Introducción al Sistema de Control de Versiones Distribuido Git

Antonio García Domínguez

Universidad de Cádiz

2 de diciembre de 2010

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

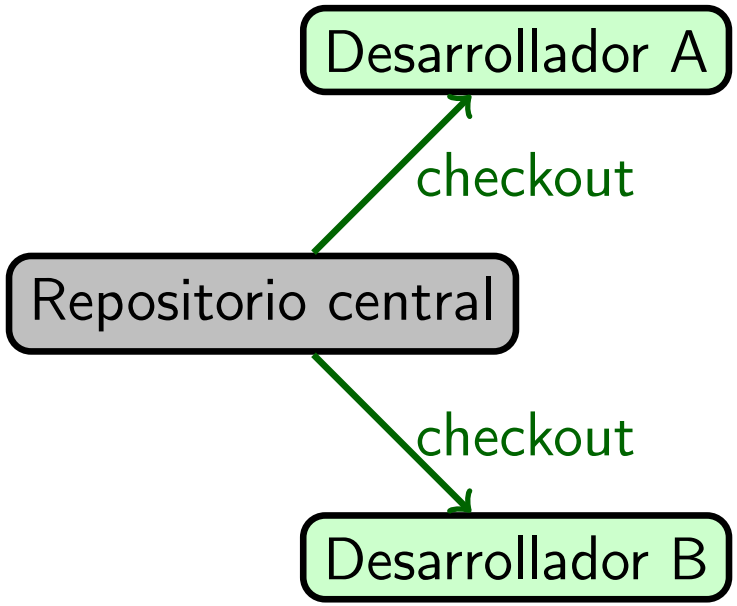
Git

- 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

SCV centralizados

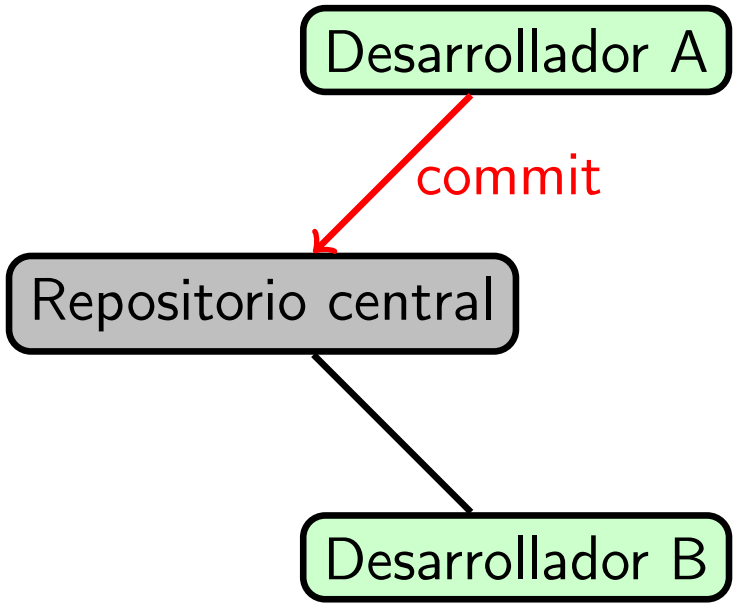
Tenemos nuestro repositorio central con todo dentro.

SCV centralizados



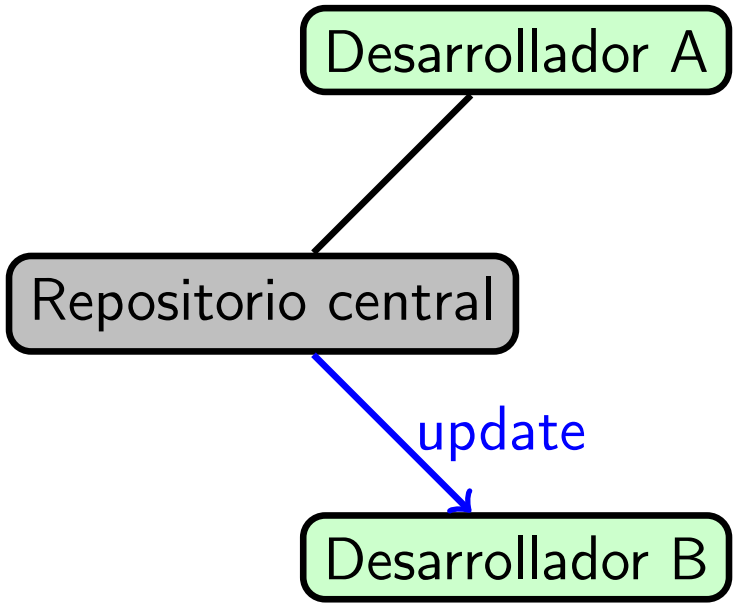
Los desarrolladores crean copias de trabajo.

SCV centralizados



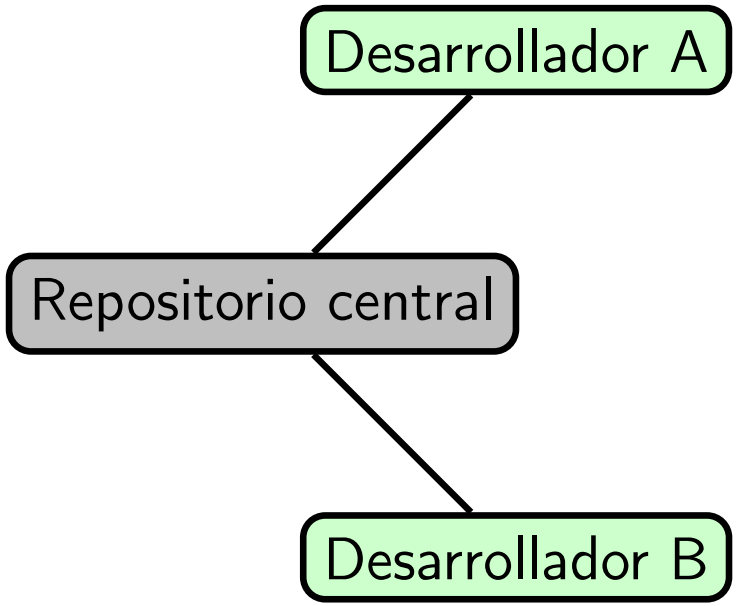
El desarrollador A manda sus cambios al servidor.

SCV centralizados



El desarrollador B los recibe.

SCV centralizados

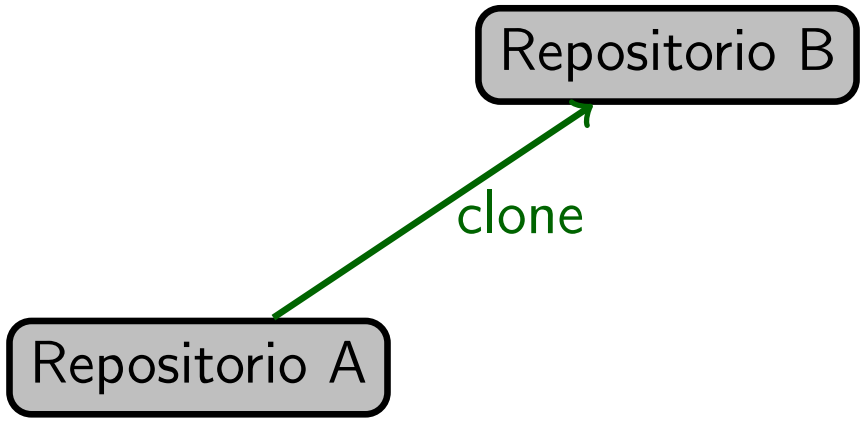


¿Y si se cae el servidor, o la red?

SCV distribuidos

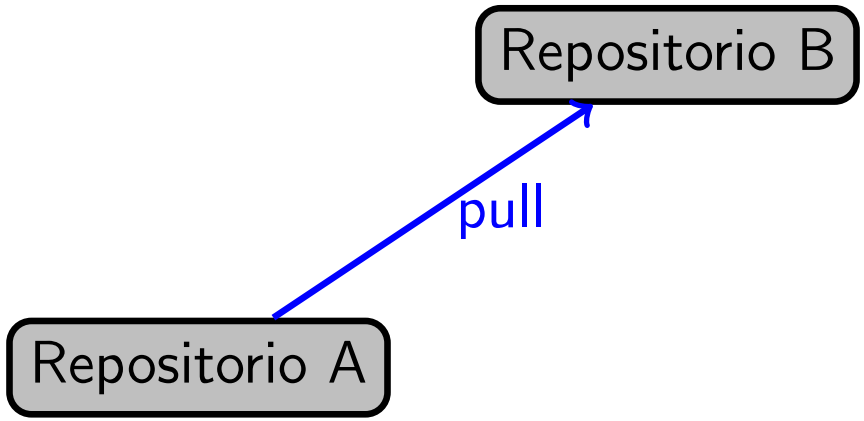
Tenemos nuestro repositorio.

SCV distribuidos



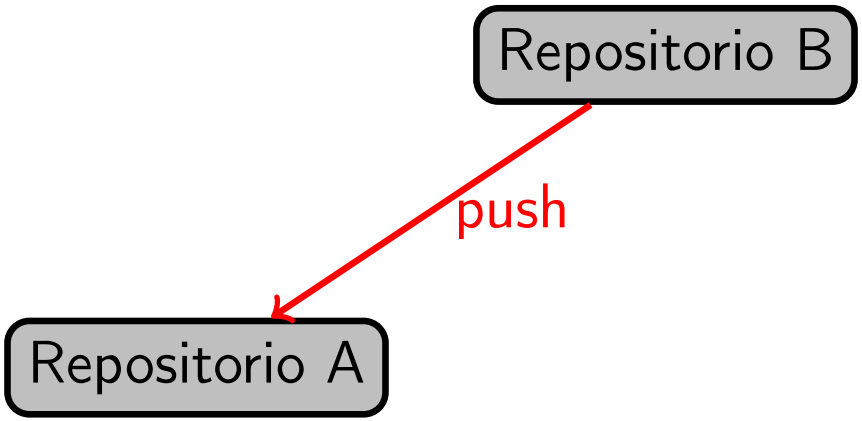
Alguien clona el repositorio.

SCV distribuidos



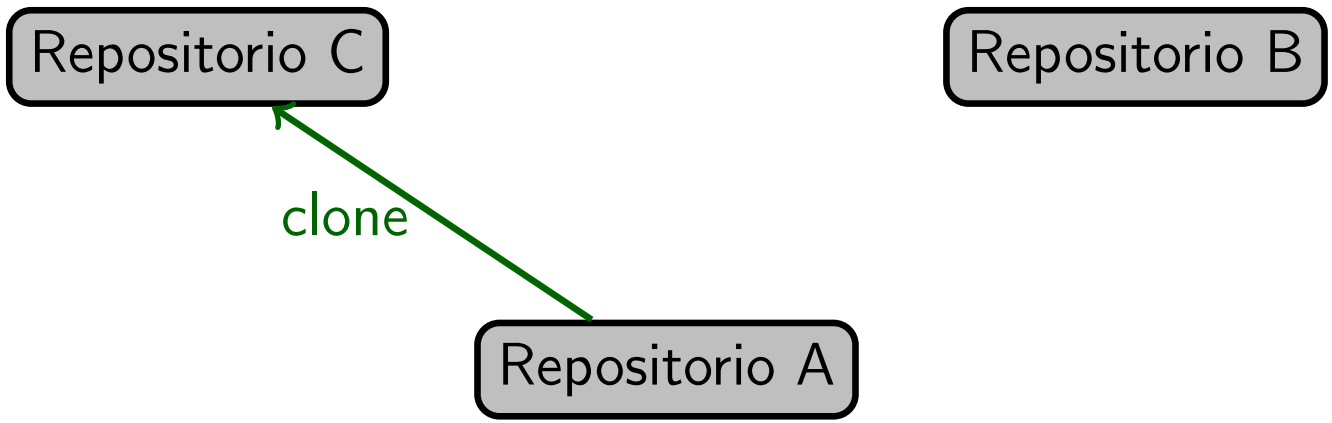
De vez en cuando se trae nuestros cambios recientes.

SCV distribuidos



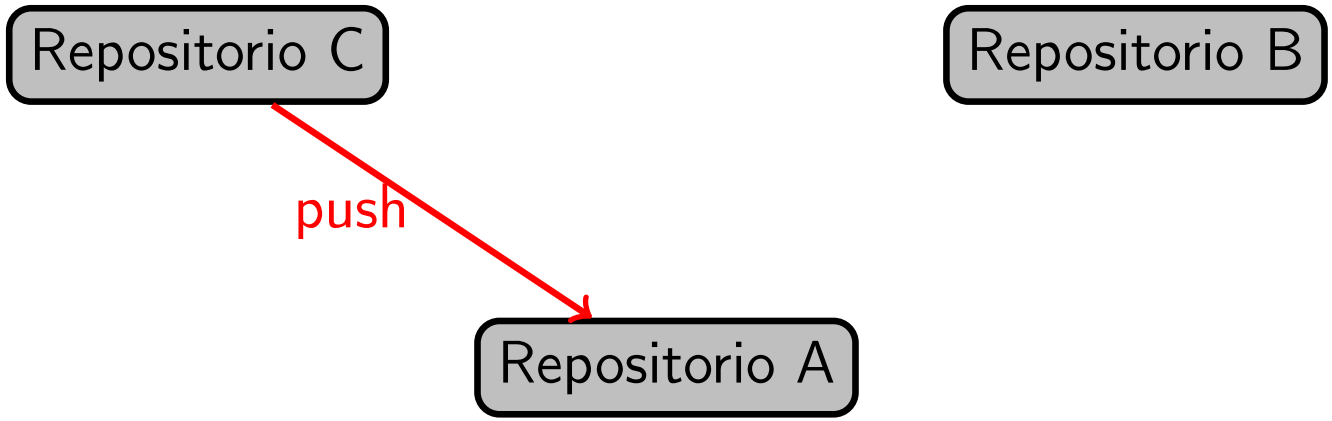
De vez en cuando nos manda sus cambios.

SCV distribuidos



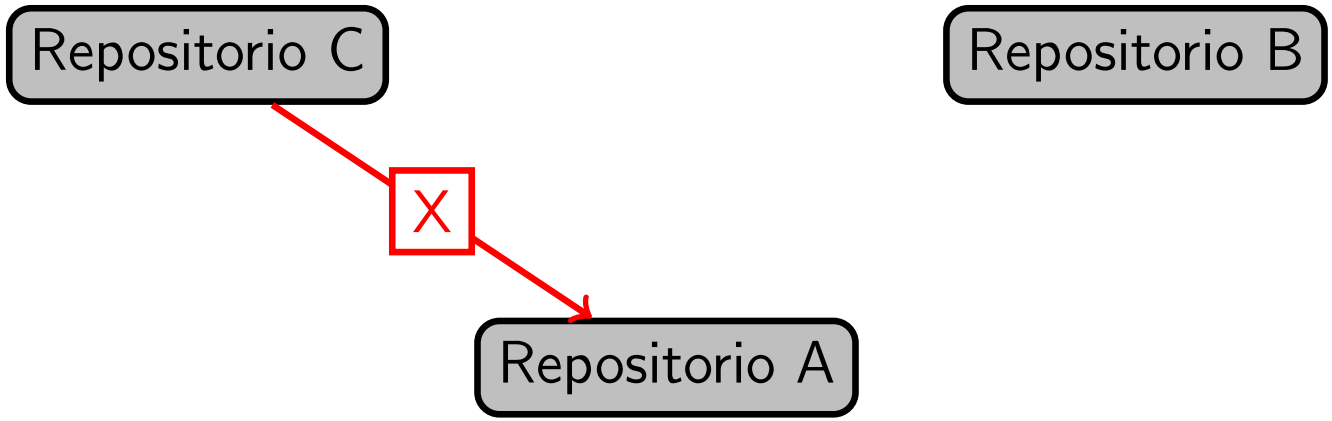
Viene otro desarrollador.

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

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
```

Contenidos

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
```

Estructura de un repositorio

Un repositorio Git no es más que un directorio con un par de ficheros. La mayoría están en texto plano.

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

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 rep
```

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*

info/exclude

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 o
# exclude patterns (uncomment them if you want to use them):
# *.[oa]
# *~
```

Algunos de los ficheros en *.git*

objects

Objetos del repositorio, en formato suelto justo dentro de *objects* o empaquetado en *pack*. Ahora mismo no hay.

```
$ ls -r objects
pack
info
```

Algunos de los ficheros en *.git*

refs

Referencias simbólicas a las puntas de cada rama y a las etiquetas privadas. Ahora mismo no hay.

```
$ ls -r refs
tags
heads
```

Nuestras primeras revisiones

Vamos a crear las 2 primeras revisiones de la rama master:

```
$ echo "hola" > f.txt
$ git add f.txt
$ git commit -m "primer commit"
[master (root-commit) ddbd81f] 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 e6f6259] segundo commit
 1 files changed, 1 insertions(+), 0 deletions(-)
```

- ¿Qué es ese número extraño después de «root-commit»?
- ¿Por qué hemos hecho `git add` dos veces?

Modelo de datos de Git

Idea central

4 tipos de objetos direccionables por contenido (resumen SHA1)

Revisiones (*commits*)

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

```
$ git cat-file -p HEAD
tree 65de8c1fce51aedbc5b0c838d5d2be0883b3ab0e
parent ddbd81f721df4983cd0b66bda66e719fcdcdb1b2
author Antonio <a@b.com> 1291249900 +0100
committer Antonio <a@b.com> 1291249900 +0100

segundo commit
```

Modelo de datos de Git

Idea central

4 tipos de objetos direccionables por contenido (resumen SHA1)

Árboles (*trees*)

Lista *blobs* y *trees*, dándoles nombres

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

Modelo de datos de Git

Idea central

4 tipos de objetos direccionables por contenido (resumen SHA1)

Ficheros normales (*blobs*)

Secuencias de bytes sin ningún significado particular

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

Modelo de datos de Git

Idea central

4 tipos de objetos direccionables por contenido (resumen SHA1)

Etiquetas (*tags*)

Referencias simbólicas inmutables a objetos (p.ej. *commits*)

```
$ git tag -a v1.0 -m "version 1.0" HEAD
$ git cat-file -p v1.0
object e6f6259d0b7e05d5c9ec8d7ee4c022661001704d
type commit
tag v1.0
tagger Antonio <a@b.com> Thu Dec 2 01:31:40 2010 +0100

version 1.0
```

Área de preparación

Conseguir ayuda

Contenidos

Trabajar con Git

Contenidos

Interoperar con SVN

Colaborando entre varios repositorios

Esculpir revisiones con add -i

Replantear ramas con rebase

Reorganizar ramas con rebase -i

Fin de la presentación

¡Gracias por su atención!

antonio.garciadominguez@uca.es