

Tema: II

Control de versiones

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Alicante

Curso 2014-2015, Copyleft © 2011-2015.
Reproducción permitida bajo los términos de la licencia de
documentación libre GNU.



1 / 33

Contenido

- 1 Introducción
- 2 ¿En qué consiste el control de versiones?
- 3 Clasificaciones de los scv
- 4 Conceptos generales de los scv
- 5 Un poco de historia
- 6 Git: Historia
- 7 Git: Implementación
- 8 Git: Directorio '.git'
- 9 Uso (I)
- 10 Uso (II)
- 11 Uso (III)
- 12 Uso (IV)
- 13 Uso (V)
- 14 Uso (VI)
- 15 Uso (VII)
- 16 Uso (VIII)
- 17 Uso (IX)
- 18 Uso (X)
- 19 Instalación
- 20 Interacción con cvs y svn
- 21 Casos de uso
- 22 ¿Sería posible un tutorial interactivo?
- 23 Un ejemplo sencillo paso a paso
- 24 Webs de interés



2 / 33

El Control de versiones en la práctica

Veamos con un ejemplo sencillo la utilidad del control de versiones:

Supongamos la siguiente situación en la que hemos escrito este código ↓

```
1  /*
2  * Hola mundo.
3  * fecha: 27/12/2009
4  */
5
6  #include <stdio.h>
7  int main(int argc, char* argv[])
8  {
9      puts("Hola mundo!");
10 }
```

Posteriormente hacemos una serie de cambios en él ↓

```
/*
 * Hola mundo.
 * fecha: 22/01/2010
 */

#include <stdio.h>
int main(int argc, char** argv) {
    printf("Hola mundo!");
}
```



3 / 33

El Control de versiones en la práctica

Con los cambios realizados podemos:

- Conservar sólo la última versión del archivo.
- Mantener la versión anterior por si los cambios introducen algún fallo¹.
- Conocer quién los realizó (caso de trabajar en grupo).
- Deshacerlos para recuperar la version anterior (en caso de haberla perdido).
- Aislarlos para enviárselos a otro desarrollador para que los aplique a la version del archivo que el tiene (*parche*).

Pero todo esto...se puede hacer manualmente!

¹¿Y la anterior de la anterior?...



4 / 33

Por tanto, ¿qué nos aportan los sistemas de control de versiones²?:

- La gestión automática de los cambios que se realizan sobre uno o varios ficheros de un proyecto.
 - Restaurar cada uno de los ficheros de un proyecto a un estado de los anteriores por los que ha ido pasando (no solo al inmediatamente anterior).
 - Permitir la colaboración de diversos programadores en el desarrollo de un proyecto.
- Por la forma de almacenar los contenidos:
 - 1 Centralizados
 - 2 Distribuidos
 - Por la manera en la que permiten que cada desarrollador pueda modificar la copia local de los datos extraídos del repositorio:
 - 1 Colaborativos
 - 2 Exclusivos

²En adelante scv .

Conceptos generales de los scv (I)

- **Repositorio** Es la copia maestra donde se guardan todas las versiones de los archivos de un proyecto. En el caso de git se trata de un directorio. Cada desarrollador tiene su propia copia local de este directorio.
- **Copia de trabajo** La copia de los ficheros del proyecto que podemos modificar.

Conceptos generales de los scv (II)

- **Check Out / Clone** La acción empleada para obtener una copia de trabajo desde el repositorio. En los scv distribuidos -como Git- esta operación se conoce como **clonar** el repositorio por que, además de la copia de trabajo, proporciona a cada programador su copia local del repositorio a partir de la *copia maestra* del mismo.
- **Check In / Commit** La acción empleada para llevar los cambios hechos en la copia de trabajo a la copia local del repositorio³. Esto crea una nueva revisión de los archivos modificados. Cada 'commit' debe ir acompañado de un **Log Message** el cual es un comentario⁴ que añadimos a una revisión cuando hacemos el commit oportuno.
- **Push** La acción que traslada los contenidos de la copia local del repositorio de un programador a la copia maestra del mismo.

³Check In.

⁴Una cadena de texto que explica el commit.

- **Update/Pull/Fetch+Merge/Rebase** Acción empleada para actualizar nuestra copia local del repositorio a partir de la copia maestra del mismo, además de actualizar la copia de trabajo con el contenido actual del repositorio local.
- **Conflicto** Situación que surge cuando dos desarrolladores hacen un commit con cambios en la *misma región del mismo fichero*. El scv lo detecta, pero es el programador el que debe corregirlo.

Son muchos y variados los sistemas de control de versiones existentes...

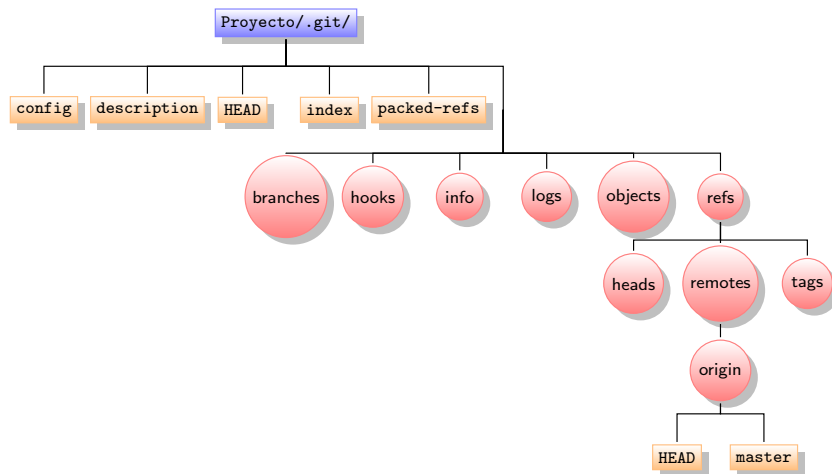
- **SCCS** implementación libre de GNU, **RCS**
- **Cvs** , **Subversion**
- **BitKeeper**
- **Bazaar, bzt**
- **mercurial** , **monotone** , **darcs** , **Perforce**
- **Git** es el que usaremos a lo largo de la asignatura.

Git: Historia

- Los desarrolladores de linux emplean **BitKeeper** hasta 2005.
- BitKeeper es un scv distribuido. Git también lo es, al igual que Darcs, Mercurial, SVK, Bazaar y Monotone.
- Linus comienza el desarrollo de git el 3 de abril de 2005, lo anuncia el día 6 de abril.
- Git se auto-hospeda el 7 de abril de 2005.
- El primer núcleo de linux gestionado con git se libera el 16 de junio de 2005, fue el 2.6.12.
- ¿Qué significa git ?... pues depende, según el propio Linus puede ser:
 - 1 "I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git."
 - 2 "Global Information Tracker".
- Su página oficial: **web oficial de git** .

Git: Implementación

- La parte de bajo nivel (**plumbing**) se puede ver como un sistema de ficheros direccionable por el contenido.
- Por encima incorpora todas las herramientas necesarias que lo convierten en un scv más o menos amigable (**porcelain**).
- Cuenta con aplicaciones escritas en C y en shell. Con el paso del tiempo algunas de estas últimas se han reescrito en C.
- Los elementos u objetos en los que git almacena su información se identifican por su valor **SHA-1** .



- La orden principal: git

Comprobamos la versión instalada

```
1 > git --version
git version 1.7.8.3
```

- Creamos el repositorio:

Iniciación

```
2 > mkdir Proyecto; cd Proyecto; git init
> git init Proyecto
```

Uso (II)

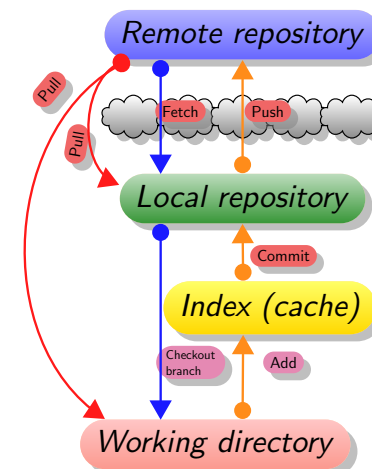
Uso (IIb)

- Añadimos archivos y guardamos:

Añadir, guardar

```
2 > git add .
> git status
> git commit -m 'Primer commit.' -m "
    Descripción detallada."
4 > git commit -a
```

- El 'escenario', '*stage*' o también '*index*'. Relación con "git add".
- Lo podemos ver gráficamente en la siguiente imagen:



- Configuración: archivos “.git/config” o “~/.gitconfig”.
- El primero es particular del proyecto actual y el segundo es general para todos los proyectos del usuario.

Configurar

```

> git config user.name "nombre apellidos"
2 > git config user.email "usuario@email.com"
...
4 > git config --global user.name "nombre
  apellidos"
  > git config --global user.email "usuario@email
    .com"

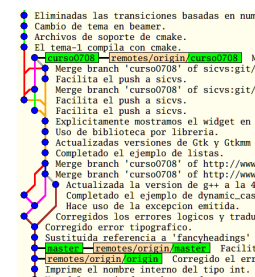
```

Ramas

```

1 > git branch [-a] [-r]
  > git show-branch
3 ...
  > git checkout [-b] [rama-de-partida]
5 > git log [-p]
  > gitk --all

```



Información

```

> git status
2 > git log
...
4 > git show
  > git diff

```

Descartar cambios

```

1 > git reset --hard
  > git checkout ruta-archivos o rama

```

Repositorios remotos

```

> git remote add nombre protocolo
2 > git remote add origin maquina:ruta/hasta/repo
...
4 [ssh] [http] [git] [git+ssh]
  > git clone maquina:ruta/hasta/repo

```

Operaciones con repositorios remotos

```

1  > git pull [origin] [rama]
   > git push [repo] [rama]
3  > git checkout -b rama origin/rama-remota
   > git fetch
5  > git merge
   > git pull = git fetch + git merge
7  > git rebase otra-rama

```

stash

```
1  > git stash [list | show | drop | ...]
```

Echa un vistazo a este tutorial sobre [git stash](#) .

bisect

```
1  > git bisect [help | start | bad | good | ...]
```

Echa un vistazo a este tutorial sobre [git bisect](#) .

Repositorio en máquina remota

```

1  > GIT_DIR=ruta/hasta/Proyecto.git git init
   > git clone maquina-remota:ruta/hasta/Proyecto.git

```

- gitk
- git gui
- git view
- gitg
- giggle
- gource
- Interfaz desde [anjuta](#) , [geany](#) , [eclipse](#) , [emacs](#) [magit](#) o [emacs git](#) .

- Ubuntu/Debian: `apt-get install git`.
- Paquetes recomendados: `git-doc`, `git-arch`, `git-cvs`, `git-svn`, `git-email`, `git-daemon-run`, `git-gui`, `gitk`, `gitweb`.
- No confundir con el paquete `git`, el cual recientemente se ha renombrado a “gnuit”.

CVS

```
> git cvsimport -d :pserver:user@machine:/cvsroot
/module name
```

SVN

```
1 > git svn clone file:///tmp/test-svn -T trunk -b
   branches -t tags
2 > git svn clone file:///tmp/test-svn -s
3 ...
4 > git commit -am 'Adding git-svn instructions to
   the README'
5 > git svn dcommit
6 ...
7 > git svn rebase
```

Casos de uso (I)

- ¿Cómo creo una rama local que ‘siga’ los cambios en una remota al hacer ‘pull’?
`git branch --track ramalocal origin/master`
- ¿Se puede crear una rama que no parta del último commit de otra?...sí:
`git branch --no-track feature3 HEAD~4`
- ¿Quién hizo qué ‘commit’ en un fichero?:
`git blame fichero`
- ¿Cómo creo una rama para resolver un bug y lo integro de nuevo en la rama principal?:
`git checkout -b fixes`
`hack...hack...hack`
`git commit -a -m "Crashing bug solved."`
`git checkout master`
`git merge fixes`

Casos de uso (II)

- He modificado localmente el fichero ‘src/main.vala’ y no me gustan los cambios hechos. ¿Cómo lo devuelvo a la última versión bajo control de versiones?:
`git checkout -- src/main.vala`
- ¿Y un directorio completo, p.e. a la penúltima versión de la rama ‘test’?:
`git checkout test~1 -- src/`
- ¿Y si he modificado varios ficheros y quiero dejar todo como estaba antes de la modificación?...tenemos varias maneras:
`git checkout -f`
o también:
`git reset --HARD`

- ¿Se puede deshacer un 'commit' que es una mezcla (merge) de varios 'commits'?...**sí**, hay que elegir cuál o cuáles de los commits que forman la mezcla (merge) así:
`git revert HEAD~1 -m 1`
 En este ejemplo estaríamos deshaciendo sólo el primero de los 'commits' que formaban este 'merge'.
- ¿Cómo puedo obtener un archivo tal y como se encontraba en una versión determinada del proyecto?, de varias maneras:
`git show HEAD~4:index.html > oldIndex.html`
 o también así:
`git checkout HEAD~4 -- index.html`

- ¿De qué maneras distintas puedo ver los cambios que ha habido en el repositorio?:
`git diff`
`git log --stat`
`git whatchanged`
- ¿Cómo puedo saber cuántos commits ha hecho cada miembro del proyecto en la rama actual?:
`git shortlog -s -n`
 ¿Y en todas las ramas?:
`git shortlog -s -n --all`
- ¿Cómo puedo corregir el mensaje de explicación del último commit que he hecho?:
`git commit --amend`
 Abre el editor por defecto y nos permite modificarlo.

¿Sería posible un tutorial interactivo?

¡Pues claro!... echa un vistazo a [Try Git](#) .

Un ejemplo sencillo paso a paso

- Elige un directorio que contenga el código de una práctica de cualquier asignatura. Cambiate a él.
- Inicia el repositorio en este directorio.
- Añade los archivos que haya inicialmente en él.
- Haz el primer "commit" de los archivos recién importados.
- Haz una modificación a uno o varios de ellos. Comprueba cuáles han cambiado, cómo lo han hecho. Añádelos al siguiente commit.
- Contribuye los cambios creando el "commit".
- Crea una rama en el proyecto y cámbiate a ella automáticamente.
- Haz cambios y commits en esta rama.
- Vuelve a la rama "master".

- [git](#)
- [git guide](#)
- [Carl's Worth tutorial](#)
- [git-for-computer-scientists](#)
- [gitmagic](#)
- [freedesktop](#)
- [gitready](#)
- [progit](#)
- [winehq](#)
- Presentación en [vídeo de git](#) hecha por Linus Torvalds