

Tema: II

Control de versions

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Departament de Llenguatges i Sistemes Informàtics
Universitat d'Alacant

Curs 2014-2015 , Copyleft © 2011-2015 .

Reproducció permesa sota els termes de la llicència de documentació
lliure GNU.

Contenido

- 1 Introducció
- 2 En què consisteix el control de versions?
- 3 Classificacions dels scm
- 4 Conceptes generals dels scm
- 5 Una mica d'història
- 6 Git: Història
- 7 Git: Implementació
- 8 Git: Directori '.git'
- 9 Ús (I)
- 10 Ús (II)
- 11 Ús (III)
- 12 Ús (IV)
- 13 Ús (V)
- 14 Ús (VI)
- 15 Ús (VII)
- 16 Ús (VIII)
- 17 Ús (IX)
- 18 Ús (X)
- 19 Instal·lació
- 20 Interacció amb cvs i svn
- 21 Casos d'ús
- 22 Seria possible un tutorial interactiu?
- 23 Un exemple senzill pas a pas
- 24 Webs d'interès

El Control de versions en la pràctica

Vegem amb un exemple senzill la utilitat del control de versions:

Suposem la següent situació en la qual hem escrit est codi ↓

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

Posteriorment fem una sèrie de canvis en ell ↓

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

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

Amb els canvis realitzats podem:

- Conservar només l'última versió de l'arxiu.
- Mantenir la versió anterior per si els canvis introdueixen alguna fallada¹.
- Conèixer qui els va realitzar (cas de treballar en grup).
- Desfer-los per recuperar la version anterior (en cas de haver-la perdut).
- Aïllar-los per enviarselos a un altre desenvolupador perquè els aplicació a la version de l'arxiu que el té (*pegat*).

Però tot això...es pot fer manualment!

¹ | l'anterior de l'anterior?...

En què consisteix el control de versions?

Per tant, què ens aporten els sistemes de control de versions²?:

- La gestió automàtica dels canvis que es realitzen sobre un o diversos fitxers d'un projecte.
- Restaurar cadascun dels fitxers d'un projecte a un estat dels anteriors pels quals ha anat passant (no solament al immediatament anterior).
- Permetre la col·laboració de diversos programadors en el desenvolupament d'un projecte.

²D'ara endavant scv .

- Per la forma d'emmagatzemar els continguts:
 - 1 Centralitzats
 - 2 Distribuïts
- Per la manera en la qual permeten que cada desenvolupador pugui modificar la còpia local de les dades extretes del repositori:
 - 1 Col·laboratius
 - 2 Exclusius

Conceptes generals dels scv (I)

- **Repositori** És la còpia mestra on es guarden totes les versions dels arxius d'un projecte. En el cas de git es tracta d'un directori. Cada desenvolupador té la seva pròpia còpia local d'aquest directori.
- **Còpia de treball** La còpia dels fitxers del projecte que podem modificar.

Conceptes generals dels scv (II)

- **Check Out / Cloni** L'acció emprada para obtenir una còpia de treball des del repositori. En els scv distribuïdos -com Git- aquesta operació es coneix com **clonar** el repositori per que, a més de la còpia de treball, proporciona a cada programador la seva còpia local del repositori a partir de la *còpia mestra* del mateix.
- **Check In / Commit** L'acció emprada para portar els canvis fets en la còpia de treball a la còpia local del repositori³. Això crea una nova revisió dels arxius modificats. Cada 'commit' ha d'anar acompanyat d'un **Log Message** el qual és un comentari⁴ que afegim a una revisió quan fem el commit oportú.
- **Push** La accion que trasllada els continguts de la còpia local del repositori d'un programador a la còpia mestra del mateix.

³Check In.

⁴Una cadena de text que explica el commit.

Conceptes generals dels scv (III)

- **Update/Pull/Fetch+Merge/Depassi** Acció emprada per actualitzar la nostra còpia local del repositori a partir de la còpia mestra del mateix, a més d'actualitzar la còpia de treball amb el contingut actual del repositori local.
- **Conflicte** Situació que sorgeix quan dues desenvolupadors fan un commit amb canvis en la *mateixa regió del mateix fitxer*. El scv ho detecta, però és el programador el que ha de corregir-ho.

Un poco de historia

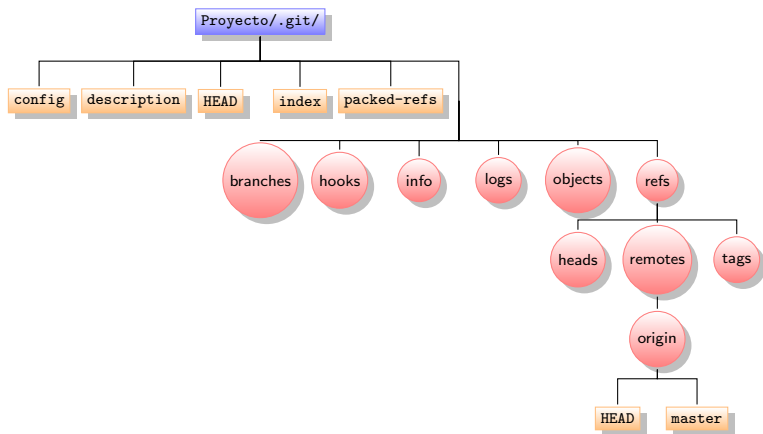
Són molts i variats els sistemes de control de versions existents...

- `SCCS` implementació lliure de GNU, `RCS`
- `Cvs` , `Subversion`
- `BitKeeper`
- `Bazaar`, `bzr`
- `mercurial` , `monotone` , `darcs` , `Perforce`
- `Git` és el que usarem al llarg de l'assignatura.

- Els desenvolupadors de linux empren [BitKeeper](#) fins a 2005.
- BitKeeper és un scv distribuïdo. Git també ho és, a l'igual que Darcs, Mercurial, SVK, Bazaar i Monotone.
- Linus comença el desenvolupament de git el 3 d'abril de 2005, el anuncia el dia 6 d'abril.
- Git s'acte-allotja el 7 d'abril de 2005.
- El primer nucli de linux gestionat amb git s'allibera el 16 de juny de 2005, va anar el 2.6.12.
- Què significa git ?... doncs depèn, segons el propi Linus pot ser:
 - ① *"I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git."*
 - ② *"Global Information Tracker"*.
- La seva pàgina oficial: [web oficial de git](#) .

- La part de baix nivell ([plumbing](#)) es pot veure com un sistema de fitxers adreçable per el contingut.
- Per damunt incorpora totes les eines necessàries que el converteixen en un scm més o menys amigable ([porcelain](#)).
- Compta amb aplicacions escrites en C i en shell. Amb el pas del temps algunes d'aquestes últimes es han reescrit en C.
- Els elements o objectes en els quals git emmagatzema el seu informació s'identifiquen pel seu valor `SHA-1`.

Git: Directorio '.git'



- L'ordre principal: git

Comprovem la versió instal·lada

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

- Vam crear el repositori:

Iniciación

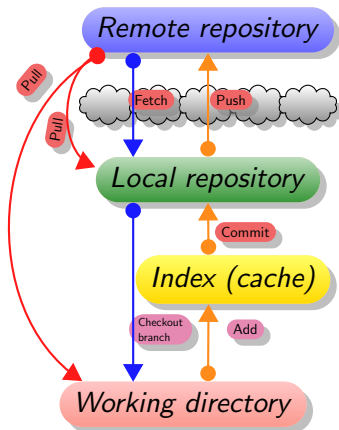
```
2      > mkdir Proyecto; cd Proyecto; git init  
      > git init Proyecto
```

- Afegim arxius i guardem:

Afegir, guardar

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

- El 'escenari', '*stage*' o també '*index*'. Relació amb "git add".
- El podem veure gràficament en la següent imatge:



- Configuració: arxius “.git/config” o “~/ .gitconfig”.
- El primer és particular del projecte actual i el segon és general per a tots els projectes de l'usuari.

Configurar

```
2      > git config user.name "nombre apellidos"
      > 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
    
```

```

● Eliminadas las transiciones basadas en num
● Cambio de tema en beamer.
● Archivos de soporte de cmake.
● El tema-1 compila con cmake.
● curso0708 - remotes/origin/curso0708 y
● Merge branch 'curso0708' of sicvs:git/
● Facilita el push a sicvs.
● Merge branch 'curso0708' of sicvs:git/
● Facilita el push a sicvs.
● Facilita el push a sicvs.
● Explicitamente mostramos el widget en
● Uso de biblioteca por libreria.
● Actualizadas versiones de Gtk y Gtkmm
● Completado el ejemplo de listas.
● Merge branch 'curso0708' of http://www
● Merge branch 'curso0708' of http://www
● Actualizada la version de g++ a la 4
● Completado el ejemplo de dynamic_cas
● Hace uso de la excepcion entida.
● Corregidos los errores logicos y tradu
● Corregido error tipografico.
● Sustituida referencia a 'fancyheadings'
● master - remotes/origin/master Facilit
● remotes/origin/origin Corregido el err
● Imprime el nombre interno del tipo int.
    
```

Informació

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

Descartar canvis

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

Repositoris remots

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

Operacions amb repositoris remots

```
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 | ...]
```

Fa un cop d'ull a aquest tutorial sobri [git stash](#) .

bisect

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

Fa un cop d'ull a aquest tutorial sobri [git bisect](#) .

Repositori en màquina 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`.
- Paquets recomanats: `git-doc`, `git-arch`, `git-cvs`, `git-svn`, `git-email`, `git-daemon-run`, `git-gui`, `gitk`, `gitweb`.
- No confondre amb el paquet `git`, el qual recentment es ha renombrado a “gnuit”.

Interacción con cvs y svn

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  
    > git svn clone file:///tmp/test-svn -s  
3 ...  
    > git commit -am 'Adding git-svn instructions to  
        the README'  
5 > git svn dcommit  
    ...  
7 > git svn rebase
```

Casos d'ús (I)

- Com creo una branca local que 'segueixi' els canvis en una remota en fer 'pull'?

```
git branch --track ramalocal origin/master
```

- Es pot crear una branca que no parteixi de l'últim commit d'una altra?...sí:

```
git branch --no-track feature3 HEAD~4
```

- Qui va fer què 'commit' en un fitxer?:

```
git blame fichero
```

- Com creo una branca per resoldre un bug i ho integro de nou en la branca principal?:

```
git checkout -b fixes
```

```
hack...hack...hack
```

```
git commit -a -m "Crashing bug solved."
```

```
git checkout master
```

```
git merge fixes
```

- He modificat localment el fitxer 'src/main.vala' i no m'agraden els canvis fets. Com ho retorno a l'última versió sota control de versions?:
`git checkout -- src/main.vala`
- I un directori complet, p.i. a la penúltima versió de la branca 'test'?:
`git checkout test~1 -- src/`
- I si he modificat diversos fitxers i vull deixar tot com estava abans de la modificació?...tenim diverses maneres:
`git checkout -f`
o també:
`git reset --HARD`

- Es pot desfer un 'commit' que és una barreja (merge) de diversos 'commits'?...**sí**, cal triar quin o quins de els commits que formen la barreja (merge) així:

```
git revert HEAD~1 -m 1
```

En aquest exemple estaríem desfent només el primer dels 'commits' que formaven aquest 'merge'.

- Com puc obtenir un arxiu tal com es trobava en una versió determinada del projecte?, de diverses maneres:

```
git show HEAD~4:index.html > oldIndex.html
```

o també així:

```
git checkout HEAD~4 -- index.html
```

Casos d'ús (IV)

- De quines maneres diferents puc veure els canvis que hi ha hagut en el repositori?:
`git diff`
`git log --stat`
`git whatchanged`
- Com puc saber quants commits ha fet cada membre del projecte en la branca actual?:
`git shortlog -s -n`
I en totes les branques?:
`git shortlog -s -n --all`
- Com puc corregir el missatge d'explicació de l'últim commit que he fet?:
`git commit --amend`
Obre l'editor per defecte i ens permet modificar-ho.

¿Sería posible un tutorial interactivo?

Doncs clar!... fa un cop d'ull a [Try Git](#) .

Un ejemplo sencillo paso a paso

- Tria un directori que contingui el codi d'una pràctica de qualsevol assignatura. Cambiate a ell.
- Inicia el repositori en aquest directori.
- Afegeix els arxius que hi hagi inicialment en ell.
- Fes el primer “commit” dels arxius recentment importats.
- Fes una modificació a un o varis d'ells. Comprova quins han canviat, com ho han fet. Afegeix-los al següent commit.
- Contribueix els canvis creant el “commit”.
- Crea una branca en el projecte i canvia't a ella automàticament.
- Fes canvis i commits en aquesta branca.
- Torna a la branca “master”.

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