Unit: II
Version Control Systems

Herramientas Avanzadas para el Desarrollo de Aplicaciones

Languages and computing systems
University of Alicante

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

Content

lsi Department of Software and Computing Systems

Version control in practice

Let's see a simple example to understand the utility of version control systems:

Let's suppose the next case in which we have written this code ↓

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

Next we do some modifications on it ↓

```
/*
* Hello world.
* date: 22/01/2010
*/

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

Version control in practice

With the modifications done we can:

- Keep only the last version of the file.
- Keep the previous version in case the modifications have any mistake[1].
- Know who did the modifications (in the case of a group of work).
- Undo the modifications to access the previous version (in the case of having lost it).
- Isolate the modifications to send them to another developer so he can add them to the version of the file s/he has (*patch*).

But all this...can be done manually!

[1] and the previous of the previous version?...

What is the version control about?

Therefore, what do the **v**ersion **c**ontrol **s**ystems bring us[2]?:

- The automatic management of the changes that are done in one or several files of a project.
- Restore each of the files of a project to a previous status (not just to the previous one).
- Allow the collaboration of several programmers in the development of a project.

[2] from now on `vcs` .

Clasifying the vcs

- By the way of storing the data:
  1. Centralized
  2. Distributed
- By the way of allowing each developer to modify the local copy of the data extracted from the repository:
  1. Collaborative
  2. Exclusive

General concepts of the `vcs` (I)

- Repository It is the master copy where all the versions of the files of a project are stored. In the case of `git` the repository is a directory. Each developer has his/her own local copy of this directory.
- Working copy The copy of the files of the project we can modify.

General concepts of the `vcs` (II)

- Check Out / Clone This is the action for obtaining a working copy from the repository. In the distributed `vcs` -like Git- this operation is known as clone the repository because, besides the working copy, it provides every programmer with his/her local copy of the repository from its *master copy*.
- Check In / Commit This is the action used to upload the modifications done in the working copy to the local copy of the repository[3]. This action creates a new revision of the modified files. Each 'commit' must go together with a Log Message which is a comment[4] that we add to a revision when we do the proper `commit`.
- Push This is the action that moves the contents of the local copy of the repository of a programmer to its master copy.

[3] Check In.

[4] A string that explains the commit.

General concepts of the `vcs` (III)

- Update/Pull/Fetch+Merge/Rebase Action used to update our local copy of the repository from its master copy, besides updating the working copy with the current content of the local repository.
- Conflict Situation that occurs when two developers do a `commit` with modifications in *the same part of the same file*. The `vcs` detects it, but is the programmer who has to correct it.

A bit of history

There exist many version control systems...

- SCCS free GNU implementation, RCS
- Cvs , Subversion
- BitKeeper
- Bazaar, bzr
- mercurial , monotone , darcs , Perforce
- Git this is the one we will use in this subject.

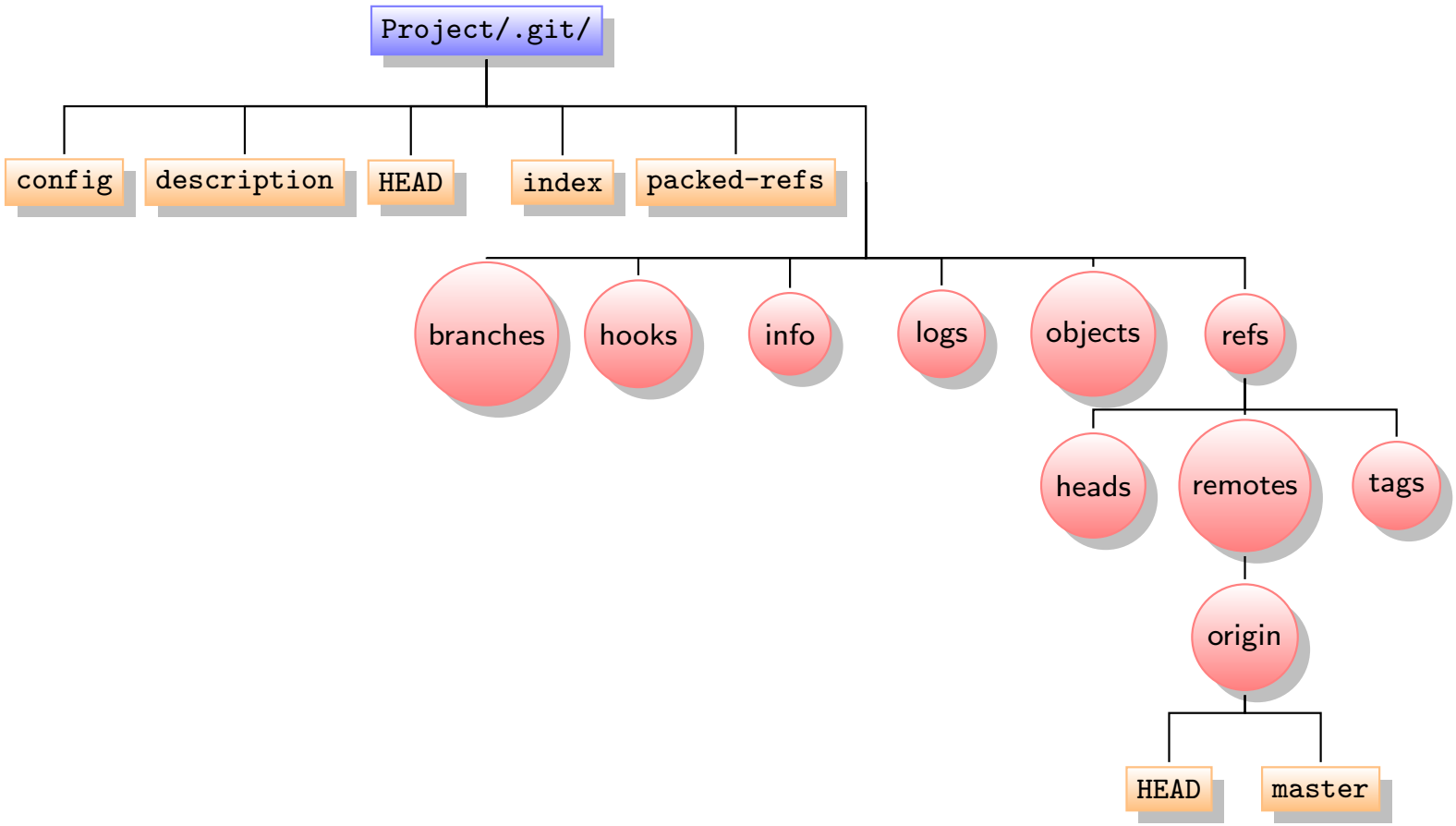
Git: History

- `Linux` developers use BitKeeper until 2005.
- BitKeeper is a distributed `vcs` . Git it is also distributed, such as `Darcs`, `Mercurial`, `SVK`, `Bazaar` and `Monotone`.
- Linus starts the development of `git` on 3 April 2005, it is announced on 6 April.
- Git becames self-hosting as of 7 April 2005.
- On June 16, the kernel 2.6.12 release was managed by Git.
- What does `git` means?... it depends, Linus Tolvards said:
  1. "*I'm an egotistical bastard, and I name all my projects after myself. First Linux, now git.*".
  2. "*Global Information Tracker*".
- Official website: official `git` website .

Git: Implementation

- The low level part (`plumbing`) can be seen as a file system addressed by the content.
- On top it has all the needed tools that makes `git` a more or less friendly `vcs` (`porcelain`).
- It has applications written in `C` and in `shell`. Some of the last ones have been rewritten using `C`.
- The elements or objects in which `git` stores the information are identifyied by its value SHA-1 .

Git: Directory '.git'



Use (I)

- The main command: `git`

We check the installed version

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

- We create the repository:

Initiate

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

Use (II)

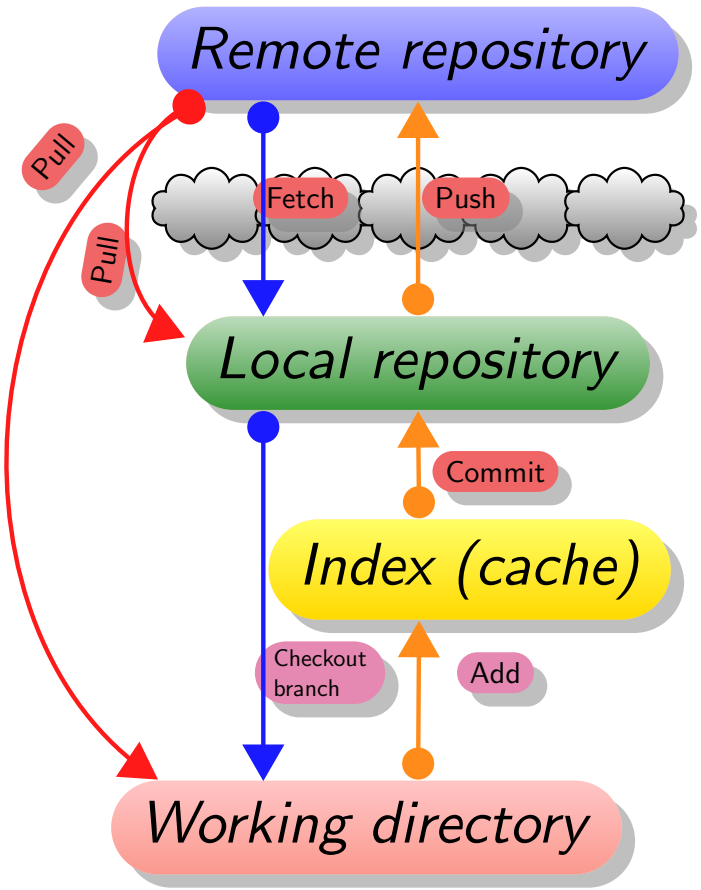
- We add files and store:

Add, save

```
> git add .
> git status
> git commit -m 'First commit.' -m "Detailed
    description."
> git commit -a
```

- The '*stage*' or also '*index*'. Related with "`git add`".
- We can see this graphically in the next slide:

Use (IIb)


Remote repository
Pull
Pull
Fetch
Push
Local repository
Commit
Index (cache)
Checkout branch
Add
Working directory

Use (III)

- Configuration: files "`.git/config`" or "`~/.gitconfig`".
- The first one belongs to the current project and the second is general for all the user projects.

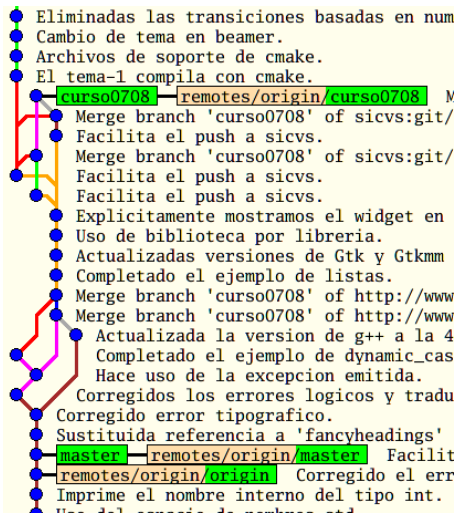
Configure

```
        > git config user.name "name surname"
2       > git config user.email "user@email.com"
        ...
4       > git config --global user.name "name surname"
        > git config --global user.email "user@email.
            com"
```

Use (IV)

Branches

```
1       > git branch [-a] [-r]
        > git show-branch
3       ...
        > git checkout [-b] [start-branch]
5       > git log [-p]
        > gitk --all
```



Use (V)

Information

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

Use (VI)

Discard changes

```
1       > git reset --hard
        > git checkout file path or branch
```

Remote repositories

```
        > git remote add name protocol
2       > git remote add origin machine:path/to/repo
        ...
4       [ssh] [http] [git] [git+ssh]
        > git clone machine:path/to/repo
```

Use (VII)

Operations with remote repositories

```
> git pull [origin] [branch]
> git push [repo] [branch]
> git checkout -b branch origin/remote-branch
> git fetch
> git merge
> git pull = git fetch + git merge
> git rebase another-branch
```

Uso (VIII)

stash

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

Have a look to this tutorial about git stash .

bisect

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

Have a look to this tutorial about git bisect .

Use (IX)

Repository in a remote machine

```
1       > GIT_DIR=path/to/Project.git   git init
       > git clone remote-machine:path/to/Project.git
```

Use (X)
Graphic tools

- gitk
- git gui
- git view
- gitg
- gource
- Interface from anjuta , geany , eclipse , emacs magit or emacs git .

Instalation

- Ubuntu/Debian: `apt-get install git-core`.
- Recommended packages: `git-doc`, `git-arch`, `git-cvs`, `git-svn`, `git-email`, `git-daemon-run`, `git-gui`, `gitk`, `gitweb`.
- Do not confuse with the package git, that has been recently renamed to "`gnuit`".

Interaction with cvs and svn

cvs

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

svn

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

Use cases (I)

- How do I create a local branch that 'follows' the modifications of a remote brach when doing 'pull'?
  ```
  git branch --track ramalocal origin/master
  ```
- Is it possible to create a branch that does not start from the last commit of another one?...**yes**:
  ```
  git branch --no-track feature3 HEAD~4
  ```
- Who did which 'commit' in a file?:
  ```
  git blame fichero
  ```
- How do I create a branch to solve a bug and how do I integrate it again to the main branch?:
  ```
  git checkout -b fixes
  hack...hack...hack
  git commit -a -m "Crashing bug solved."
  git checkout master
  git merge fixes
  ```

lsi Department of Software and Computing Systems

Use cases (II)

- I have modified locally the file '`src/main.vala`' and I do not like the changes done. How do I come back to the last version of the version control system?:
  `git checkout -- src/main.vala`
- And a complete directory, e.g. the last but one version of the branch '`test`'?:
  `git checkout test~1 -- src/`
- And if I have modified several files and I want to leave it all as it was before the modifications?...we have several ways:
  `git checkout -f`
  or also:
  `git reset --HARD`

Use cases (III)

- Can we undo a 'commit' which is a merge of several 'commits'?...**yes**, we have to choose which one/s of the commits that are part of the merge:
  `git revert HEAD~1 -m 1`
  In this example we would undo only the first of the 'commits' which were part of this 'merge'.
- How can I obtain a file as it was in a certain version of the project? , there are several ways:
  `git show HEAD~4:index.html > oldIndex.html`
  or also:
  `git checkout HEAD~4 -- index.html`

Use cases (IV)

- In which different ways can I see the changes done in the repository?:
  `git diff`
  `git log --stat`
  `git whatchanged`
- How can I know how many commits has done each member of the project in the current branch?:
  `git shortlog -s -n`
  And in all the branches?:
  `git shortlog -s -n --all`
- How can I modify the explaining message of the last commit I have done?:
  `git commit --amend`
  It opens the default editor and allows us to modify it.

Would it be possible an interactive tutorial?

Of course!. . . have a look to Try Git .

An easy example step by step

- Choose the directory that contains the code of a practical assignment of any course. Change to it.
- Start the repository in that directory.
- Add two files that are initially on it.
- Do the first "`commit`" of the just imported files.
- Do a modification to one or several of them. Check which ones have changed and how do they have changed. Add them to the next commit.
- Contribute the changes creating the "`commit`".
- Create a branch in the project and change to it automatically.
- Do changes and commits in that branch.
- Come back to the "`master`" branch.

Webs of interest

- git
- git guide
- Carl's Worth tutorial
- git-for-computer-scientists
- gitmagic
- freedesktop
- gitready
- progit
- winehq
- Presentation git video made by Linus Torvalds