

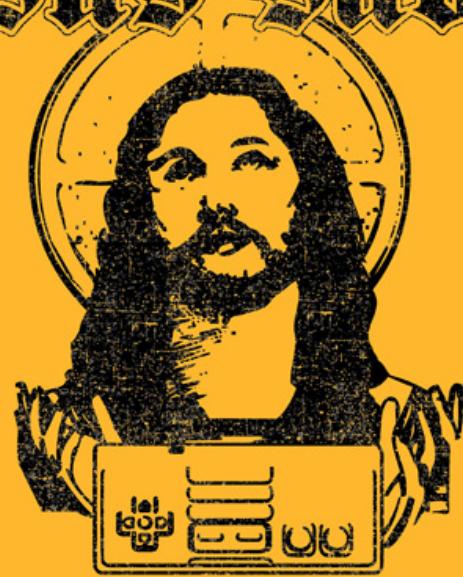
# Git, GitHub y SourceTree



Copyright © AGBO . Todos los derechos reservados

# ¿Qué es Git?

jesus-sabes



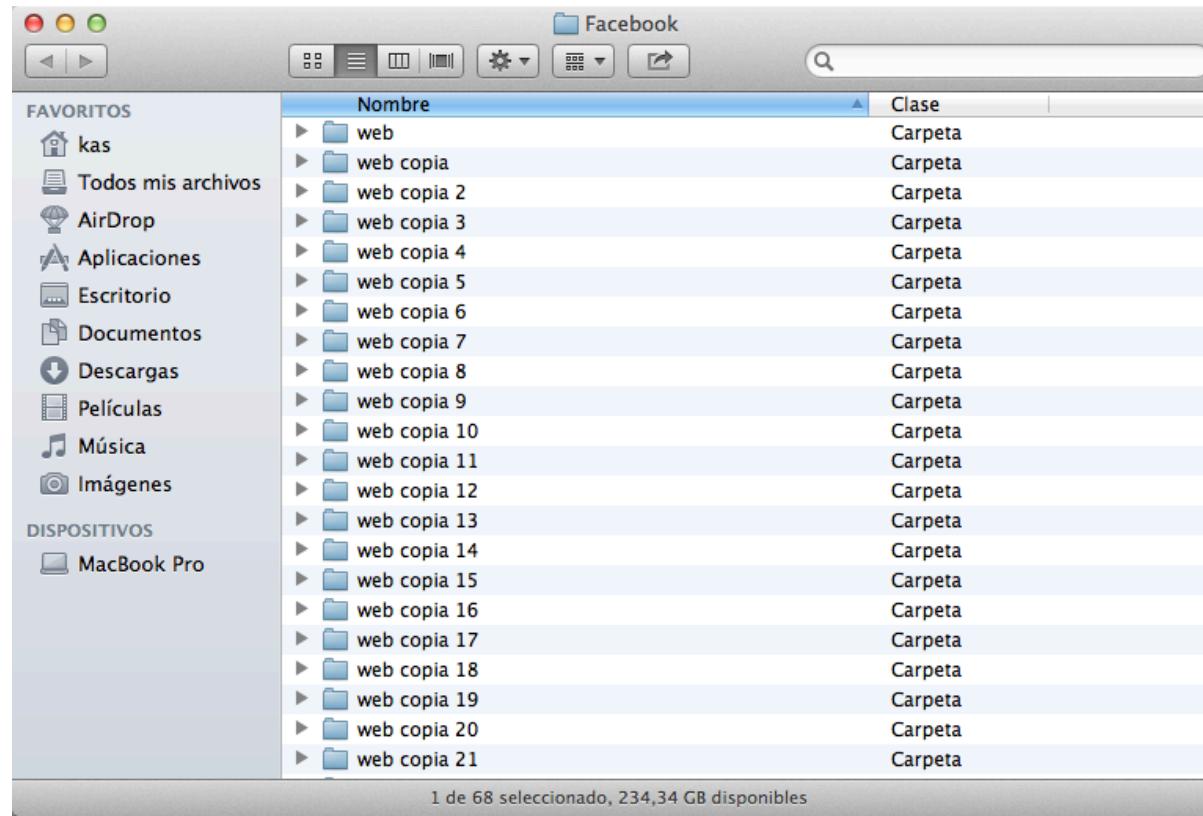
after he passes each level

«Jesús guarda la partida al pasarse cada nivel»

# Ok, pero ¿qué es Git?

- Un **sistema** de control de versiones
- Un software de apoyo para desarrollar
- Es distribuido, no centralizado

# ¿Para qué sirve? ¿Qué nos aporta?



# **¿Para qué sirve? ¿Qué nos aporta?**

- Control sobre la evolución de un proyecto
- Control sobre el desarrollo colaborativo
- Desarrollo en paralelo de funcionalidades
- Estructuración y mantenimiento de versiones

# ¿Qué vamos a aprender?

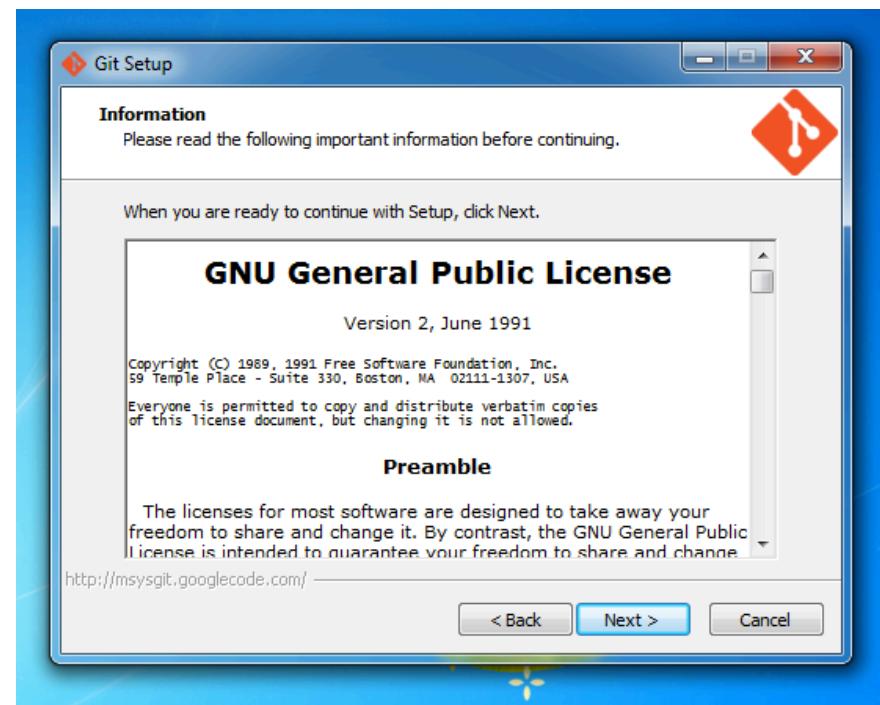
- Instalación de las herramientas en los 3 sistemas
- El origen del control de versiones: diff y patch
- Intro a Git (el gráfico)
- Un montón de comandos git: checkout, add, commit, diff, reflog, reset, clone
- Cómo trabajar con ramas y hacer merging entre ellas
- Cómo solucionar conflictos de versiones
- Trabajar con otros geeks en GitHub
- Extras para divertirnos un poco más

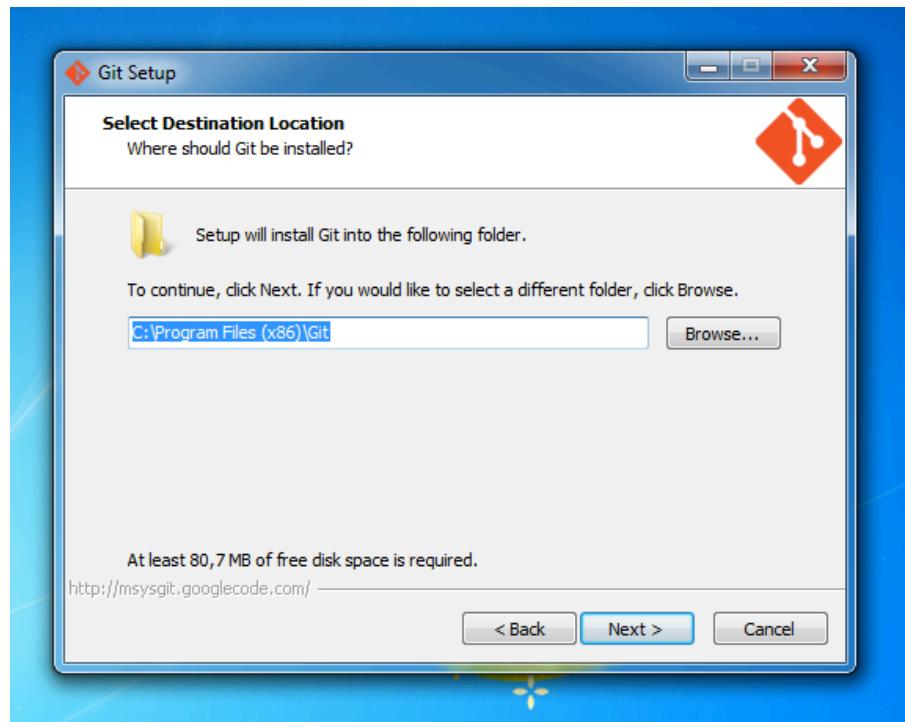
# **Instalando las herramientas**

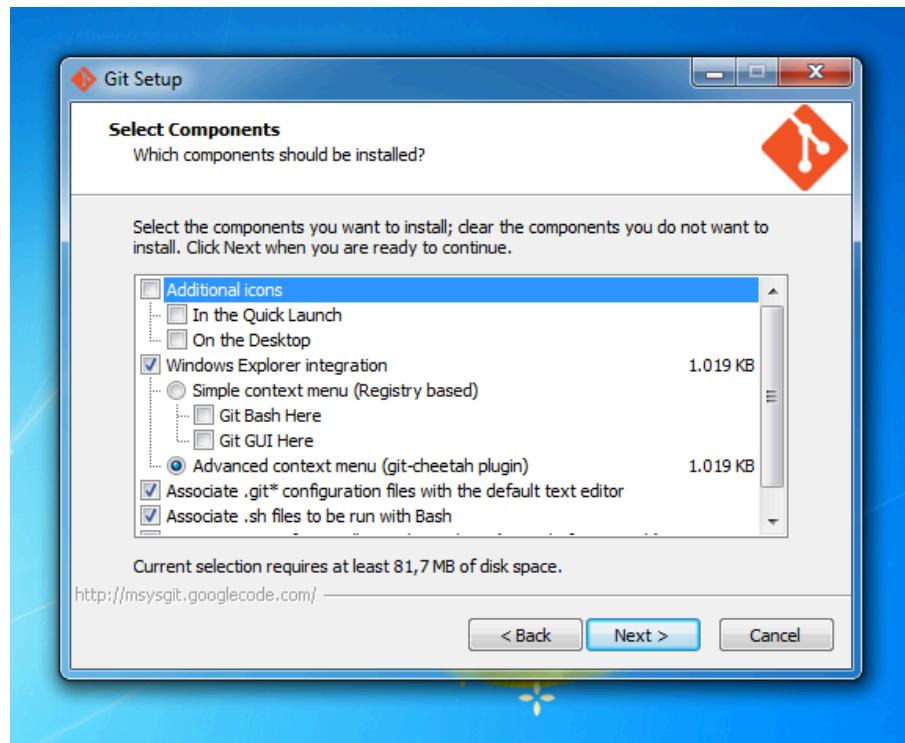
# Instalando Git en Windows

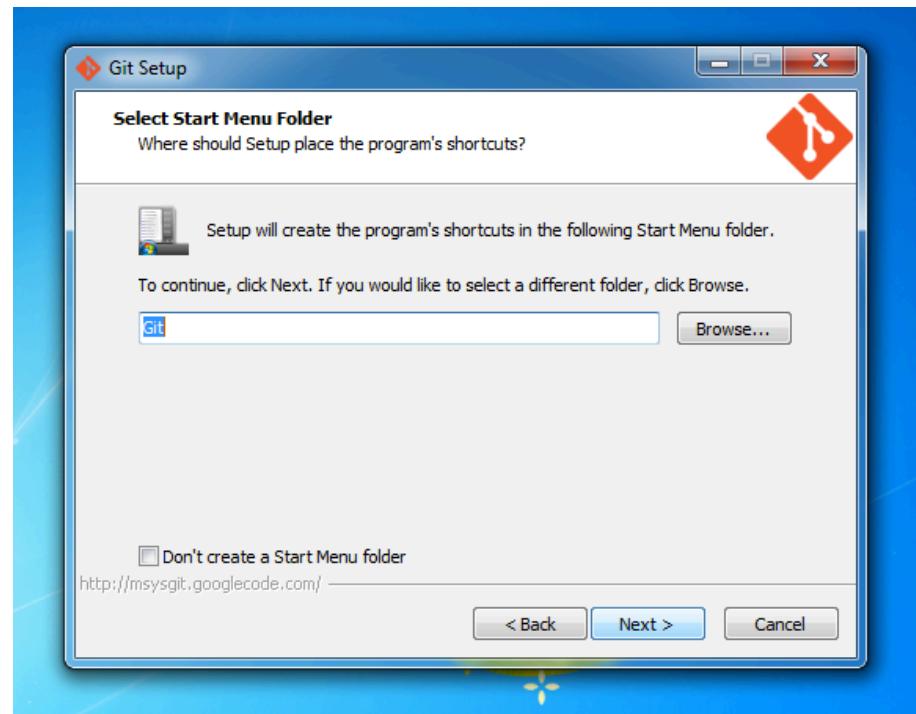


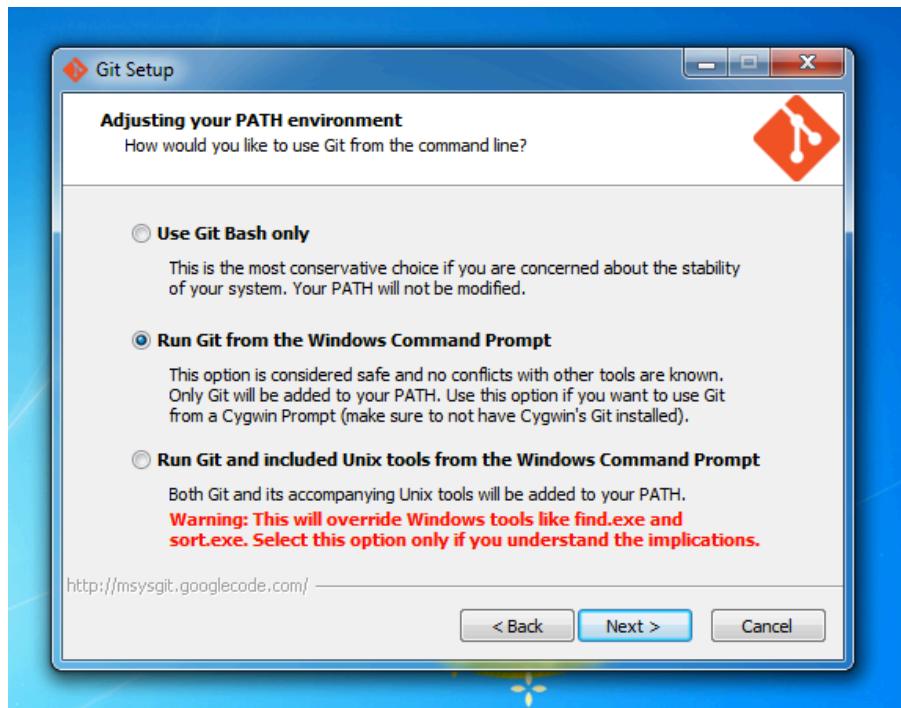
Instalador gráfico <http://git-scm.com/download/win>



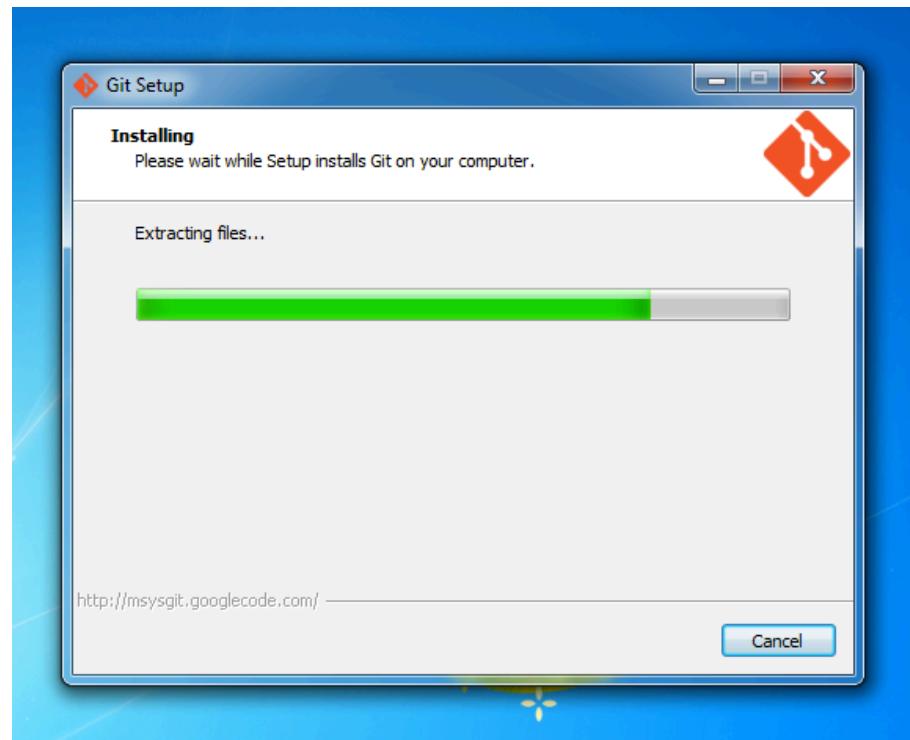






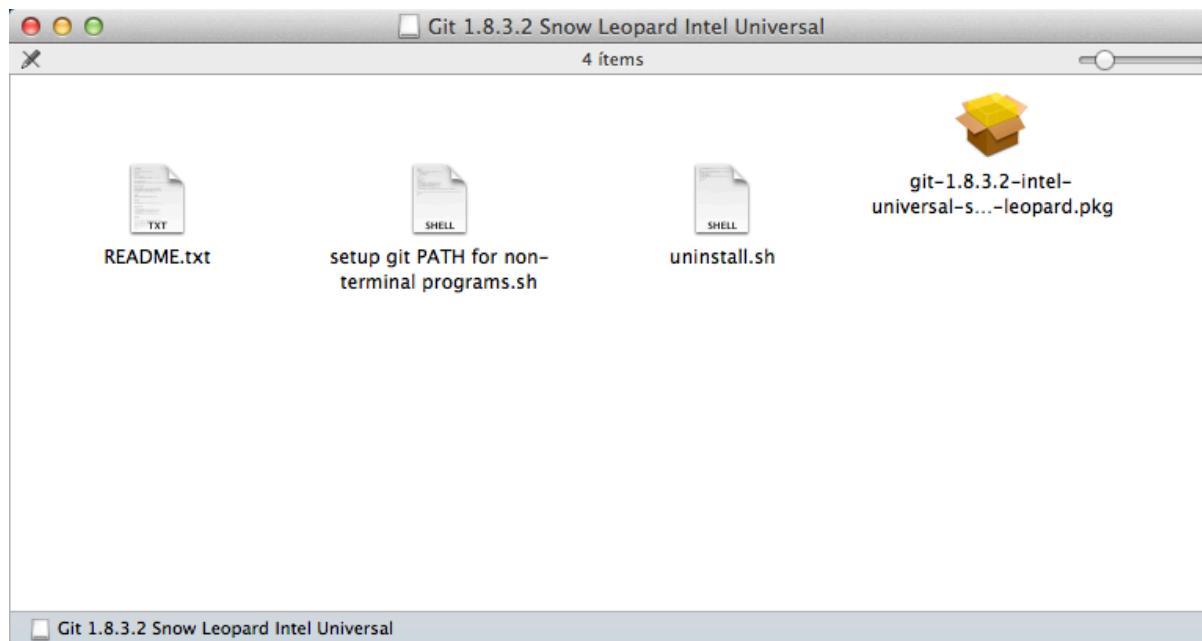






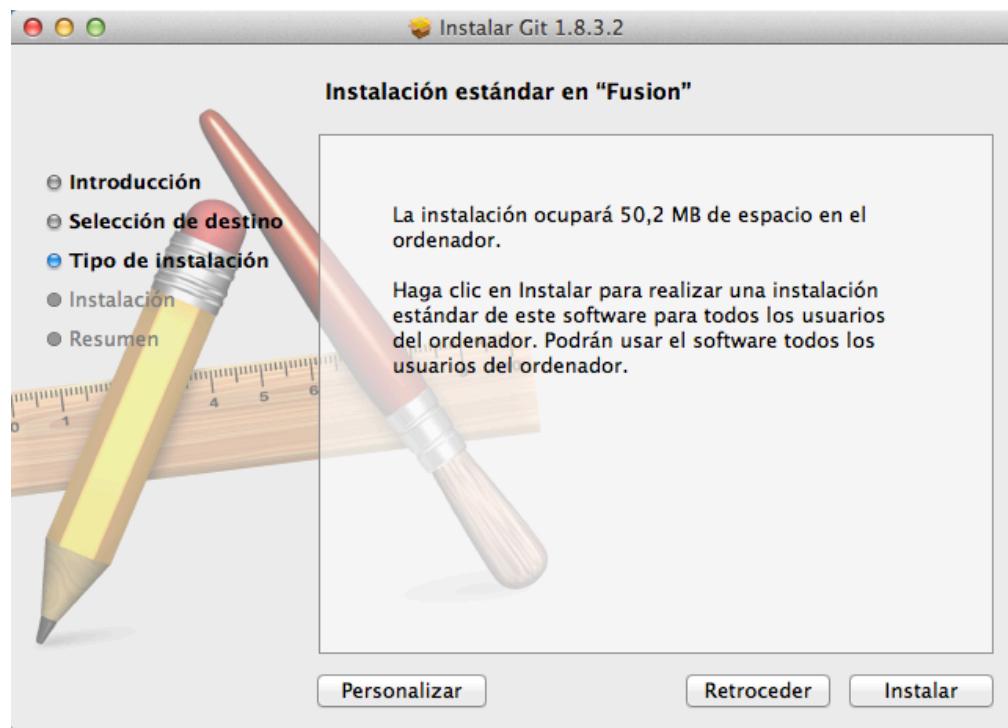


# Instalando Git en Mac

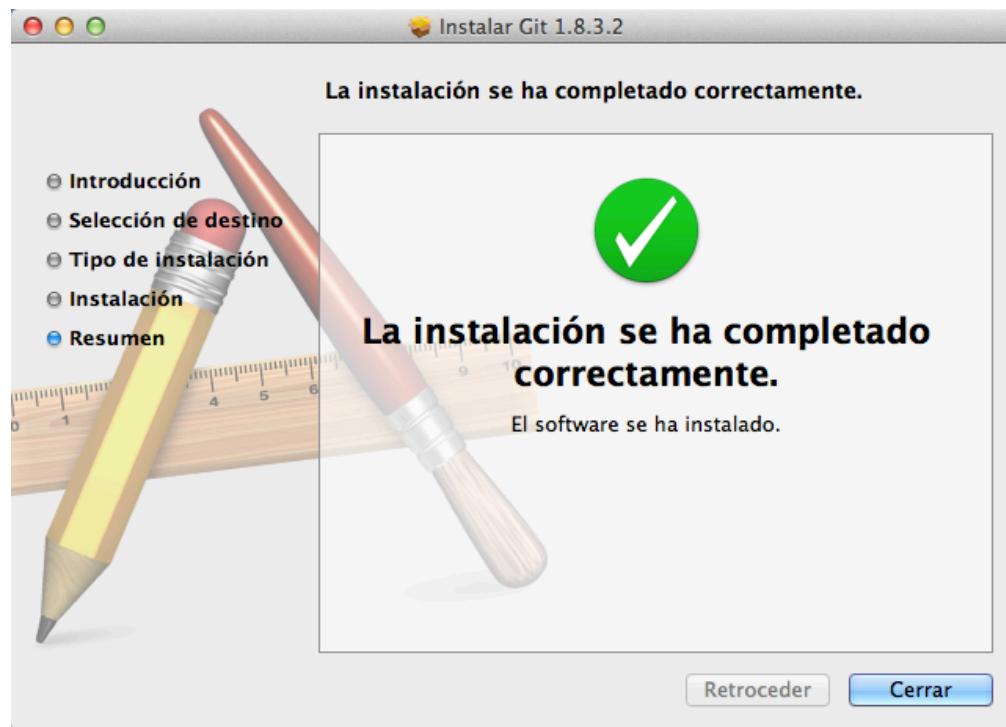


Con el instalador gráfico <http://git-scm.com/download/mac>









Si tienes MacPorts (<http://www.macports.org>)

```
$ sudo port install git-core +svn +doc +bash_completion +gitweb
```

# Instalando Git en Linux

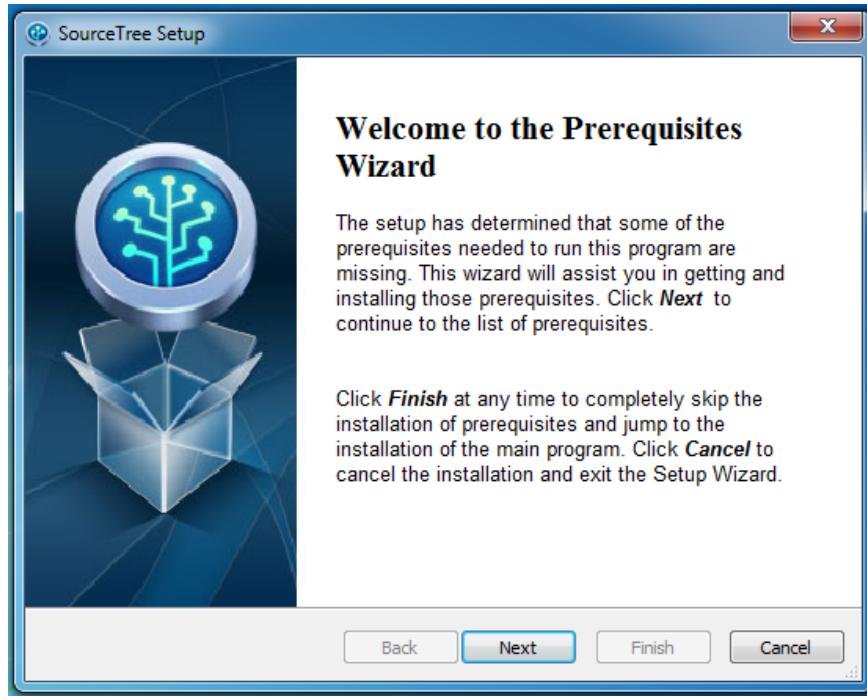
Si estás en Debian/Ubuntu

```
$ apt-get install git
```

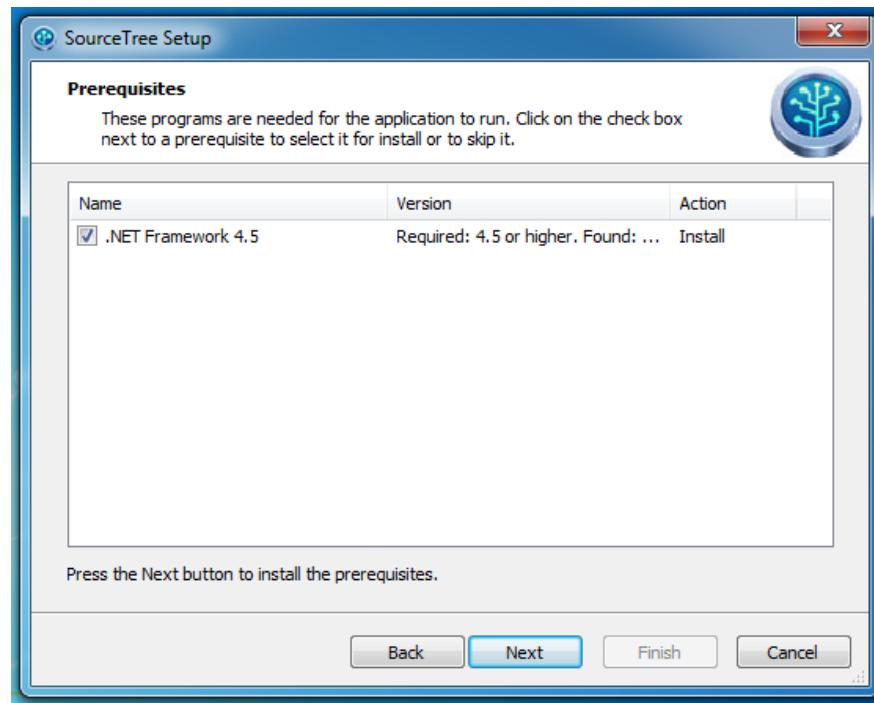
Si estás en Fedora

```
$ yum install git-core
```

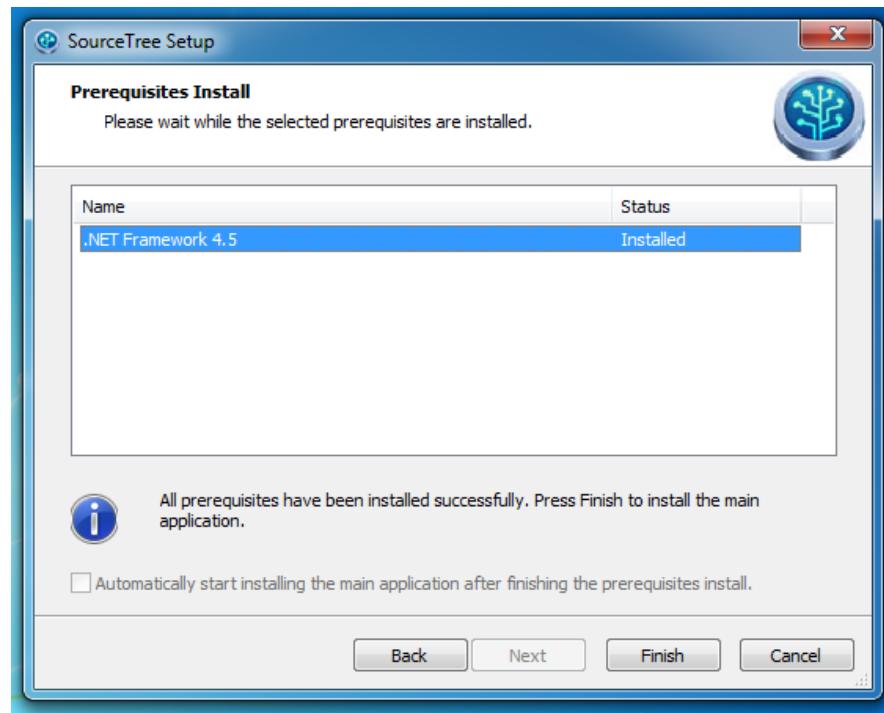
# Instalando SourceTree en Windows



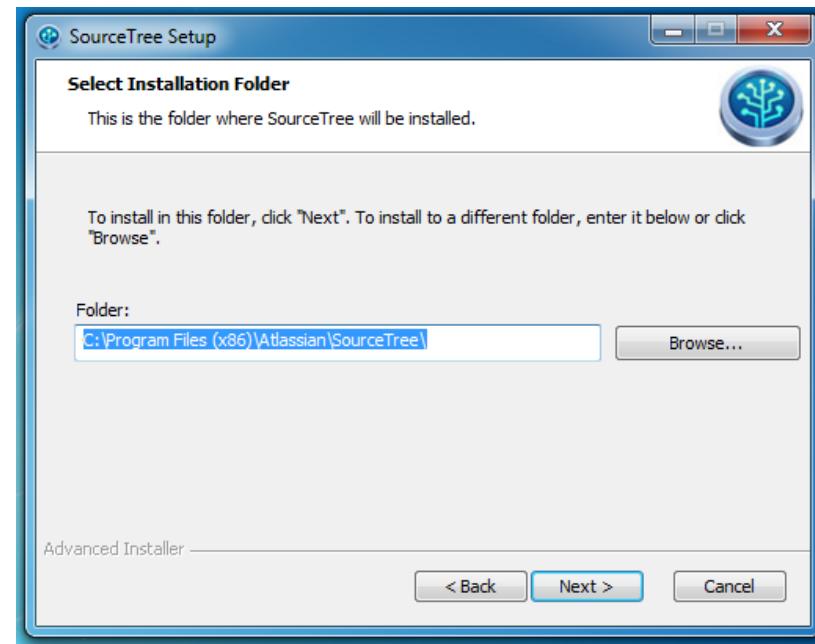
Instalador gráfico <http://www.sourcetreeapp.com/>

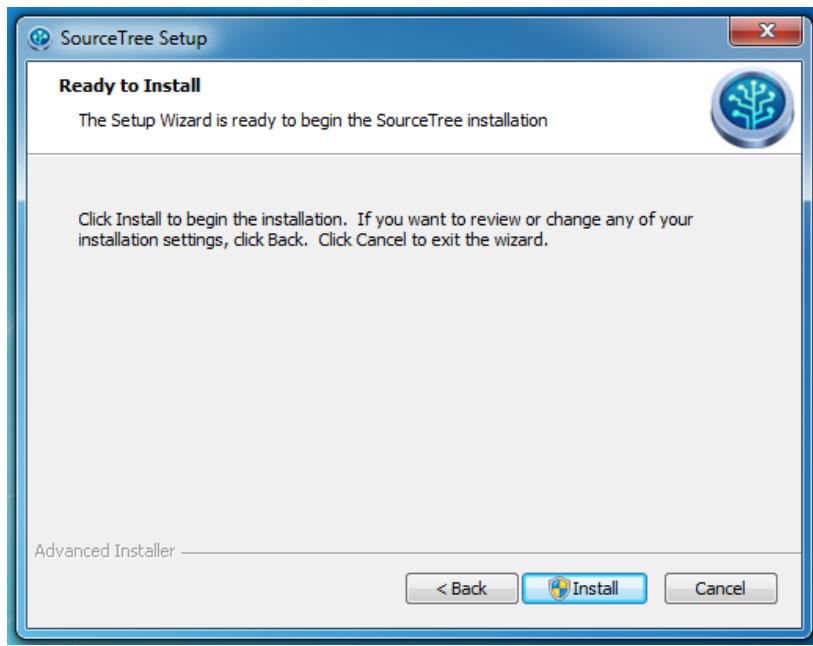


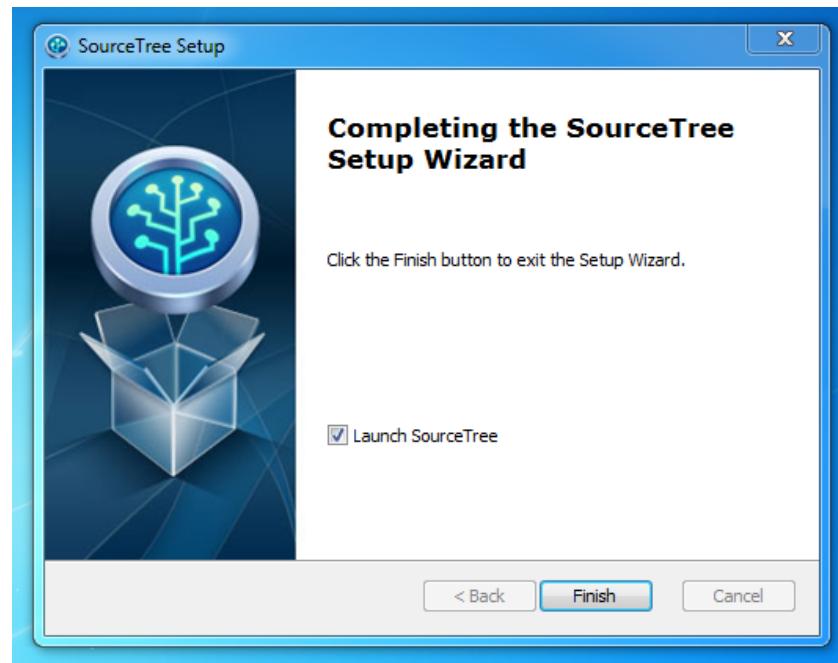


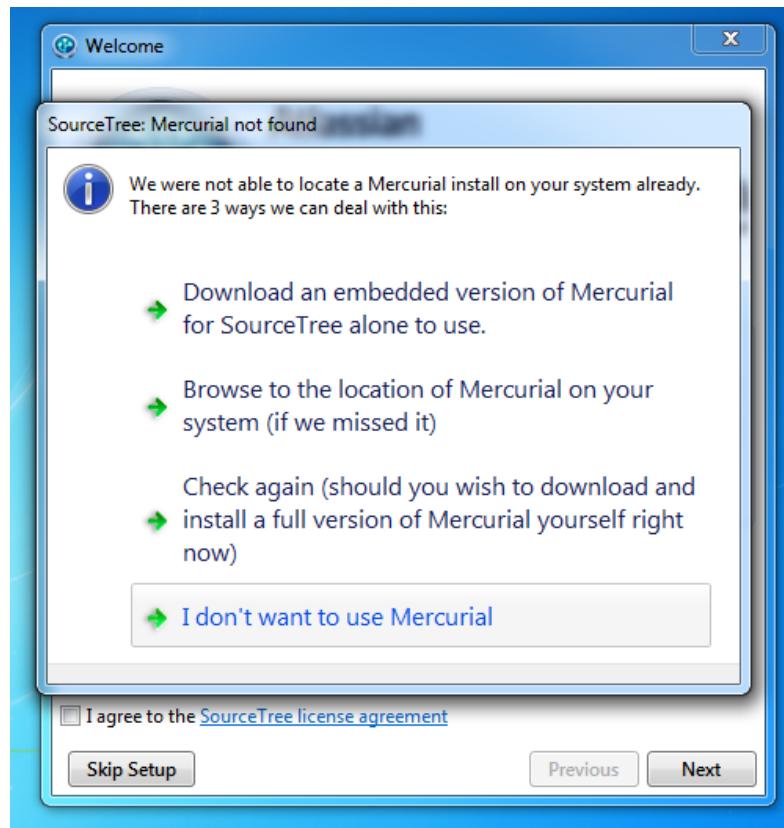






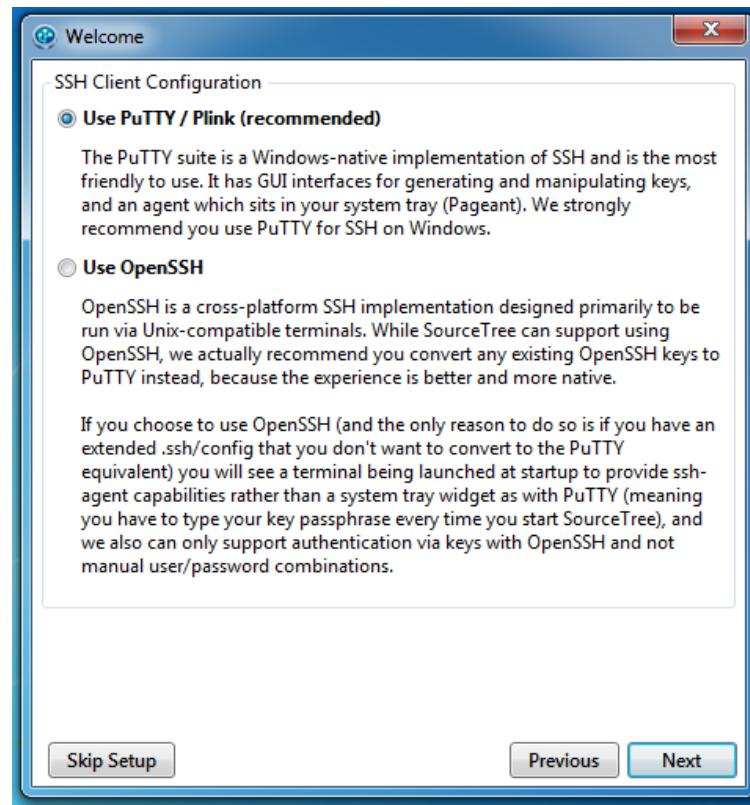


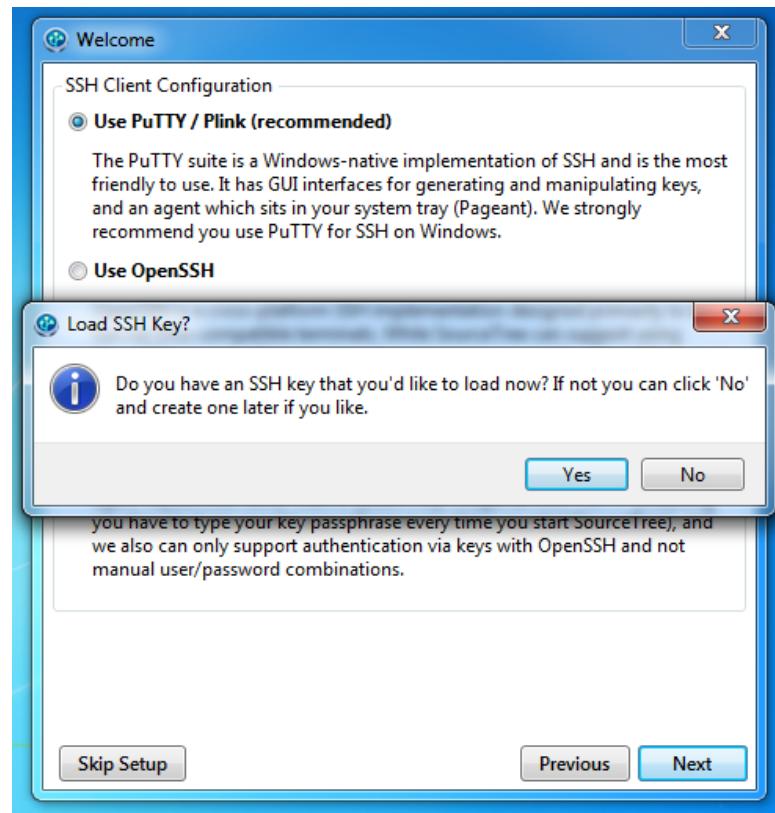


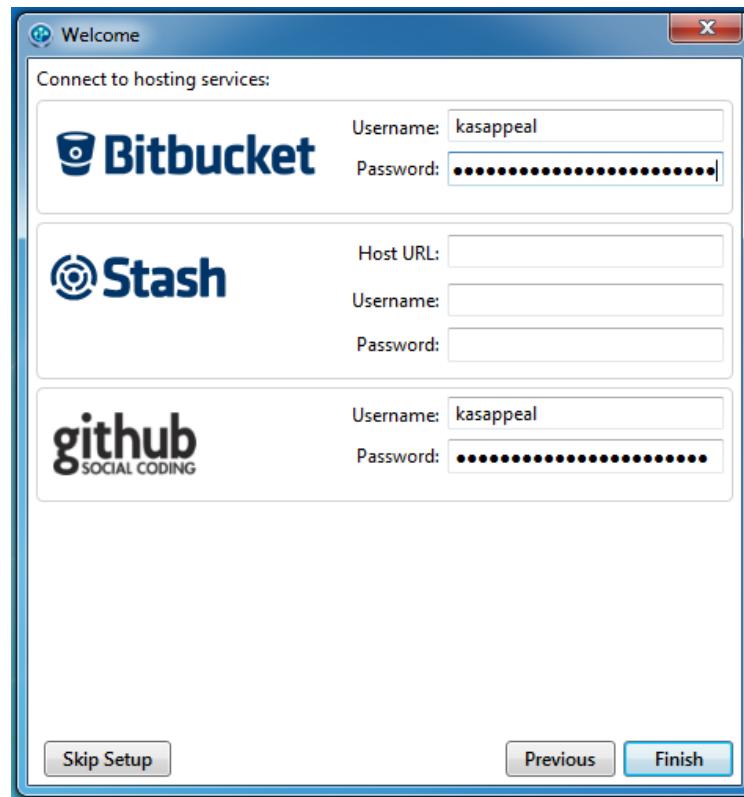








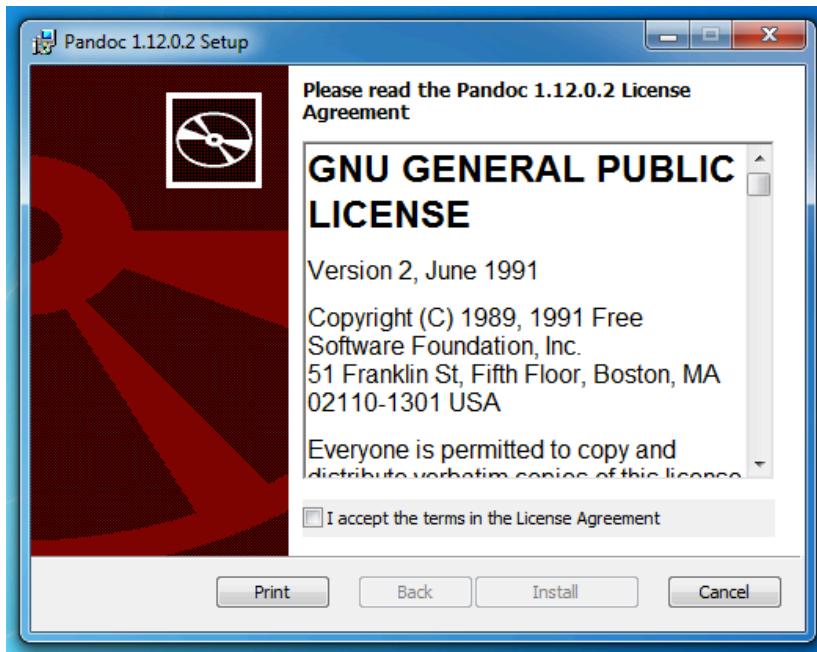




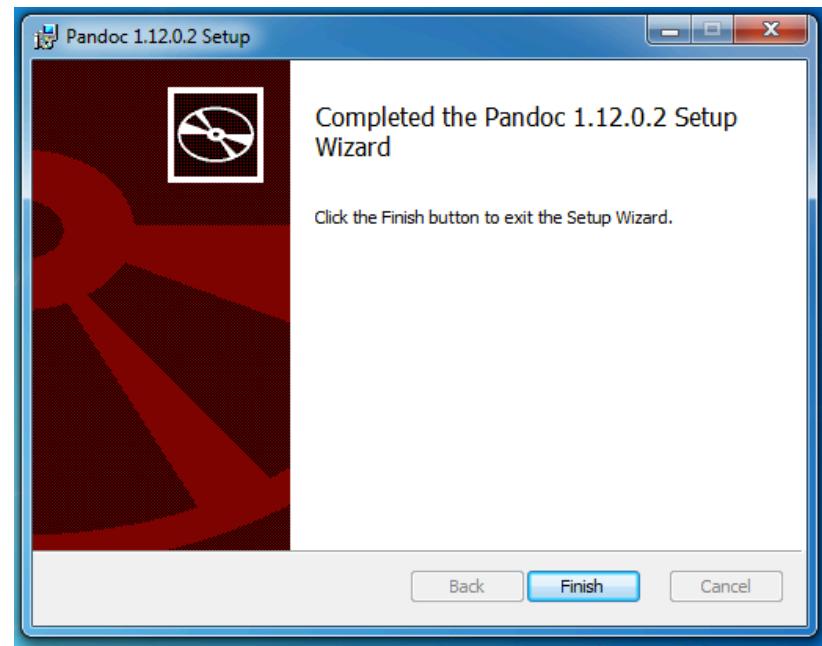
# **Instalando SourceTree en Mac**

# **Instalando SourceTree en Linux**

# Instalando Pandoc en Windows



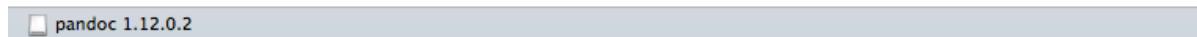
Instalador gráfico  
<https://code.google.com/p/pandoc/downloads/list>



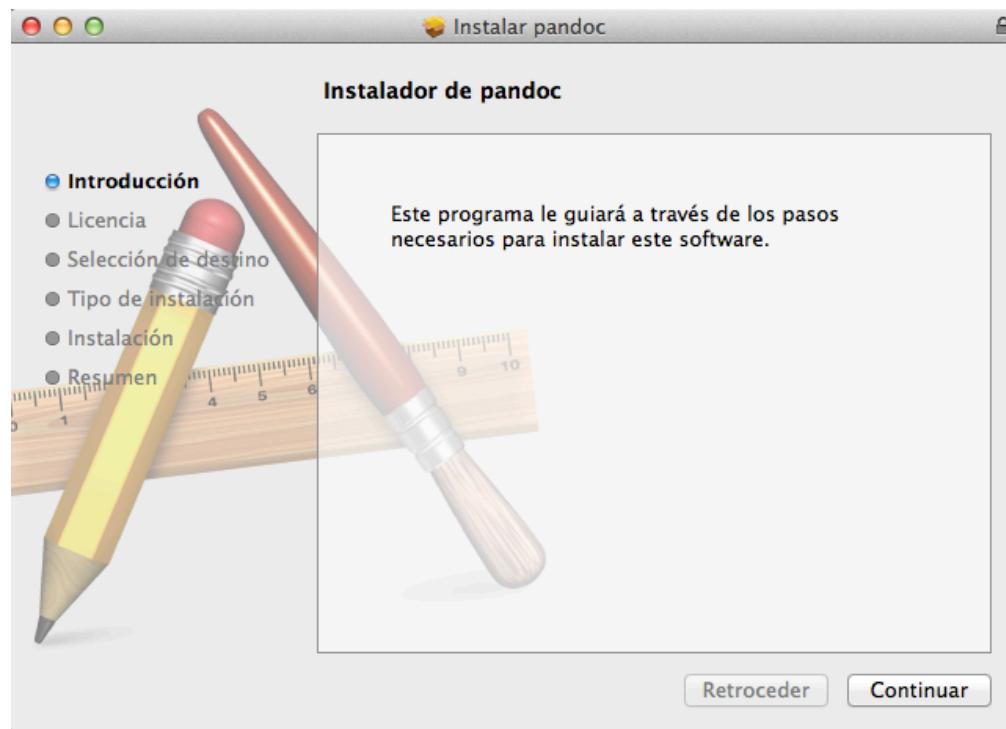
# Instalando Pandoc en Mac

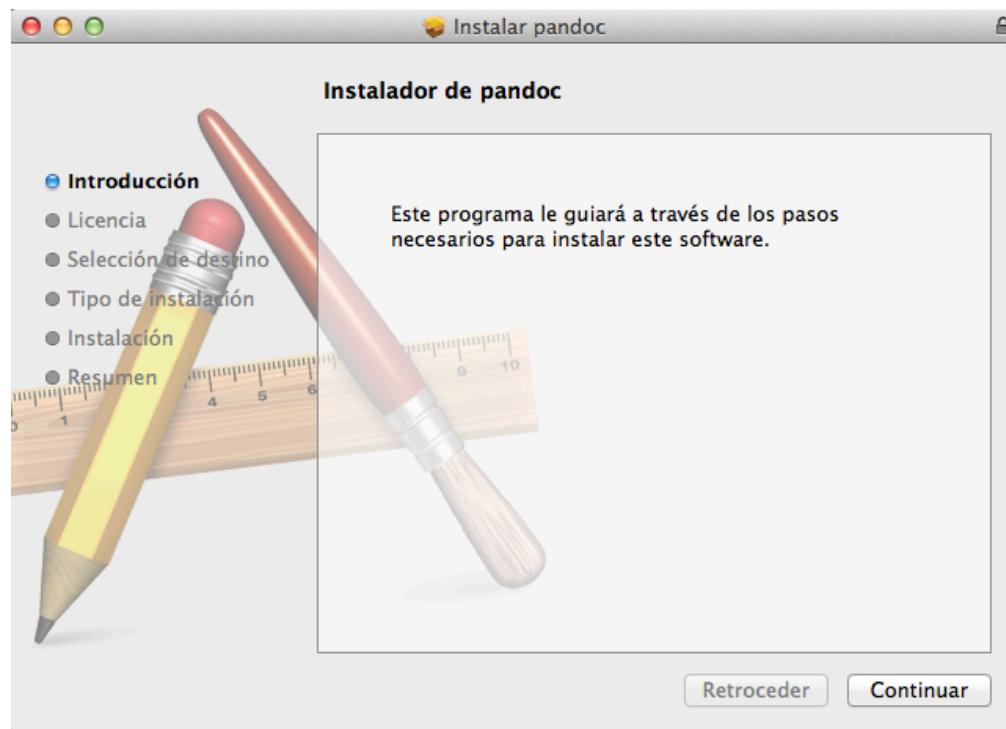


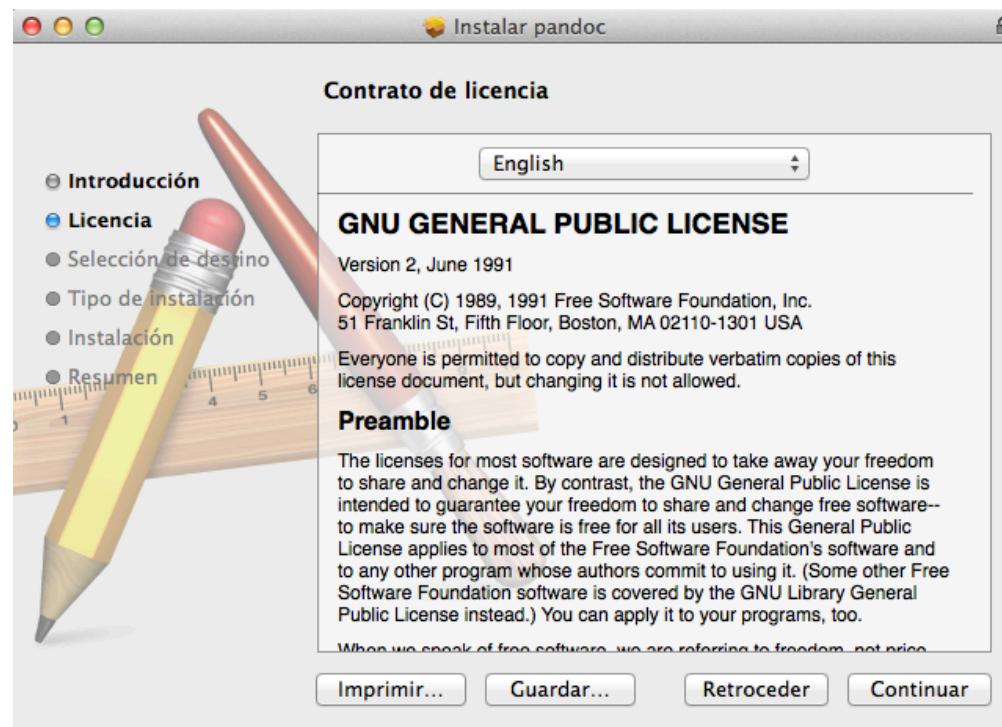
pandoc-1.12.0.2.pkg

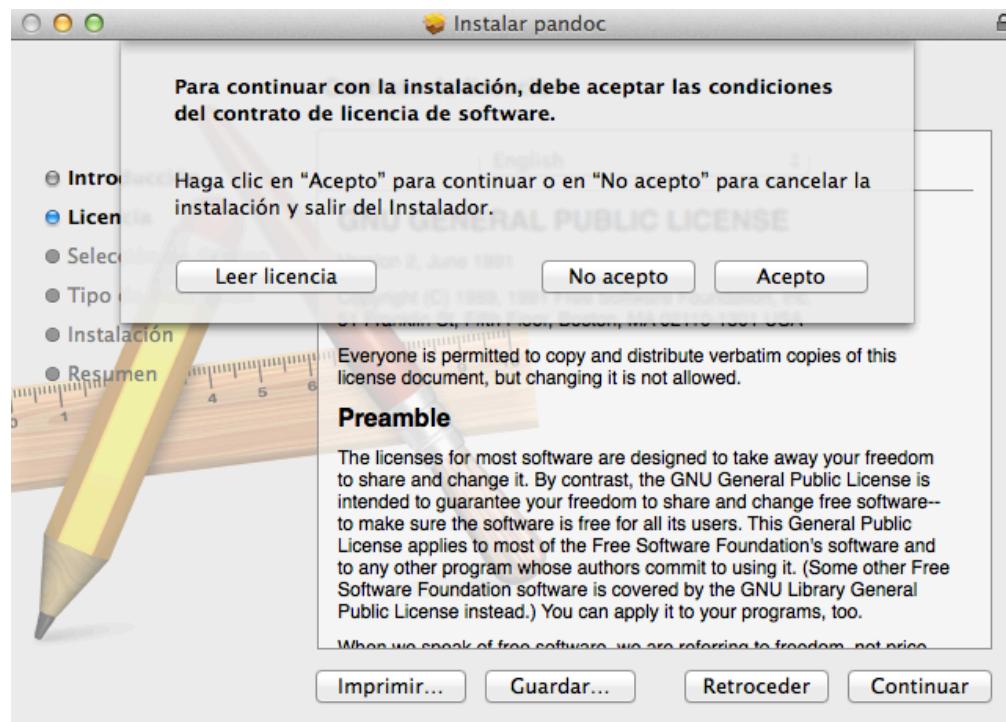


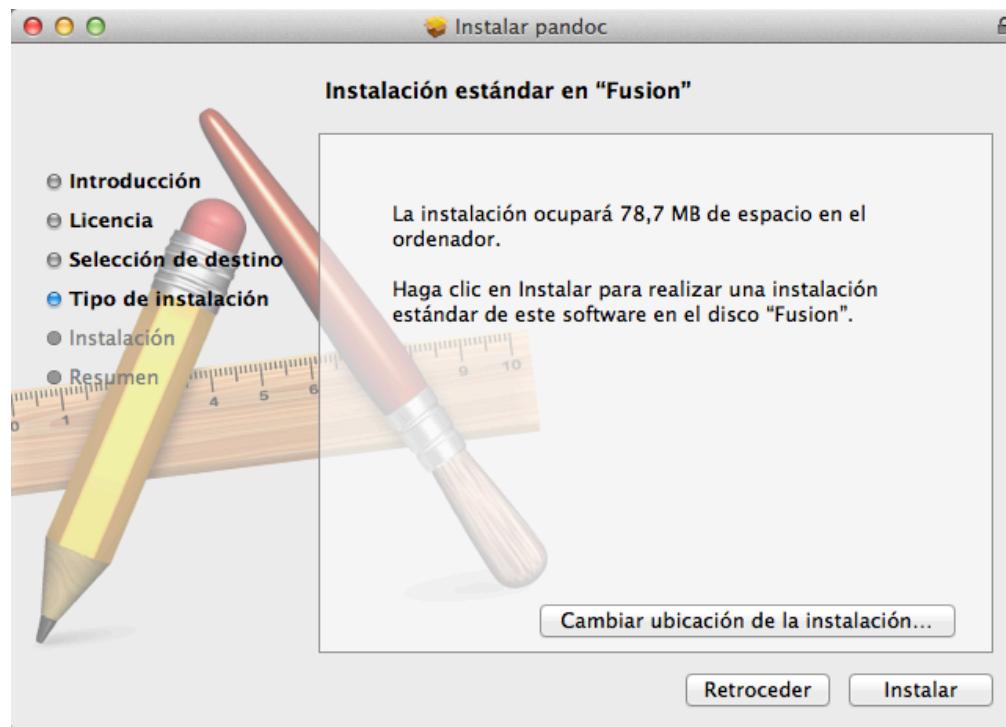
Instalador gráfico  
<https://code.google.com/p/pandoc/downloads/list>











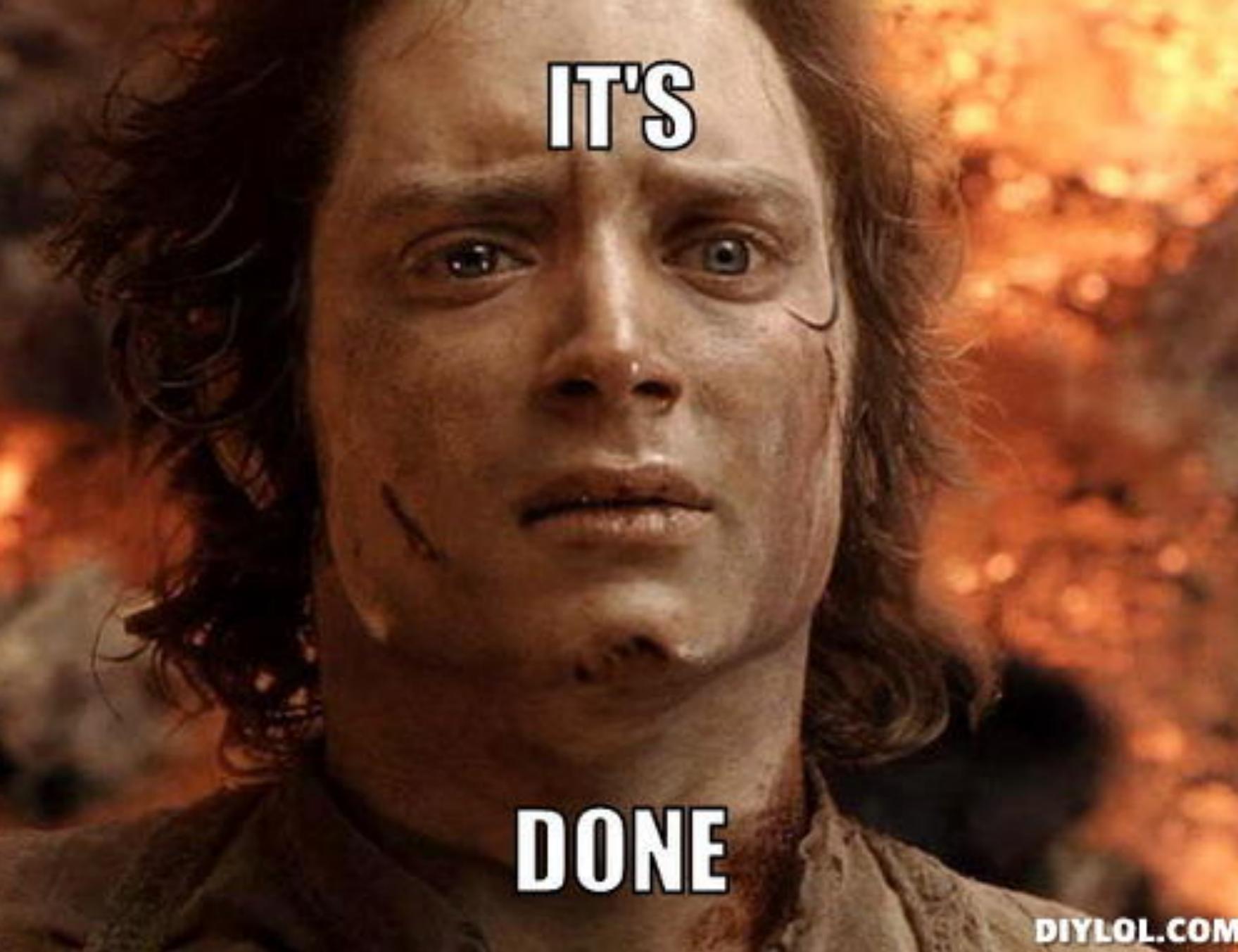
# Instalando Pandoc en Linux

Si estás en Debian/Ubuntu

```
$ apt-get install pandoc
```

Si estás en Fedora

```
$ yum install pandoc
```



**IT'S**

**DONE**

**DIYLOL.COM**

# Configurando git

```
# Cambiar la configuración global  
$ git config --global user.name "Homer J. Simpson"  
$ git config --global user.email "beer@simpsons.com"
```

```
# Cambiar los colores de la consola  
$ git config --global color.ui true  
# Cambiar el editor por defecto a nano  
$ git config --global core.editor "nano"  
# Cambiar el editor por defecto al Bloc de notas (sólo Windows)  
$ git config --global core.editor "notepad"
```

Esta configuración puede ser establecida para cada proyecto

**Al principio todo  
era diff y patch**

# **El comando diff**

Sirve para encontrar diferencias entre archivos

## *celebrities.txt*

```
Bud Spencer
Larry Page
Robert De Niro
Luis de Guindos
Steve Jobs
Mr. Potato
Bill Gates
Mark Zuckerberg
```

## *geekebrities.txt*

```
Bash Spencer
Larry Page
Blogger De Niro
Luis de Windows
Steve Jobs
Bill Gates
Mark Zuckerberg
Flickr Casillas
```

```
$ diff celebrities.txt geekebrities.txt
```

```
1c1
< Bud Spencer
---
> Bash Spencer
3,4c3,4
< Robert De Niro
< Luis de Guindos
---
> Blogger De Niro
> Luis de Windows
6d5
< Mr. Potato
8a8
> Flickr Casillas
```

```
1c1 # Cambiar L1 de celebrities.txt por L1 de geekebrities.txt
< Bud Spencer
---
> Bash Spencer
3,4c3,4 # Cambiar L3-L4 de celebrities.txt por L3-L4 de geekebrities.txt
< Robert De Niro
< Luis de Guindos
---
> Blogger De Niro
> Luis de Windows
6d6 # Eliminar L6 de celebrities.txt
< Mr. Potato
8a8 # Añadir después de L8 de celebrities.txt, L8 de geekebrities.txt
> Flickr Casillas
```

# **El comando patch**

Actualiza un archivo basándose en la información de un *patch file*  
generado con *diff*

## Creamos el parche

```
$ diff celebrities.txt geekebrities.txt > geekify.patch
```

## Lo aplicamos

```
$ patch celebrities.txt < geekify.patch
```

## Comprobamos resultado

```
$ cat geekebrities.txt
```

```
Bash Spencer  
Larry Page  
Blogger De Niro  
Luis de Windows  
Steve Jobs  
Bill Gates  
Mark Zuckerberg  
Flickr Casillas
```

```
$ cat celebrities.txt
```

```
Bash Spencer  
Larry Page  
Blogger De Niro  
Luis de Windows  
Steve Jobs  
Bill Gates  
Mark Zuckerberg  
Flickr Casillas
```

```
$ diff celebrities.txt geekebrities.txt  
$
```

Podemos decir que diff genera una serie de comandos que transforman un fichero en otro con patch

# Modo contextual de diff

```
$ diff -c celebrities.txt geekebrities.txt
```

```
*** celebrities.txt      2013-11-05 10:55:31.000000000 +0100
--- geekebrities.txt    2013-11-05 10:56:25.000000000 +0100
*****
```

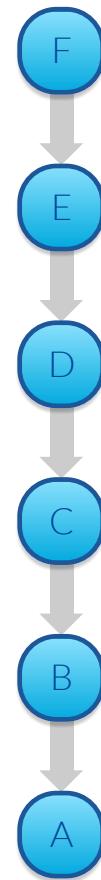
```
*** 1,8 ****
! Bud Spencer
  Larry Page
! Robert De Niro
! Luis de Guindos
  Steve Jobs
- Mr. Potato
  Bill Gates
  Mark Zuckerberg
```

```
--- 1,8 ----
! Bash Spencer
  Larry Page
! Blogger De Niro
! Luis de Windows
  Steve Jobs
  Bill Gates
  Mark Zuckerberg
+ Flickr Casillas
```

# Modo unificado de diff

```
$ diff -u celebrities.txt geekebrities.txt
```

```
--- celebrities.txt      2013-11-05 10:55:31.000000000 +0100
+++ geekebrities.txt    2013-11-05 10:56:25.000000000 +0100
@@ -1,8 +1,8 @@
-Bud Spencer
+Bash Spencer
 Larry Page
-Robert De Niro
-Luis de Guindos
+Blogger De Niro
+Luis de Windows
 Steve Jobs
-Mr. Potato
 Bill Gates
 Mark Zuckerberg
+Flickr Casillas
```



# **Introducción a Git**

# Creando un repositorio

`git init`

Creamos una carpeta para el proyecto y accedemos a ella

```
$ mkdir MiProyecto
```

```
$ cd MiProyecto
```

Creamos el repositorio

```
$ git init  
Initialized empty Git repository in [...]/MiProyecto/.git/
```



# Las tres zonas

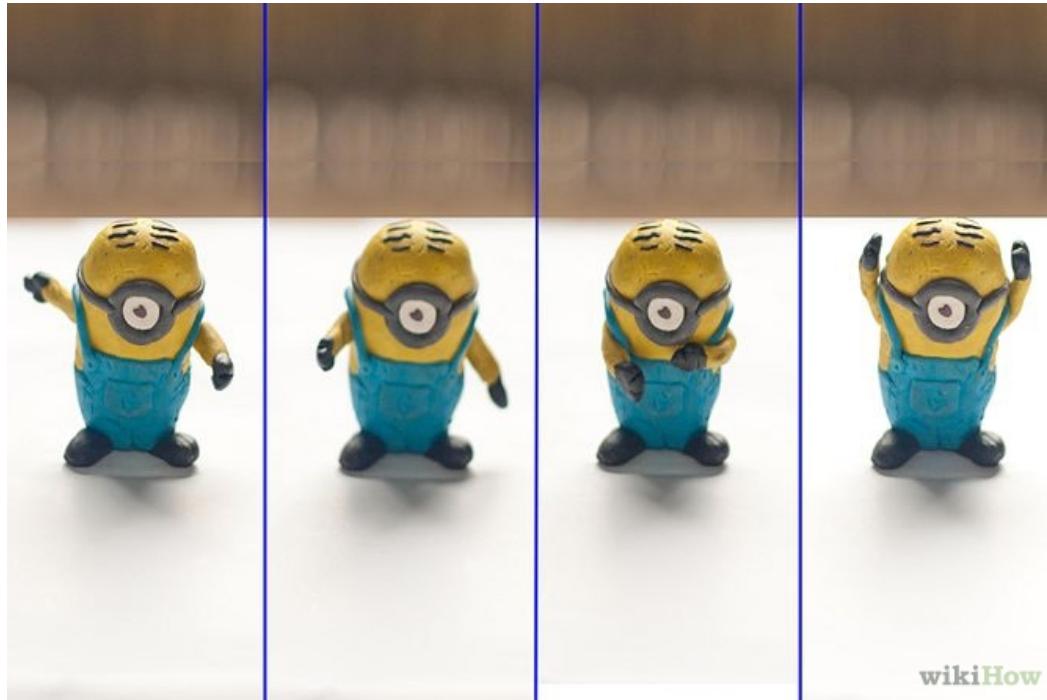
Working Copy

Staging Area

Repository

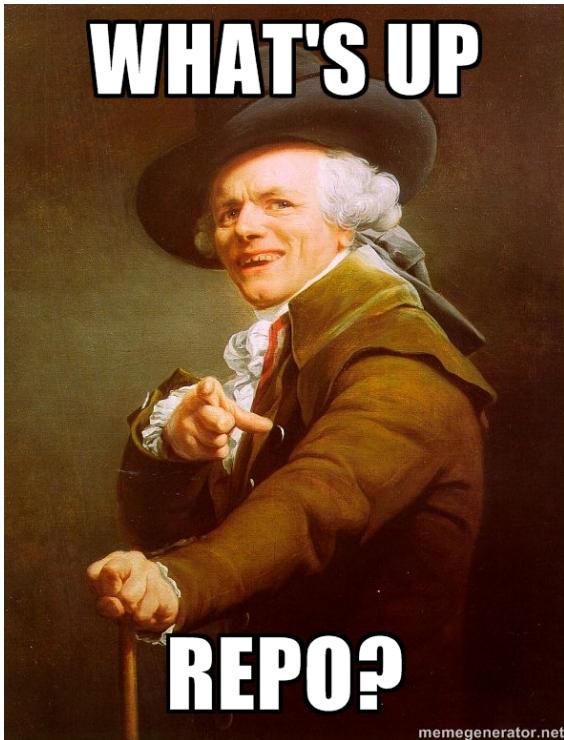
- **working copy**: la carpeta de nuestro proyecto
- **staging area**: donde pondremos los archivos a *trackear* (también se le conoce por index o cache)
- **repositorio**: donde se almacena toda la información de los cambios

# Ejemplo: animación por fotogramas



# ¿Cómo está mi repositorio?

git status



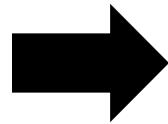
```
$ git status
```

Nos resume el estado de nuestro repositorio:

- qué archivos están en el **staging area** y cuales no
- qué archivos son trackeados
- qué archivos han sido modificados

# De **working copy** a **staging area**

Working Copy



Staging Area

**git add**

```
$ git add <filename>
```

Añade el archivo al **staging area**

```
$ git add <folder>
```

Añade el directorio al **staging area**

```
$ git add *.md
```

Añade los archivos .md al **staging area**



wiki



wiki



wiki

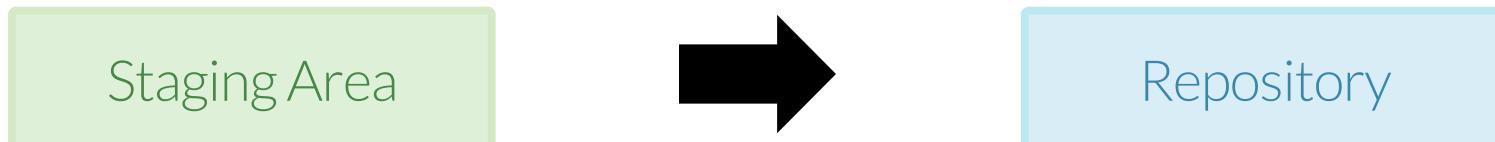


**git add *minion***



wiki

## Del staging area al repositorio



hacemos un commit

# ¿Qué es un commit?



Hacer un commit es como "hacer una foto"

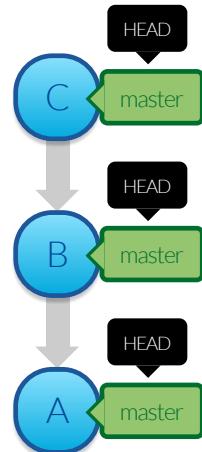
# **Ok, pero ¿qué es un commit?**

Un commit es un paquete que contiene:

- Uno o más «hunks» (el resultado de un diff)
- Un mensaje que describe qué cambios van en este commit
- Un hash SHA para identificar el commit
- Un enlace con su "commit padre"



```
$ git add *.c *.h  
$ git commit -m "Primer commit"  
$ git add *.c *.h  
$ git commit -m "Segundo commit"  
$ git add *.c *.h  
$ git commit -m "Tercer commit"
```



## git commit

```
$ git commit
```

Nos abrirá nuestro editor de texto para escribir un mensaje descriptivo del commit. Guardando el comentario como si fuera un archivo, creará el commit.

```
$ git commit -m "Mensaje rápido para el commit"
```

Hace el commit sin abrir el editor de texto usando el mensaje tras -m.

**¡ATENCIÓN!** Sólo se guardará lo que esté en el **staging area**

**¿Cómo podemos comprobar que se han hecho correctamente los commit?**

# Viendo nuestro log

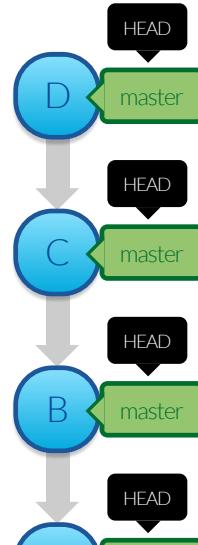
git log

```
$ git log
```

Log de los commits realizados

**¿Cómo ponemos en el **staging area** los archivos que borramos?**

```
$ git add *.h  
$ git commit -m "Añadir .h"  
$ git add *.c  
$ git commit -m "Añadir .c"  
$ rm *.h  
$ git rm *.h  
$ git commit -m "Borro los .c"
```





# Resumen

Working Copy

Staging Area

Repository



`git add`  
`git rm`



`git commit`

# Comparando, que es gerundio

¿Cómo saber qué diferencias hay entre mi **working copy**, el **staging area** y el **repo**?

**git diff**

```
$ git diff
```

**working copy** vs. **staging area**

```
$ git diff --staged
```

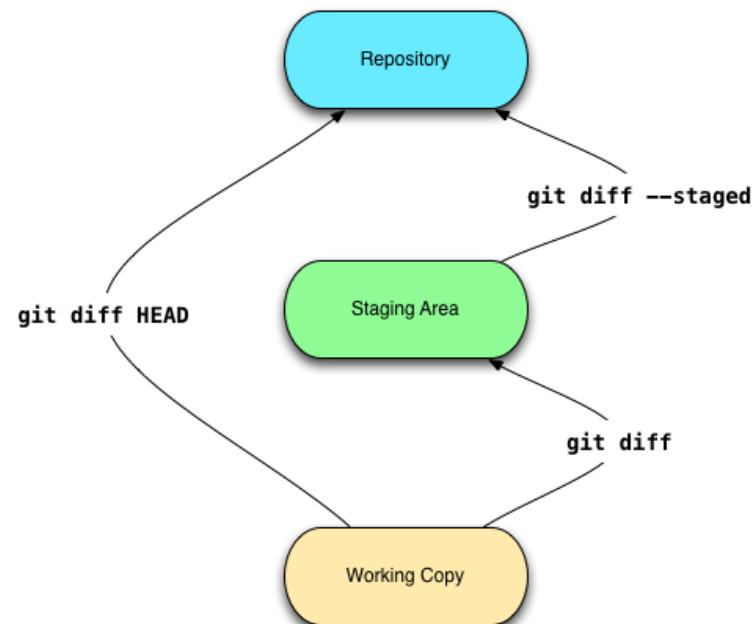
**staging area** vs. **repo**

También se puede usar **git diff --cached**

```
$ git diff HEAD
```

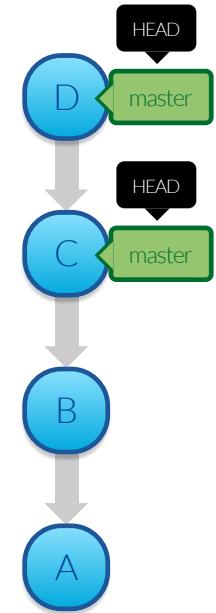
**working copy** vs. **repo**

## **git diff**



**Donde dije digo,  
digo Diego  
Deshacer lo hecho**

# Deshaciendo un commit



# Deshaciendo el último commit

`git reset`

Dos estrategias

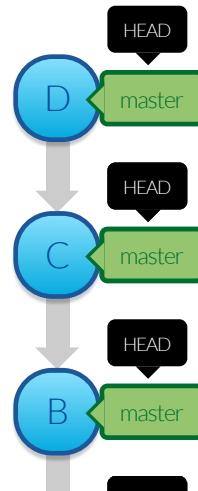
```
$ git reset HEAD~1
```

Deshacer el último commit pero mantener lo que había en mi **working copy**. Nuestro **staging area** queda vacío.

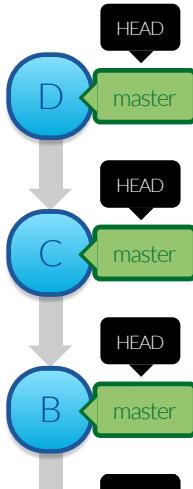
```
$ git reset --hard HEAD~1
```

Deshacer el último commit y lo que había en mi **working copy** de manera que todo quede como estaba antes. Nuestro **staging area** queda vacío.

```
$ git add *.h  
$ git commit -m "Añadir .h"  
$ git add *.c  
$ git commit -m "Añadir .c"  
$ rm *.h  
$ git rm *.h  
$ git commit -m "Borro los .c"  
$ git reset HEAD~1
```

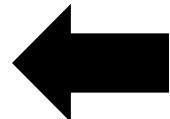


```
$ git add *.h  
$ git commit -m "Añadir .h"  
$ git add *.c  
$ git commit -m "Añadir .c"  
$ rm *.h  
$ git rm *.h  
$ git commit -m "Borro los .c"  
$ git reset --hard HEAD~1
```



# Dehaciendo un add

Working Copy



Staging Area

`git reset HEAD`

```
$ git reset HEAD <filename>
```

Saca el archivo del `staging area`

```
$ git reset HEAD <folder>
```

Saca el directorio del `staging area`

```
$ git reset HEAD *.txt
```

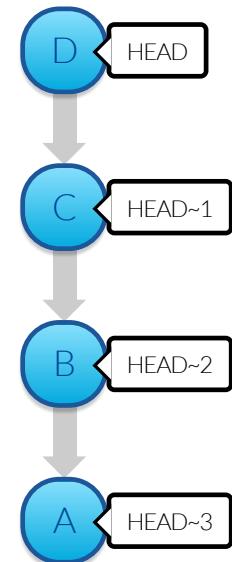
Saca los archivos.txt del `staging area`

```
$ git add *.c *.h
```

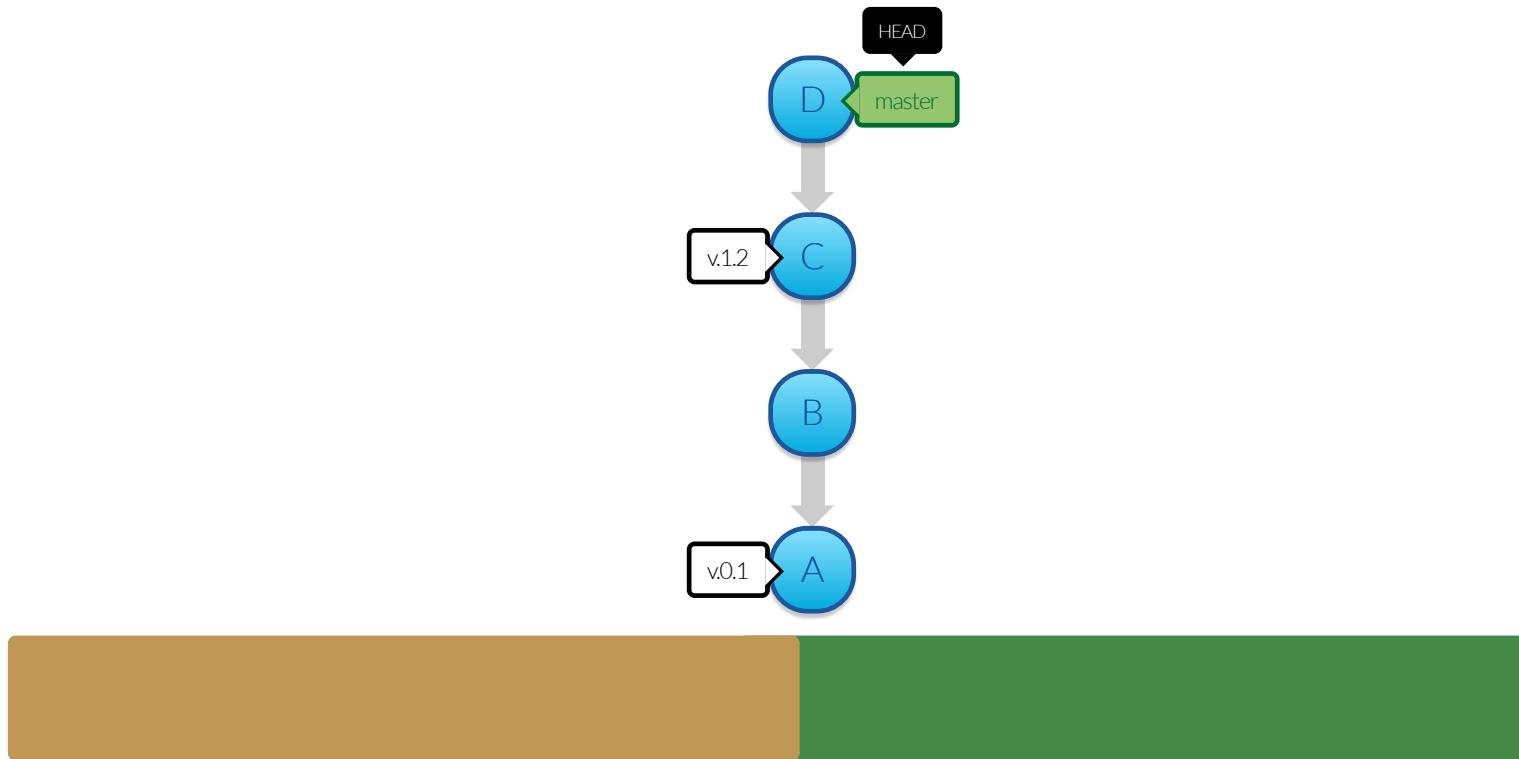
```
$ git reset HEAD *.h
```



# ¿Qué significa HEAD~1?



# Etiquetando commits: tags



# Viendo los tags

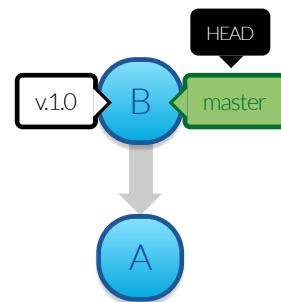
git tag

```
$ git tag
```

Nos da un listado de los tags del repositorio.

# Crear un tag

```
$ git tag v.1.0
```



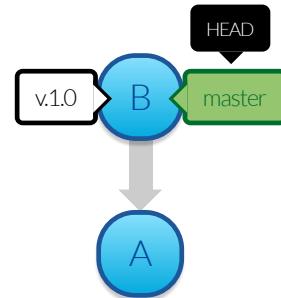
## git tag

```
$ git tag <tag_name>
```

Crea un tag de nombre <tag\_name> ligado al commit actual  
(apuntado por HEAD).

# Borrar un tag

```
$ git tag -d v.1.0
```



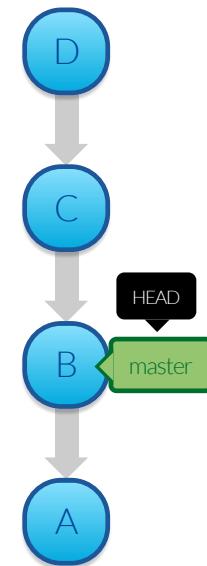
## git tag

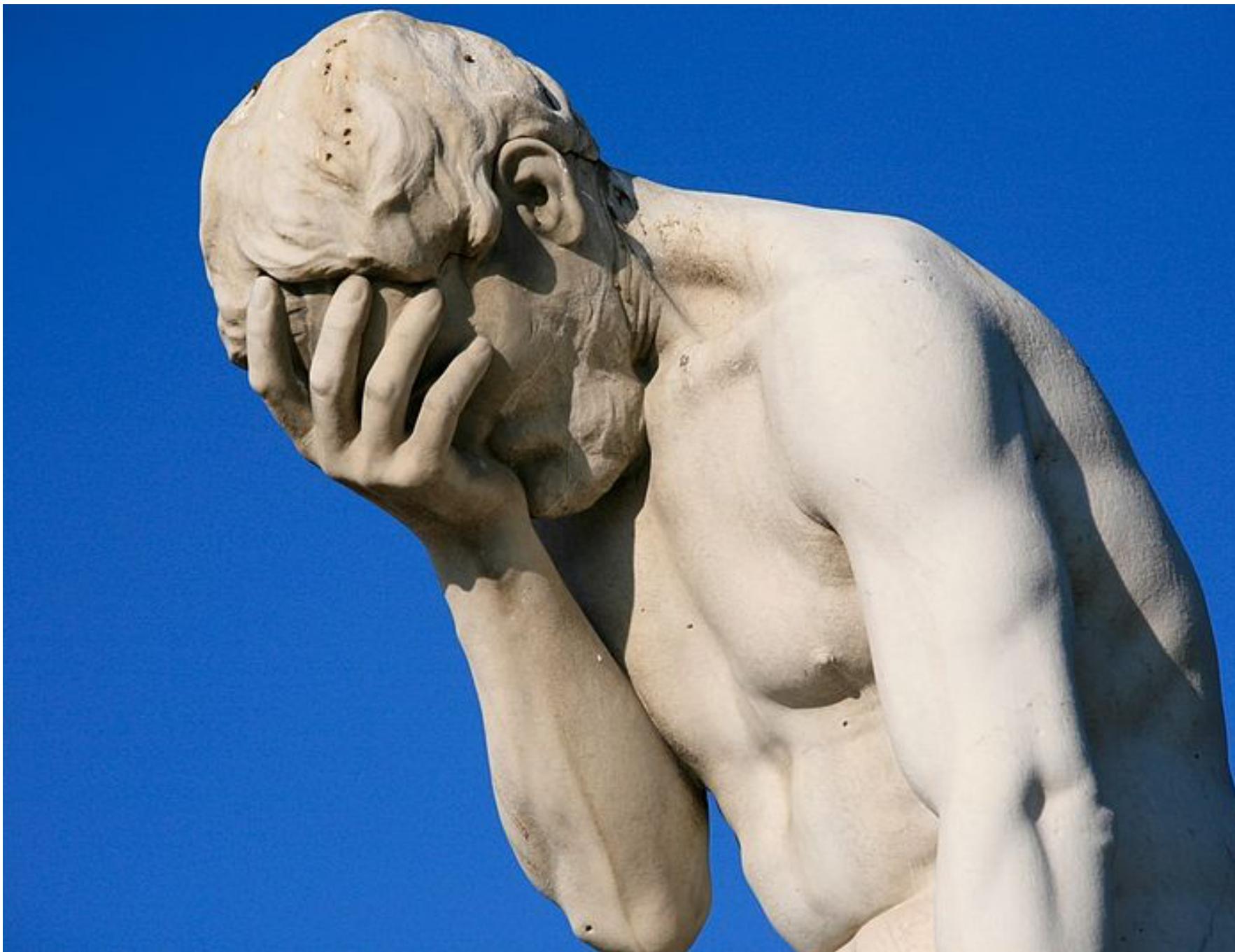
```
$ git tag -d <tag_name>
```

Eliminar el tag de nombre <tag\_name>.

**Donde dije Die-  
go, digo digo  
Deshaciendo lo "*deshecho*"**

# ¿Cómo volvemos a D?





# Git tiene síndrome de diógenes

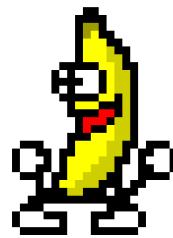
`git reflog`

Nos muestra todo lo que ha pasado en nuestro repositorio

```
$ git reflog
```

```
9e7ddad HEAD@{0}: reset: moving to HEAD~1
fc9dc03 HEAD@{1}: commit: D
9e7ddad HEAD@{2}: commit: C
fec3bd0 HEAD@{3}: commit: B
ac78fe7 HEAD@{4}: commit: A
```

¡Si vamos a **fc9dc03** estaremos en D!

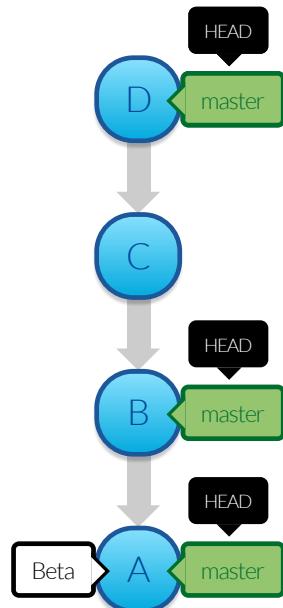


```
$ git reflog  
fc9dc03 HEAD@{0}: commit: D  
9e7ddad HEAD@{1}: commit: C  
fec3bd0 HEAD@{2}: commit: B  
ac78fe7 HEAD@{3}: commit: A
```

```
$ git reset fec3bd0
```

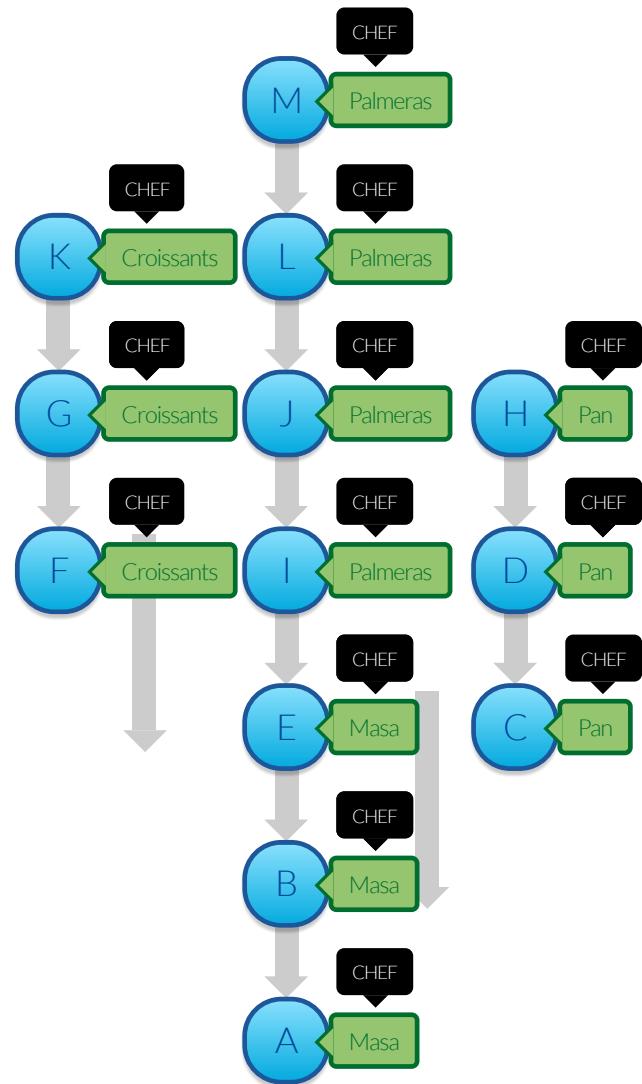
```
$ git reset Beta
```

```
$ git reset fc9dc03
```



# **Ramas**

## **Branches, para los amigos**



## **¿Qué son?**

- Son directorios virtuales.
- Universos paralelos con un punto en común.

## **¿Para qué sirven?**

- Diferentes punto de entrada al grafo
- Nos permiten desarollar diferentes cosas en paralelo

# Ver las ramas existentes

git branch

```
$ git branch
```

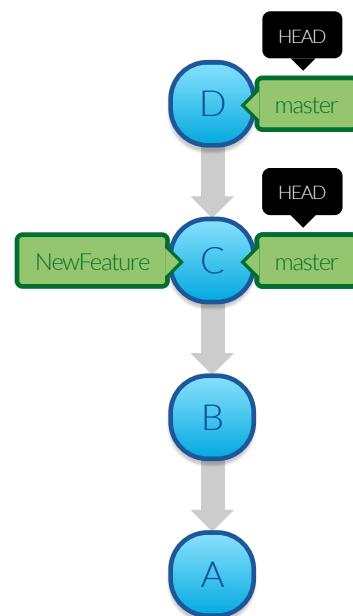
```
Capitulo-01
Capitulo-02
* Rama-en-la-que-estamos
Otra-rama
Valderrama
```

# **Crear una rama**

```
$ git branch NewFeature
```

```
$ git commit -m "Nuevo commit"
```

```
$ git reset HEAD~1
```



# Crear una rama

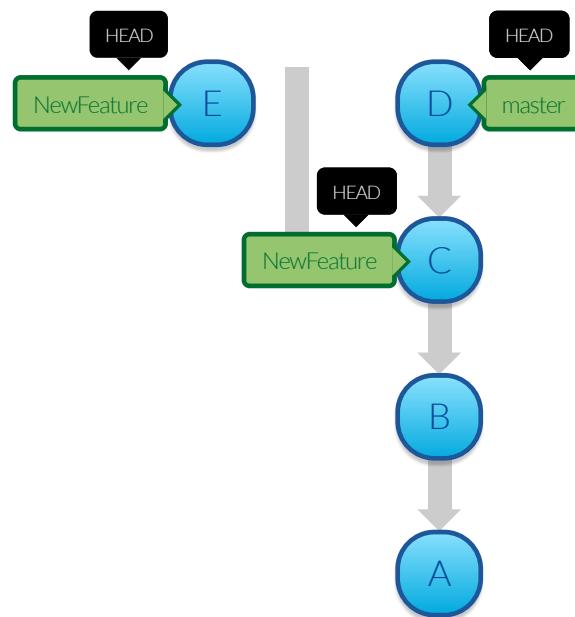
git branch

```
$ git branch <new_branch_name>
```

¡El contenido de working copy y staging area no varía!

# **Cambiar de una rama a otra**

```
$ git checkout NewFeature  
$ git checkout master  
$ git checkout NewFeature  
$ git commit -m "Primer commit en branch NewFeature"  
$ git checkout master
```



# Cambiar de una rama a otra

git checkout

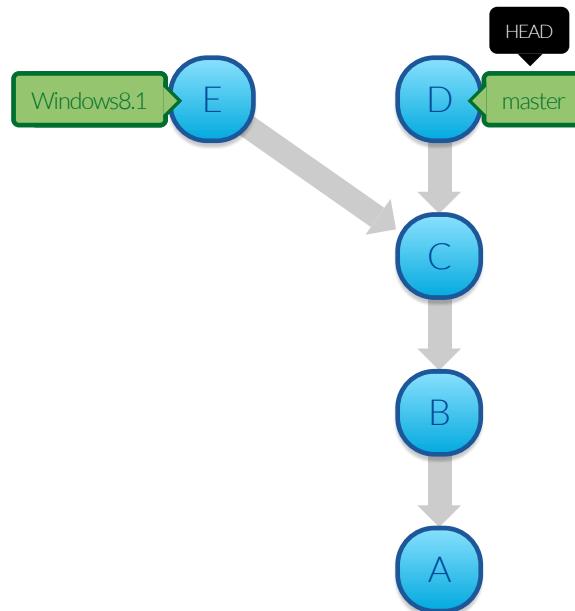
```
$ git checkout <branch_name>
```

```
Switched to branch '<branch_name>'
```

¡El contenido de working copy varía!

# **Renombrar ramas**

```
$ git branch -m Windows8 Windows8.1
```



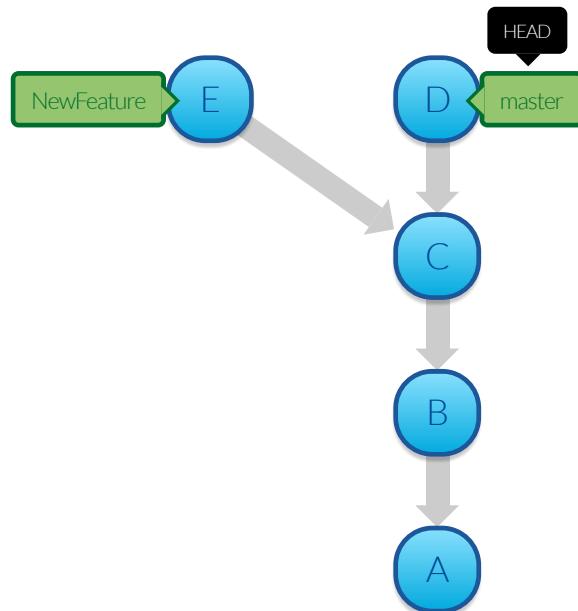
# Renombrar una rama

git branch -m

```
$ git branch -m <current_branch_name> <new_branch_name>
```

# **Eliminar ramas**

```
$ git branch -D NewFeature
```



# Eliminar una rama

git branch -D

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

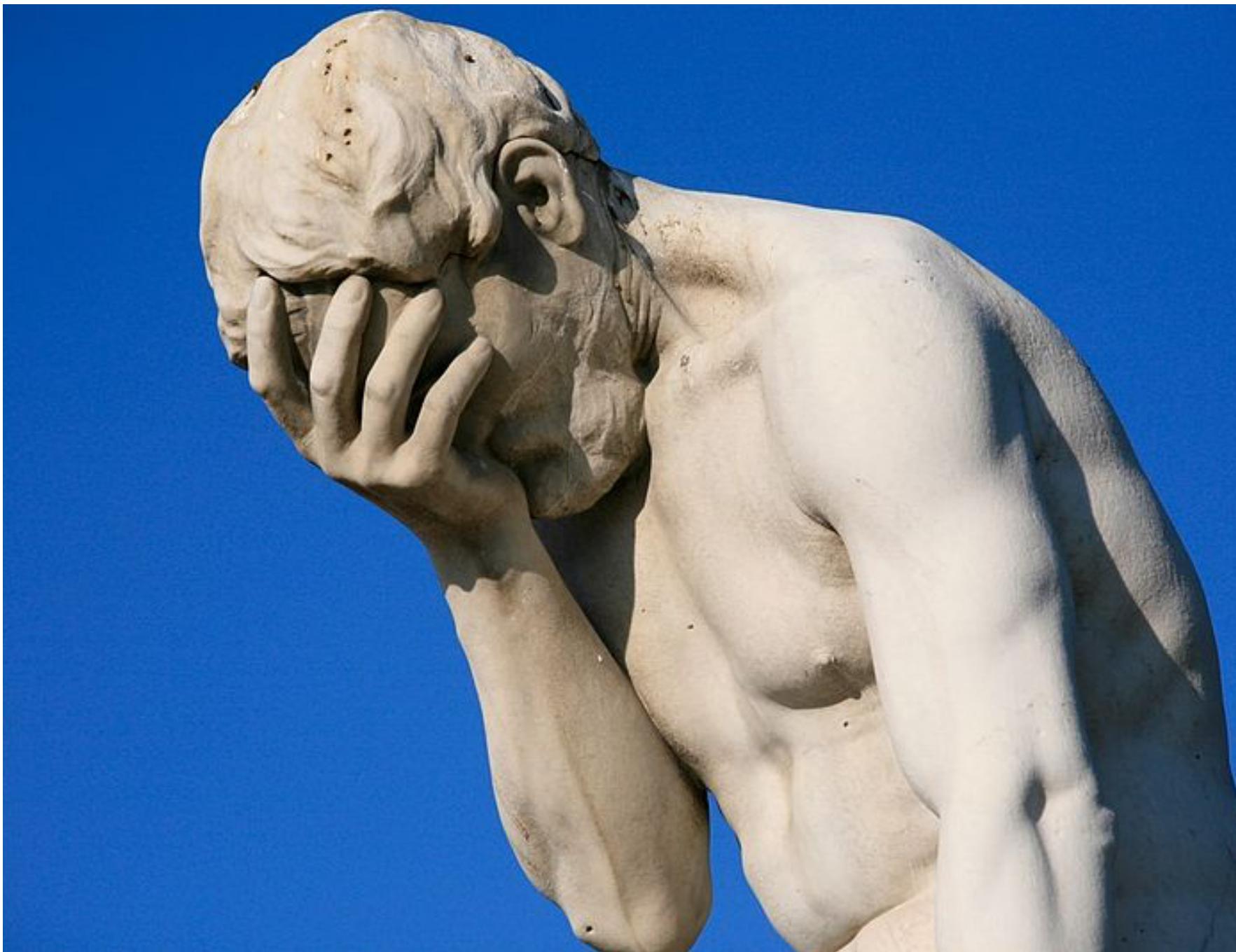
No es más que eliminar un puntero del grafo (un punto de acceso al grafo)

¡El contenido de **working copy** y **staging area** no varía!

Ojo! podemos dejar commits "inalcanzables"

No podemos eliminar la rama en la que estamos

# **¿Qué pasa con el commit inalcanzable?**





**KEEP  
CALM  
AND  
GIT REFLOG**



KEEP  
CALM  
AND  
GIT  
CHECKOUT

# Pero...

## ¿checkout no era para cambiar de rama?

Sí, pero también sirve para mover HEAD a cualquier commit (por su hash, por un tag o una referencia)

Vuelca el contenido de un commit a tu working copy

# ¿Cómo?

```
$ git checkout <branch_name>
```

Indicando el nombre de un branch

```
$ git checkout HEAD~<SOMETHING>
```

Referencia desde el HEAD

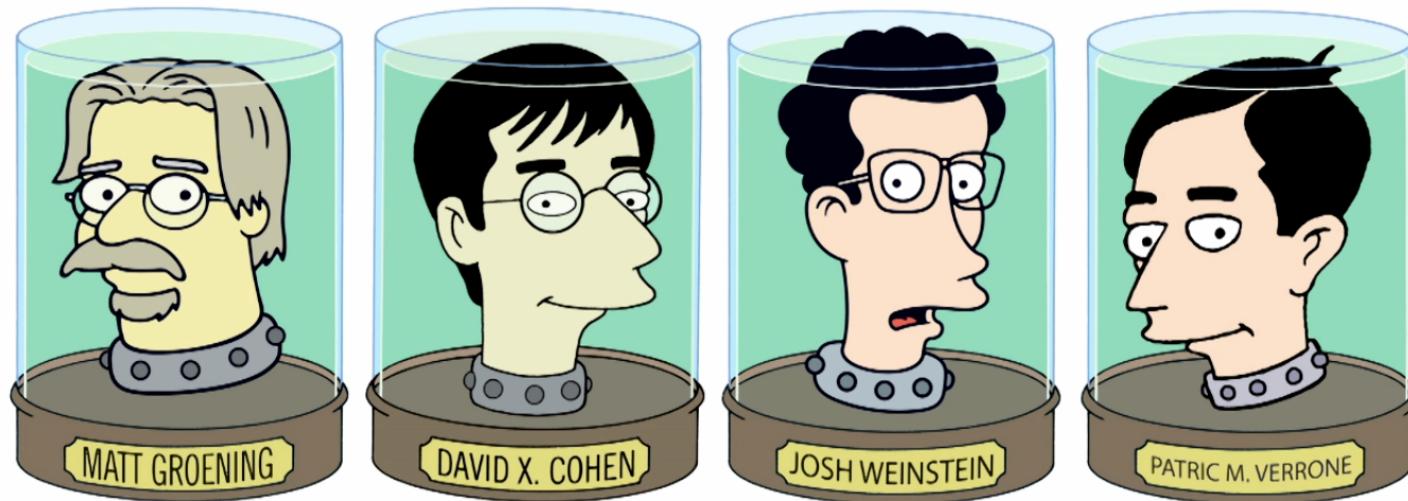
```
$ git checkout <commit_hash>
```

Indicando el hash de commit

```
$ git checkout <tag_name>
```

Indicando el nombre de un tag

# El estado 'detached HEAD'

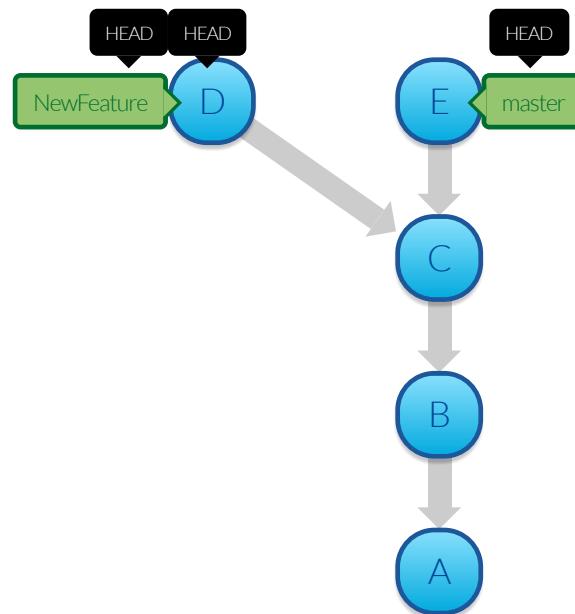


```
$ git reflog
```

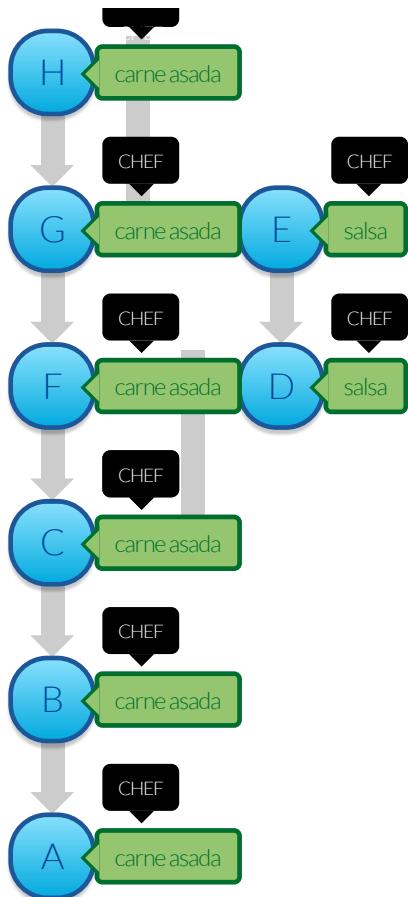
```
158a4da HEAD@{0}: commit: Commit con hash E  
fb62ffa HEAD@{1}: commit: Commit con hash D  
fc9dc03 HEAD@{2}: commit: Commit con hash C  
9e7ddad HEAD@{3}: commit: Commit con hash B  
fec3bd0 HEAD@{4}: commit: Commit con hash A
```

```
$ git checkout fb62ffa
```

```
$ git branch NewFeature
```



# **Merging Uniendo branches**



# Unir dos ramas

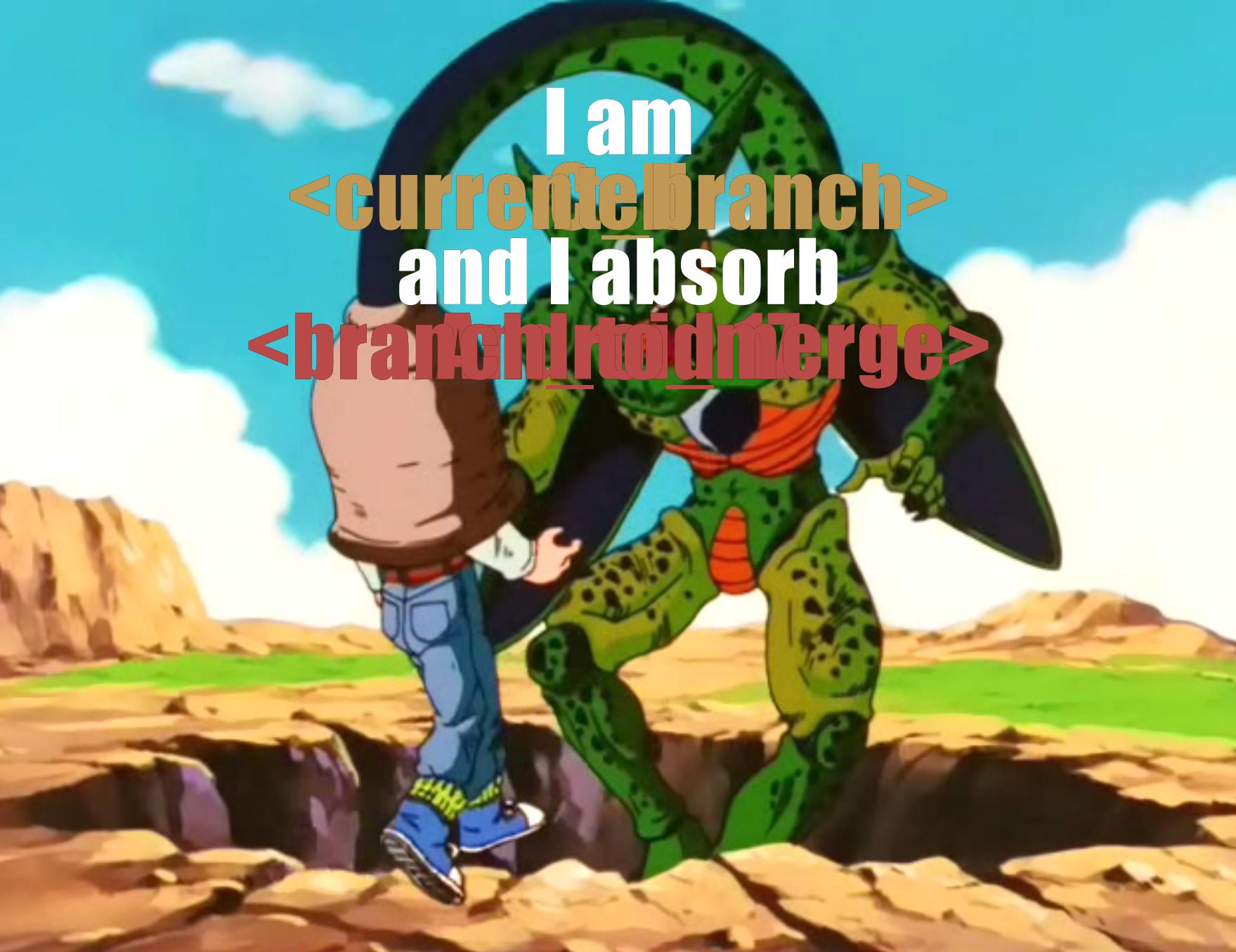
git merge

```
$ git merge <branch_to_merge>
```

¿Quién absorbe a quién?



I am  
**<current branch>**  
and all  
**<branchy or merge>**  
features will be assimilated.



I am  
**<currentbranch>**  
and I absorb  
**<branchtoidmerge>**

# **La rama en la que estamos, absorbe la que le indicamos con git merge**

Si estamos en 'carneAsada'  
y queremos hacer merge con 'salsa'

```
$ git merge salsa
```

Si estamos en 'master'  
y queremos hacer merge con 'feature'

```
$ git merge feature
```

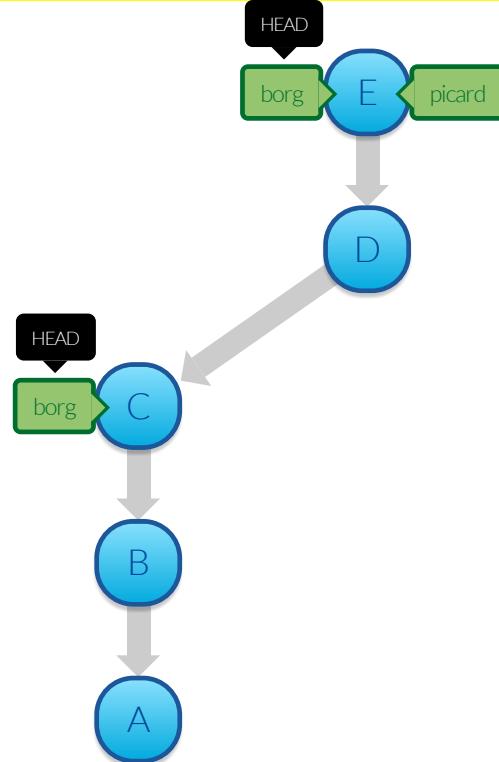
# **Cuando las ramas forman una lista**

Dos opciones:

- Con fast-forward (por defecto)
- Sin fast-forward

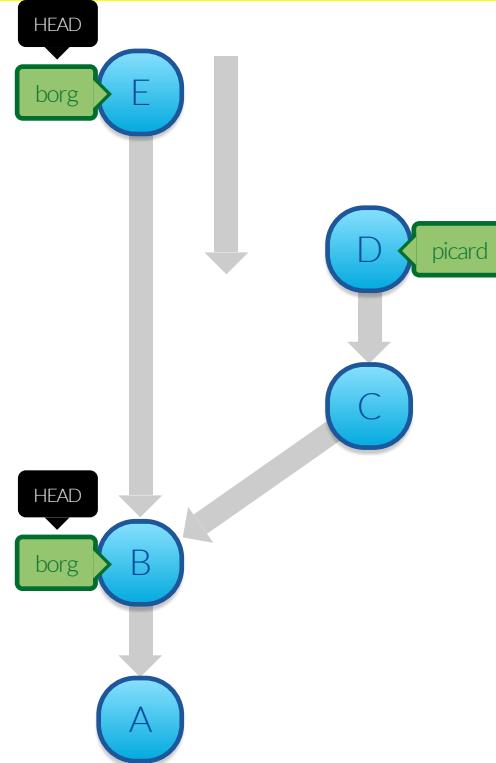
# Merge con fast-forward

```
$ git merge picard
```

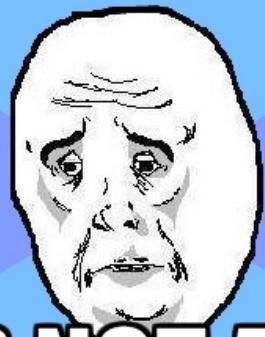


## Merge sin fast-forward

```
$ git merge --no-ff picard
```



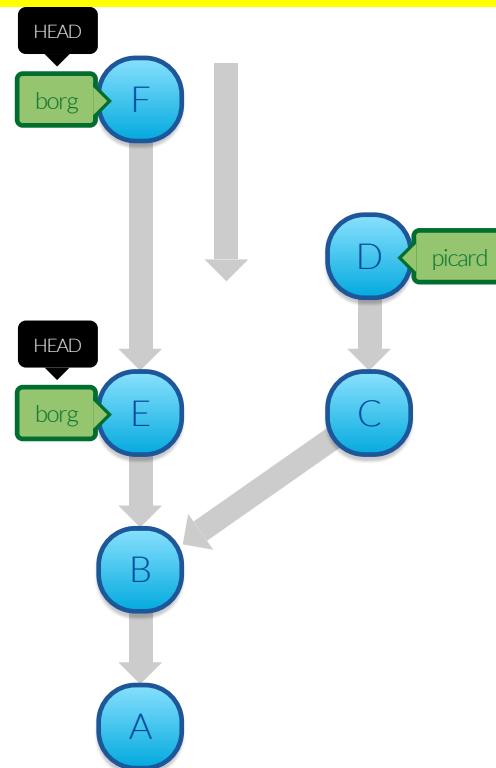
**SOMETIMES FAST-FORWARD**



**IS NOT AN  
OPTION**

memegenerator.net

```
$ git merge picard
```



# **¿Cómo ver los gráficos si no tenemos SourceTree?**



**KEEP  
CALM  
AND  
GIT LOG**

# Viendo nuestro log

git log

```
$ git log
```

Log de los commits de nuestro branch

```
$ git log --graph
```

Muestra el gráfico

```
$ git log --decorate
```

Muestra los punteros

```
$ git log --pretty=oneline
```

Muestra cada commit resumido en una linea

```
$ git log --graph --decorate --pretty=oneline
```

¡Todos juntos!

# **Selección de nodos**

# **Selección directa**

- Con el hash SHA de un commit
- Con el hash SHA reducido
- Con la referencia de reflog

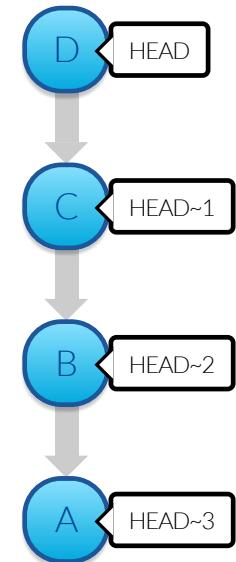
# Referencia de reflog

```
$ git reflog  
fec3bd0 HEAD@{0}: reset: moving to HEAD~2  
fc9dc03 HEAD@{1}: commit: D  
9e7ddad HEAD@{2}: commit: C  
fec3bd0 HEAD@{3}: commit: B  
ac78fe7 HEAD@{4}: commit: A
```

- HEAD@{1} - commit anterior al último comando
- HEAD@{2} - commit anterior al penúltimo comando
- HEAD@{3} - commit anterior al antepenúltimo comando

¡¡¡ HEAD@{1} no es lo mismo que HEAD~1 !!!

# Selección indirecta



```
HEAD = E
```

```
HEAD~0 = E
```

```
HEAD~1 = B
```

```
HEAD~2 = A
```

```
HEAD~3 = i i !!
```

```
HEAD~ = HEAD~1 = B
```

```
HEAD^0 = E
```

```
HEAD^1 = B
```

```
HEAD^2 = D
```

```
HEAD^2~1 = C
```

```
HEAD^2~2 = A
```

```
HEAD^3 = i i !!
```

```
A^2~1 = F
```

HEAD

master

E

experimental

C

B

D

# **Conflictos**

## **Cuando la resistencia nos es futil**

# MERGE CONFLICTS

MERGE CONFLICTS EVERYWHERE

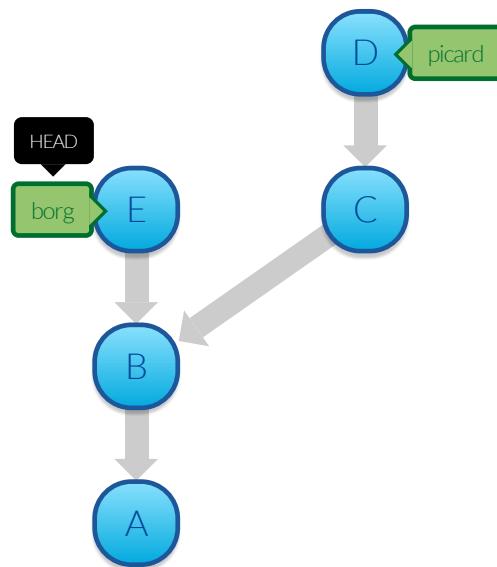
memegenerator.net

```
$ git merge master
```

```
Auto-merging war.txt
CONFLICT (content): Merge conflict in war.txt
Automatic merge failed; fix conflicts and then commit the result.
```

**¿Cómo y cuándo  
se genera un  
conflicto?**

Cuando dos archivos han sido editados  
en la misma línea en dos ramas diferentes



**cómo se solu-  
ciona?**

I DON'T ALWAYS HAVE  
MERGE CONFLICTS

A meme featuring the character "The Most Interesting Man in the World" from the TV show "Granite State". He is an older man with a full grey beard and mustache, wearing a dark pinstripe suit jacket over a white shirt. He is holding a cigar in his right hand and a small dog wearing sunglasses in his left arm. A green bottle of beer sits on the table next to him. The background is dark and textured.

BUT WHEN I DO I HAVE  
NO IDEA HOW TO SOLVE

```
Roses are red  
violets are #0000ff  
all my base  
Are belong to you.
```

Archivo en mi branch actual

```
Roses are red  
Violets are blue,  
All of my base  
Are belong to you.
```

Archivo en el branch con el que hago merge

```
Roses are red  
<<<<<< HEAD  
violets are #0000ff  
all my base  
=====  
Violets are blue,  
All of my base  
>>>>>> BranchToMerge  
Are belong to you.
```

Archivo en conflicto tras el merge

# **Resolver conflictos y darlos por resueltos**

1. Editar cada uno de los archivos en conflicto, quedándonos con el código que realmente nos interesa.
2. Hacer `git add` de esos archivos al `staging-area`.
3. Hacer `git commit` con los cambios.

# **Cancelar un merge con conflicto**

`git merge --abort`

```
$ git merge --abort
```

Cancela el merge que estábamos haciendo dejando todo como  
estaba

# Ejercicio

Ya se KUNG FU





# **Crear un repositorio**

## Crear un archivo poem.md con el contenido

```
Roses are red,  
Violets are blue.  
All of my base  
are belong to you.
```

**Añadir 'poem.md' del working copy al staging area**

**Mover lo que hay en el **staging area** a **repository****

## **Dibujar el diagrama**

**Crear un branch llamado 'htmlify'**

**Listamos los branch que tenemos**

**Nos movemos al branch 'htmlify'**

**Comprobamos que estamos en el branch correcto**

**Modificamos los colores 'red' por '#ff0000' y 'blue' por  
'#0000ff'**

**Añadir los cambios al staging area y luego pasarlos al repository**

## **Dibujar el diagrama**

## **Deshacer el último commit (sin perder los cambios)**

## **Dibujar el diagrama**

**Rehacer el último commit (el que acabamos de deshacer)**

**Hacer un merge con 'master' (htmlify absorbe master)**

**¿Ha causado algún conflicto?**

**¿Por qué?**

**¿Podemos deshacer el merge?**

## **Dibujar el diagrama**

**Volver al branch 'master'**

**Crear un nuevo branch llamado 'matrix'**

**Entrar en 'matrix'**



## **Modificar 'Roses' por 'Red pills' and 'Violets' por 'Blue pills'**

Red pills are red,  
Blue pills are blue.  
All of my base  
are belong to you.

## **Hacer un commit**

## **Dibujar el diagrama**

**Hacer un merge de 'matrix' en 'htmlify'**

**I am htmlify of Borg. All 'matrix features will be assimilated'**

**¿Causa conflictos?**

**¿Por qué?**

**Resolver el conflicto quedándonos con lo que había en 'htm-lify'**

# **Hacer un commit**

## **Dibujar el diagrama**

**Desde 'master', hacer un merge con 'htmlify'**

## **Dibujar el diagrama**

**Crear una nueva rama llamada 'title' y entrar en la misma**

## **Dibujar el diagrama**

**Añadir un título al poema y hacer un commit**

**Volver a 'master'**

## **Dibujar el diagrama**

**Hacer un merge "no fast-forward" de 'title' en 'master'**

**¿Pordía ser un merge con FF?**

**¿Por qué?**

## **Deshacer el merge (perdiendo los cambios)**

## **Eliminar el branch 'title'**

## **Dibujar el diagrama**

**Rehacer el merge que hemos deshecho**

**Deshacer todos los branch temporales: no los necesitaremos más**

## **Dibujar el diagrama**

**Deshacer todo y volver al estado inicial (cuando creamos el poema)**

**Pista (2 comandos)**

## **Dibujar el diagrama**

**Volver al estado final (con título) de nuestro poema**

## **Dibujar el diagrama**

## **Crear tags para los puntos más importantes:**

- initial state
- htmlified
- matrix
- html & matrix
- title

## **Dibujar el diagrama**

## **Movernos al tag 'matrix'**

**Repos remotos**  
**Júntate con los demás frikis**  
**en GitHub**

# ¿Qué es GitHub?

- Plataforma para alojar proyectos con Git
- El Facebook para los frikis
- Gratuita para proyectos open source
- Proporciona un issue tracker (gestor de incidencias)
- Y también un wiki (editor de contenidos colaborativo)
- Con páginas de **404 not found** y **500 server error** muy chulas
- Su logo es un "octogato" (Octocat)



# ¿Cómo empezar?

- Crea una cuenta gratuita en [github.com](https://github.com)
- Crea un repositorio
- Clona tu repositorio

# Clonar un repositorio

git clone

```
$ git clone <repo_url>
```

```
$ git clone https://github.com/kasappeal/startreklices.git
```

```
Cloning into 'startreklices'...
remote: Counting objects: 6, done.
remote: Compressing objects: 100% (5/5), done.
remote: Total 6 (delta 0), reused 3 (delta 0)
Unpacking objects: 100% (6/6), done.
```

Copiamos un repositorio remoto  
para trabajar en nuestra máquina

Nos crea una carpeta con el  
mismo nombre que el repositorio

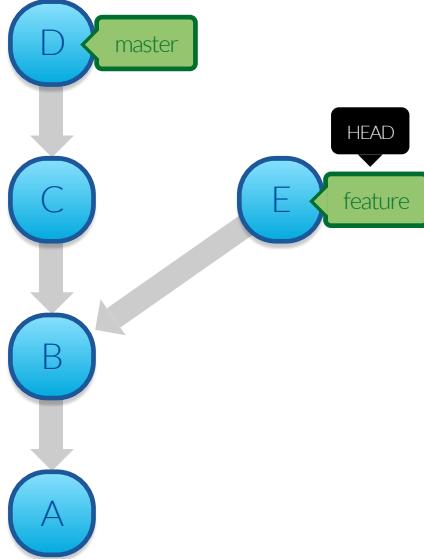
# Listar repositorios remotos de nuestro repo local

git remote

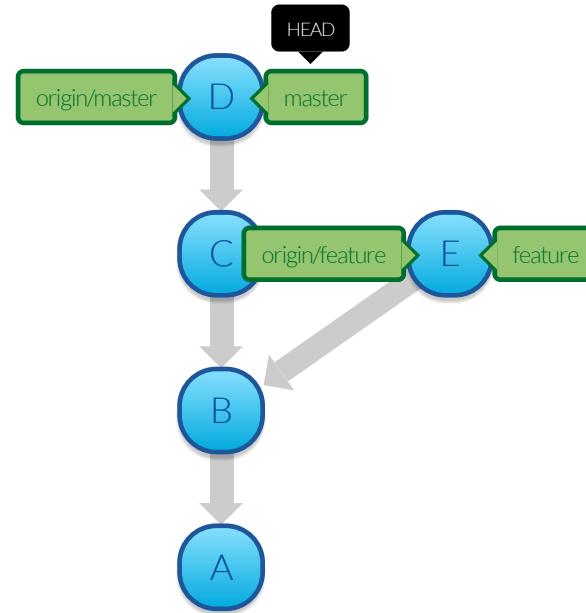
```
$ git remote
```

Nos muestra los repositorios remotos que tenemos

origin



local



# Añadir un repositorio remoto

git remote add

```
$ git remote add <remote_name> <remote_url>
```

Añade <remote\_name> como repositorio remoto

# Subir cambios al servidor

git push

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

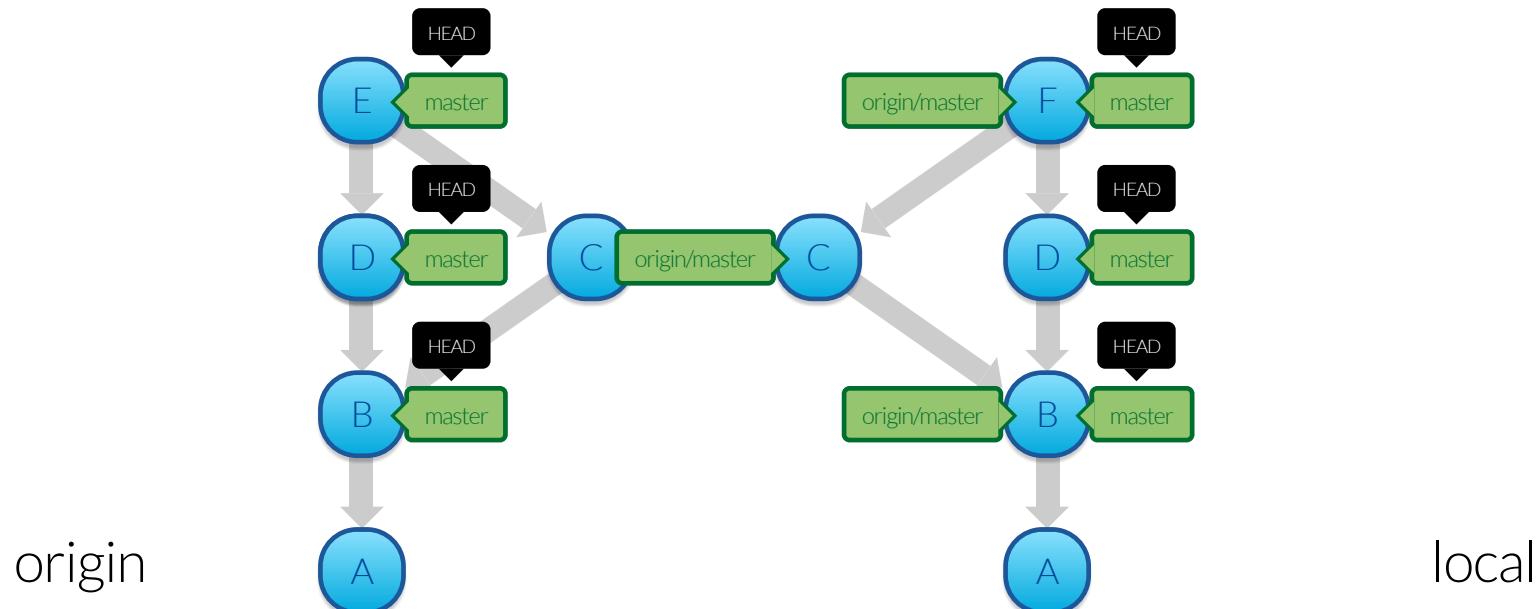
Sube nuestros cambios en la rama <branch> a <remote>

Si nuestro repo está anticuado, se rechazará

```
$ git commit -m "D"  
$ git push origin master
```

! [rejected] master -> master (non-fast-forward)

```
$ git pull origin  
$ git push origin master
```



# Descargar los cambios remotos

`git fetch`

```
$ git fetch <remote>
```

Descarga todos los cambios de todos branch de <remote> desde la última vez que descargamos.

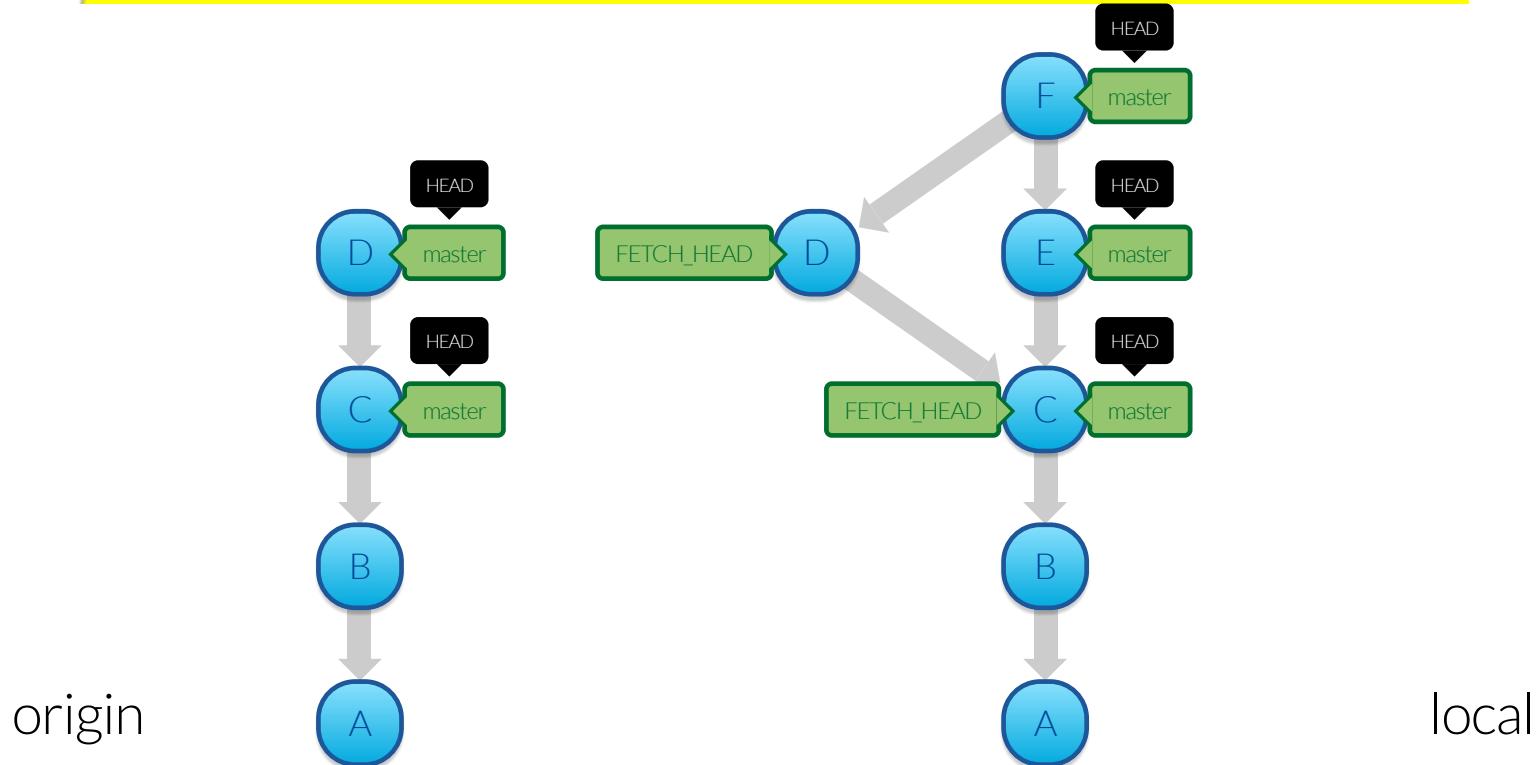
Crea nuevas ramas si es necesario.

```
$ git fetch <remote> <remote_branch>
```

Descarga los cambios del branch <remote\_branch> de <remote> desde la última vez que descargamos.

Crea una nueva rama si no la tenemos.

```
$ git commit -m "E"  
$ git fetch origin  
$ git merge FETCH_HEAD
```



# Descargar y aplicar los cambios remotos

git pull

```
$ git pull <remote>
```

Hace un `git fetch` y `git merge`

Perfecto para hacer actualizaciones directas

# Crear una rama en el servidor

Primero, crearla en nuestro repo local

```
$ git branch <new_branch>
```

Si no existe en local, no nos dejará.

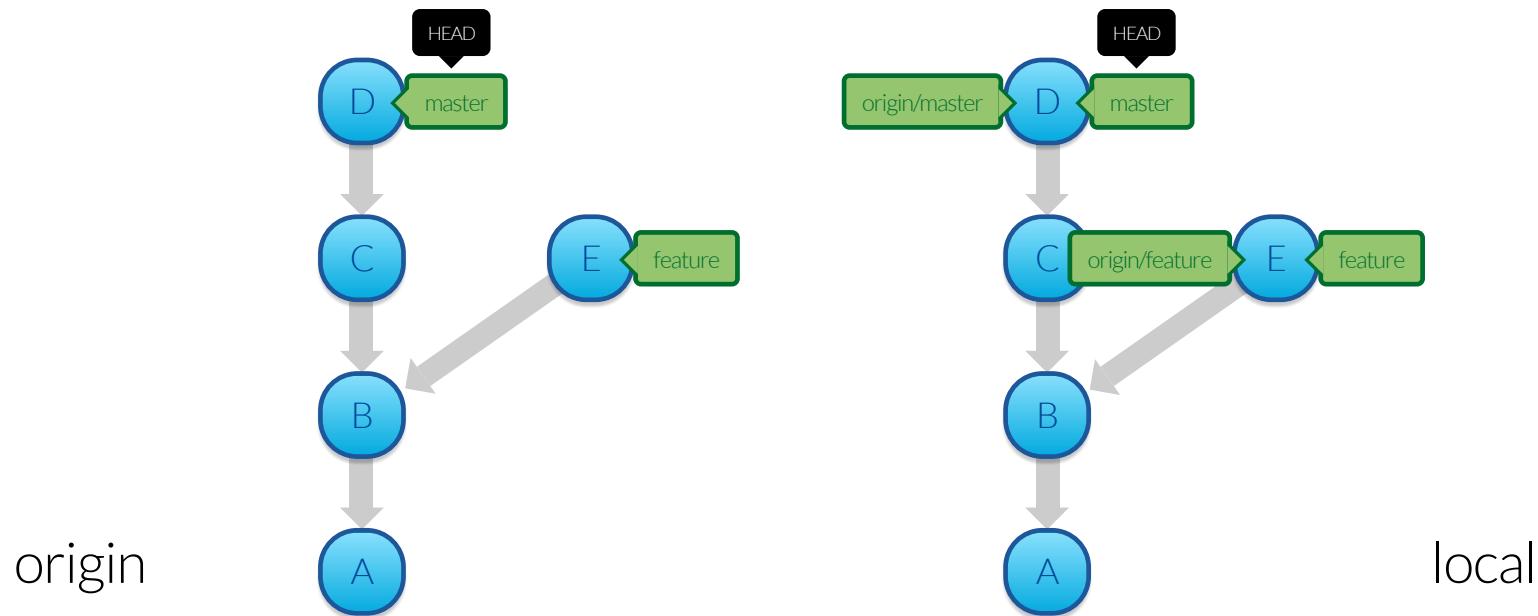
Después, crearla en el servidor

```
$ git push <remote> <new_branch>
```

¡Hacer lo mismo para los tags!

```
$ git branch feature
```

```
$ git push origin feature
```

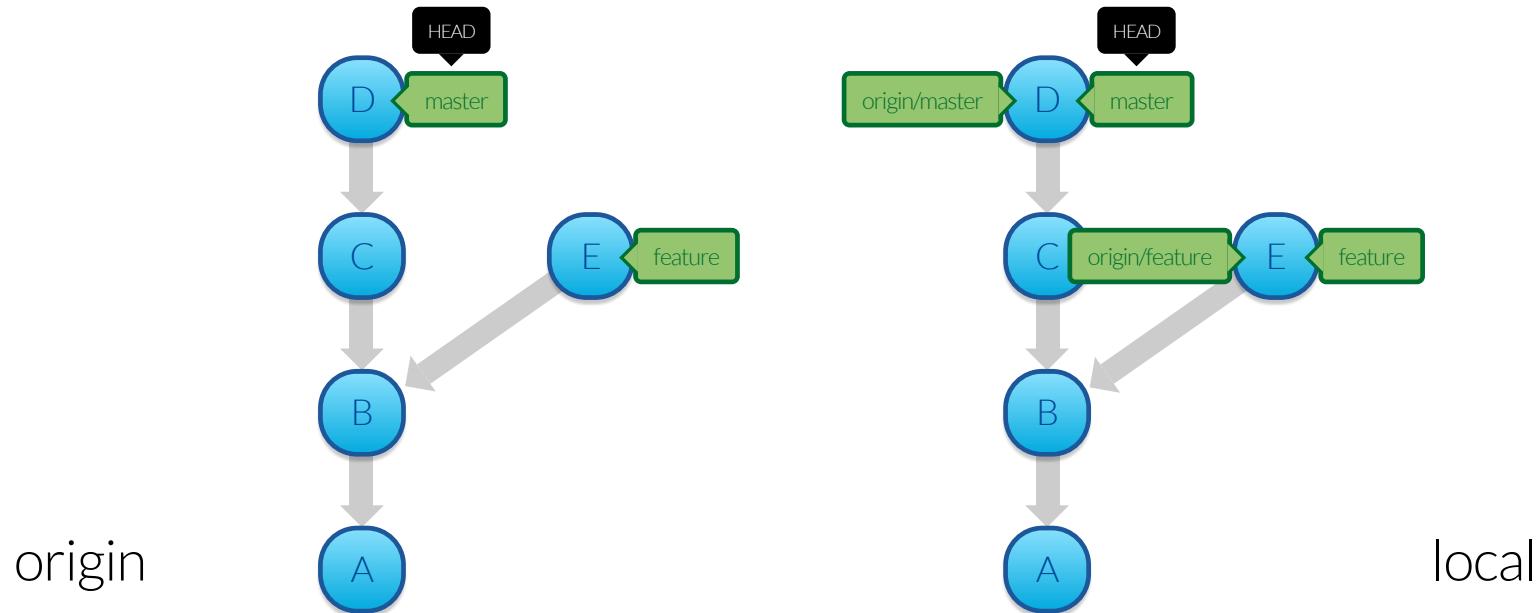


# Borrar una rama en el servidor

```
$ git push <remote> --delete <branch_to_delete>
```

¡Hacer lo mismo para los tags!

```
$ git push origin --delete feature
```



# Ejercicio

**Clonar <http://github.com/kasappeal/gitexample.git>**

# **Hacer parejas**

# **Añadir a GitHub como colaboradores**

**(Alber, esto va para ti)**

**Cada alumno debe crear un archivo con su nombre de usuario de GitHub**

# **Generar un conflicto y solucionarlo**

**Ahora, el que ha solucionado el conflicto debe generarlo y el que lo generó debe solucionarlo**

**Cada uno, crear un repositorio con  
nuestra cuenta de GitHub**

# **Añadir el repositorio remoto al proyecto**

**Subimos los cambios a nuestro repositorio recién creado (no a origin)**



# **¿Qué es hacer fork?**

Básicamente hacerse una copia de un proyecto en tu cuenta de GitHub

# **¿Qué es un pull request?**

Con un *pull request* solicitas que incorporen tus cambios a un repositorio a partir de un fork.

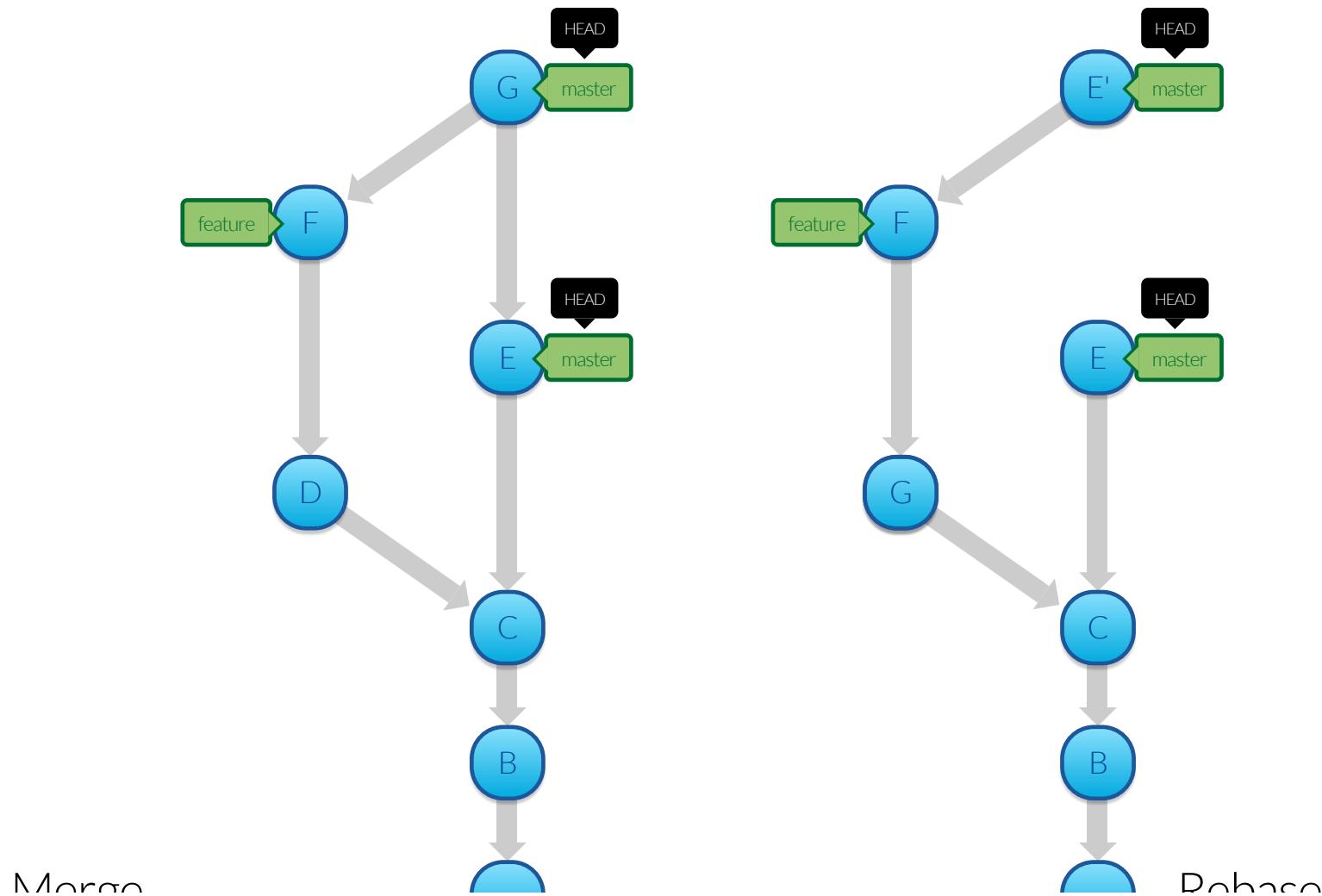
# **Rebase: haciendo injertos**

# ¿Qué es? ¿para qué sirve?

- Parecido a merge
- Cambiar el ancestro de nuestro “branch”
- No hace fast-forward
- Tampoco añade commit extra
- Al “unir”, reescribe los cambios (crea nuevos hash)
- Algo peligroso en ciertas situaciones

```
$ git merge feature
```

```
$ git rebase feature
```



# **Rebase**

## **git rebase**

```
$ git rebase <NuevoAncestro>
```

Como merge, hace el rebase desde el branch en el que estamos

# ¿Cuándo usarlo?

- Sólo en branches en los que trabajamos nosotros sólo
- Para incorporar commits de otros "branches"
- Actualizar nuestro repo local con uno remoto

```
$ git pull --rebase
```

# **¿Cuándo NO usarlo?**

- Cuando trabajamos en una rama pública
- Para unir con una rama pública (origin/master)

# Modo interactivo

## git rebase

```
$ git rebase -i <NuevoAncestro>
```

- Modo interactivo de rebase
- Permite reescribir el histórico de una manera más intuitiva y fácil

# **Modo interactivo**

pick f7f3f6d Me quedo como estoy

squash 310154e Me fusiono con f7f3f6d

squash a5f4a0d Me fusiono con f7f3f6d

edit c89102a Editarme antes de terminar rebase

# **Modo interactivo**

```
edit c89102a Editarme antes de terminar rebase
squash 310154e Me fusiono con c89102a
squash a5f4a0d Me fusiono con c89102a
pick f7f3f6d Me quedo como estoy
```

# Rebase onto

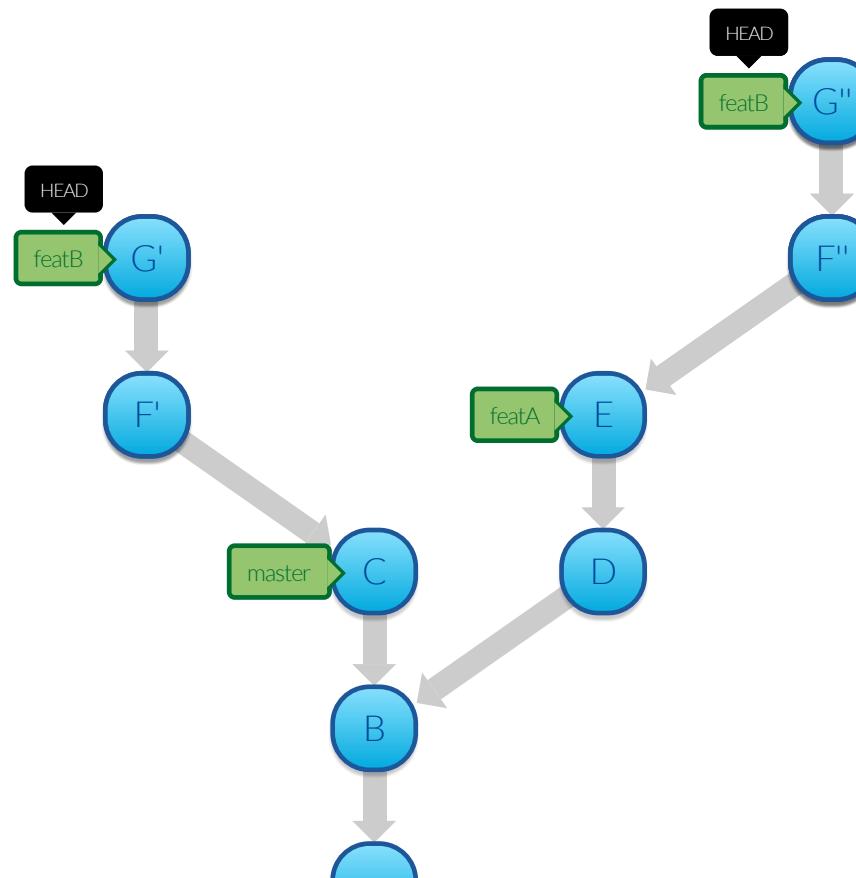
git rebase

```
$ git rebase --onto <NuevoAncestro> <AncestroActual>
```

- Nos permite cambiar el ancestro de un “branch” saltándose otros ancestros

```
$ git rebase --onto master featA
```

```
$ git rebase featA
```



# **Eliminar cosas de nuestro repo**

voy a matar a moe



# Borrar un archivo de todos los commit

git filter-branch

```
$ git filter-branch --tree-filter 'rm -f <filename>' HEAD
```

Borra el archivo <filename> de todos commit.

# **Extras**

## **Para divertirnos un poco más**

# **Ignorando archivos en cada proyecto**

## **.gitignore**

.gitignore es un archivo que nos permite indicar qué archivos han de ignorarse en git

Debe estar en el raíz de nuestro repositorio.

.gitignore es un archivo que nos permite indicar qué archivos han de ignorarse en git

```
# Compilados y ejecutables
*.class
*.dll
*.exe

# Archivos comprimidos
*.dmg
*.zip

# Logs y BBDD
*.log
*.sql

# Archivos generados por SSOO
.DS_Store
Thumbs.db
._*
```

# Ignorando globalmente

```
$ git config --global core.excludesfile /path/al/archivo
```

Establece de manera global (para todos los proyectos) este gitignore

# .gitignore para Android con Eclipse

```
# Compilados y ejecutables
# generated files
bin/
gen/

# Eclipse project files
.classpath
.project
```

# **.gitignore para Xcode**

[goo.gl/dfJZnh](http://goo.gl/dfJZnh)

# **Workflow Feature Branch**

# Gitflow Feature Branch



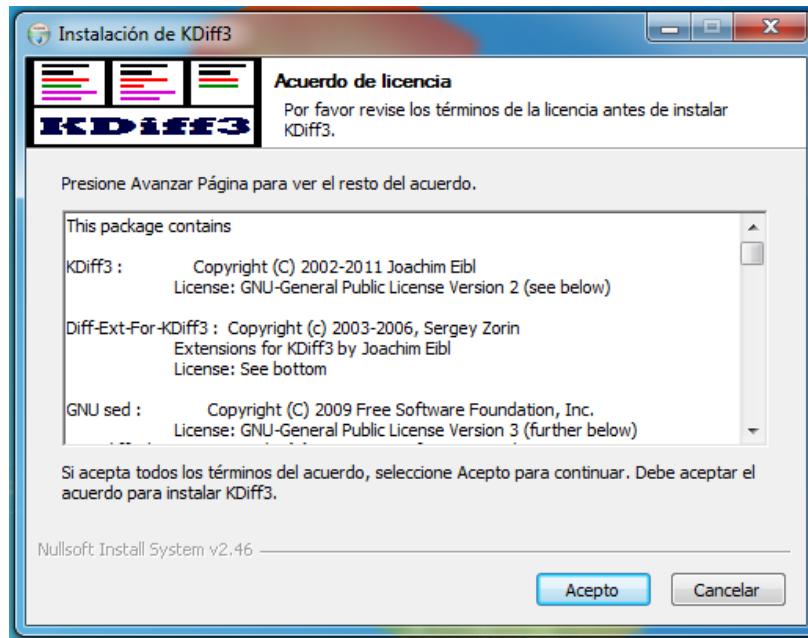
Fuente: atlassian.com <http://goo.gl/RHvtFc>

# Resolviendo conflictos visualmente con Kdiff3

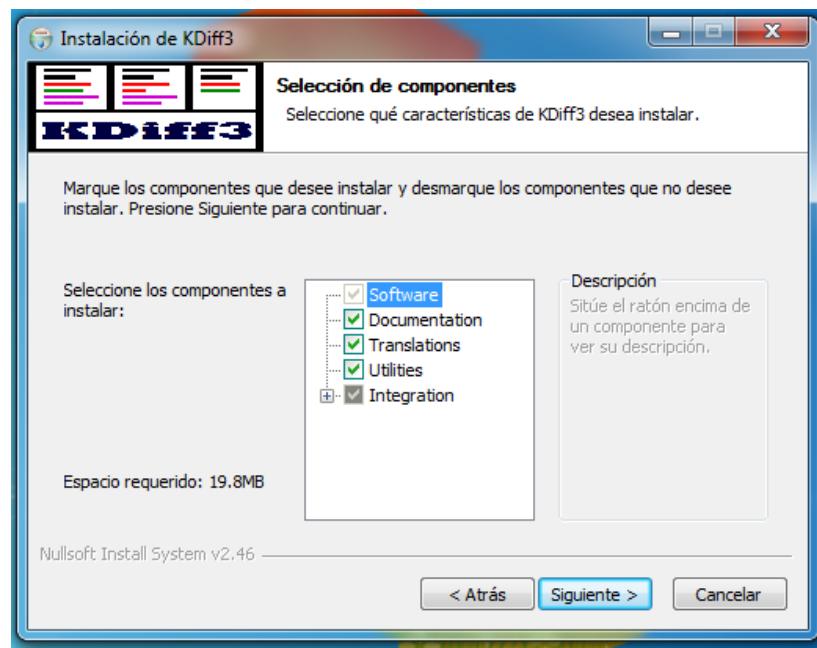
The screenshot shows the Kdiff3 application window with three tabs: A (Base), B (Local), and C (Remote). The code editor displays a Java file named guestInfo.jspx. The local file (B) has changes in lines 78 and 79, while the remote file (C) has changes in lines 78 and 80. A merge conflict is indicated at line 79 with the message "Merge Conflict". The output pane at the bottom shows the merged code.

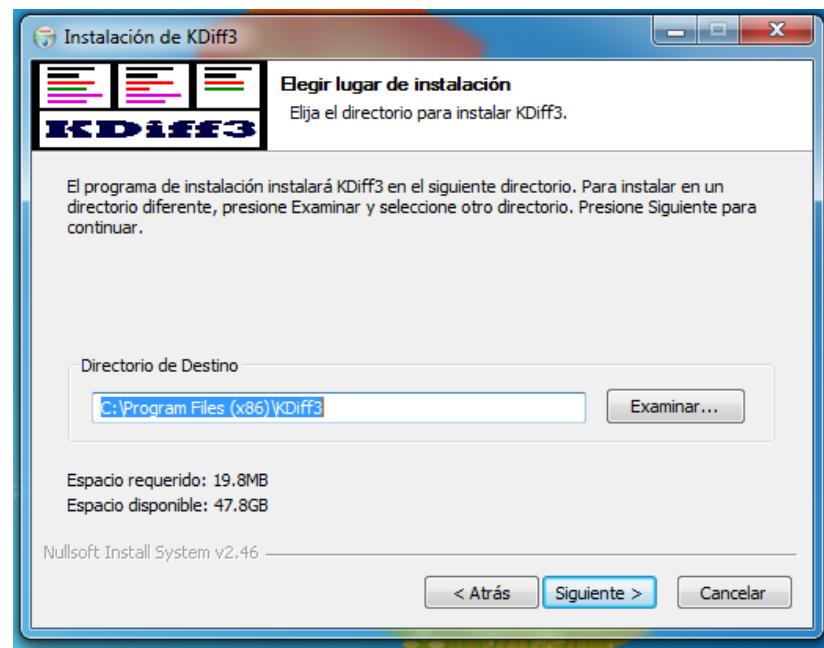
```
Diff3 .../guestInfo.jspx (Base) <-> .../guestInfo.jspx (Local) <-> .../guestInfo.jspx (Remote) - KDiff3
File Edit Directory Movement Diffview Merge Window Settings Help
A (Base): \src\main\webapp\WEB-INF\views\agreementflow\guestInfo.jspx (Base) B: \src\main\webapp\WEB-INF\views\agreementflow\guestInfo.jspx (Local) C: \src\main\webapp\WEB-INF\views\agreementflow\guestInfo.jspx (Remote)
Top line 75 Encoding: System Line end style: DOS Top line 74 Encoding: System Line end style: DOS Top line 75 Encoding: System Line end style: DOS
075 "guest.address.line": "Please en 074 "guest.address.line": "Please en 075 "guest.address.line": "Please en
076 "guest.address.city": "Please en 075 "guest.address.city": "Please en 076 "guest.address.city": "Please en
077 "guest.address.state": "Please e 076 "guest.address.state": "Please e 077 "guest.address.state": "Please e
078 "guest.address.zipCode": "Please 077 078 required: "Please enter the 079 "guest.address.zipCode": ( 078 required: "Please enter the
079 079 number: "An invalid zip code 080 ), 079 number: "An invalid zip code 080
080 ) , 080 ) ,
081 "guest.identification.number": " 081 "guest.identification.number": " 082 "guest.identification.number": "
082 "guest.identification.state": "P 082 "guest.identification.state": "P 083 "guest.identification.state": "
083 "guest.identification.expiryDate 083 "guest.identification.expiryDate 084 "guest.identification.expiryDate "
084 "noIdRentalApprovedTeamMemberNum 084 "noIdRentalApprovedTeamMemberNum 085 "noIdRentalApprovedTeamMemberNum "
085 "vehicleRentalDetail.insurance.c 085 "vehicleRentalDetail.insurance.c 086 "vehicleRentalDetail.insurance.c "
086 "vehicleRentalDetail.insurance.a 086 "vehicleRentalDetail.insurance.a 087 "vehicleRentalDetail.insurance.a "
087 "vehicleRentalDetail.insurance.p 087 "vehicleRentalDetail.insurance.p 088 "vehicleRentalDetail.insurance.p "
088 "vehicleRentalDetail.insurance.a 088 "vehicleRentalDetail.insurance.a 089 "vehicleRentalDetail.insurance.a "
089 "vehicleRentalDetail.insurance.a 089 "vehicleRentalDetail.insurance.a 090 "vehicleRentalDetail.insurance.a "
090 "vehicleRentalDetail.insurance.a 090 "vehicleRentalDetail.insurance.a 091 "vehicleRentalDetail.insurance.a "
Output: \src\main\webapp\WEB-INF\views\agreementflow\guestInfo.jspx Encoding for saving: Codec From C: System Line end style: DOS (A, B, C)
"guest.address.city": "Please enter the city",
"guest.address.state": "Please enter the state",
"guest.address.zipCode": (
    required: "Please enter the zip code",
? Merge Conflict
),
"guest.identification.number": "Please enter guest license/ID number",
"guest.identification.state": "Please enter guest license state",
"guest.identification.expiryDate": "Please enter guest license expiration date in mm/dd/yyyy format",
"noIdRentalApprovedTeamMemberNumber": "Please select the GM who approved the rental without any ID",
"vehicleRentalDetail.insurance.companyName": "Please enter insurance company name",
"vehicleRentalDetail.insurance.agent.name": "Please enter insurance agent name",
"vehicleRentalDetail.insurance.policyNumber": "Please enter insurance policy number",
"vehicleRentalDetail.insurance.agent.phoneNumber": "Please enter a valid insurance agent phone number in ###-###-### format",
"vehicleRentalDetail.insurance.agent.address.line": "Please enter insurance agent address",
"
Number of remaining unsolved conflicts: 2 (of which 0 are whitespace)
```

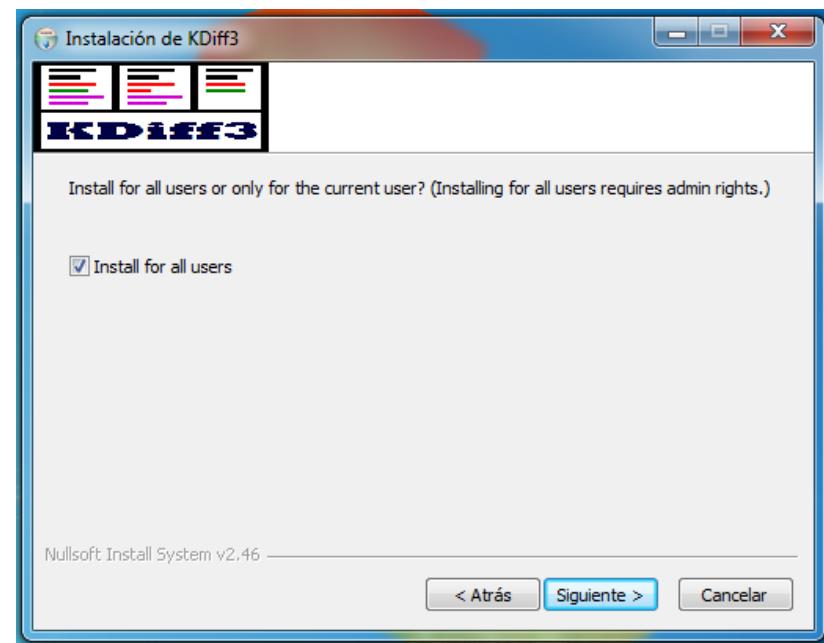
# Instalando Kdiff3 en Windows

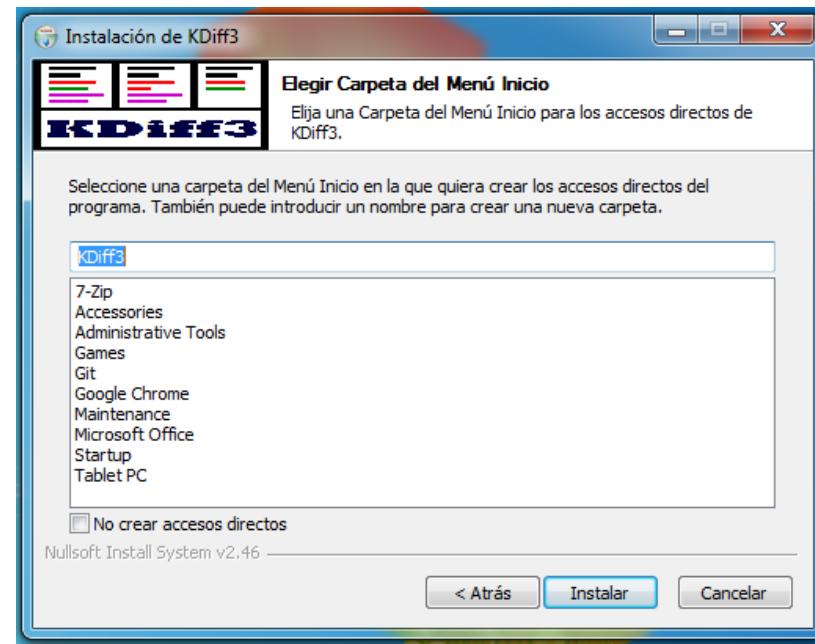


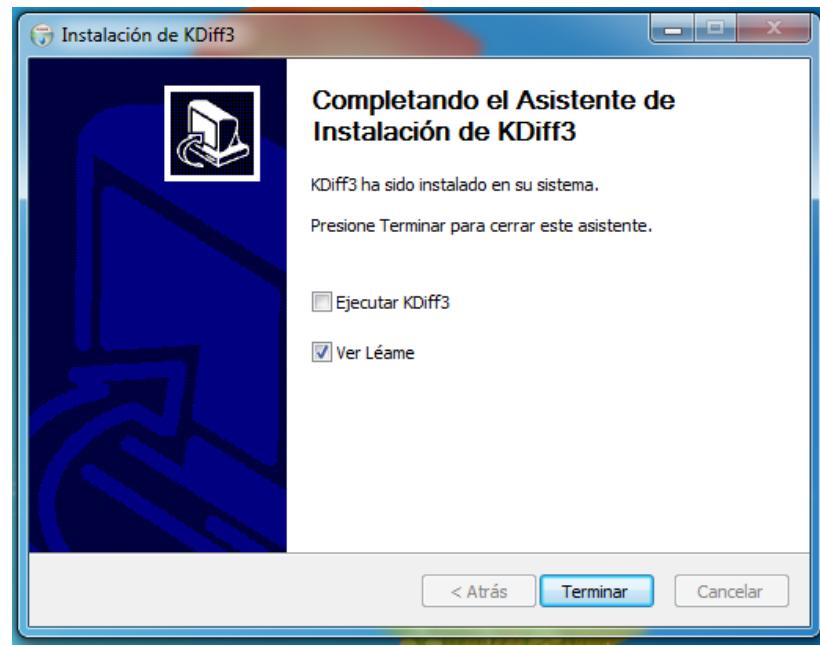
Con el instalador gráfico <http://sourceforge.net/projects/kdiff3/>



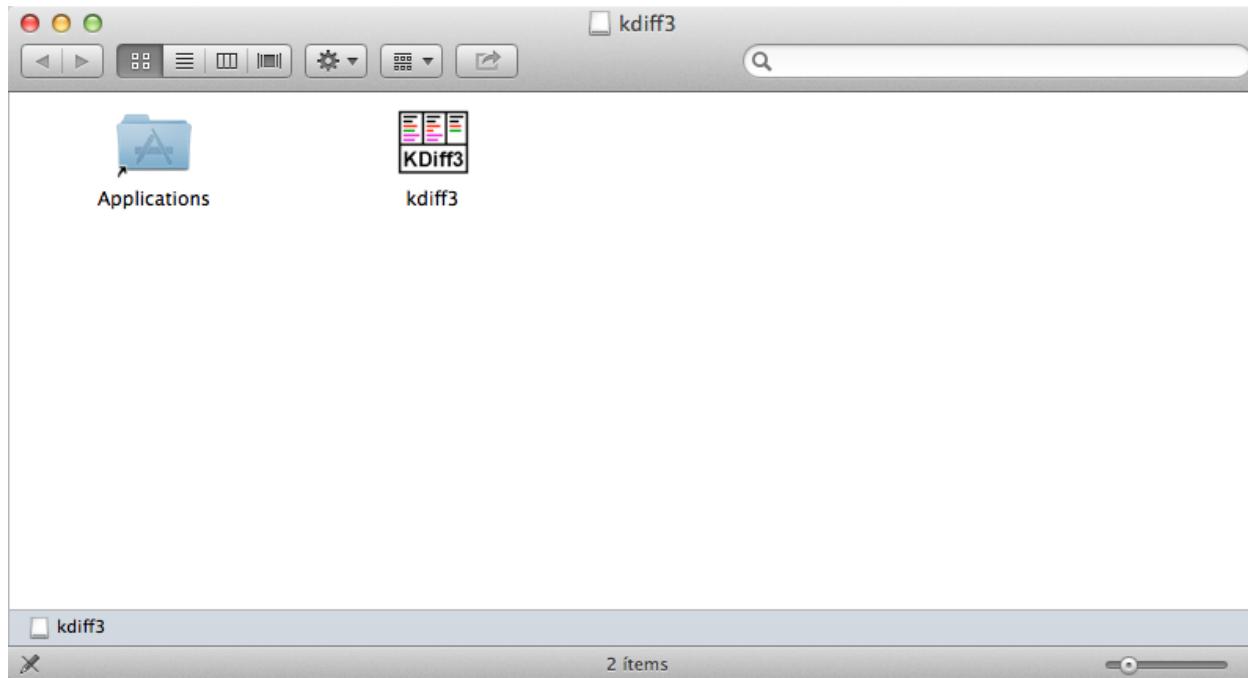








# Instalando Kdiff3 en Mac



Descargando el DMG con la aplicación de  
<http://sourceforge.net/projects/kdiff3/>

# Instalando Kdiff3 en Linux

```
if Ubuntu:  
    print "Instalar desde Ubuntu app directory"  
else:  
    print "Descarga el kdiff3-0.9.96.tar.gz de http://sourceforge.net/project"
```

Descomprime el paquete

```
$ tar -xzf kdiff3-0.9.96.tar.gz
```

Si estás en Debian:

```
$ sudo apt-get install libqt4-dev
```

Si estás en Fedora

```
$ yum install qt-webkit-devel
```

A compilar!

```
$ cd kdiff3-0.9.96/src-QT4  
$ qmake kdiff3.pro  
$ make  
$ make install
```

# Resolviendo conflictos visualmente con Kdiff3

```
$ git config merge.conflictstyle diff3
```

Modificamos el modo de informe de conflictos

```
$ git config merge.tool kdiff3
```

Ponemos kdiff3 como herramienta de merging

```
$ git config --global mergetool.kdiff3.path </path/to/kdiff3>
```

Indicamos la ruta al ejecutable En Mac:

```
/Applications/kdiff3.app/Contents/MacOS/kdiff3
```

Ante un conflicto

```
$ git mergetool
```

Arranca kdiff3



- Parecido a GitHub
- Permite alojar gratis proyectos privados (hasta 5 colaboradores)



# **¡Gracias!**

---

Alberto Casero

[acasero@agbotraining.com](mailto:acasero@agbotraining.com)

@KasAppeal