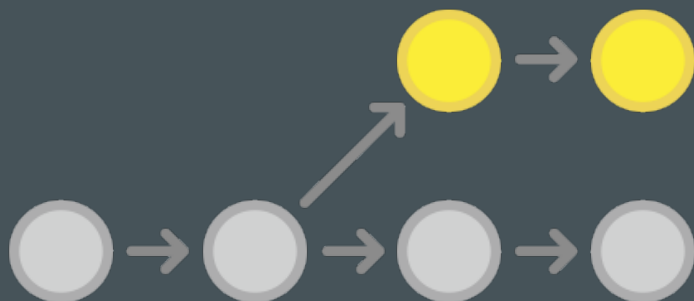




HERRAMIENTA DE CONTROL DE VERSIONES EN ENTORNOS COLABORATIVOS



Soledad Valero



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

CAMPUS DE GANDIA



ÍNDICE

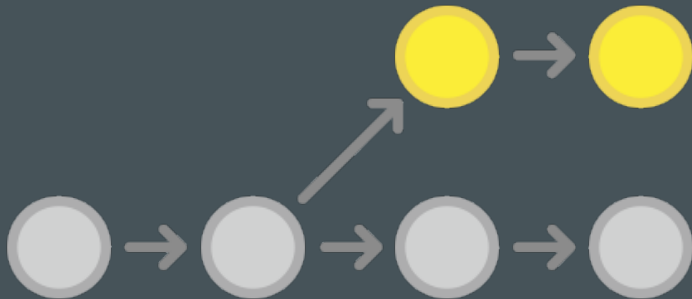
- ¿Quiero usar git! ¿Por donde empiezo?
- Principales comandos colaborativos
- Comandos del día a día
- Resumen de comandos
- Bibliografía



¡QUIERO USAR GIT A FONDO!

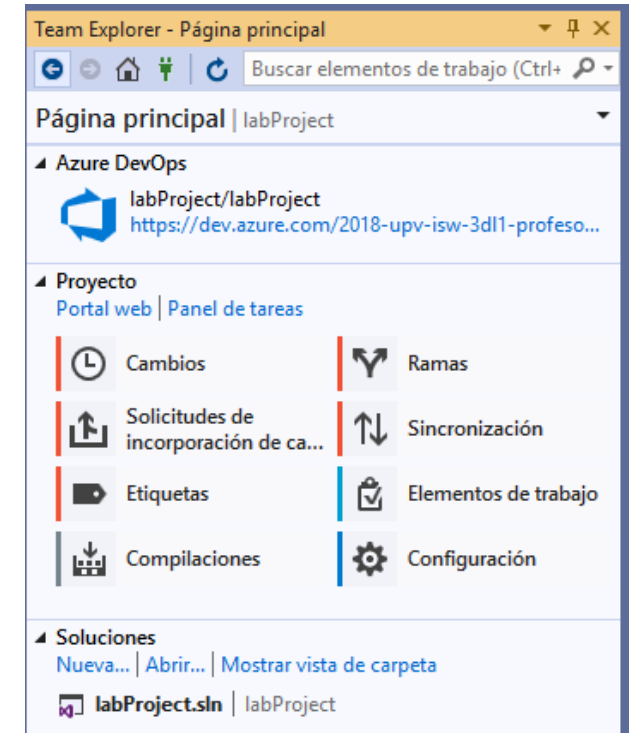
¿POR DONDE EMPIEZO?

INSTALACIÓN



ENTORNO GRÁFICO EN VISUAL STUDIO

- Visual Studio nos ofrece una interfaz gráfica para realizar las principales tareas a realizar en git:
 - Clonar un repositorio (git clone)
 - Guardar cambios (git commit)
 - Tareas de sincronización: Recuperar y extraer confirmaciones de entrada (git fetch y git pull)
 - Insertar confirmaciones de salida (git push)
- Git ofrece muchas más funcionalidades, pero se necesita instalar la herramienta



APROVECHAR GIT AL MÁXIMO: **INSTALAR HERRAMIENTA CONSOLA**

- Es posible instalar Git en Linux, MacOS y Windows
- La versión Linux es la más fácil de usar
- Instalar en Linux:

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

- Instalar en ArchLinux

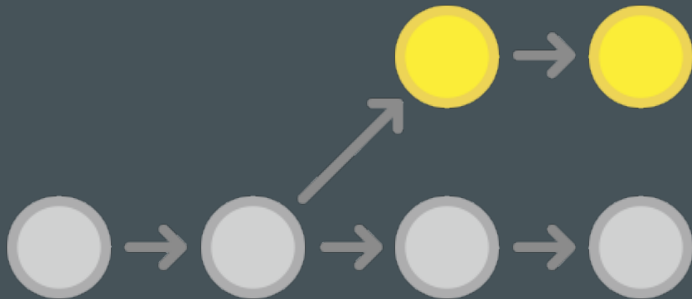
```
$ sudo pacman -S git
```

- Windows instalar el ejecutable
- MacOS lanza el instalador

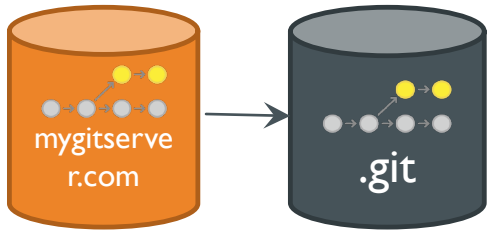


PRINCIPALES COMANDOS COLABORATIVOS

CLONAR – SUBIR CAMBIOS - ACTUALIZAR



CLONAR UN REPOSITORIO



git clone

- Obtener una copia de un repositorio Git existente:
 - Crea un nuevo directorio
 - Dentro se tendrá un repositorio local (.git) que tendrá una copia del historial de commits y ramas del repositorio remoto
 - Archivos con el estado HEAD del repositorio

```
$ git clone https://direccionRepo/repo.git
```



CLONAR UN REPOSITORIO:

Ejercicio

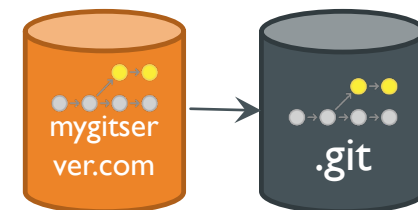
1. Situar en el directorio de destino, por ejemplo tmp:

```
$ cd /tmp
```

2. Clonar el repositorio, añadiendo al comando la url copiada, pulsando de nuevo el botón central del ratón

```
$ git clone https://github.com/programacion2GTI/sesion_github.git
```

```
epsg@archlinux:~/tmp
[epsg@archlinux tmp]$ git clone https://github.com/programacion2GTI/sesion_github.git
Cloning into 'sesion_github'...
remote: Counting objects: 13, done.
remote: Compressing objects: 100% (11/11), done.
remote: Total 13 (delta 2), reused 4 (delta 1), pack-reused 0
Unpacking objects: 100% (13/13), done.
```



git clone



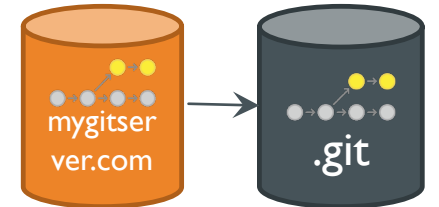
CLONAR UN REPOSITORIO:

Ejercicio

3. ¿Qué tenemos?

```
$ ls
```

```
epsg@archlinux:~/tmp
[epsg@archlinux tmp]$ ll
total 24
-rw-r--r-- 1 epsg epsg 178 nov 3 15:56 mainEchoHTTP-1.js
-rw-r----- 1 epsg epsg 284 nov 3 16:06 mainEchoHTTP-2.js
drwxr-xr-x 4 epsg epsg 4096 feb 7 10:51 node_modules
-rw-r--r-- 1 epsg epsg 296 feb 7 10:51 package-lock.json
drwxr-xr-x 3 epsg epsg 4096 ene 28 12:21 prac_gti
drwxr-xr-x 4 epsg epsg 4096 feb 9 19:56 sesion_github
[epsg@archlinux tmp]$
```



git clone



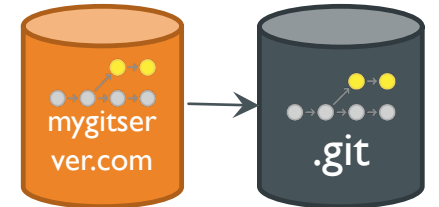
CLONAR UN REPOSITORIO:

Ejercicio

3. ¿Qué tenemos?

```
$ ls
```

```
epsg@archlinux:~/tmp
[epsg@archlinux tmp]$ ll
total 24
-rw-r--r-- 1 epsg epsg 178 nov 3 15:56 mainEchoHTTP-1.js
-rw-r----- 1 epsg epsg 284 nov 3 16:06 mainEchoHTTP-2.js
drwxr-xr-x 4 epsg epsg 4096 feb 7 10:51 node_modules
-rw-r--r-- 1 epsg epsg 296 feb 7 10:51 package-lock.json
drwxr-xr-x 3 epsg epsg 4096 ene 28 12:21 prac_gti
drwxr-xr-x 4 epsg epsg 4096 feb 9 19:56 sesion_github
[epsg@archlinux tmp]$
```



git clone



CLONAR UN REPOSITORIO:

Ejercicio

6. Moveros al directorio para ver que hay dentro:

```
epsg@archlinux:~/tmp/sesion_github
[epsg@archlinux tmp]$ cd sesion_github/
[epsg@archlinux sesion_github]$ ls
LICENSE  README.md  src
[epsg@archlinux sesion_github]$ git log
commit 2b513f7df06734ce315c65f715cdca90f58d3d2a
Author: svalero <svalero@dsic.upv.es>
Date:   Fri Feb 9 19:27:57 2018 +0100

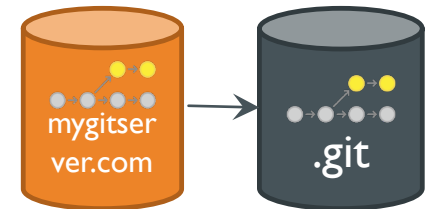
    Src folder for code created

commit 2be81b1f6cb7ec020db92451daf93342c670da5f
Author: programacion2GTI <36307023+programacion2GTI@users.noreply.github.com>
Date:   Fri Feb 9 18:43:15 2018 +0100

    Initial files added

commit 28a4009d8c2d5b0eb8ea756f5e26eb7d84e201bf
Author: programacion2GTI <36307023+programacion2GTI@users.noreply.github.com>
Date:   Fri Feb 9 18:41:26 2018 +0100

    Initial commit
[epsg@archlinux sesion_github]$
```



git clone

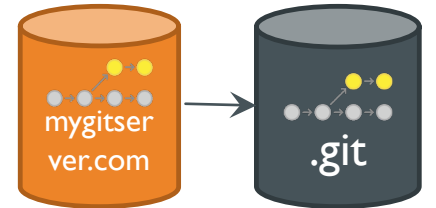


CLONAR UN REPOSITORIO:

Ejercicio

- Tenemos una copia exacta del repositorio remoto:

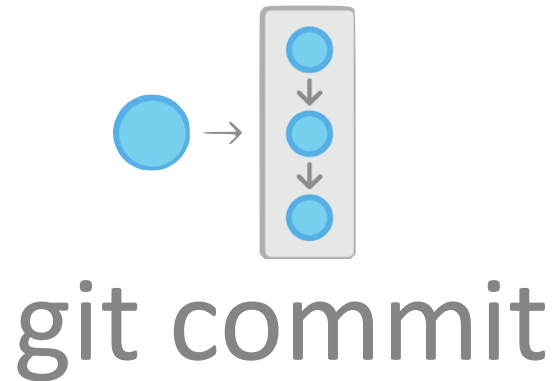
Código + Historial de commits



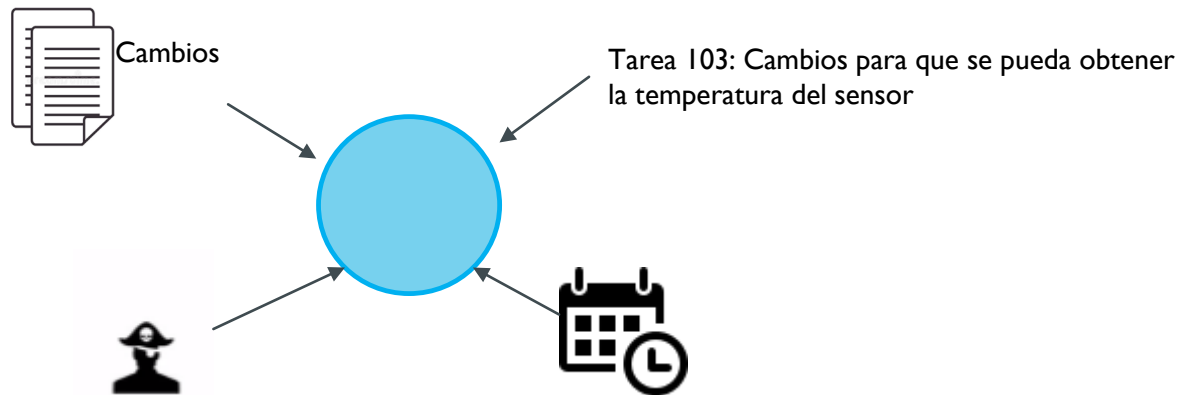
git clone



CONFIRMAR TUS CAMBIOS



- Comando para confirmar/enviar tus cambios al repositorio git:
 - Añadir la foto de los cambios preparados para confirmar al repositorio
 - Asociar a estos cambios un mensaje que describa que cambios se han hecho
 - Se añade quién ha hecho la confirmación
 - Se añade cuándo se ha hecho.



CONFIRMAR TUS CAMBIOS:

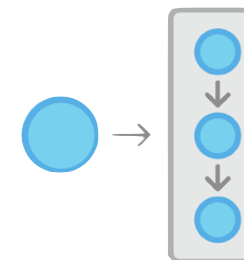
Ejercicio

1. Vamos a crear nuestra primer commit (foto), para que se quede registrado que tenemos un nuevo fichero en nuestro proyecto.

```
$ git commit -a -m "Creado el hola mundo"
```

2. Veamos ahora que nos dice git de su estado:

```
$ git status
```



git commit



CONFIRMAR TUS CAMBIOS:

```
[epsg@archlinux prac_gti]$ git commit -a -m "Creado el hola mundo"
[master (root-commit) b06d27c] Creado el hola mundo
 1 file changed, 1 insertion(+)
 create mode 100644 holaMundo.js
[epsg@archlinux prac_gti]$ git status
On branch master
nothing to commit, working tree clean
[epsg@archlinux prac_gti]$ █
```

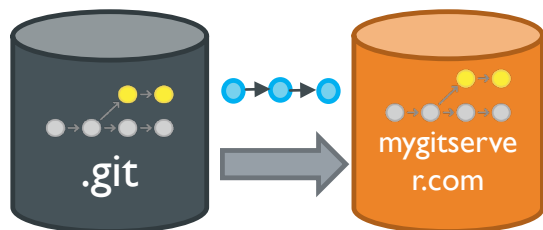


CONFIRMAR TUS CAMBIOS:

```
[epsg@archlinux prac_gti]$ git commit -a -m "Creado el hola mundo"
[master (root-commit) b06d27c] Creado el hola mundo
 1 file changed, 1 insertion(+)
 create mode 100644 holaMundo.js
[epsg@archlinux prac_gti]$ git status
On branch master
nothing to commit, working tree clean
[epsg@archlinux prac_gti]$
```



SUBIR CONFIRMACIONES AL REPOSITORIO REMOTO



git push

- Sube las confirmaciones realizadas en el repositorio local al remoto
 - Se requiere acceso de escritura al otro repositorio: pide usuario y contraseña

```
$ git push
```



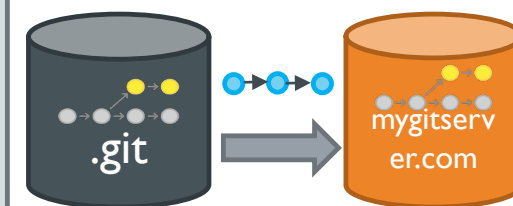
SUBIR CONFIRMACIONES AL REPOSITORIO REMOTO:

Ejercicio

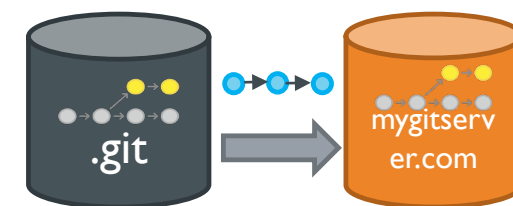
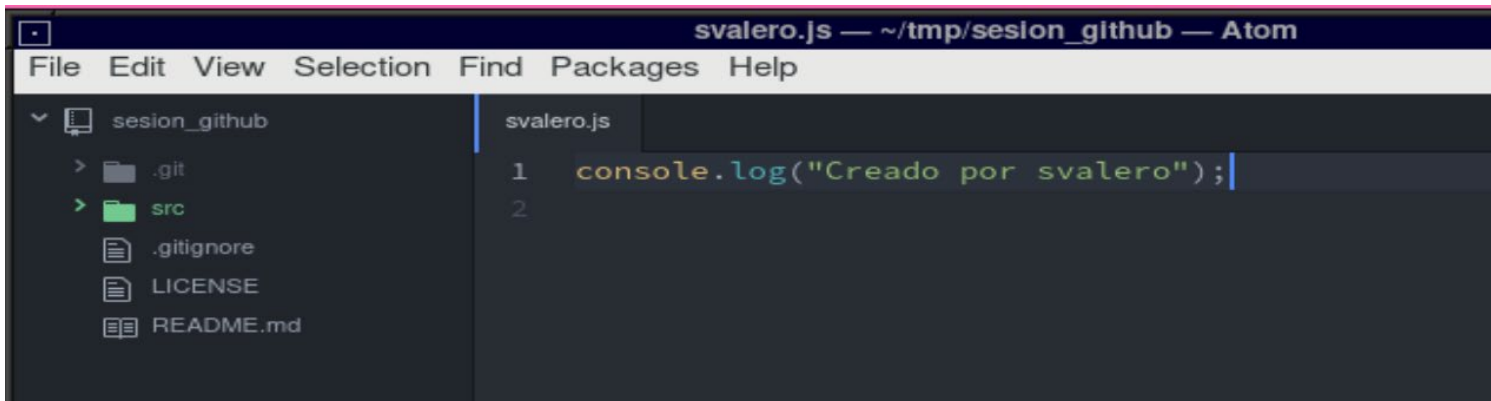
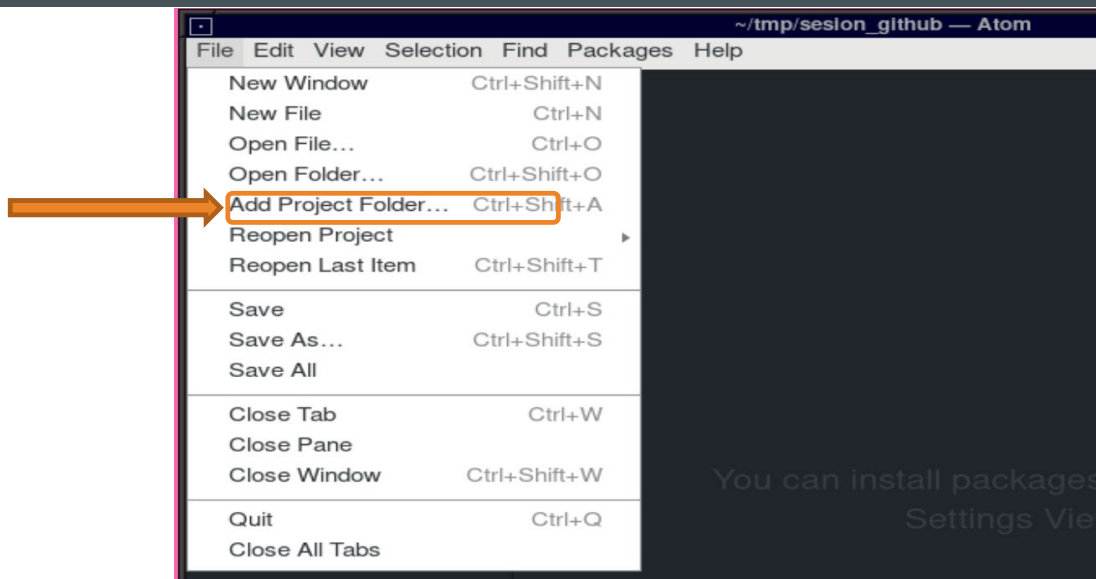
1. Sitúate en la carpeta src del directorio de trabajo del proyecto (si no lo estás)

```
$ cd /home/epsg/tmp/sesion_github/src
```

2. Abre un editor de texto para crear un nuevo fichero que se llame “nombreUsuario.js”. Incluye una línea para mostrar un mensaje por consola.



SUBIR CONFIRMACIONES AL REPOSITORIO REMOTO:



git push



SUBIR CONFIRMACIONES AL REPOSITORIO REMOTO:

Ejercicio

3. Confirma los cambios:

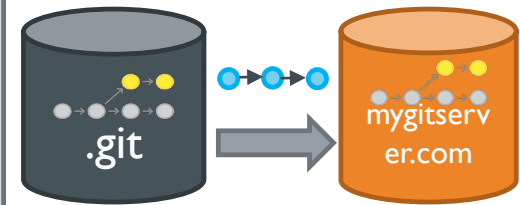
```
$ git commit -a -m "Creado el fichero nombreUsuario.js"
```

4. Sube los cambios al repositorio:

```
$ git push
```

5. Fíjate en los cambios:

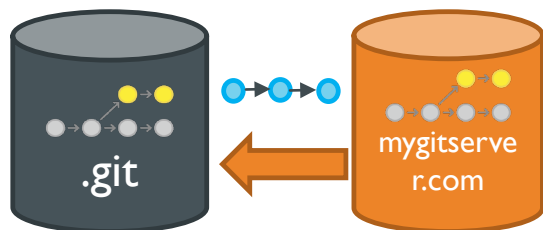
```
$ git log
```



git push



DESCARGAR CONFIRMACIONES AL REPOSITORIO LOCAL



git pull

- Descargará desde el repositorio remoto las confirmaciones que no tengas, y a continuación, de forma inmediata intentará combinarlo en la rama en la que te encuentres.

```
$ git pull
```



DESCARGAR CONFIRMACIONES AL REPOSITORIO LOCAL: EJERCICIO 4

Ejercicio

1. Sitúate en la raíz del directorio de trabajo del proyecto (si no lo estás)

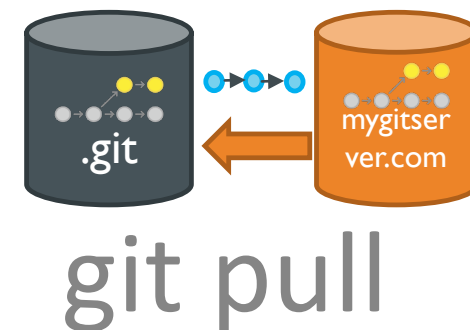
```
$ cd /home/epsg/tmp/sesion_github
```

2. Vamos a descargar las confirmaciones realizadas por todos los miembros del proyecto:

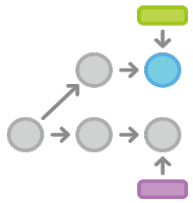
```
$ git pull
```

3. Observa como ahora dentro de src aparecen todos los archivos de tus compañeros. Además, también tienes todos los commits realizados:

```
$ ls /home/epsg/tmp/sesion_github/src  
$ git log
```

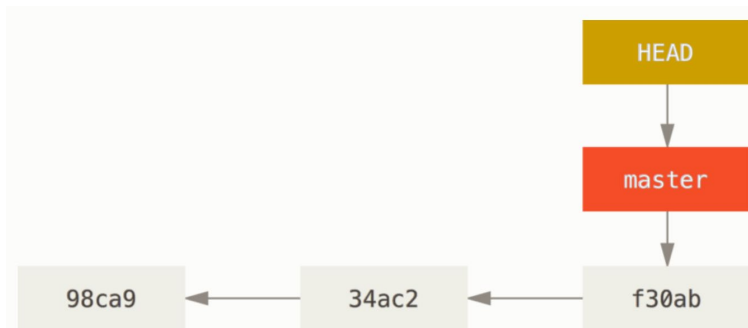


CREAR UNA RAMA

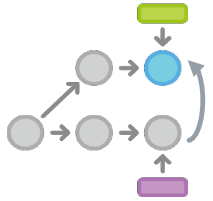


git branch

- Permite listar las ramas que tienes, crear una nueva rama, eliminar ramas y cambiar el nombre de las ramas.
 - Rama:
 - camino de confirmaciones
 - Permite separar el trabajo entre diferentes características de un sistema, evitar conflictos, etc
 - Crear rama: `$ git branch nombre_rama`
 - Listar ramas: `$ git branch`



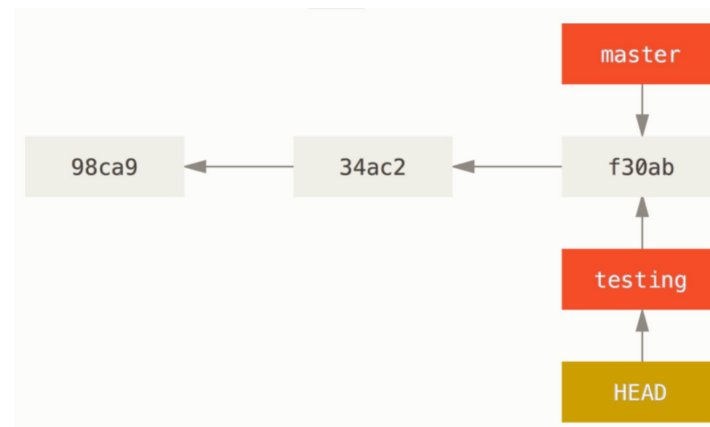
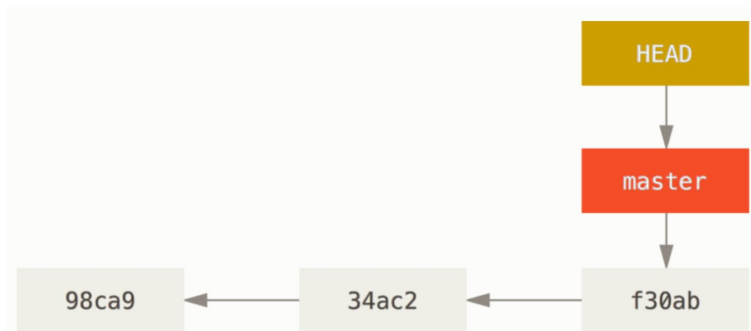
MOVERSE A UNA RAMA



git checkout

- Mueve el espacio de trabajo a la rama indicada (HEAD apunta a esa rama):
 - Las confirmaciones se registrarán en la rama destino
 - El directorio de trabajo debe estar limpio

```
$ git checkout -b nombre_rama
```



CREAR Y MOVERSE A UNA RAMA:

Ejercicio

1. Sitúate en la raíz del directorio de trabajo del proyecto (si no lo estás)

```
$ cd /home/epsg/tmp/sesion_github
```

2. Vamos a crear una rama con la palabra rama más tu nombre de usuario:

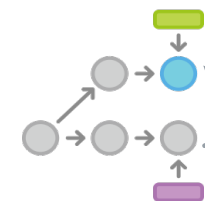
```
$ git branch rama_nombre_usuario
```

3. Muévete a esa nueva rama:

```
$ git checkout rama_nombre_usuario
```

4. Mira el estado del repositorio, debe indicarte que estás en la rama recién creada y que no hay cambios que confirmar.

```
$ git status
```



git checkout



CREAR Y MOVERSE A UNA RAMA:

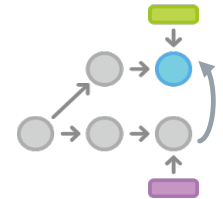
Ejercicio 5

5. Abre el fichero `/home/epsg/tmp/sesion_github/src/nombre_usuario.js` con un editor de texto, por ejemplo atom
6. Modifica el fichero de alguna forma, por ejemplo cambiando el mensaje
7. Confirma los cambios

```
$ git commit -a -m "Rama_nombre_usuario: Mensaje cambiado"
```

8. Mira el estado del repositorio:

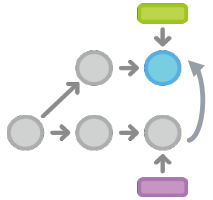
```
$ git log --graph --oneline --all --decorate
```



git checkout



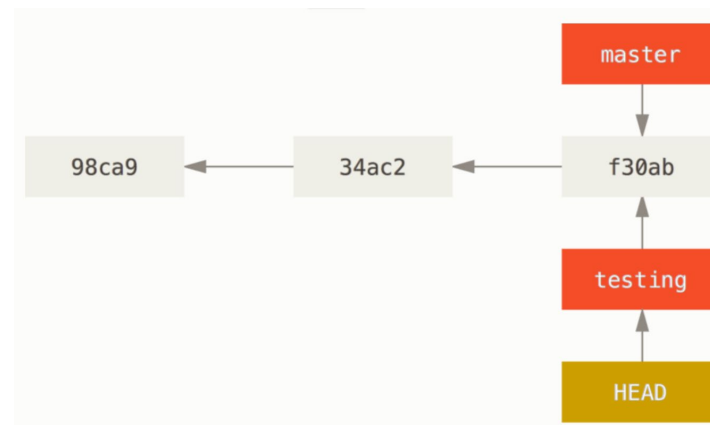
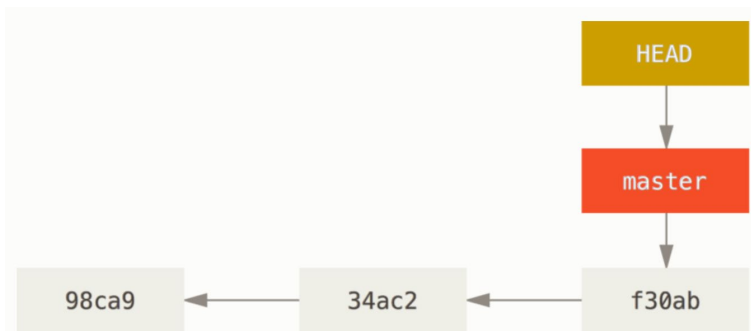
CREAR Y MOVERSE A UNA RAMA



git checkout -b

- Crea una rama y mueve el directorio de trabajo a esa nueva rama (HEAD apunta a esa rama):
 - Las confirmaciones se registrarán en la rama destino
 - El directorio de trabajo debe estar limpio

```
$ git checkout -b nombre_rama
```



CREAR Y MOVERSE A UNA RAMA:

Ejercicio

1. Vamos a volver a la rama master

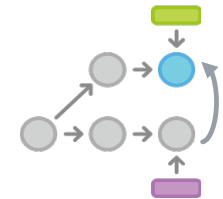
```
$ git checkout master
```

2. Vamos a crear y movernos a una rama con la palabra rama más tu nombre de usuario más 2:

```
$ git checkout -b rama_nombre_usuario2
```

3. Mira el estado del repositorio, debe indicarte que estás en la rama recién creada y que no hay cambios que confirmar.

```
$ git status
```



git checkout -b



CREAR Y MOVERSE A UNA RAMA:

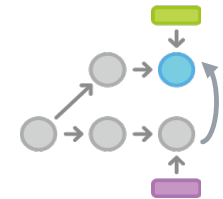
Ejercicio

4. Abre el fichero `/home/epsg/tmp/sesion_github/src/nombre_usuario.js` con un editor de texto, por ejemplo atom
5. Modifica el fichero de alguna forma, por ejemplo cambiando el mensaje
6. Confirma los cambios

```
$ git commit -a -m "Rama_nombre_usuario2: Mensaje cambiado"
```

7. Mira el estado del repositorio:

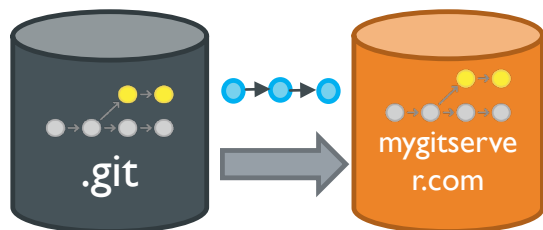
```
$ git log --graph --oneline --all --decorate
```



git checkout -b



SUBIR UNA RAMA AL REPOSITORIO REMOTO



git push

- Para compartir tus ramas con el resto de miembros del proyecto se utiliza también el comando push:

```
$ git push origin nombre_rama
```



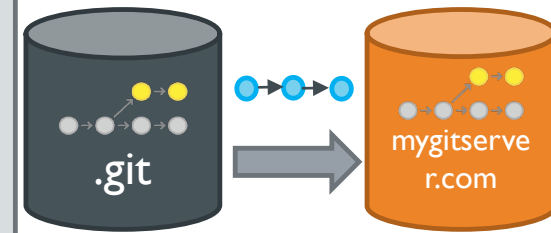
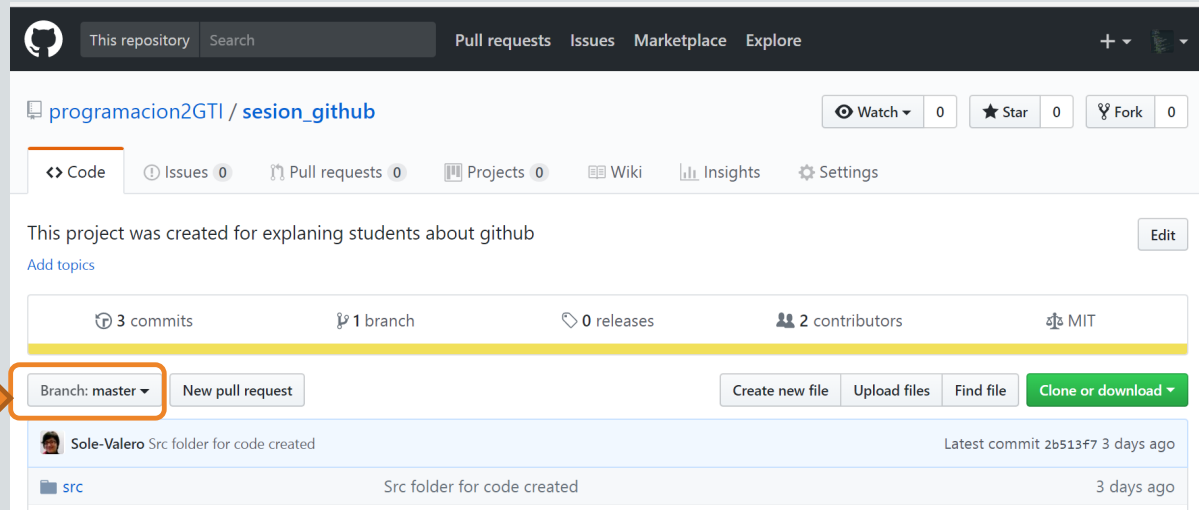
SUBIR UNA RAMA AL REPOSITORIO REMOTO:

Ejercicio

1. Vamos a subir una de las ramas al repositorio remoto:

```
$ git push origin rama_nombre_usuario
```

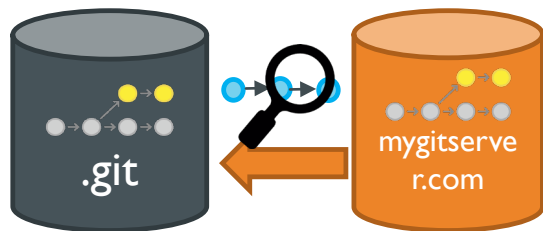
2. Comprueba en github las nuevas ramas creadas en el proyecto:



git push



DESCARGAR CONFIRMACIONES AL REPOSITORIO LOCAL



git fetch

- Actualizará la base de datos del repositorio local con toda la información que no esté actualmente: nuevas ramas, si han habido nuevas confirmaciones en las ramas, etc...

```
$ git fetch
```



DESCARGAR CONFIRMACIONES AL REPOSITORIO LOCAL:

Ejercicio

1. En la consola, ejecuta:

```
$ git branch -a
```

2. Fíjate en el número de ramas que lista
3. Actualiza el repositorio con la información del repositorio remoto:

```
$ git fetch
```

4. Vuelve a listar todas las ramas:

```
$ git branch -a
```



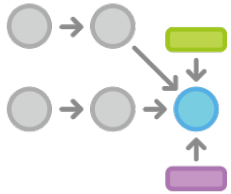
DESCARGAR CONFIRMACIONES AL REPOSITORIO LOCAL:

Ejercicio

¿Hay diferencias?



FUSIONAR RAMAS Y RESOLVER CONFLICTOS



git merge

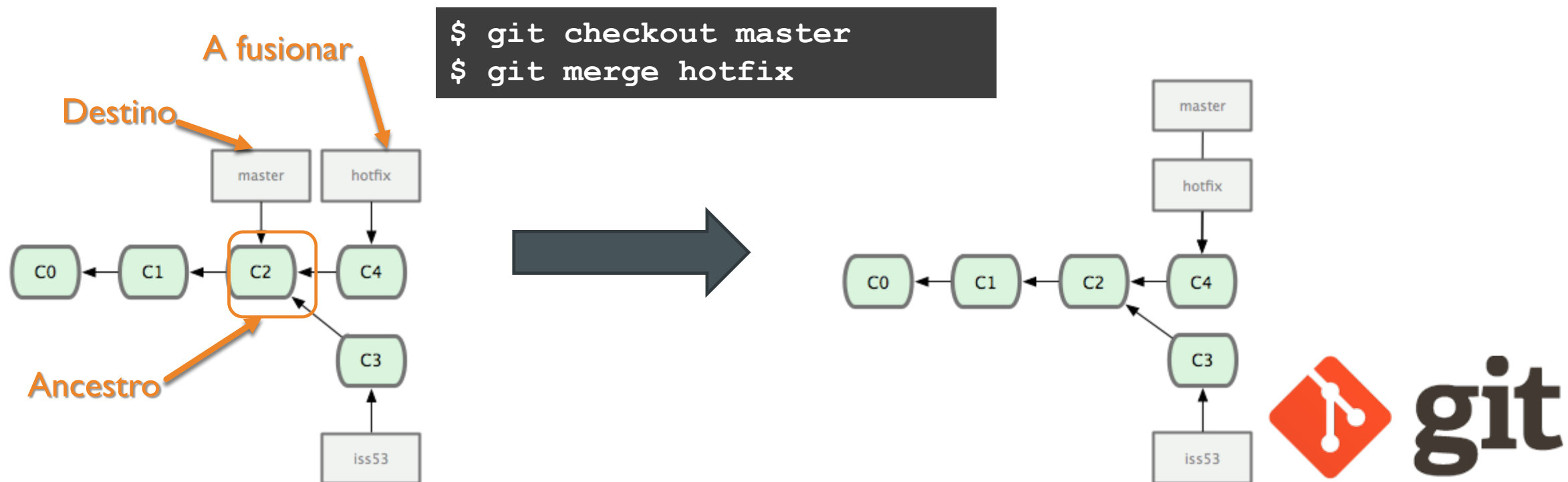
- Actualizará la base de datos del repositorio local con toda la información que no esté actualmente: nuevas ramas, si han habido nuevas confirmaciones en las ramas, etc...
- Primero hay que moverse a la rama en la que se introducirán los cambios, y luego fusionar:

```
$ git checkout rama_destino  
$ git merge rama_a_fusionar
```



FUSIONAR RAMAS Y RESOLVER CONFLICTOS. *TRES BANDAS*

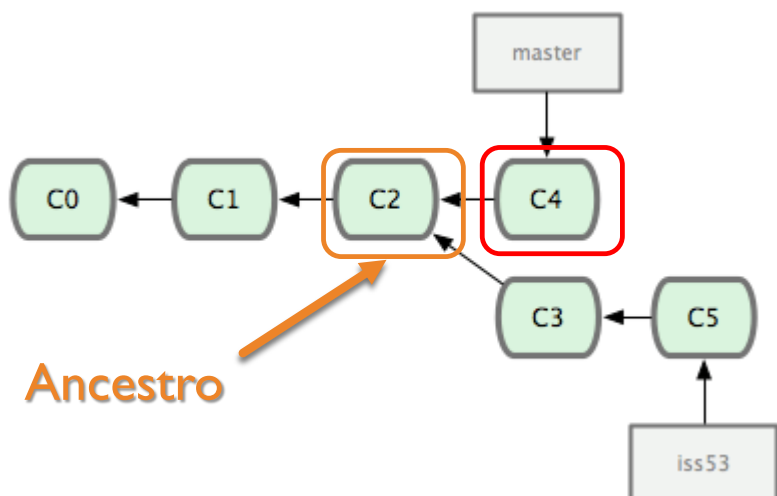
- En el caso de que no hay ninguna confirmación desde el ancestro común de la rama a fusionar
- No hay conflictos



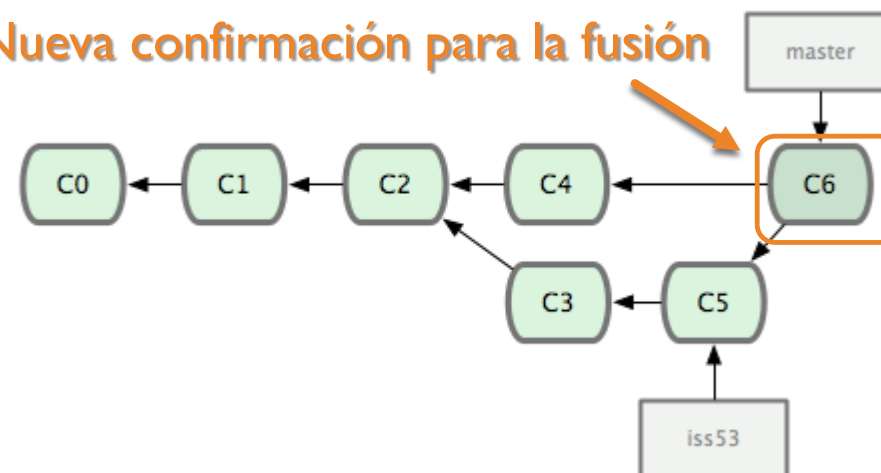
FUSIONAR RAMAS Y RESOLVER CONFLICTOS.

- Hay confirmaciones desde el ancestro común de la rama a fusionar
- Hay **conflictos**: mismos ficheros, ...

```
$ git checkout master  
$ git merge iss53
```



Nueva confirmación para la fusión

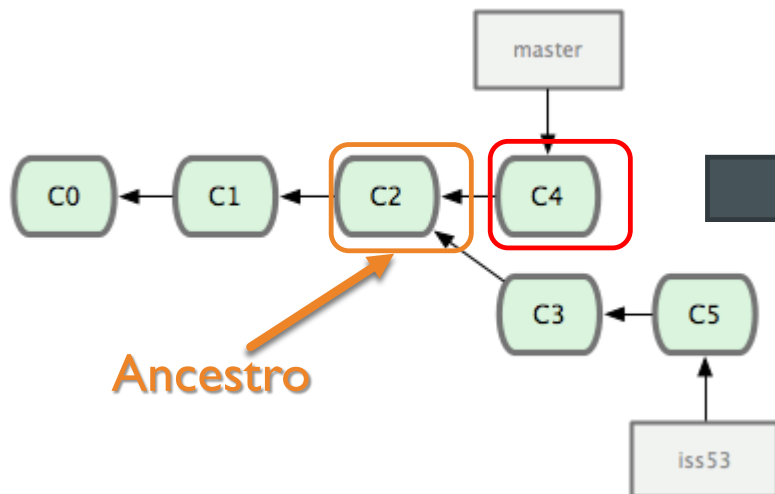


La confirmación de fusión
tiene más de un padre

FUSIONAR RAMAS Y RESOLVER CONFLICTOS. *CONFLICTO*

- Hay confirmaciones desde el ancestro común de la rama a fusionar
- No hay conflictos: distintos ficheros, ...

```
$ git checkout master  
$ git merge iss53
```

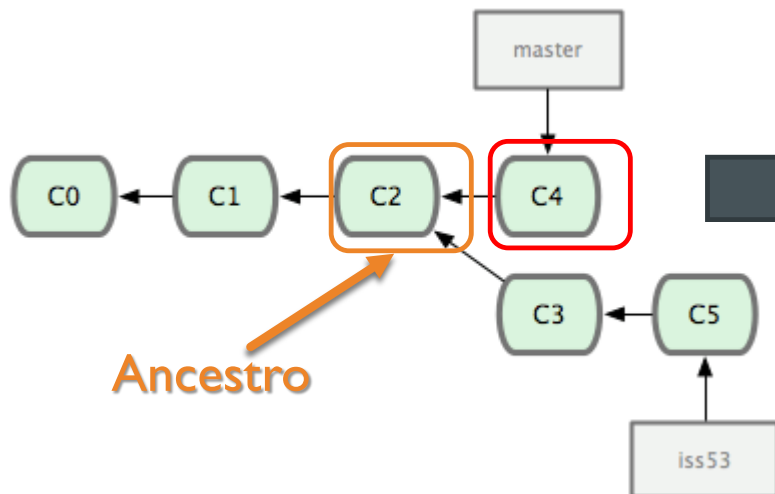


```
$ git merge iss53  
Auto-merging index.html  
CONFLICT (content): Merge conflict in index.html  
Automatic merge failed; fix conflicts and then commit the result.
```

FUSIONAR RAMAS Y RESOLVER CONFLICTOS. *CONFLICTO*

- Hay confirmaciones desde el ancestro común de la rama a fusionar
- No hay conflictos: distintos ficheros, ...

```
$ git checkout master  
$ git merge iss53
```



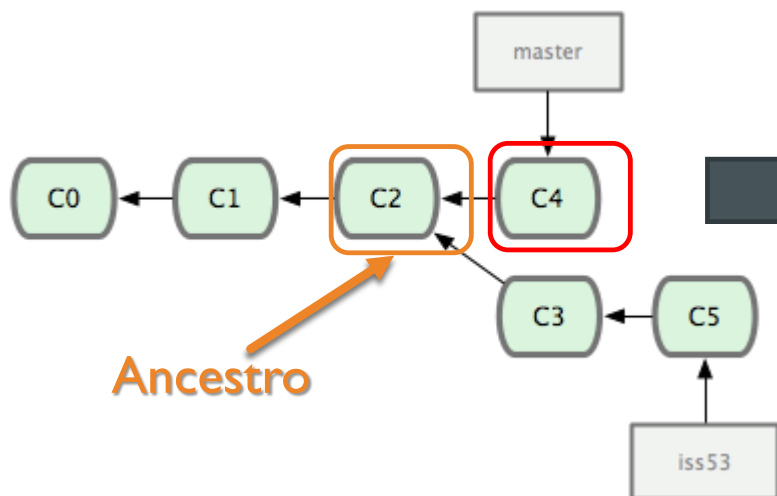
```
$ git merge iss53  
Auto-merging index.html  
CONFLICT (content): Merge conflict in index.html  
Automatic merge failed; fix conflicts and then commit the result.
```

FUSIONAR RAMAS Y RESOLVER CONFLICTOS. *CONFLICTO*

- Hay confirmaciones desde el ancestro común de la rama a fusionar
- No hay conflictos: distintos ficheros, ...

```
$ git checkout master  
$ git merge iss53
```

Al abrir index.html, tendremos:



```
<<<<<< HEAD:index.html  
<div id="footer">contact : email.support@github.com</div>  
=====  
<div id="footer">  
  please contact us at support@github.com  
</div>  
>>>>>> iss53:index.html
```


FUSIONAR RAMAS Y RESOLVER CONFLICTOS. *CONFLICTO*

Contenido del fichero index.html
en la rama master

```
<<<<<<< HEAD:index.html  
<div id="footer">contact : email.support@github.com</div>
```

```
=====  
<div id="footer">  
  please contact us at support@github.com  
</div>  
>>>>>>> iss53:index.html
```

FUSIONAR RAMAS Y RESOLVER CONFLICTOS. *CONFLICTO*

```
<<<<<<< HEAD:index.html
<div id="footer">contact : email.support@github.com</div>
=====
<div id="footer">
  please contact us at support@github.com
</div>
>>>>>>> iss53:index.html
```



Contenido del fichero index.html
en la rama master

FUSIONAR RAMAS Y RESOLVER CONFLICTOS. *CONFLICTO*

a) Decidir con qué nos quedamos y borrar las marcas <<< ===

```
<div id="footer">
please contact us at email.support@github.com
</div>
```

b) Guardar el fichero

c) Confirmar la resolución del conflicto:

```
$ git add index.html
```

FUSIONAR RAMAS Y RESOLVER CONFLICTOS. *CONFLICTO*

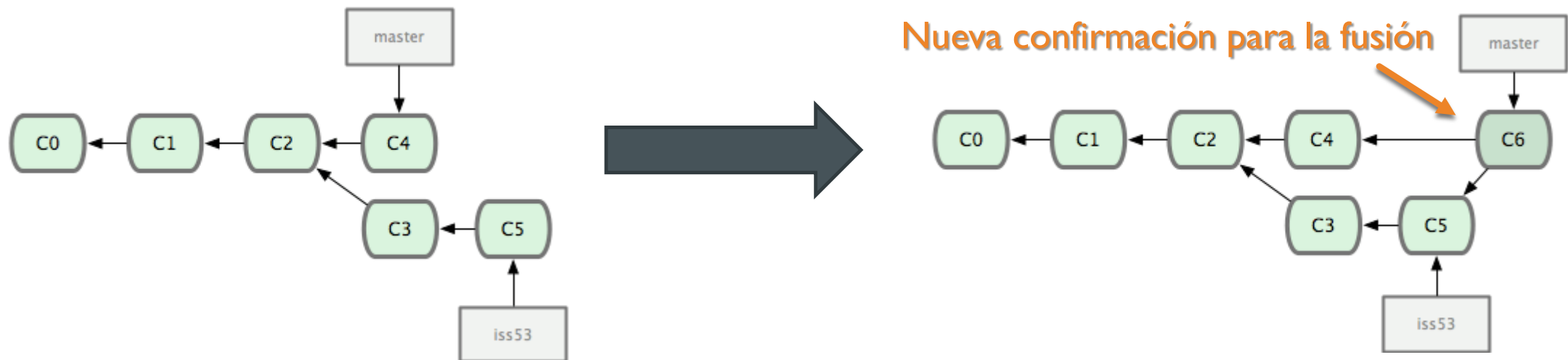
Repetir los pasos anteriores para cada fichero con conflicto, puedes conocer todos los ficheros con conflicto con:

```
$ git status
```

Al terminar, confirmar la fusión con un commit, con el correspondiente mensaje:

```
$ git commit -m "Fusión de las ramas iss53 y master. Conflictos resueltos"
```

FUSIONAR RAMAS Y RESOLVER CONFLICTOS. *CONFLICTO*



CONFIGURAR UN REPOSITORIO



- Permite obtener y establecer variables de configuración que controlan el aspecto y funcionamiento de Git.

- Tu identidad

```
$ git config --global user.name "tuNombreUsuario"
$ git config --global user.email
tucorreo@example.org
```

- Tu editor. Por defecto usa el editor **vi**. Por ejemplo, para usar **emacs**:

```
$ git config --global core.editor emacs
```

- Conocer los valores de configuración

```
$ git config --list
```



CONFIGURAR UN REPOSITORIO:

Ejercicio

1. Abre una terminal
2. Configura tu nombre de usuario:

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

3. Añade tu correo electrónico:

```
$ git config --global user.email tuCorreo@gmail.com
```

4. Cambiemos el editor por defecto por emacs:

```
$ git config --global core.editor emacs
```

5. Vamos a ver qué ha pasado:

```
$ git config --list
```



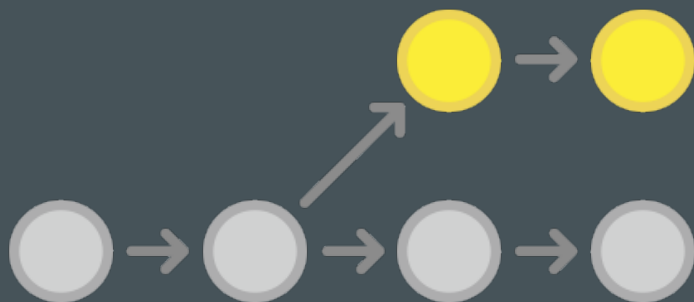
CONFIGURAR UN REPOSITORIO:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git config --list
user.email=svalero@dsic.upv.es
user.name=svalero
core.editor=emacs
core.repositoryformatversion=0
core.filemode=true
core.bare=false
core.logallrefupdates=true
[epsg@archlinux prac_gti]$
```

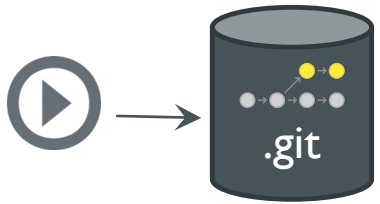


SOLO USO LOCAL

INICIAR UN REPOSITORIO



INICIAR UN REPOSITORIO LOCAL



git init

- Es posible usar git sobre cualquier directorio de trabajo
- Crear el directorio donde se va a trabajar o moverte a un directorio ya creado, y luego:

```
$ git init
```

- Se creará un repositorio local, donde se podrá utilizar la herramienta, pero no podrán compartir los cambios con otras personas.



INICIAR UN REPOSITORIO:

Ejercicio

1. Abre una terminal
2. Crea un directorio llamado `prac_git`:
3. Sitúate dentro del directorio:
4. Veamos que hay dentro:
5. Inicialicemos el repositorio local:
6. Vamos a ver qué ha pasado:

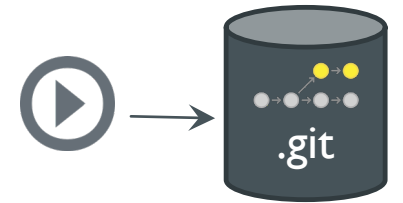
```
$ mkdir prac_git
```

```
$ cd prac_git
```

```
$ ls -la
```

```
$ git init
```

```
$ ls -la
```



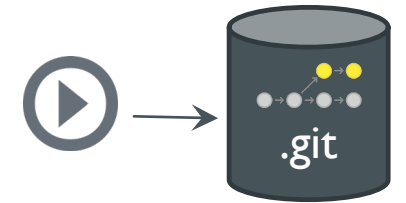
git init



INICIAR UN REPOSITORIO:

```
svalero@bicho: ~/gitRepositorios/prac_git

svalero@bicho:~/gitRepositorios$ mkdir prac_git
svalero@bicho:~/gitRepositorios$ cd prac_git
svalero@bicho:~/gitRepositorios/prac_git$ ls -la
total 8
drwxrwxr-x  2 svalero svalero 4096 ene 25 12:56 .
drwxrwxr-x 30 svalero svalero 4096 ene 25 12:56 ..
svalero@bicho:~/gitRepositorios/prac_git$ git init
Initialized empty Git repository in /home/svalero/gitRepositorios/prac_git/.git/
svalero@bicho:~/gitRepositorios/prac_git$ ls -la
total 12
drwxrwxr-x  3 svalero svalero 4096 ene 25 12:58 .
drwxrwxr-x 30 svalero svalero 4096 ene 25 12:56 ..
drwxrwxr-x  7 svalero svalero 4096 ene 25 12:58 .git
svalero@bicho:~/gitRepositorios/prac_git$
```

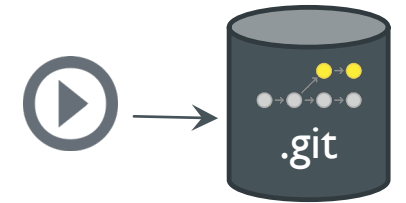


git init



INICIAR UN REPOSITORIO:

```
svalero@bicho: ~/gitRepositorios/prac_git  
  
svalero@bicho:~/gitRepositorios$ mkdir prac_git  
svalero@bicho:~/gitRepositorios$ cd prac_git  
svalero@bicho:~/gitRepositorios/prac_git$ ls -la  
total 8  
drwxrwxr-x  2 svalero svalero 4096 ene 25 12:56 .  
drwxrwxr-x 30 svalero svalero 4096 ene 25 12:56 ..  
svalero@bicho:~/gitRepositorios/prac_git$ git init  
Initialized empty Git repository in /home/svalero/gitRepositorios/prac_git/.git/  
svalero@bicho:~/gitRepositorios/prac_git$ ls -la  
total 12  
drwxrwxr-x  3 svalero svalero 4096 ene 25 12:58 .  
drwxrwxr-x 30 svalero svalero 4096 ene 25 12:56 ..  
drwxrwxr-x  7 svalero svalero 4096 ene 25 12:58 .git  
svalero@bicho:~/gitRepositorios/prac_git$
```

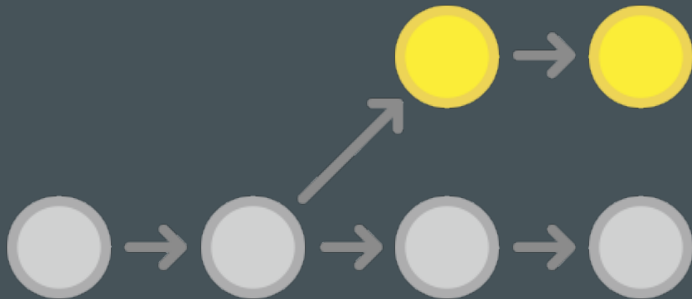


git init

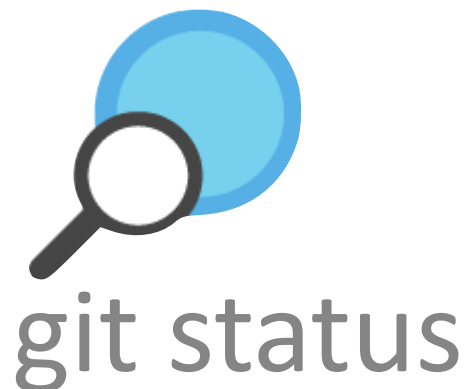


COMANDOS DEL DIA A DIA

PRINCIPALES COMANDOS



REVISAR EL ESTADO DE TUS ARCHIVOS



- Comando para saber en qué estado están los archivos de tu repositorio:
 - Qué archivos no están bajo seguimiento (no están rastreados)
 - Qué archivos bajo seguimiento han sido modificados
 - Qué archivos se incluirán en la siguiente foto del repositorio
- Es importante **leer la información que muestra**, puesto que te indica incluso los comandos que debes utilizar en cada momento según sea el estado de los archivos



REVISAR EL ESTADO DE TUS ARCHIVOS:

Ejercicio

1. Abre una terminal
2. Sitúate dentro del directorio `prac_gti`
3. Veamos el estado del repositorio:

```
$ git status
```



git status



REVISAR EL ESTADO DE TUS ARCHIVOS

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
[epsg@archlinux prac_gti]$
```



git status



REVISAR EL ESTADO DE TUS ARCHIVOS

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
[epsg@archlinux prac_gti]$
```



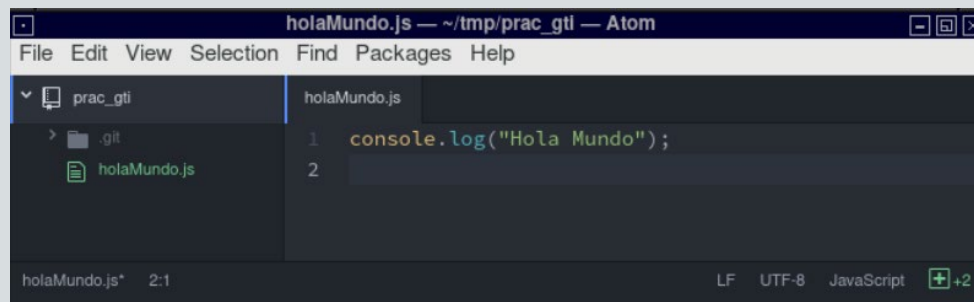
git status



REVISAR EL ESTADO DE TUS ARCHIVOS:

Ejercicio

1. Abre una terminal
2. Sitúate dentro del directorio `prac_gti`
3. Vamos a crear un archivo con el editor que desees, por ejemplo atom: `$ atom`
4. Llámalo `holaMundo.js`. Dentro añade:



```
holaMundo.js — ~/tmp/prac_gti — Atom
File Edit View Selection Find Packages Help
└─ prac_gti
  └─ .git
    └─ holaMundo.js
      1 console.log("Hola Mundo");
      2
holaMundo.js* 2:1 LF UTF-8 JavaScript +2
```



git status



REVISAR EL ESTADO DE TUS ARCHIVOS:

Ejercicio

5. Veamos qué ha pasado. Primero veamos como el contenido del directorio de trabajo ha cambiado, puesto que tenemos un fichero nuevo. Ejecuta:

```
$ ls -ls
```

6. Ahora veamos qué ha pasado a nivel de git. Nuevamente, preguntemos el estado:

```
$ git status
```



git status



REVISAR EL ESTADO DE TUS ARCHIVOS:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ ls -la
total 16
drwxr-xr-x 3 epsg epsg 4096 ene 26 10:38 .
drwxr-xr-x 3 epsg epsg 4096 ene 25 13:35 ..
drwxr-xr-x 7 epsg epsg 4096 ene 26 10:41 .git
-rw-r--r-- 1 epsg epsg  27 ene 26 10:40 holaMundo.js
[epsg@archlinux prac_gti]$ git status
On branch master

Initial commit

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

        holaMundo.js

nothing added to commit but untracked files present (use "git add" to track)
[epsg@archlinux prac_gti]$
```



git status



REVISAR EL ESTADO DE TUS ARCHIVOS:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ ls -la
total 16
drwxr-xr-x 3 epsg epsg 4096 ene 26 10:38 .
drwxr-xr-x 3 epsg epsg 4096 ene 25 13:35 ..
drwxr-xr-x 7 epsg epsg 4096 ene 26 10:41 .git
-rw-r--r-- 1 epsg epsg  27 ene 26 10:40 holaMundo.js
[epsg@archlinux prac_gti]$ git status
On branch master

Initial commit

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

        holaMundo.js

nothing added to commit but untracked files present (use "git add" to track)
[epsg@archlinux prac_gti]$
```



git status



REVISAR EL ESTADO DE TUS ARCHIVOS:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ ls -la
total 16
drwxr-xr-x 3 epsg epsg 4096 ene 26 10:38 .
drwxr-xr-x 3 epsg epsg 4096 ene 25 13:35 ..
drwxr-xr-x 7 epsg epsg 4096 ene 26 10:41 .git
-rw-r--r-- 1 epsg epsg  27 ene 26 10:40 holaMundo.js
[epsg@archlinux prac_gti]$ git status
On branch master

Initial commit

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

    holaMundo.js

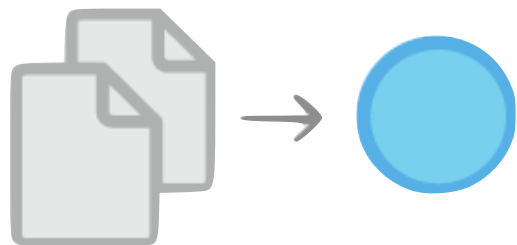
nothing added to commit but untracked files present (use "git add" to track)
[epsg@archlinux prac_gti]$
```



git status



PONER BAJO SEGUIMIENTO UN ARCHIVO



git add

- Comando para indicar a git que debe seguir/trazar el fichero para registrar las modificaciones que se hagan en su contenido.
- Como parámetro puede recibir:
 - El nombre de un fichero
 - El nombre de un directorio: pone bajo seguimiento todos los ficheros que contenga



PONER BAJO SEGUIMIENTO UN ARCHIVO:

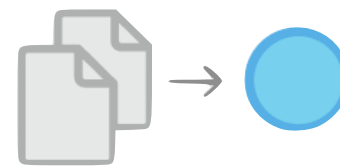
Ejercicio

1. Vamos a poner bajo seguimiento el fichero que habíamos creado:

```
$ git add holaMundo.js
```

2. Veamos ahora que nos dice git de su estado:

```
$ git status
```



git add



PONER BAJO SEGUIMIENTO UN ARCHIVO: EJERCICIO 5

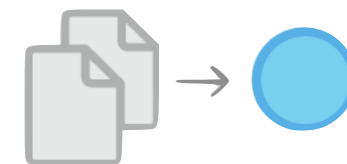
```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git add holaMundo.js
[epsg@archlinux prac_gti]$ git status
On branch master

Initial commit

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

    new file:   holaMundo.js

[epsg@archlinux prac_gti]$
```



git add



PONER BAJO SEGUIMIENTO UN ARCHIVO:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git add holaMundo.js
[epsg@archlinux prac_gti]$ git status
On branch master

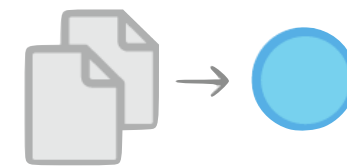
Initial commit

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

    new file:   holaMundo.js

[epsg@archlinux prac_gti]$
```

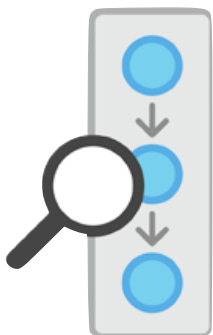
Git comienza a seguir el fichero, haciendo una “foto” inicial de su contenido



git add

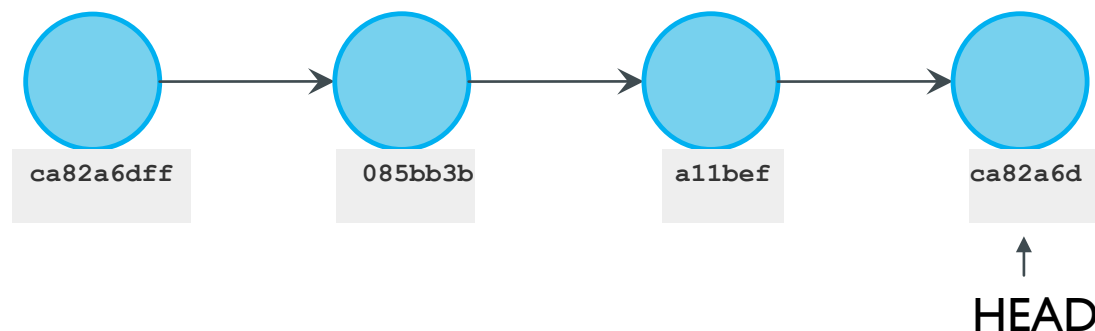


VER EL HISTORIAL DE CONFIRMACIONES



git log

- Te permite conocer qué modificaciones (commits) se han llevado a cabo, quien las realizó y por qué.
- Dependiendo de los parámetros que le pases, te dará una información o otra:
 - - 2: hace que se muestren únicamente las dos últimas entradas del historial
 - -all :todas las modificaciones
 - -graph: dibuja una representación gráfica basada en texto del historial de confirmaciones en el lado izquierdo de la salida.
 - - p: muestra las diferencias introducidas en cada confirmación.
 - - online: usa una única línea y solo muestra el prefijo de cada identificador de un commit
 - Etc..



Confirmación en la que actualmente estás trabajando. Normalmente el último



VER TUS CAMBIOS:

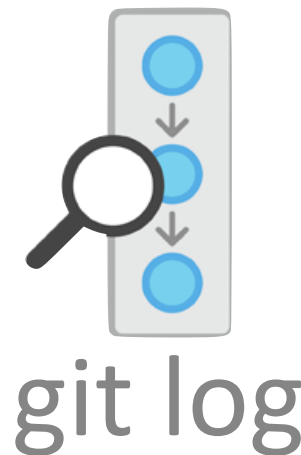
Ejercicio

1. Veamos el historial de confirmaciones de nuestro repositorio:

```
$ git log
```

2. Configuremos el git log para ver más cosas:

```
$ git log --graph --oneline --all --  
decorate
```

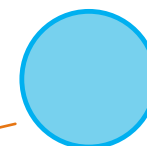


CONFIRMAR TUS CAMBIOS:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git log
commit b06d27cf7de3d6c2f33502ce7d51b04135fd09cd
Author: svalero <svalero@dsic.upv.es>
Date:   Fri Jan 26 14:18:21 2018 +0100

    Creado el hola mundo
[epsg@archlinux prac_gti]$
```

Sólo hay un commit en
nuestro repositorio



← HEAD

b06d27cf

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git log --graph --decorate --all --decorate
* commit b06d27cf7de3d6c2f33502ce7d51b04135fd09cd (HEAD -> master)
   Author: svalero <svalero@dsic.upv.es>
   Date:   Fri Jan 26 14:18:21 2018 +0100

       Creado el hola mundo
[epsg@archlinux prac_gti]$
```

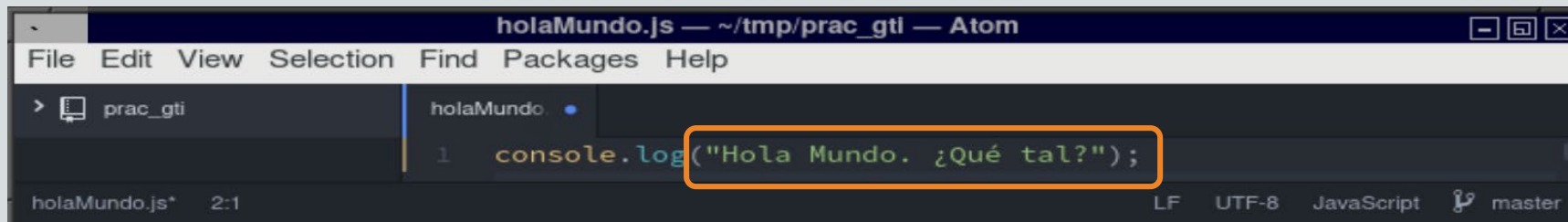
Rama de trabajo que se crea
por defecto



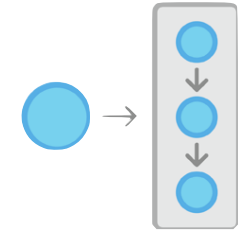
CONFIRMAR Y VER TUS CAMBIOS:

Ejercicio

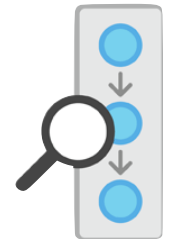
1. Hagamos que nuestra rama de trabajo tenga más confirmaciones. Abre el fichero holaMundo.js con el editor que quieras, atom por ejemplo. `$ atom`
2. Modifica el mensaje a mostrar por consola:



```
holaMundo.js — ~/tmp/prac_gti — Atom
File Edit View Selection Find Packages Help
> prac_gti holaMundo
1 console.log("Hola Mundo. ¿Qué tal?");
holaMundo.js* 2:1 LF UTF-8 JavaScript master
```



git commit



git log



CONFIRMAR Y VER TUS CAMBIOS:

Ejercicio

3. Veamos el estado del directorio de trabajo: `$ git status`

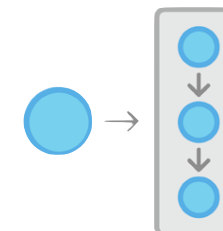
```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git checkout -- <file>..." to discard changes in working directory)

    modified:   holaMundo.js

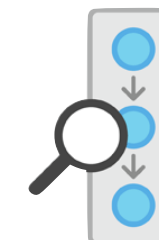
no changes added to commit (use "git add" and/or "git commit -a")
[epsg@archlinux prac_gti]$
```

4. Confirmemos el cambio en el repositorio:

```
$ git commit -a -m "Cambiado el mensaje a mostrar"
```



git commit



git log



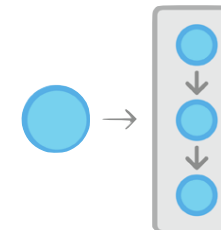
CONFIRMAR Y VER TUS CAMBIOS:

Ejercicio

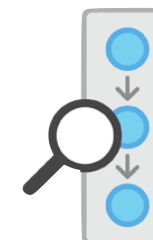
5. ¿qué veremos en el repositorio local? ¿Qué cambios ha detectado?

```
$ git log -2 -p
```

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



git commit



git log



CONFIRMAR Y VER TUS CAMBIOS: EJERCICIO 8

The diagram illustrates the Git commit workflow. On the left, a terminal window shows the execution of `git log -2 -p`, displaying the last two commits. On the right, a commit history visualization shows two commits as blue circles, with the top one labeled `HEAD` and the bottom one labeled `b06d27c`. The terminal output is annotated with icons and labels: a person icon for the commit message, a calendar icon for the date, and a document icon for the changes. The commit message is `Combiado el mensaje a mostrar`. The changes are shown as a diff between `a/holaMundo.js` and `b/holaMundo.js`, highlighting the addition of `console.log("Hola Mundo. ¿Qué tal?");` and the removal of `console.log("Hola Mundo");`. The terminal also shows the creation of the `hola mundo` file.

```
[epsg@archlinux prac_gti]$ git log -2 -p
commit b943f6ace9d5552701a5fec36798d0ccee07eb69
Author: svalero <svalero@dsic.upv.es>
Date: Sat Jan 27 19:06:34 2018 +0100

    Combiado el mensaje a mostrar

diff --git a/holaMundo.js b/holaMundo.js
index 49a88d0..8a605ca 100644
--- a/holaMundo.js
+++ b/holaMundo.js
@@ -1,1 @@
-console.log("Hola Mundo");
+console.log("Hola Mundo. ¿Qué tal?");

commit b06d27cf7de3d6c2f33502ce7d51b04135fd09cd
Author: svalero <svalero@dsic.upv.es>
Date: Fri Jan 26 14:18:21 2018 +0100

    Creado el hola mundo

diff --git a/holaMundo.js b/holaMundo.js
new file mode 100644
index 0000000..49a88d0
--- /dev/null
+++ b/holaMundo.js
@@ -0,0 +1 @@
+console.log("Hola Mundo");
[epsg@archlinux prac_gti]$
```

HEAD

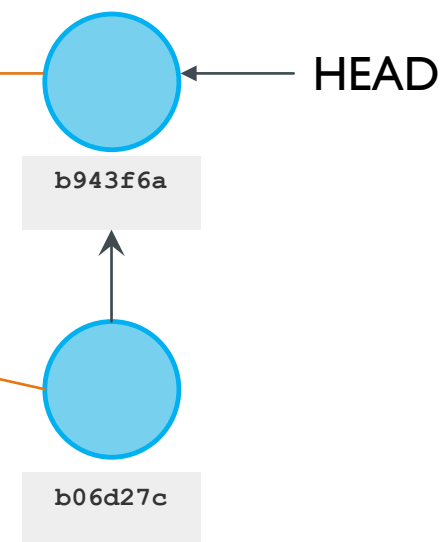
b943f6a

b06d27c

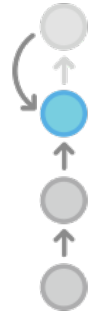
git

CONFIRMAR Y VER TUS CAMBIOS:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git log --graph --oneline --decorate
* b943f6a (HEAD -> master) Cambiado el mensaje a mostrar
* b06d27c Creado el hola mundo
[epsg@archlinux prac_gti]$
```

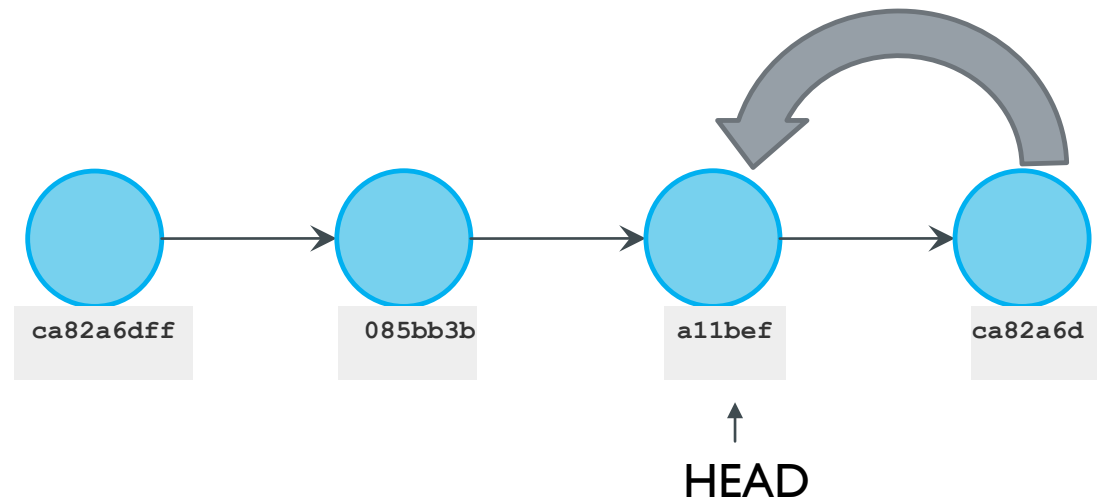


MOVERTE A UNA CONFIRMACIÓN



git checkout

- Te permite reestablecer el directorio de trabajo al estado en el que estaba en un commit determinado:
 - Hace que HEAD sea el commit que se pasa como argumento
- También permite cambiar entre ramas



MOVERTE A UNA CONFIRMACIÓN:

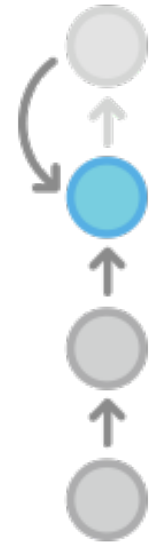
Ejercicio

1. Vamos a sacar el id del commit al que nos moveremos:

```
$ git log --oneline
```

2. Ahora, movamos nuestro directorio de trabajo al estado en el que estaba en nuestro commit inicial (el id de tu máquina, no será el mismo que este)

```
$ git checkout b0627c
```



git checkout



MOVERTE A UNA CONFIRMACIÓN:

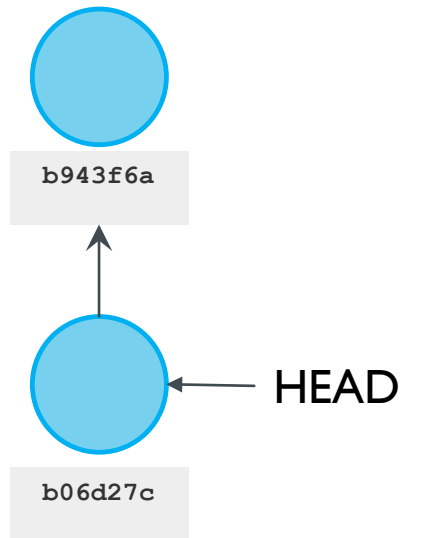
```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git log --oneline
b943f6a Cambiado el mensaje a mostrar
b06d27c Creado el hola mundo
[epsg@archlinux prac_gti]$ git checkout b06d27c
Note: checking out 'b06d27c'.

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 b06d27c... Creado el hola mundo
[epsg@archlinux prac_gti]$
```



MOVERTE A UNA CONFIRMACIÓN:

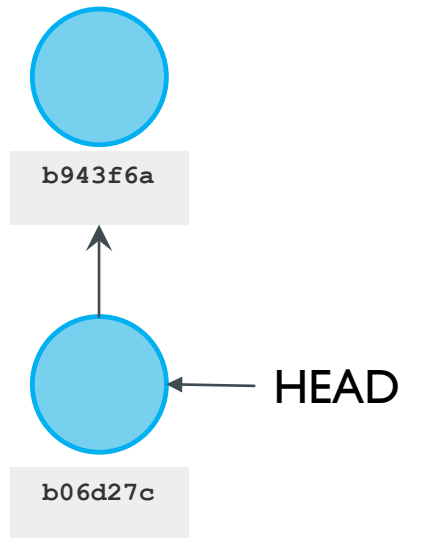
```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git log --oneline
b943f6a Cambiado el mensaje a mostrar
b06d27c Creado el hola mundo
[epsg@archlinux prac_gti]$ git checkout b06d27c
Note: checking out 'b06d27c'.

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>

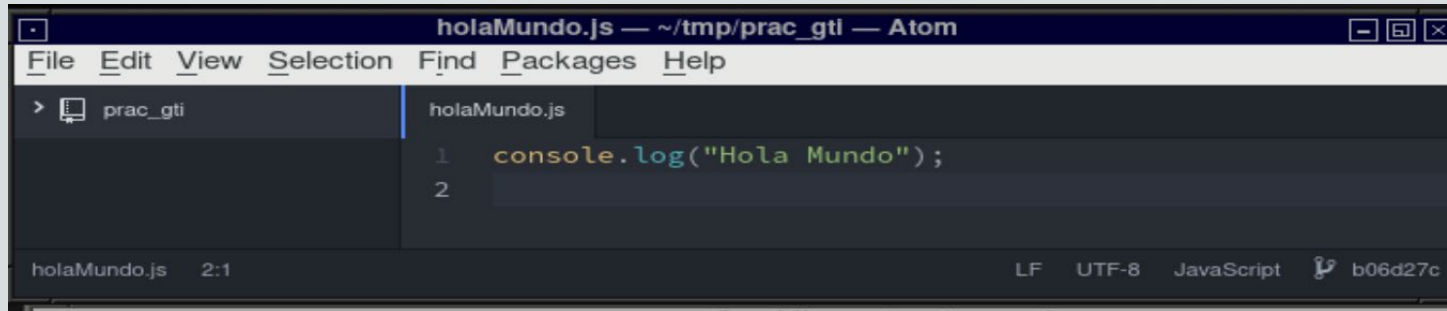
HEAD is now at b06d27c... Creado el hola mundo
[epsg@archlinux prac_gti]$
```



MOVERTE A UNA CONFIRMACIÓN:

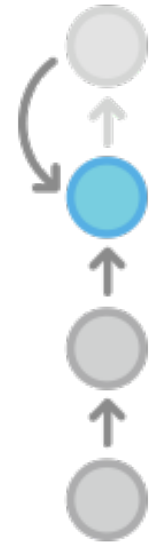
Ejercicio

3. ¿Qué contenido tiene ahora el fichero holaMundo.js?



```
holaMundo.js — ~/tmp/prac_gti — Atom
File Edit View Selection Find Packages Help
> prac_gti holaMundo.js
1 console.log("Hola Mundo");
2
holaMundo.js 2:1 LF UTF-8 JavaScript b06d27c
```

¿MAGIA?



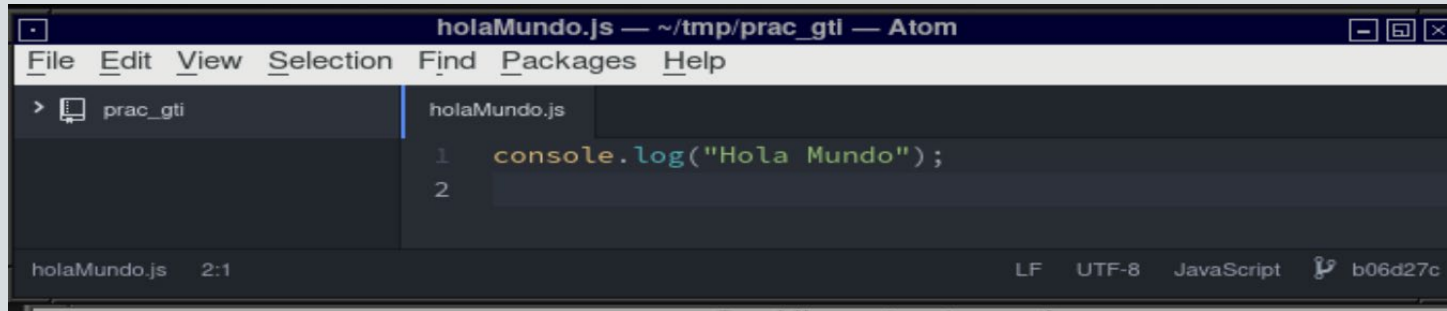
git checkout



MOVERTE A UNA CONFIRMACIÓN:

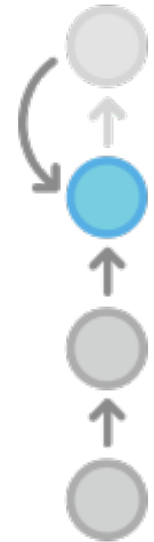
Ejercicio

3. ¿Qué contenido tiene ahora el fichero holaMundo.js?



```
holaMundo.js — ~/tmp/prac_gti — Atom
File Edit View Selection Find Packages Help
> prac_gti
holaMundo.js
1 console.log("Hola Mundo");
2
holaMundo.js 2:1 LF UTF-8 JavaScript b06d27c
```

¿MAGIA? NO, **git**



git checkout

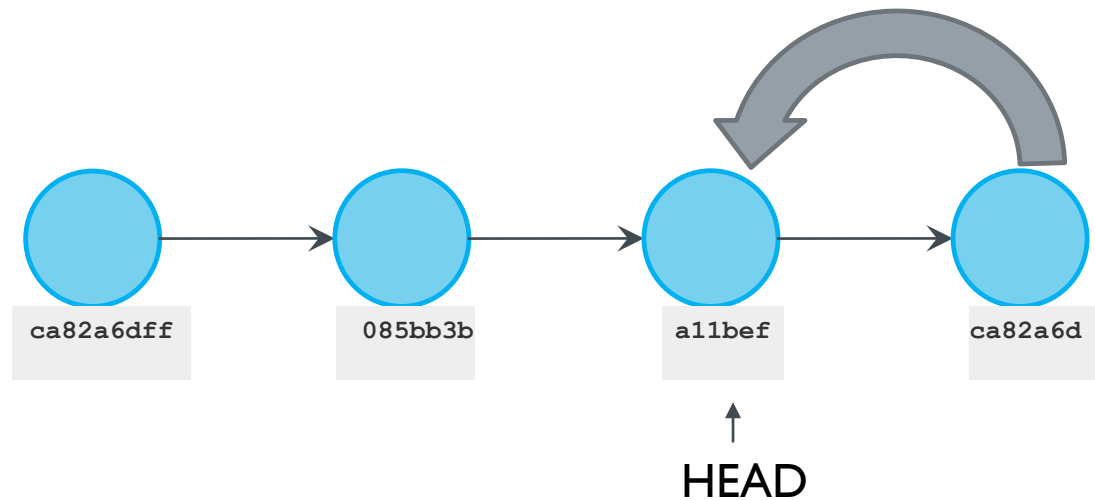


MOVERTE A UNA CONFIRMACIÓN: ¿QUÉ TE APORTA?

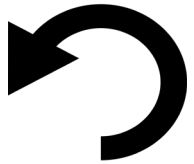


git checkout

- Si desde el último commit nada funciona, y no encuentras el error, puedes volver a un commit anterior y ver si en ese estado el problema se corrige, de ser así:
 - En alguno de los cambios del último commit está el problema
 - Puedes ayudarte de log para intentar descubrirlo o seguir desde ese punto



DESHACER CAMBIOS



git reset

- Deshace cambios realizados en el directorio de trabajo. Los usos más habituales:

- Eliminar los últimos cambios realizados en los ficheros (sin estar confirmados) y volver al estado del último commit:

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

- Eliminar el último commit del repositorio local:

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

- Eliminar el último fichero que se ha añadido para hacer seguimiento o añadir a un commit:

```
$ git add unFichero.txt  
$ git reset unFichero.txt
```



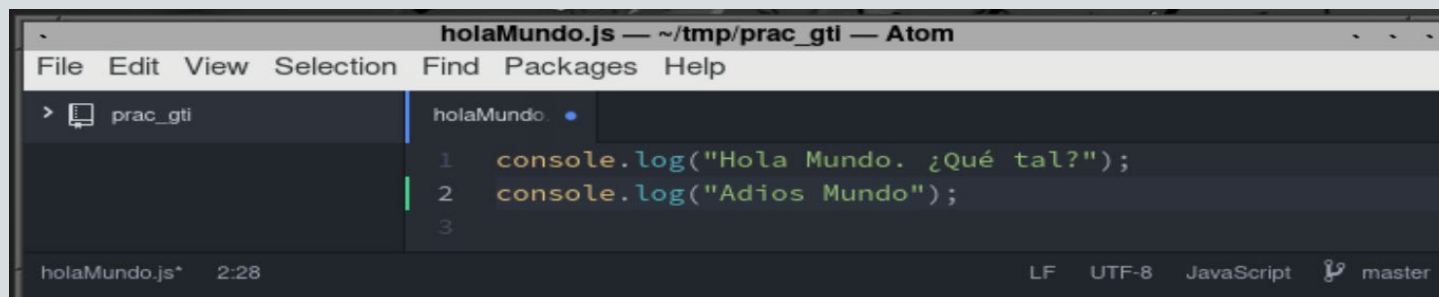
DESHACER CAMBIOS:

Ejercicio

1. Volvamos a colocarnos en el último commit:

```
$ git checkout master
```

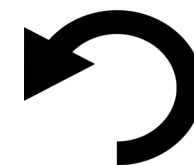
2. Modifica holaMundo.js, añadiendo otro mensaje de consola:



```
holaMundo.js — ~/tmp/prac_gti — Atom
File Edit View Selection Find Packages Help
> | prac_gti | holaMundo.js
1 console.log("Hola Mundo. ¿Qué tal?");
2 console.log("Adios Mundo");
3
holaMundo.js* 2:28 LF UTF-8 JavaScript master
```

3. Veamos que nos dice git:

```
$ git status
```



git reset



DESHACER CAMBIOS:

Ejercicio

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ 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)

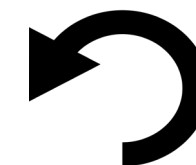
        modified:   holaMundo.js

no changes added to commit (use "git add" and/or "git commit -a")
[epsg@archlinux prac_gti]$
```

4. Descartemos las modificaciones:

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

5. ¿Qué contiene el fichero ahora? ¿En qué estado está nuestro directorio de trabajo?



git reset

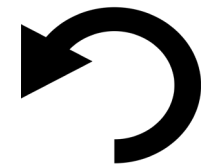
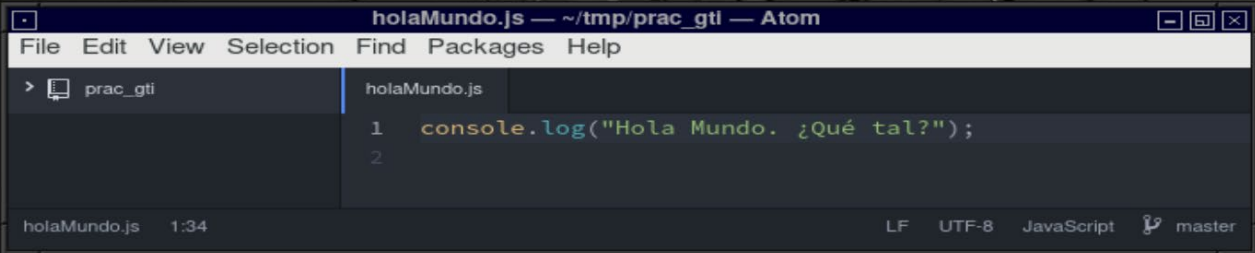


DESHACER CAMBIOS:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ 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:   holaMundo.js

no changes added to commit (use "git add" and/or "git commit -a")
[epsg@archlinux prac_gti]$ git reset --hard
HEAD is now at b943f6a Cambiado el mensaje a mostrar
[epsg@archlinux prac_gti]$ git status
On branch master
nothing to commit, working tree clean
[epsg@archlinux prac_gti]$
```



git reset



GUARDAR CAMBIOS DE FORMA TEMPORAL



git stash

- Deshace cambios realizados en el directorio de trabajo, pero los guarda para aplicarlos después, si se desea.
- Deja limpio el directorio de trabajo, en el estado indicado por el último commit

```
$ git stash
```

- Recuperar la lista de cambios guardados de forma temporal:

```
$ git stash list
```

- Aplicar en el directorio un cambio guardado:

```
$ git stash apply
```

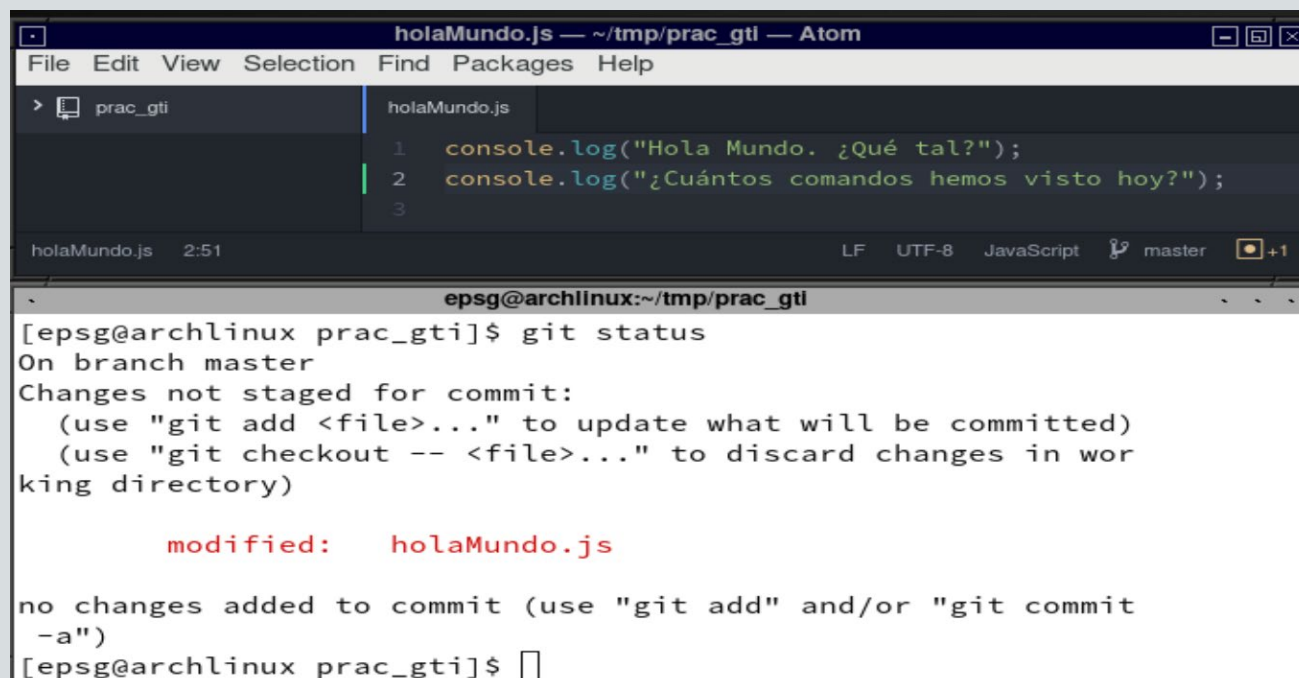
```
$ git stash apply stash@{2}
```



GUARDAR CAMBIOS DE FORMA TEMPORAL:

Ejercicio

1. Modifica el fichero holaMundo.js y mira el estado del repositorio:



The screenshot shows the Atom text editor with the file `holaMundo.js` open. The file contains two lines of JavaScript code: `console.log("Hola Mundo. ¿Qué tal?");` and `console.log("¿Cuántos comandos hemos visto hoy?");`. Below the editor, a terminal window shows the output of the `git status` command. The output indicates that the file `holaMundo.js` has been modified but is not yet staged for commit.

```
holaMundo.js — ~/tmp/prac_gti — Atom
File Edit View Selection Find Packages Help
> | prac_gti | holaMundo.js
1 console.log("Hola Mundo. ¿Qué tal?");
2 console.log("¿Cuántos comandos hemos visto hoy?");
3
holaMundo.js 2:51 LF UTF-8 JavaScript master +1

epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ 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:   holaMundo.js

no changes added to commit (use "git add" and/or "git commit -a")
[epsg@archlinux prac_gti]$
```

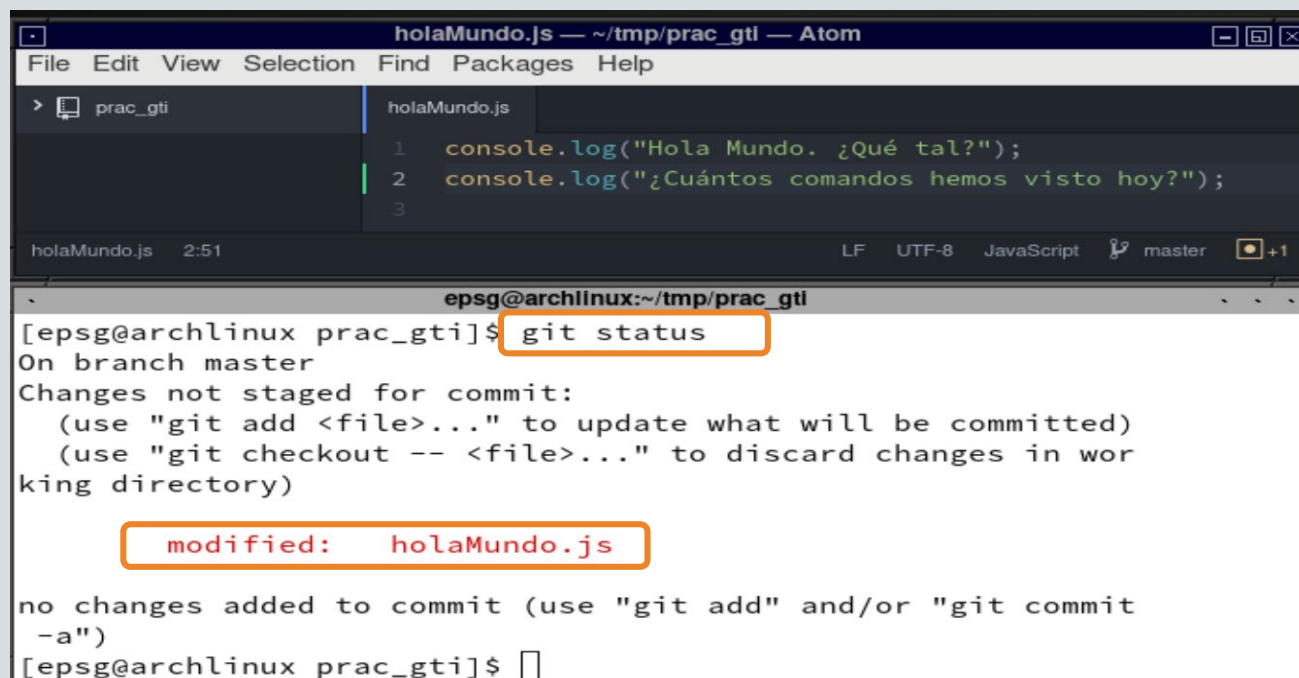

git stash



GUARDAR CAMBIOS DE FORMA TEMPORAL:

Ejercicio

1. Modifica el fichero holaMundo.js y mira el estado del repositorio:



The screenshot shows the Atom text editor with the file `holaMundo.js` open. The file contains two lines of JavaScript code: `console.log("Hola Mundo. ¿Qué tal?");` and `console.log("¿Cuántos comandos hemos visto hoy?");`. Below the editor, a terminal window shows the command `git status` being executed. The output indicates that the file `holaMundo.js` has been modified but is not yet staged for commit. The terminal text is as follows:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ 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:   holaMundo.js

no changes added to commit (use "git add" and/or "git commit -a")
[epsg@archlinux prac_gti]$
```


git stash



GUARDAR CAMBIOS DE FORMA TEMPORAL:

Ejercicio

- Guardemos de forma temporal los cambios:

```
$ git stash
```

- Mira el estado del repositorio de nuevo:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git status
On branch master
nothing to commit, working tree clean
[epsg@archlinux prac_gti]$
```

- Veamos que se ha creado un nuevo registro en la lista temporal de cambios, pero no se ha creado ningún commit:

```
$ git stash list
```

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



git stash



GUARDAR CAMBIOS DE FORMA TEMPORAL:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git stash list
stash@{0}: WIP on master: b943f6a Cambiado el mensaje a mostrar
[epsg@archlinux prac_gti]$ git log --oneline --graph
* b943f6a Cambiado el mensaje a mostrar
* b06d27c Creado el hola mundo
[epsg@archlinux prac_gti]$
```



git stash



GUARDAR CAMBIOS DE FORMA TEMPORAL:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git stash list
stash@{0}: WIP on master: b943f6a Cambiado el mensaje a mostrar
[epsg@archlinux prac_gti]$ git log --oneline --graph
* b943f6a Cambiado el mensaje a mostrar
* b06d27c Creado el hola mundo
[epsg@archlinux prac_gti]$
```

```
holaMundo.js — ~/tmp/prac_gti — Atom
File Edit View Selection Find Packages Help
> prac_gti
holaMundo.js
1 console.log("Hola Mundo. ¿Qué tal?");
2
holaMundo.js 2:1 LF UTF-8 JavaScript master
```



git stash



GUARDAR CAMBIOS DE FORMA TEMPORAL:

Ejercicio

5. Apliquemos de nuevo los cambios, de forma que recuperemos nuestro segundo mensaje

```
$ git stash apply
```

6. Tras aplicarlo, git te informa con el estado del repositorio. Fíjate en el contenido de tu fichero de nuevo.



git stash



ETIQUETAR UNA CONFIRMACIÓN

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git stash apply
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:   holaMundo.js

no changes added to commit (use "git add" and/or "git commit -a")
[epsg@archlinux prac_gti]$
```



```
holaMundo.js — ~/tmp/prac_gti — Atom
File Edit View Selection Find Packages Help
> prac_gti holaMundo.js
1 console.log("Hola Mundo. ¿Qué tal?");
2 console.log("¿Cuántos comandos hemos visto hoy?");
3

holaMundo.js 2:1 LF UTF-8 JavaScript master +1
```



git stash



ETIQUETAR UNA CONFIRMACIÓN



git tag

- Permite etiquetar un commit en concreto con un nombre representativo para nosotros:
 - Suele usarse para etiquetar versiones estables de una aplicación

```
$ git tag -a nombreEtiqueta -m "descripción etiqueta"
```

- También permite conocer todas las etiquetas creadas:

```
$ git tag
```



ETIQUETAR UNA CONFIRMACIÓN:

Ejercicio

1. Vamos a confirmar nuestros cambios, de forma que dejemos el directorio de trabajo limpio y todos nuestros cambios guardados en el repositorio local:

```
$ git commit -a -m "Dos mensajes de consola"
```

2. Vamos a crear un tag para el último commit que hemos creado:

```
$ git tag -a v0.0 -m "Versión inicial de la aplicación"
```

3. Saquemos el listado de tags y la descripción del que hemos creado:

```
$ git tag  
$ git show v0.0
```



git tag



ETIQUETAR UNA CONFIRMACIÓN:

```
epsg@archlinux:~/tmp/prac_gti
[epsg@archlinux prac_gti]$ git tag
v0.0
[epsg@archlinux prac_gti]$ git show v0.0
tag v0.0
Tagger: svalero <svalero@dsic.upv.es>
Date: Sun Jan 28 12:44:35 2018 +0100

Version inicial de la aplicación

commit 5503b427f06116dcc59bcc3212cc604f307f9501
Author: svalero <svalero@dsic.upv.es>
Date: Sun Jan 28 12:44:04 2018 +0100

    Dos mensajes de consola

diff --git a/holaMundo.js b/holaMundo.js
index 8a605ca..64ab902 100644
--- a/holaMundo.js
+++ b/holaMundo.js
@@ -1,2 @@
 console.log("Hola Mundo. ¿Qué tal?");
+console.log("¿Cuántos comandos hemos visto hoy?");
[epsg@archlinux prac_gti]$
```

Listado de tags, sólo hay uno

Datos del tag

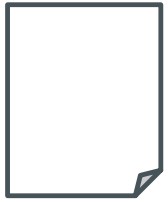
Datos del commit
que etiqueta



git tag



INDICAR QUÉ FICHEROS NO DEBE CONSIDERAR



.gitignore

- Fichero en el que se indica a git qué ficheros debe obviar
- Se debe llamar “.gitignore”
- Debe estar guardado en la raíz del directorio del proyecto
- Es posible obtener de internet ficheros ya configurados para todo tipo de lenguajes: <https://github.com/github/gitignore>
- Por ejemplo, para node



INDICAR QUÉ FICHEROS NO DEBE CONSIDERAR: PARA NODE

.gitignore

```
# Logs
```

```
logs
```

```
*.log
```

```
# Dependency directories
```

```
node_modules/
```

```
jspm_packages/
```

```
# Optional npm cache directory
```

```
.npm
```

Comentario

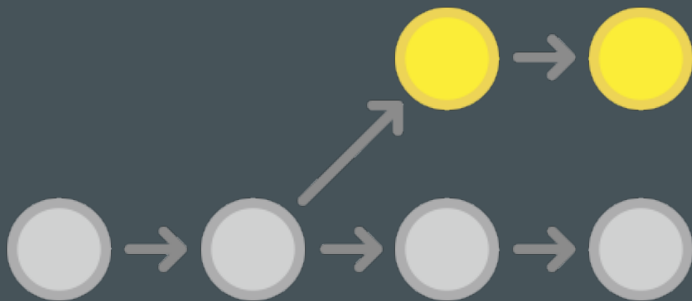
Todos los ficheros con extensión .log

Todo el contenido del directorio



RESUMEN DE COMANDOS

REFERENCIA RÁPIDA



COMANDOS MÁS USADOS

- **git clone:** permite crearse una copia de un repositorio remoto a tu repositorio local.
- **git commit:** agregar una confirmación con los últimos cambios realizados.
- **git push:** sube confirmaciones al repositorio remoto.
- **git pull:** baja confirmaciones al repositorio local.
- **git branch:** Listar ramas
- **git branch rama_local:** Crea una rama a partir de donde está HEAD.
- **git checkout rama_local:** mueve el directorio de trabajo a la rama indicada.
- **git checkout -b rama_local:** Crea una rama y mueve el directorio de trabajo a la rama indicada.
- **git push origin rama_local:** sube una rama local al repositorio remoto.
- **git fetch:** actualiza la base de datos del repositorio local con la información del repositorio remoto.



COMANDOS MÁS USADOS

- **git init:** convierte un directorio en un repositorio git local.
- **git config:** configurar tu herramienta git para todos los proyectos
- **git status:** conocer el estado de tu directorio de trabajo: ficheros preparados para confirmar, ficheros modificados, sin seguimiento, etc.
- **git add:** añadir ficheros o directorios para seguimiento. Añadir las modificaciones de un fichero en seguimiento al siguiente commit.
- **git log:** ver el histórico de confirmaciones o commits.
- **git checkout:** mover el directorio de trabajo al estado indicado por un commit en concreto. Cambiar de rama.
- **git reset:** deshacer cambios.
- **git stash:** almacenar cambios de forma temporal.
- **git tag:** etiquetar un commit.



BIBLIOGRAFÍA

- Página oficial GIT: <https://git-scm.com/>
- Libro en línea en castellano: **Pro Git** 2nd ed. Edition (2014). Scott Chacon. Ben Straub. Ed. Apress <https://git-scm.com/book/es/v2>
- Gajda, W., 2013. Git recipes : a problem-solution approach,
- Pidoux, E., 2014. Git best practices guide : master the best practices of Git with the help of real-time scenarios to maximize team efficiency and workflow,
- Daityari, S., 2015. Jump start git

