

Sistemas de control de versiones

Git y GitHub

Talleres y Seminarios de Tecnologías Emergentes I

Grado en Tecnología Digital y Multimedia

Partes I y II

F. J. Martínez Zaldívar



TELECOM ESCUELA
TÉCNICA VLC SUPERIOR
DE INGENIERIA DE
TELECOMUNICACIÓN



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Índice

- 1 Introducción
- 2 Ventana o interfaz de línea de comandos (CLI)
- 3 Interacción básica con Git en un repositorio local
- 4 Comandos `reset` y `checkout`
- 5 Ramas
- 6 Interacción con repositorios remotos
- 7 Aspectos básicos de GitHub

Índice

- 1 Introducción
- 2 Ventana o interfaz de línea de comandos (CLI)
- 3 Interacción básica con Git en un repositorio local
- 4 Comandos reset y checkout
- 5 Ramas
- 6 Interacción con repositorios remotos
- 7 Aspectos básicos de GitHub

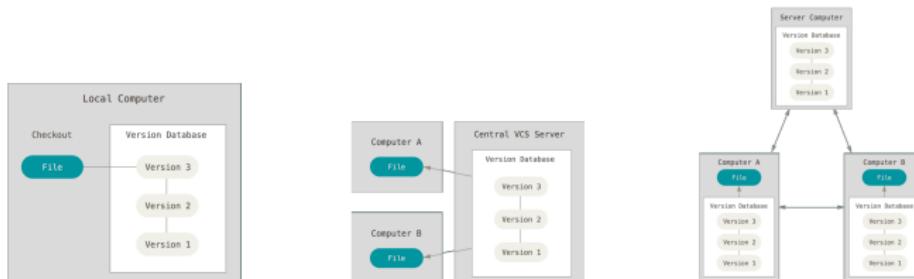
Índice

1 Introducción

- Sistemas de control de versiones
- Notación
- Contexto
- Software necesario

VCS

- VCS: Version Control System. Software que permite:
 - Seguimiento de cambios en código/documentación generada
 - Se puede volver a versiones antiguas
 - Permite realizar ramificaciones y fusiones
 - Sincroniza código entre distintas personas
- Modelos: local, centralizado y distribuido



- Genéricamente **Git**: VCS que sigue modelo distribuido y potencialmente *replicado*, aunque podríamos hacerlo encajar en cualquier modelo.
- **Git** **no** adopta modelo incremental: menos eficiente en almacenamiento pero más en velocidad

Algunas referencias y simuladores de Git

- Referencias:

- <https://git-scm.com/doc>
- <https://git-scm.com/docs/gittutorial>
- <https://www.w3schools.com/git/>
- <http://marklodato.github.io/visual-git-guide/index-en.html>
- <http://www.vogella.com/tutorials/Git/article.html>
- ...

- Simuladores de Git:

- <http://git-school.github.io/visualizing-git/>
- <https://learngitbranching.js.org/>
- No es propiamente un simulador sino un resumen gráfico interactivo sobre comandos git:
http://ndpsoftware.com/git-cheatsheet.html#loc=local_repo;

Índice

1 Introducción

- Sistemas de control de versiones
- **Notación**
- Contexto
- Software necesario

Notación

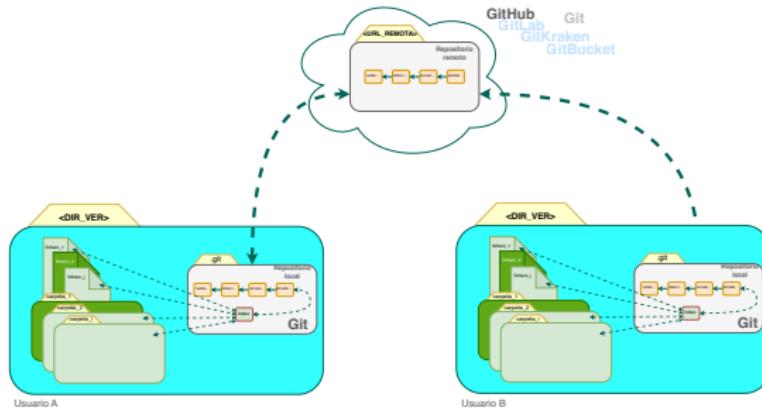
- Nombres genéricos: <IDENTIFICADOR>
 - En la práctica, deberá ser sustituido por un identificativo **concreto**.
 - Ejemplo: <URL> ⇒
 - <https://github.com/GIT-GTDM-24-25/Prueba1>
- Opcionalidad: []
 - El contenido interior puede emplearse u omitirse
 - Ejemplo: dir [<DIRECTORIO>] ⇒
 - dir
 - dir graficas
- Alternativa: |
 - Separador de alternativas posibles
 - Ejemplo: dir [graficas | texto | <RESULTADOS>] tmp ⇒
 - dir tmp
 - dir graficas tmp
 - dir texto tmp
 - dir solucion tmp

Índice

1 Introducción

- Sistemas de control de versiones
- Notación
- Contexto**
- Software necesario

Contexto y conceptos



- **Directorio versionado: <DIR._VER>**
 - Directorio que incluye estructura de ficheros y directorios de nuestro proyecto a versionar
 - También denominado: *árbol de trabajo (working tree)* o directorio de árbol de trabajo
 - Característica esencial
 - Es el directorio que incluye a la carpeta .git
- **Repositorio local:**
 - Almacenamiento estructurado de las *instantáneas* del directorio versionado
 - Situado en carpeta o directorio .git
- **Respositorio remoto: <URL_Remoto>, repositorio en la nube (productos comerciales: GitHub, GitLab, GitKraken, BitBucket, ...)**

Índice

1 Introducción

- Sistemas de control de versiones
- Notación
- Contexto
- Software necesario

Git SCM: Git Source Control Management



- Sistema de control de versiones **Git**
- Visítese <http://git-scm.com>
- Instalación:
 - Síganse las indicaciones de la página web, según plataforma

Editor de textos **plano**

- Cualquier editor será válido (debe ser de texto **plano**)



- Proponemos: Visual Studio Code



- **DISTINTO DE:** Visual Studio (es un IDE)

- Instálese desde <https://code.visualstudio.com/>

Cuenta de usuario en GitHub

- Servicio de Git en la nube  **GitHub**
- Visítese <https://github.com>
- Obténgase una cuenta gratuita
 - Se sugiere que se proporcione email de UPV para centralizar comunicaciones

Índice

- 1 Introducción
- 2 Ventana o interfaz de línea de comandos (CLI)
- 3 Interacción básica con Git en un repositorio local
- 4 Comandos reset y checkout
- 5 Ramas
- 6 Interacción con repositorios remotos
- 7 Aspectos básicos de GitHub

Índice

2 Ventana o interfaz de línea de comandos (CLI)

- CLI (*Command-Line Interface*)
- Directorios y *paths*
- Algunos comandos básicos

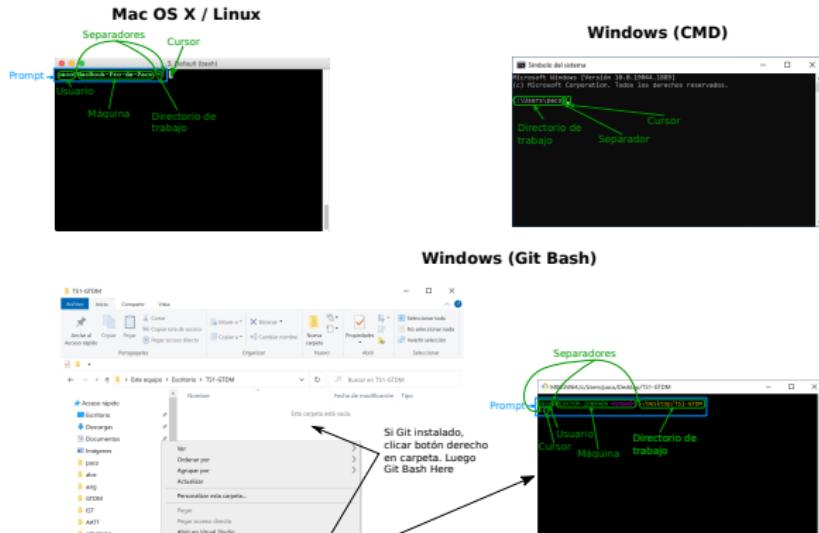
CLI y shells

- CLI: (*Command-Line Interface*)
- Es una interfaz o ventana que admite comandos escritos en una línea
 - Es una interfaz textual con el ordenador.
 - Conocido también como *ventana de línea de comandos*
- Forma originaria de interacción con el sistema operativo/máquina:
comandos escritos
- El término ha ido evolucionando con el tiempo y la tecnología
- Ejemplos CLI:
 - Linux, Mac OS X: terminal
 - Windows: CMD, PowerShell, Git Bash (si Git instalado)...
- *Shell*:
 - Programa que interpreta los comandos escritos en el CLI.
 - Ejemplos: Bash, zsh, ksh, ...

El *prompt*

- *Prompt*: invitación a introducción de comando

- Forma genérica <USUARIO>@<MAQUINA>:<DIR_TRABAJO>\$_



- Forma simplificada: \$ o bien > explícitamente en el CLI CMD de Windows
- Comentarios sobre instrucciones en CLI: # Comentario

CLI frente a GUI para comandos Git

- GUI: *Graphical User Interface*

- Más intuitivo
- Muchos interfaces
- Muy distintos entre ellos
- Falta de notación común
- Incompletos

- CLI: *Command-Line Interface*

- Menos intuitivo (al principio)
- La interfaz es única, incluso entre distintos *shells* y sistemas operativos
- Son todos ellos idénticos, pues es realmente el mismo software

- Académicamente nos decantamos por la versión CLI.

- Por simplicidad, omitiremos el *prompt* en muchos de los ejemplos de comandos Git
- ...o usaremos la versión simplificada:

- \$ en un terminal genérico (con instrucciones, en principio, ejecutables también en CMD de Windows)
- o bien > en el CMD de Windows (con instrucciones exclusivas para el CMD de Windows)).

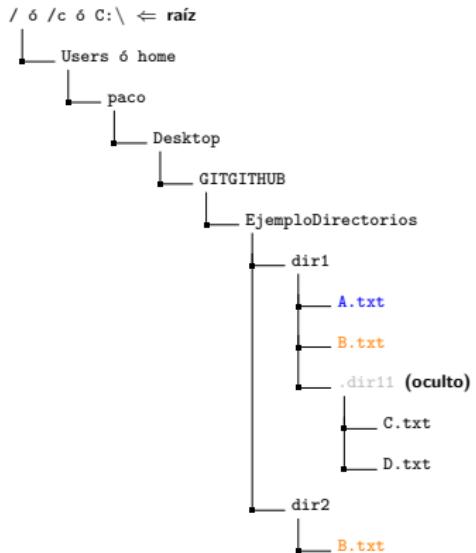
Índice

2 Ventana o interfaz de línea de comandos (CLI)

- CLI (*Command-Line Interface*)
- Directorios y *paths*
- Algunos comandos básicos

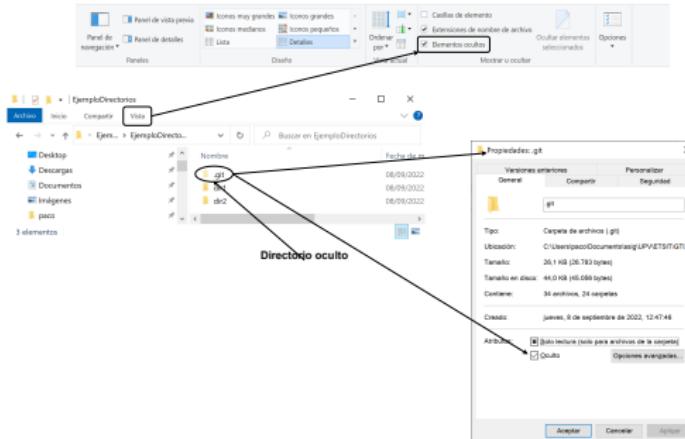
Estructura de directorios y ficheros, y paths

- Sistema de ficheros: organización de directorios y ficheros de manera estructurada
 - **raíz**: punto de partida o referencia de la estructura del sistema de ficheros
- Directorio ≡ carpeta
 - Convenio con carpetas/ficheros **ocultos**: comienzan por `[.]`
 - Tratamiento distinto según sistema operativo
- **Path**: secuencia de directorios desde cierta **referencia** hasta llegar a un fichero o directorio:
 - **Path absoluto**: si la **referencia** es la raíz de sistema de ficheros (aparece `/` ó `/c` ó `C:\`, ... al principio del **path**)
 - **Path relativo**: si la **referencia** es un directorio (no aparece raíz como primera referencia)
 - Separadores de directorios
 - `\` (Windows CMD y PowerShell)
 - `/` (Linux, Mac-OSX, Windows GitBash y PowerShell)
 - Podemos tener dos ficheros con el mismo nombre (pero distinto **path**). Ejemplo: `B.txt`



Directarios ocultos

- Windows:



- Mac OS X

- GUI: mostrar/ocultar archivos ocultos en Finder
 - Combinación COMANDO-MAYÚSCULAS-PUNTO
- CLI:
 - Mostrar archivos ocultos en un terminal: opción de listado -a: \$ ls -a
 - Ocultar: automáticamente comenzando el fichero/directorio por punto

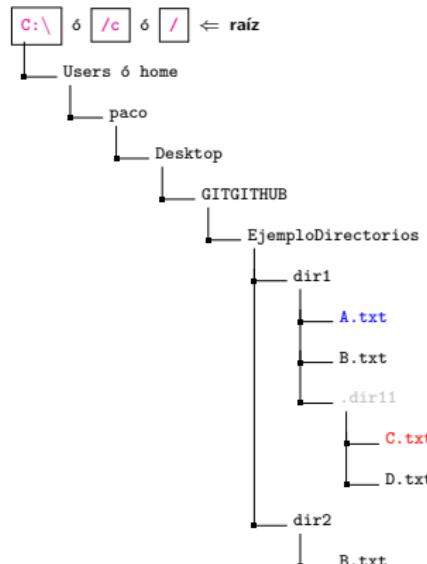
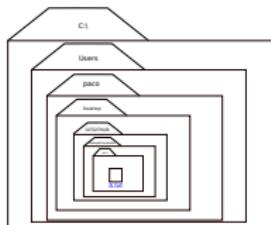
- Linux (Ubuntu)

- GUI: mostrar/ocultar archivos ocultos en administrador de archivos
 - Combinación CONTROL-H
 - O bien, menú Ver ⇒ Mostrar archivos ocultos
- CLI:
 - Mostrar archivos ocultos en un terminal: opción de listado -a: \$ ls -a
 - Ocultar: automáticamente comenzando el fichero/directorio por punto

Ejemplos de *paths* absolutos

Paths absolutos hacia A.txt

```
C:\Users\paco\Desktop\GITGITHUB\EjemploDirectarios\dir1\A.txt → Windows
c/Users/paco/Desktop/GITGITHUB/EjemploDirectarios/dir1/A.txt → Git Bash en Windows
/Users/paco/Desktop/GITGITHUB/EjemploDirectarios/dir1/A.txt → Mac OSX
/home/paco/Desktop/GITGITHUB/EjemploDirectarios/dir1/A.txt → Linux
```



Paths absolutos hacia C.txt

```
C:\Users\paco\Desktop\GITGITHUB\EjemploDirectarios\dir1\.dir11\C.txt → Windows
c/Users/paco/Desktop/GITGITHUB/EjemploDirectarios\dir1/.dir11/C.txt → Git Bash en Windows
/Users/paco/Desktop/GITGITHUB/EjemploDirectarios\dir1/.dir11/C.txt → Mac OSX
/home/paco/Desktop/GITGITHUB/EjemploDirectarios\dir1/.dir11/C.txt → Linux
```

Directarios singulares y *path* relativos

- Directarios singulares:

- **Directorio de trabajo:** directorio actual en el que se está (WD: *Working Directory*)
- : directorio **raíz del usuario o home del usuario**

- En Windows CMD: NO
- En Windows (PowerShell): `~` = `C:\Users\usuario` ó `c:/Users/usuario`
- En Windows (Git Bash): `~` = `c/Users/usuario`
- En Mac OS X: `~` = `/Users/usuario`
- En Linux: `~` = `/home/usuario`

- Directorio **padre**:

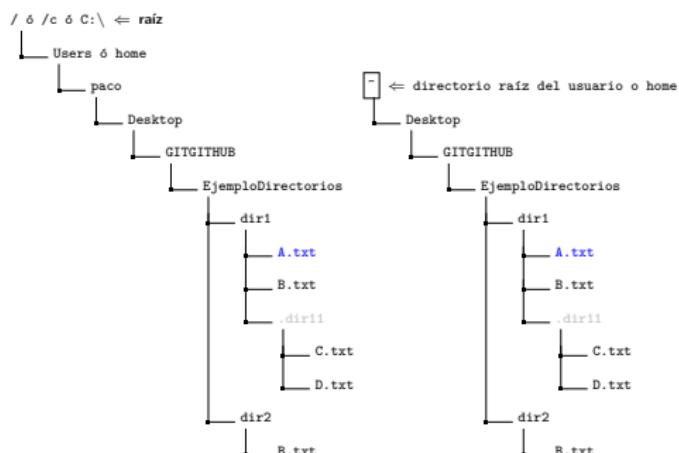
- (Windows CMD o PowerShell)
o bien (resto y Windows PowerShell)

- Directorio **actual**:

- (Windows CMD o PowerShell) o bien (resto y Windows PowerShell); también simplemente

- *Paths relativos* (al directorio de trabajo o a la raíz de usuario):

- El uso de algún directorio singular da lugar a un *path* relativo (al directorio de trabajo o a la raíz del usuario)



Ejemplos de *paths* relativos

Paths relativos hacia A.txt (Ejemplos para CLI CMD)

```
# desde dir2
..\\dir1\A.txt

# desde EjemploDirectorios
dir1\A.txt

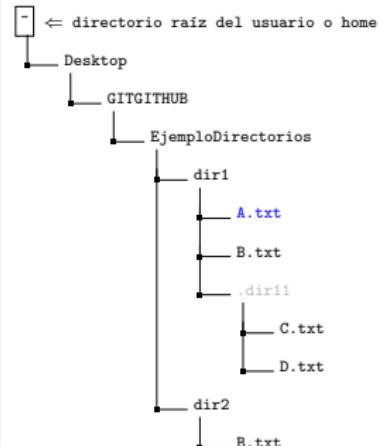
# o equivalentemente
.\dir1\A.txt

# y desde .dir11
..\A.txt

# también desde .dir11, pero dando un rodeo, llegando hasta EjemploDirectorios
..\..\dir1\A.txt

# desde ~ ó /Users/paco ó /home/paco ó c:\Users\paco ...
Desktop\GITGITHUB\EjemploDirectorios\dir1\A.txt

# path absoluto en el contexto del usuario, pero relativo al usuario (si cambio de usuario,
# entonces cambia el valor de ~):
# !!!NO EN CMD!!!
~/Desktop/GITGITHUB/EjemploDirectorios/dir1/A.txt
# En CMD
%userprofile%\Desktop\GITGITHUB\EjemploDirectorios\dir1\A.txt
```



Índice

2 Ventana o interfaz de línea de comandos (CLI)

- CLI (*Command-Line Interface*)
- Directorios y *paths*
- Algunos comandos básicos

Comandos básicos del shell

- **clear** (`cls` en Windows CMD):
 - Limpia la pantalla (*clear, clear screen*)
- **pwd** (no operativo en Windows CMD):
 - Imprime en pantalla el directorio actual de trabajo (*print working directory*)
- **cd <DIR>**:
 - Cambia de directorio (*change dir*)
- **mkdir <DIR>**:
 - Crea directorio (*make dir*)
- **ls [<DIR>]** (dir [<DIR>] en CMD):
 - Lista contenido de directorio (*list*)
- **vi <FICHERO> o vim:**
 - editor de textos (sólo en Linux, MacOS, CygWin, Git Bash de Windows... : *visual*)
- **echo <STRING>:**
 - Imprime <STRING> en pantalla (*echo*). Con `> o >>`: redirección a fichero

- **cat <FICHERO>** (type <FICHERO> en Windows):
 - Imprime en pantalla el contenido de fichero <FICHERO> (*catenate, type*)
- **rm <FICHERO>** (del <FICHERO> en Windows):
 - Borra fichero <FICHERO> (**¡CUIDADO! Irreversible**) (*remove, delete*)
- **cp <FICHERO_ORIGEN> <FICHERO_DESTINO>** (copy en Windwos):
 - Copia un fichero de un origen a un destino (*copy*). Comportamiento singular en función de <FICHERO_DESTINO>
- **mv <FICHERO_ORIGEN> <FICHERO_DESTINO>** (move en Windows):
 - Mueve en vez de copiar un fichero en otro (*move*); comportamiento singular en función del destino.
- **rm -rf <DIR>** (rmdir /s <DIR> en Windows):
 - Borra directorio y su contenido (**¡CUIDADO! Irreversible**) (*remove, remove-directory*)

Ejercicios sobre direccionamiento de ficheros y directorios

● Apertura de CLI

- Git Bash (Windows)
- CMD (Windows)
- Powershell (Windows)
- Terminal (Mac OS X, Linux)
- ...

● Recreación de EjemploDirectorios en local:

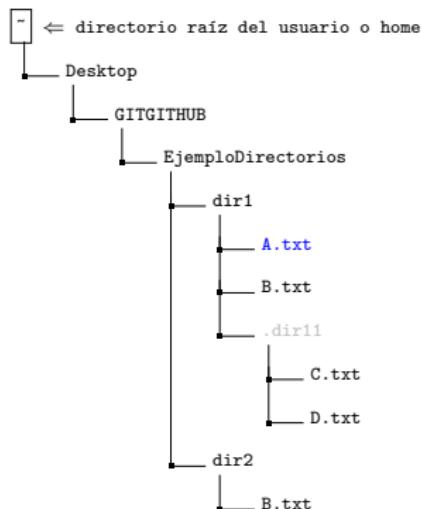
```
_____ Ejercicios de direccionamiento de ficheros y directorios _____
# Cambiamos a directorio de trabajo ~/Desktop
$ cd ~/Desktop
# En CMD de Windows:
> cd %userprofile%\Desktop

# Creamos bajo el directorio de trabajo la carpeta GITGITHUB
$ mkdir GITGITHUB

# Cambiamos de directorio de trabajo a la carpeta recién creada
$ cd GITGITHUB

# !!! PRIMERA CLONACIÓN DESDE GITHUB !!!
$ git clone https://github.com/GIT-GTDM-24-25/EjemploDirectorios.git

# ... Ejercicios
$
```



Ejercicios: condiciones

- Realíicense las operaciones que se indican a continuación
- Los únicos comandos necesarios serán:
 - cd <DIR>: cambiar a directorio <DIR>
 - ls [<DIR> | <FICHERO>]: listar el contenido de un directorio o listar un fichero
 - dir [<DIR> | <FICHERO>]: listar el contenido de un directorio o listar un fichero (CLI CMD)
 - cat <FICHERO>: imprimir fichero en pantalla
 - type <FICHERO>: imprimir fichero en pantalla (CLI CMD)
- Utilícese direccionamiento relativo, salvo que se indique explícitamente lo contrario
- **IMPORTANTE:** se sugiere que en todo momento se sea consciente de cuál es el directorio de trabajo (el *prompt* puede ayudar a ello).

Ejercicios

- Ejercicios

- ① Tras la ejecución de los comandos anteriores, ¿cuál es el directorio actual o directorio de trabajo?
- ② Comando para ir a directorio dir2, es decir, nuevo directorio actual o de trabajo, dir2
- ③ Listar contenido de directorio dir2
- ④ Listar contenido de directorio (oculto) .dir11 desde el directorio actual o de trabajo (supuestamente dir2)
- ⑤ Cambiar a directorio de trabajo GITGITHUB de manera absoluta
- ⑥ Listar el contenido del directorio de trabajo
- ⑦ Imprimir en pantalla el fichero C.txt
- ⑧ Cambiar a directorio de trabajo .dir11
- ⑨ Imprimir en pantalla el fichero B.txt: ambigüedad ⇒ imprimir ambos
- ⑩ Cambiar a directorio de trabajo dir2
- ⑪ Imprimir en pantalla el fichero D.txt

Índice

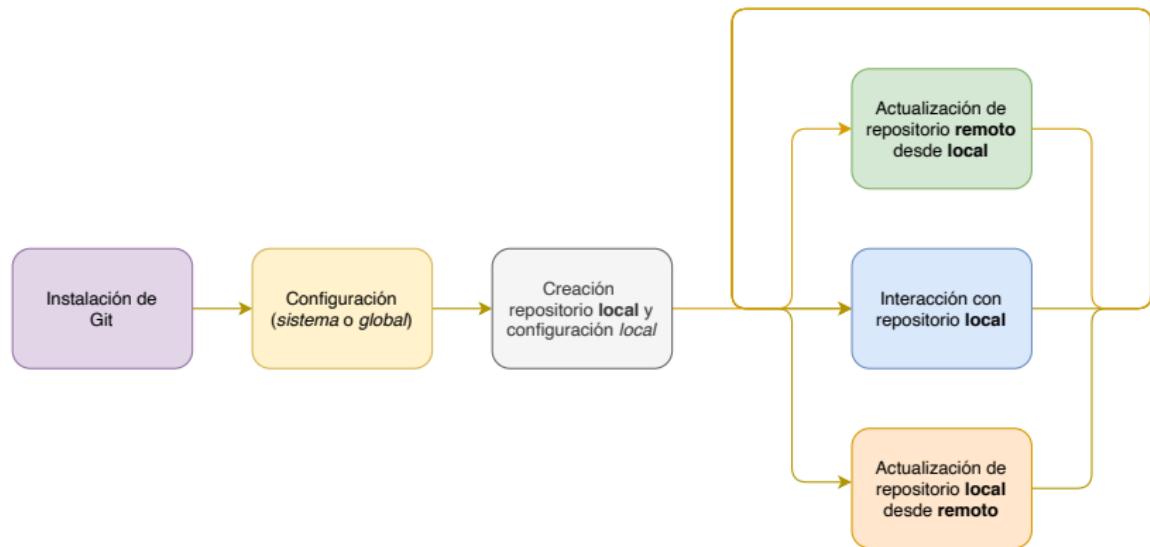
- 1 Introducción
- 2 Ventana o interfaz de línea de comandos (CLI)
- 3 Interacción básica con Git en un repositorio local
- 4 Comandos reset y checkout
- 5 Ramas
- 6 Interacción con repositorios remotos
- 7 Aspectos básicos de GitHub

Índice

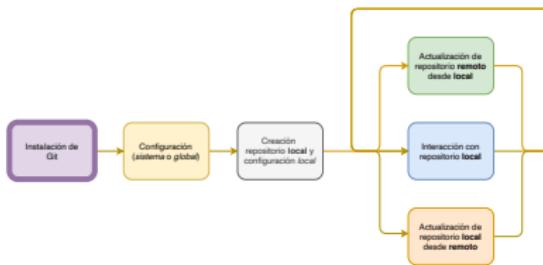
3 Interacción básica con Git en un repositorio local

- Uso, instalación y configuración
 - El repositorio local
 - Esquema add-commit
 - Detalles de la adición al *index*
 - *Commit* al repositorio
 - *Commits*, ramas, punteros y listados
 - Diferencias entre ficheros: diff y difftool
 - Tags o etiquetas
 - Sinopsis

Procedimiento de uso de Git



Instalación



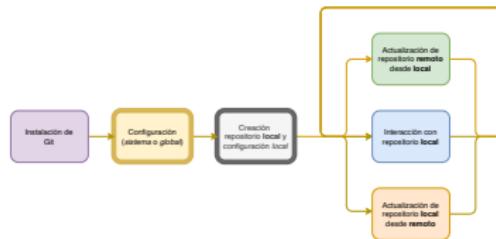
- Descarga desde URL: <https://git-scm.com/>
- Documentación: <https://git-scm.com/doc>. **¡¡¡Síganse las instrucciones!!!**
- Uso de consola CLI (*Command Line Interface*) en vez de GUI (*Graphical User Interface*).
- Git **requiere** un editor de textos (para complementar ciertos comandos)
 - La instalación trae vi(m)
 - Por sencillez, sugerimos **Visual Studio Code** (tras instalación, ejecutable: code)
- Solo en Windows, tras la instalación, se añade un nuevo CLI al sistema:
 - Git Bash: CLI de tipo terminal de UNIX/Linux con un *shell* Bash.
- Comandos en línea de comandos
 - Formato: \$ git <COMANDO> ...
 - El *prompt* (\$ en los ejemplos) **debe** omitirse

Configuración

- Configuración básica: ábrase un CLI cualquiera (terminal, CMD, PowerShell o Git Bash)

- Niveles de configuración:

- global: configuración se aplicará a todos los repositorios del usuario
- local: configuración se aplicará solo al repositorio actual (requiere de uno ya creado)
- system: configuración se aplicará a todos los repositorios de todos los usuarios



Ejemplo de configuración para nivel "global".

```

// Indicación de usuario, email
$ git config --global user.name "Paco Martinez"
$ git config --global user.email "fjmartin@dcom.upv.es"

// Indicación de editor (Visual Studio Code -code- en vez de vi/vim)
$ git config --global core.editor "code --wait"

// Indicación de herramienta difftool: Visual Studio code
// IMPORTANTE: emplense comillas simples ', no dobles " en Linux/Mac o Windows Git Bash
//             emplense comillas dobles " en Windows CMD
$ git config --global diff.tool vscode
$ git config --global difftool.vscode.cmd 'code --wait --diff $LOCAL $REMOTE'

// Indicación de herramienta mergetool: Visual Studio code
// IMPORTANTE: emplense comillas simples ', no dobles " en Linux/Mac o Windows Git Bash
//             emplense comillas dobles " en Windows CMD
$ git config --global merge.tool vscode
$ git config --global mergetool.vscode.cmd 'code --wait --merge $REMOTE $LOCAL $BASE $MERGED'
  
```



Configuración: prioridad, verificación y ajustes posteriores

- Puede haber especificaciones para distintos niveles. Prioridades:
 - Especificación --system sobreescrita por --global
 - Especificación --global sobreescrita por --local
- Verificación:

Verificación

```
// Listado de configuración (mostrando fichero de configuración si --show-origin)
$ git config [--system | --global | --local] --list [--show-origin]
```

- Ajustes posteriores:

Ajustes posteriores

```
// Vuelta a valores por defecto de un <ITEM> (user.email, user.name, ...):
$ git config --global --unset <ITEM>
```

```
// Edición de la configuración (no aconsejable por sintaxis)
$ git config --global --edit
```

Índice

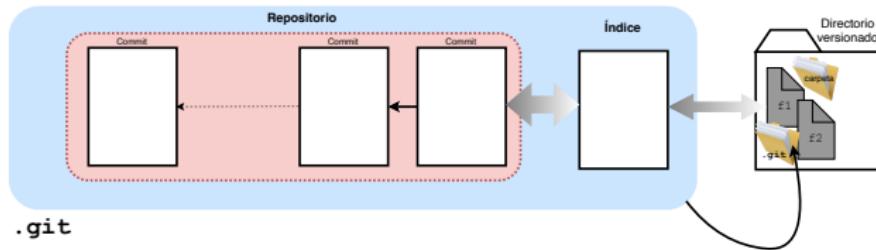
3 Interacción básica con Git en un repositorio local

- Uso, instalación y configuración
- **El repositorio local**
- Esquema add-commit
- Detalles de la adición al *index*
- *Commit* al repositorio
- *Commits*, ramas, punteros y listados
- Diferencias entre ficheros: diff y difftool
- Tags o etiquetas
- Sinopsis

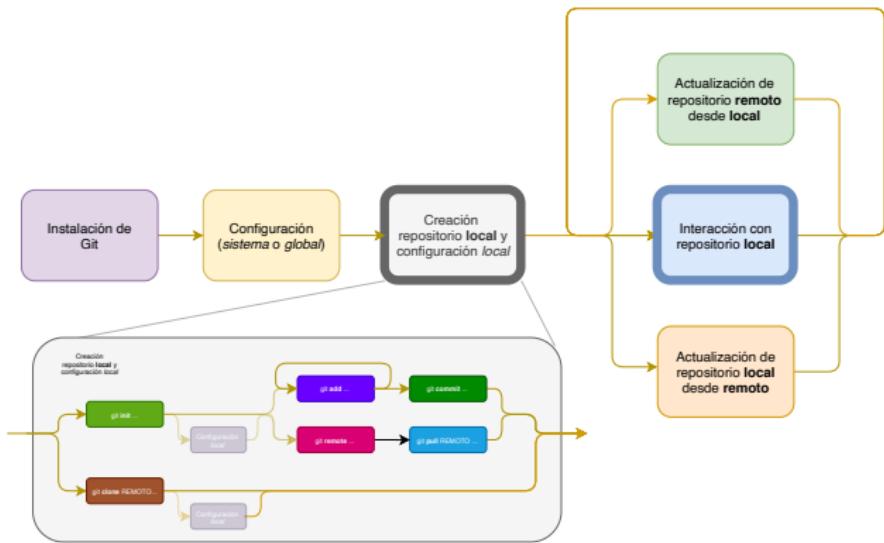
Esquema de almacenamiento

- *Directorio versionado*

- Conjunto de ficheros y directorios (*a versionar*)
- Infraestructura de versionado: directorio *oculto* `.git`
 - Índice o zona de *preparación* (*index* o *staging area*)
 - Repositorio. Concepto de *commit*: *instantánea del directorio versionado*



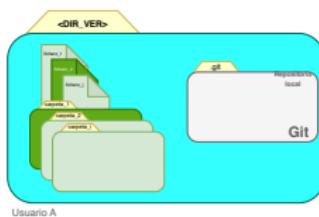
Alternativas de creación



Casuística

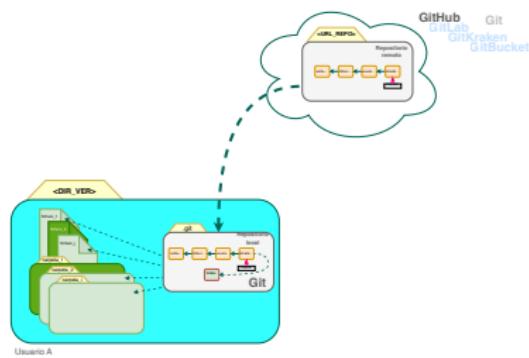
a) Nosotros **originamos** el repositorio

```
$ git init <DIR_VER>
o bien
$ mkdir <DIR_VER>
$ cd <DIR_VER>
$ git init
```



b) El repositorio está ya creado **remotamente**

```
$ git clone <URL_REPO> [ <DIR_VER> ]
```



- Si <DIR_VER> no existe, lo crea
- <DIR_VER> no tiene porqué estar vacío
- Crea carpeta .git con infraestructura interna para control de versiones
- Eliminación de control de versiones: borrado de carpeta .git
- Si no se especifica <DIR_VER>:
 - <DIR_VER> ⇌ <NOMBRE_REPOSITORIO_DE_URL_REPO>
- Si <DIR_VER> existe, debe estar vacío

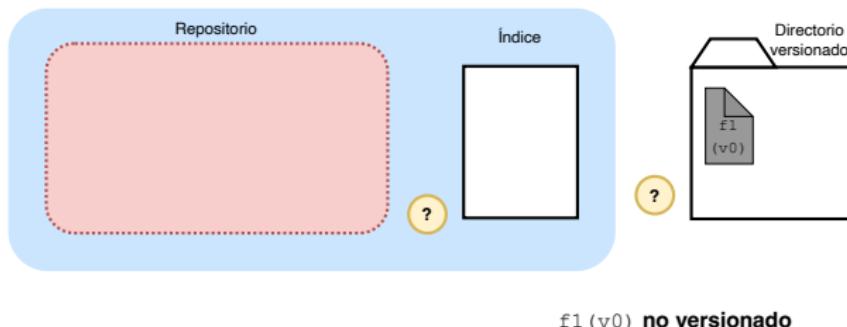
Índice

3 Interacción básica con Git en un repositorio local

- Uso, instalación y configuración
- El repositorio local
- **Esquema add-commit**
- Detalles de la adición al *index*
- *Commit* al repositorio
- *Commits*, ramas, punteros y listados
- Diferencias entre ficheros: *diff* y *difftool*
- *Tags* o etiquetas
- Sinopsis

Comandos add y commit

- Ejemplo:



?

Sin versionar

?

Sin modificaciones

A

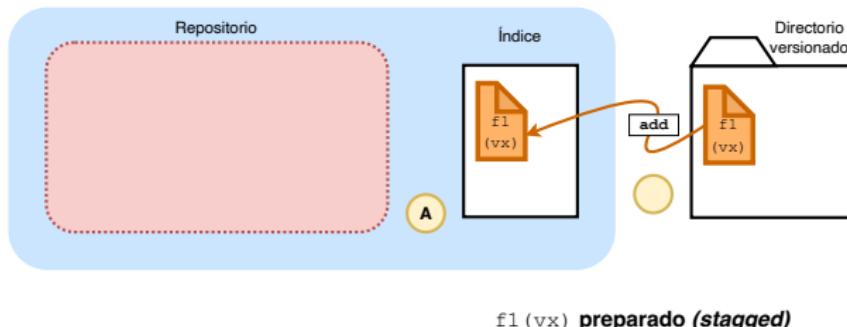
Añadido al index, pero
todavia no al repositorio

M

Con modificaciones

Comandos add y commit

- Ejemplo:



?

Sin versionar

?

Sin modificaciones

A

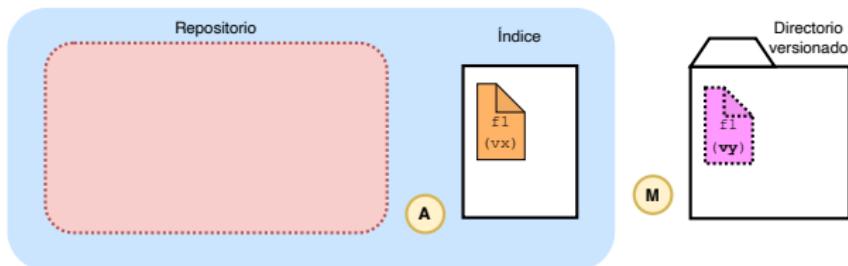
Añadido al index, pero
todavia no al repositorio

M

Con modificaciones

Comandos add y commit

- Ejemplo:



? Sin versionar

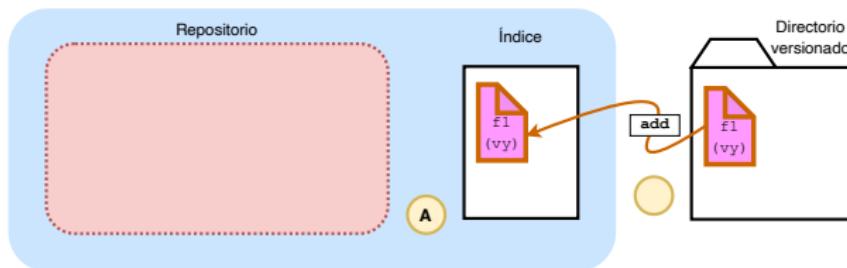
○ Sin modificaciones

A Añadido al index, pero todavía no al repositorio

M Con modificaciones

Comandos add y commit

- Ejemplo:

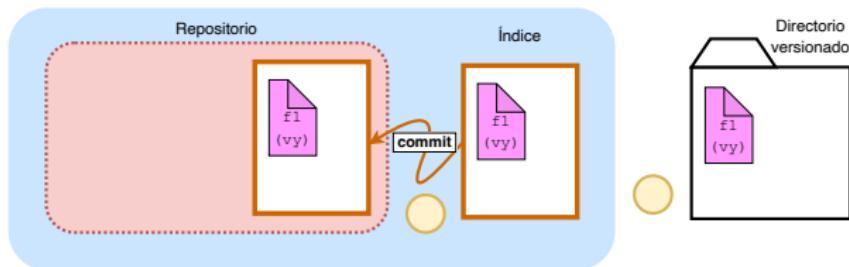


f1 (vy) **preparado (staged)**

- ? Sin versionar
- Sin modificaciones
- Aañadido al index, pero todavía no al repositorio
- M Con modificaciones

Comandos add y commit

- Ejemplo:

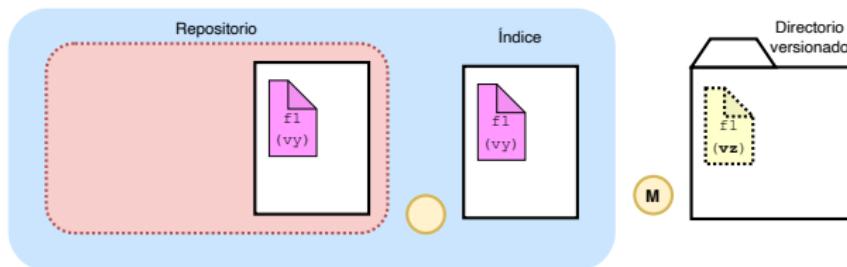


f1 (vy) consignado ()

- ? Sin versionar
- Sin modificaciones
- A** Añadido al index, pero todavía no al repositorio
- M** Con modificaciones

Comandos add y commit

- Ejemplo:



f1 (vz) **modificado** (en el dir. de trabajo respecto del índice)

?

Sin versionar

?

Sin modificaciones

A

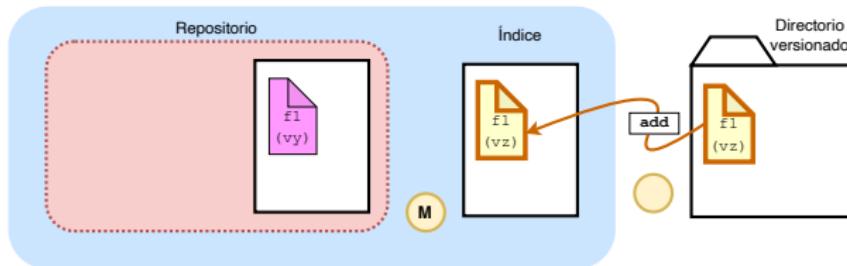
Añadido al index, pero todavía no al repositorio

M

Con modificaciones

Comandos add y commit

- Ejemplo:



f1 (vz) **modificado** (en el índice respecto del último commit del repo.)

f1 (vz) **preparado (staged)**

?

Sin versionar

○

Sin modificaciones

A

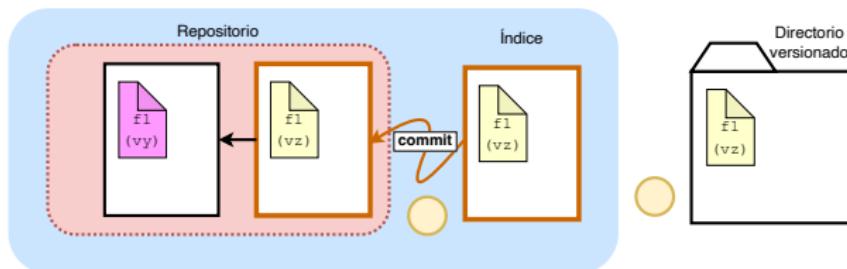
Añadido al index, pero
todavia no al repositorio

M

Con modificaciones

Comandos add y commit

- Ejemplo:



?

Sin versionar

?

Sin modificaciones

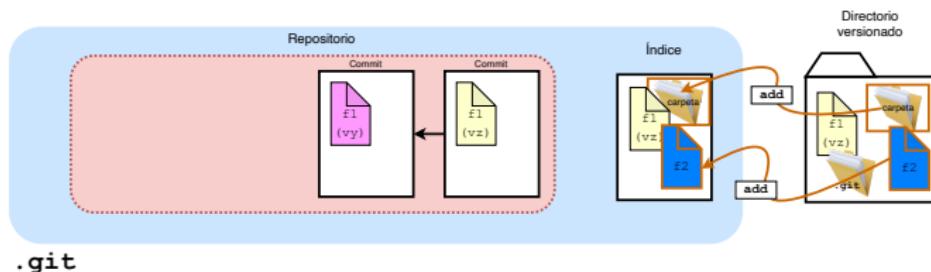
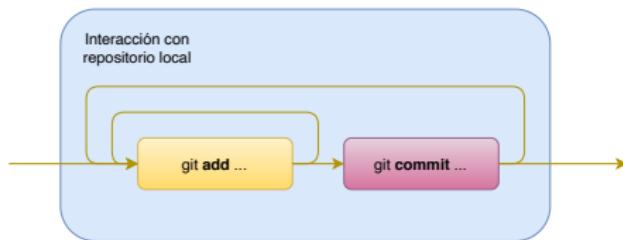
A

Añadido al index, pero todavía no al repositorio

M

Con modificaciones

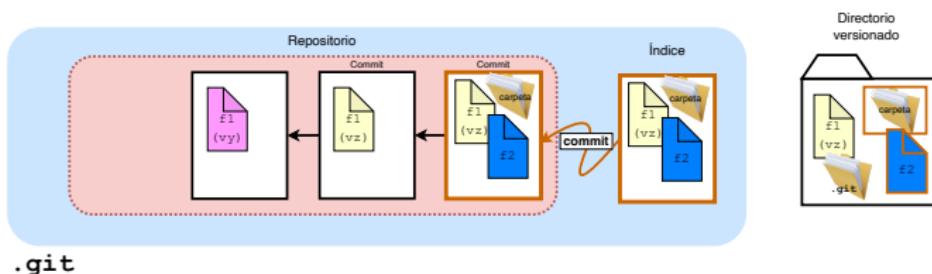
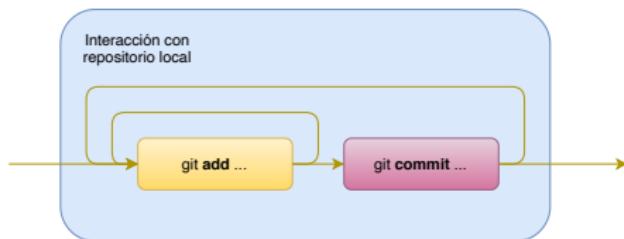
Procedimiento habitual y repetitivo



.git

```
$ git add carpeta  
$ git add f2  
  
# o bien  
$ git add carpeta f2
```

Procedimiento habitual y repetitivo



.git

```
$ git commit -m "Comentario"
```

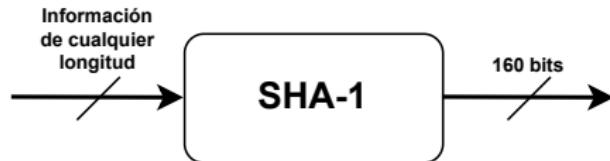
Índice

3 Interacción básica con Git en un repositorio local

- Uso, instalación y configuración
- El repositorio local
- Esquema add-commit
- Detalles de la adición al *index*
- Commit al repositorio
- Commits, ramas, punteros y listados
- Diferencias entre ficheros: diff y difftool
- Tags o etiquetas
- Sinopsis

Función Hash o resumen

- Síntesis de una secuencia de bits
- Numerosos algoritmos de *hash*
 - SHA-1: algoritmo que a partir de cualquier secuencia de bits, la *resume* en:
 - 160 bits (20 bytes o 40 dígitos hexadecimales)



- ...

Hash de un fichero en Git (*git-hash*)

- Git calcula el *hash* de un fichero *basándose* en SHA-1 con ciertas modificaciones. Lo denominaremos *git-hash*

```
$ echo aldkfj > a.txt
$ cat a.txt
aldkfj

% git-hash del fichero a.txt
$ git hash-object a.txt
40b6860eadfb1842efb9e58928481b5b12f13a98
$ echo aldkfk > a.txt
$ cat a.txt
aldkfk

% git-hash del fichero a.txt
$ git hash-object a.txt
d362ae2ab10f60af4ee44ad89d91666b4a207af3
```

- Comprobación de modificación de ficheros (e incluso **identificación**):
 - NO se comparan bit a bit
 - Comparación de sus *hashes*: es más eficiente en términos generales
- Listado y *hashes* de ficheros en *index*:

```
$ git add a.txt
$ git ls-files -s
100644 d362ae2ab10f60af4ee44ad89d91666b4a207af3 0 a.txt
```

- Para comprobar si un fichero es distinto en el *index* respecto al *directorio versionado*, se comparan sus *hashes* respectivos.

Adición de múltiples ficheros al *index*: parámetros para add

- Elección de ficheros a actualizar en el *index* con el comando `git add`:

Comando	Fich. Nuevos	Fich. Modificados	Fich. Borrados	¿Qué fich.?
<code>git add <PATHSPEC></code>	Sí	Sí	Sí	<PATHSPEC>
<code>git add --ignore-removal <PATHSPEC></code>	Sí	Sí	No	<PATHSPEC>
<code>git add -A</code>	Sí	Sí	Sí	<DV>
<code>git add -u [<PATHSPEC>]</code>	No	Sí	Sí	<PATHSPEC> <DV>

- <DV>: *directorio versionado*
- <PATHSPEC>: selección de ficheros o directorios y sus ficheros a considerar, dentro del *directorio versionado*
 - Sucesión de ficheros o directorios y sus ficheros o expresiones regulares
- Git no añade **directorios vacíos** con el comando add.
- Operación *dryrun*: indica lo que se va a añadir al *index*, pero no lo hace (especie de *simulacro* de add)

Prueba de add (realmente no se ejecuta)

```
$ git add -n <resto_de_opciones>
```

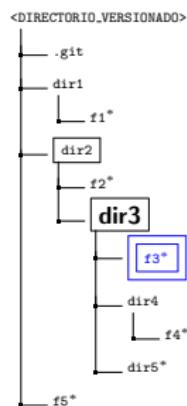
- Documentación:

<https://git-scm.com/docs/gitglossary#Documentation/gitglossary.txt-aiddefpathspecapathspec>

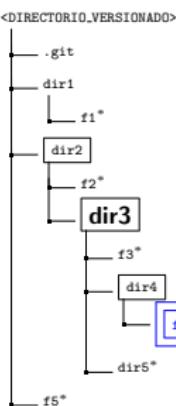
Ejemplos del comando add

- Directorio de trabajo: **dir3** (por ejemplo: \$ cd dir2/dir3 desde <DIRECTORIO_VERSIONADO>)
- **[fn*]**: fichero nuevo, actualizado (o borrado) a añadir al *index*
- Si un fichero se va a añadir al *index*, automáticamente se añaden todos los directorios necesarios si no están añadidos ya, denotados con **[dir]**.
- Ejemplo: <https://github.com/GIT-GTDM-24-25/EjemplosAdd.git>.
 - Descomprímase el fichero .zip
 - Elimíñese el fichero .zip
 - Reubíquese el contenido a la raíz del directorio versionado

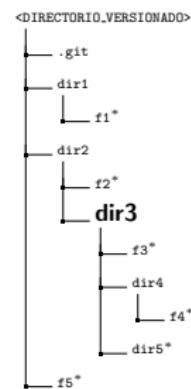
\$ git add f3



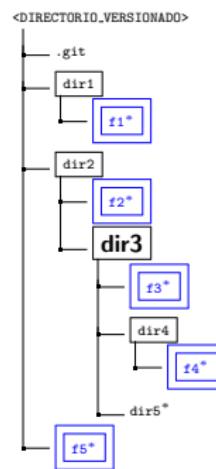
\$ git add dir4



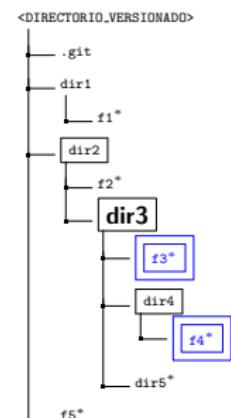
\$ git add dir5



\$ git add -A

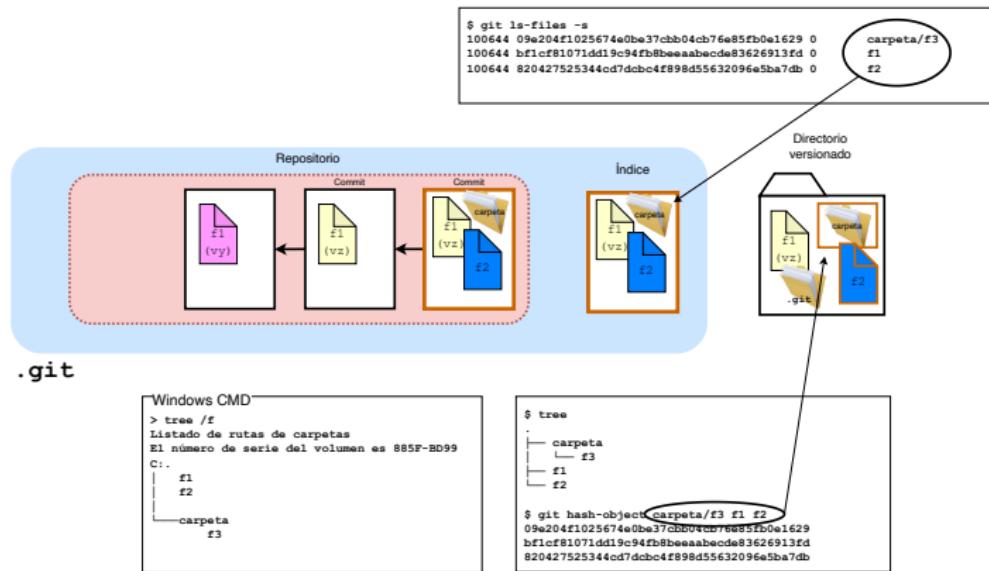


\$ git add .



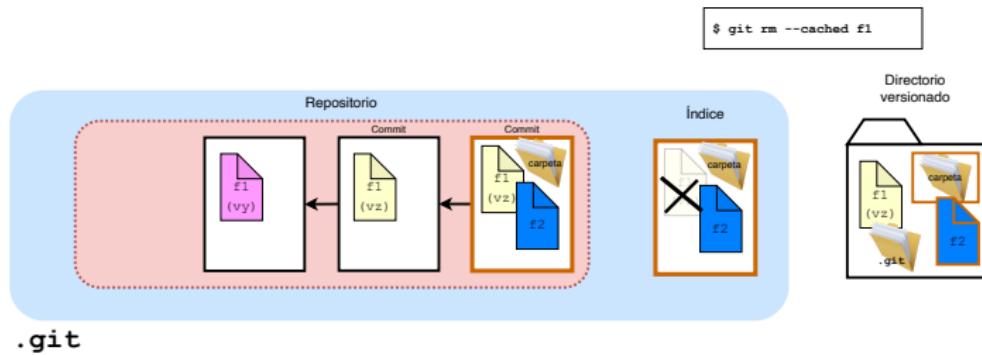
Listado de ficheros del *index*

- Ejemplo:
<https://github.com/GIT-GTDM-24-25/OperacionesIndex.git>
- Listado de los ficheros en el *index* (sensible al directorio de trabajo en el que se está).



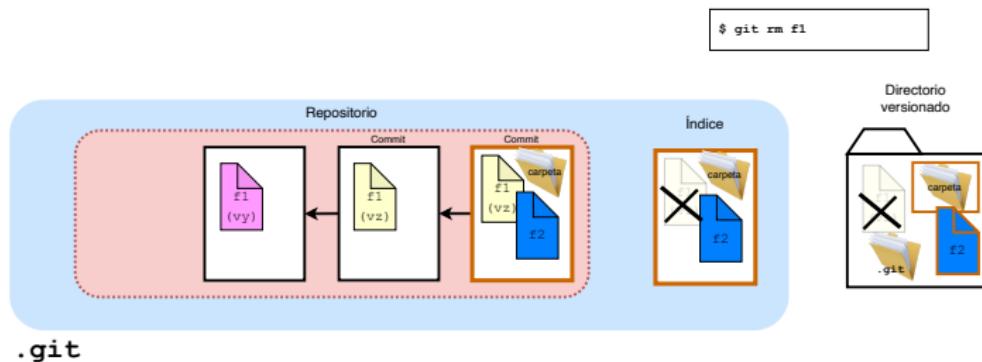
Borrado de ficheros del *index*

- Eliminación de fichero SOLO en el *index* (no se borra del directorio versionado y queda en estado *sin versionar*: ?):



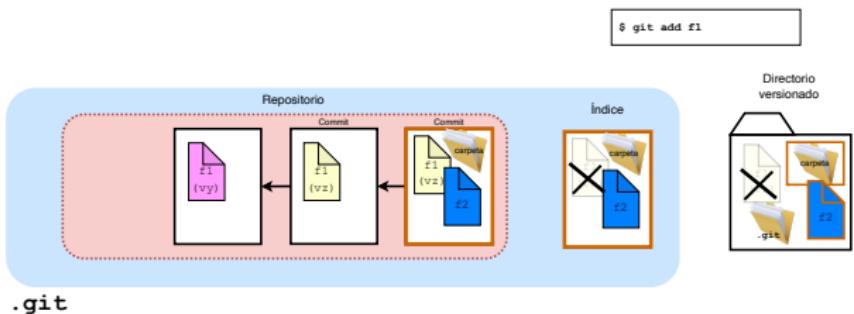
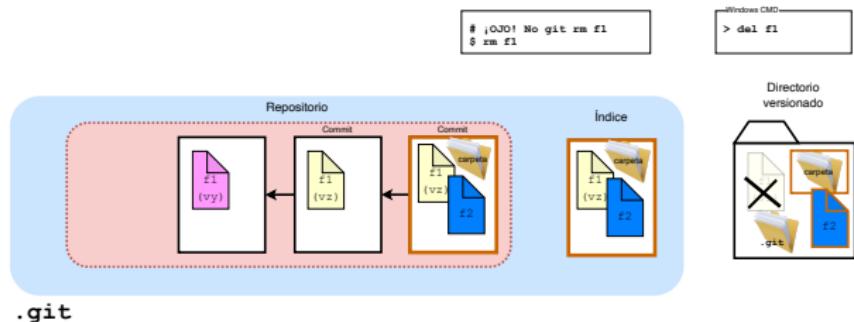
Borrado de ficheros del *index* y del directorio

- Borrado de un fichero tanto en el directorio versionado como en el *index*



Borrado de ficheros del *index* y del directorio (alternativa)

- Borrado de un fichero en directorio versionado y después *reflejo* en el *index*



Mover o renombrar ficheros

```
$ git mv fichero_original fichero_final  
$ git status -s  
R fichero_original -> fichero_final  
  
# Opcionalmente: commit posterior  
$ git commit -m "fichero movido"  
...  
$ git status -s
```

Equivalencia

```
$ mv fichero_original fichero_final  
$ git add fichero_original fichero_final  
  
# Opcionalmente: commit posterior  
$ git commit -m "fichero movido"  
...  
$ git status -s
```

Fichero(s) .gitignore

- Su contenido son patrones de ficheros que serán ignorados en el comando add (salvo si se indica la opción -f)
- La sintaxis puede contemplar patrones complejos de exclusión de ficheros o directorios
- Ejemplo:

Contenido del fichero .gitignore

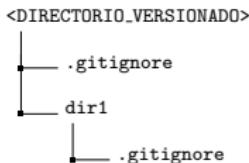
```
pepe.txt
!juan.txt
dir1
output/*.log
**/lib/*.a
obj/**/ejec.exe
```

- Patrones, ejemplos y propuestas:

- <https://git-scm.com/docs/gitignore>
- <https://www.atlassian.com/git/tutorials/saving-changes/gitignore>
- <https://github.com/github/gitignore>

Fichero(s) .gitignore: anidamiento, precedencia y eliminación de reglas

- Puede haber diferentes `.gitignore` en la estructura de directorios del *directorio versionado*. Se heredan reglas y si hay contradicción, prevalece la última.



- Si un fichero ha sido versionado en el pasado, se sigue versionando aunque `.gitignore` indique ahora lo contrario. Para cambiar esta actitud:

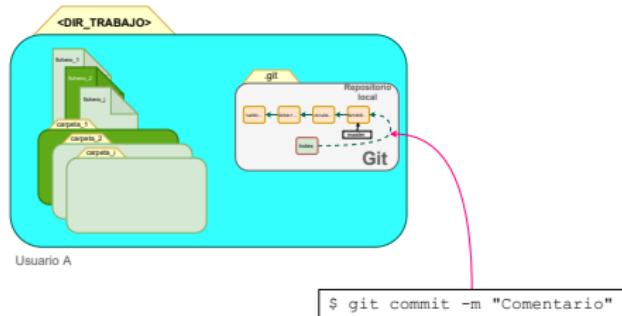
```
exclusión explícita en .gitignore y eliminación del index.  
// Control de <FICHERO> en .gitignore  
// Acceso al .gitignore oportuno y control de exclusión en el mismo  
  
// Eliminación de <FICHERO> en el index  
$ git rm --cached <FICHERO>
```

Índice

3 Interacción básica con Git en un repositorio local

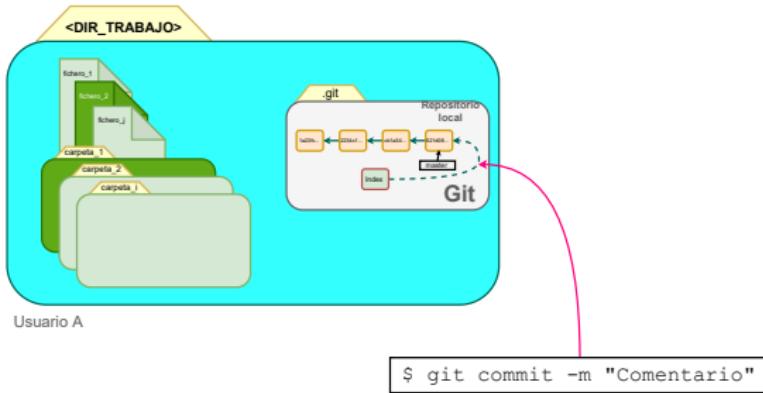
- Uso, instalación y configuración
- El repositorio local
- Esquema add-commit
- Detalles de la adición al *index*
- **Commit al repositorio**
- *Commits*, ramas, punteros y listados
- Diferencias entre ficheros: diff y difftool
- Tags o etiquetas
- Sinopsis

Guardar el contenido del *index* al repositorio **local**: commit



- *commit*: múltiples acepciones aunque en el fondo denotan lo mismo
 - Acción de guardar todo el contenido del *index* en una estructura denominada, a su vez, *commit* en el repositorio
 - Conjunto de ficheros guardados con la acción *commit*
 - Estado del *directorio versionado* en cierto instante de tiempo (contenido del *index*) a guardar en el repositorio
- *Hash de commit*
- Los *commits* se van *acumulando* o *apilando* sucesivamente
- *Rama*: secuencia lineal de *commits* consecutivos
- *Puntero de rama*: puntero al último *commit* de una rama (`master`)
- Cada adición de nuevo *commit*:
 - *Acumulación* del nuevo *commit* en el repositorio
 - Actualización del puntero de rama al nuevo (último) *commit*

Guardar el contenido del *index* al repositorio **local**: commit (cont.)



- En principio, requiere un comentario: `-m "Comentario"`
 - Si no se especifica (`$ git commit`), salta un editor de textos para proporcionar comentario asociado a *commit*.
 - Elección de editor. Recuérdese de sección de configuración:

Elección de editor (Visual Studio Code) en configuración de git

```
$ git config --global core.editor "code --wait"
```

Estructura del comentario -m

- Comando típico

```
$ git commit -m <COMENTARIO>
```

- Si no se especifica opción `-m` salta el editor de textos por defecto para introducirlo obligatoriamente.
- Partes del comentario:
 - Descripción abreviada o resumen: **obligatorio**
 - Descripción detallada: *opcional*

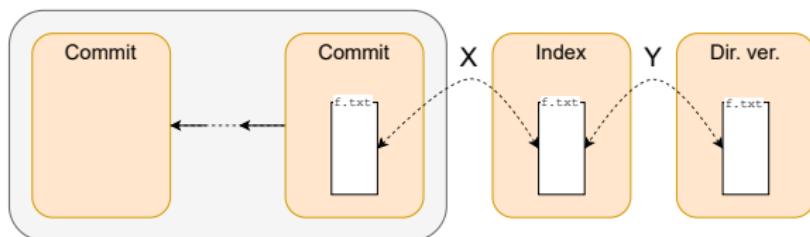
```
# Comentario resumido:  
$ git commit -m "Resumen"  
...  
# Comentario resumido y detallado  
$ git commit -m "Resumen" -m "Descripción más detallada"  
$ ...  
# Comentario resumido y detallado: alternativa  
$ git commit -m "Resumen"  
  
Descripción más detallada"  
$
```

- Si el comentario no incluye espacios en blanco o saltos de línea, se pueden omitir las comillas.
- Se debería ser disciplinado con la *filosofía* y estructura de los comentarios, especialmente cuando se trabaja en equipo

Estado relativo de los ficheros

- Descripción textual del estado, comando: `$ git status`
- Abreviado: `$ git status -s`

Repositorio



```
$ git status -s
XY f.txt
```

En X o Y:

	espacio en blanco, sin modificación
A	añadido (nuevo)
M	modificado
D	borrado (en X)
R	renombrado
U	unmerged, sin fusionar todavía
?	sin seguimiento o sin versionar
...	...

Algunas combinaciones en *index* y *directorio versionado*:

index resp. commit X	dir. ver. resp. index Y	Significado
?	?	Sin seguimiento (no versionado)
A	?	Añadido por primera vez al index, la versión en el directorio versionado actualizada en el index
A	M	Añadido por primera vez al index, la versión en el directorio versionado modificada respecto al index
	M	Versión del index actualizada en el repositorio; versión en directorio versionado modificada en el index
	M	Versión del index actualizada en el repositorio; versión en directorio versionado actualizada en el index
M	M	Versión del index modificada en el repositorio; versión en directorio versionado modificada en el index

Estado del repositorio: git status vs. git status -s

\$ git status	Comentario	\$ git status -s
On branch <RAMA_DE_TRABAJO_ACTUAL>	Siempre se muestra la rama de trabajo	
No commits yet	Situación inicial	
nothing to commit (create/copy files and use "git add" to track)		
Untracked files: (use "git add <file>..." to include in what will be committed) <FICHERO_SIN_SEGUIMIENTO> ... <FICHERO_SIN_SEGUIMIENTO>	Hay ficheros que no están ni versionados ni ignorados	?? <FICHERO_SIN_SEGUIMIENTO> ... ?? <FICHERO_SIN_SEGUIMIENTO>
Changes to be committed: (use "git rm --cached <file>..." to unstage) new file: <FICHERO_PRIMERA_VEZ_EN_INDEX> modified: <FICHERO_YA_ESTABA_EN_INDEX>	Hay ficheros que se aparecerán en el próximo commit, por ser nuevos (A), o por estar modificados (M)	A_ <FICHERO_PRIMERA_VEZ_EN_INDEX> M_ <FICHERO_YA_ESTABA_EN_INDEX>
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: <FICHERO_VERSIONADO_MODIF._EN_DV> no changes added to commit (use "git add" and/or "git commit -a")	Modificación de fichero en directorio de trabajo no guardado en index	_M <FICHERO_VERSIONADO_MODIF._EN_DV>
Ignored files: (use "git add -f <file>..." to include in what will be committed) <FICHERO_IGNORADO>	Si se añade opción git status --ignored, se indican los ficheros ignorados por .gitignore	!!<FICHERO_IGNORADO>
nothing to commit, working tree clean	Coinciden versiones en directorio versionado, index y último commit	

Listados y logs del repositorio local

- Listado de los ficheros en el *index* (sensible al directorio de trabajo en el que se está).

```
$ git ls-files -s
# listado por HEAD (lo instantaneo del directorio versionado en cierto instante de tiempo)
100644 c122d8a17fb51:907372b5ca42f113032598cf7 0 a.txt
100644 d4c3b6d011a3f6ed1ff9e0d5ac5b589981370da 0 dir1/b.txt
```

- Listado de los ficheros de cierto *commit* (sensible al directorio de trabajo en el que se está).

```
* lista el commit (lo instantaneo del directorio versionado en cierto instante de tiempo)
# apuntado por HEAD (ultimo commit)
$ git ls-tree -r HEAD
100644 blob 25fd12f7chfcf99c214a9d44de20ad41af09c06 a.txt
100644 blob d4c3b6d011a3f6ed1ff9e0d5ac5b589981370da dir1/b.txt
```

- Listado de todos los *commits realizados* pertenecientes a la **rama actual** de trabajo

```
# log detallado
$ git log

# log detallado solo de los ultimos 3 commits
$ git log -3

# log detallado de todos los commits en los el patron <PATRON> cambia la cantidad de veces que aparece
$ git log --grep <PATRON>

# log detallado de todos los commits en los que aparece el patron <PATRON> como parte del comentario
$ git log --grep <PATRON>

# log detallado de todos los commits que modificaron <TIEMPO>
$ git log <TIEMPO>

# log abreviado con una linea por commit
$ git log --oneline

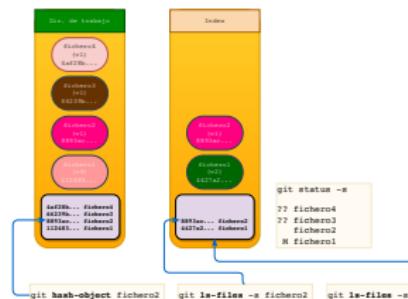
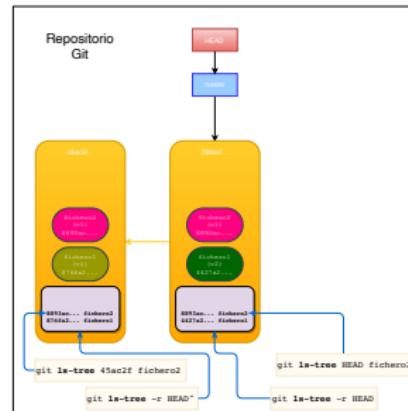
# log abreviado con una linea por commit añadiendo algunos detalles
$ git log --oneline --decorate=full
...
```

- Idem pero incluyendo los commits de **todas** las ramas (*--all*) y con descripción gráfica (*--graph*):

```
# log detallado
$ git log --oneline --all --graph ...
```

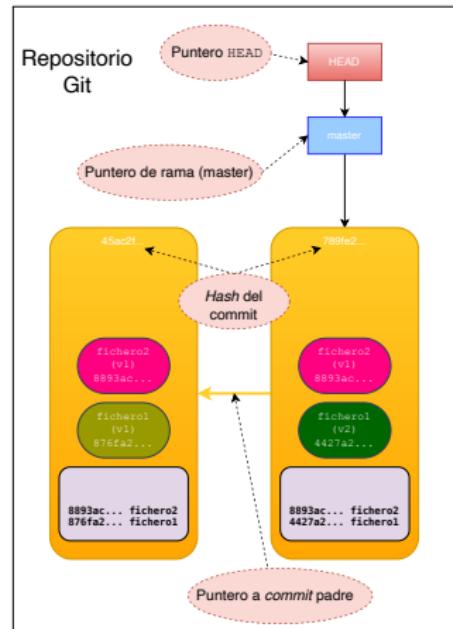
- Listado de **todos** los *commits realizados* localmente aunque pertenezcan a ramas ya borradas o inexistentes.

```
$ git reflog show --all % log de todos los commits realizados
```

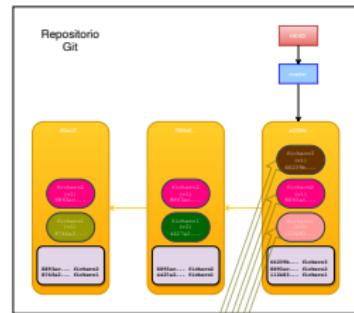
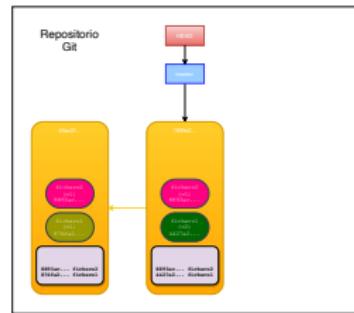
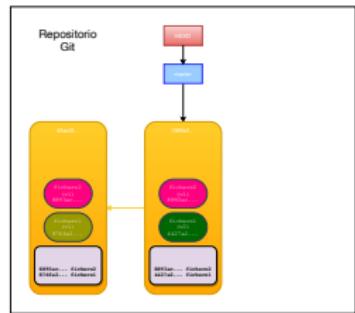


Commits y punteros en el repositorio local

- Cada *commit* tiene un puntero que **apunta** al *commit* anterior (*su padre*) formando todos ellos una *secuencia de commits o rama*: **¡¡¡cuidado con sentido de la flecha!!!**
- Cada *commit* puede identificarse de forma única mediante el *hash* de su contenido (y más información): 40 dígitos hexadecimales (los primeros con tal de que sean distintos al resto)
- El **último commit** de la secuencia o rama (el más reciente) es apuntado por un puntero denominado **puntero de rama**
- El último *commit* guardado en el repositorio apuntado por el puntero **HEAD**
 - Misión de HEAD: apuntar al *commit* (o puntero de rama que apunta a dicho *commit*) que será el *padre* del siguiente *commit* a guardar
 - Suele ser solidario con el *puntero de rama*
 - Un nuevo *commit* implica actualización de tanto el *puntero de rama* como del puntero HEAD



Modelo de funcionamiento add-commit: ejemplo



git status -s

A ficherol	ficherol
A ficherol2	ficherol2
A ficherol3	ficherol3
M ficherol	

git add ficherol ficherol2

// @ bien

git add ficherol

git add ficherol2

// @ bien...

git status -s

A ficherol	ficherol
A ficherol2	ficherol2
A ficherol3	ficherol3
M ficherol	

git add ficherol ficherol2

// @ bien...

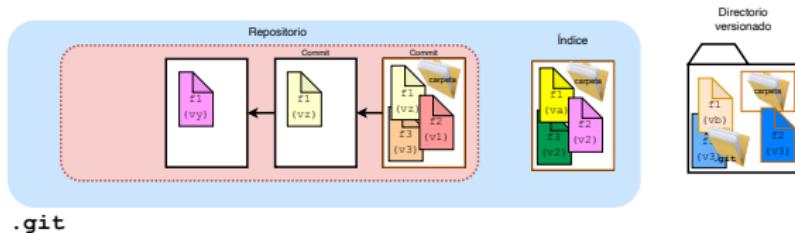
git status -s

A ficherol	ficherol
A ficherol2	ficherol2
A ficherol3	ficherol3
M ficherol	

git commit -m "Mensaje"

Deshacer selectivamente operaciones commit y add: git restore

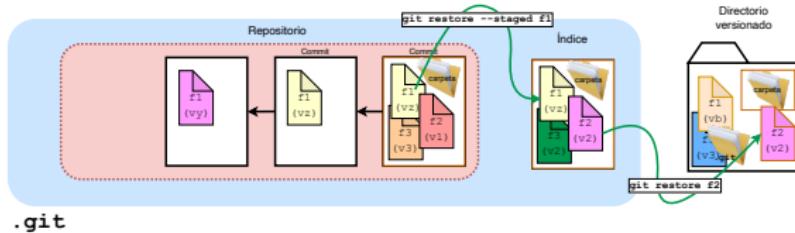
- Situación de partida



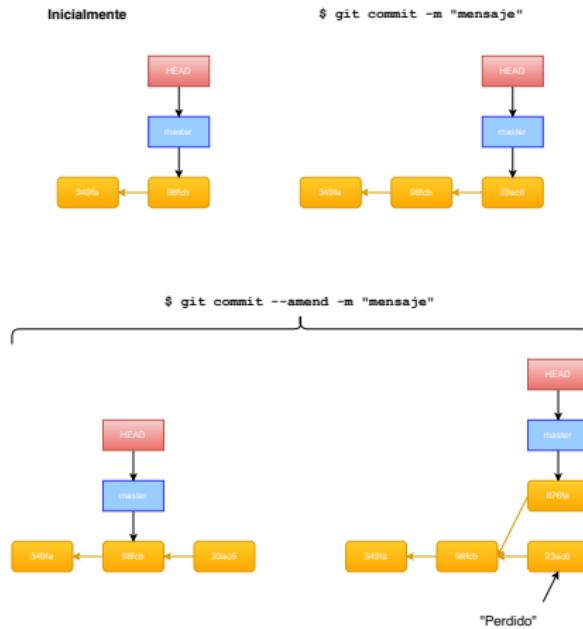
- Restauración selectiva de ficheros:

```
Restauración
# restauración de <FICHERO> de ultimo commit a index
$ git restore --staged <FICHERO>

# restauración de <FICHERO> de index a directorio versionado
$ git restore <FICHERO>
```



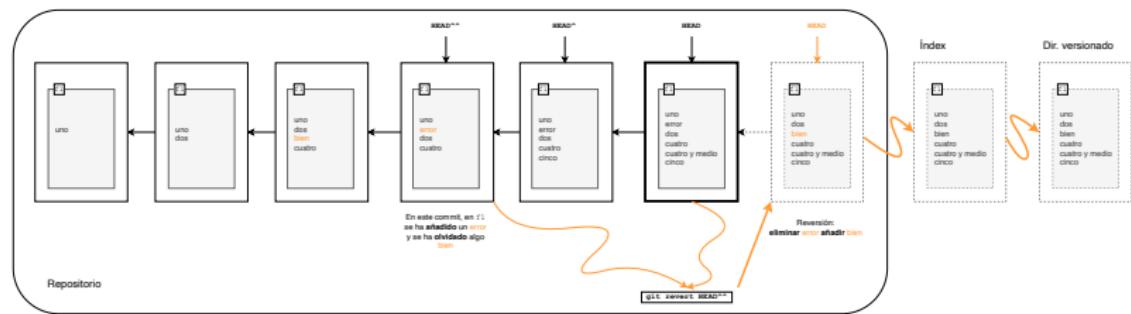
Rectificar un commit: opción --amend



- Un commit puede considerarse *inadecuado* por no tener los ficheros el contenido oportuno o por no tener el mensaje asociado correcto
 - Tras el commit *inadecuado* se puede modificar el index y preparar un nuevo commit
 - El mensaje se puede modificar al invocar de nuevo al comando commit, a no ser que se indique la opción `--no-edit`
- El commit *inadecuado* no se borra realmente: aunque se pierde, todavía puede ser accesible buscándolo con el comando `reflog`
- Operación *peligrosa* y potencialmente *conflictiva* si se comparte el repositorio.

Revertir un commit: git revert

- Revertir la acción realizada por un *commit*
 - Revertir cierto *commit*: aplicar al último *commit* (apuntado por HEAD) los cambios *diferenciales* del *commit* a revertir respecto de su *commit* anterior.
- Ejemplo: <https://github.com/GIT-GTDM-24-25/EjemploRevert.git>



- Se genera un nuevo *commit* que se añade al repositorio como último *commit* y se recarga el *index* y el *directorio versionado*.
- Si el *commit* a revertir es el actual: trivial. Si no:
 - Se puede crear un conflicto si los patrones a añadir/borrar no pueden ser encajados en el último *commit*: resolución *manual*.
- Posibilidad de cancelar: \$ git revert --abort

Índice

3 Interacción básica con Git en un repositorio local

- Uso, instalación y configuración
- El repositorio local
- Esquema add-commit
- Detalles de la adición al *index*
- *Commit* al repositorio
- **Commits, ramas, punteros y listados**
- Diferencias entre ficheros: diff y difftool
- Tags o etiquetas
- Sinopsis

Identificadores de *commits*, ramas y punteros

- **Commit:**

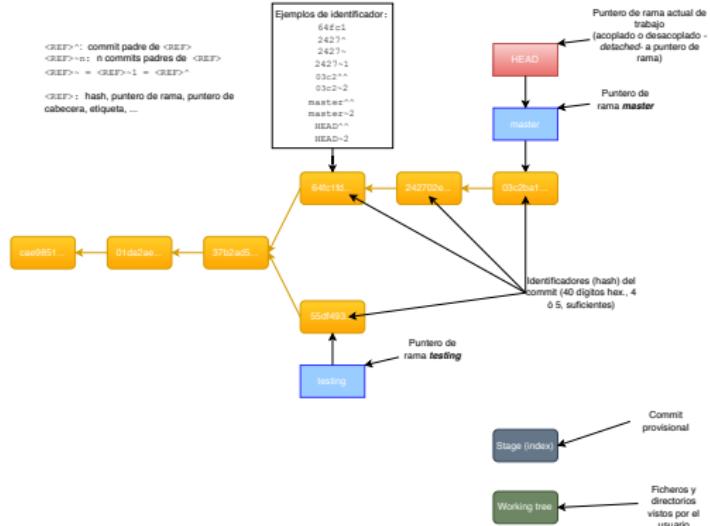
- Selección de ficheros y directorios del *directorio versionado* en un instante dado
- Se forma una estructura de datos con todos ellos y se le asigna un *hash* (como si fuera un fichero)
- Un *commit* se puede identificar de forma *absoluta* o *relativa* mediante
 - Su *hash*
 - Punteros de rama

- **Rama:** secuencias de *commits*. Tiene nombre, ejemplo: `master`, `main`, ...

- Puntero a último *commit* de la rama: **puntero de rama**
 - Nombre del puntero \equiv nombre de la rama
- Adicionalmente, todo commit apunta al commit anterior (su padre)
- Otros punteros
 - HEAD: puntero al commit (o al puntero de rama que apunta al último commit de la rama) que será el *padre* del siguiente commit
 - ...

Repositorio con varias ramas

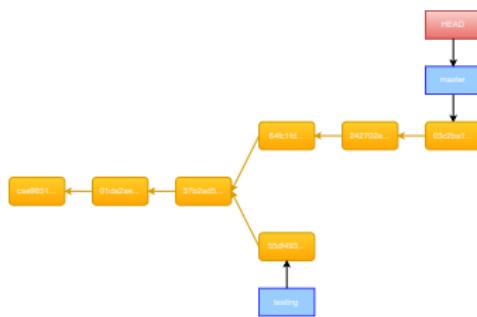
- Una rama se identifica por un puntero de rama (al último *commit* de la misma)
- En un repositorio pueden existir varias ramas
 - La rama de trabajo escogida es aquella a la que apunta HEAD, luego HEAD → *puntero de rama*
 - Cambiar a otra rama de trabajo: reapunte de HEAD a otro *puntero de rama*
 - Doble misión del puntero HEAD
 - Apuntar al puntero de rama de la rama de trabajo
 - Apuntar al *commit* (indirectamente a través del puntero de rama) que hará de *padre* del siguiente *commit*
- Ejemplo: <https://github.com/GIT-GTDM-24-25/RepoRamas.git>



Notación .../... en git log

Con esta notación, git log muestra un subconjunto específico de *commits*

- <RAMA_1>...<RAMA_2>: secuencia de *commits* que pertenecen a la <RAMA_2> y **no** a la rama <RAMA_1>
- <RAMA_1>...<RAMA_2>: secuencia de *commits* que pertenecen tanto a la rama <RAMA_2> como a la rama <RAMA_1>, pero **no a ambas**



```
$ git log --oneline 55df..master
03c2ba1 (HEAD -> master, origin/master) Sexto commit
242702e Quinto commit
64fc1fd Cuarto commit

$ git log --oneline master..55df
55df493 (origin/testing, testing) Commit en rama testing

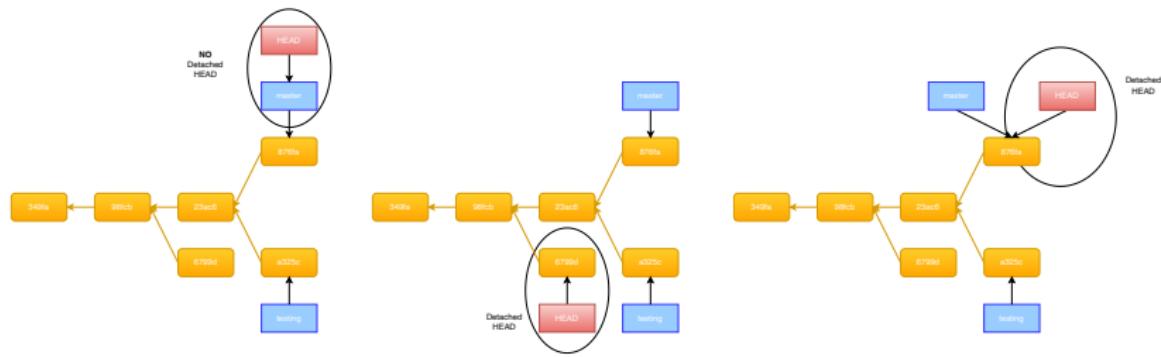
$ git log --oneline testing...03c2
55df493 (origin/testing, testing) Commit en rama testing
03c2ba1 (HEAD -> master, origin/master) Sexto commit
242702e Quinto commit
64fc1fd Cuarto commit

$ git log --oneline master...testing
55df493 (origin/testing, testing) Commit en rama testing
03c2ba1 (HEAD -> master, origin/master) Sexto commit
242702e Quinto commit
64fc1fd Cuarto commit
```

Estado *detached HEAD*

- Normalmente punteros HEAD y de rama son solidarios:
 - Se mueven simultáneamente
 - Simbólicamente: puntero HEAD *apunta* a puntero de rama
- En ocasiones se pueden *desacoplar*:
 - El movimiento de HEAD no viene acompañado por movimiento de ningún puntero de rama
 - Se dice que HEAD apunta a un *commit* (no a un puntero de rama)
 - HEAD y un puntero de rama podrían apuntar al mismo commit y estar *desacoplados*
 - Denominación de situación: **detached HEAD**
- En modo *detached HEAD*
 - Se debe ser consciente de que se está en dicha situación
 - O bien, abandonarla con técnicas que se verán más adelante
 - Si no: puede crear confusión o incluso pérdidas de *commits* en algunos casos cuando se conmute a otra rama

Ejemplos de (*no*) detached HEAD



Índice

3 Interacción básica con Git en un repositorio local

- Uso, instalación y configuración
- El repositorio local
- Esquema add-commit
- Detalles de la adición al *index*
- *Commit* al repositorio
- *Commits*, ramas, punteros y listados
- Diferencias entre ficheros: **diff** y **difftool**
- *Tags* o etiquetas
- Sinopsis

Diferencias entre ficheros: git diff

- Ficheros *textuales*: diferencias por líneas
 - Diferencias entre ficheros genéricos a (-), primero, y b (+), segundo
- Formato de salida de comando git diff:

f1.txt: fichero a (-)

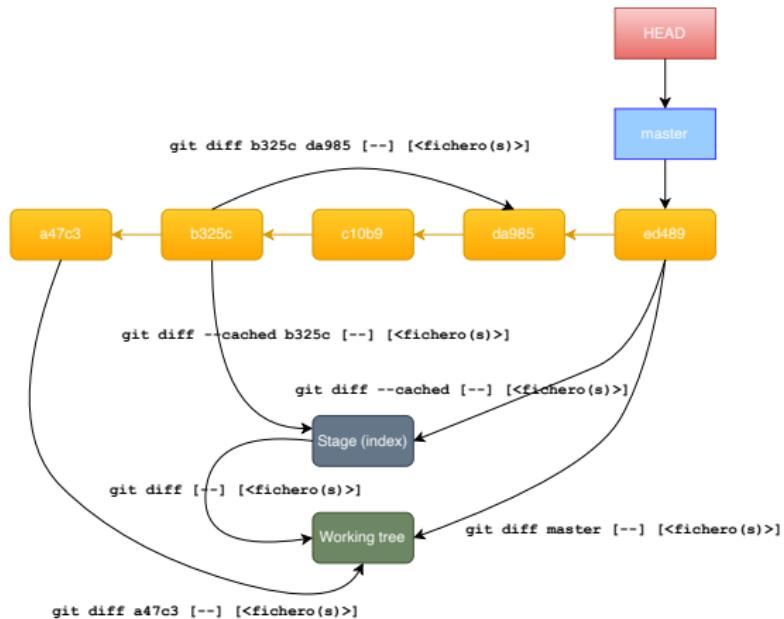
```
111
222
333
444
555
```

f2.txt: fichero b (+)

```
111
333
xxx
444
yyy
zzz
```

```
# Diferencia entre los ficheros f1.txt y f2.txt
$ git diff f1.txt f2.txt
diff --git a/a.txt b/a.txt
index 8d47907..9717822 100644
--- a/a.txt
+++ b/a.txt
@@ -1,5 +1,6 @@
 111
-222
 333
+xxx
 444
-555
+yyy
+zzz
```

Expresión de diferencias entre ficheros: sintaxis



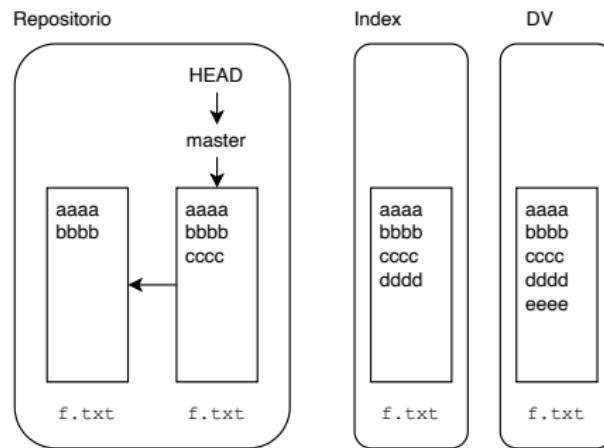
Roles a/b

Comando	a (-)	b (+)
<code>git diff [-] [<FICHERO(S)>]</code>	Index	dir. ver.
<code>git diff --cached <COMMIT> [--] [<FICHERO(S)>]</code>	<COMMIT>	Index
<code>git diff <COMMIT> --cached [-] [<FICHERO(S)>]</code>	<COMMIT>	Index
<code>git diff --cached [-] [<FICHERO(S)>]</code>	HEAD	Index
<code>git diff <COMMIT> [-] [<FICHERO(S)>]</code>	<COMMIT>	dir. ver.
<code>git diff <COMMIT_1> <COMMIT_2> [-] [<FICHERO(S)>]</code>	<COMMIT_1>	<COMMIT_2>

Si no se especifica `<FICHERO(S)>`, se muestran las diferencias de **todos** los ficheros que sean *distintos*

Expresión de diferencias entre ficheros: ejemplos

- Algunos ejemplos de extracción de diferencias
- Ejercicio previo: recrear evolución del repositorio



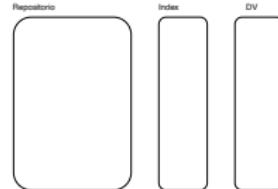
Ejercicio de recreación de repositorio para ejemplo de comando git diff (I)

```
$ mkdir WD  
$ cd WD
```



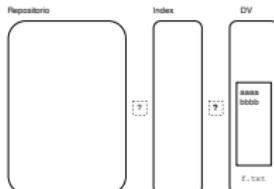
1)

```
$ git init
```



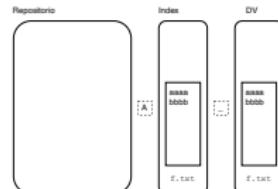
2)

```
$ code f.txt  
$ git status -s  
?? f.txt  
$
```



3)

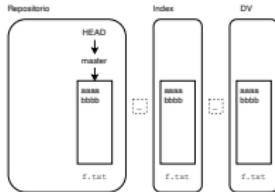
```
$ git add f.txt  
$ git status -s  
A f.txt  
$
```



4)

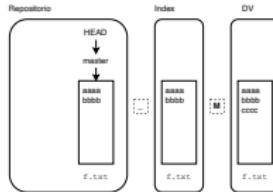
Ejercicio de recreación de repositorio para ejemplo de comando git diff (II)

```
$ git commit -m "..."  
$ git status -s  
S
```



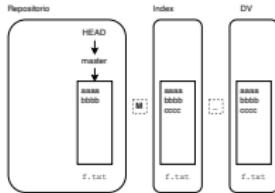
5)

```
$ code f.txt  
$ git status -s  
M  
S
```



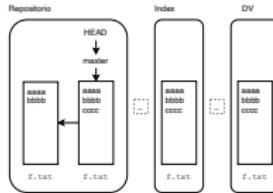
6)

```
$ git add f.txt  
$ git status -s  
M  
S
```



7)

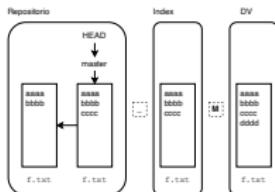
```
$ git commit -m "..."  
$ git status -s  
S
```



8)

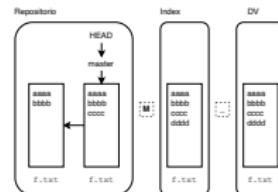
Ejercicio de recreación de repositorio para ejemplo de comando git diff (y III)

```
$ code f.txt  
$ git status -a  
M  
$
```



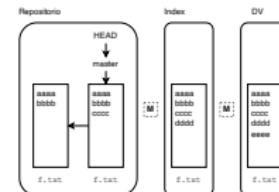
9)

```
$ git add f.txt  
$ git status -a  
M  
$
```



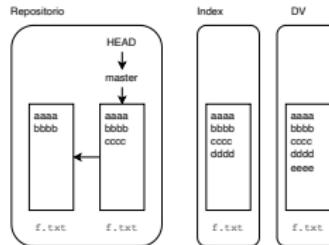
10)

```
$ code f.txt  
$ git status -a  
M  
$
```



11)

Expresión de diferencias entre ficheros: ejemplos



Roles a/b

Comando	a (-)	b (+)
git diff [--] [<FICHERO(S)>]	Index	<DV>
git diff --cached <COMMIT> [-] [<FICHERO(S)>]	<COMMIT>	Index
git diff <COMMIT> --cached [-] [<FICHERO(S)>]	<COMMIT>	Index
git diff --cached [-] [<FICHERO(S)>]	HEAD	Index
git diff <COMMIT> [-] [<FICHERO(S)>]	<COMMIT>	<DV>
git diff <COMMIT_1> <COMMIT_2> [-] [<FICHERO(S)>]	<COMMIT_1>	<COMMIT_2>

// Si no se especifica ningún fichero entonces se muestran las
// diferencias entre todos los ficheros del directorio de trabajo

// [-] indicador explícito de que siguiente parámetro es un fichero.
// Uso opcional, pero obligatorio (sin corchetes) si nombre de fichero
// coincide con identificador de commit. Ejemplo: nombre de fichero HEAD
// DV: directorio versionado
\$ git diff HEAD -> ambigüedad
-> Index(-) <DV> (+) con fichero HEAD
-> <COMMIT> (-) <DV> (+) sin especificar ficheros (todos)
// Correcto:
\$ git diff -- HEAD
\$ git diff HEAD --

// Diferencias respecto en el fichero f.txt

```

// Index(-) <DV>(+)
$ git diff f.txt

// <COMMIT>(-) Index(+)
$ git diff --cached HEAD f.txt
$ git diff HEAD --cached f.txt

$ git diff --cached HEAD~ f.txt
$ git diff HEAD~ --cached f.txt

// HEAD(-) Index(+)
$ git diff --cached f.txt

// <COMMIT>(-) <DV>(+)
$ git diff HEAD f.txt
$ git diff HEAD~ f.txt

// <COMMIT_1>(-) <COMMIT_2>(+)
$ git diff HEAD HEAD~ f.txt
$ git diff HEAD~ HEAD f.txt

```

Expresión de diferencias entre ficheros: difftool

difftool: herramienta alternativa a diff. Configuración para Visual Studio Code:

Configuración de Visual Studio Code como herramienta diff

```
// Indicación de herramienta difftool: Visual Studio code
// IMPORTANTE: emplense comillas simples ', no dobles " en Linux/Mac o Windows Git Bash
//               emplense comillas dobles " en Windows CMD
$ git config --global diff.tool vscode
$ git config --global difftool.vscode.cmd 'code --wait --diff $LOCAL $REMOTE'
```

Idem. que ejemplos anteriores, pero con difftool en vez de diff

```
// Diferencias respecto en el fichero f.txt

// Index(-) <DV>(+)
$ git difftool f.txt

// <COMMIT>(-) Index(+)
$ git difftool --cached HEAD f.txt
$ git difftool HEAD --cached f.txt

$ git difftool --cached HEAD^ f.txt
$ git difftool HEAD^ --cached f.txt

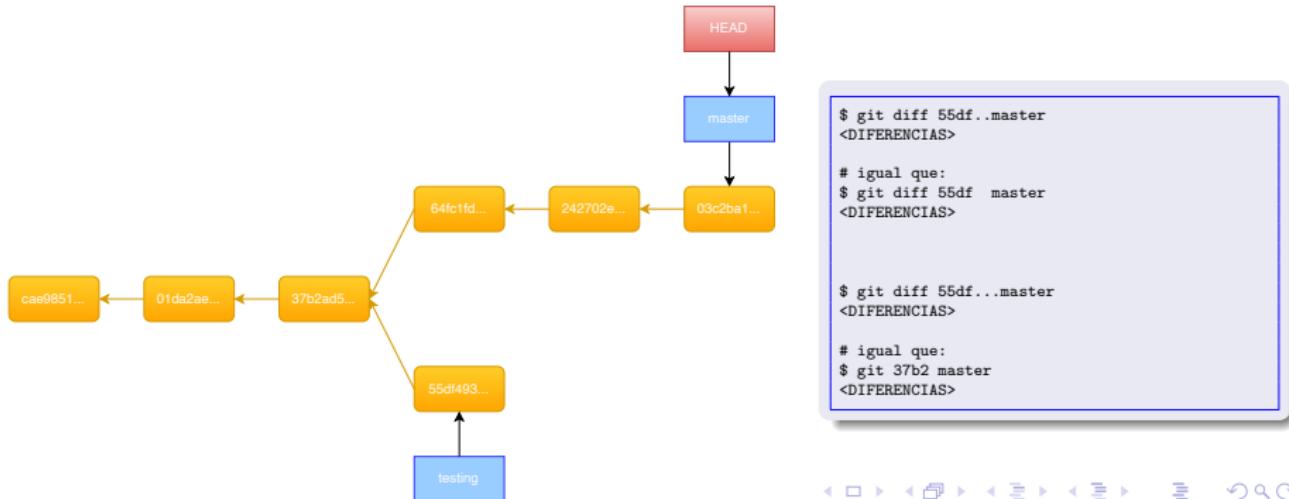
// HEAD(-) Index(+)
$ git difftool --cached f.txt

// <COMMIT>(-) <DV>(+)
$ git difftool HEAD f.txt
$ git difftool HEAD^ f.txt

// <COMMIT_1>(-) <COMMIT_2>(+)
$ git difftool HEAD HEAD^ f.txt
$ git difftool HEAD^ HEAD f.txt
```

Notación .../... en git diff

- Se puede emplear la notación .../... con el comando `git diff[tool]`
- Significado distinto a cuando se emplea con `git log`
- Notación ...: Equivalencia
 - `git diff <COMMIT_A>..<COMMIT_B>` equivalente a
`git diff <COMMIT_A> <COMMIT_B>`
- Notación: Equivalencia
 - `git diff <COMMIT_A>...<COMMIT_B>` equivalente a
`git diff <ANCESTRO_COMUN_MAS_RECIENTE> <COMMIT_B>`



Índice

3 Interacción básica con Git en un repositorio local

- Uso, instalación y configuración
- El repositorio local
- Esquema add-commit
- Detalles de la adición al *index*
- *Commit* al repositorio
- *Commits*, ramas, punteros y listados
- Diferencias entre ficheros: diff y difftool
- **Tags o etiquetas**
- Sinopsis

Etiquetas (*tags*)

- Puntos específicos del repositorio: nombres propios de *commits*
 - La <ETIQUETA> no debe contener espacios en blanco
- Equivalente a cualquier otra referencia de commit
- Dos tipos: anotadas (con más información, y otros detalles...) y ligeras

```
// Creación de etiqueta ligera.  
// El nombre de la etiqueta no puede llevar espacios en blanco  
// por lo que no requiere entrecorillarse. En este caso <ETIQUETA>=v3.0  
$ git tag v3.0  
  
// Creación de etiqueta anotada (se incluye más información):  
$ git tag -a <ETIQUETA> -m <COMENTARIO>  
// Creación de etiqueta anotada: v2.0  
//   (cualquier string sin espacios en blanco)  
$ git tag -a v2.0 -m "Versión 2.0 del código"  
  
// Lista todas las etiquetas  
$ git tag  
  
// Detalla una etiqueta  
$ git show v2.0  
  
// Creación de etiqueta ligera a posteriori asociada al commit 9fcdd04  
$ git tag v4.0 9fcdd04  
  
// Borrar una etiqueta:  
$ git tag --delete <ETIQUETA>
```

Índice

3 Interacción básica con Git en un repositorio local

- Uso, instalación y configuración
- El repositorio local
- Esquema add-commit
- Detalles de la adición al *index*
- *Commit* al repositorio
- *Commits*, ramas, punteros y listados
- Diferencias entre ficheros: diff y difftool
- Tags o etiquetas
- Sinopsis

Sinopsis: procedimiento rutinario

- Antes de comenzar a trabajar (tras `git clone ...`, o `git init`):
 - Si no existe `.gitignore`: crearlo
 - Si existe: si es necesario, modificarlo oportunamente
- De forma repetitiva:

Para verificar estado

```
$ git status [-s] % estado e index
$ git log [--oneline --all --graph ...] % Commits en repositorio
```

Cuando se estime necesario

```
$ git add -A (u opción oportuna)
...
$ git add -A (u opción oportuna)
$ git commit -m <COMENTARIO>
```

Para observar los *git-hashes* y ficheros en *index* y repositorio

```
$ git hash-object <FICHERO(S)>           // en directorio versionado
$ git ls-files -s [ <FICHERO(S)> ]        // en index
$ git ls-tree -r <commit> [ <FICHERO(S)> ] // en commit
```

Etiquetado: Poner *nombre propio* a un commit

```
$ git tag <ETIQUETA> <COMMIT>
// Si se omite <COMMIT>
// se supone HEAD
```

Para observar diferencias entre versiones distintas de ficheros

```
$ git diff [ <PARAMETROS> ] [ <FICHERO(S)> ]
// Mejor si está configurado con un editor cómodo
$ git difftool [ <PARAMETROS> ] [ <FICHERO(S)> ]
```

Ejercicios de recapitulación

- Ejercicios de Git en W3Schools:

 - Desde **Git Get Started** hasta **Git Help** inclusive

Índice

- 1 Introducción
- 2 Ventana o interfaz de línea de comandos (CLI)
- 3 Interacción básica con Git en un repositorio local
- 4 Comandos `reset` y `checkout`
- 5 Ramas
- 6 Interacción con repositorios remotos
- 7 Aspectos básicos de GitHub

Uso

- Control del repositorio, del index y del directorio versionado
- *Potencial* manipulación de:
 - Puntero HEAD
 - Puntero de rama (solo comando `reset`)
 - Inversión de flujo de información:
 - Carga de `commit` a `index`
 - Carga de `index` a directorio de trabajo

Índice

4 Comandos reset y checkout

- Comando reset
- Comando checkout
- Sinopsis

Comando reset: planteamiento

- Forma genérica:

```
$ git reset [ --soft | --mixed | --hard ] [ <COMMIT> ] [ -- ] [ <FICHERO(S)> ]
```

- El comportamiento de `reset` depende del tipo de parámetros

- Si **NO** intervienen ficheros concretos

```
$ git reset [ --soft | --mixed | --hard ] [ <COMMIT> ]
```

- se **redirigen** al unísono el puntero `HEAD` y el puntero de rama (este último, si no se está en modo *detached head*) al *commit* (puntero) indicado en el comando
 - Además, tres modos: `--soft`, `--mixed` y `--hard`

- Si intervienen ficheros concretos (no hay opciones)

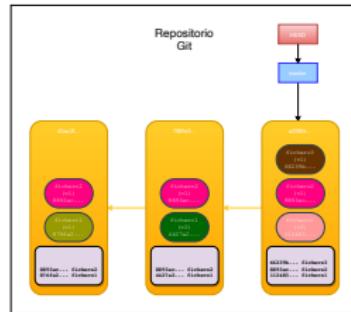
```
$ git reset [ <COMMIT> ] [ -- ] <FICHERO(S)>
```

- No hay movimiento alguno de punteros
 - Los ficheros indicados son *recargados* desde el *commit* indicado en el comando al *index*

Punto de partida

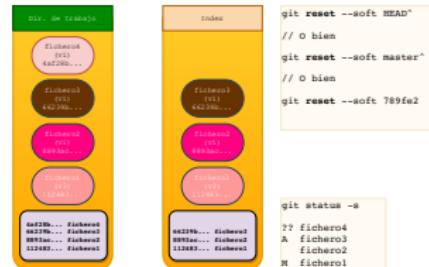
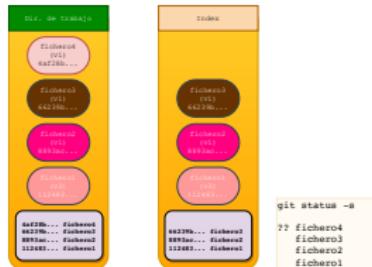
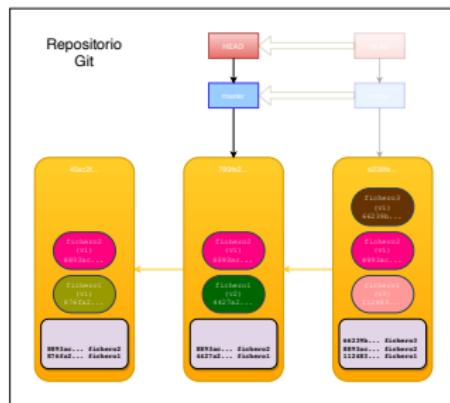
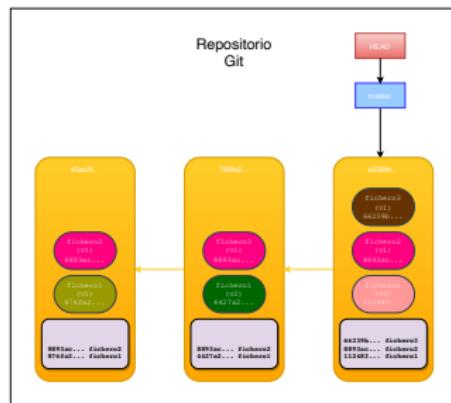
```
Clonación de EjemploReset  
$ cd ~/Desktop/GITGITHUB  
$ git clone https://github.com/GIT-GTDM-24-25/EjemploReset  
$ cd EjemploReset  
$ git log --oneline  
% Caso omiso de momento a ramas remotas origin/master y origin/HEAD  
  
% Creación de fichero no versionado fichero4  
$ echo "Version 1 de fichero 4" > fichero4
```

IMPORTANTE PARA DESPUES: !!!TOMAD NOTA DEL HASH DEL COMMIT MÁS RECIENTE!!!
Lo denominaremos <ULTIMO_COMMIT>



Reset soft sin ficheros: git reset --soft <COMMIT>

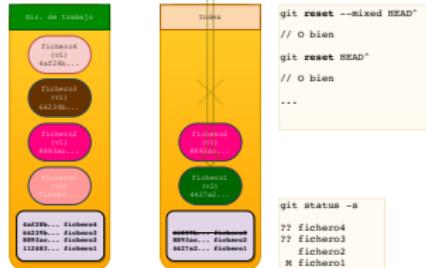
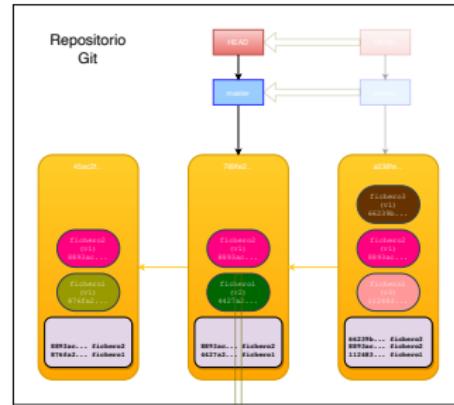
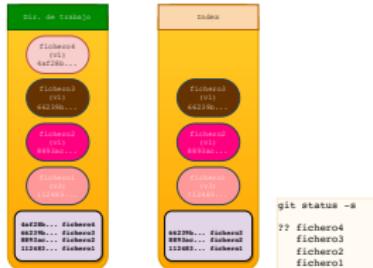
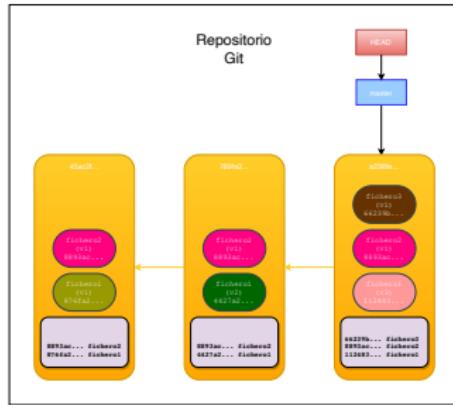
- Movimiento de puntero HEAD, y de rama (si HEAD apunta a un puntero de rama —no *detached head*—)



Reset mixed sin ficheros: git reset [--mixed] <COMMIT>

- Como --soft y además recarga index con commit
- Prueba: recarga de repositorio o bien

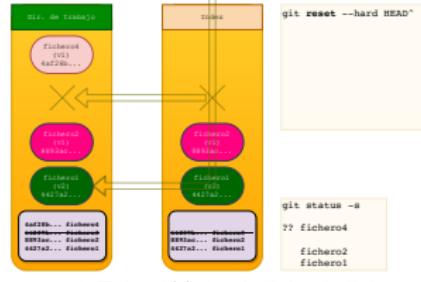
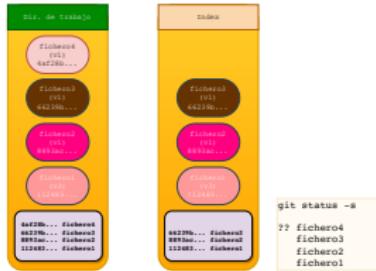
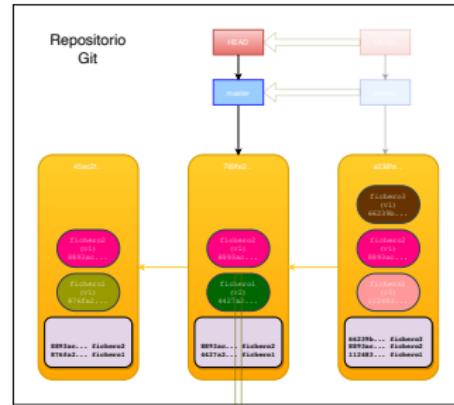
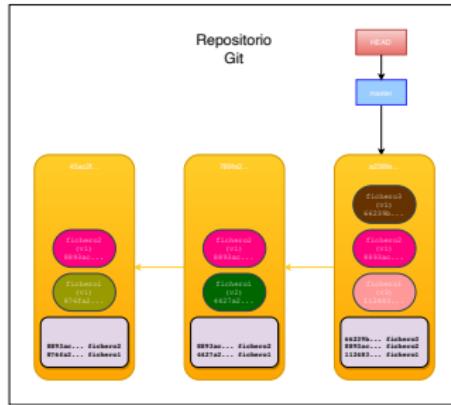
Repositorio a estado original
`$ git reset --hard <ULTIMO_COMMIT>`



Reset hard sin ficheros: git reset --hard <COMMIT>

- Como --mixed y además recarga directorio de trabajo con el contenido del *index*
- Prueba: recarga de repositorio o bien

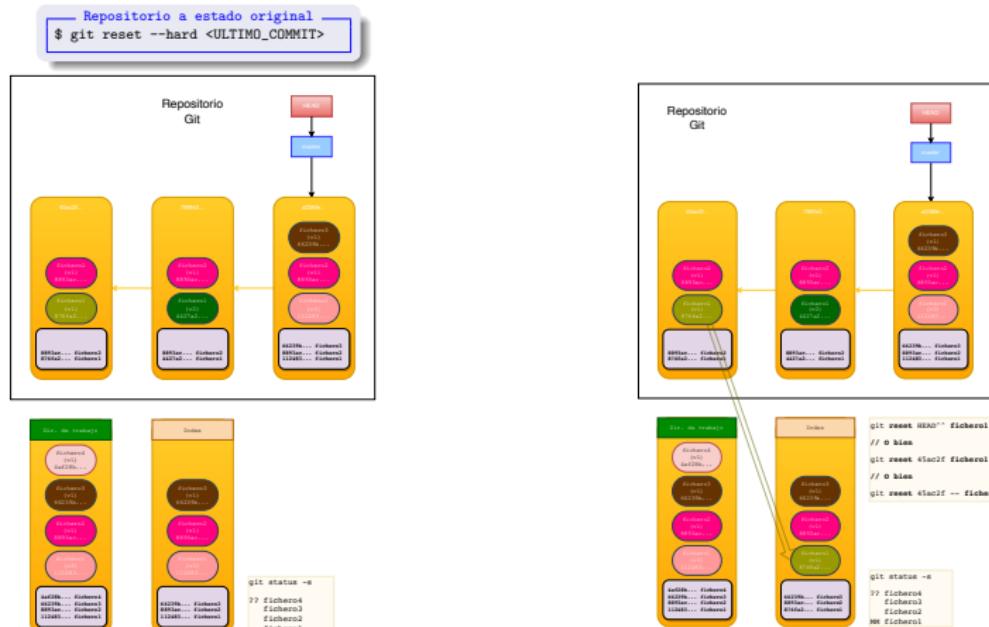
Repositorio a estado original
`$ git reset --hard <ULTIMO_COMMIT>`



reset con fichero(s)

- Si hay fichero(s) acompañando al comando:

- NO hay movimiento de HEAD ni de puntero de rama
- No hay opciones --hard ni --soft ni --mixed, aunque actúa como si estuviera por defecto activa la opción --mixed, recargando por tanto fichero(s) en el index.
- Equivalente a: \$ git restore --staged <FICHERO(S)>
- Prueba: recarga de repositorio o bien



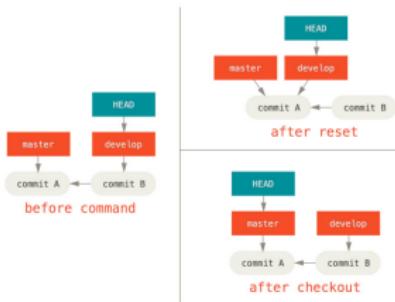
Índice

4 Comandos reset y checkout

- Comando reset
- **Comando checkout**
- Sinopsis

Checkout sin ficheros: git checkout <COMMIT>

- checkout modifica HEAD actualizando el *index* y el directorio de trabajo (similar a git reset --hard [<COMMIT>])
- Diferencias entre git checkout <COMMIT> y git reset --hard [<COMMIT>]
 - checkout verifica que el directorio de trabajo tiene todos los cambios guardados antes de commutar al nuevo directorio de trabajo correspondiente al especificado en el comando *checkout* (*reset*, no hace esta verificación)
 - checkout solo actualiza HEAD para apuntar a la nueva rama; *reset* mueve HEAD y el puntero de la rama (si no se está en modo *detached head*)



Diferencias (supondremos que estamos en la rama develop, —es decir, HEAD \Rightarrow develop)

`$ git reset master`

reset

`$ git checkout master`

checkout

- `git checkout HEAD` o `git checkout` no hacen lo que supuestamente deben hacer: no tienen efecto (solo proporcionan información).
 - Para *forzar* a hacerlo: `git checkout -f HEAD`, o bien, `git reset --hard HEAD`
- Comando similar: `$ git switch <RAMA>`

checkout: punto de partida

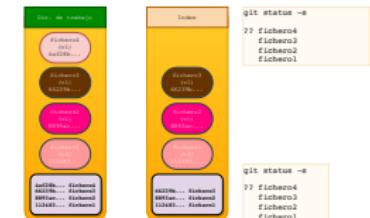
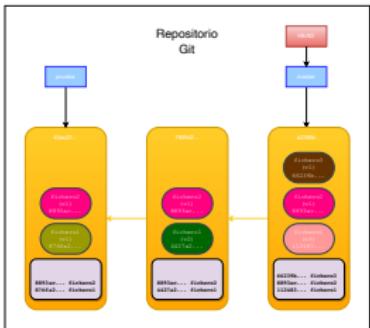
Repetir clonación de EjemploCheckout

```
$ git clone https://github.com/GIT-GTDM-24-25/EjemploCheckout
$ cd EjemploCheckout
$ git log --oneline --all
% Caso omiso de momento a ramas remotas origin/master y origin/HEAD

% Creación de fichero no versionado fichero4
$ echo "Version 1 de fichero 4" > fichero4
```

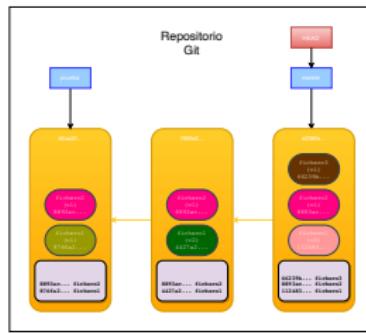
Solo se baja rama master: conviene tener también rama prueba

```
// Ya se explicará: ejecútese comandos obviando el porqué.
$ git checkout prueba
$ git checkout master
$ git log --oneline --all
```

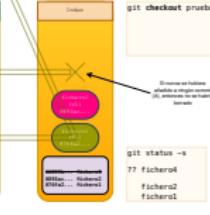
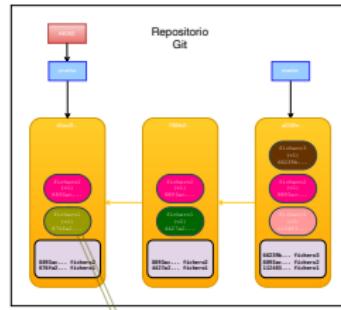


checkout a rama sin especificación de ficheros

- **Importante:** contenido debe ser idéntico en dir. trabajo, en index y en HEAD; si no, no se ejecuta.
 - Si no se da dicha condición, se puede *forzar* con opción `-f`
- **Recuérdese** que si ya se está en dicha rama: no se hace nada (no se recarga ni index ni directorio de trabajo)
 - Se puede *forzar* recarga de index y de dir. de trabajo con opción `-f`



```
git status -s
?? fichero4
fichero3
fichero2
fichero1
```

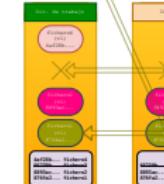
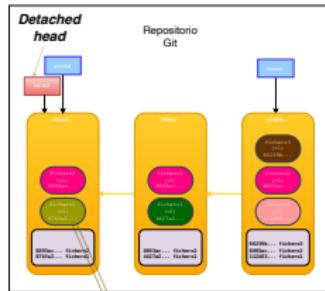
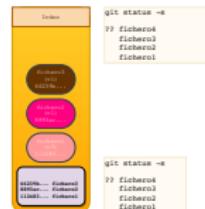
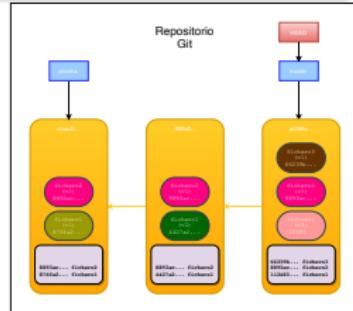


```
git status -s
?? fichero4
fichero3
fichero2
fichero1
```

checkout a *commit hash* (no a rama) sin especificación de ficheros: *detached head*

- Idem que anterior, pero con una diferencia **importante**: provoca situación de **detached head**
- Prueba: recarga de repositorio o bien

Repositorio a estado original
`$ git checkout master`

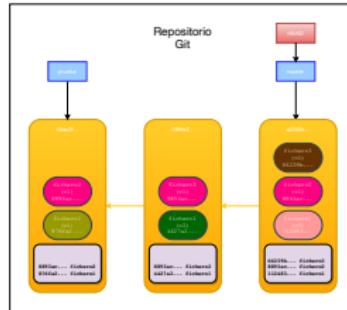


checkout especificando ficheros

- Al igual que `reset` con ficheros, `checkout` con ficheros **NO** mueve HEAD (ni evidentemente el puntero la rama). No avisa de posible sobreescritura con pérdida de información
- Se recargan en el *index* y en el directorio de trabajo los ficheros del *commit* especificado
- Equivalente a: `$ git restore <FICHERO(S)>`
- Prueba: recarga de repositorio o bien

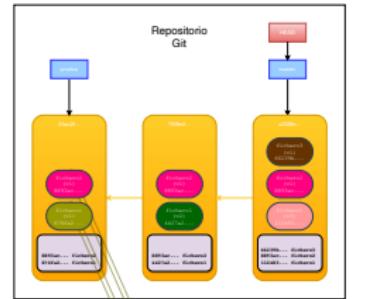
Repositorio a estado original

```
$ git checkout master
$ code fichero2
$ git add fichero2
$ code fichero1
```



`git status -s`

```
?? fichero4
?? fichero3
?? fichero2
M fichero1
M fichero0
```



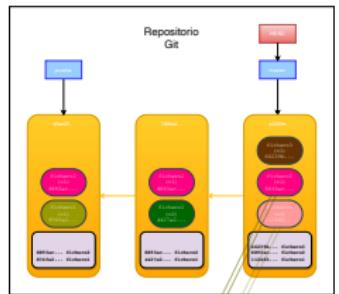
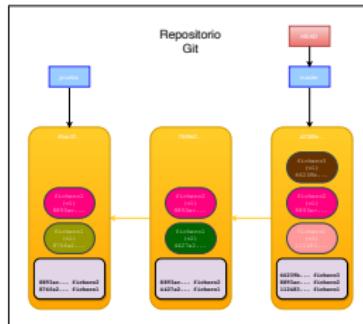
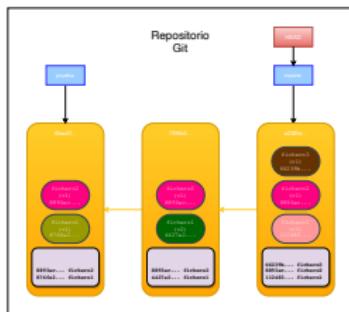
`git status -s`

```
?? fichero4
fichero3
fichero2
M fichero1
M fichero0
```

checkout especificando ficheros: singularidad

- Diferencia **importante** entre especificar el commit HEAD o no hacerlo: comportamiento singular (recarga desde el *index*)
- Prueba: recarga de repositorio o bien (**IMPORTANTE**: opción *-f*)

```
Repositorio a estado original
$ git checkout -f master
$ code fichero2
$ git add fichero2
$ code fichero1
```



```
git status -s
?? fichero1
fichero2
M fichero2
M fichero1
```

```
git status -s
git checkout fichero1 fichero2
?? fichero1
fichero2
M fichero2
M fichero1
```

```
git status -s
git checkout HEAD fichero1 fichero2
?? fichero1
fichero2
fichero1
fichero2
```

Comparativa general: reset y checkout

- Tabla resumen y comparativa:

	Actualización	Index	Dir. trabajo	Seguro*
Especificación sin ficheros				
git reset --soft [<COMMIT>]	Puntero rama + HEAD	No	No	Sí
git reset [--mixed] [<COMMIT>]	Puntero rama + HEAD	Sí	No	Sí
git reset --hard [<COMMIT>]	Puntero rama + HEAD	Sí	Sí	NO
git checkout [<COMMIT>] †	HEAD	Sí	Sí	Sí
Especificación con fichero(s)				
git reset [<COMMIT>] <FICHERO(S)>	—	Sí	No	Sí
git checkout [<COMMIT>] <FICHERO(S)> ‡	—	Sí‡	Sí	NO

†: si <COMMIT>==HEAD o se omite, no se hace absolutamente nada, a no ser que se emplee la opción -f.

‡: si se omite <COMMIT>, solo se recarga el directorio de trabajo con el index, pero el propio index no es recargado o actualizado. Si <COMMIT>==HEAD, funciona como se supone que debe hacerlo.

*: “NO” significa que sobreescribe ficheros en el directorio de trabajo sin preguntar o verificar que todo está guardado.

Salvo en los casos † y ‡, cuando <COMMIT> no se especifica, el valor por defecto es HEAD

Índice

4 Comandos reset y checkout

- Comando reset
- Comando checkout
- Sinopsis

Sinopsis: control

Reset: control de HEAD y puntero de rama —si no hay <FICHERO(S)>—

```
$ git reset [ --soft | --mixed | --hard ] [ <COMMIT> ] [ -- ] [ <FICHERO(S)> ]
```

Checkout: control de HEAD —si no hay <FICHERO(S)>—

```
$ git checkout [ <COMMIT> ] [ -- ] [ <FICHERO(S)> ]
```

- Y *potencialmente* recarga de:
 - *index*
 - Directorio de trabajo
- En repositorios **compartidos**: comando **reset** **¡¡¡conflictivo y peligroso!!!**

Índice

- 1 Introducción
- 2 Ventana o interfaz de línea de comandos (CLI)
- 3 Interacción básica con Git en un repositorio local
- 4 Comandos reset y checkout
- 5 Ramas
- 6 Interacción con repositorios remotos
- 7 Aspectos básicos de GitHub

Índice

5 Ramas

- Simulador
- Creación y cambio
- Fusión de ramas
- Información y borrado de ramas
- Sinopsis
- Ejemplo

Ejemplos

- Creación y cambio de ramas
- Control de punteros de ramas
- Situaciones de *detached head*
- Fusiones
- ...
- Ejemplos en simulador:
<http://git-school.github.io/visualizing-git/>

Índice

5 Ramas

- Simulador
- **Creación y cambio**
- Fusión de ramas
- Información y borrado de ramas
- Sinopsis
- Ejemplo

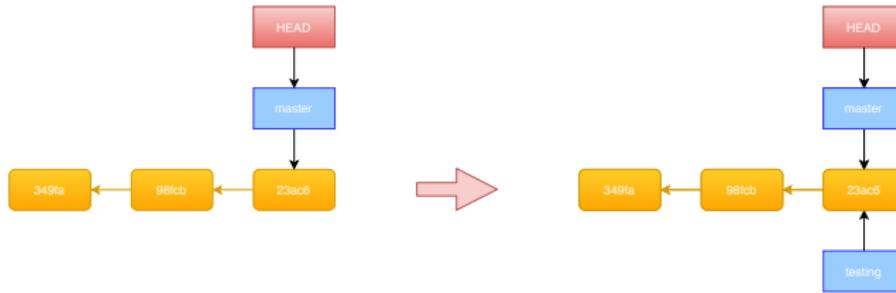
Creación de nueva rama

- Creación de nueva rama:

```
$ git log --oneline --all --graph
* 23ac654 (HEAD -> master) Segunda modificación
* 98fcb87 Primera modificación
* 349fa2e Inic.

// Creación de nueva rama
$ git branch testing

$ git log --oneline --all --graph
* 23ac654 (HEAD -> master, testing) Segunda modificación
* 98fcb87 Primera modificación
* 349fa2e Inic.
```



Conmutación a nueva rama

- Antes del cambio de rama:
 - Todos los cambios deben estar guardados en el repositorio: directorio de trabajo e index
- El comando `git checkout <RAMA>` provoca que HEAD *reapunte* a <RAMA>

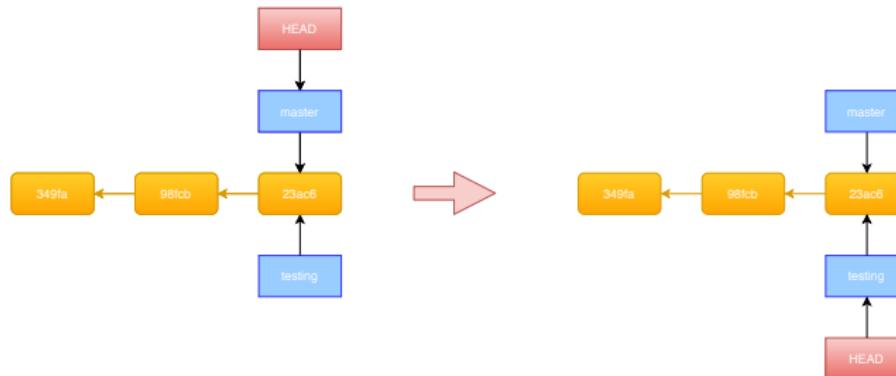
```
$ git log --oneline --all --graph
* 23ac654 (HEAD -> master, testing) Segunda modificación
* 98fcb87 Primera modificación
* 349fa2e Inic.

// Conmutación a nueva rama
$ git checkout testing

$ git log --oneline --all --graph
* 23ac654 (HEAD -> testing, master) Segunda modificación
* 98fcb87 Primera modificación
* 349fa2e Inic.
```

// Los dos comandos siguientes:
`git branch <RAMA>`
`git checkout <RAMA>`

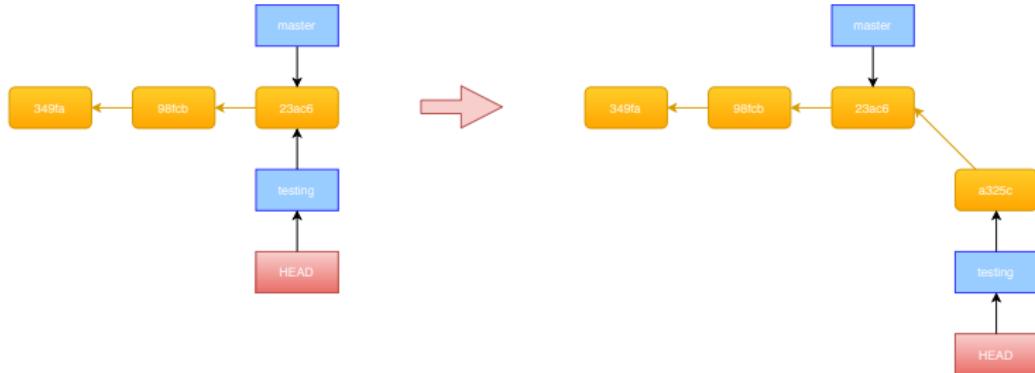
// son equivalentes a:
`git checkout -b <RAMA>`



commit desde la nueva rama

```
// Nuevo/modificación de ficheros
...
$ git add <FICHERO(S)>
// Commit desde la nueva rama
$ git commit -m "Modificación en rama"

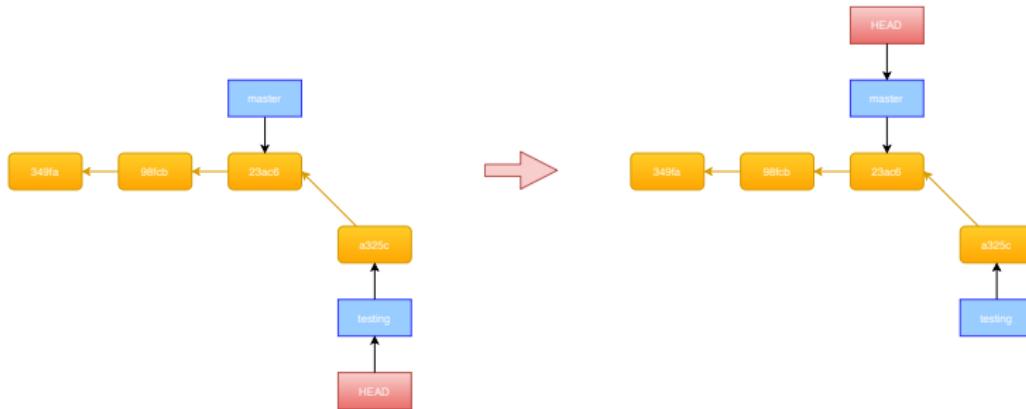
$ git log --oneline --all --graph
* a325cdf (HEAD -> testing) Modificación en rama
* 23ac654 (master) Segunda modificación
* 98fcf87 Primera modificación
* 349fa2e Inic.
```



Vuelta a rama master

```
// Comutación a rama master
$ git checkout master

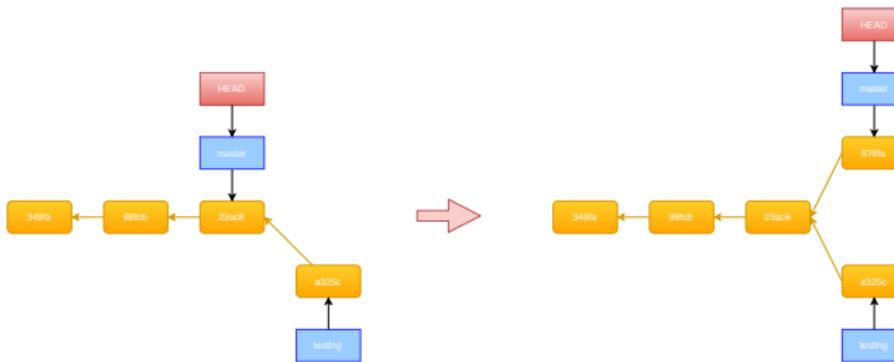
$ git log --oneline --all --graph
* a325cdf (testing) Modificación en rama
* 23ac654 (HEAD -> master) Segunda modificación
* 98fc87 Primera modificación
* 349fa2e Inic.
```



Nuevo commit desde la rama master

```
// commit desde master
$ git add -A
$ git commit -m "Tercera modificación en master"

$ git log --oneline --decorate --graph --all
* 876fa23 (HEAD -> master) Tercera modificación en master
| * a325cdf (testing) Modificación en rama
|/
* 23ac654 Segunda modificación
* 98fcb87 Primera modificación
* 349fa2e Inic.
```



Checkout a *commit hash* (no a rama): *detached HEAD*

```
$ git checkout 98fc87 // No se ha hecho un checkout a una RAMA (puntero), sino a un COMMIT hash
// Equivalentemente: git checkout HEAD^~, o git checkout HEAD^2
// o git checkout master^~, o git checkout 23ac654^~, o git checkout testing^~, ...

Note: checking out '98fc87'.

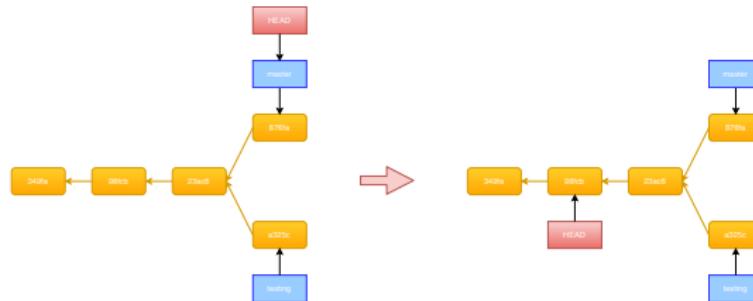
You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

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>

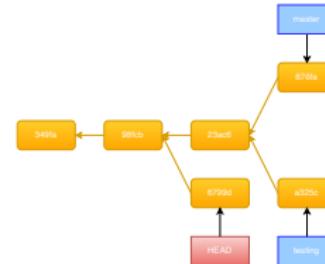
HEAD is now at 98fc87... Añado 1234

git log --oneline --decorate --graph --all
* 876fa23 (master) Tercera modificación en master
| * a325cdf (testing) Modificación en rama
|/
* 23ac654 Segunda modificación
* 98fc87 (HEAD) Primera modificación
* 349fa2e Inic.
```



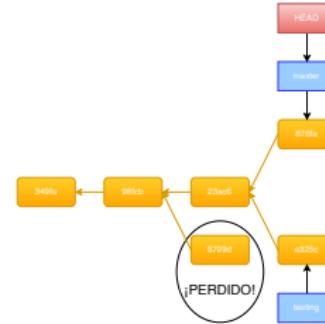
Commits con *detached HEAD*

```
$ git checkout 98fcb
$ git add <FICHERO(S)>
$ git commit -m "Modificación en rama sin nombre"
```



- ¡Cuidado!: problemas en el futuro si el *commit* no pertenece a ninguna rama

```
$ git checkout master
$ git log --oneline --graph --all
```

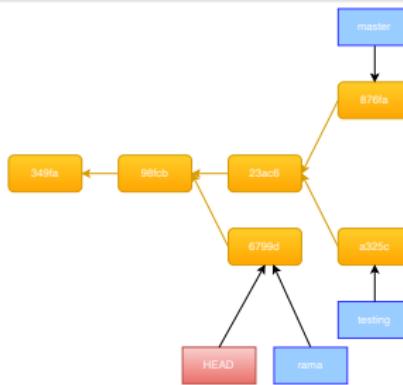


Commits con *detached HEAD* (cont.)

- Solución: crear una rama en el último commit (apuntado por el puntero HEAD desacoplado)

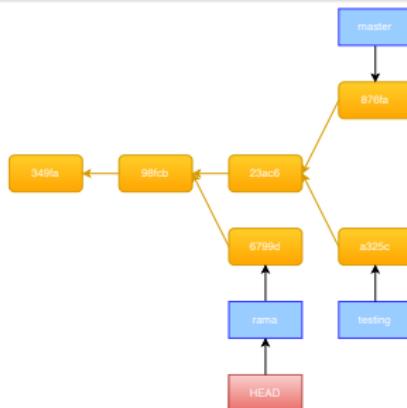
Sin acople, pero con nueva rama

```
$ git branch rama
$ git log --oneline --graph --all
```



Con acople a nueva rama

```
$ git checkout -b rama
$ git log --oneline --graph --all
```



reflog: log de todas las operaciones

```
$ git reflog
a325cdf (HEAD -> testing) HEAD@0: checkout: moving from
6799deefb2e68d9a9f097b92b5e2a59f3d7867c5 to testing
6799dee HEAD@1: commit: Modificación en rama sin nombre
98fc87 HEAD@2: checkout: moving from master to 98fc87
876fa23 (master) HEAD@3: commit: Tercera modificación en master
23ac654 HEAD@4: checkout: moving from testing to master
a325cdf (HEAD -> testing) HEAD@5: commit: Modificación en rama
23ac654 HEAD@6: checkout: moving from master to testing
23ac654 HEAD@7: commit: Segunda modificación
98fc87 HEAD@8: commit: Primera modificación
349fa2e HEAD@9: commit (initial): Inic.
```

- Si se van añadiendo commits con el puntero de cabecera desacoplado (*detached HEAD*)
 - Se pueden perder si no se nombra rama
- El comando `reflog` lista todos los *commits* estén o no en ramas
- Recuperación de número de *commit* a partir del comentario
- Solo en local y no se asegura que permanezca en el tiempo

Sinopsis

- Creación de rama

Creación de una rama

```
$ git branch <RAMA>
```

- Creación de rama y cambio

Creación de la rama <RAMA> y conmutación a ella

```
$ git branch <RAMA>  
$ git checkout <RAMA>
```

Creación de la rama <RAMA> y conmutación a ella

```
$ git checkout -b <RAMA>
```

Creación de la rama <RAMA> partiendo de <COMMIT> y conmutación a ella

```
$ git checkout -b <RAMA> <COMMIT>
```

Índice

5 Ramas

- Simulador
- Creación y cambio
- Fusión de ramas**
- Información y borrado de ramas
- Sinopsis
- Ejemplo

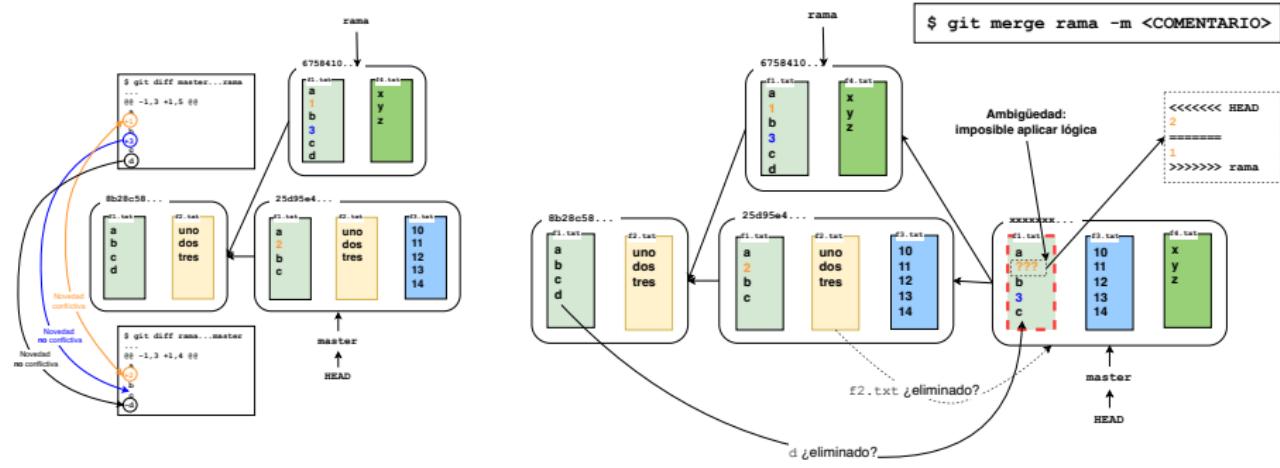
Concepto de fusión, mezcla o *merge*

Merge: mezcla o fusión de dos (o más) *commits* en un nuevo *commit de fusión*

- La intención es recoger las *aportaciones* de ambas ramas (tomando como referencia un *commit ancestro común*) y dejarlas en un nuevo *commit* en la rama de trabajo.
- Comando:
`$ git merge <RAMA> -m <COMENTARIO_COMMIT_DE_FUSION>`
- Mientras las aportaciones en ambas ramas sean en ficheros disjuntos, no hay problema. En caso contrario:
 - Aplicación de algoritmos complejos de *mezcla* (*ort*, *recursivo*, ...). Resultado
 - OK: resultado **supuestamente** correcto
 - Con conflicto: resolución manual

Concepto de fusión, mezcla o *merge*: gráficamente

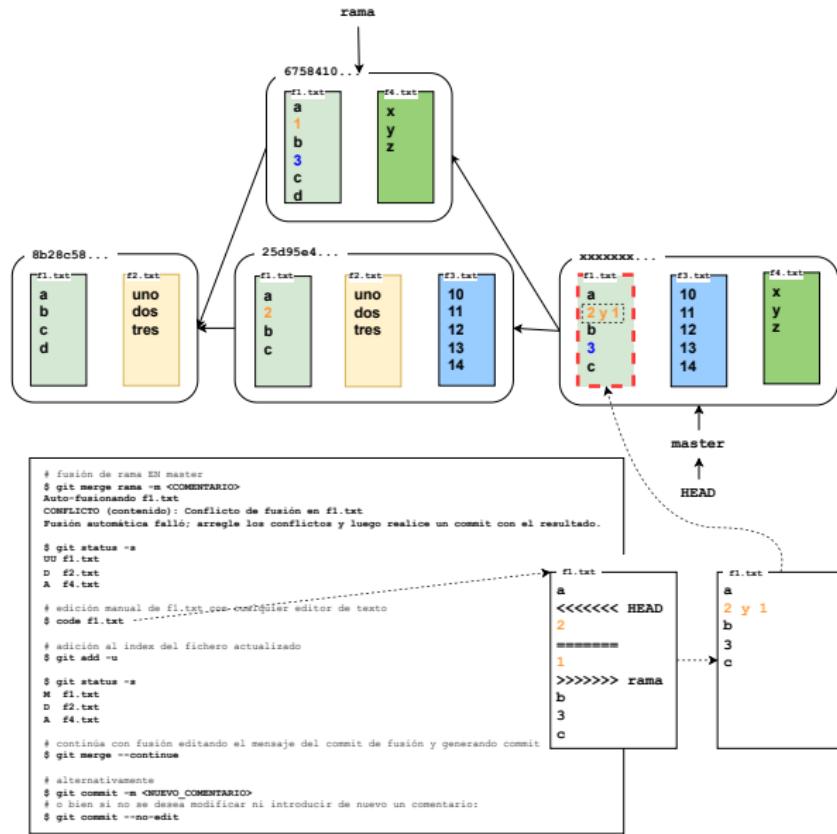
- Algoritmos de fusión: se prima aceptar la *novedad* (en cualquier sentido)
- Si hay novedades *contradicторias*: inacción y solución manual
- Ejemplo: <https://github.com/GTDM-GIT-24-25/ConceptoMerge>



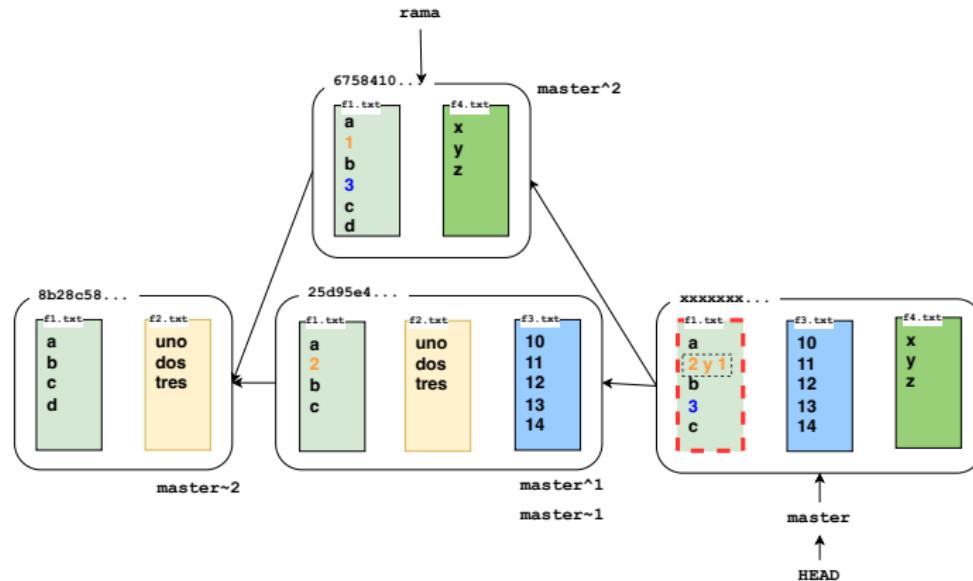
Concepto de fusión, mezcla o *merge* (cont.)

- Si la fusión es exitosa se crea automáticamente un *commit* denominado *de fusión*
- Si no, el conflicto se debe resolver manualmente, añadir la resolución al *index* y después generar el *commit*, todo ello manualmente.
- El nuevo *commit* de fusión formará parte de la rama de trabajo: fusión de *la otra rama* (<RAMA>) **en** la rama de trabajo
 - Con estrategias de fusión convencionales, el contenido del nuevo *commit* de fusión es el mismo se fusione desde donde se fusioné

Ejemplo de fusión



Ejemplo de *fusión* (cont.)



- El *commit* de fusión (que pertenece a la rama `master`) tiene **dos padres**: `HEAD^1` o `master^1` (padre en la rama `master`) y `HEAD^2` o `master^2` (padre en la rama `rama`).
- Con `$ git merge --no-commit <RAMA> -m <COMENTARIO>` se informa sobre posibilidad de realizar la fusión automáticamente, pero no se realiza
 - Con `$ git merge --continue`, se continuaría con el proceso total

Anulación de una fusión

- Si no ha finalizado la fusión

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

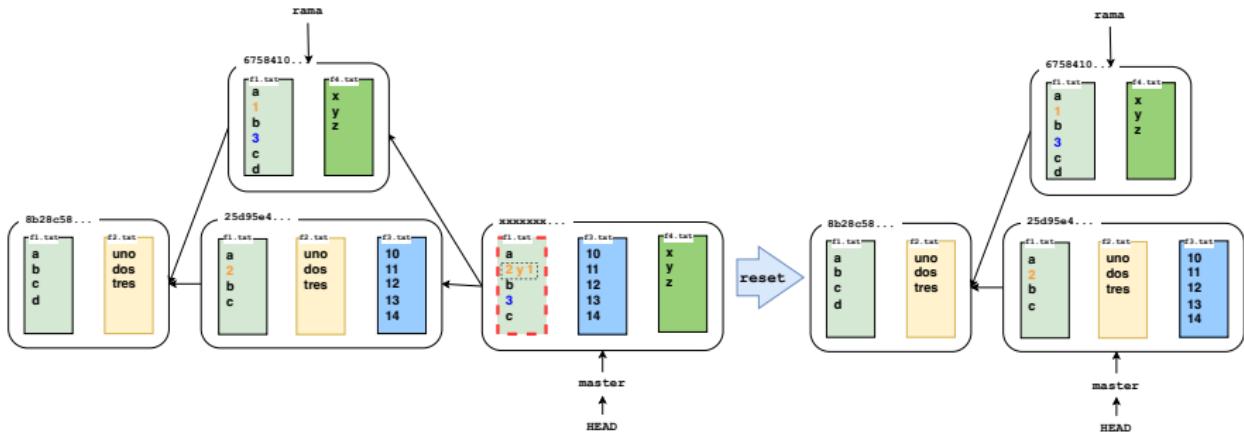
- Si ya se realizó la fusión (operación reset):

```
// master^1: padre de la misma rama
$ git reset --hard master^1

// HEAD^1: idem
$ git reset --hard HEAD^1

// master: master^1 por defecto
$ git reset --hard master

// Cuidado: ¡detached head!
/// Ya no es lo mismo!!!
$ git reset --hard 25d95e4
```



Fusión manual selectiva (cuando fusión automática **no** es posible)

- Opciones:
 - Fusión manual:
 - Edición de ficheros conflictivos
 - Selección del fichero de una rama: la intención no es fundir sino seleccionar una opción
 - La de la rama de trabajo:
 - La otra rama:

```
Elección de fichero <FICHERO> de la rama actual de trabajo  
$ git checkout --ours <FICHERO>
```

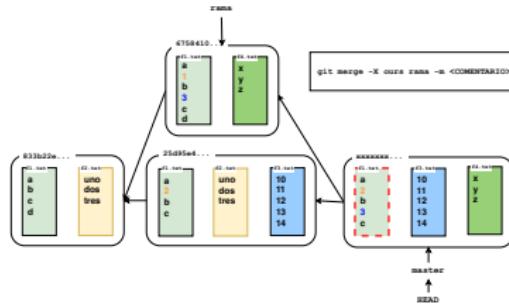
- La otra rama:

```
Elección de fichero <FICHERO> de la otra rama  
$ git checkout --theirs <FICHERO>
```

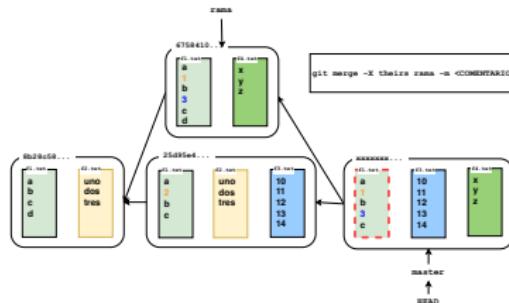
- En cualquier caso, tras la edición o selección por conflicto:
 - Adición al *index*: \$ git add <FICHERO_CONFLICTIVO>
 - Finalización del proceso: \$ git merge --continue

Algunas estrategias de fusión **libres** de conflictos: ejemplos

- Supongamos que estamos trabajando en la rama master.
- Estrategia -X ours

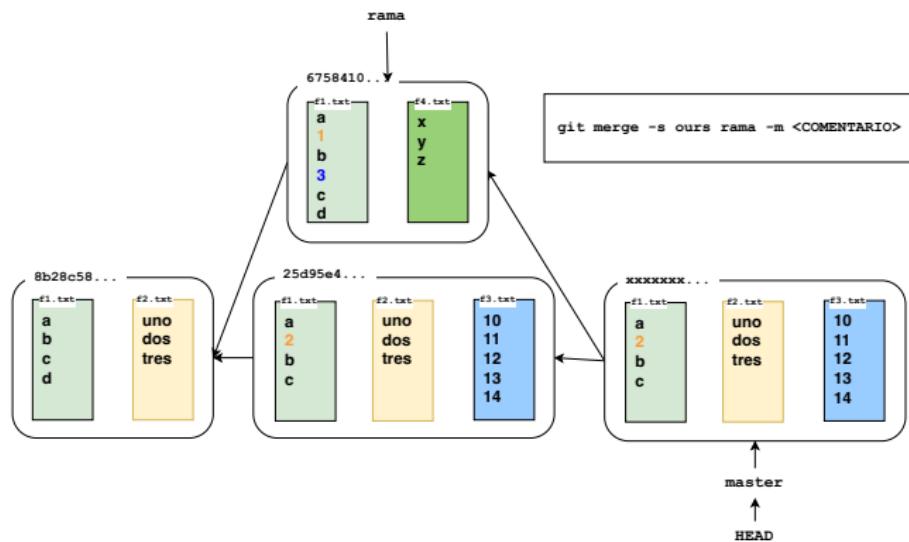


- Estrategia -X theirs



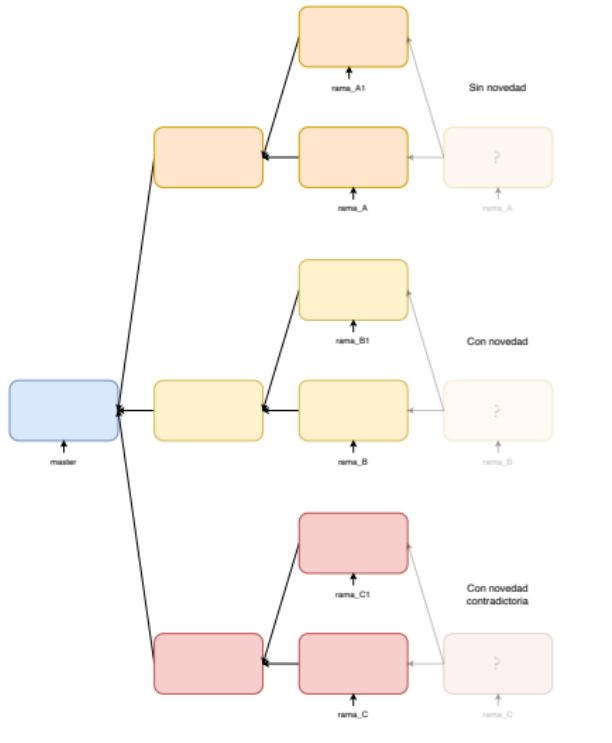
Algunas estrategias de fusión **libres** de conflictos: ejemplos (cont.)

- Estrategia `-s ours`



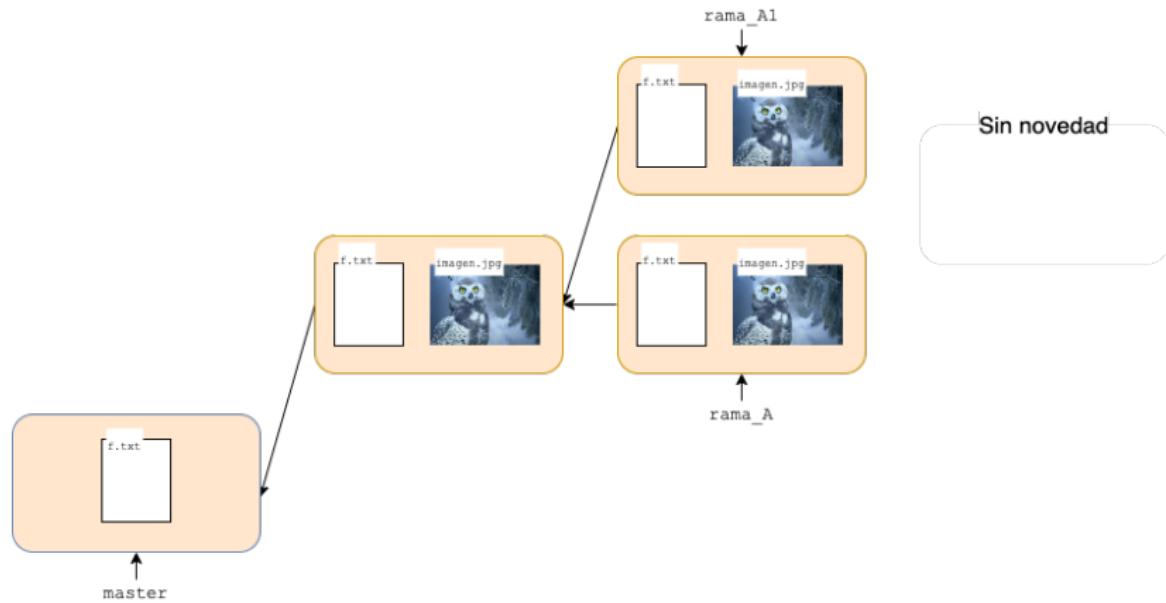
Fusión (elección *automática*) de ficheros binarios

- La fusión de ficheros binarios suele ser compleja (ficheros .jpg, .png, .exe, .dll, .avi, ...):
 - O no tiene sentido o requeriría software especial
 - En git, mecanismos de elección (no de fusión): similar a fusión textual, pero a nivel de fichero, no de línea
 - Respecto al ancestro común más reciente
 - Si no hay cambio en ninguna de las ramas: se mantiene
 - Si hay cambio en alguna rama, se acepta dicha novedad
 - Si hay cambio en ambas ramas: conflicto. A resolver (elegir) manualmente
- En ocasiones, en consideración en ficheros .gitignore

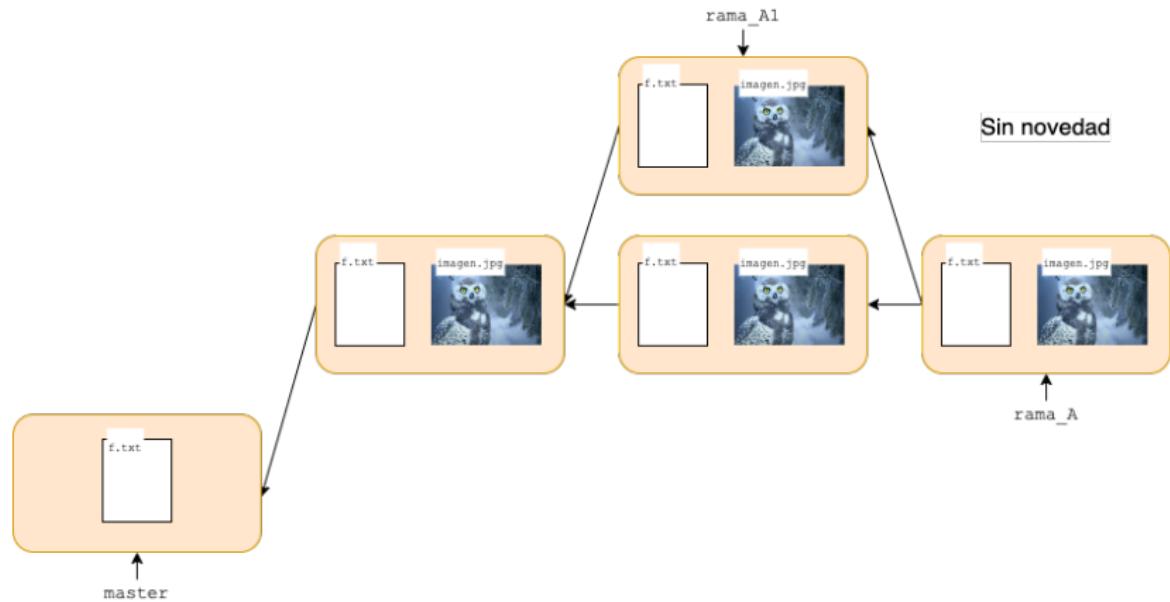


Ejemplo: <https://github.com/GIT-GTDM-24-25/EjemploMergeBinarios.git>

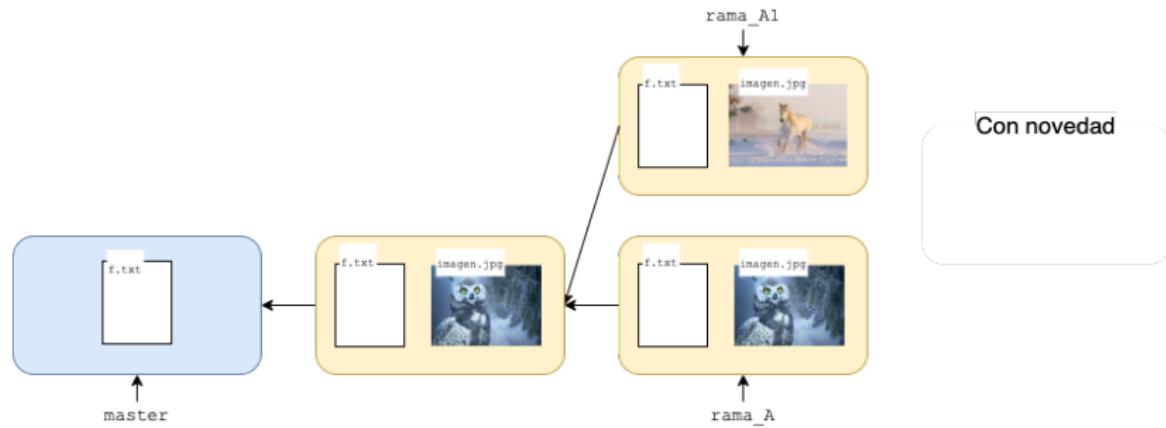
Fusión (elección *automática*) de ficheros binarios: ejemplos



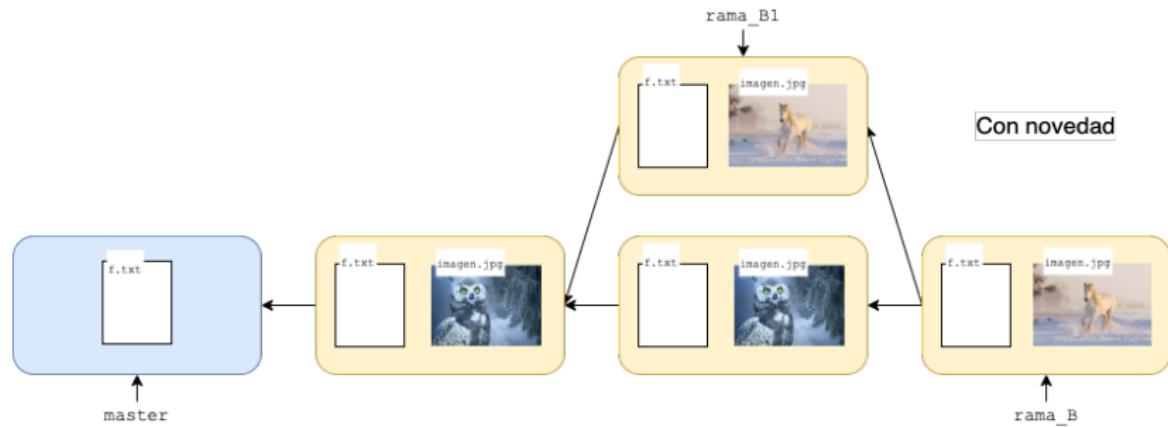
Fusión (elección *automática*) de ficheros binarios: ejemplos



Fusión (elección *automática*) de ficheros binarios: ejemplos



Fusión (elección *automática*) de ficheros binarios: ejemplos



Fusión (elección **manual**) de ficheros binarios

- Recuérdese que para ficheros binarios no hay verdadera fusión porque ni suelen existir interfaces de *mezcla* ni suele tener sentido
 - Si no hay conflicto en la fusión: se elige automáticamente la versión de una de las ramas
 - Si hay conflicto: elección manual de la versión de una de las ramas
 - Para elegir *nuestra* versión (versión de la rama actual):

Elección de fichero binario <FICHERO> de la rama actual

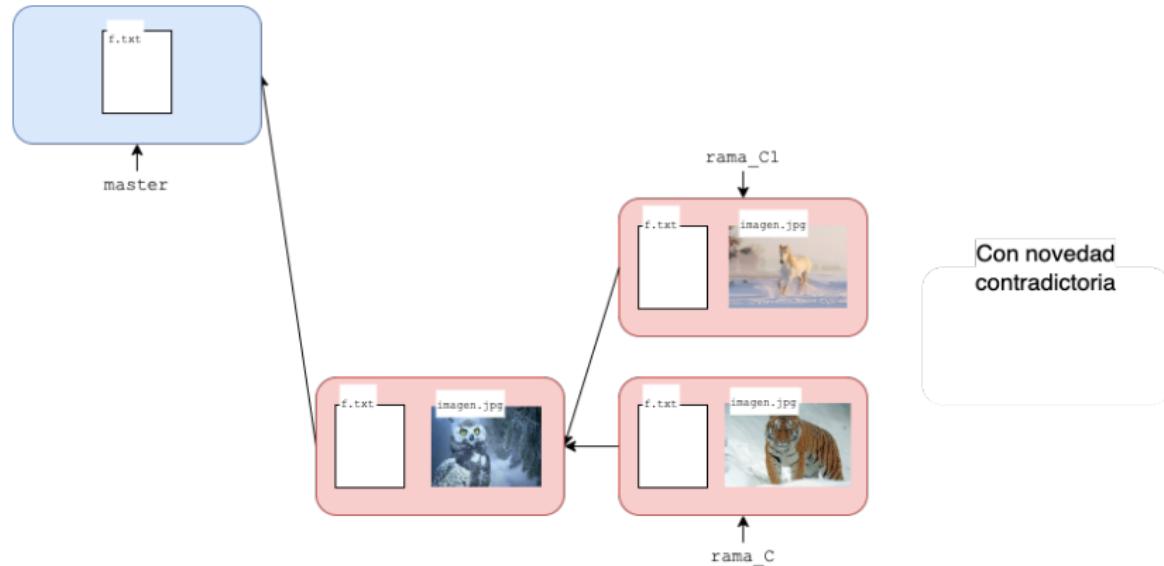
```
$ git checkout --ours <FICHERO>
```

- Para elegir la versión de la *otra* rama:

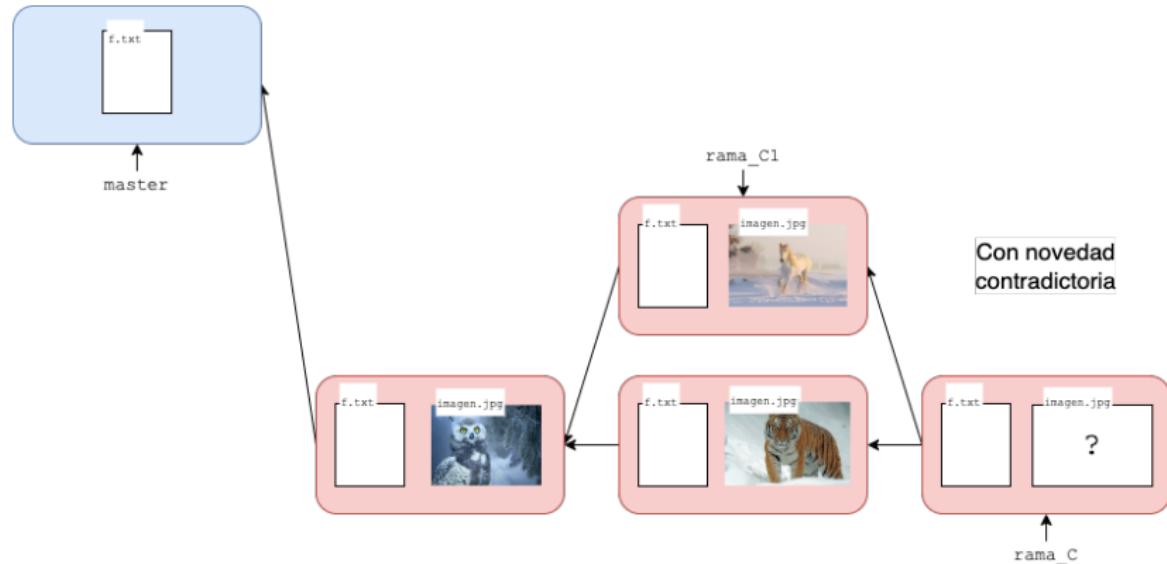
Elección de fichero binario <FICHERO> de la otra rama

```
$ git checkout --theirs <FICHERO>
```

Fusión (elección **manual**) de ficheros binarios: ejemplos



Fusión (elección **manual**) de ficheros binarios: ejemplos



Herramientas de fusión

Uso de herramientas para fusión

```
// Fusión
// <RAMA>=hotfix en el ejemplo gráfico
$ git merge <RAMA> -m <COMENTARIO>

// Resolución de conflicto con herramienta (mergetool)
$ git mergetool

// Seguir indicaciones
$ git status
```

- Configuración de Visual Studio Code como herramienta de fusión `mergetool`:

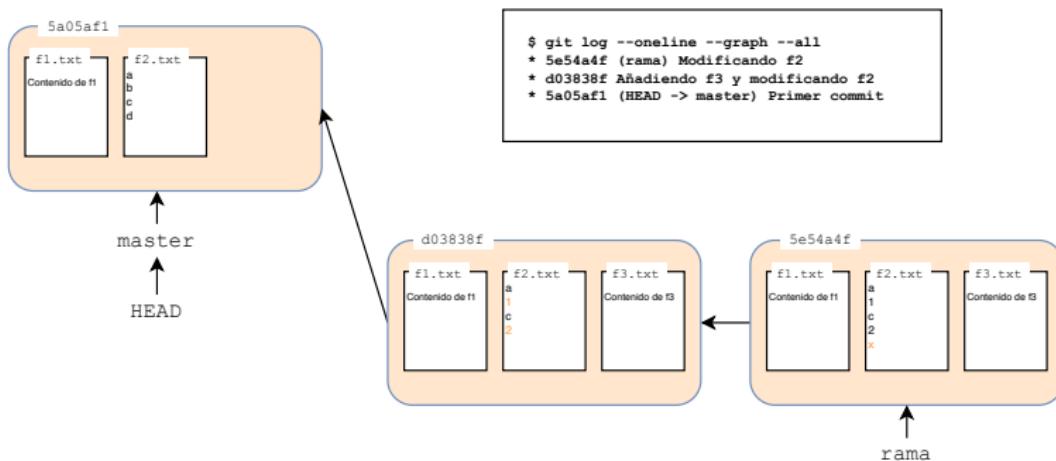
Configuración herramienta de fusión `mergetool`

```
$ git config --global merge.tool vscode
$ git config --global mergetool.vscode.cmd 'code --wait --merge $REMOTE $LOCAL $BASE $MERGED'
```

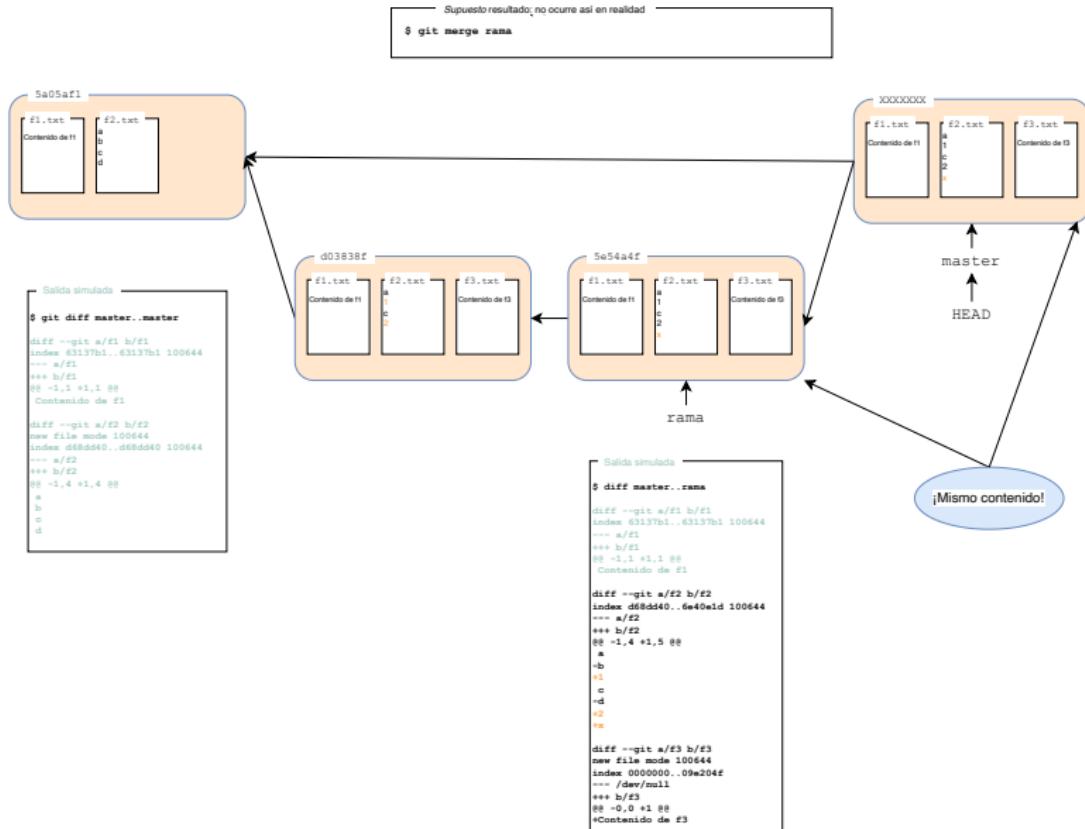
Fusión *fast-forward*

- Fusión: adición de *aportaciones* de una rama en otra
- Fusión *fast-forward*: la aportación de la otra rama es *total*
- IMPORTANTE: elección de rama de referencia
- Ejemplo:

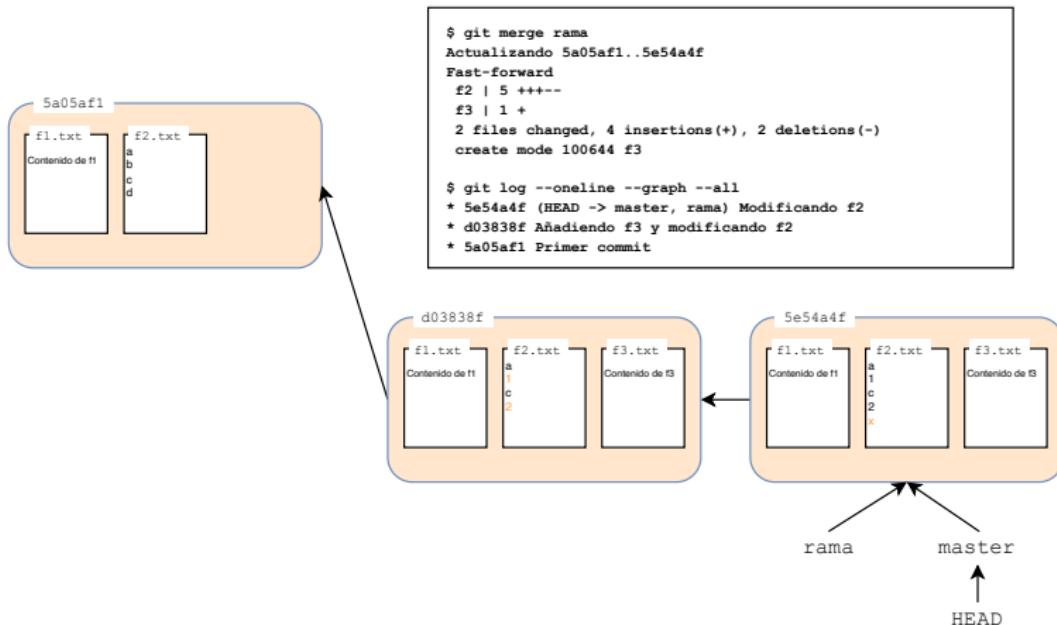
<https://github.com/GIT-GTDM-24-25/EjemploFastForward.git>



Fusión fast-forward (cont.)



Fusión *fast-forward* (cont.)



Fusión *fast-forward*

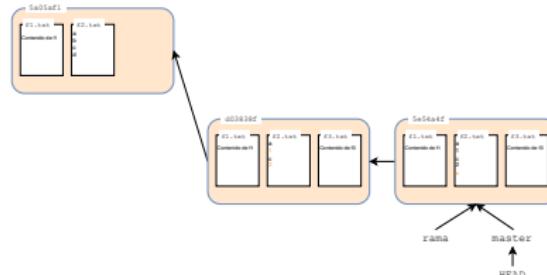
- IMPORTANTE: elección de rama de referencia

```
$ git checkout master
$ git merge rama
Updating ...
Fast-forward
...
```

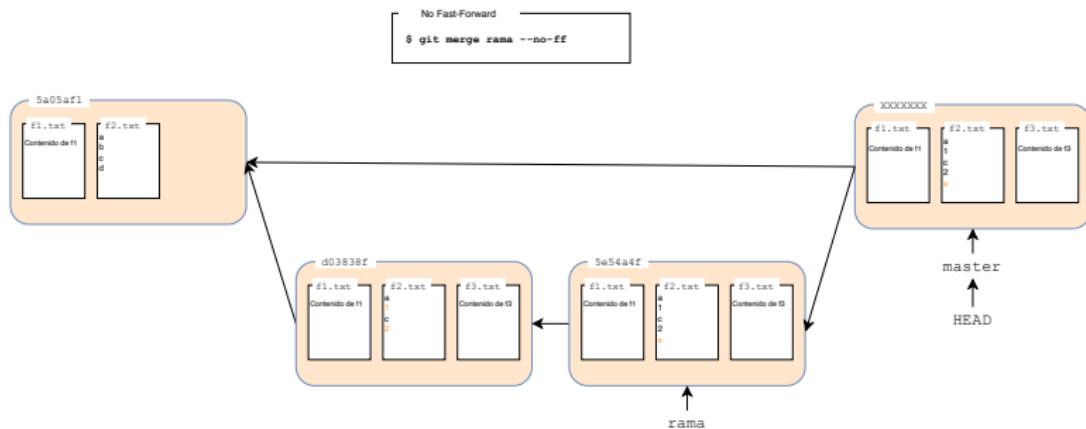
```
$ git checkout rama
# !!! master no aporta nada nuevo en rama !!!
$ git merge master
Already up to date
```

- Deshacer fusión con reset: commit de referencia no será HEAD^1

```
$ git reset --hard <COMMIT_REFERENCIA>
```



Fusión forzada a **no fast-forward**



Stash: almacenamiento temporal

- Guardado provisional de estado en rama sin salvar en repositorio, para posteriormente volver. Aparición automática de rama refs/stash

```
// Por ejemplo, estamos en rama master
$ git checkout master

// Edicion de un fichero
// No se salva el estado de la rama actual en el repositorio
// pero sí en una "pila"
$ git stash

// Por ejemplo se conmuta momentaneamente a otra rama
$ git checkout rama

// Vuelta a rama
$ git checkout master
// Visualización de ficheros: no están tal y como estaban antes de stash.

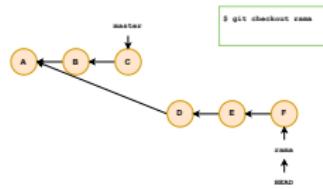
// Lista de "stashes"
$ git stash list

// Aplicación del último stash o de uno en concreto [stash@]
$ git stash apply
// o de uno en concreto [stash@]
$ git stash apply stash@{n}

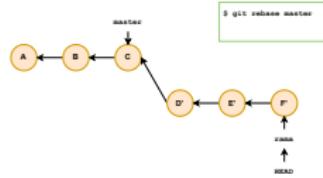
// Eliminación de la rama refs/stash
$ git stash drop
```

Rebase: *linealización* del historial

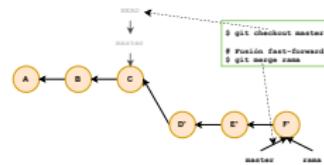
- Reestructuración de *commits* para eliminar ramas y fusiones en historial: alternativa a `merge`
- Situación de partida: cambio a rama a incorporar



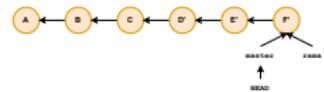
- Rebase de rama de referencia



- Cambio a rama de referencia y fusión *ast-forward*



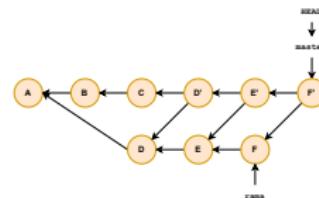
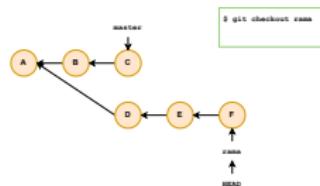
- Linealización:



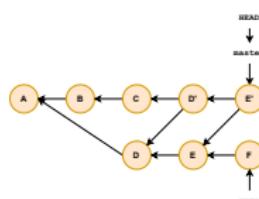
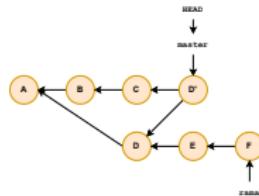
- Rebase también puede utilizarse para *compactar* y *rectificar* *commits*: *squashing*

Rebase: equivalencia

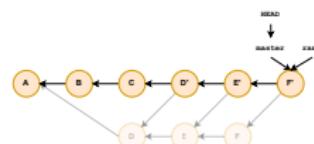
- Situación de partida



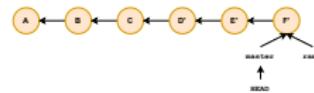
- Secuencia de fusiones



- Eliminación de *commits*



- Linealización equivalente:



Índice

5 Ramas

- Simulador
- Creación y cambio
- Fusión de ramas
- Información y borrado de ramas**
- Sinopsis
- Ejemplo

Información y borrado

- En algunas ocasiones, tras una fusión, la *otra rama* se suele borrar
 - Para la reutilización posterior de su nombre
 - Solo se borran los punteros: ello no implica el borrado de los commits que la conforman (se puede acceder a ellos a través del *segundo* padre: <COMMIT>^{^2})

```
// Lista ramas
$ git branch

// Lista ramas con detalles
$ git branch -v

// Lista las ramas fusionadas
$ git branch --merged

// Lista las ramas no fusionadas
$ git branch --no-merged

// Borra ramas fusionadas (punteros, no commits)
// (puede que ya no se vean con git log)
$ git branch -d rama

// Borra ramas fusionadas y no fusionadas (punteros, no commits)
// (pueden que ya no se vean con git log)
$ git branch -D rama

// Con reflog pueden observarse todos los commits realizados y recuperarlos en su caso,
// si es que sigue siendo posible
```

Índice

5 Ramas

- Simulador
- Creación y cambio
- Fusión de ramas
- Información y borrado de ramas
- **Sinopsis**
- Ejemplo

Resumen

- Con conflictos potenciales *pre-resueltos*

Fusión de <RAMA> en rama actual

```
$ git merge -X ours | -X theirs | -a ours | ... <RAMA> -m <COMENTARIO>
```

- Con conflictos potenciales sin resolver

Fusión de <RAMA> en rama actual

```
$ git merge <RAMA> -m <COMENTARIO>
```

- Si no hay conflictos: commit de fusión generado y añadido a rama de trabajo
- Si hay conflicto en la fusión:
 - Abortar:

Deshacer fusión

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

- Resolver conflictos manualmente. Posibilidades:

- Uso de *mergetool* (debe estar configurado previamente y aparecen ficheros auxiliares)
- Edición de ficheros conflictivos directamente ("<<<<< ===== >>>>>").
- Elección de fichero

Selección manual de fichero íntegro

```
$ git checkout --ours | --theirs <FICHERO>
```

Con todos los conflictos resueltos, se da por finalizada la fusión con

Resolución manual de todos los conflictos

```
# opción -u para actualizar solo los ficheros que han sido editados (modificados)
$ git add -u
$ git merge --continue
```

- Tras una fusión finalizada: vuelta atrás:

Reset a commit anterior -padre 1-: cuidado con fusiones fast-forward

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

Ejercicios de recapitulación

- Ejercicios de Git en W3Schools:
The logo for W3Schools, featuring a stylized green 'W' above the number '3', all contained within a green rectangular box with the word 'schools' written below it.
 - Desde **Git Branch** hasta **Git Branch Merge** inclusive

Índice

5 Ramas

- Simulador
- Creación y cambio
- Fusión de ramas
- Información y borrado de ramas
- Sinopsis
- Ejemplo

Ejemplo

- Conceptos:
 - Creación de ramas
 - Saltos (algún ejemplo con *detached head*)
 - Fusiones *fast-forward*
 - Fusiones *no fast-forward*
 - Fusiones con conflictos

