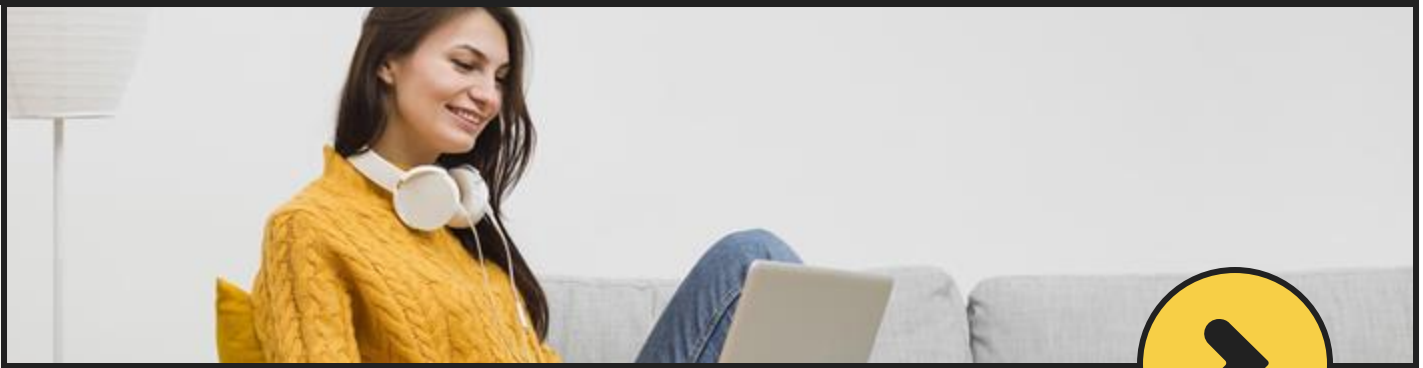
Desarrollo Web en Entorno Cliente

Alicia García Martín
CIFP Camino de la Miranda – 2024/25



Control de versiones con git

¿Por qué necesitamos control de versiones?

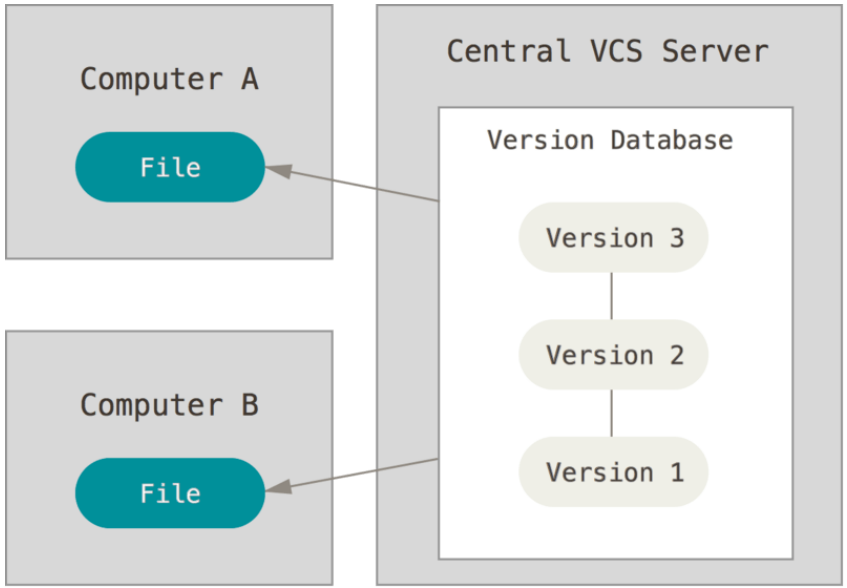
- Permite la colaboración: varias personas pueden trabajar en la misma base de código simultáneamente
- Documenta los cambios que hemos realizado
- Nos facilita deshacer errores

etc,

etc,

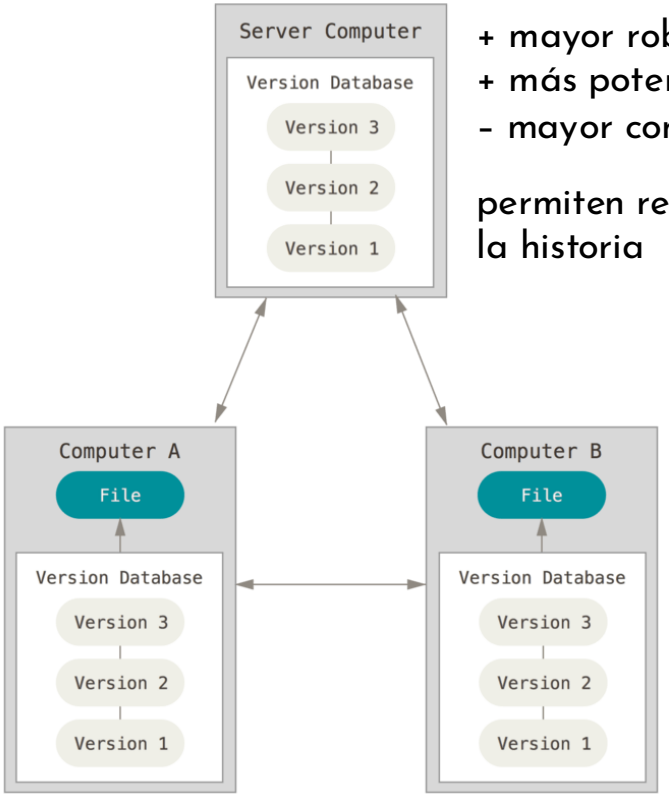
etc...

Centralizados



+ más sencillos
+ más auditables
- menos potentes
- mayor fragilidad

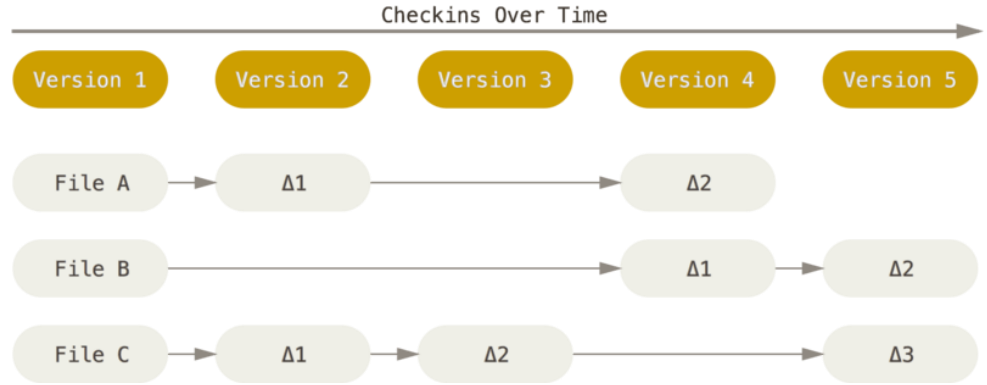
Distribuidos



+ mayor robustez
+ más potente
- mayor complejidad

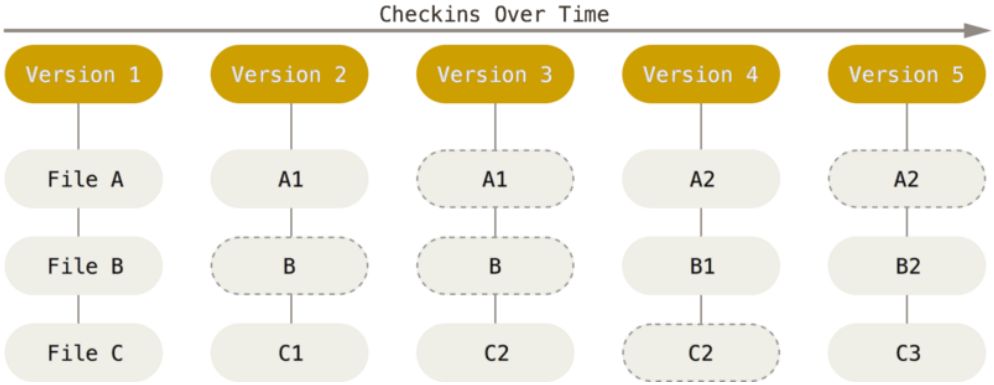
permiten reescribir la historia

git



Almacenamiento de datos como cambios en una versión de la base de cada archivo:

- lento
- potencia limitada



Almacenamiento de datos como instantáneas del proyecto a través del tiempo:

- \+ velocidad
- \+ potencia

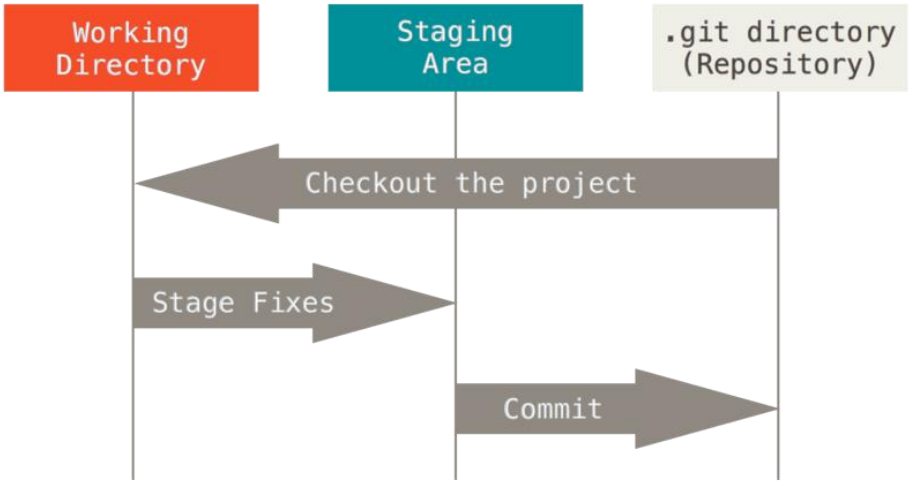
git

Casi todas las operaciones son locales: puedo trabajar off-line.

Integridad: cada elemento (archivo, commit,...) se identifica por su hash SHA-1, por lo que no puede haber corrupción de datos sin que git la identifique.

Es difícil perder información: la mayoría de operaciones sólamente **añaden** información, no la modifican ni la eliminan.

git



Un archivo versionado está en uno de tres estados: confirmado (committed), modificado (modified), y preparado (staged).

- **Confirmado**: significa que los datos están almacenados de manera segura en la base de datos local.
- **Modificado**: significa que se ha modificado el archivo pero todavía no se ha confirmado a la base de datos.
- **Preparado**: significa que se ha marcado un archivo modificado para que se incluya en la próxima confirmación.

git

- Instalación
- Configuración

```
$ git config --global user.name "John Doe"
$ git config --global user.email johndoe@example.com
$ git config --global core.editor emacs
```

- Ayuda

```
$ git help <verb>
$ git <verb> --help
$ man git-<verb>
```

Flujo de trabajo

Nuevo repositorio

- Inicializar repositorio en el directorio actual

```
$ git init
```

- Marcar archivos como "preparados" (staging)

```
$ git add *.java
$ git add README.md
```

- Confirmar cambios (commit)

```
$ git commit -m "Commit inicial"
```

- Ver listado de cambios

```
$ git log
```

Flujo de trabajo

Repositorio existente

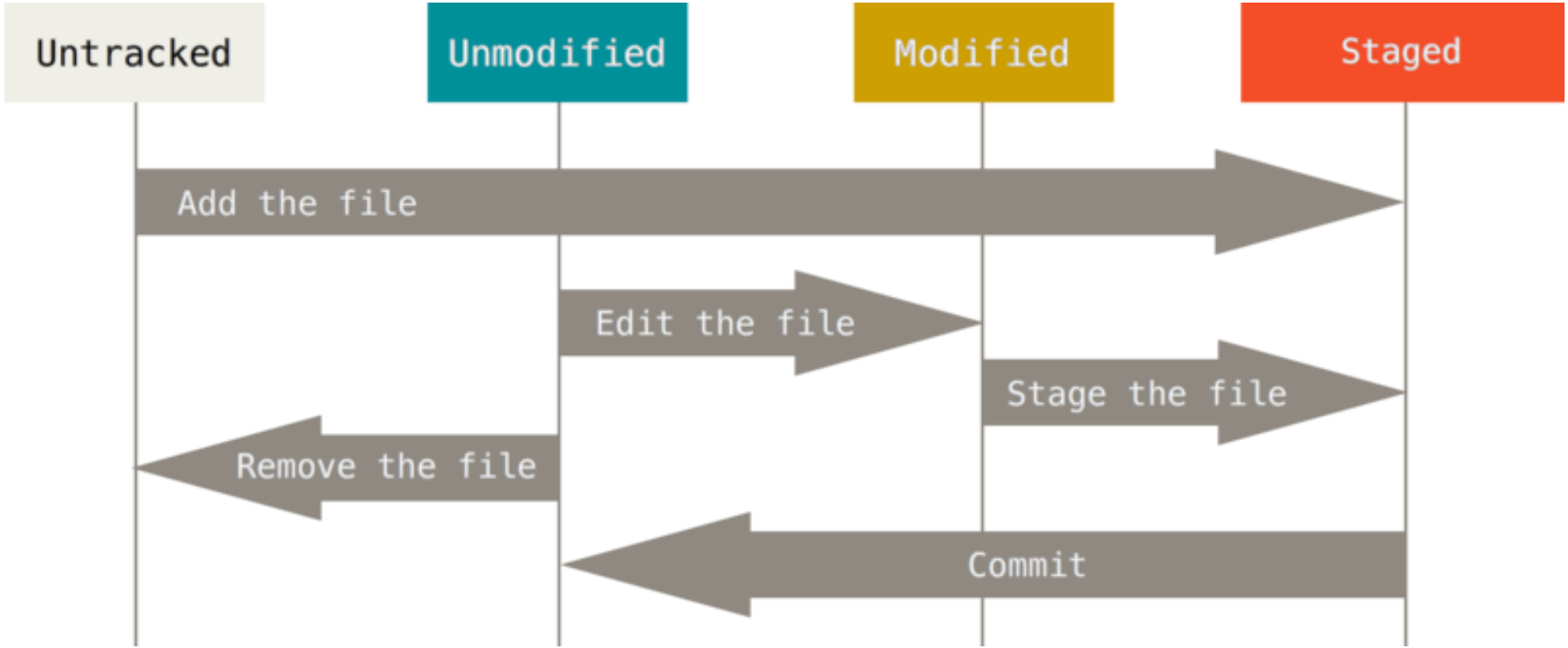
- Clonar repositorio remoto

```
$ git clone https://github.com/libgit2/libgit2
$ git clone git://github.com/libgit2/libgit2
$ git clone usuario@servidor.com:/ruta/a/libgit2
(...)
$ cd libgit2
```

- Clonar repositorio remoto con otro nombre

```
$ git clone https://github.com/libgit2/libgit2 mi_proyecto
(...)
$ cd mi_proyecto
```

Ciclo de vida de un archivo



Ciclo de vida de un archivo

- Ver estado del repositorio (“limpio”)

```
$ git status
On branch master
nothing to commit, working directory clean
```

- Cuando hay cambios

```
$ echo 'My Project' > README
$ git status
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    README

nothing added to commit but untracked files present (use "git add" to track)
```

Ciclo de vida de un archivo

- Comenzar a rastrear archivos nuevos

```
$ git add README
On branch master
nothing to commit, working directory clean
```

- Ahora el archivo está preparado (*staged*, listo para confirmar):

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
```

Ciclo de vida de un archivo

- Añadir cambios de archivos ya rastreados. Supongamos que hemos editado el archivo CONTRIBUTING.md que ya está rastreado:

```
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README

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:   CONTRIBUTING.md
```

Ciclo de vida de un archivo

- Añadir cambios al próximo commit

```
$ git add CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
    modified:   CONTRIBUTING.md
```

Ciclo de vida de un archivo

- Si ahora modifico un archivo preparado de nuevo:

```
$ vim CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
    modified:   CONTRIBUTING.md

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:   CONTRIBUTING.md
```

Ciclo de vida de un archivo

- Añado los nuevos cambios al próximo commit

```
$ git add CONTRIBUTING.md
$ git status
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   README
    modified:   CONTRIBUTING.md
```

- Confirmo los cambios (commit)

```
$ git commit
$ git commit -m "Descripción del commit"
```

- Confirmar cambios directamente, sin pasar por staging **(cuidado!)**

```
$ git commit -a -m "Descripción del commit"
```

Monitorizar cambios

- Ver listado de cambios

```
$ git log
```

- Ver contenido del último cambio

```
$ git show
```

- Ver contenido de un cambio concreto (identificado por su SHA-1)

```
$ git show a65fb3
```

- Ver cambios no preparados en el repositorio actual

```
$ git diff
```

- Ver cambios preparados en el repositorio actual

```
$ git diff --cached
```

Borrar/renombrar archivos

- Borrar archivo y marcar el cambio como preparado

```
$ git rm README.txt
```

- Quitar archivo del área de preparación

```
$ git rm --cached *.tmp
```

- Borrar todos los archivos que cumplen un patrón (glob)

```
$ git rm log/\*.log
$ git rm \*~
```

- Renombrar archivo

```
$ git mv README.md README
```

equivale a:

```
$ mv README.md README
$ git rm README.md
$ git add README
```

Deshacer cosas

- Arreglar el último commit

```
$ git commit --amend
```

- Despreparar un archivo preparado

```
$ git reset HEAD README.md
```

- Deshacer cambios no preparados

```
$ git checkout -- README.md
```

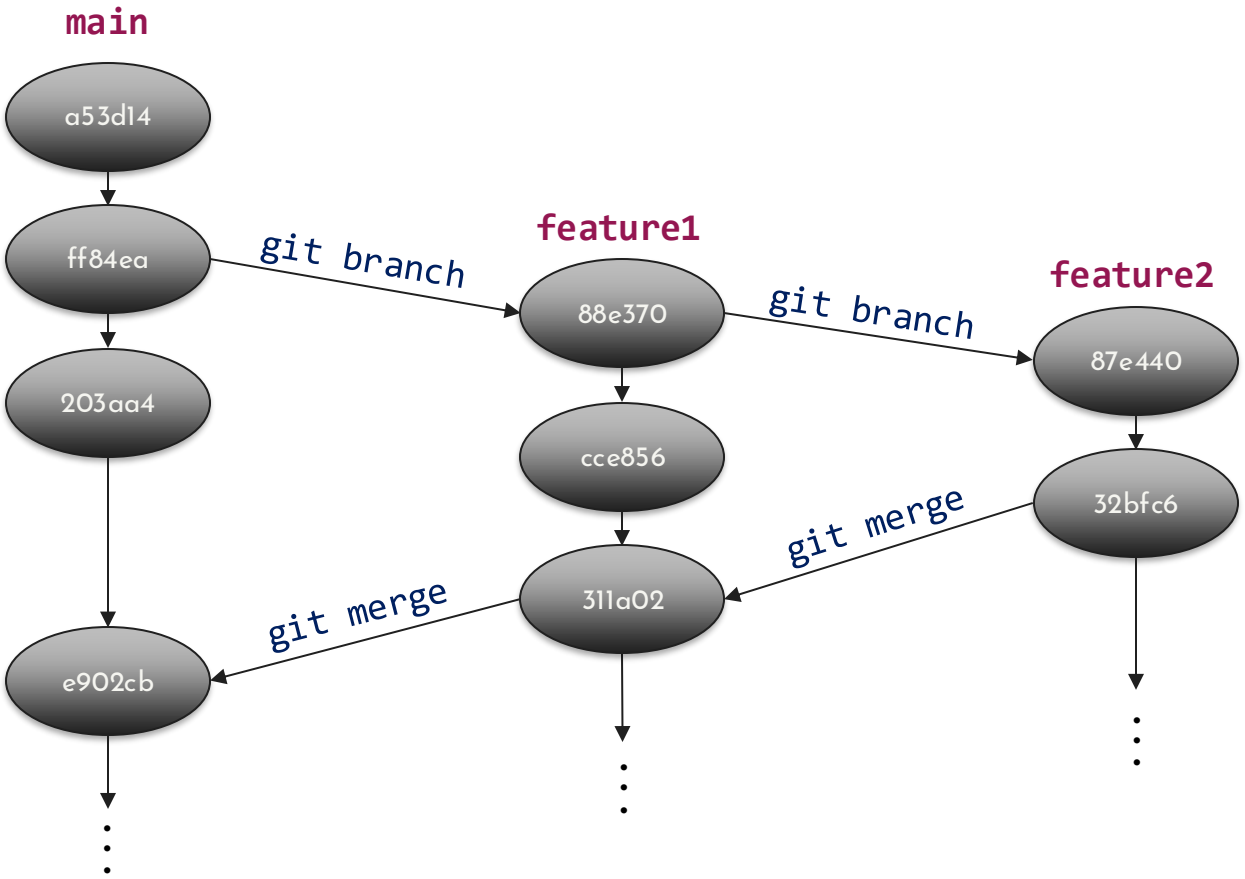
- Deshacer ("pisar") cambios a lo bruto **(cuidado!)**

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

Ramificación

- Una de las ventajas de git es la facilidad para ramificar (branch) la historia del repositorio y fusionar las ramas creadas.
- Esto permite un flujo de trabajo en el que múltiples cambios y versiones del código se pueden desarrollan en paralelo, y posteriormente incorporar a la base de código principal.
- En git podemos ramificar a partir de cualquier punto, cualquier cantidad de niveles, y podemos (intentar) fusionar cualquier rama en la actual.
- Cuando fusionamos dos ramas, puede que aparezcan conflictos si los cambios realizados en ambas son incompatibles o si git no logra entenderlos correctamente. En ese caso tendremos que solucionar los conflictos manualmente y reintentar.

Ramificación



Ramificación

- Trabajar en una rama existente llamada “foo”

```
$ git checkout foo
```

- Crear una rama nueva llamada “foo” a partir del estado actual

```
$ git branch foo
```

- Crear una rama nueva llamada “foo” y comenzar a trabajar en ella

```
$ git checkout -b foo
```

es equivalente a

```
$ git branch foo
$ git checkout foo
```

Ramificación

- En condiciones normales, siempre estamos trabajando en una rama. La rama por defecto suele llamarse **master** o **main**.
- Trabajar fuera de una rama es una situación anómala (detached HEAD) que debemos evitar salvo en muy contadas ocasiones.
- Git nos avisará de esta circunstancia anómala al provocarla y cada vez que miremos el estado del proyecto.

```
❯ git co acbea0ae085bd3ac7cec17d80404888a85421cb9
A       ejercicio_06.js
Note: switching to 'acbea0ae085bd3ac7cec17d80404888a85421cb9'.

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 switching back to a branch.

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

  git switch -c <new-branch-name>

Or undo this operation with:

  git switch -

Turn off this advice by setting config variable advice.detachedHead to false

HEAD is now at acbea0a Añade Ejercicios 1 y 2
 ~/p/cl/solid-couscous  @acbea0ae *1 +1 !1
```

Ramificación

```
❯ git s
HEAD detached at acbea0a
Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        new file:   ejercicio_06.js

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   ejercicio_06.js

 ~/p/cl/solid-couscous  @acbea0ae *1 +1 !1
```

Ramificación

- Para fusionar dos ramas, primero nos situamos en la rama destino:

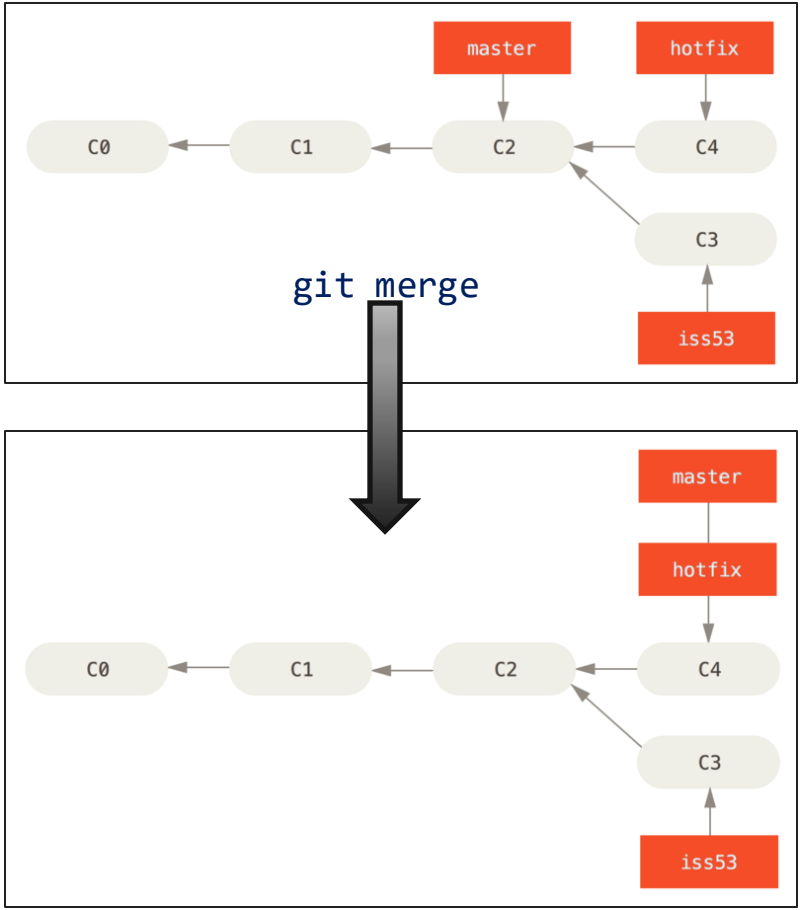
```
$ git checkout master
```

- A continuación realizamos la fusión:

```
$ git merge hotfix
```

- Una vez fusionada una rama, podemos borrarla si no la necesitamos más

```
$ git branch -d hotfix
```



Remotos

- Dado que git es descentralizado, necesitamos mecanismos para coordinar las distintas copias del repositorio existentes en distintos clientes.
- Un "remoto" representa una copia remota del repositorio.
- Cuando hemos clonado el repositorio a partir de otro, encontraremos un remoto por defecto: **origin**
- De lo contrario, necesitaremos configurarlo a mano usando

```
$ git remote add origin <URL>
```

- Ver lista de remotos

```
$ git remote -v
```

Remotos

- Para bajarnos los cambios existentes en el remoto que no nos hemos traído aún a nuestro repositorio local, pero sin fusionarlos

```
$ git fetch origin
```

- Para fusionar los cambios que acabamos de bajarnos

```
$ git merge origin/master
```

- Para bajarnos los cambios y fusionarlos (fetch+merge=pull)

```
$ git pull
$ git pull origin2 strange_branch
```

- Para enviar nuestros cambios locales al repositorio

```
$ git push
$ git push origin local_branch
```

Varios

- Podemos excluir uno o varios archivos o tipos de archivos, evitando que git los incluya en nuestros commits, añadiéndolos al archivo **.gitignore**
- El archivo .gitignore ha de formar parte de nuestro repositorio
- Podemos marcar un punto concreto de la historia de nuestro repositorio asignándole una **etiqueta**:

```
$ git tag v1.0
$ git tag v1.0 main
$ git tag v1.0 e374cff
```

- Una etiqueta se puede utilizar como si fuera un commit o una rama:

```
$ git checkout v1.0
$ git push origin v1.0
```

Github

- Tokens de autenticación en lugar de contraseña de usuario
- Fork: crear una copia propia del repositorio en tu cuenta. Se mantiene vinculada al repositorio original y permite interactuar.
- Pull Request.

Práctica

- Crea una cuenta de GitHub y un token de autenticación.
- Haz un fork del repositorio **alicia-ciclos/git-intro**
- Clona a tu ordenador la versión que acabas de crear.
- Crea un archivo llamado practica.txt, con cualquier contenido.
- Añade y confirma dicho archivo.
- Sube los cambios a tu repositorio.
- Haz una Pull Request desde tu repositorio al repositorio original.
- Envía la URL de dicha Pull Request como respuesta a la tarea asignada en Teams.

Referencias

- [es] Libro oficial de git, disponible en varios formatos
  https://git-scm.com/book/es/v2
- [en] Cómo resolver conflictos al fusionar ramas
  https://hs2010-git.github.io
- [en] Presentación muy completa
  https://git.mikeward.org
- [en] Otra presentación interesante
  https://courses.cs.washington.edu/courses/cse403/13au/lectures/git.ppt.pdf
- Stackoverflow, Google, YouTube...

GRACIAS!

Preguntas?

alicew.garmar@educa.jcyl.es