git

Profesor: Vladimir Bataller

git

Febrero 2018

Introducción

Características e instalación

git

Introducción

Profesor Vladimir Bataller

Características

Git es un sistema de control de versiones con gran aceptación actualmente. Fue desarrollado en 2005 por Linus Torvalds para gestionar las versiones del desarrollo de Linux.

Es una herramienta que se emplea en la línea de comandos aunque hay herramientas con interface gráfica que simplifican el trabajo, no consiguen aplicar todas las opciones de la línea de comandos.

En resumen se podría decir que git sirve para:

Control de versiones / Respaldo del software/ Sistema colaborativo/ Registro de cambios

Documentación:

https://git-scm.com/documentation

Tutorial:

https://www.atlassian.com/git/tutorials/setting-up-a-repository

Descarga:

https://git-scm.com/

Comandos Git

Los comandos git se ejecutan desde una consola con la instrucción "git" seguida del nombre del comando.

Para ver la ayuda de git y ver todos los comandos disponibles, se debe teclear:

git --help

Si se desea ver la ayuda de un comando en concreto, se pone el nombre del comando entre git y --help. Por ejemplo para ver la ayudad del comando init, se tecleará:

git init --help

Credenciales

Comandos para dar de alta de forma global (para todos los repositorios, por defecto) el nombre y email del desarrollador, para que cualquier operación que se realice, se sepa quien la ha realizado:

git config --global user.name "Vladimir Bataller"

git config --global user.email "vladimirbataller@yahoo.es"

Para consultar todos los parámetros que tiene configurados git tenemos el comando:

git config --list

o si se desea ver solamente un párametro, por ejemplo user.name

git config user.name

Carpeta del proyecto

En local, el usuario guarda el proyecto en una carpeta (directorio de trabajo) que será la que esté bajo la supervisión de git.

A continuación se describen algunos archivos y carpetas típicos de los repositorios git públicos de bibliotecas de código abierto. archivos:

README.md	archivo de resumen del proyecto en formato GitHub markdown.
LISENCE	archivo con la licencia del proyecto

Elementos de gestión de git que residen en la carpeta:

.git	carpeta oculta que contiene la información de los cambios que se han producido.
.gitignore	archivo de texto en el que especificamos que archivos y carpetas no se deben supervisar por git.

Elementos git

Profesor Vladimir Bataller

archivo .gitignore

El archivo .gitignore permite indicar mediante una lista de patrones de nombres qué archivos no se deben monitorizar ni sincronizar con el repositorio (ni local, ni remoto). Consultar su sintaxis en: **https://git-scm.com/docs/gitignore**

Puede haber un archivo .gitignore en cada carpeta y se aplicará a esa carpeta y sus subcarpetas.

Ejemplo de .gitignore

```
/node_modules
/build
/war
.DS_Store
.env.local
.env.development.local
.env.test.local
.env.production.local
npm-debug.log*
yarn-debug.log*
yarn-error.log*
*-lock.json
```

Ciclo de trabajo básico

Elementos del ciclo de trabajo básico

git

Ciclo de trabajo básico

Profesor Vladimir Bataller

Directorio de trabajo

El directorio, área o árbol de trabajo es la carpeta raíz a partir de la cualse desarrolla el proyecto.

Inicialización del directorio de trabajo. Estando ubicados en tal directorio, teclear:

git init

Esto genera el directorio de commits que es oculto y se llama ***.git*** y contiene la bbdd con toda la información de los commits que se han ido realizando en el repositorio local.

Una forma alternativa de crear un directorio de trabajo es clonando un repositorio remoto existente:

git clone *https://github.com/vladimirbat/ws_react.git*

En este caso, creará una carpeta, dentro del directorio actual, con el directorio de proyecto clonado. Si cambiamos a él podremos ver su contenido:

cd ws_react

Ciclo de trabajo básico

Profesor Vladimir Bataller

Ramas (I)

En un proyecto git, se pueden seguir varias líneas de desarrollo en paralelo, en todo momento se debe ser consciente de en que rama se está trabajando. Una rama se puede ver como un puntero a un determinado commit

La línea de desarrollo por defecto es la rama ***master*** que es la rama de la versión principal. Normalmente no se trabaja directamente sobre ella, se trabaja sobre ramas paralelas de desarrollo y una vez los cambios están probados y aceptados, se pueden incorporar a la rama master.

Para crear una nueva rama (por ejemplo *desarrollo*), se ejecutará:

git branch *desarrollo*

Para ver las ramas existentes se debe teclear:

git branch

En nuestro ejemplo mostrará:

desarrollo

*** master**

El asterisco indica la rama actual. Para moverse a la rama desarrollo se tecleará:

git checkout *desarrollo*

Ramas (II)

Normalmente la fusión con la rama master solamente la hace una persona.

Si se emplea la consola de Git, en el símbolo del sistema, cuando estamos en una carpeta de proyecto, indica e que rama nos encontramos, por ejemplo si estamos en la rama *desarrollo* se mostrará:

git (*desarrollo***)**

Para incluir una nueva funcionalidad, se suele crear una rama específica para esa funcionalidad y luego se fusionará con la de desarrollo (con la opción -b de checkout, además de cambiar a la rama, esta se crea):

git checkout -b *nueva-funcionalidad*

Ciclo de trabajo básico

Profesor Vladimir Bataller

Fases de un archivo de una rama

Los archivos de un proyecto, tienen un ciclo de vida relativo al estado en que se encuentran respecto a git.

Untracked: Archivo no monitorizado por git, no añadido con add, ni existente en commits anteriores.

Tracked: Archivo monitorizado por git.

Modified: Que tiene cambios realizados.

Staged: Añadido (comando **add**) para hacer un commit (o snapshot).

Commited: Cambios guardados en el repositorio local (comando **commit**).

	UNTRACKED	TRACKED	MODIFIED	STAGED	COMMITED
INICIO	⌂				
CAMBIOS			⌂		
ADD		⌂		⌂	
COMMIT		⌂			⌂
CAMBIOS		⌂	⌂		

Ciclo de trabajo básico

Profesor Vladimir Bataller

git status

El comando git status permite conocer el estado de los archivos del proyecto.

Si en la rama actual no se han cambiado los archivos, al ejecutar **git status**, se mostrará:

On branch *nueva-funcionalidad*

nothing to commit, working directory clean

Si ahora se modifica un archivo, por ejemplo *index.html*, y se ejecuta de nuevo **git status**, se verá:

Changes not stagged for commit:

modified: *index.html*

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

Si se emplea la opción -s se ve el status en formato abreviado:

git status -s

Y esto mostrará en es ejemplo lo siguiente:

M *index.html*

donde ***M*** representa ***Modified***.

Ciclo de trabajo básico

Profesor Vladimir Bataller

git add

El comando git add permite indicar que archivos se quieren añadir al índice de cambios (Staging área) para ser enviados en el próximo commit al repositorio local.

A continuación se añade el archivo que se desea preparar para ser confirmado (commit), para ello se debe ejecutar:

git add *index.html*

y se ejecuta de nuevo **git status**, se verá:

STAGED

Changes to be commited:

modified: *index.html*

Ciclo de trabajo básico

Profesor Vladimir Bataller

Índice de cambios

El índice o área de cambios (staging area) es la colección de archivos que están supervisados y que si cambian serán enviados en el próximo commit.

Para añadir un archivo al índice o área de cambios se debe ejecutar la siguiente instrucción:

git add *nombreArchvivo*

git add *listaDeArchivosSeparadosPorEspacios*

git add .

Para quitar los archivos del índice se emplea el comando

git reset .

git reset *nombreArchivo*

Ciclo de trabajo básico

Profesor Vladimir Bataller

git commit

Commit almacena, en el repositorio local, los cambios en la rama actual de los archivos añadidos al índice de cambios (Staging área).

Los commits siembre se ejecutan sobre la rama actual, supongamos que nos encontramos en la rama *nueva-funcionalidad.* Para ejecutar un commit, se suele agregar la opción -m para incluir un comentario:

git commit -m *"Cambio en index.html"*

Si en algún commit hemos cometido un error, se puede realizar un nuevo commit que enmienda el anterior y elimina sus cambios, para eso se emplea la opción ***-amend*** . Solamente se debe emplear si el commit anterior no ha sido distribuido ya que podríamos generar inconsistencias.

Si se cambia a otra rama:

git checkout *desarrollo*

y se abre de nuevo el archivo, se verá que los cambios del archivo desaparecen.

Si se retorna a la rama donde se hicieron los cambios, estos volverán a ser visibles (no se pone -b porque esta rama ya existe):

git checkout *nueva-funcionalidad*

Identificador de commit

Cada commit tiene un hash asociado que es un número de 40 dígitos hexadecimales generados con un algoritmo SHA1. Normalmente, para identificar un commit solamente se emplean los 7 u 8 primeros dígitos de este identificador.

Para ver los identificadores de los commits de una rama, se ejecuta el siguiente comando:

git log --oneline

Al poner la opción ***--oneline*** se muestran solamente los identificadores cortos y el comentario del commit.

bb4a4ba *fin del merge de nueva-funcionalidad*

...

Si se agrega la opción ***--all***, se muestran, ordenados por fecha, todos los commits de todas las ramas. Si se quiere limitar el número de commits mostrados se puede poner un número precedido de menos por ejemplo ***-9***.

Si se desea ver el grafo completo de commits del repositorio, además de ***--all***, se incluirá la opción ***--graph***.

Para comparar dos commits, se puede emplear el comando **git diff**

El comando **git show** muestra las diferencias de un commit con el anterior.

Ciclo de trabajo básico

Profesor Vladimir Bataller

git

git push

Para enviar los cambios de una rama al servidor central se emplea el comando ***git push***. Para ello se debe haber dado un alias a la url del servidor central, por defecto este alias es siempre ***origin***.

Si el repositorio ha sido clonado (mediante ***git clone***) entonces la url del repositorio central ya está almacenada y asociada al alias ***origin***, pero si es un repositorio recién creado, entonces se debe indicar dicha url con el comando ***remote***:

git remote add origin *https://github.com/vladimirbat/ws_react.git*

Para enviar los cambios de la rama con la nueva funcionalidad, se ejecutará:

git push origin *nueva-funcionalidad*

Si no se hubieran subido cambios con anterioridad, el sistema solicitará el usuario y contraseña para realizar los cambios en el repositorio central. Para subsiguientes envíos, esta información no se volverá a solicitar.

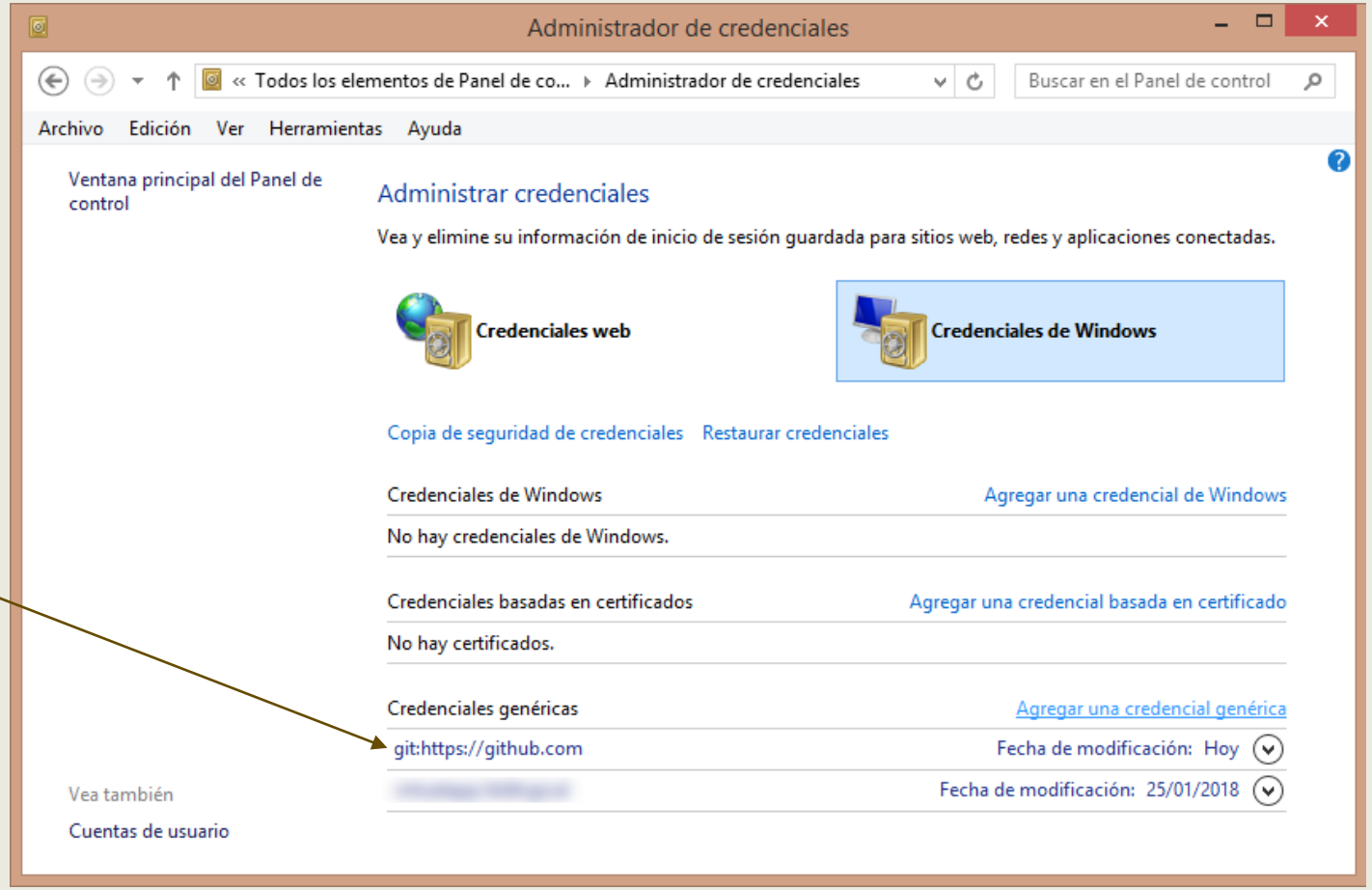
Hay que tener mucho cuidado y poner correctamente el nombre de la rama ya que si en su lugar se pusiera ***master***, esto haría que se se sobrescribiera la rama ***master***.

Para enviar los cambios de todas las ramas, en lugar del nombre de la rama, se indica --all (y -q para que sea más conpacto), así que este comando será:

git push --all -q origin

Dónde se guardan las credenciales en Windows

Panel de control > Administrador de credenciales > Credenciales Windows



Gestión de ramas

git

Gestión de ramas

Profesor Vladimir Bataller

Referencia a commits de ramas

Las rama se pueden considerar punteros a su último commit y a partir de ellos se puede hacer referencia a los commits anteriores mediante la sintaxis que se explica a continuación.

Padres

HEAD: es un sinónimo del último commit de la rama actual.

HEAD^ (o HEAD^1): es el commit padre (inmediatamente anterior) de HEAD.

Cuando un commit proviene de la fusión de varias ramas (y por lo tanto de varios commits), se dice que tiene varios padres. En ese caso HEAD^2 sería el segundo padre y sería el último commit de una rama que se ha fusionado con la actual (con el commit actual).

Antecesores o ancestros

Para referirse a los commits estrictamente de la rama actual, se emplea ~. De modo que HEAD~1 sería el commit anterior al actual y HEAD~2 sería el anterior del anterior al commit actual. Nota: siempre HEAD^1 = HEAD~1

Referirse a cualquier rama

Para referirse a padres (^) o ancestros(~) de cualquier otra rama distinta de la actual (HEAD), en lugar de HEAD se puede emplear el nombre de la rama con los commits a los que nos queramos referir. Por ejemplo: master~3 o desarrollo^2.

Gestión de ramas

Profesor Vladimir Bataller

Grafo de commits

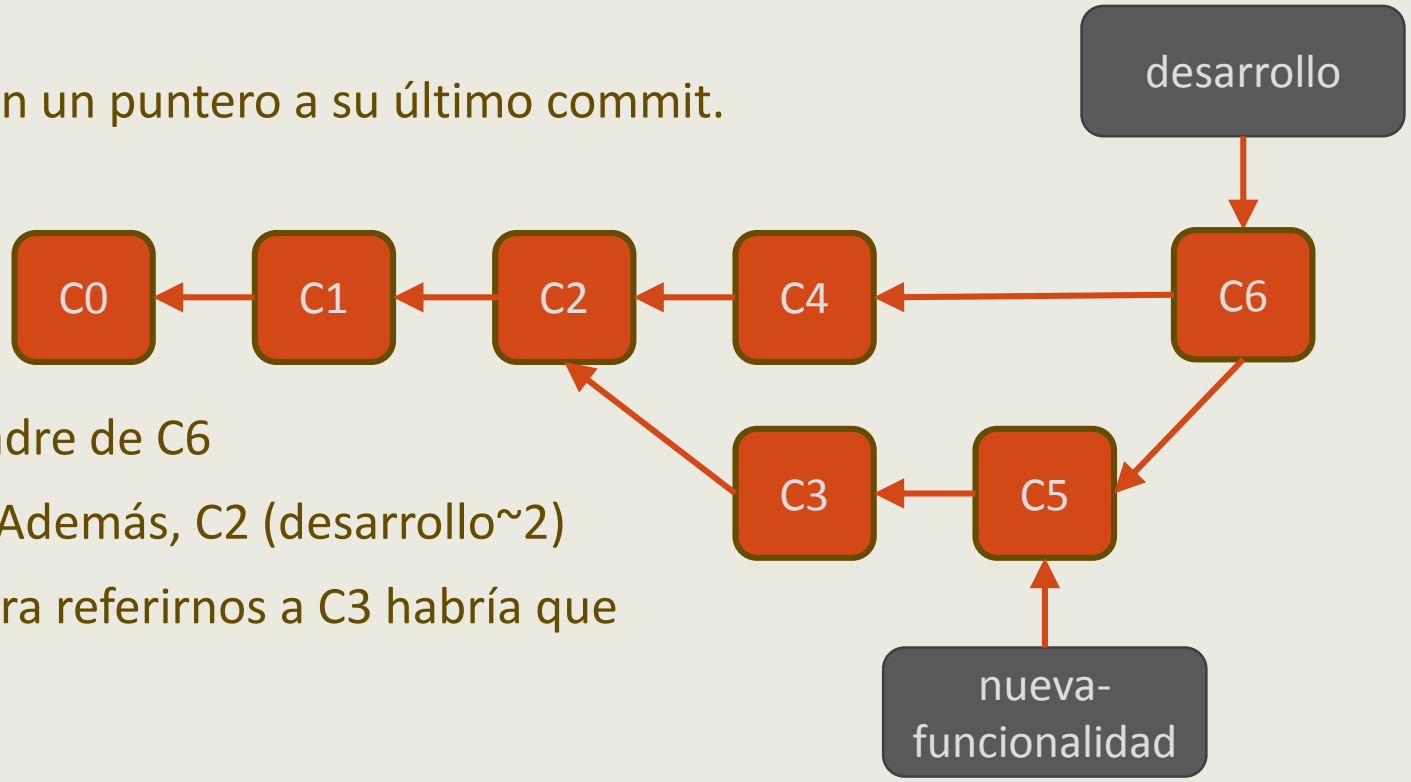
El grafo de commits muestra la relación de generación entre los commits de un repositorio. Por lo que muestra sus dependencias y pertenencia a ramas.

Las flechas apuntan en el sentido de la dependencia, indicando que el commit origen de la flecha se ha originado a partir del commit destino de la flecha.

Cuando de un commit salen varias flechas, indica que se ha compuesto por la fusión de varios commits pertenecientes a diferentes ramas.

Las ramas se representan como un nodo con un puntero a su último commit.

Historia de un commit: es la secuencia de commits ordenados por fecha que preceden al commit en el grafo.



Padres: C4 (desarrollo^1) sería el primer padre de C6 y C5 sería (desarrollo^2 su segundo padre. Además, C2 (desarrollo~2) sería su segundo ancestro. Por otro lado para referirnos a C3 habría que indicar nueva-funcionalidad~2.

git merge

El comando git merge se emplea para volcar los cambios de una rama sobre otra.

Supongamos que queremos volcar los cambios de la rama desarrollo sobre la rama master. Para ello debemos estar en la rama master:

git checkout *master*

y fusionar sobre merge la rama desarrollo:

git merge *desarrollo*

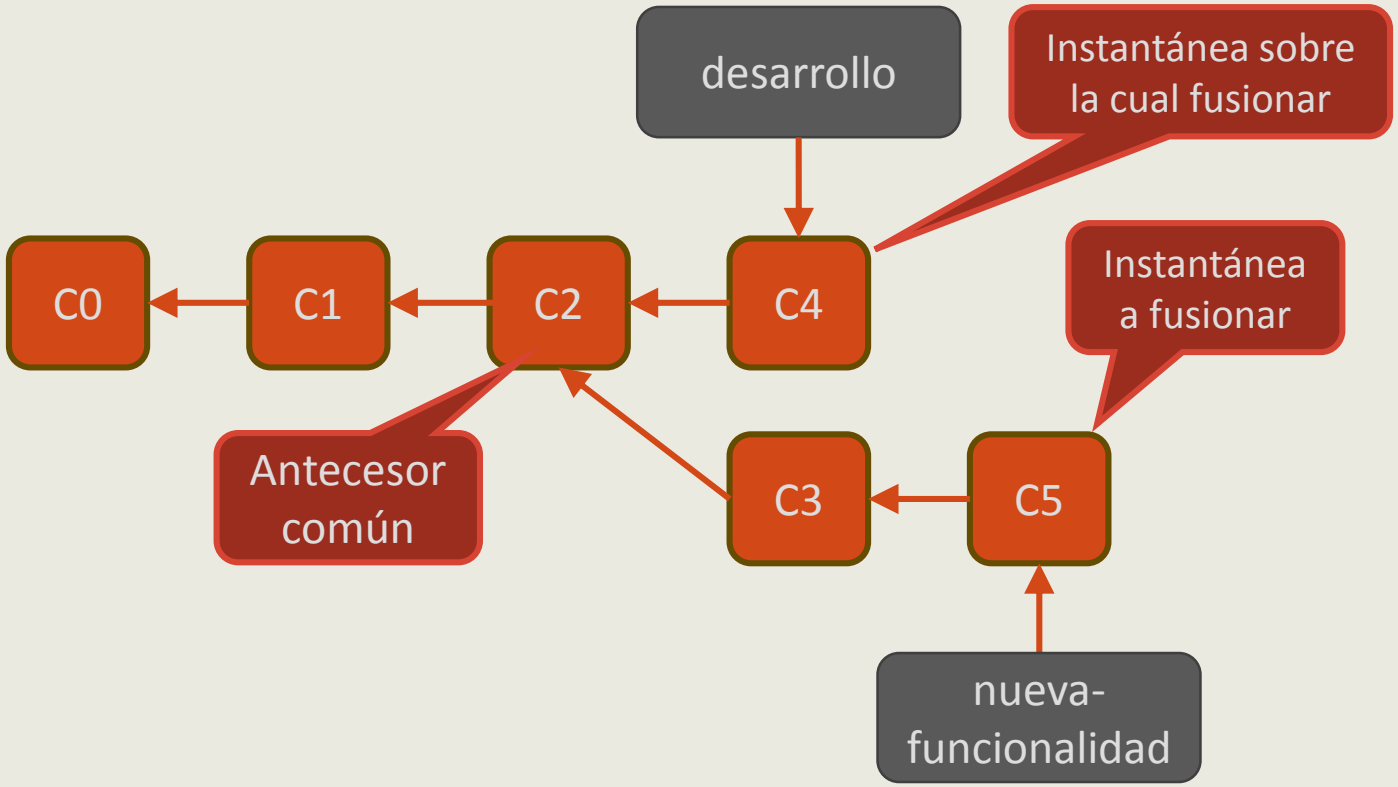
Gestión de ramas

Profesor Vladimir Bataller

git

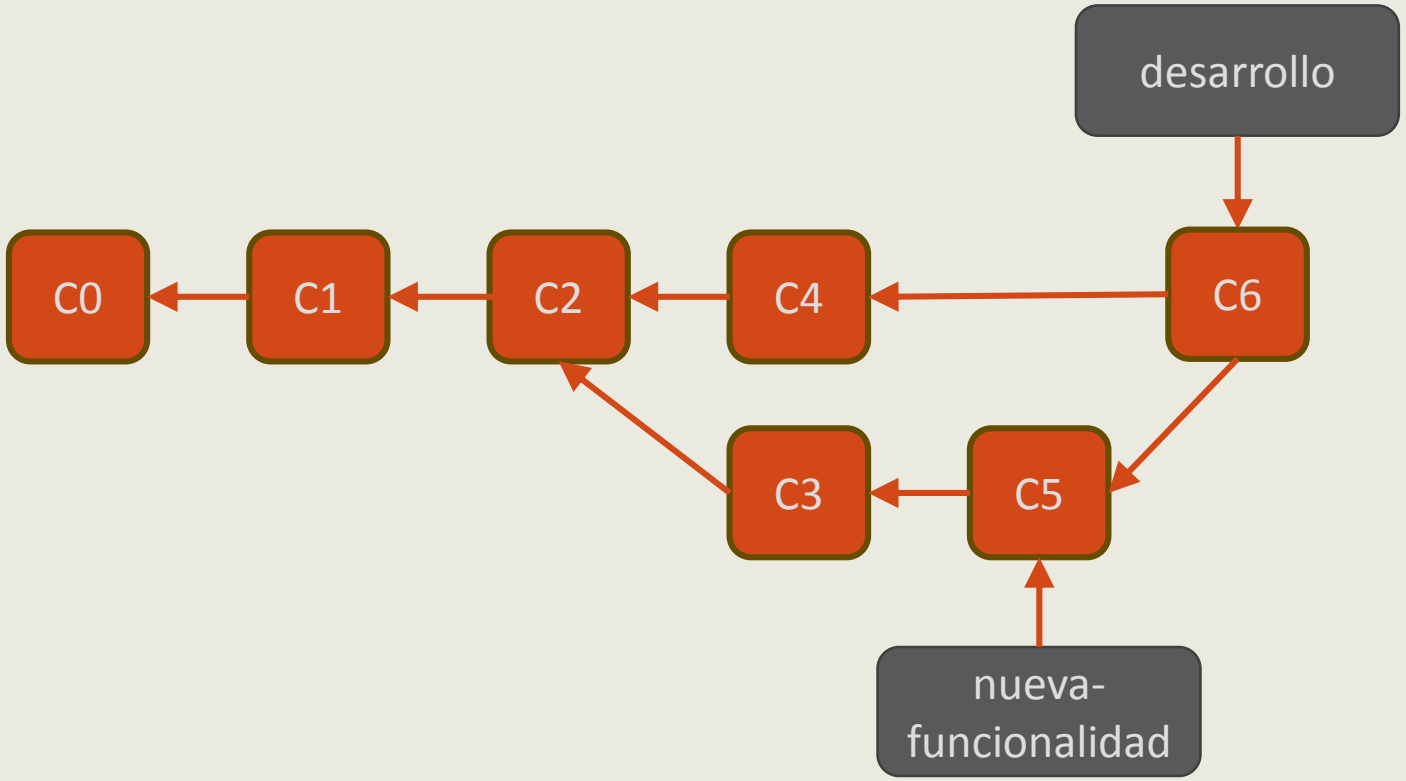
Procedimiento interno de fusión de commits entre ramas

Git buscará el commit (C2) desde el cual se separó la rama que se quiere fusionar (nueva-funcionalidad) de la rama con la que se quiere fusionar. Ambas ramas pueden haber experimentado commits (C4 en desarrollo y C3 y C5 en nueva-funcionalidad).



Procedimiento interno de fusión de commits entre ramas

Tras el merge, la rama de desarrollo tendrá un nuevo comit que recoge los comits de ambas ramas fusionadas. Normalmente, la rama fusionada (nueva-funcionalidad) se elimina y se sigue trabajando con la fusionada.



Gestión de ramas

Profesor Vladimir Bataller

Comandos para explorar la historia de una rama (I)

Git proporciona varios comandos para ver la historia de una rama o un commit. En ellos se emplea la sintaxis de referencia a los commits de las ramas vista con anterioridad.

git log

git log -3	muestra los tres últimos commits de la rama actual (HEAD).
git log --oneline -3	tres últimos commits de la rama actual (HEAD) en formato abreviado.
git log --oneline desarrollo	muestra todos los comits de la rama desarrollo.
git log --oneline desarrollo~2	muestra el commit del ancestro 2 y anteriores ancestros.
git log --oneline --all	muestra todos los commits independientemente de la rama.
git log --oneline --all --graph	muestra el grafo de todo el repositorio.

```
$ git log --oneline --all --graph
*   a234a59 (HEAD -> master) fusion
|\
| * be0dd43 (desarrollo) modificado
* | 664be4c he añadido un titulo h2
|/
* e2d7704 (origin/master) equivalen
* a4980a1 ejemplos
* 82792b9 dependencias
* 126bf37 local js
* 5d52daa inicial
```

Gestión de ramas

Profesor Vladimir Bataller

Comandos para explorar la historia de una rama (II)

El comando git diff muestra las diferencias entre dos commits, si se ha declarado la herramienta P4Merge, esta también se puede emplear para ver las diferencias.

git diff

git dif rama1 rama2	compara dos ramas
git diff master~2	campara el commit indicado con el anterior
git diff rama1 rama2 --NombreArchivo	solamente compara el archivo indicado en las ramas indicadas

git show

git show rama	muestra los metadatos de la rama y diferencias con el commit anterior
git show	muestra metadatos de HEAD y diferencias con el commit anterior

Comandos para explorar la historia de una rama (III)

El comando git reset permite quitar commits.

git reset

git reset commit	Elimina todos los commits desde el HEAD hasta el commit indicado. Moviendo el puntero de la rama a dicho commit. Es decir el commit indicado será el nuevo HEAD de la rama actual. Deja las diferencias de esos commits quedan en el directorio de trabajo, como modificaciones; por lo que todos esos cambios se pueden enviar en un único y nuevo commit. De esta forma se pueden comprimir varios commits en uno solo. ejemplo: git reset desarrollo~2
git reset commit --hard	Mueve le puntero HEAD al commit indicado, pero NO guarda los cambios de los commit eliminados en el directorio de trabajo. PELIGRO DE PERDIDA DE CÓDIGO.

Gestión de ramas

Profesor Vladimir Bataller

Estrategia automática de merge

Git, por defecto, al realizar un *merge*, emplea una estrategia recursiva automática para realizar la fusión de los directorios de trabajo procedentes de los dos commits implicados.

Estado de los archivos	Comportamiento de git en el merge
Iguales en ambos commits	Al ser común, incluye el fichero tal cual (sin conflicto)
Está solamente en uno de los commits	incluye el fichero (sin conflicto)
Con diferencias disjuntas (diferentes secciones del código)	une ambos ficheros con auto-merge (sin conflicto)
Con ancestro común en sus historias	incluye el último fichero (sin conflicto)
Con diferencias en la misma sección del código	une ambos, marca diferencias y genera conflicto

Si al acabar, no hay conflictos, genera un commit de integración (de tipo auto-merge).

Si al acabar, hay conflictos, la rama queda en estado ***merging*** y los conflictos se deben resolver a mano y hacer explícitamente el commit. Los archivos en conflicto estarán en el estado ***unmerged*** y los que no estén en conflicto estará en el estado ***staged***.

En el caso de encontrar conflictos, se puede cancelar la integración y por tanto no generar commit:

git merge --abort

Gestión de conflictos (I)

Si se intenta hacer un merge de dos ramas que han cambiado la misma porción de código dentro del mismo archivo, entonces se producirá un conflicto.

Al hacer esto:

git checkout *master*

git merge *desarrollo*

Si hubiera conflictos, se mostraría esto:

Auto-merging *index.html*

CONFLICT (content): Merge conflict in *index.html*

Automatic merge failed; fix conflicts and then commit the result.

En el símbolo del sistema se mostrará: **desarrollo|MERGING** . Para ver qué archivos permanecen sin fusionar, se debe teclear:

git status

Entre otras líneas, se mostrará:

both modified: index.html

Gestión de conflictos (II)

Los archivos en conflicto se verán modificados para recoger los valores de ambas versiones y que el desarrollador decida si se queda con uno o con otro o realiza una fusión a mano.

Si se abre el archivo en conflicto, se verá:

<<<<<<< HEAD:index.html

* © Mi empresa *

=======

* Derechos reservados a Mi empresa *

>>>>>>> nueva-funcionalidad:*index.html*

Todo lo que está sobre **=======** es lo de la rama ***HEAD*** (***desarrollo*** en este ejemplo) y lo que está debajo es lo de la rama que se quiere fusionar (***nueva-funcionalidad***). Se deben resolver de forma manual los cambios y dejar el bloque en conflicto, por ejemplo de la siguiente forma:

* © Mi empresa. Todos los derechos reservados *

Posteriormente se debe ejecutar:

git add

Gestión de ramas

Profesor Vladimir Bataller

P4Merge

Es una herramienta de gestión de conflictos en Merges integrable con git. La url de descarga es: **https://www.perforce.com/downloads/visual-merge-tool**

Una vez instalado, por ejemplo en *C:\Program Files\Perforce*, para poder usarlo desde git, se debe declarar como herramienta de diferencias y gestión de merges:

```
git config --global diff.tool p4merge
git config --global difftool.p4merge.path 'C:\Program Files\Perforce\p4merge.exe'
git config --global merge.tool p4merge
git config --global mergetool.p4merge.path 'C:\Program Files\Perforce\p4merge.exe'
```

Para lanzar la herramienta de gestión de conflictos, si se debe ejecutar: **git mergetool**

Una vez guardados los cambios fusionados en la herramienta, en la consola se mostrará:

Merging: *index.html*

Normal merge conflict for '*index.html***':**

{local}: modified file

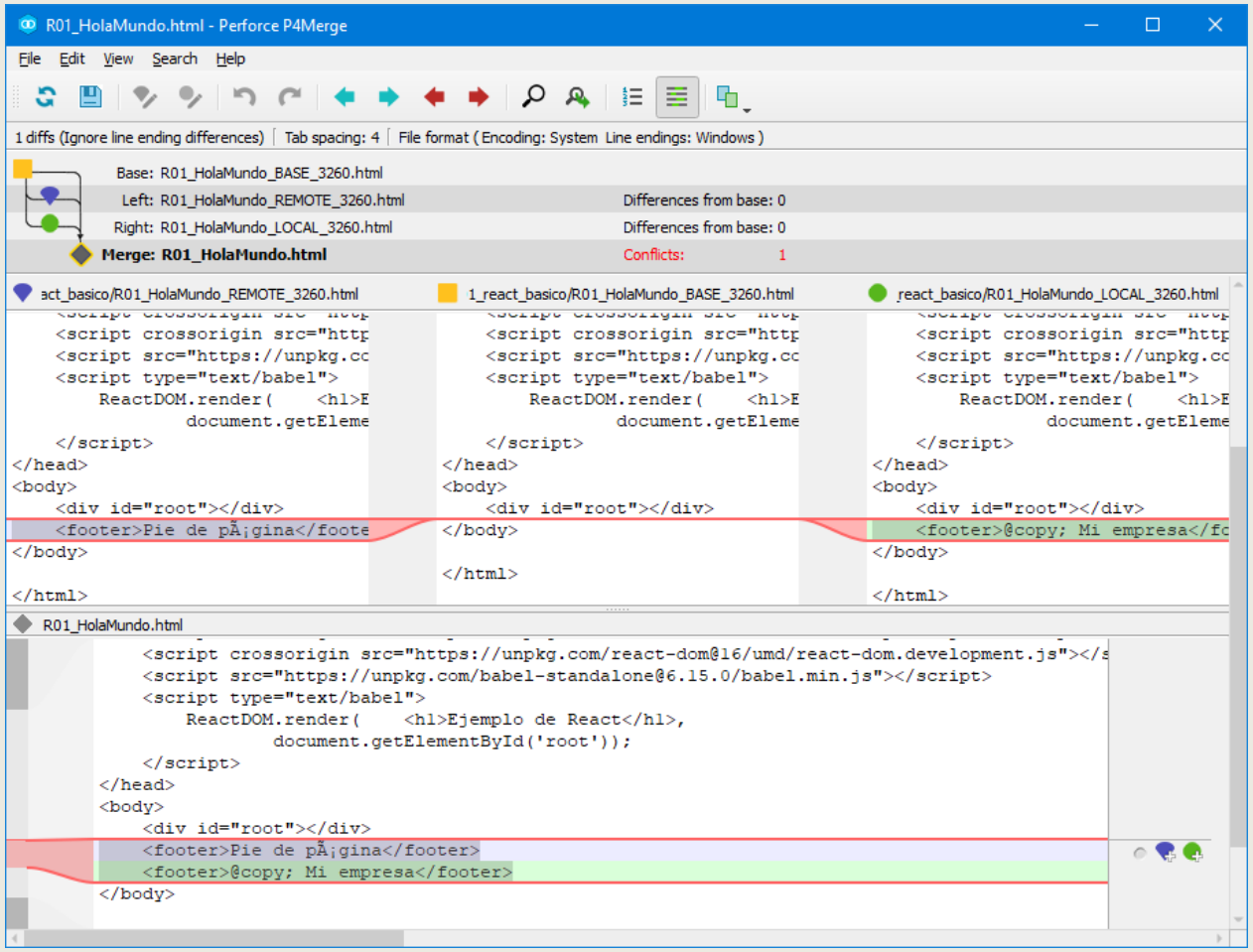
{remote}: modified file

Gestión de ramas

Profesor Vladimir Bataller

P4Merge

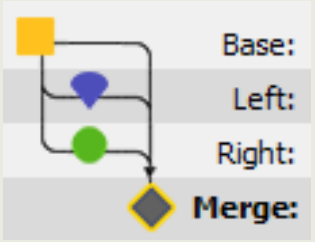
Al ejecutar **git mergetool**, se mostrará la herramienta que muestra las dos instantáneas en conflicto (una a cada lado) y en el centro la instantánea original. en la parte inferior se muestra como quedará el archivo final.



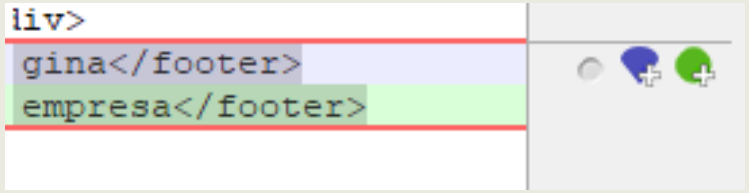
P4Merge

Al ejecutar **git mergetool**, se mostrará la herramienta que muestra las dos instantáneas en conflicto (una a cada lado) y en el centro la instantánea original. en la parte inferior se muestra como quedará el archivo final.

En la parte superior se muestra un esquema de como se separaron las dos versiones de la versión original, cada una tiene asignado un color y una forma para identificarla en toda la herramienta.



En la parte inferior, donde se muestra como quedará el archivo final, se puede seleccionar mediante los iconos de las versiones, cual o cuales de ellas incorporarán código. Además se puede editar manualmente.



Finalizar el merge en conflicto

Una vez se hayan realizado los cambios bien manualmente en el editor de código o bien con la herramienta de gestión de merges, el proceso se termina con un commit.

Para ver el estado del merge, se debe ejecutar de nuevo:

git status

y se verá lo siguiente:

On branch desarrollo

All conflicts fixed but you are still merging. (use "git commit" to conclude merge)

Changes to be committed:

modified: *index.html*

Tras esto se puede realizar el commit y terminar el proceso, para ello se ejecutará:

git commit -a -m "*fin del merge de nueva-funcionalidad***"**

Para no tener que realizar un **git add** con los archivos modificados, se emplea **-a** que lo hace implícitamente.

Tras el commit, en el símbolo del sistema ya no se mostrará ***(desarrollo\|MERGING)***, mostrándose solamente ***(desarrollo)***.

borrar una rama

Ya sea porque se haya hecho un merge de la rama o porque se descarten sus cambios, puede interesar borrar una rama. A continuación se indica como realizarlo.

Borrar la rama local:

git branch -d *nueva-funcionalidad*

Enviar los cambios al repositorio remote:

git push origin *nueva-funcionalidad*

Otra alternativa para realizar ambas cosas en un solo paso es:

git push origin --delete *nueva-funcionalidad*

Gestión de ramas

Profesor Vladimir Bataller

Sincronizar los cambios de un repositorio local

Antes de empezar una nueva rama y por lo tanto un nuevo desarrollo, será una buena práctica descargar los últimos cambios que puedan haber realizado otros usuarios y hayan sido subidos al repositorio remoto.

Para descargar los cambios desde el repositorio remoto (en este caso de la rama master):

git pull origin *master*

Para traer las ramas existentes en el repositorio remoto:

git fetch *nombreRama*

Para traer todas las ramas del repositorio remoto:

git fetch --all

Gestión de ramas

Profesor Vladimir Bataller

Fast Forward [FF] (I)

La técnica ***Fast Forwoard*** consiste en hacer la integración de dos ramas, integrando primero la principal (o master) sobre la secundaria (o desarrollo) y luego una vez resuelta esta, hacer la integración de la rama secundaria (o desarrollo) sobre la principal (o master).

Supongamos que partimos de un desarrollo realizado sobre una rama llamada *desarrollo* que ya está terminado para integrarlo en la principal, para realizarlo al modo Fast Forward, seguiermos los siguientes pasos:

git checkout desarrollo -> Cambiar a la rama de desarrollo si no nos encontramos ya en ella.

git merge master -> Fusionar la rama master sobre la rama de desarrollo (si hay gestionar confl).

git checkout master -> Cambiar a la rama master.

git checkout -b master_V1 -> Crear una rama copia de master en la situación previa (V1) a la fusión

git checkout master -> Volver a master.

git merge desarrollo -> Fusionar desarrollo sobre master, como ya estaban fusionadas, no se realiza un commit nuevo, lo que ocurre es que master apunta al commit de la fusión. Mostrando:

```
Updating e55c911..ee465e3
Fast-forward
 index.html | 4 ++++
1 file changed, 4 insertions(+)
```

Fast Forward [FF] (I)

La técnica ***Fast Forwoard*** consiste en hacer la integración de dos ramas, integrando primero la principal (o master) sobre la secundaria (o desarrollo) y luego una vez resuelta esta, hacer la integración de la rama secundaria (o desarrollo) sobre la principal (o master).

Si tras un Fast-forward se ejecuta:

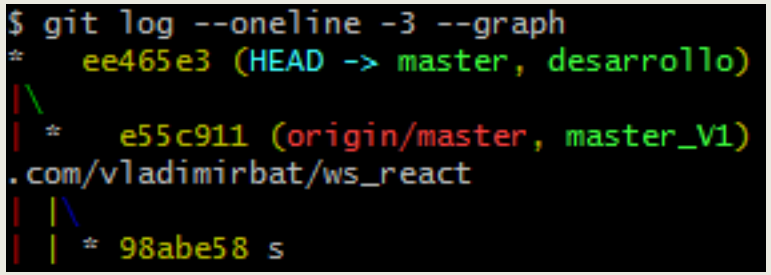
git Branch -v

Se verá que las ramas ***desarrollo*** y ***master*** apuntan al mismo commit, mientras que ***master_v1*** al commit anterior de ***master***.

y si se ejecuta:

git log --oneline -3 --graph

se verá lo mismo y además la cronología y fusiones de estas ramas:

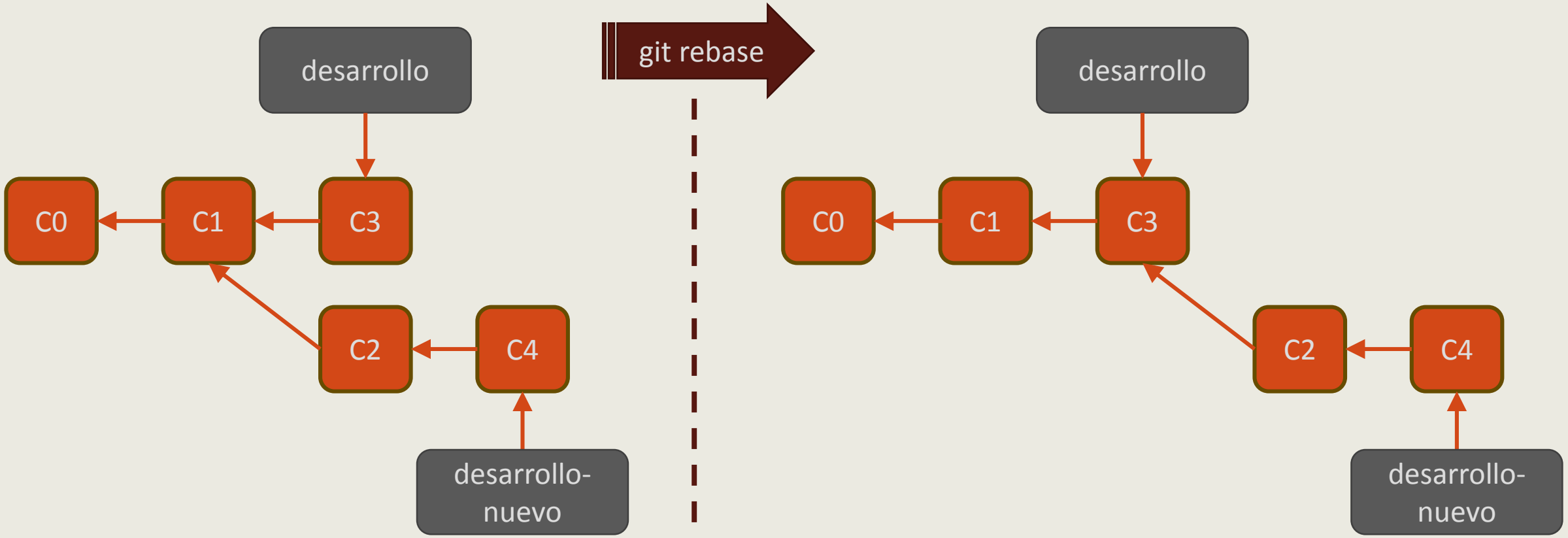


Gestión de ramas

Profesor Vladimir Bataller

Rebase (I)

El comando **git rebase** permite cambiar hacia delante en el tiempo el commit en el que una rama (por ejemplo *desarrollo-nuevo*) se separó de otra (por ejemplo *desarrollo*).



Si después del **rebase**, se hiciera un **mege** de ***nuevo-desarrollo*** sobre ***desarrollo***, este sería automático (***fast forward***)

Rebase (II)

A continuación se muestra un ejemplo de realización del rebase de las dos ramas del ejemplo anterior.

Se parte de una línea desarrollo de la que se ha derivado desarrollo-nuevo, luego se han añadido funcionalidades en ambas ramas y se han realizado los respectivos commits "Funcionalidad desarrollo" y "Funcionalidad desarrollo-nuevo".

```
$ git log --oneline --all --graph
* 2d64fa6 (HEAD -> desarrollo) Funcionalidad desarrollo
| * bb5c380 (desarrollo-nuevo) Funcionalidad desarrollo-nuevo
|/
* 4d51838 Commit inicial de desarrollo
* ecfb508 (master) Commit inicial
```

Desde desarrollo-nuevo se crea un nuevo puntero a ***desarrollo-nuevo-despues***:

git checkout -b desarrollo-nuevo-despues

Esto se hace así, para tener un puntero a la rama antes (***desarrollo-nuevo***) y otro a la rama después del rebase (***desarrollo-nuevo-después***), pero de momento apuntan al mismo commit.

Rebase (III)

A continuación se muestra un ejemplo de realización del rebase de las dos ramas del ejemplo anterior.

Tras los pasos anteriores, se puede iniciar el rebase

git rebase -q -m "*Funcionalidad desarrollo-nuevo + desarrollo***" desarrollo**

Si existen conflictos, nos dice que arreglemos los conflictos que han quedado marcados, que añadamos o quitemos los archivos con ***git add*** o ***git rm*** y que para proseguir ejecutemos ***git rebase*** con una de las tres opciones*:* ***--continue*** para seguir afirmativamente, ***--skip*** para saltarse los cambios y ***--abort*** para volver al punto inicial y abandonar el rebase.

git add *index.html*

git rebase --continue

```
$ git log --oneline --all --graph
* 9b2b186 (HEAD -> desarrollo-nuevo-despues) Funcionalidad desarrollo-nuevo + desarrollo
* 2d64fa6 (desarrollo) Funcionalidad desarrollo
| * bb5c380 (desarrollo-nuevo) Funcionalidad desarrollo-nuevo
|/
* 4d51838 Commit inicial de desarrollo
* ecfb508 (master) Commit inicial
```

Apéndice

Aspectos adicionales

git

Más comandos

git mv *oldName newName*

Permite cambiar el nombre de un archivo tanto en el disco como en todos los comits anteriores.

git rm *files*

Elimina un archivo del directorio de trabajo y registra en el índice de cambios ese borrado para que quede reflejado en el próximo commit.

git config core.editor notepad

Establece el block de notas de Windows como editor de comentarios.

git commit --amend

Permite editar el comentario de un commit.

Crear repositorios en GitHub

GitHub permite albergar repositorios remotos git de forma gratuita siempre que sean públicos. Aunque es el más difundido hay otras opciones como por ejemplo Bitbucket.

Una vez se ha dado de alta un usuario en GitHub se puede crear un nuevo repositorio con uno de los siguientes métodos:

New Repository: Crear un repositorio inicialmente vacío.

Import Repository: Copiar un repositorio a partir de su url pública.

Fork: Copiar un repositorio nuestro a otra cuenta a la que tengamos acceso.

Los repositorios de GitHub son repositorios git de tipo *bare*, esto significa que solamente almacena commits, pero no tiene directorio de trabajo ya que no se desarrolla directamente en él. La creación de un repositorio de tipo *bare* se realiza con el siguiente comando:

```
git init --bare
```