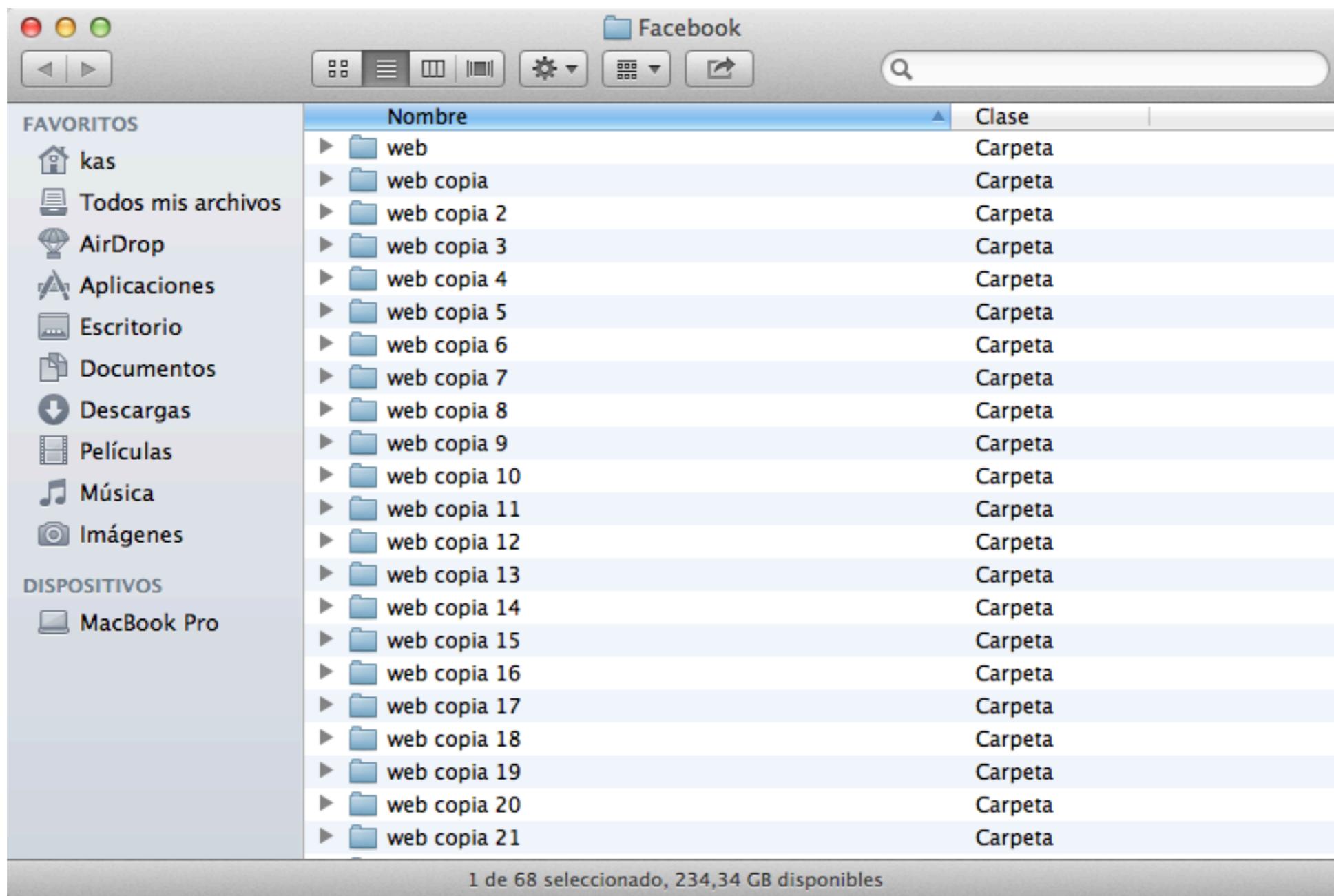


cQué es Git?

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?

- 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 otras personas en GitHub.

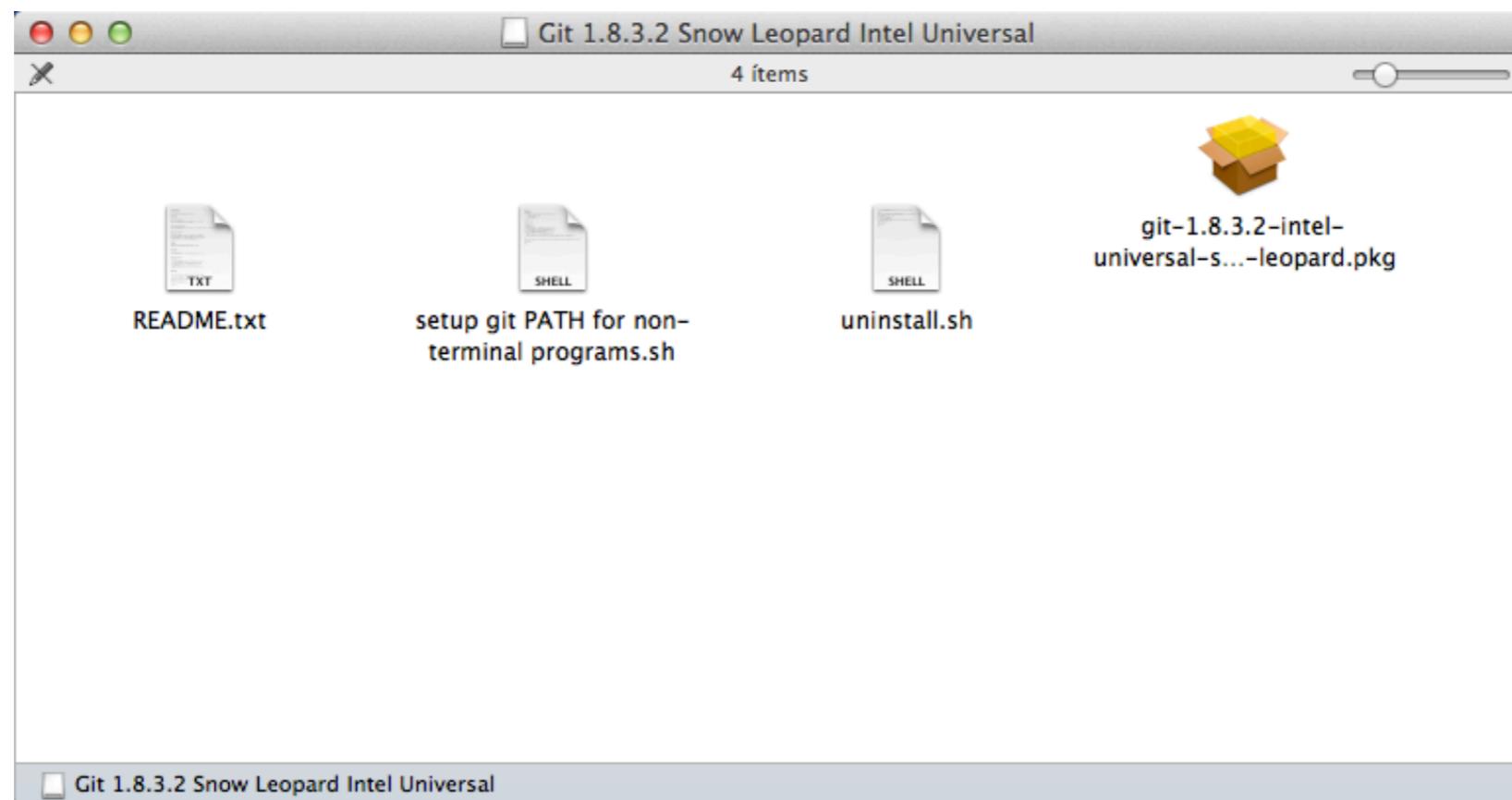
Instalando las herramientas

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>

Configurando git

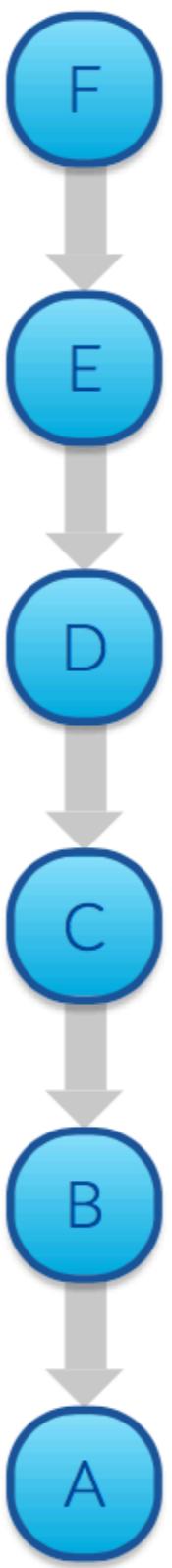
```
# Cambiar la configuración global
$ git config --global user.name "Homer J. Simpson"
$ git config --global user.email "beer@simpsons.com"

# 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



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

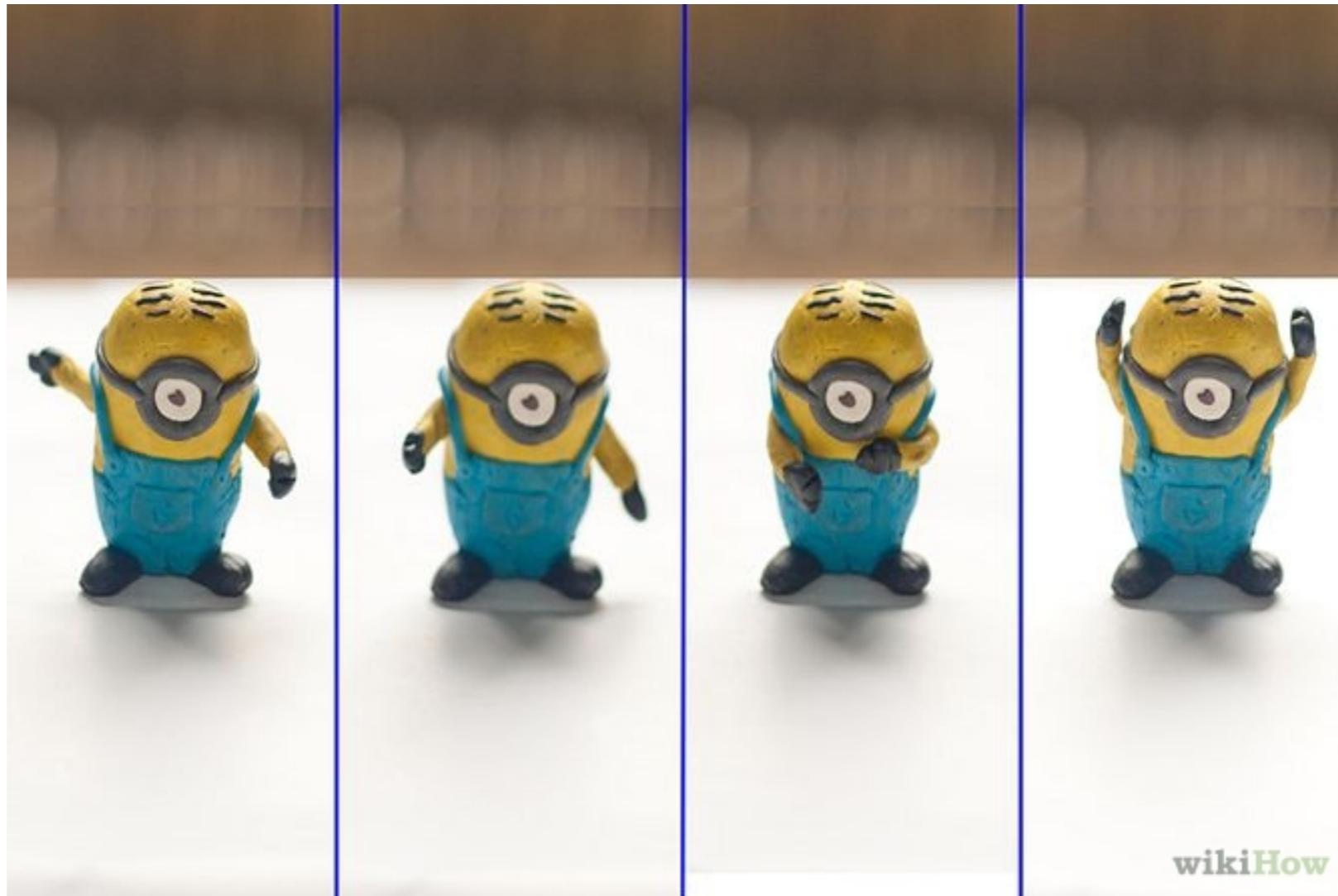
Working Copy

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



wikiHow

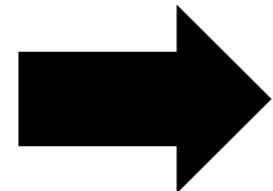
¿Cómo está mi repositorio?

`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>
```

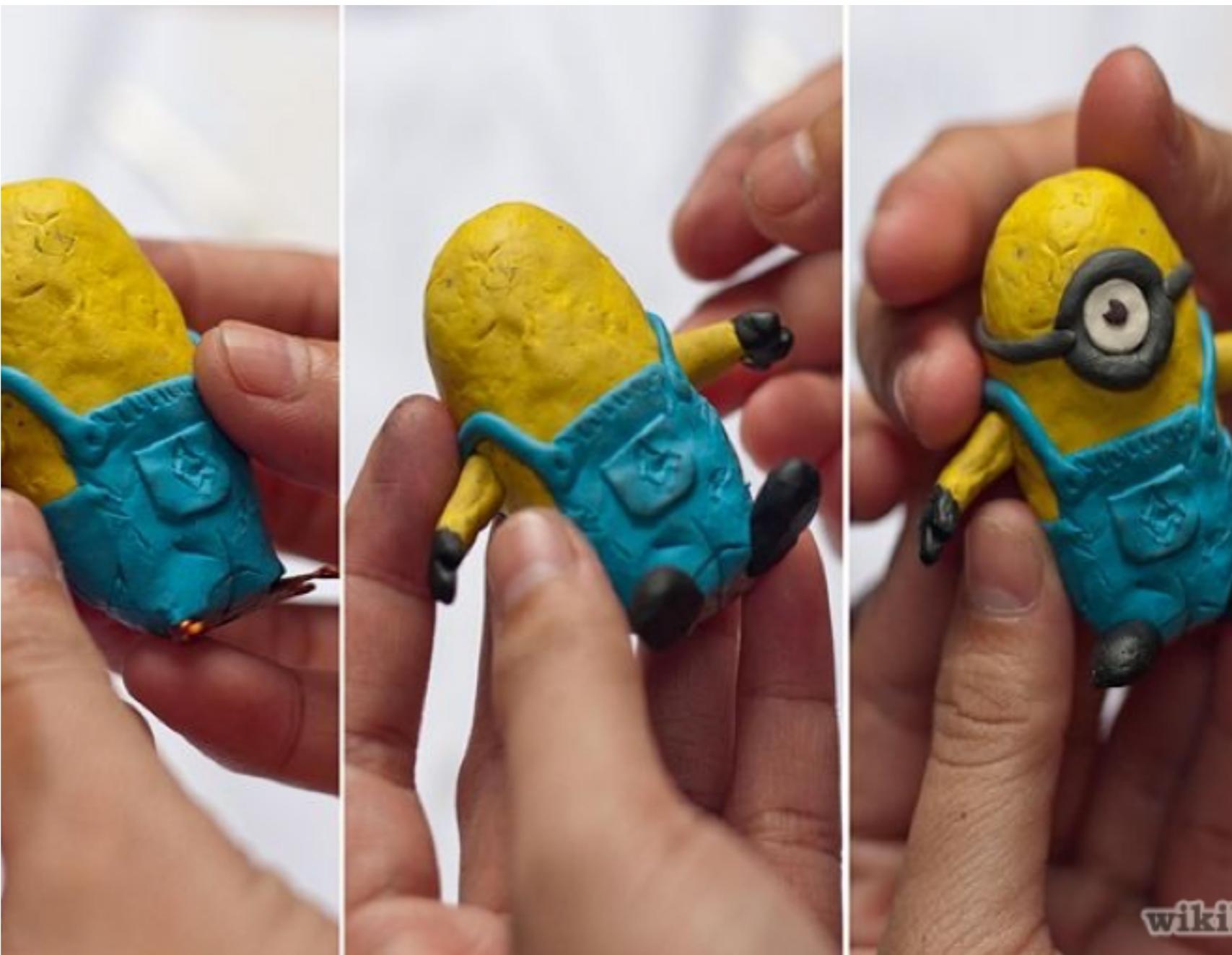
Añade el directorio al **staging area**

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

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

```
$ git add .
```

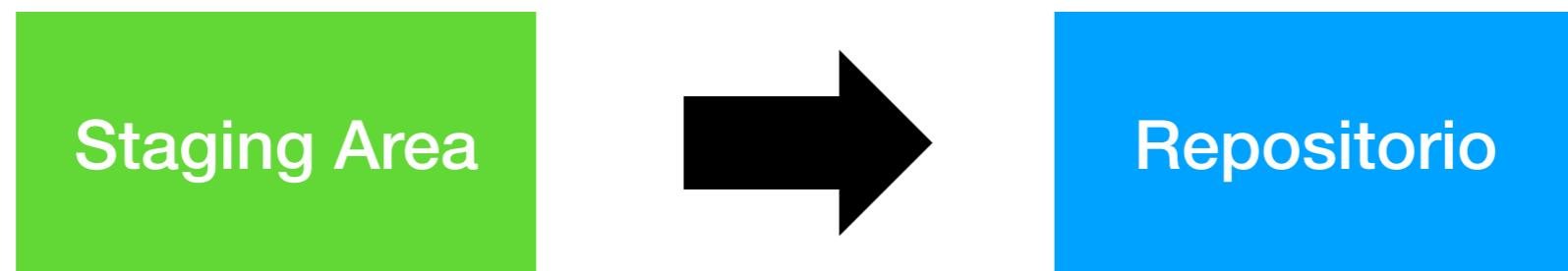
Añade todo lo que haya cambiado al **staging area**



git add minion



De **staging area** al **repositorio**



hacemos un commit

¿Qué es un commit?

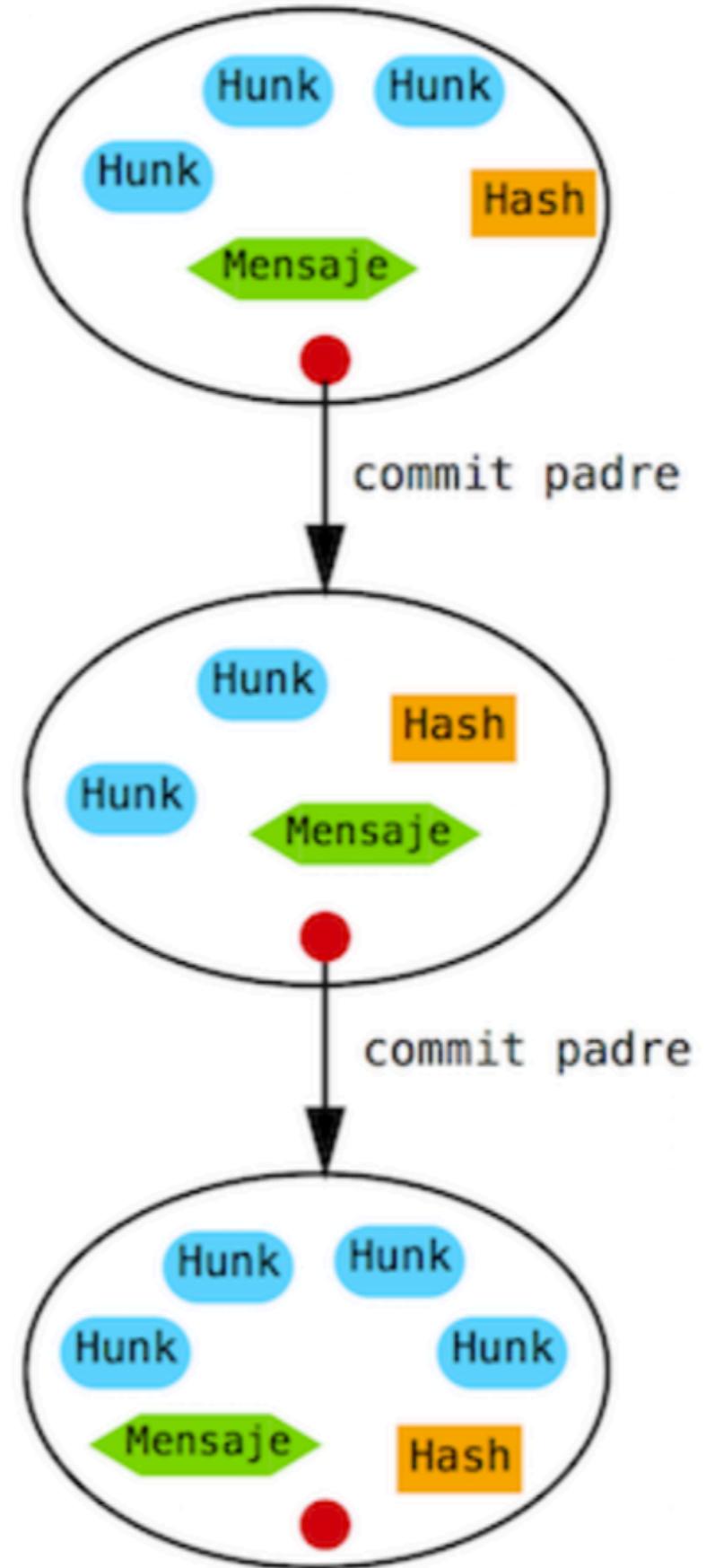


Hacer un commit es como "hacer una foto"

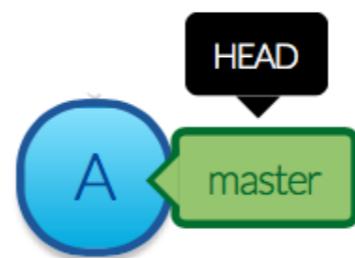
Ok, pero ¿qué es un commit?

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"

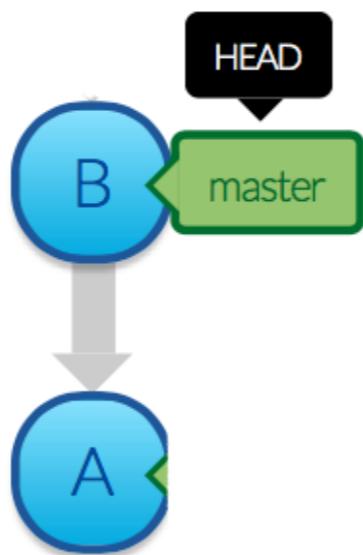


```
$ git add *.c *.h  
$ git commit -m "Primer commit"
```



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

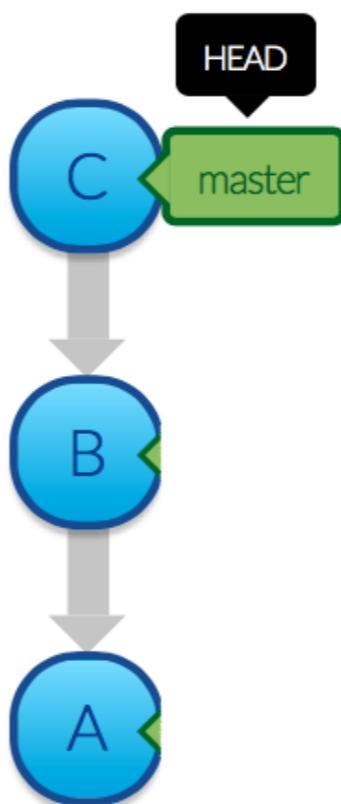
```
$ git add *.c *.h  
$ git commit -m "Segundo commit"
```



```
$ 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?**

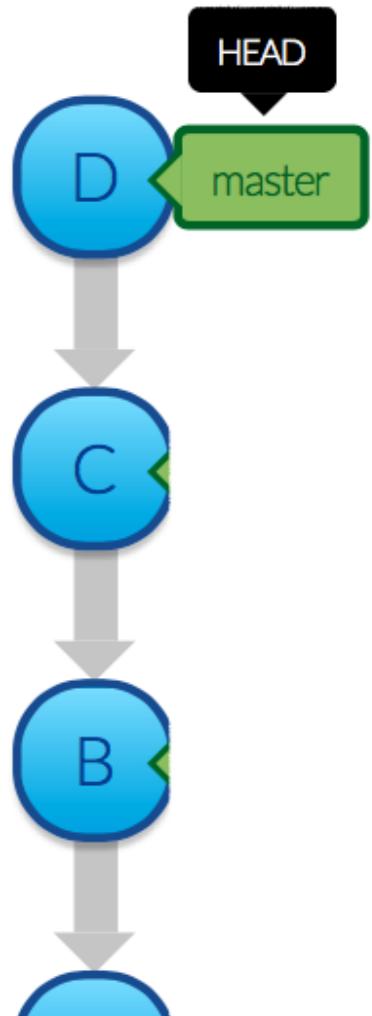
Viendo nuestro log

git log

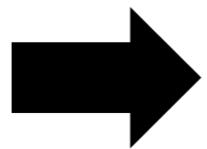
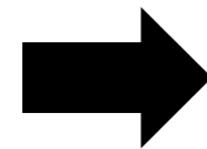
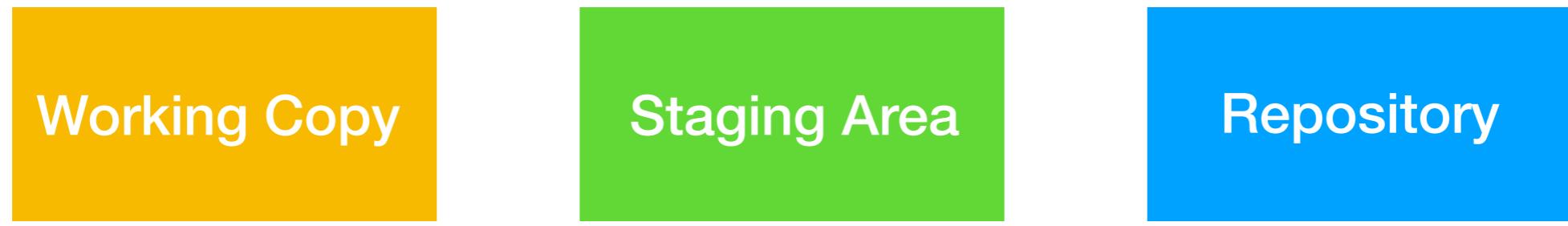
Log de los commits realizados

¿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

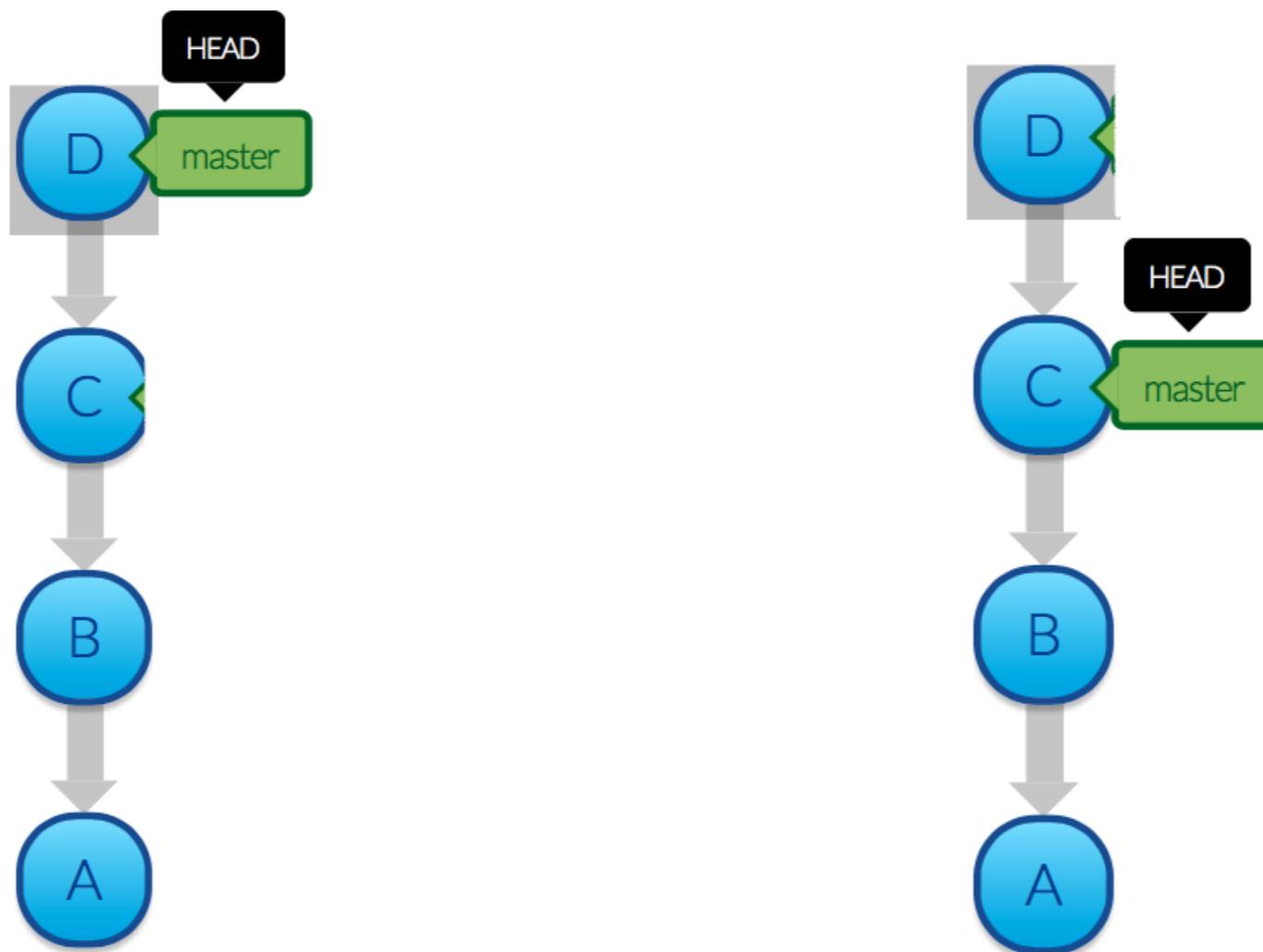


`git add`

`git rm`

`git commit`

Deshaciendo un commit



Deshaciendo el último commit

`git reset`

Dos estrategias

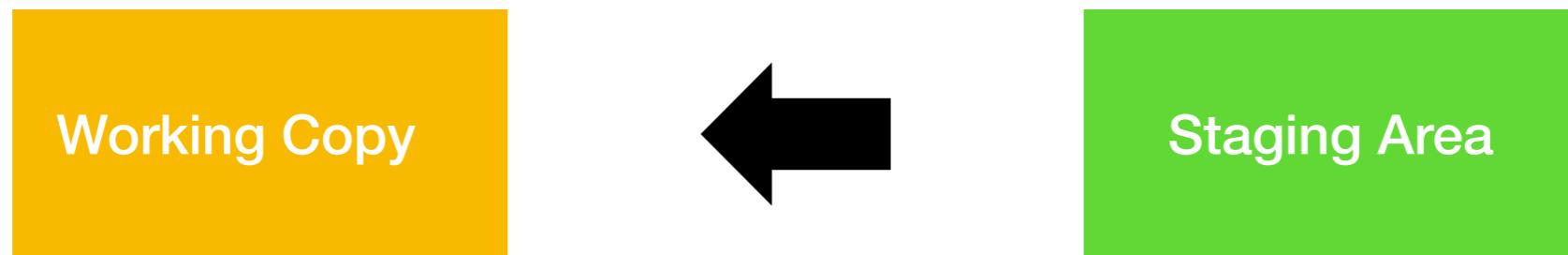
```
$ git reset HEAD~1
```

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.

Deshaciendo un add



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 **stagingarea**



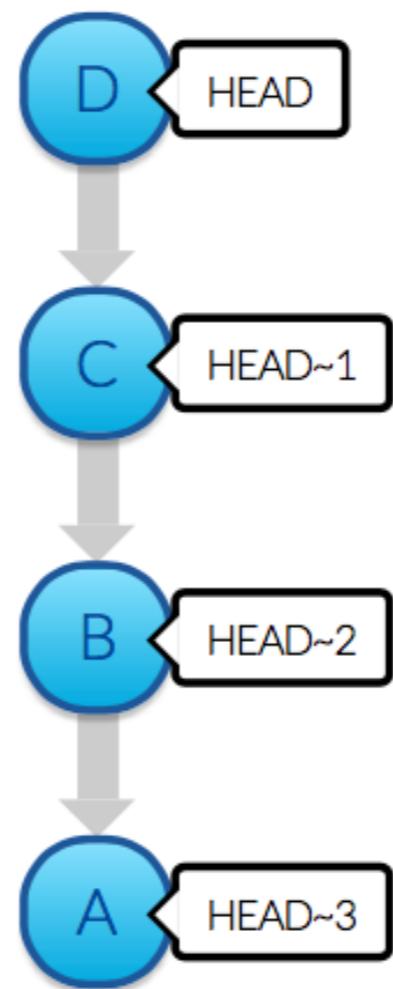
```
$ git add *.c *.h
```



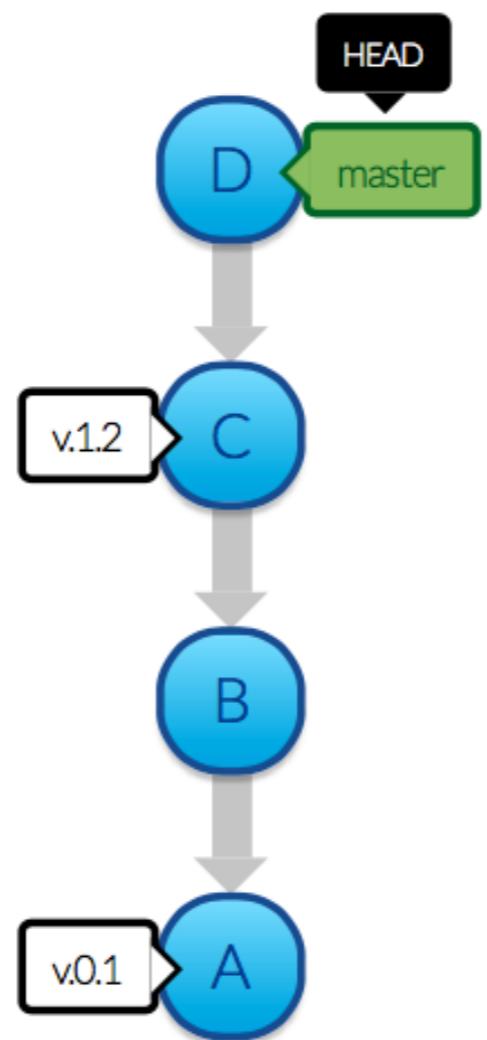
```
$ git add *.c *.h  
$ git reset HEAD *.h
```



¿Qué significa HEAD~1?



Etiquetando commits: tags



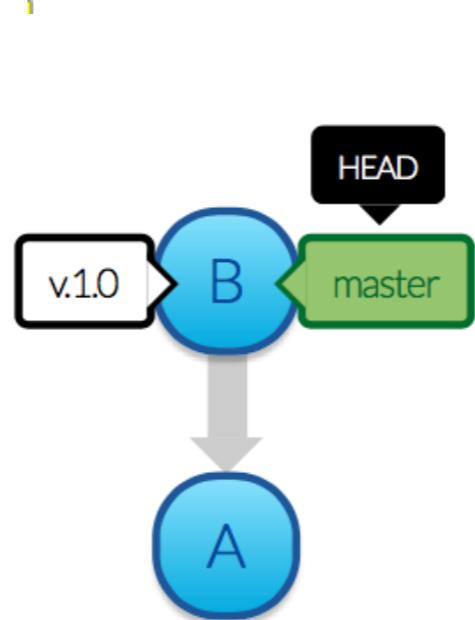
Viendo los tags

git tag

Nos da un listado de los tags del repositorio.

Crear un tag

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



git tag

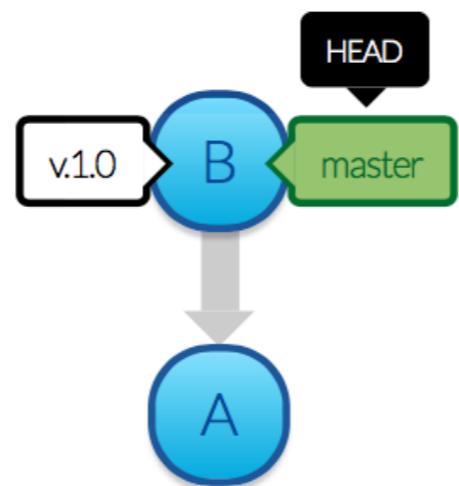
```
$ git tag <tag_name>
```

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

Borrar un tag

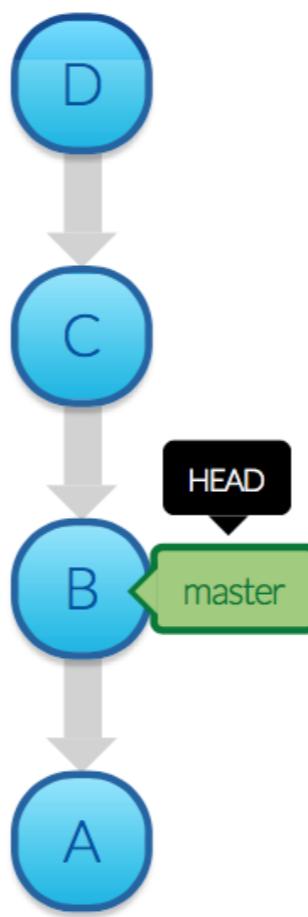
```
$ git tag -d v.1.0
```

Eliminar el tag de nombre <v.1.0>.



Deshaciendo lo "deshecho"

¿Cómo volvemos a D?



Git tiene síndrome de diógenes

`git reflog`

Nos muestra todo lo que ha pasado en nuestro repositorio

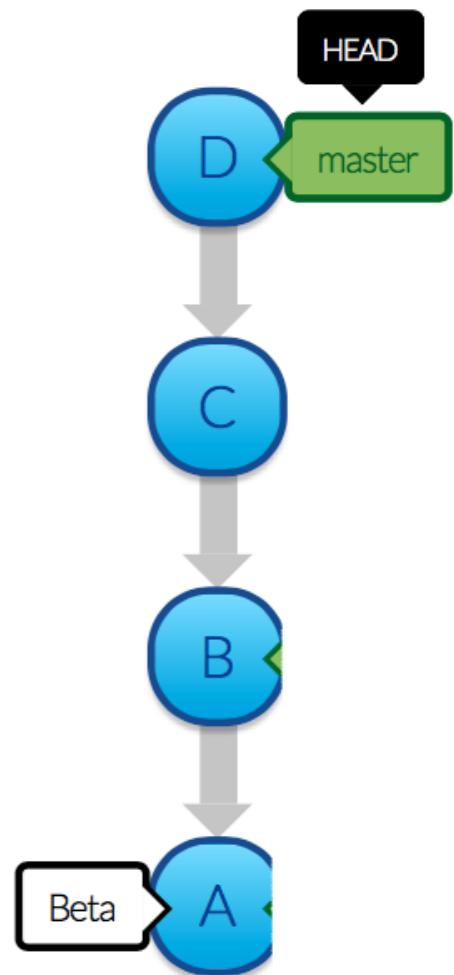
`$ git reflog`

```
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
```

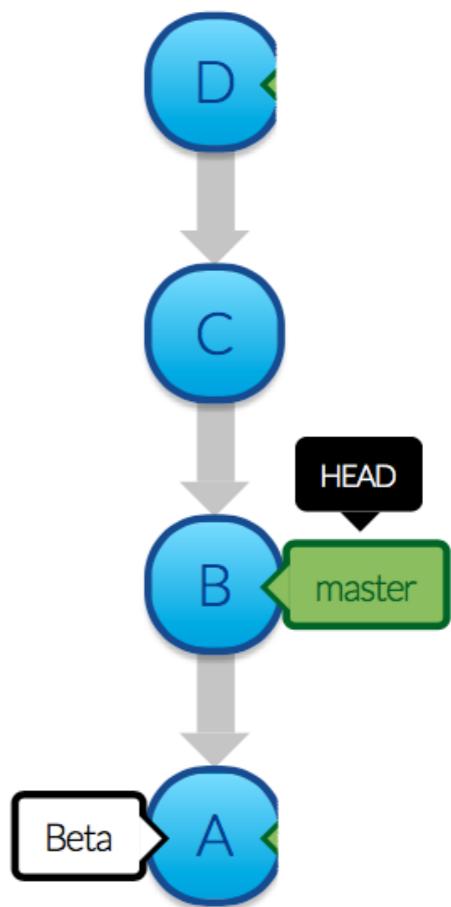
```
fc9dc03 HEAD@{0}: commit: D  
9e7ddad HEAD@{1}: commit: C  
fec3bd0 HEAD@{2}: commit: B  
ac78fe7 HEAD@{3}: commit: A
```



```
$ git reflog
```

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

```
$ git reset fec3bd0
```

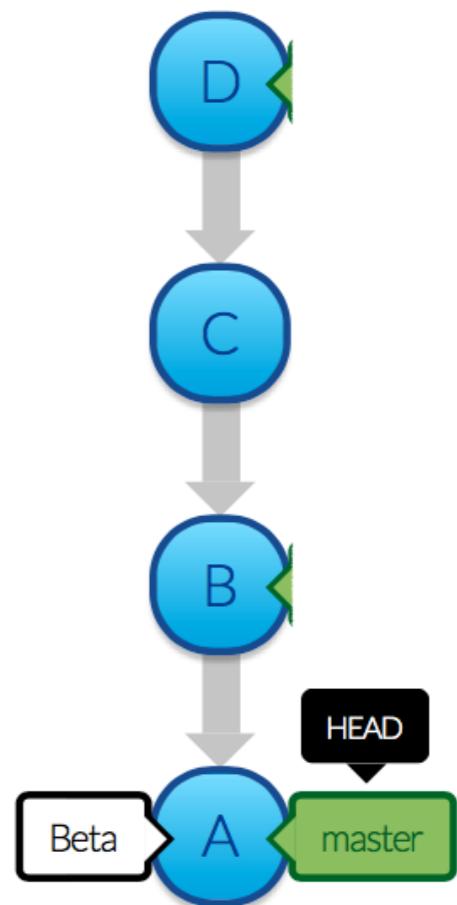


```
$ git reflog
```

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

```
$ git reset fec3bd0
```

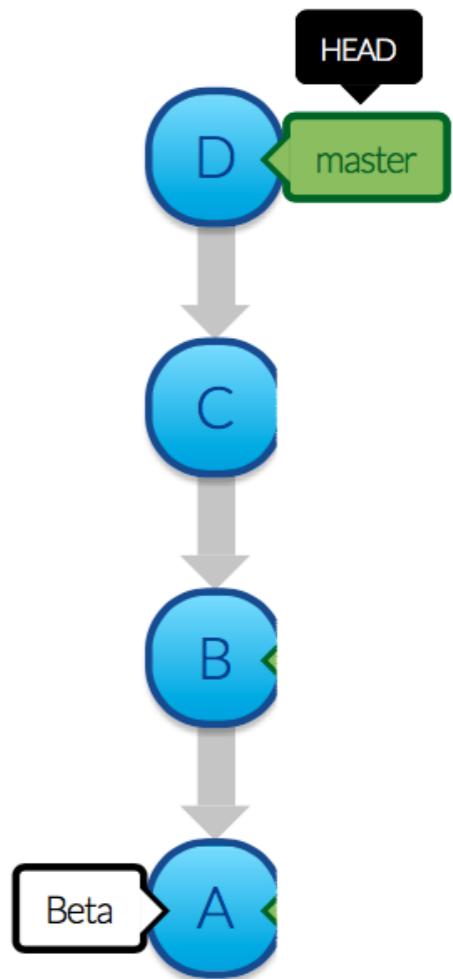
```
$ git reset Beta
```



```
$ git 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

¿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

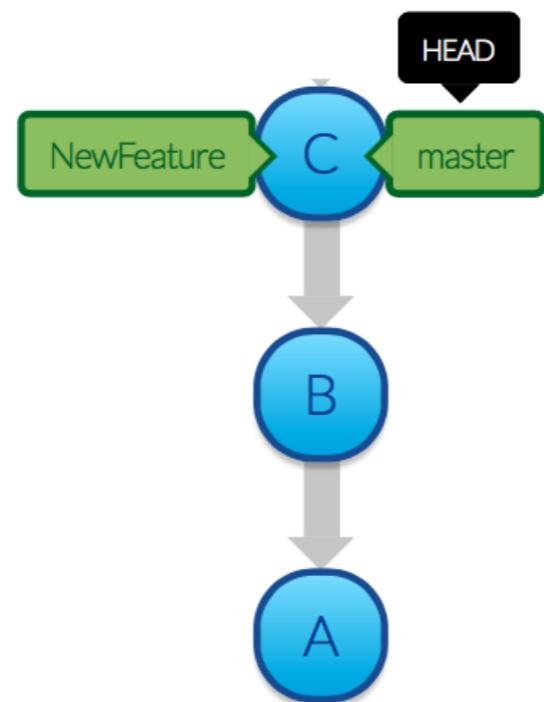
Otra-rama

Valderrama

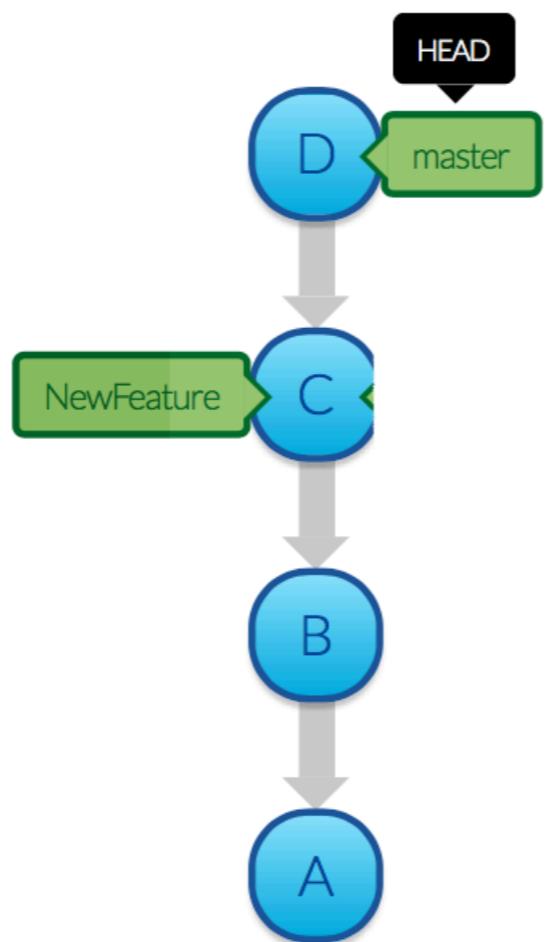
Crear una rama

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

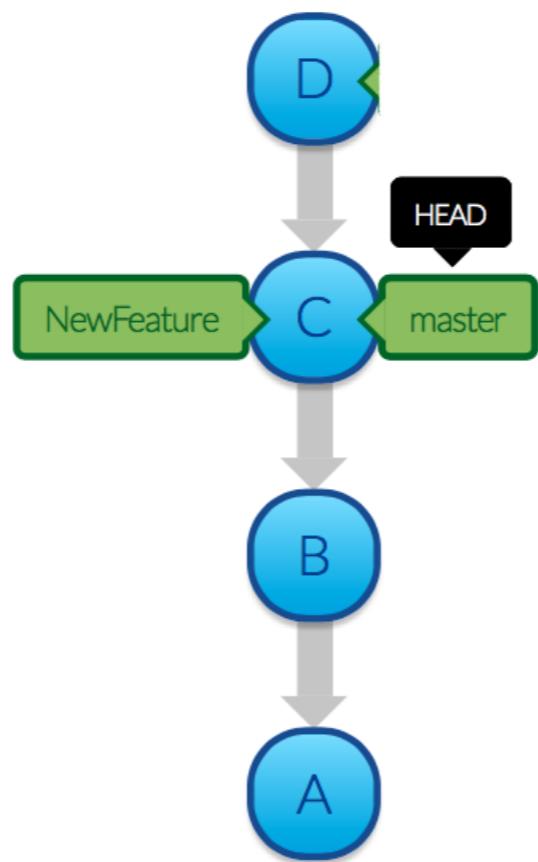
```
$ git branch NewFeature
```



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

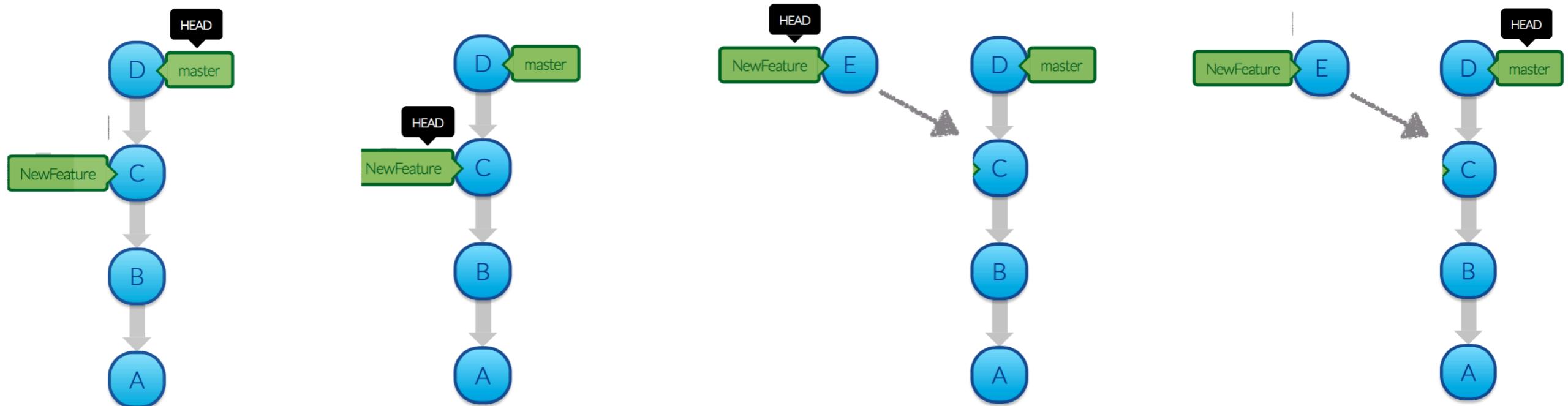


```
$ git commit -m "Nuevo commit"  
$ git reset HEAD~1
```



Cambiar de una rama a otra

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



Cambiar de una rama a otra

`git checkout`

¡El contenido de **working copy** varía!

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

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

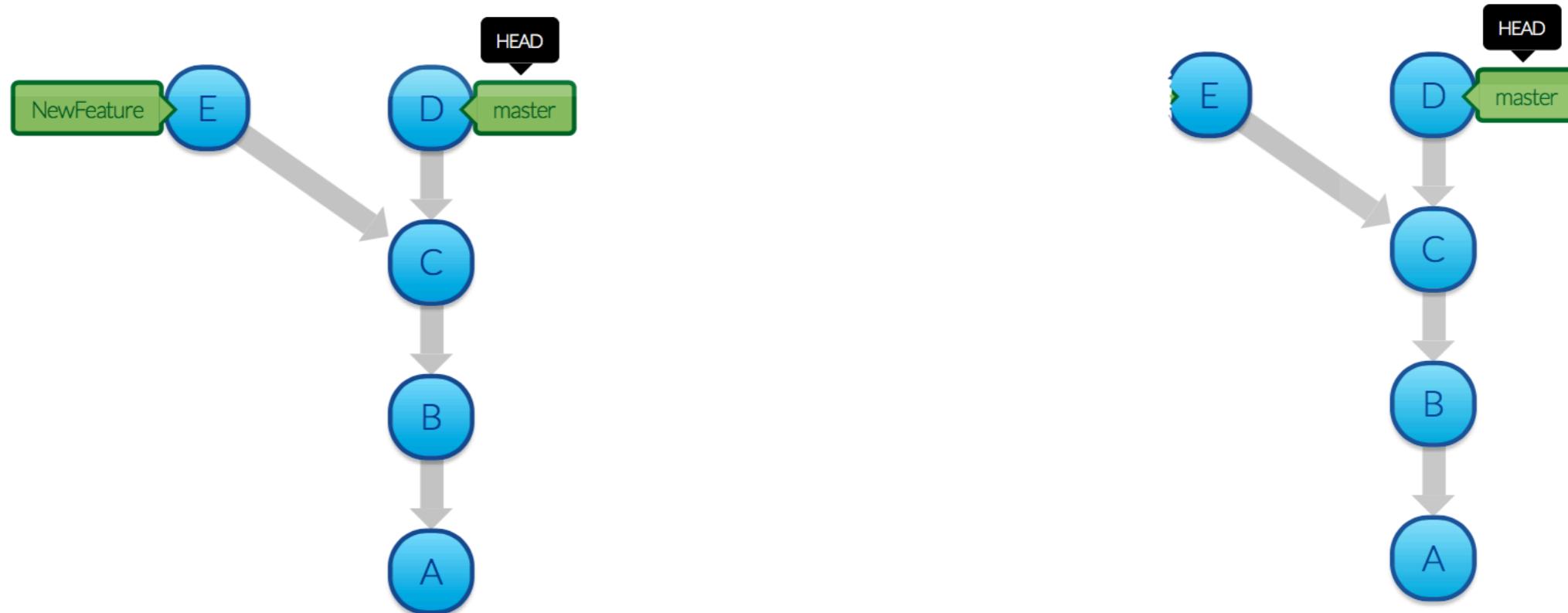
Renombrar una rama

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?

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>
```

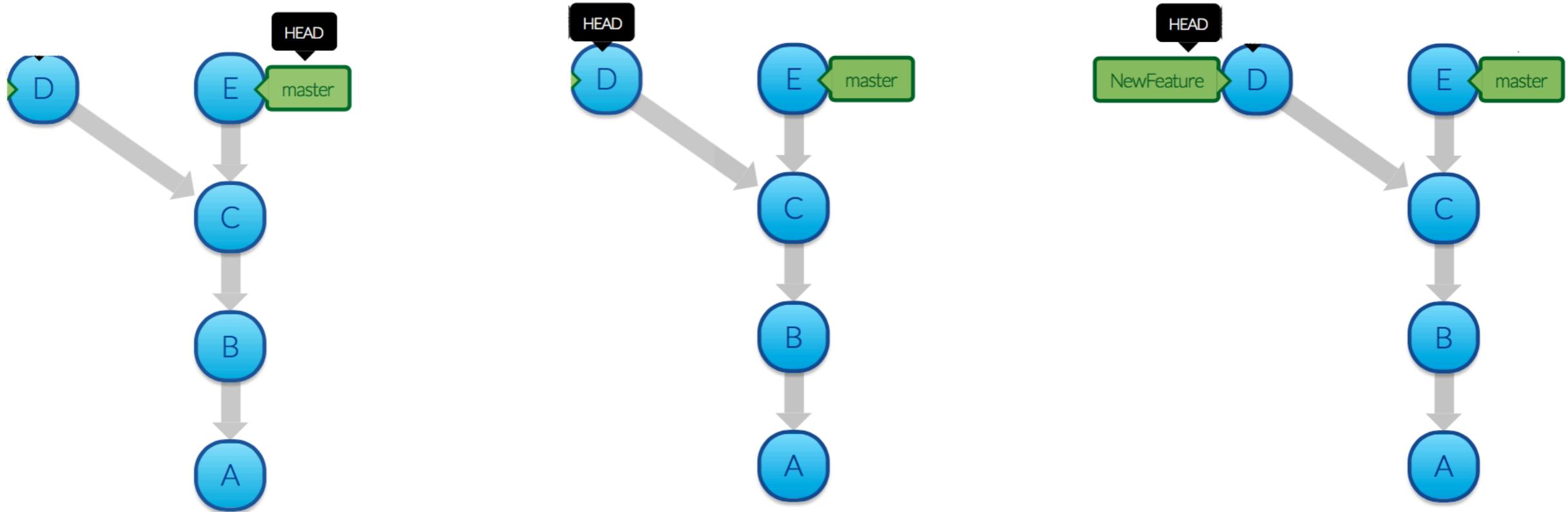
Indicando el nombre de un tag

```
$ git reflog
```

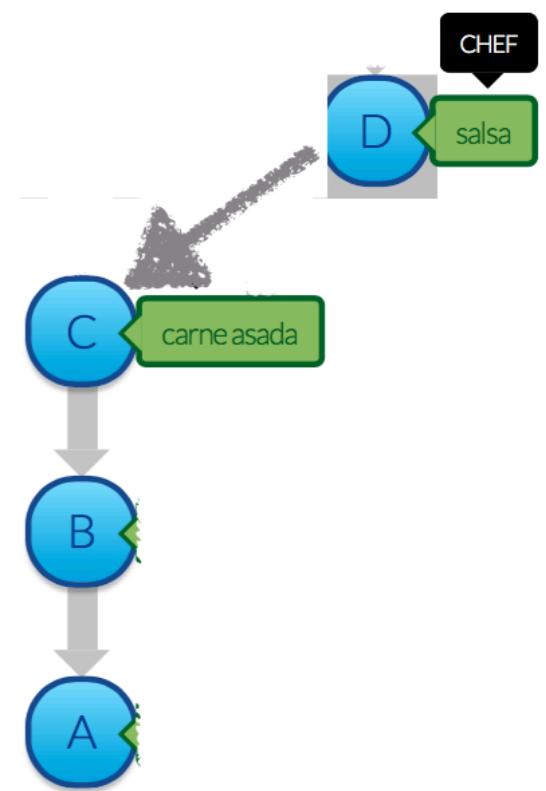
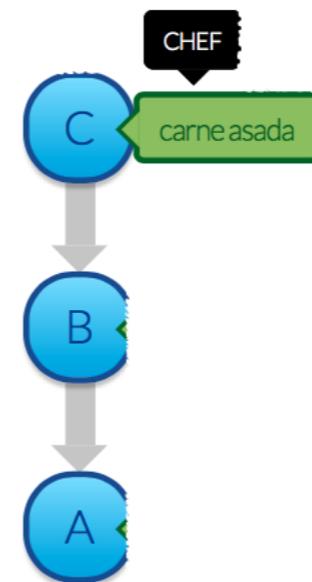
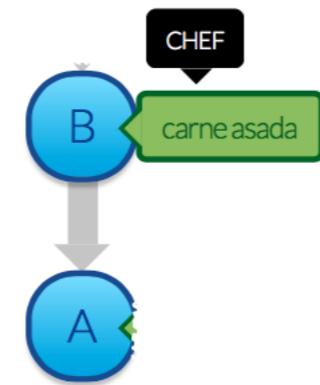
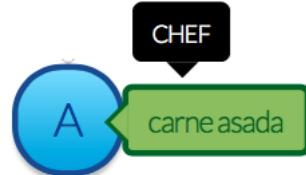
```
158a4da HEAD@{0}: commit: Commit con hash E  
fb62ffa HEAD@{1}: commit: Commit con hash D  
fc9dc03 HEAD@{2}: commit: Commit con hash C  
9e7ddad HEAD@{3}: commit: Commit con hash B  
fec3bd0 HEAD@{4}: commit: Commit con hash A
```

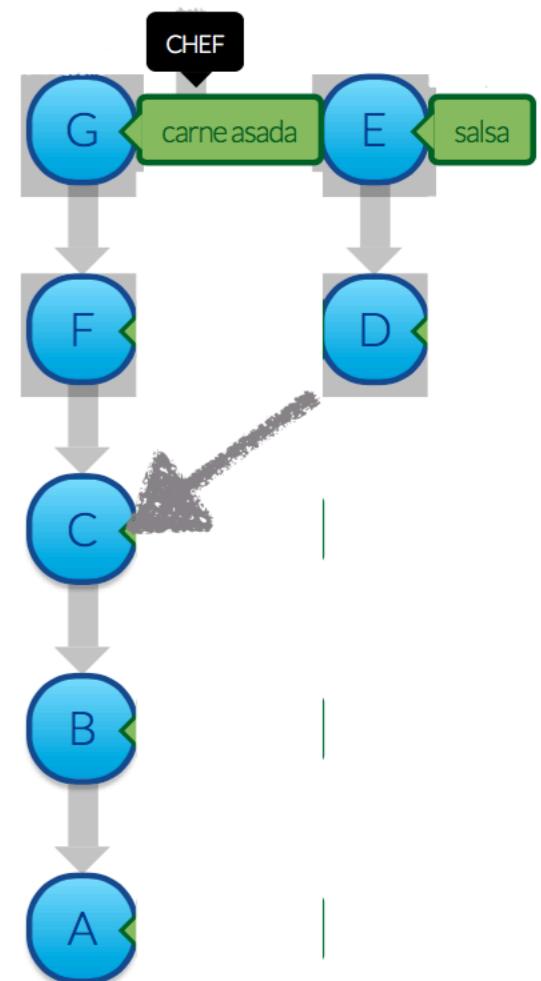
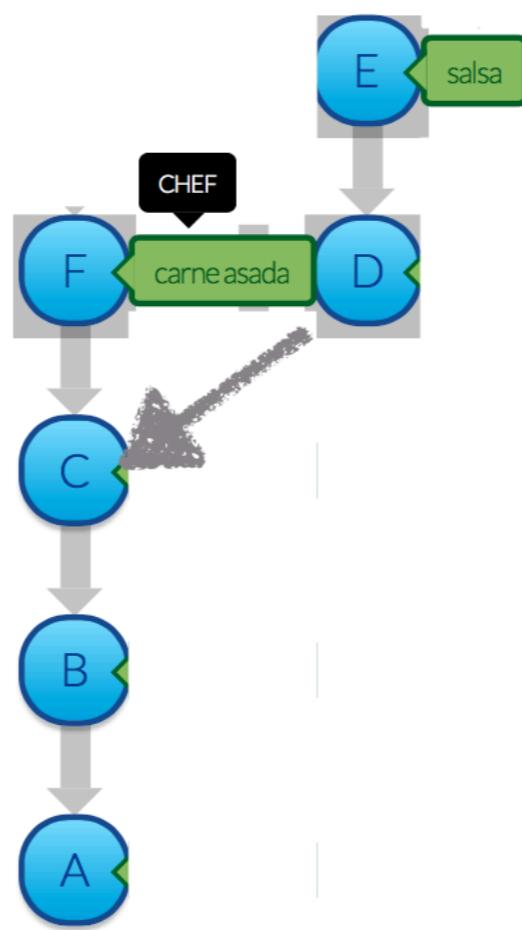
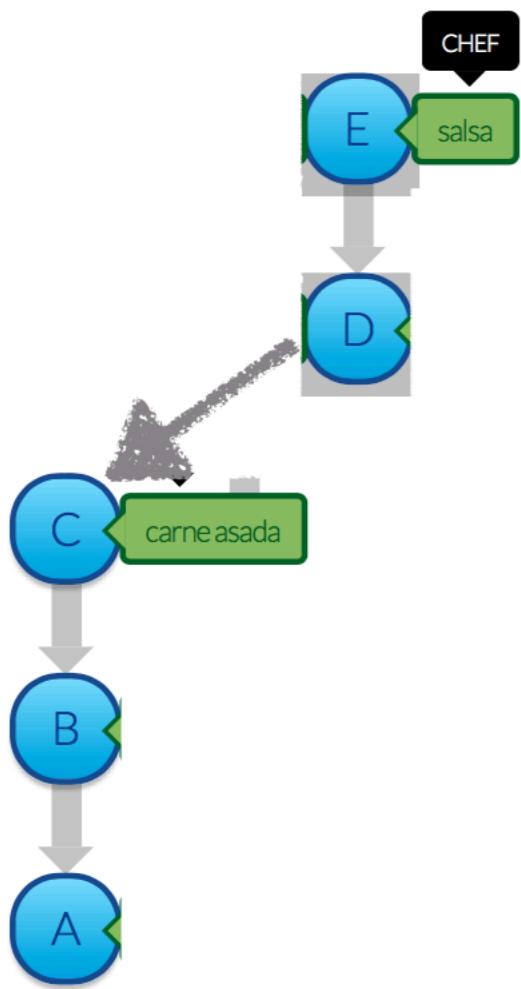
```
$ git checkout fb62ffa
```

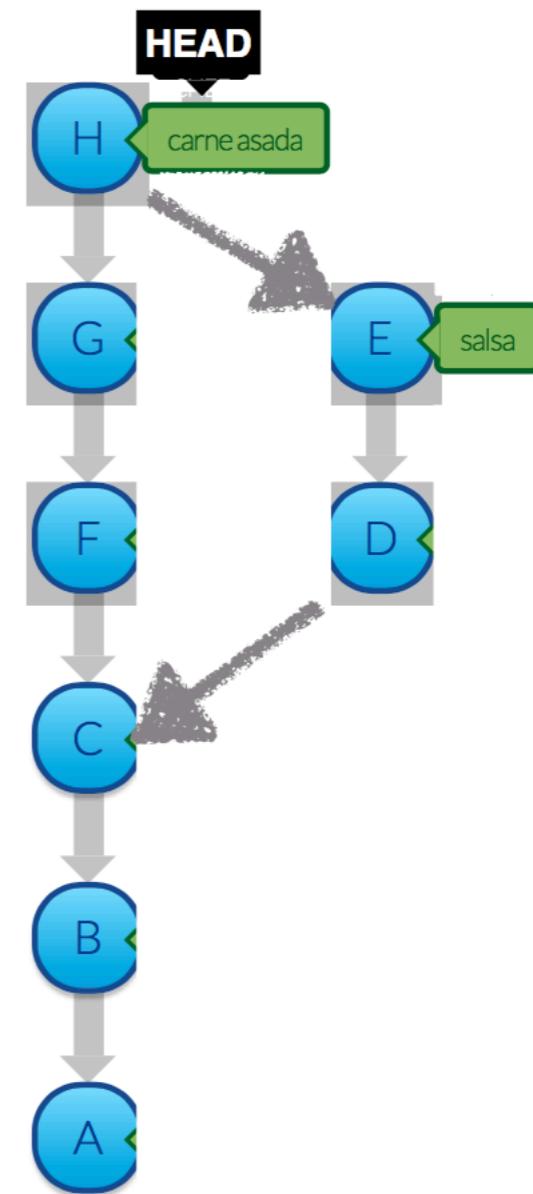
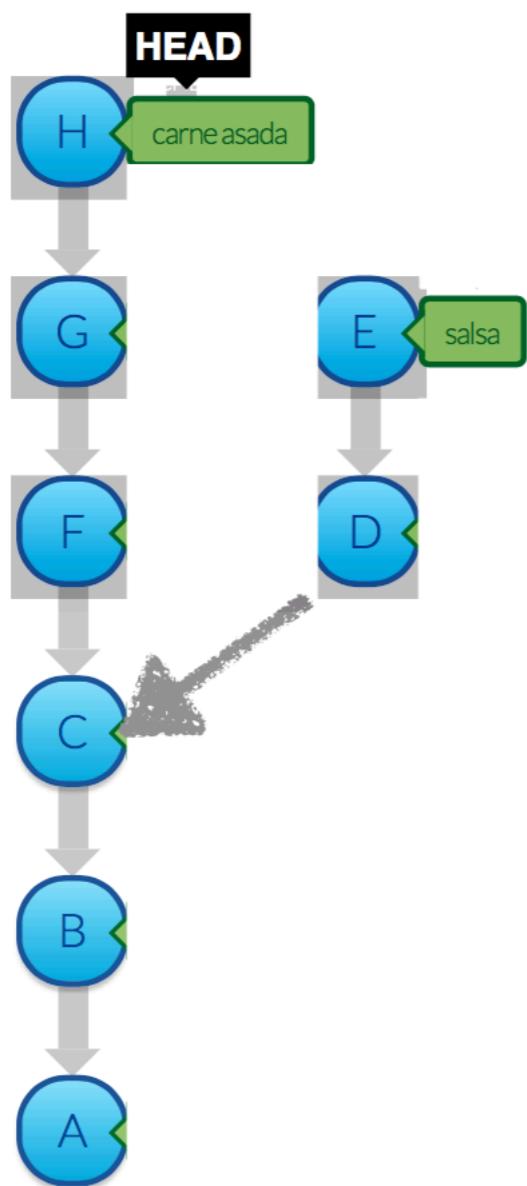
```
$ git branch NewFeature
```



Merging Uniendo branches







Unir dos ramas

`git merge`

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

¿Quién absorbe a quién?

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
```

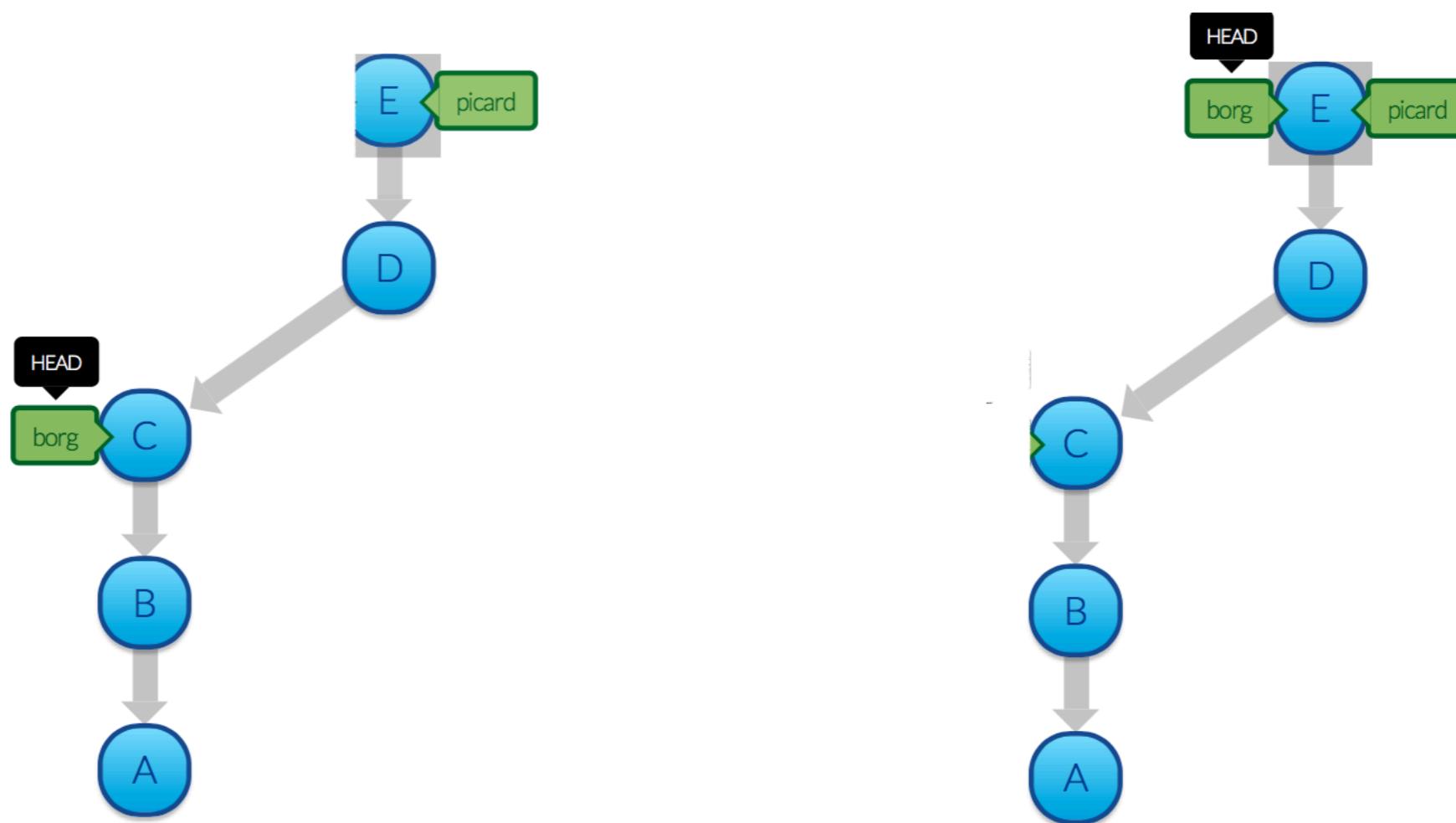
Cuando las ramas forman una lista

Dos opciones:

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

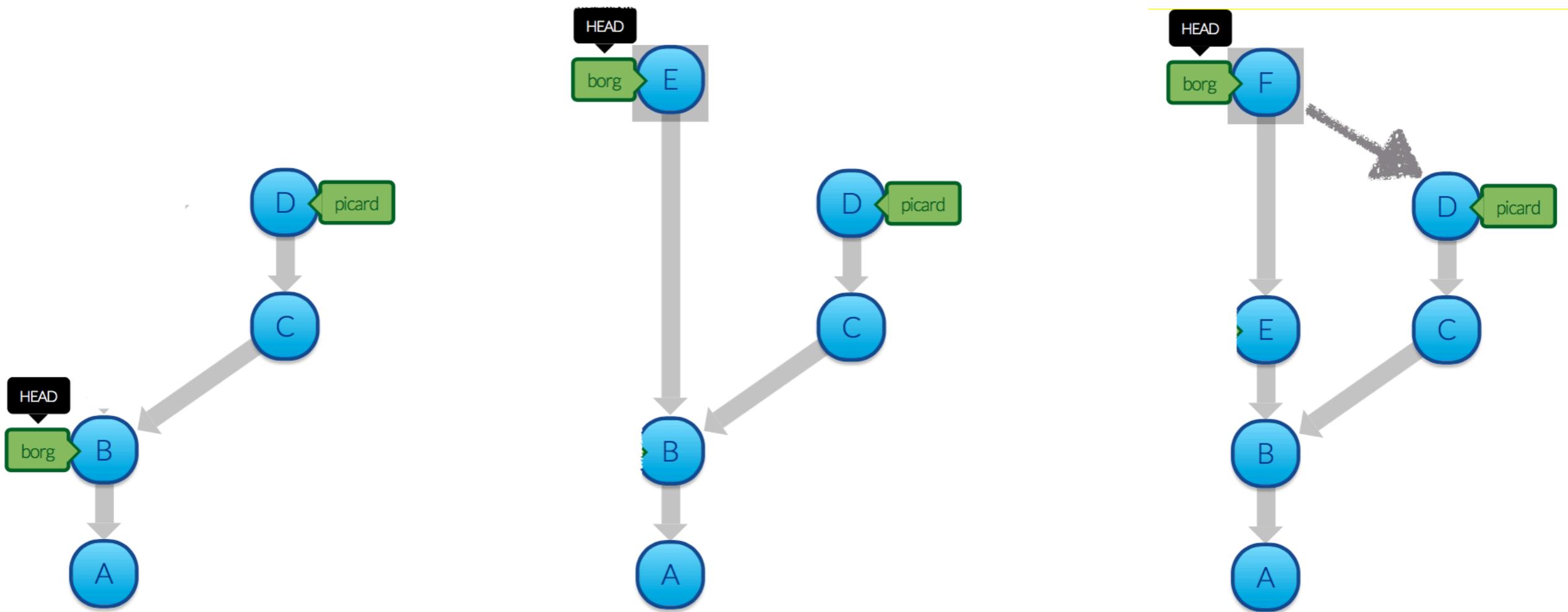
Merge con fast-forward

```
$ git merge picard
```



Merge sin fast-forward

```
$ git merge picard
```



¿Cómo ver los gráficos?

Viendo nuestro 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
```

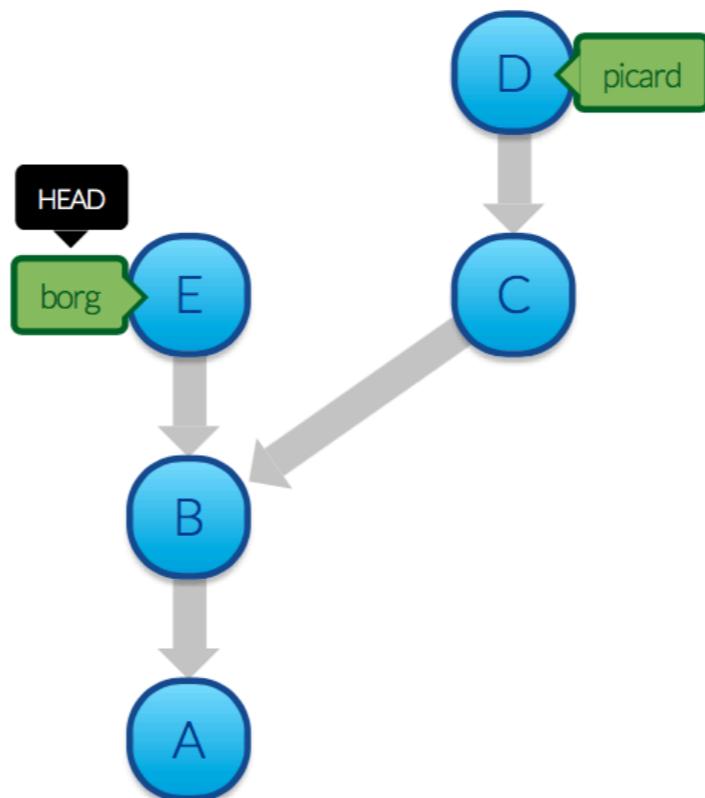
Muestra cada commit resumido en una linea

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

¡Todos juntos!

¿Cómo y cuándo se genera un conflicto?

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



¿Cómo se soluciona?

```
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

Cancela el merge que estábamos haciendo dejando todo como estaba

Repos remotos

¿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).
- Su logo es un “octogato” (Octocat).



¿Cómo empezar?

- Crea una cuenta gratuita en github.com.
- Crea un repositorio
- Clona tu repositorio

Clonar un repositorio

`git clone`

```
$ git clone <repo_url  
$ git clone https://github.com/kasappeal/startrekfaces.git
```

```
Cloning into 'startrekfaces'...  
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.
```

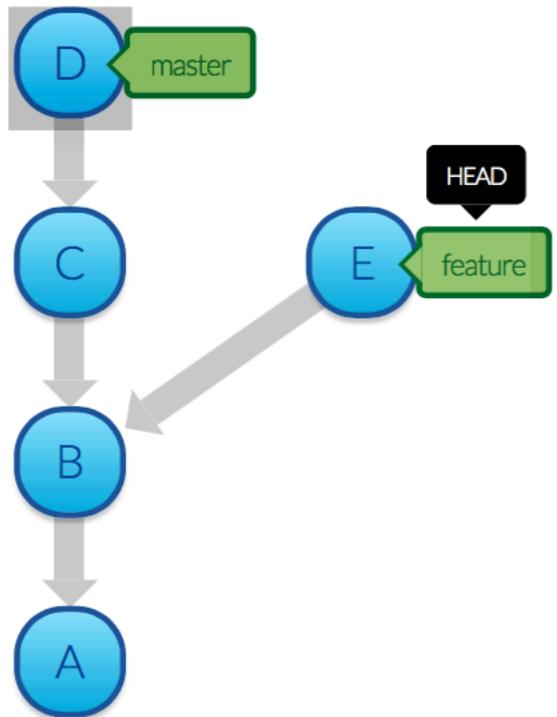
Copiamos un repositorio remoto para trabajar en nuestra máquina
Nos crea una carpeta con el mismo nombre que el repositorio

Listar repositorios remotos de nuestro repo local

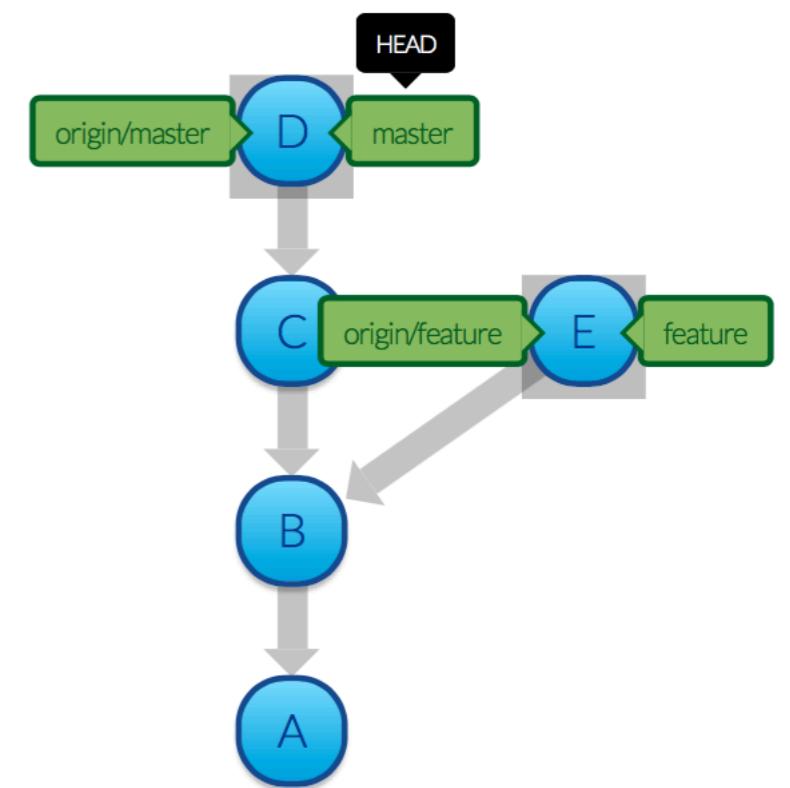
`git remote`

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

Descargar los cambios remotos

git fetch

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

Descarga todos los cambios de 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.

Descargar y aplicar los cambios remotos

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

git pull



Hace un git fetch y un 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.

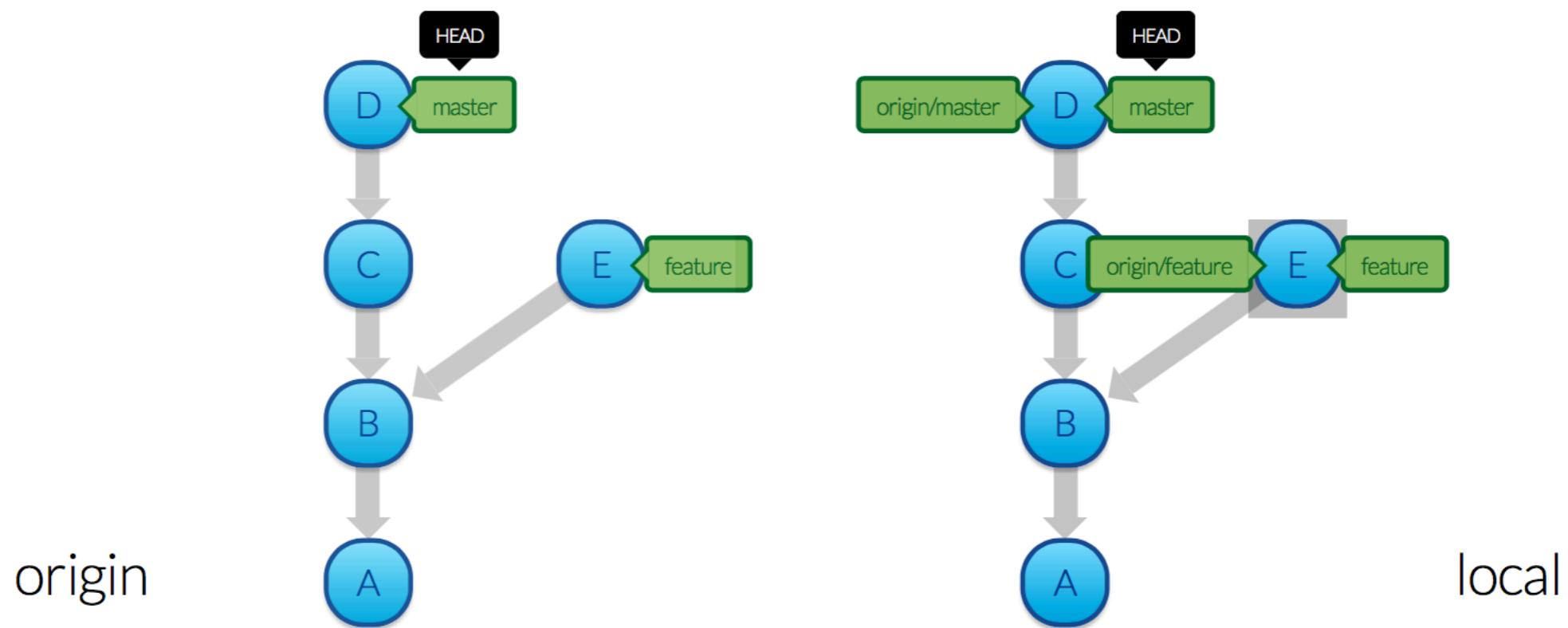
Después, crearla en el servidor

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

¡Hacer lo mismo para los tags!

```
$ git branch feature
```

```
$ git push origin feature
```

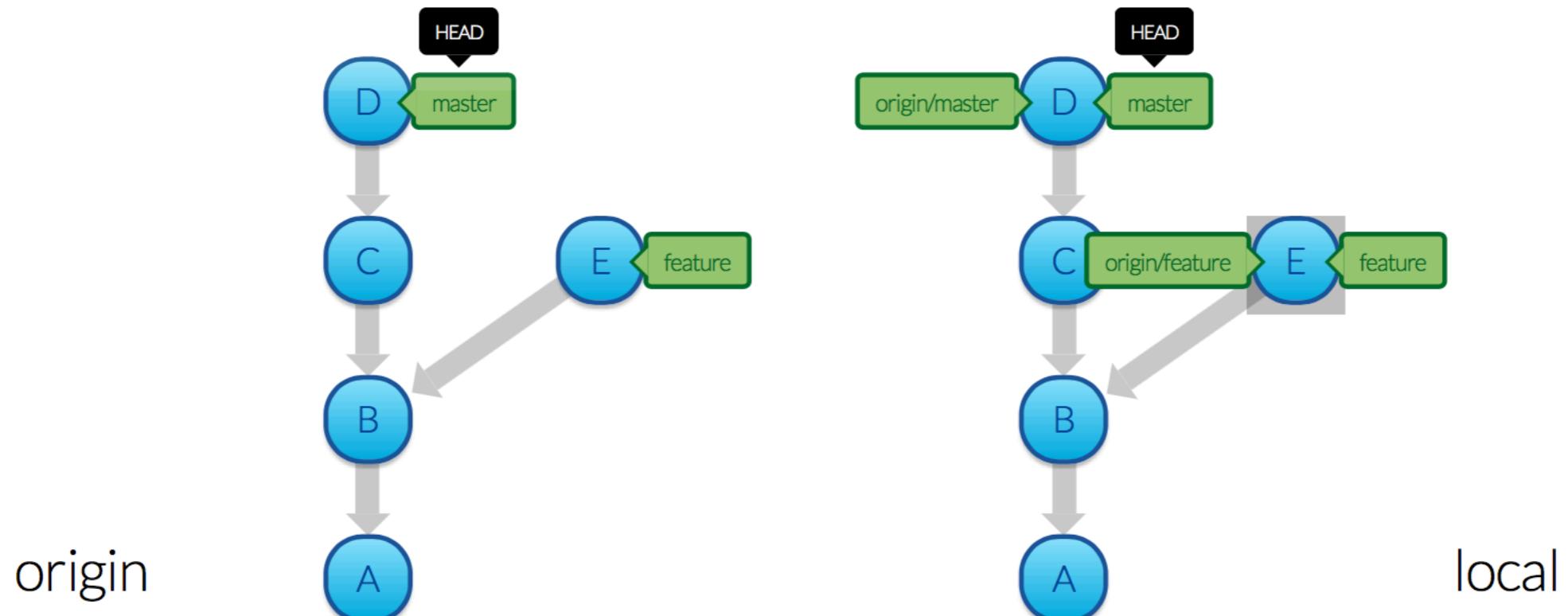


Borrar una rama en el servidor

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

¡Hacer lo mismo para los tags!

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



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.

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

# Archivos comprimidos
*.dmg
*.zip

# Logs y BBDD
*.log
*.sql

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

COMANDOS PARA SUBIR LOS REPOSITORIOS A GITHUB

Para enlazar el repositorio local con el que hemos creado en GitHub

git remote add origin https://github.com/usuario/nombre-repositorio.git

Para subir la rama master a GitHub

git push -u origin master

Si tuviésemos más ramas

git push -u origin otraRama

Para subir los tags a GitHub

git push --tags