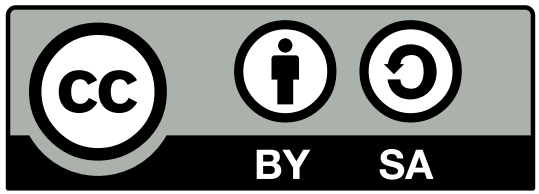
Introducción al Sistema de Control de Versiones Distribuido Git

Antonio García Domínguez

Universidad de Cádiz



14 de febrero de 2011

Contenidos

Materiales en
`http://osl2.uca.es/wikiformacion/index.php/Git` y
`http://gitorious.org/curso-git-osluca`.

Contenidos

Contenidos

¿Por qué usar un SCV?

Copiar ficheros y mandar correos no escala

- ¿Cuál era la última versión?
- ¿Cómo vuelvo a la anterior?
- ¿Cómo reúno mis cambios con los de otro?

SCV: todo ventajas a cambio de alguna disciplina

- Llevamos un historial de los cambios
- Podemos ir trabajando en varias cosas a la vez
- Podemos colaborar con otros
- Hacemos copia de seguridad de todo el historial

Historia de los SCV

Sin red, un desarrollador

- 1972 Source Code Control System
- 1980 Revision Control System

Centralizados

- 1986 Concurrent Version System
- 1999 Subversion («CVS done right»)

Distribuidos

- 2001 Arch, monotone
- 2002 Darcs
- 2005 Git, Mercurial (hg), Bazaar (bzr)

Historia de Git

Antes de BitKeeper

Para desarrollar Linux, se usaban parches y `tar.gz`.

BitKeeper

- 02/2002 BitMover regala licencia BitKeeper (privativo)
- 04/2005 BitMover retira la licencia tras roces

Git

- 04/2005 Linus Torvalds presenta Git, que ya reúne ramas
- 06/2005 Git se usa para gestionar Linux
- 02/2007 Git 1.5.0 es utilizable por mortales
- 12/2010 Última versión: Git 1.7.3.2

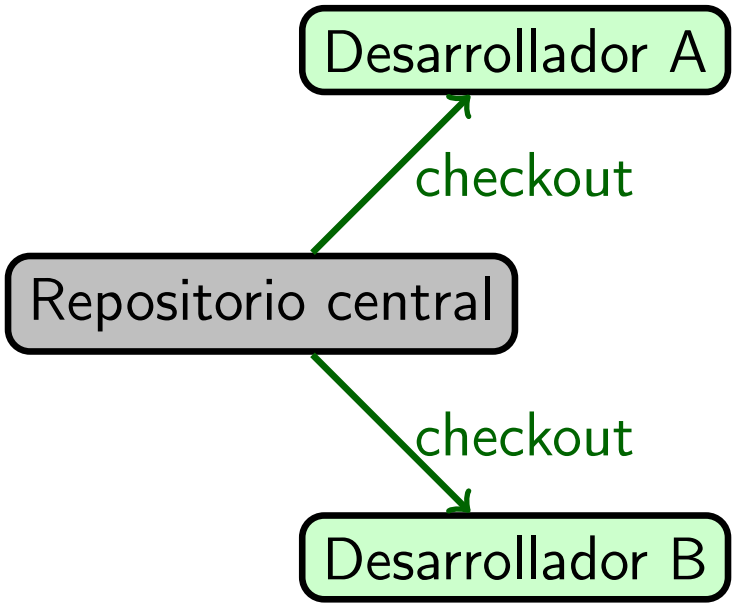
Contenidos

SCV centralizados

Repositorio central

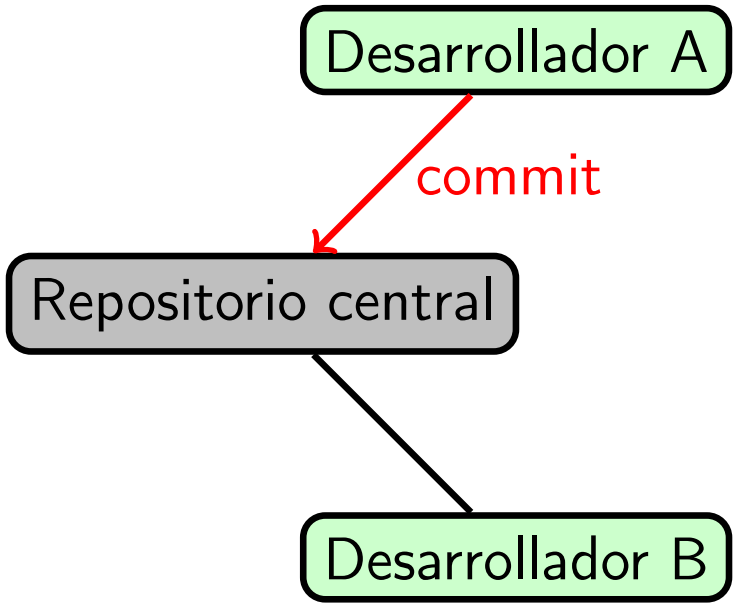
Tenemos nuestro repositorio central con todo dentro.

SCV centralizados



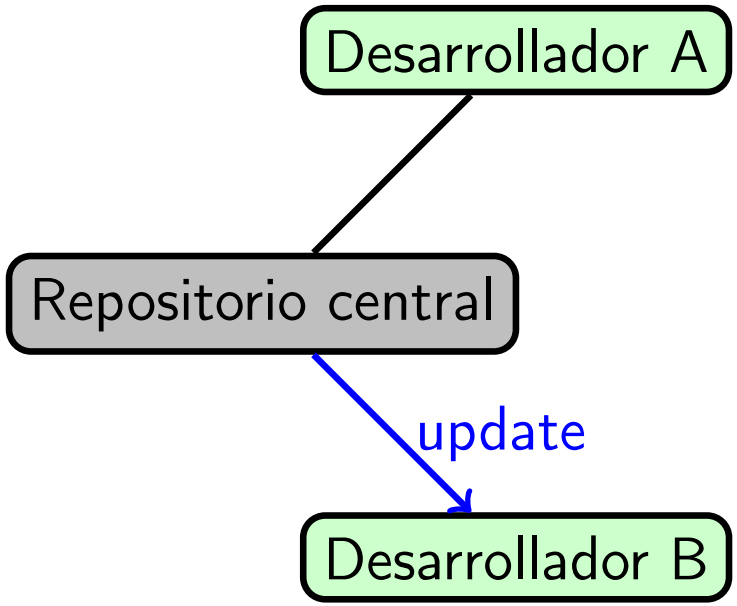
Los desarrolladores crean copias de trabajo.

SCV centralizados



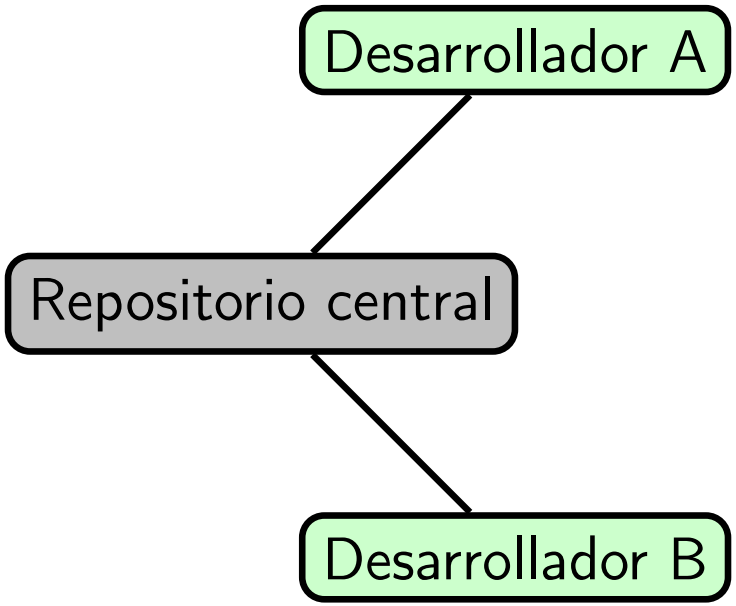
El desarrollador A manda sus cambios al servidor.

SCV centralizados



El desarrollador B los recibe.

SCV centralizados



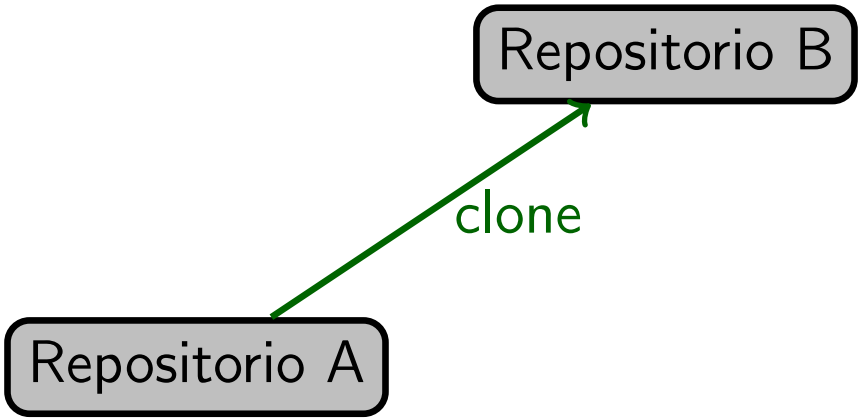
¿Y si se cae el servidor, o la red?

SCV distribuidos

Repositorio A

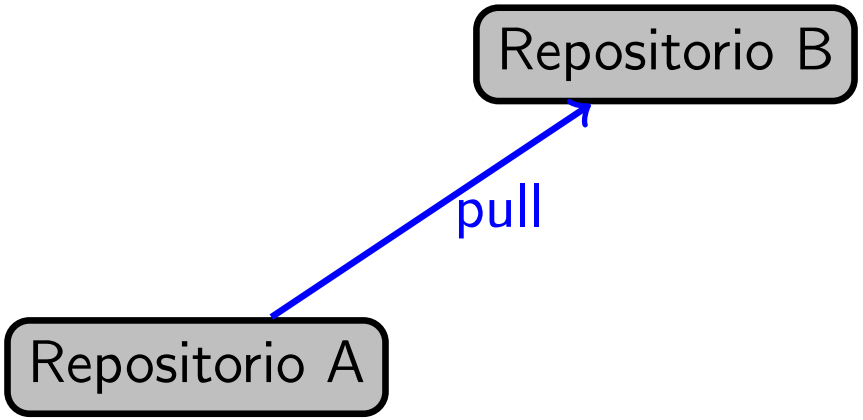
Tenemos nuestro repositorio.

SCV distribuidos



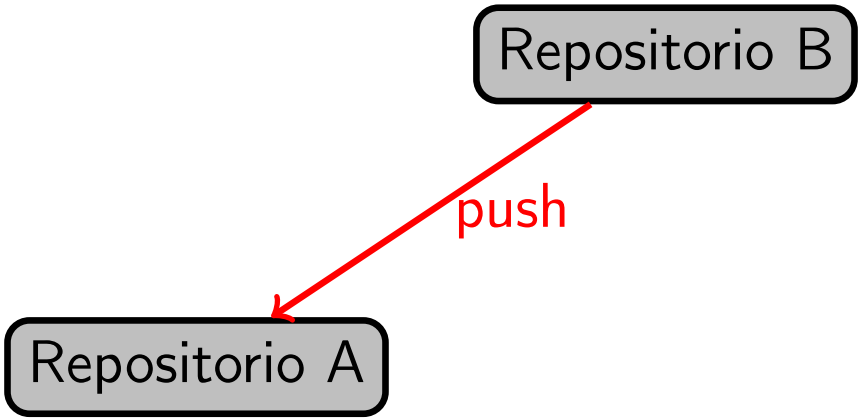
Alguien clona el repositorio.

SCV distribuidos



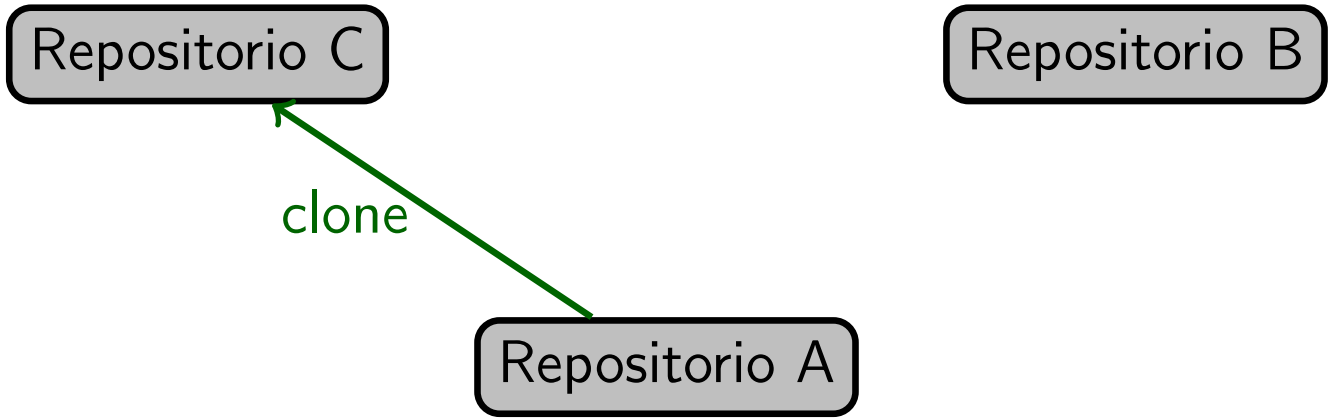
De vez en cuando se trae nuestros cambios recientes.

SCV distribuidos



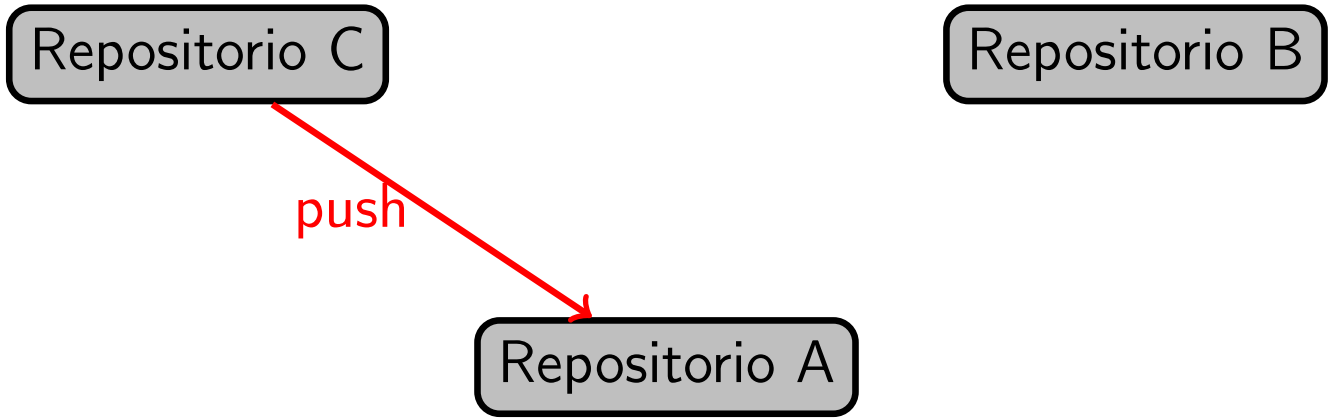
De vez en cuando nos manda sus cambios.

SCV distribuidos



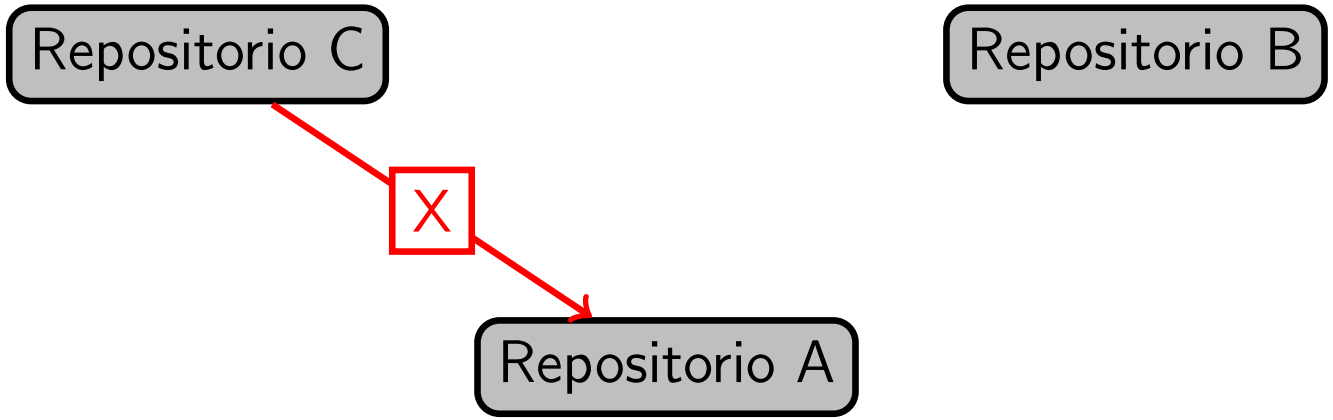
Viene otro desarrollador.

SCV distribuidos



Intenta hacer sus cambios locales...

SCV distribuidos



Pero no le funciona, o no tiene permisos para ello.

SCV distribuidos

Se los pasa al otro desarrollador sin más.

SCV distribuidos

La diferencia entre los repositorios es *social*, no técnica.

Ventajas de un SCV distribuido (I)

Rapidez

- Todo se hace en local: el disco duro es más rápido que la red, y cuando esté todo en caché será más rápido aún
- Clonar un repositorio Git suele tardar *menos* que crear una copia de trabajo de SVN, y ocupa menos

Revisiones pequeñas y sin molestar

- Nadie ve nada nuestro hasta que lo mandamos
- Podemos ir haciendo revisiones pequeñas intermedias
- Sólo mandamos cuando compila y supera las pruebas
- Podemos hacer experimentos de usar y tirar

Ventajas de un SCV distribuido (I)

Rapidez

- Todo se hace en local: el disco duro es más rápido que la red, y cuando esté todo en caché será más rápido aún
- Clonar un repositorio Git suele tardar *menos* que crear una copia de trabajo de SVN, y ocupa menos

Revisiones pequeñas y sin molestar

- Nadie ve nada nuestro hasta que lo mandamos
- Podemos ir haciendo revisiones pequeñas intermedias
- Sólo mandamos cuando compila y supera las pruebas
- Podemos hacer experimentos de usar y tirar

Ventajas de un SCV distribuido (II)

Trabajo sin conexión

- En el tren, avión, autobús, etc.
- Aunque no tengamos permisos de escritura
- Aunque se caiga la red, se puede colaborar

Robustez

Falla el disco duro del repositorio bendito. ¿Qué hacer?

- Centralizado: copias de seguridad
- Distribuido: copias de seguridad y/o colaborar por otros medios

Ventajas de un SCV distribuido (II)

Trabajo sin conexión

- En el tren, avión, autobús, etc.
- Aunque no tengamos permisos de escritura
- Aunque se caiga la red, se puede colaborar

Robustez

Falla el disco duro del repositorio bendito. ¿Qué hacer?

- Centralizado: copias de seguridad
- Distribuido: copias de seguridad y/o colaborar por otros medios

Cuándo NO usar Git

Git no escala ante muchos ficheros binarios

- No sirve para llevar las fotos
- Ni para almacenar vídeos

Git no guarda metadatos

No sirve como sistema de copias de seguridad

Contenidos

Contenidos

Instalación de Git

Ubuntu Linux

- 9.10: descargar paquete de Squeeze (Debian)
- 10.04: instalar `git-*`
- 10.10: instalar `git-all`
- Instalad `tkdiff` (para conflictos) y un buen editor
- Fuentes: guión *install-git.sh* en materiales del curso

Windows

- Usuarios: msysGit (`https://code.google.com/p/msysgit/`)
- Desarrolladores: Cygwin (`http://www.cygwin.com/`)

Configuración inicial

Cambiamos la configuración global en *$HOME/.gitconfig*

Identificación

```
$ git config --global user.name "Mi Nombre"
$ git config --global user.email mi@correo
```

Editor: por defecto Vi/Vim

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

Herramienta para resolver conflictos

```
$ git config --global merge.tool tkdiff
```

Algunos alias útiles

```
$ git config --global alias.ci commit
$ git config --global alias.st status
$ git config --global alias.ai "add -i"
```

Creación de un repositorio

Sólo tenemos que ir a un directorio y decirle a Git que cree un repositorio ahí.

```
$ mkdir ejemplo
$ cd ejemplo
$ git init
Initialized empty Git repository in
/home/antonio/Documentos/curso-git-osluca/transparencias/ejemplo/.git/
```

Nuestras dos primeras revisiones en la rama master

```
$ echo "hola" > f.txt
$ git add f.txt
$ git commit -m "primer commit"
[master (root-commit) 559724e] primer commit
 1 files changed, 1 insertions(+), 0 deletions(-)
 create mode 100644 f.txt
$ echo "adios" >> f.txt
$ git add f.txt
$ git commit -m "segundo commit"
[master 1162f99] segundo commit
 1 files changed, 1 insertions(+), 0 deletions(-)
```

- ¿Qué es ese identificador después de «root-commit»?
- ¿Dónde se guardan mis cosas?
- Usuario de SVN: «¿`git add` dos veces?»

Contenidos

Modelo de datos de Git

Características

- Un repositorio es un grafo orientado acíclico de objetos
- Hay 4 tipos de objetos: *commit*, *tree*, *blob* y *tag*
- Los objetos son direccionables por contenido (resumen SHA1)

Consecuencias del diseño

- Los objetos son inmutables: al cambiar su contenido, cambia su SHA1
- Git *no* gestiona información de ficheros movidos y demás
- Git *nunca* guarda más de un objeto una vez en el DAG, aunque aparezca en muchos sitios

Modelo de datos de Git: revisiones (*commits*)

Contenido

- Fecha, hora, autoría, fuente y un mensaje
- Referencia a revisión padre y a un *tree*

```
$ git cat-file -p HEAD
tree 65de8c1fce51aedbc5b0c838d5d2be0883b3ab0e
parent 559724e046b77e9818b8eaf6b33d7fe25cc0613a
author Antonio <a@b.com> 1297711033 +0100
committer Antonio <a@b.com> 1297711033 +0100

segundo commit
```

Modelo de datos de Git: árboles (*trees*)

Contenido

- Lista de *blobs* y *trees*
- Separa el nombre de un fichero/directorio de su contenido
- Sólo gestiona los bits de ejecución de los ficheros
- No se guardan directorios vacíos

```
$ git cat-file -p HEAD:
100644 blob 9114647dde3052c36811e94668f951f623d8005d f.txt
```

Modelo de datos de Git: ficheros (*blobs*)

Contenido

Secuencias de bytes sin ningún significado particular.

```
$ git cat-file -p HEAD:f.txt
hola
adios
```

Modelo de datos de Git: etiquetas (*tags*)

Contenido

- Referencias simbólicas inmutables a otros objetos
- Normalmente apuntan a *commits*
- Pueden firmarse mediante GnuPG, protegiendo la integridad de todo el historial hasta entonces

```
$ git tag -a v1.0 -m "version 1.0" HEAD
$ git cat-file -p v1.0
object 1162f996ee455e8a1d0ff0801881816c105ab244
type commit
tag v1.0
tagger Antonio <a@b.com> Mon Feb 14 20:17:14 2011 +0100

version 1.0
```

Estructura física de un repositorio Git

Partes de un repositorio Git

- Directorio de trabajo
- Grafo de objetos: *.git*
- Área de preparación: *.git/index*

```
$ ls .git
branches         config       HEAD   index  logs     refs
COMMIT_EDITMSG   description  hooks  info   objects
```

Algunos de los ficheros en *.git*

config

Contiene la configuración local.

```
$ cat config
[core]
repositoryformatversion = 0
filemode = true
bare = false
logallrefupdates = true
```

Algunos de los ficheros en *.git*

description

Descripción corta textual para `gitweb`.

```
$ cat description
Unnamed repository; edit this file 'description' to name the repository.
```

Algunos de los ficheros en *.git*

HEAD

Referencia simbólica a la revisión sobre la que estamos trabajando.

```
$ cat HEAD
ref: refs/heads/master
```

Algunos de los ficheros en *.git*

hooks

Manejadores de eventos. Ahora sólo tenemos ejemplos.

```
$ ls hooks | head -5
applypatch-msg.sample
commit-msg.sample
post-commit.sample
post-receive.sample
post-update.sample
```

Algunos de los ficheros en *.git*

index

Contiene el área de preparación (la veremos después).

```
$ git ls-files -s
100644 9114647dde3052c36811e94668f951f623d8005d 0 f.txt
```

Algunos de los ficheros en *.git*

info/exclude (también .gitignore y/o global)

Patrones de ficheros a ignorar.

```
$ cat info/exclude
# git ls-files --others --exclude-from=.git/info/exclude
# Lines that start with '#' are comments.
# For a project mostly in C, the following would be a good set of
# exclude patterns (uncomment them if you want to use them):
# *.[oa]
# *~
```

Algunos de los ficheros en *.git*

logs

Historial de las referencias: medida de seguridad.

```
$ git reflog
1162f99 HEAD@{0}: commit: segundo commit
559724e HEAD@{1}: commit (initial): primer commit
```

Algunos de los ficheros en *.git*

refs

Referencias simbólicas a puntas de cada rama y etiquetas.

```
$ ls -R refs
ejemplo/.git/refs:
heads  tags

ejemplo/.git/refs/heads:
master

ejemplo/.git/refs/tags:
v1.0
```

Algunos de los ficheros en *.git*

objects

Objetos, sueltos (gzip) o empaquetados (delta + gzip).

```
$ ls objects
11  46  55  5a  5c  65  91  info  pack
$ ls objects/pack
$ git gc
$ ls objects
info  pack
$ ls objects/pack
pack-4e6ce7f87a42762cf66c09f8fef5bc6da78c38cd.idx
pack-4e6ce7f87a42762cf66c09f8fef5bc6da78c38cd.pack
```

Área de preparación, caché o índice

Concepto

Instantánea que vamos construyendo de la siguiente revisión.

Diferencia entre Git y otros SCV

- `svn add` = añadir fichero a control de versiones
- `git add` = añadir contenido a área de preparación

Consecuencias

- Controlamos exactamente qué va en cada revisión
- Algo raro hasta acostumbrarse, pero es *muy* potente

Preparando revisiones (I)

```
$ echo "bueno" >> f.txt
$ git add f.txt
$ echo "malo" >> f.txt
$ echo "nuevo" > g.txt
```

Preparando revisiones (II)

```
$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
# modified:   f.txt
#
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified:   f.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
# g.txt
```

Preparando revisiones (III)

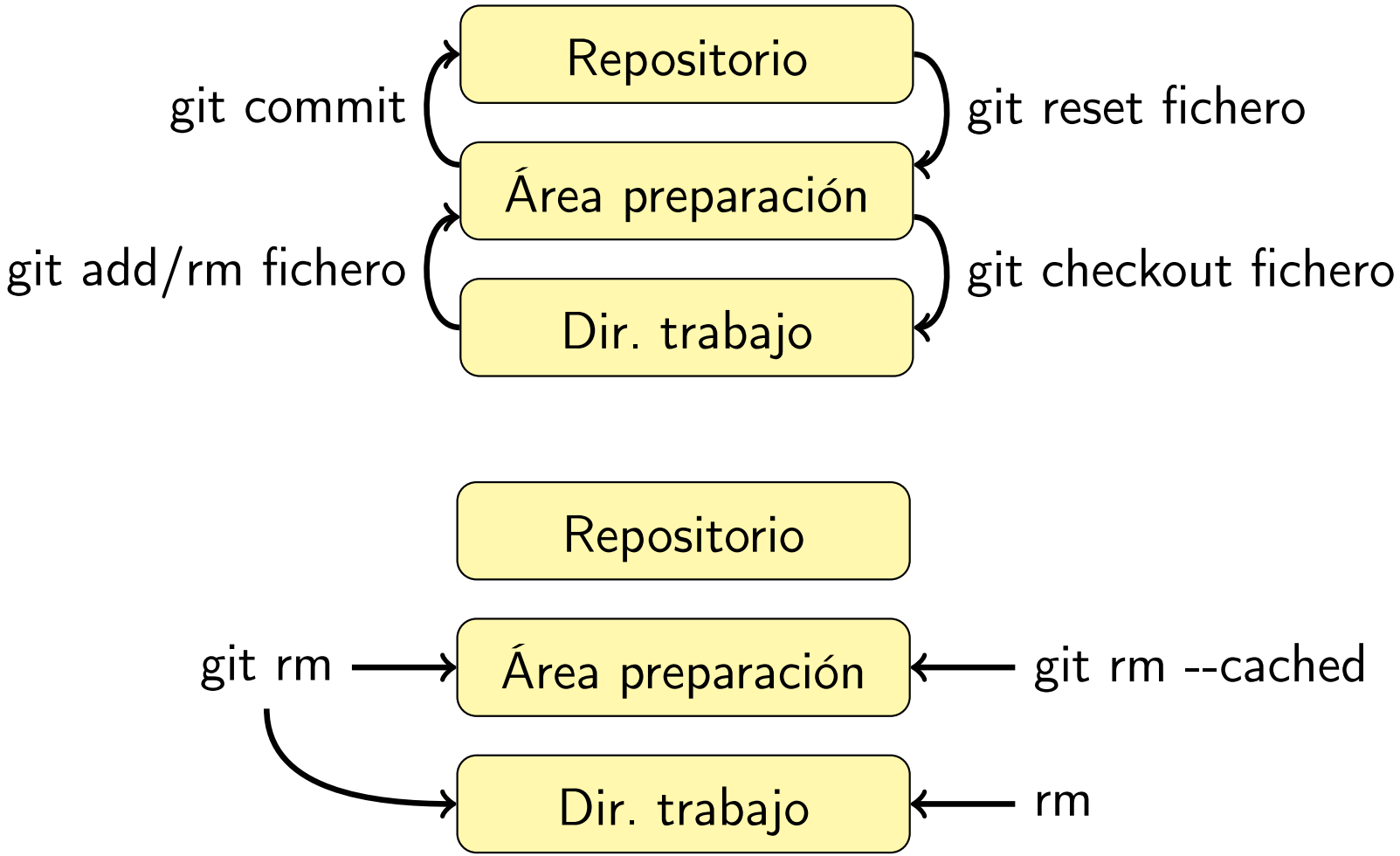
```
$ git diff --staged
diff --git a/f.txt b/f.txt
index 9114647..3d0a14e 100644
--- a/f.txt
+++ b/f.txt
@@ -1,2 +1,3 @@
 hola
 adios
+bueno
```

```
$ git diff
diff --git a/f.txt b/f.txt
index 3d0a14e..ad3ec81 100644
--- a/f.txt
+++ b/f.txt
@@ -1,3 +1,4 @@
 hola
 adios
 bueno
+malo
```

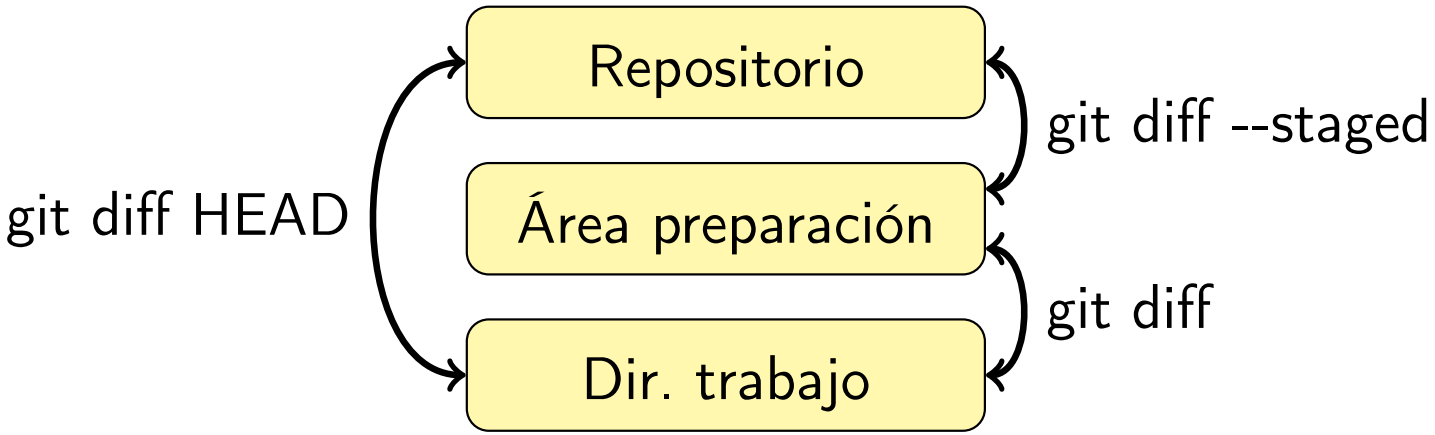
Preparando revisiones (IV)

```
$ git commit -m "tercer commit"
[master d8b704f] tercer commit
 1 files changed, 1 insertions(+), 0 deletions(-)
$ git status
# On branch master
# Changed but not updated:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
# modified:   f.txt
#
# Untracked files:
#   (use "git add <file>..." to include in what will be committed)
#
# g.txt
no changes added to commit (use "git add" and/or "git commit -a")
```

Esquemas de órdenes para añadir y eliminar



Esquema de órdenes para comparar



Contenidos

Buscar cosas

`git grep` es como `grep -R *`, pero sólo busca entre los ficheros bajo control de versiones.

```
$ git grep hola
f.txt:hola
```

Historial: por línea de órdenes (I)

```
$ git log
commit d8b704f8ed5119b18b8d923f4ae8c23764fc891c
Author: Antonio <a@b.com>
Date:   Mon Feb 14 20:17:17 2011 +0100

    tercer commit

commit 1162f996ee455e8a1d0ff0801881816c105ab244
Author: Antonio <a@b.com>
Date:   Mon Feb 14 20:17:13 2011 +0100

    segundo commit

commit 559724e046b77e9818b8eaf6b33d7fe25cc0613a
Author: Antonio <a@b.com>
Date:   Mon Feb 14 20:17:13 2011 +0100

    primer commit
```

Historial: por línea de órdenes (II)

```
$ git log --pretty=oneline
d8b704f8ed5119b18b8d923f4ae8c23764fc891c tercer commit
1162f996ee455e8a1d0ff0801881816c105ab244 segundo commit
559724e046b77e9818b8eaf6b33d7fe25cc0613a primer commit
```

Historial: por línea de órdenes (III)

```
$ git log --graph --pretty=oneline --decorate=short --abbrev-commit
* 7eec99d Transparencias: movido descripción del repositorio más adela...
* 8c90bb1 transparencias.tex: configure UI coloring during repo initia...
* b40fa05 Remove cmd.tmp after running the command, to avoid infinite ...
* b5cb1b3 Transparencias: comenzado a trabajar en nuevas transparencia...
* 1f74aee Añadido PDF del taller de la Quincena de la Ingeniería
*   c550afd Merge branch 'spanish' of gitorious.org:spanish-git-reflec...
|\
| * 4d536a3 (origin/spanish-git-reflections, spanish-git-reflections) ...
* | c7859e7 Añadido guión de instalación de Git
* |   8d33923 Merge branch 'spanish' of gitorious.org:spanish-git-refl...
|\ \
| |/
| * 2769854 Advanced topics: forgot some text at the last line of the ...
* db25cd9 Añadido guión para instalar Git
```

Historial: más cosas

Opciones útiles

- Por autor: `--author`, `--committer`
- Por fecha: `--since`, `--until`
- Por cambio: `-S`
- Por mensaje: `--grep`
- Con parches: `-p` (resumidos: `--stat`)

Otras órdenes

- `git show`: una revisión determinada
- `git whatchanged`: estilo SVN
- `gitk`: interfaz gráfica
- `git instaweb`: interfaz Web (instalad desde fuentes)

Ayuda

Listado de órdenes

- `git help` da un listado breve
- `git help --all` las lista todas (144+)
- Muchas son «fontanería»: sólo usamos la «porcelana»

Sobre una orden concreta

- `man git-orden`
- `git help orden`
- `git help -w orden` (en navegador)
- `git orden -h`

Contenidos

Contenidos

Clonar un repositorio

Métodos de acceso

- `git://`, SSH, HTTP(S) y *rsync*
- Neptuno: «smart HTTP» + SSL (nuevo en Git 1.6.6.2)
- Integrado con autenticación Redmine:
  `http://www.redmine.org/issues/4905`

Autenticación

- Proyecto público, pero requiere autenticación para escritura
- Hay que crear *$HOME/.netrc*, legible sólo por nosotros, con:

```
machine neptuno.uca.es
login miusuario
password micontraseña
```

```
$ git clone https://neptuno.uca.es/git/sandbox-git
Cloning into sandbox-git...
```

Ramas en Git

Concepto: líneas de desarrollo

master	Rama principal, equivalente a trunk
develop	Rama de desarrollo
nueva-cosa	Rama para añadir algo concreto («feature branch»)

Diferencias con otros SCV

- No son apaños con directorios, sino parte del modelo de datos
- Rama en Git: referencia mutable y compartible a una revisión
- Etiqueta en Git: referencia *inmutable* a un objeto

Listando las ramas

Ramas locales

```
$ git branch
* master
```

Ramas remotas

```
$ git branch -r
  origin/HEAD -> origin/master
  origin/ejemplo-conflicto
  origin/ejemplo-heuristicas
  origin/ejemplo-merge-ff
  origin/ejemplo-merge-master
  origin/ejemplo-merge-noff
  origin/ejemplo-rebase-i
  origin/master
```

Gestionando ramas

Crear ramas

```
$ git branch mirama HEAD
$ git branch
* master
  mirama
```

Borrar ramas

```
$ git branch -d mirama
Deleted branch mirama (was 7b40c82).
$ git branch -d master
error: Cannot delete the branch 'master' which you are currently on.
$ git branch
* master
```

Cambiando entre ramas

A una rama local

No confundir con `git checkout -- master`, que copia el fichero *master* del índice al directorio de trabajo.

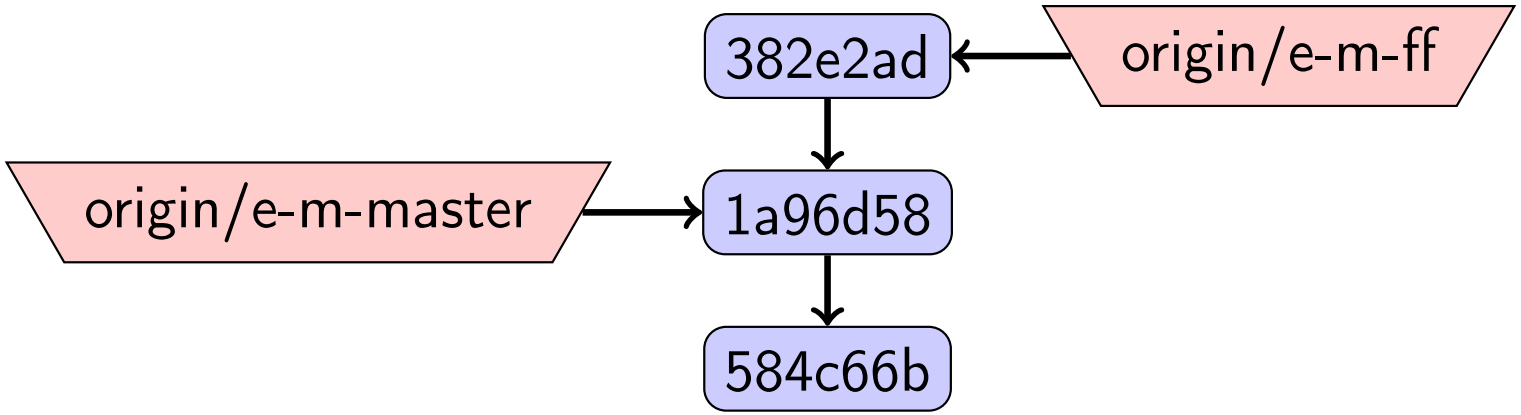
```
$ git checkout master
Already on 'master'
```

A una rama remota

Es de sólo lectura: creamos una rama local que la siga.

```
$ git checkout -b ejemplo-merge-ff origin/ejemplo-merge-ff
Switched to a new branch 'ejemplo-merge-ff'
Branch ejemplo-merge-ff set up to track remote branch
ejemplo-merge-ff from origin.
```

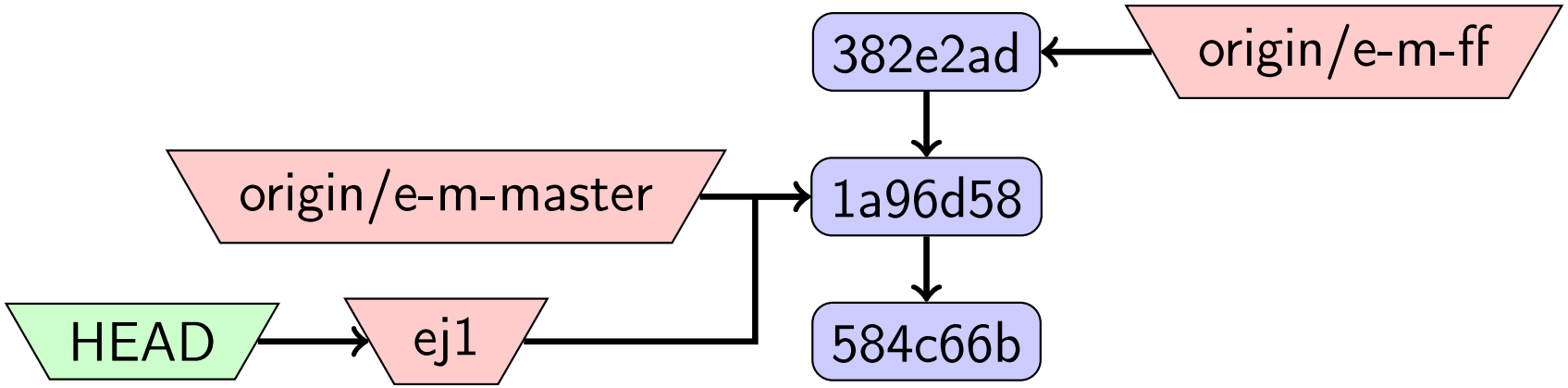
Reuniendo ramas: «fast-forward»



Podemos comprobar cómo están las ramas con:

```
$ gitk origin/ejemplo-merge-master origin/ejemplo-merge-ff
```

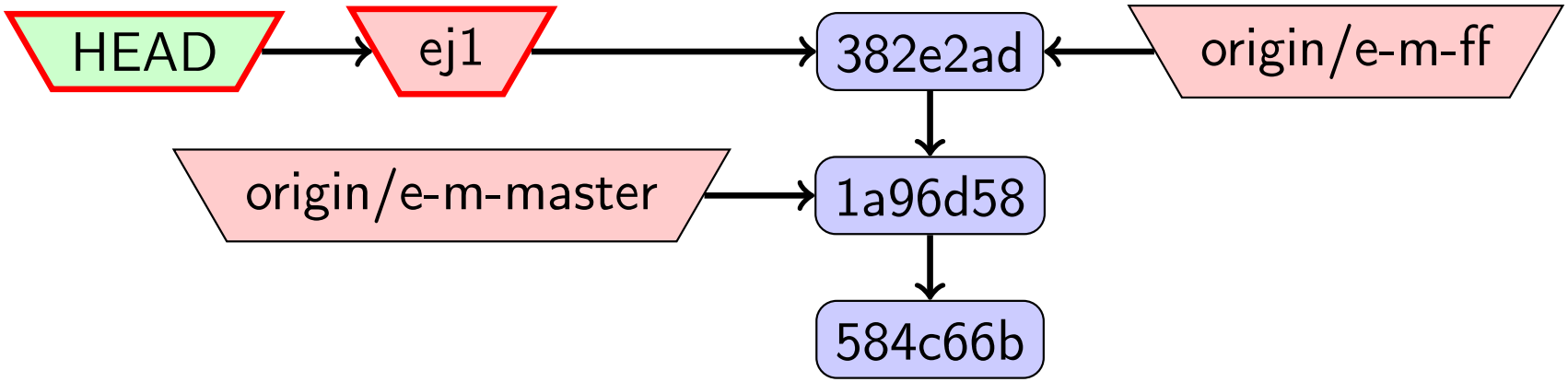
Reuniendo ramas: «fast-forward»



Vamos a crear la rama desde la que haremos la reunión:

```
$ git checkout -b ej1 origin/ejemplo-merge-master
Switched to a new branch 'ej1'
Branch ej1 set up to track remote branch ejemplo-merge-master from
origin.
```

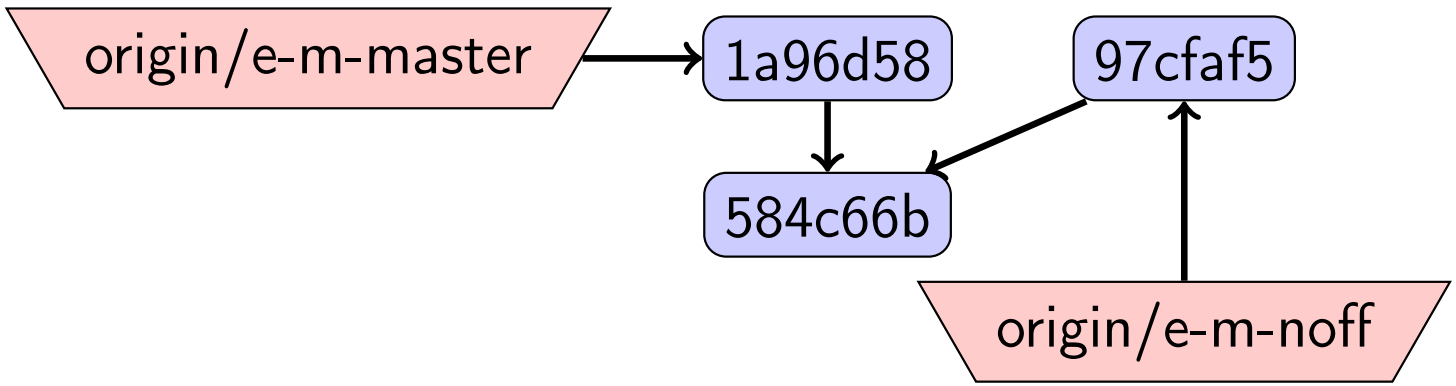
Reuniendo ramas: «fast-forward»



La reunión consiste en adelantar la referencia sin más:

```
$ git merge origin/ejemplo-merge-ff
Updating 1a96d58..382e2ad
Fast-forward
 hola_mundo.c |    2 +-
 1 files changed, 1 insertions(+), 1 deletions(-)
```

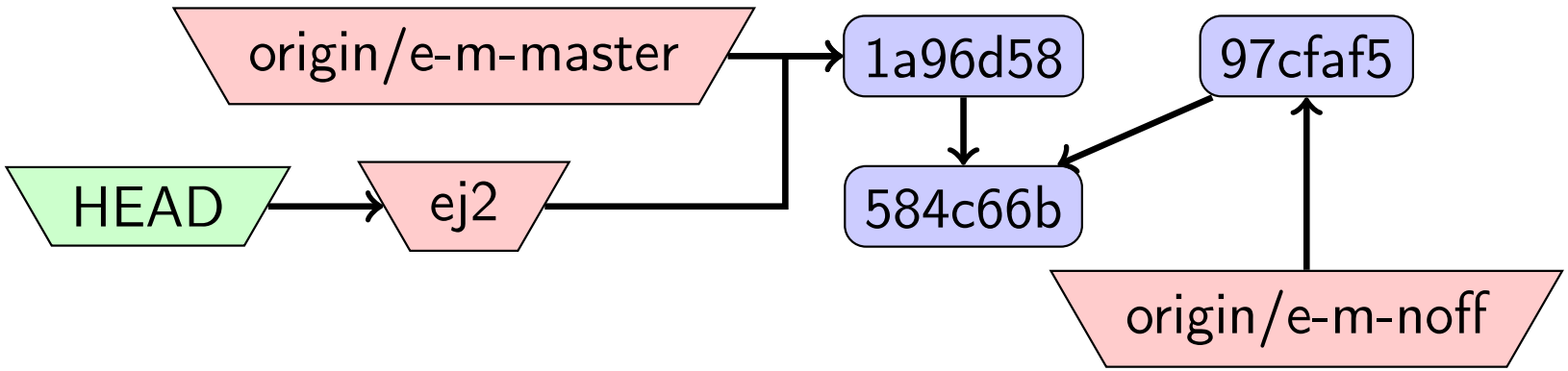
Reuniendo ramas: normal



Otra forma de ver las ramas es con:

```
$ git log --graph --decorate \
origin/ejemplo-merge-master origin/ejemplo-merge-noff
```

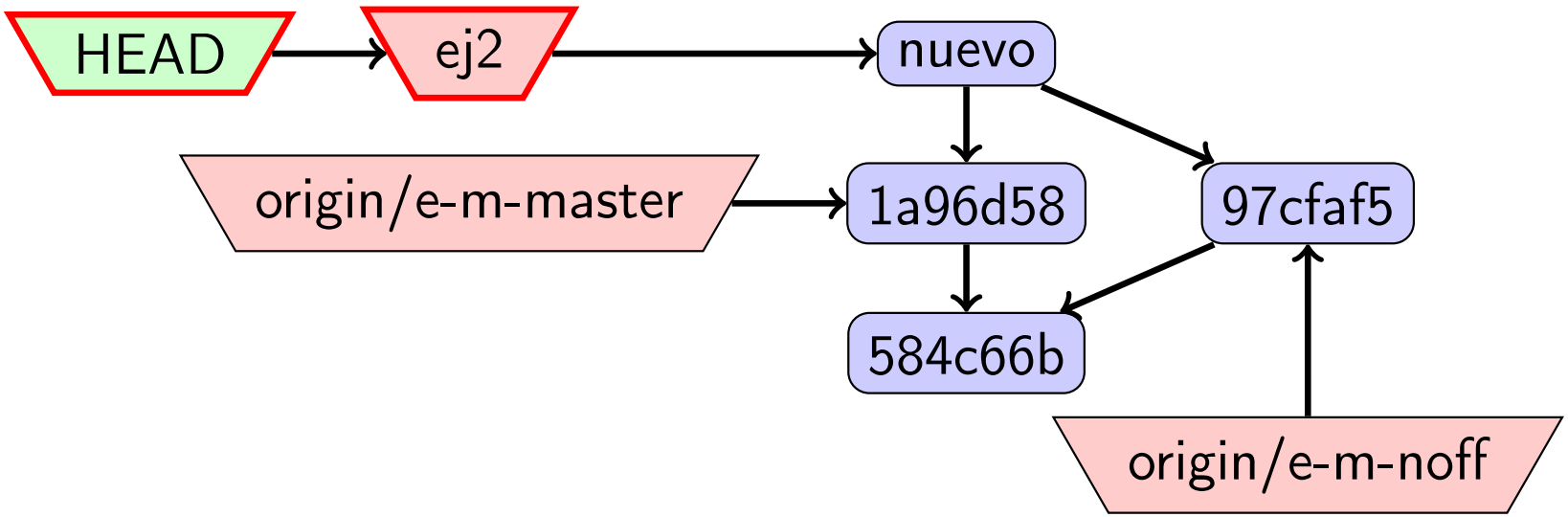
Reuniendo ramas: normal



Creamos otra vez una rama nueva para el punto de partida:

```
$ git checkout -b ej2 origin/ejemplo-merge-master
Switched to a new branch 'ej2'
Branch ej2 set up to track remote branch ejemplo-merge-master from
origin.
```

Reuniendo ramas: normal



La reunión tiene que crear una nueva revisión con dos padres:

```
$ git merge origin/ejemplo-merge-noff
Auto-merging hola_mundo.c
Merge made by recursive.
 hola_mundo.c |    4 ++++
 1 files changed, 4 insertions(+), 0 deletions(-)
```

Reuniendo ramas: conflicto (I)

Miremos el historial de dos ramas, a ver qué cambian:

```
$ gitk origin/ejemplo-merge-master origin/ejemplo-conflicto
```

Vamos a intentar reunirlas:

```
$ git checkout -b ej3 origin/ejemplo-merge-master
Switched to a new branch 'ej3'
Branch ej3 set up to track remote branch ejemplo-merge-master from
origin.
$ git merge origin/ejemplo-conflicto
Recorded preimage for 'hola_mundo.c'
Auto-merging hola_mundo.c
CONFLICT (content): Merge conflict in hola_mundo.c
Automatic merge failed; fix conflicts and then commit the result.
```

Reuniendo ramas: conflicto (II)

Si es texto: `git mergetool`

Lanza herramienta gráfica para resolver todos los conflictos.

Si es binario o no nos gusta `git mergetool` (!)

- `git checkout --ours`: nos quedamos con lo que teníamos
- `git checkout --theirs`: nos quedamos con lo que reunimos
- Editamos los marcadores a mano (como en SVN)
- Después preparamos con `git add` y creamos la revisión con `git commit`, dejando la información acerca del conflicto resuelto en el mensaje

Contenidos

Envío de objetos

En general: `git push URL origen:destino`

- `URL` se puede reemplazar por apodo (origin)
- `origen` es rama o etiqueta local
- `destino` es rama o etiqueta remota
- Actualiza `destino` en rep. remoto a `origen`
- Sólo tiene éxito si es un «fast-forward»

Observaciones

- `git push URL x` = `git push URL x:x`
- `git push URL` actualiza todas las ramas remotas que se llamen igual que las locales
- `git push` es `git push origin`

Recepción de objetos

```
$ git fetch --all
Fetching origin
```

Efecto

Esta orden recibe todos los objetos nuevos de todos los repositorios remotos que conozcamos.

Nota

- Actualiza las ramas remotas
- Seguramente nos interesará traernos sus cambios con `git merge` después
- Como es muy típico, `git pull` combina las dos órdenes: `git fetch` seguido de `git merge`

Flujo de trabajo centralizado

Secuencia típica: muy similar a SVN

- Creamos un clon del repositorio *dorado*
- Nos actualizamos con `git pull`
- Si hay conflictos, los resolvemos
- Hacemos nuestros cambios sin preocuparnos mucho de Git
- Los convertimos en revisiones cohesivas y pequeñas
- Los enviamos con `git push`

Flujos de trabajo distribuidos: 2+ repositorios remotos

```
$ git remote add gitorious \
git://gitorious.org/curso-git-osluca/mainline.git
$ git remote show gitorious
* remote gitorious
  Fetch URL: git://gitorious.org/curso-git-osluca/mainline.git
  Push  URL: git://gitorious.org/curso-git-osluca/mainline.git
  HEAD branch: master
  Remote branches:
    ejemplo-conflicto       new (next fetch will store in remotes/gitorious)
    ejemplo-heuristicas     new (next fetch will store in remotes/gitorious)
    ejemplo-merge-ff        new (next fetch will store in remotes/gitorious)
    ejemplo-merge-master    new (next fetch will store in remotes/gitorious)
    ejemplo-merge-noff      new (next fetch will store in remotes/gitorious)
    ejemplo-rebase-i        new (next fetch will store in remotes/gitorious)
    git-reflections         new (next fetch will store in remotes/gitorious)
    master                  new (next fetch will store in remotes/gitorious)
    spanish-git-reflections new (next fetch will store in remotes/gitorious)
  Local refs configured for 'git push':
    ejemplo-merge-ff pushes to ejemplo-merge-ff (up to date)
    master           pushes to master           (local out of date)
$ git remote rm gitorious
```

Flujos de trabajo distribuidos: variantes

Un integrador

- Cada desarrollador tiene rep. privado y rep. público
- Los desarrolladores colaboran entre sí
- El integrador accede a sus rep. públicos y actualiza el repositorio oficial
- Del repositorio oficial salen los binarios
- Los desarrolladores se actualizan periódicamente al oficial

Director y tenientes

- El integrador (dictador) es un cuello de botella
- Se ponen intermediarios dedicados a un subsistema (teniente)

Forjas con alojamiento Git

Sencillas y libres

- Gitorious: `http://gitorious.org`
- `http://repo.or.cz`

La más popular: Github (`http://github.com`)

- Gratis para proyectos libres
- De pago para proyectos cerrados
- Integra aspectos sociales y nuevas funcionalidades
- Basada en JGit, una reimplementación de Git en Java

Interoperar con SVN

Limitaciones

- No funciona con repositorios vacíos (al menos una revisión)
- Hay que llamar a `git svn` para ciertas cosas

Órdenes básicas

- `git svn clone URL` clona (usar `-s` si se sigue el esquema branches/tags/trunk usual)
- `git svn rebase` = `svn up` + replantea nuestros *commits* locales en base a los nuevos
- `git svn dcommit` = `svn commit` en bloque de todo lo que tengamos pendiente (con `--rmdir` borra directorios vacíos)

Usemos `https://neptuno.uca.es/svn/sandbox-svn-git`.

Aspectos avanzados

- `bisect`: búsqueda binaria de defectos
- `blame`: autoría por líneas
- `bundle`: colaborar sin red
- `clean`: retirar ficheros fuera de control de versiones
- `daemon`: acceso eficiente sin autenticación
- `format-patch`: preparar parches para enviar por correo
- `rebase -i`: navaja suiza para reorganizar ramas
- `rebase`: replantear ramas en base a otras
- `reflog`: historial de referencias para recuperación
- `stash`: aparcar cambios en zona temporal
- `submodule`: incluir repositorios externos

Fin de la presentación

¡Gracias por su atención!

antonio.garciadominguez@uca.es