm
  - o git rm --cached (no borra del disco, sólo del repositorio)
- información de los commits y modificaciones
  - o git log
  - o formato de la salida git log --pretty=...
  - o git commit --amend
- eliminación de archivos de la staging area
  - o git rm --cached <file>
  - o git reset HEAD -- <file>
  - o git checkout -- <file> -> eliminación del archivo del area de trabajo
- repositorios remotos
- config alias

- ▷ git config
- ▷ git init
- ▷ git add
- ▷ git clone
- ▷ git status (ver siguiente slide)
- ▷ El .gitignore
- ▷ git diff
- ▷ git commit
- ▷ git rm
- ▷ git remote

<https://try.github.io/levels/1/challenges/1>

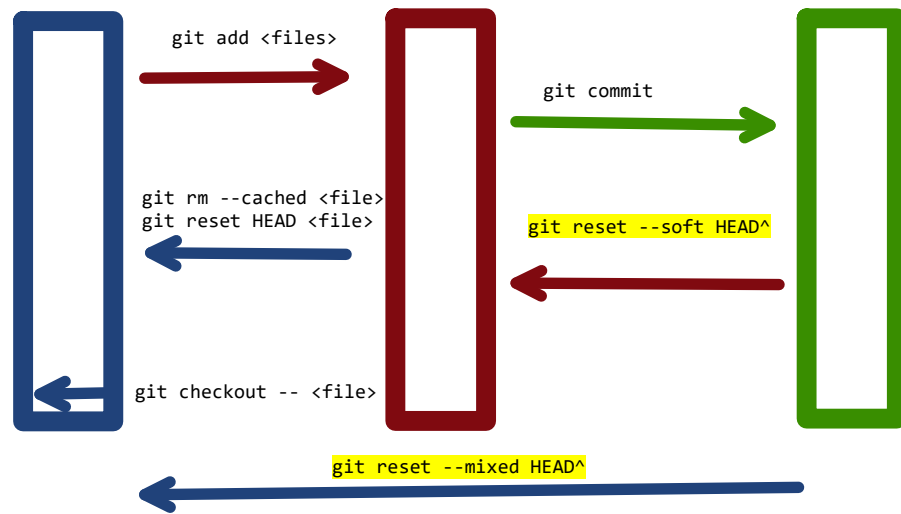
# Organización de código

lunes, 8 de enero de 2018 17:57

- Repositorios locales
  - Stashing
  - Reset
  - Revert
- Repositorios remotos

# Repositorios locales

martes, 9 de enero de 2018 0:23



# Stashing

miércoles, 27 de diciembre de 2017 19:58

Un área intermedia donde puedes hacer *commit* del código inestable con el que de momento no vas a trabajar



- área intermedia o temporal
- pila de *commits*

Utilice el *stash* (escondite) cuando desee grabar el estado actual del directorio de trabajo y el índice (*staging area*), regresando luego a un directorio de trabajo limpio.

Una excelente manera de detener aquello en lo que se está trabajando actualmente pudiendo volver a ello más tarde.

Se ocultan los cambios, enviándolos como un *commit* al *stash*  
`git stash`

recuperación de un *commit* determinado

`git stash apply:`  
E.g: `git stash apply stash@{1}`



El *stash* funciona como una pila, por lo que se recupera un *commit* determinado indicando su numero

lista de la pila de *commits* del *stash*  
`git stash list:`

recuperación y borrado del último *commit* de la pila (apply the top stash)  
`git stash pop`

borrado de un *commit* determinado Manually delete stashes  
`git stash drop <id>:`

- borrado de toda la pila de *commits*  
`git stash clear:`

Se envían al *stash* los ficheros trazados que se encuentran en la workarea, pero no lo ficheros nuevos (untracked)

- `git stash list`
- `git stash save "mensaje"`
- `git stash pop`
- `git stash show stash@{0}`
- `git stash apply stash@{0}`
- `git diff stash@{0}`

# Ejercicio

lunes, 22 de enero de 2018 0:50



*Hands on : Stashing*

# Modificación de los commits

martes, 2 de enero de 2018 19:39

Descartar los cambios de un fichero trazado realizados en la copia que se encuentra en el área de trabajos (*working area*)

```
$ git checkout -- <files>
```

Descartar los cambios de un fichero ya almacenados en la *staging area*

```
$ git reset HEAD <file>
```

Revertir un fichero concreto a su estado en un *commit* anterior

```
$ git checkout <commit-id> file
```

En este caso no se cambia el HEAD, que sigue en el último *commit*

Ejemplo. Recuperación de borrados

```
D:\desarrollo\Gits\Git_Basisc [master]> del Index.js
D:\desarrollo\Gits\Git_Basisc [master +0 ~0 -1 !]> git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

       deleted:    index.js

no changes added to commit (use "git add" and/or "git commit -a")
```

Borrado de un fichero

Al ser un fichero trazado, Git detecta el borrado como información que se debe incorporar al repositorio

```
D:\desarrollo\Gits\Git_Basisc [master +0 ~0 -1 ~]> git add .
D:\desarrollo\Gits\Git_Basisc [master +0 ~0 -1 ~]> git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

       deleted:    index.js
```

Envío a la *staging area* de la información respecto al borrado:

```
git add. // git rm <file>.
```

El último paso sería el *commit* correspondiente

## Recuperación

```
D:\desarrollo\Gits\Git_Basisc [master +0 ~0 -1 ~]> git reset HEAD .
Unstaged changes after reset:
D
   index.js
D:\desarrollo\Gits\Git_Basisc [master +0 ~0 -1 !]> git status
On branch master
Changes not staged for commit:
  (use "git add/rm <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

       deleted:    index.js

no changes added to commit (use "git add" and/or "git commit -a")
```

Eliminación de los datos de borrado de la *staging area*

```
D:\desarrollo\Gits\Git_Basisc [master +0 ~0 -1 !]> git checkout -- .  
D:\desarrollo\Gits\Git_Basisc [master]> dir
```

Directorio: D:\desarrollo\Gits\Git\_Basisc

Mode		LastWriteTime	Length	Name
d----		04/01/2018 18:46		mapas
-a----		05/01/2018 9:16	107	.gitignore
-a----		05/01/2018 9:19	16	clase.class
-a----		05/01/2018 18:37	168	index.js
-a----		04/01/2018 18:40	18	README.md

Eliminación de los datos de borrado de la *working area*

Fichero recuperado



## Flujo: reset

viernes, 5 de enero de 2018 17:49

Se puede restablecer (reset) el estado de un repositorio de diversas formas:

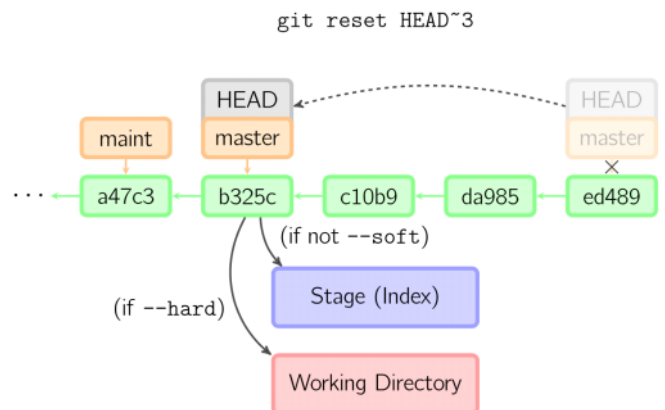
You can some status of your repository:

```
$ git reset
$ git reset HEAD <file>
$ git reset --soft <commit-id>
$ git reset --hard <commit-id>
$ git reset --mixed <commit-id>
```

<commit-id> es el commit al que se llevará el puntero, es decir el *commit* al que queremos ir.

Todos los *commits* posteriores en la rama afectada se eliminarán.

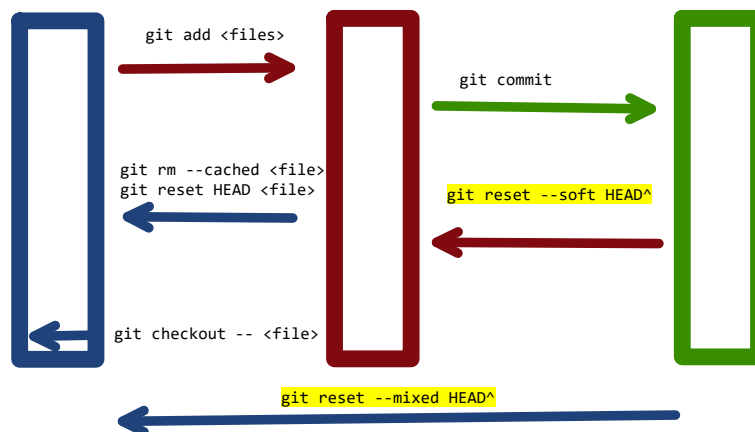
El modificador determina el destino de los cambios (*change sets*) que desaparecen del historial de *commits*



-hard : los cambios se eliminan

-soft: los cambios pasan a la *staging area*

-mixed: los cambios pasan a la *working area*



git rm --cached <file>      Depende de si ya existe HEAD o  
git reset HEAD <file>      no (sólo antes del primer *commit*)

### Fusión de *commits* = *Squashing* (aplanamiento)

Se pasan diversos *commits* a la *staging area* o a la *working area* y se reagrupan en un único *commit* para conseguir un histórico más compacto y más semántico (es decir con *commits* con significado)

En caso de un fichero modificado en diversos *commits* se conservará su estado final

Todas las operaciones que modifican los *commits* deben EVITARSE si ya se han llevado a un repositorio remoto compartido con otros usuarios

```
~/dev/git2/ejemploReset (master) $ git log --oneline
b169f69 Añadido main.js
9177059 Añadido estilos.css
e4fe1f3 Añadido index.html
aad51d8 Inicializado el repositorio
```

Commits demasiado pequeños

```
~/dev/git2/ejemploReset (master) $ git log --oneline
b169f69 Añadido main.js
9177059 Añadido estilos.css
e4fe1f3 Añadido index.html
aad51d8 Inicializado el repositorio
~/dev/git2/ejemploReset (master) $ git reset --soft master~3
~/dev/git2/ejemploReset (master) $ git commit -m "Creada página de inicio"
[master 42db63b] Creada página de inicio
3 files changed, 10 insertions(+)
create mode 100644 estilos.css
create mode 100644 index.html
create mode 100644 main.js
~/dev/git2/ejemploReset (master) $
```

commits demasiado pequeños

Un solo *commit* con  
más significado

# Revert

viernes, 5 de enero de 2018 17:50

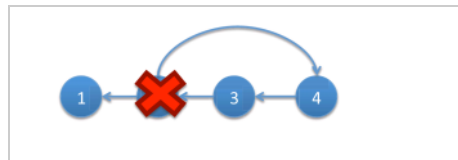
*Revert* deshace una instantánea confirmada (*commit*), pero en lugar de eliminarla, se agrega un nuevo *commit* para deshacer los cambios introducidos por el anterior.

Es decir, No elimina ningún *commit*.

Crea uno nuevo con las operaciones inversas al *commit* seleccionado

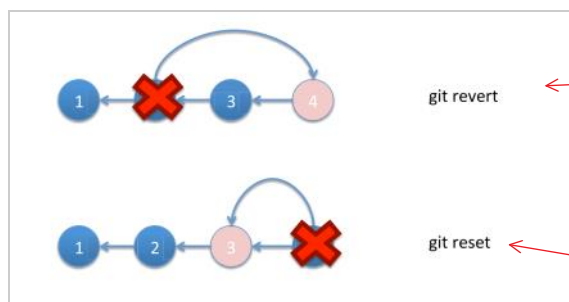
Así se evita la pérdida de ningún elemento del histórico dentro de Git

`git revert <commit>`



*Revert* deshace un único *commit*, no "revierte" al estado anterior de un proyecto.

## Revert v. Reset



git revert

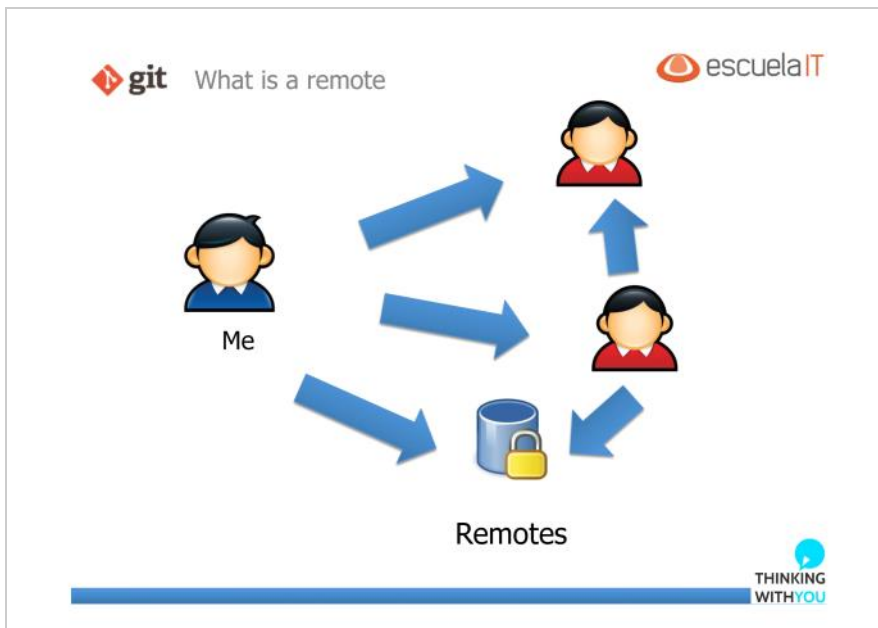
← Crea un *commit* nuevo cuyo EFECTO es eliminar un *commit* anterior

git reset

← Elimina un *commit*

# Repositorios remotos

jueves, 4 de enero de 2018 19:09



Git utiliza *remote* y ramas remotas (*tracking branch*) como referencias que facilitan la conexión con otros repositorios (conocidos como remotos)

Recordamos

Repositorio local -> subirlo o "empujarlo" (*push*) a remoto  
comando `git remote`

```
2. Proyecto git en local --> subirlo a github
a. Usuario en Github
b. Crear el repositorio local (proyectoParaGithub)
c. Crear el repositorio en Github
   http://github.com/ialcazar/proyectoParaGithub
d. Añadir el repositorio remoto dentro de mi repositorio local
   git remote add origin <url>
   git remote -v: Listar todos los repos remotos dados de alta en mi local
```

Finalmente

```
$ git push origin master
```

Además de `git clone`, otros comandos comunes de Git que hacen referencia a los repositorios remotos son:

`git fetch` ← bajar cambios

`git pull` ← bajar cambios y fusionarlos con local (Fetch + Merge)

`git push` ← subir a remoto

# Git Remote

miércoles, 27 de diciembre de 2017 20:49

El comando para manipular las referencias *remote* es .

```
$ git remote
```

Modificadores del comando:

add: añade una referencia a un remoto

rm: elimina una referencia

rename: cambia el nombre

-v: lista todas las referencias a remotos

# Git Push

martes, 23 de enero de 2018 0:02

subir a remoto  Transfiere objetos y sus metadatos relacionados a un repositorio remoto

```
$ git push <repo> <branch>
```

No es posible si el remoto está más actualizado que el local  
En ese caso es posible un push forzado, pero siendo consciente de  
porque razón se sobrescribe un remoto más reciente

```
$ git push -f <repo> <branch>
```

# Git Fetch

martes, 23 de enero de 2018

0:11

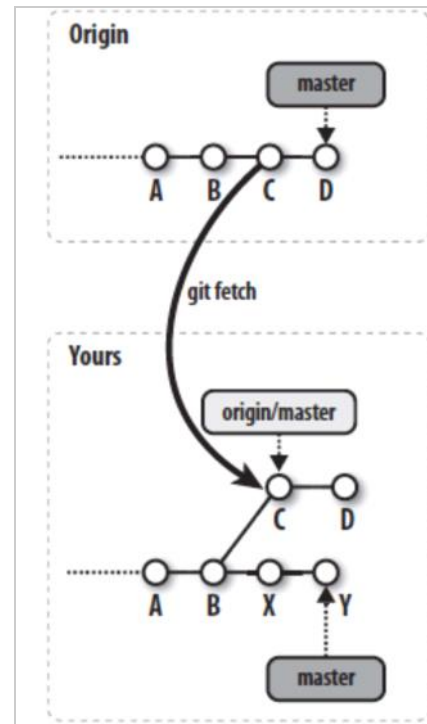
bajar cambios



Recupera objetos y sus metadatos relacionados desde un repositorio remoto

\$ git fetch <repo> <branch>

No los fusiona, sino que los deja como un rama separada



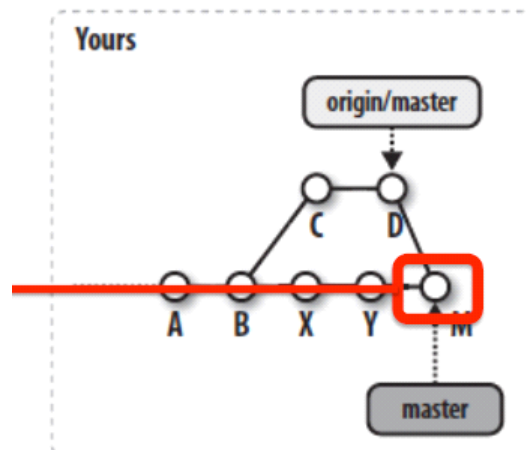
# Git Pull

lunes, 22 de enero de 2018 22:53

Combina un fetch con la operación de fusión de ramas, **merge**

```
$ git merge <branch>
```

Se crea un nuevo commit



**Git Pull** = Git Fetch + Git Merge

Recupera objetos y sus metadatos relacionados desde un repositorio remoto

Exactamente el mismo que haciendo ambos por separado

```
$ git pull <repo> <branch>
```

Cuando clonas un repositorio, solo puedes clonar la rama principal.

Comandos relacionados

```
$ git checkout --track -b <local-branch> <remoterepo> / <remote-branch>
```

Recupera una rama remota en una rama local y cambia a dicha rama.

```
$ git branch <local-branch> <repo>/<remotebranch>
```

Recupera una rama remota en una rama local



# Git Pull --rebase

miércoles, 24 de enero de 2018 23:07

Resultado del comando  
`git pull origin master`  
cuando existen modificaciones  
diferentes en el local y el remoto

local master -> commit1  
remote master -> commit2

```
1 Merge branch 'master' of https://gitlab.com/danirod/PaginaCliente
2 # Please enter a commit message to explain why this merge is necessary,
3 # especially if it merges an updated upstream into a topic branch.
4 #
5 # Lines starting with '#' will be ignored, and an empty message aborts
6 # the commit.
```

```
Username for 'https://gitlab.com': danirod
Password for 'https://danirod@gitlab.com':
remote: Counting objects: 3, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From https://gitlab.com/danirod/PaginaCliente
 * branch      master      -> FETCH_HEAD
  e86b310..6c1ca16 master    -> origin/master
Merge made by the 'recursive' strategy.
ModificacionChula | 0
1 file changed, 0 insertions(+), 0 deletions(-)
create mode 100644 ModificacionChula
```

## Alternativa

`git pull -rebase origin master`

ordena los *commits* de las 2 ramas reordenado el historial para evitar el uso de una estrategia recursiva y el consiguiente *commit* que aparece como reflejo de esta

```
remote: Counting objects: 2, done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 2 (delta 1), reused 0 (delta 0)
Unpacking objects: 100% (2/2), done.
From https://gitlab.com/danirod/PaginaCliente
 * branch      master      -> FETCH_HEAD
  6c1ca16..2d7e68a master    -> origin/master
First, rewinding head to replay your work on top of it...
Applying: Agrega un archivo de prueba.
Applying: Elimina archivo prueba
```

# Ramas remotas

miércoles, 27 de diciembre de 2017 21:04

Las ramas remotas (*Tracking branches*) son referencias al estado de las ramas en tus repositorios remotos.

- Son ramas locales que no puedes mover;
- se mueven automáticamente cuando estableces comunicaciones en la red.

Las ramas remotas funcionan como marcadores, para recordarte en qué estado se encontraban tus repositorios remotos la última vez que conectaste con ellos.



Se usan exclusivamente para seguir los cambios desde otro repositorio.



No se pueden realizar *merge* (fusionar) o *commits* en una rama remota.

# Gestión de ramas y etiquetas

martes, 9 de enero de 2018 0:23

- Ramas
- Fusión de ramas
- Etiquetas (Tags)

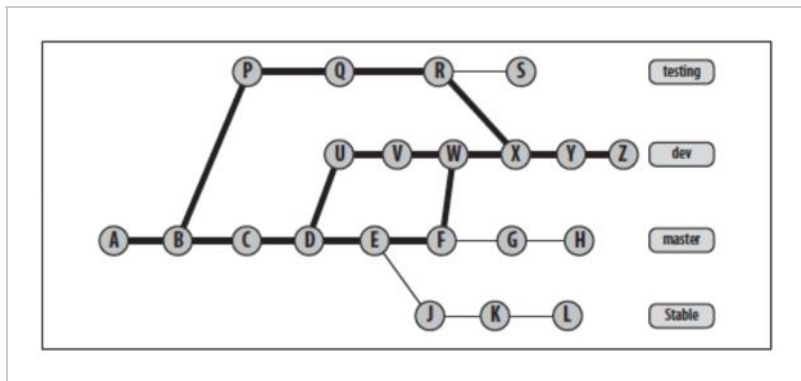
# Ramas

miércoles, 3 de enero de 2018 13:38

Una rama es una línea de desarrollo separada,  
una división a partir de un estado primario unificado, que

- permite que el desarrollo continúe en múltiples direcciones simultáneamente
- potencialmente, produzca diferentes versiones del proyecto.

A menudo, una rama se reconcilia y se fusiona con otras ramas para reunir esfuerzos dispares.



## Nombres de las ramas

La rama por defecto del repositorio es “master”

Los nombres de las demás deben respotar ciertas reglas

- / está permitido (no al final)
- Ningún componente separado por barra puede comenzar con un punto (.)  
Por ejemplo . feature / .new no es válido
- No puede contener dos puntos consecutivos (..)
- No puede contener espacios en blanco, ~, ^, :, ?, \*, [

Buena práctica nombrar las ramas como ramas/nombre : SourceTree lo representara como una carpeta con todas las ramas

## Uso de ramas

Puede haber muchas ramas diferentes dentro de un repositorio en cualquier momento dado.

Hay a lo sumo una rama "activa" que determina qué archivos están desprotegidos en el directorio de trabajo.

Por defecto, master es la rama activa.

# Creación de ramas

miércoles, 24 de enero de 2018 23:23

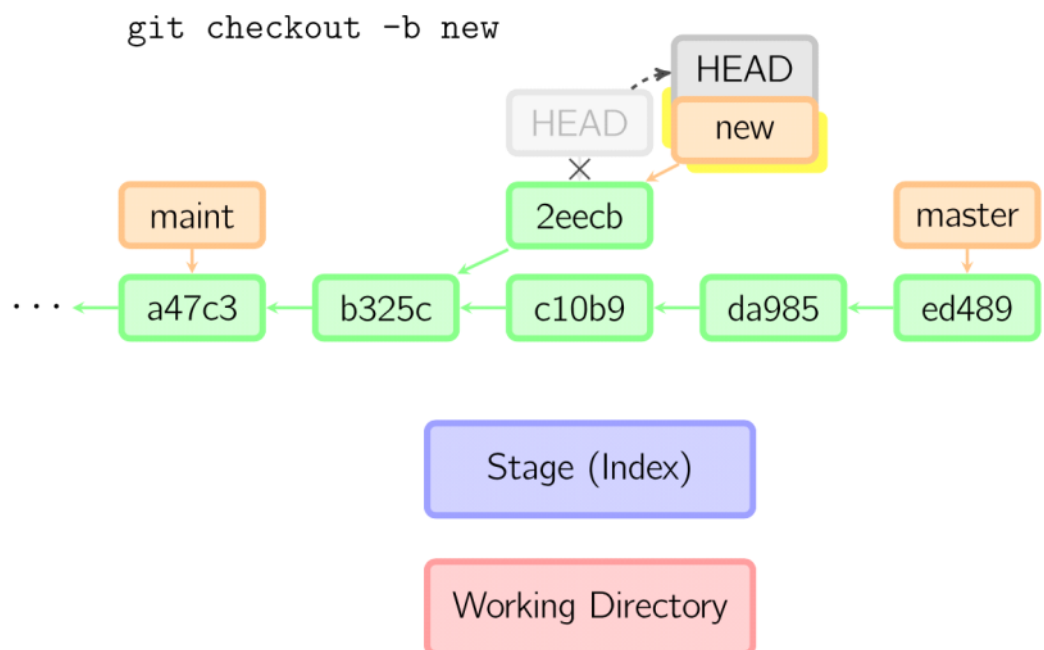
A partir de la rama activa

```
$ git branch <branch-name>
```

Sólo crea la rama

```
$ git checkout -b <branch-name>
```

Crea la rama y se posiciona en ella como rama activa



# Gestión de ramas

sábado, 6 de enero de 2018 18:51

- Creación
- Listado
- Selección -> git checkout
- Renombrado
- Borrado

## Listado de ramas

Lista de nombres de ramas encontrados en el repositorio:

```
$ git branch
```

```
$ git branch -a
```

Enumera todas las ramas, incluyendo ramas remotas (*tracked branches*)

## Vista de ramas

```
$ git show-branch <branch1> <branch2>
```

```
$ git show-branch bug/*
```

Limita el resultado a determinados nombres de ramas

## Cambio de rama

Solo hay UNA rama ACTIVA, es decir, el directorio de trabajo (working directory) solo puede ser el reflejo de una única rama en un momento dado.

Para empezar a trabajar en una rama diferente:

```
$ git checkout <branch-name>
```

En genera el papel  
del comando es:

*checkout* -> posiciona el HEAD en un *commit* determinado

- permite ir a cualquier *commit* anterior
- permite cambiar de una rama a otra

## Renombrado de ramas

```
$ git branch -m <branch-old> <branch-new>
```

## Borrado de ramas locales

No se puede borrar la rama activa

```
$ git branch -d <branch>
```

Borra sólo si la rama se ha fusionado (merge) previamente.

```
$ git branch -D <branch> :
```

Fuerza el borrado incluso si la rama aun no se ha fusionado (merge).

## Borrado de ramas remotas

2.- Push changes

```
$ git push <remote> :<branch>
```

Ejemplo:

```
git push remote :ramas/altaClientes
```

## Ejemplos

sábado, 6 de enero de 2018 18:54

Ejemplo. Cambios en el histórico a partir de un *commit* anterior  
(a partir del pasado)

```
D:\desarrollo\Gits\timetravel [master]> git checkout master^
Note: checking out 'master^'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

    git checkout -b <new-branch-name>

HEAD is now at 95431a3... Dia 1
D:\desarrollo\Gits\timetravel [(95431a3...)]> git checkout -b vidaDos
Switched to a new branch 'vidaDos'
D:\desarrollo\Gits\timetravel [vidaDos]> git log --oneline
95431a3 (HEAD -> vidaDos) Dia 1
e3d80a7 Ficheros iniciales
D:\desarrollo\Gits\timetravel [vidaDos]> code .
```

```
D:\desarrollo\Gits\timetravel [vidaDos]> git log --oneline
95431a3 (HEAD -> vidaDos) Dia 1
e3d80a7 Ficheros iniciales
D:\desarrollo\Gits\timetravel [vidaDos +0 ~1 -0 !]> git status
On branch vidaDos
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

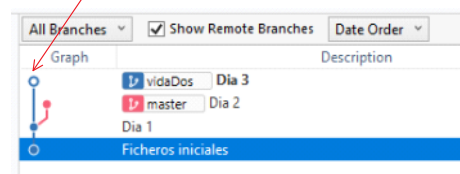
        modified:   prueba.txt

no changes added to commit (use "git add" and/or "git commit -a")
D:\desarrollo\Gits\timetravel [vidaDos +0 ~1 -0 !]> git commit -am "Dia 3"
[vidaDos 229ccc4] Dia 3
1 file changed, 0 insertions(+), 0 deletions(-)
D:\desarrollo\Gits\timetravel [vidaDos]> git log --oneline
229ccc4 (HEAD -> vidaDos) Dia 3
95431a3 Dia 1
e3d80a7 Ficheros iniciales
```

Cambiar historico en el pasado

- Posicionarme en el commit que quiero modificar  
`git checkout <id> / master^^`
- Crear una nueva rama  
`git checkout -b nombreRama`
- Trabajar en los cambios sobre mi working director
- `add / commit`
- Volver al presente  
`git checkout master`  
`git checkout <nombreRama>`

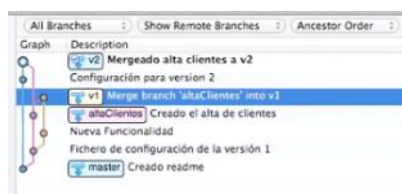
El círculo vacío indica  
Indica la rama que está activa



Ejemplo: Reset a un commit anterior a la creación de una rama: los commits que pertenecían a las dos ramas se mantienen

Los commits son nodos, y mientras exista un puntero a un nodo (i.e. una rama) el nodo no se elimina

Ejemplo con funcionalidades y versiones



Se sincroniza con un remoto en GitHub, al que se sube además de master, las ramas v1 y v2

Cuando se clona un repositorio, únicamente se crea en local la copia de la rama master.  
A continuación es posible clonar cualquier otra de las ramas que estén disponibles en el repositorio.



```
clonar ramas

git clone --> master

git branch <nombreRamaLocal> <nombreRemoto>/<nombreRamaRemota>
Clonar la rama, pero no os posicionais en la rama nueva
git checkout <nombreRamaLocal>

git checkout -b <nombreRamaLocal> <nombreRemoto>/<nombreRamaRemota>
Clonando la rama y posicionandome en ella con el mismo comando

git checkout -b v1 origin/v1

git branch v1 origin/v1 + git checkout v1
```

# Merge

miércoles, 27 de diciembre de 2017 20:42

Una fusión (*merge*) unifica dos o más ramas del historial de *commits*.

- Generalmente la fusión une solo dos ramas.
- Git admite una fusión de tres, cuatro o muchas ramas al mismo tiempo
- Todas las ramas que se fusionarán deben estar presentes en el mismo repositorio.

Cuando las modificaciones en una rama no entran en conflicto con las de la otra rama → nuevo *commit*

Cuando las ramas entran en conflicto (alteran la misma línea) Git no resuelve el conflicto automáticamente.

## Procedimiento

Tienes que cambiar a la rama de destino y ejecutar la fusión:

```
$ git checkout destinyBranch  
$ git merge anotherBranch
```

Antes de comenzar una fusión (*merge*) hay que ordenar el directorio de trabajo

Si se comienza una fusión en un estado desorganizado (*dirty*), es posible que Git no pueda combinar los cambios de todas las ramas y los de las áreas de su trabajo e índice de en una sola pasada.

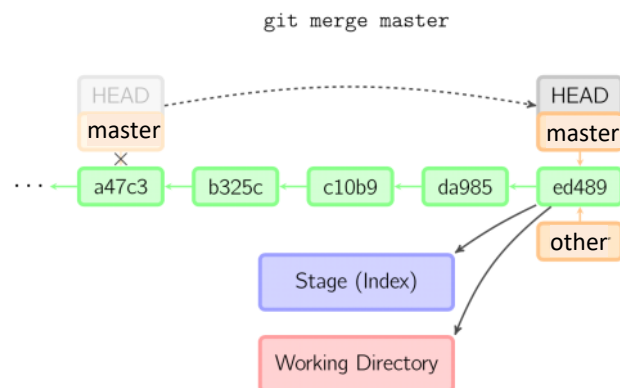
## Estrategias: fast-forward

miércoles, 24 de enero de 2018 23:03

Una rama se origina en otra que a partir de ese momento no añade ningún commit

Al hacer merge desde la rama no evolucionada, cuyo último commit es un ancestro del commit objeto de la fusión, lo que realmente sucede es una simple operación de avance linear del puntero hasta el de la rama fusionada

```
$ git checkout master
$ git merge other
```



### Ejemplo

```
* 4c794ae (HEAD -> fix-fecha) Retrasa la apertura de la web
| * 214960d (feature-newstyle) Retoca el estilo de la página
|/
* 433f33d (master) Revert "Embellece el sitio web"
```

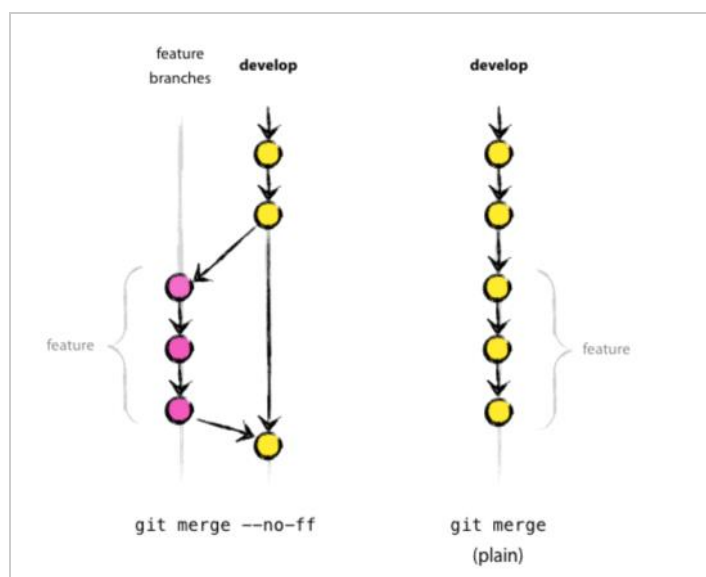
```
$ git checkout master
$ git merge fix-fecha
```

rama de destino

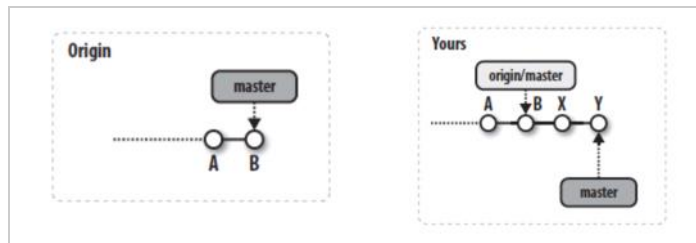
rama que se va a fusionar (origen)

```
* 4c794ae (HEAD -> master, fix-fecha) Retrasa la apertura de la web
| * 214960d (feature-newstyle) Retoca el estilo de la página
|/
* 433f33d Revert "Embellece el sitio web"
```

Aunque el fast-forward sea posible, puede evitarse mediante el modificador `--no-ff`



El mismo proceso se reproduce cuando se trata de una rama remota



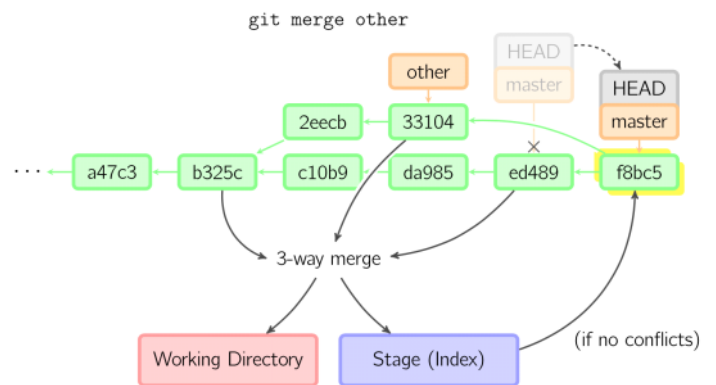
El fast forward ha ocurrido en origin desde B a X

## Estrategia recurrente

domingo, 7 de enero de 2018 11:37

Cuando el *fast forward* no es posible, por defecto se realiza una mezcla "*recursive*", la cual básicamente toma el *commit* actual (ed489 debajo), el otro *commit* (33104), y su ancestro común (b325c), y realiza una mezcla de tres vías. El resultado se guarda al directorio de trabajo y al *stage*. Luego, si no hay conflicto o el que haya se resuelve, ocurre un *commit*, con un padre adicional (33104) para el nuevo *commit*.

```
$ git checkout master
$ git merge other
```

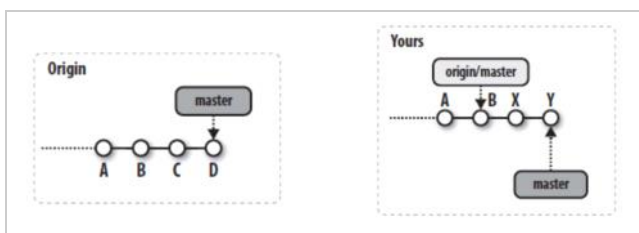


```
* 4c794ae (HEAD -> master, fix-fecha) Retrasa la apertura de la web
| * 214960d (feature-newstyle) Retoca el estilo de la página
|/
* 433f33d Revert "Embellege el sitio web"
```

```
$ git checkout master
$ git merge feature-newstyle
```

```
* 1b660bf (HEAD -> master) Merge branch 'feature-newstyle'
| \
| * 214960d (feature-newstyle) Retoca el estilo de la página
| * | 4c794ae (fix-fecha) Retrasa la apertura de la web
|/
* 433f33d Revert "Embellege el sitio web"
```

If another developer has pushed some changes (C, D) you can't push with fast forward.



You should merge your changes first

# Conflictos

martes, 9 de enero de 2018 0:46

Git te advierte sobre el conflicto:

```
israelalcazar@Mac-Lamaro:~/dev/git/examples2 (master) $ git merge newFeature
Auto-merging README3.TXT
CONFLICT (content): Merge conflict in README3.TXT
Automatic merge failed; fix conflicts and then commit the result.
```

Y marca el archivo en el que se ha producido:

```
1 <<<<<< HEAD
2 readme2 106
3 =====
4 readmeNEWF 107
5 >>>>>> newFeature
```

Para la localización de los archivos que tienen conflictos, Git realiza un seguimiento de los archivos problemáticos marcando cada uno en el índice como conflictivo o no

Puedes usar uno de estos comandos:

```
$ git status
$ git diff
```

Si los conflictos son demasiado numerosos o en otra circunstancia se puede abortar el proceso de fusión

```
$ git reset --hard ORIG_HEAD
```

← Puntero con el HEAD previo al inicio del proceso


# Herramientas de merge

miércoles, 24 de enero de 2018 23:06

<https://gist.github.com/karenyyng/f19ff75c60f18b4b8149>

## GUI mergetool editors

- gvimdiff (linux) <https://linux.die.net/man/1/gvimdiff>
- kdiff3 <http://kdiff3.sourceforge.net/>
- meld <http://meldmerge.org/>
- tortoisemerge



## KDiff3 - Home

[News](#)  
[Features](#)  
[Screenshots](#)  
[Project At Sourceforge](#)  
[Licence: GPL](#)  
[Download](#)  
[Documentation](#)  
[Questions and Answers](#)  
[Abstract \(PDF\)](#)  
[KDiff3 on Debian](#)  
[Donations](#)  
[Links](#)

Current version: 0.9.98 (2014-07-04)

Author: [Joachim Eibl](#)

Please write me your suggestions for KDiff3. ([Tracker](#), [Mailinglist](#))



KDiff3 is a diff and merge program that

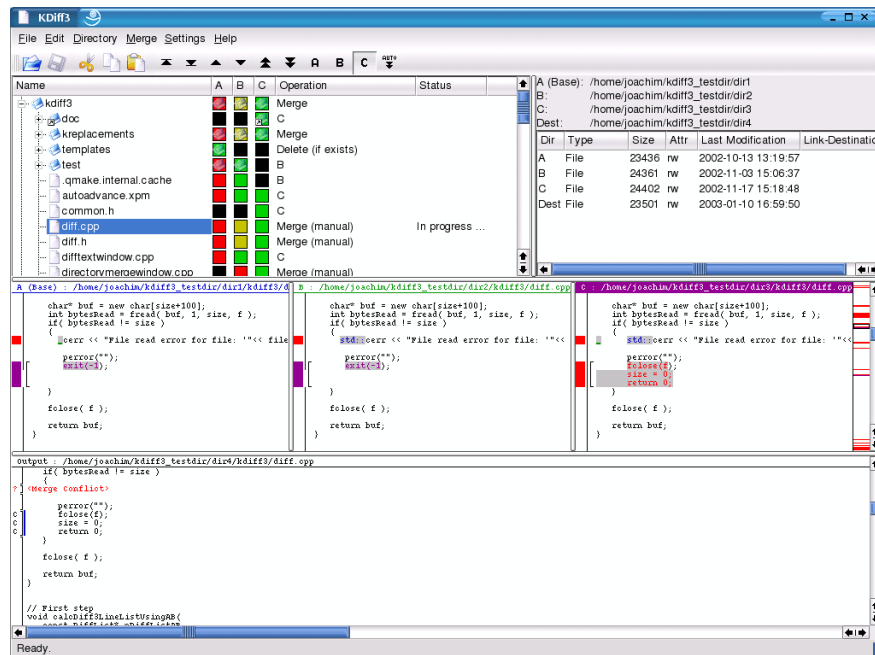
- compares or merges two or three text input files or directories,
- shows the differences line by line and character by character (!),
- provides an automatic merge-facility and
- an integrated editor for comfortable solving of merge-conflicts,
- supports Unicode, UTF-8 and other codecs, autodetection via byte-order-mark "BOM"
- supports KIO on KDE (allows accessing ftp, sftp, fish, smb etc.),
- Printing of differences,
- Manual alignment of lines,
- Automatic merging of version control history (\$Log\$),
- and has an intuitive graphical user interface.

- Windows-Explorer integration Diff-Ext-for-KDiff3 - shell extension included in installer (originally by Sergey Zorin: see also [Diff Ext](#))
- KDE-Konqueror/Dolphin service menu plugin
- Simplified integration with IBM-Rational-Clearcase for Windows ([Details](#)).
- Read what else is special in a short [abstract \(PDF\)](#).

Supported platforms:

- KDE4
- Any Un\*x that is supported by the Qt-libs from [qt-project.org](#),
- MS-Windows.
- Apple Mac OSX binary available ([0.9.98](#))

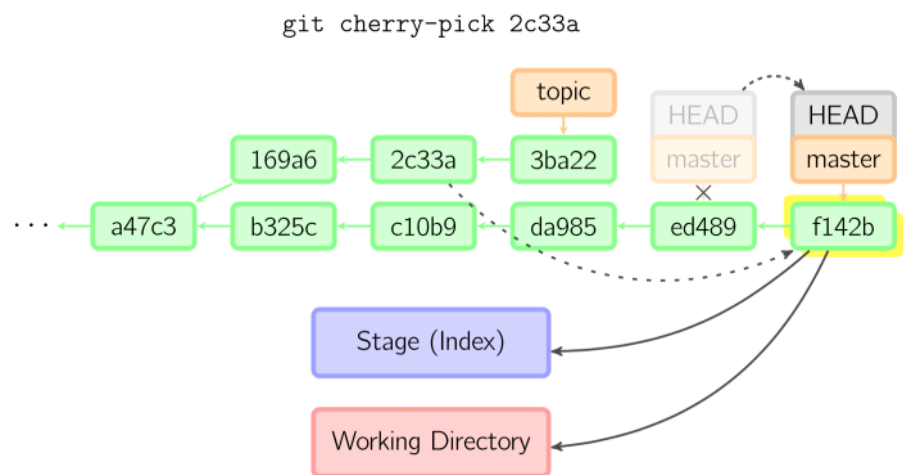




# Cherry Pick

miércoles, 24 de enero de 2018 23:10

El comando cherry-pick "copia" un *commit*, creando un nuevo *commit* en el *branch* actual con el mismo mensaje y *patch* que otro *commit*.



<https://frontendlabs.io/3084--aplicar-un-commit-otra-rama-con-git-cherry-pick>



# Tags

miércoles, 3 de enero de 2018

13:34

## Tags vs Branches

A tag is meant to be a static name that does not change or move over time. Once applied, you should leave it alone.

A branch is dynamic and moves with each commit you make. The branch name is designed to follow your continuing development.

You can give a branch and a tag the same name.

If you do, you will have to use their full ref names to distinguish them.

For example, you could use `refs/tags/v1.0` and `refs/heads/v1.0`

## Comandos

```
$ git tag : List tags
$ git tag -a <name> -m "text to ...": Create a tag
$ git show <name>
$ git push <repo> --tags: Sends all tags
$ git push <repo> <tag-name>: Sends the tag
```

## Types

### Lightweight

Like a branch that doesn't change. A pointer to a specific commit

### Annotated

Are stored as full objects in the Git database. They are checksummed; contain the tagger name, email and date. Can be signed and verified.

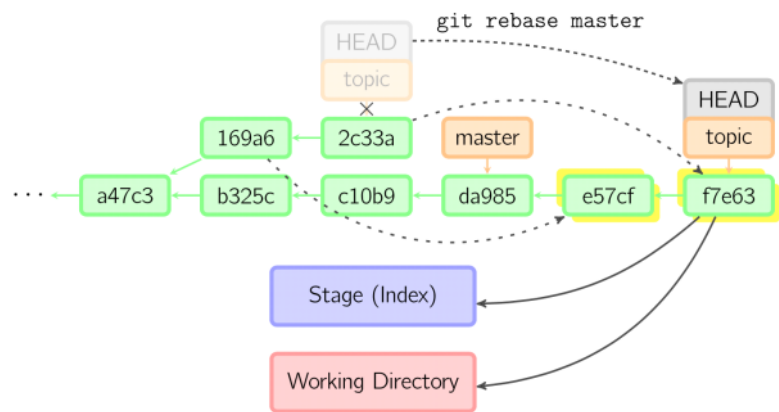
# Rebase

jueves, 25 de enero de 2018 0:16

Un rebase es una alternativa al *merge* para combinar múltiples ramas.

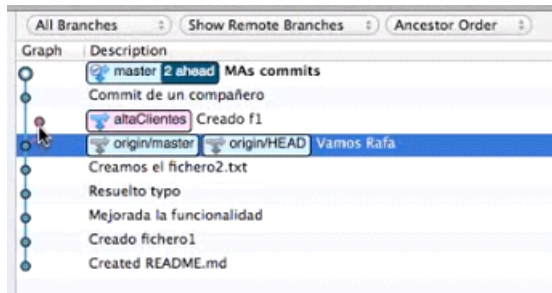
un *merge* crea un solo *commit* con dos padres, dejando una historia no lineal, un rebase reproduce los *commits* del *branch* actual en otro, dejando una historia lineal.

En esencia, esta es una forma automatizada de realizar muchas cherry-pick de una vez.



## Ejemplo

domingo, 7 de enero de 2018 12:47

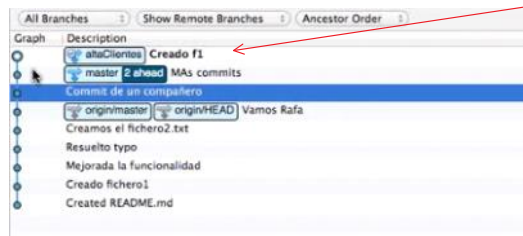


La rama alta clientes esta desactualizada respecto a la rama principal

Es posible cambiar el punto base de una rama

```
$ git checkout altaClientes  
$ git rebase master
```

```
~/dev/git2/git.11.8.rebase.branch (altaClientes) $ git rebase master  
First, rewinding head to replay your work on top of it..  
Applying: Creado f1  
~/dev/git2/git.11.8.rebase.branch (altaClientes) $
```



altaClientes actualizada y por delante de master

Git aplica uno tras otro cada uno de los *commits* de la rama principal, siempre que no haya conflictos.

sí aparece un conflicto, se resuelve y después se ejecuta

```
$ git rebase --continue
```

si la acumulación de conflicto es grande puede cancelarse el rebasa

```
$ git rebase --abort
```

Al trabajar en una rama de funcionalidad es una buena práctica hacer un rebase de la principal, al menos al inicio y al final del día

# Rebase interactivo

miércoles, 24 de enero de 2018 23:02

# Workflows

miércoles, 3 de enero de 2018

13:36

## Modelos de organización

Forking workflow

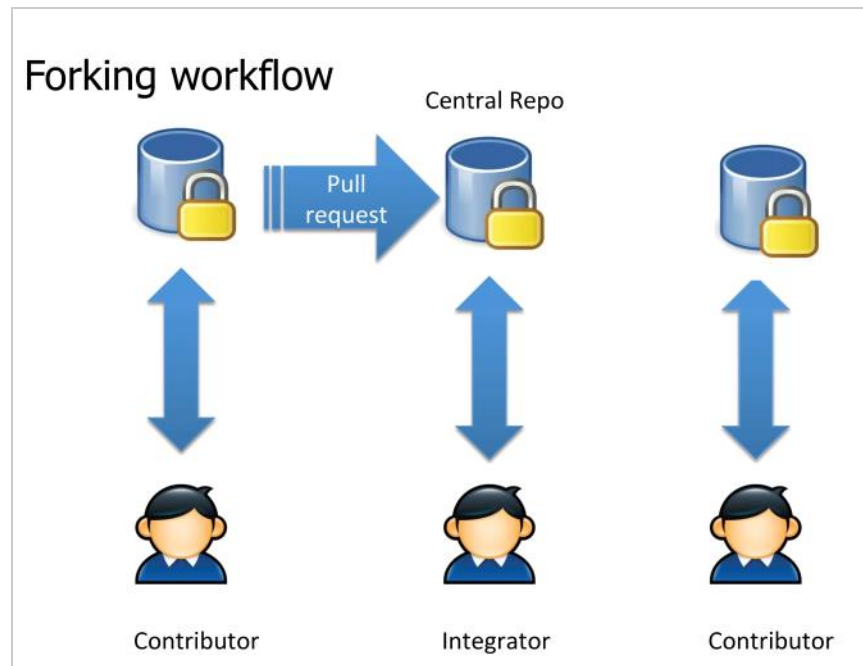
Centralized workflow

## Flujos de trabajo en Git

- 1) Two main branches: Dev & Master
- 2) One feature / one branch (local)
- 3) Every bug is fixed in an isolated branch

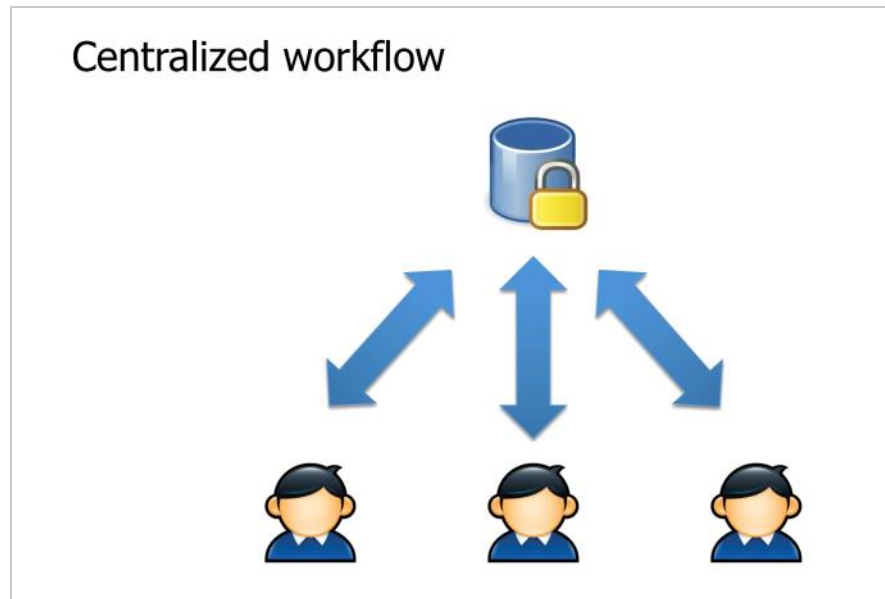
# Forking workflow

martes, 23 de enero de 2018 1:16



# Centralized workflow

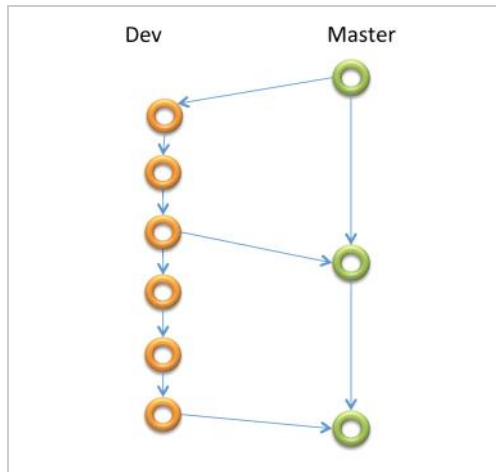
martes, 23 de enero de 2018 1:18



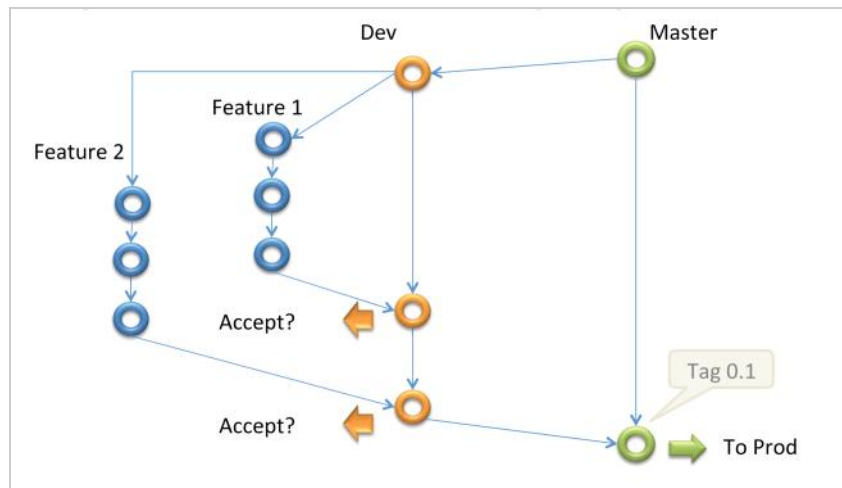
# Flujos de trabajo en Git

martes, 23 de enero de 2018 1:21

1) Two main branches:  
Dev & Master



2) One feature /  
one branch (local)



3) Every bug is fixed in  
an isolated branch

