

Software version systems

Part II: Distributed versioning systems

Matti Kortelainen

Helsinki Institute of Physics

CERN School of Computing 2008
Gjøvik University College
Tuesday 2nd September, 2008

Version control systems (VCS)

- Version control keeps track of changes
- Makes (or at least should make) easy to share code between developers

Centralized VCS

- CVS
- Subversion

Distributed VCS

- Bazaar (Unix/Mac/Win)
- Darcs (Unix/Mac/Win)
- **Git** (POSIX/Mac + Win)
- Mercurial (Unix/Mac/Win)
- Monotone (Unix/Mac/Win)

Mercurial users

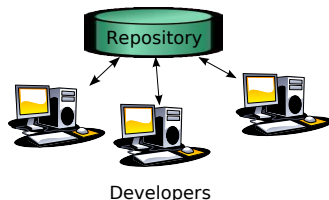
- Mozilla
- OpenSolaris

Git users

- Linux kernel (9 MLOC)
- X.org

Centralized version control

- One central repository which contains the history of code changes
- Each developer communicates only with the repository
- Commits, history browsing etc. require on-line connection between the developer and the repository
- Pitfalls
 - Developers need write access to the repository
 - One possible point of failure
 - Branching and merging might be difficult (CVS)



Distributed version control

Key ideas

- Every developer has a local repository containing the full history of the code
 - Any developer can anytime anywhere write code and commit it
 - Fast history browsing
 - Binary search can be used to find commits which introduce bugs
 - Multiple backups of the project code (automatically)
 - It is very easy for newcomers to start to contribute new code
- Typically branching and merging are technically easy and usually use of branches is encouraged
 - Conflicts will happen, but there are good tools
 - Communication!

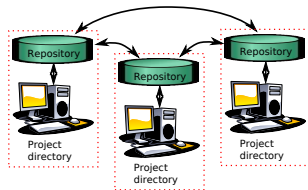
Example file tree

```
project/  
    .git/  
    include/  
    src/  
    ...
```

Distributed version control

Development model

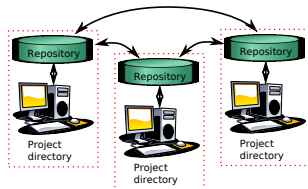
- All repositories are equal, *unless* the developers decide otherwise
- Developers can communicate and share code (individual commits or branches) directly with each other
- Enables several development models
 - Everybody shares with everybody
 - One developer acts as maintainer
 - One repository is decided to be a central repository



Distributed version control

Development model

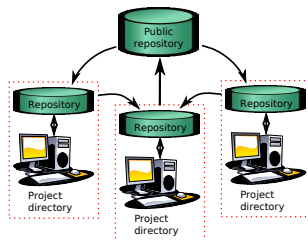
- All repositories are equal, *unless* the developers decide otherwise
- Developers can communicate and share code (individual commits or branches) directly with each other
- Enables several development models
 - **Everybody shares with everybody**
 - One developer acts as maintainer
 - One repository is decided to be a central repository



Distributed version control

Development model

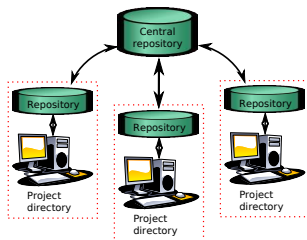
- All repositories are equal, *unless* the developers decide otherwise
- Developers can communicate and share code (individual commits or branches) directly with each other
- Enables several development models
 - Everybody shares with everybody
 - **One developer acts as maintainer**
 - One repository is decided to be a central repository



Distributed version control

Development model


- All repositories are equal, *unless* the developers decide otherwise
- Developers can communicate and share code (individual commits or branches) directly with each other
- Enables several development models
 - Everybody shares with everybody
 - One developer acts as maintainer
 - **One repository is decided to be a central repository**



Distributed version control

Interoperability, repository size

- Good interoperability
 - Import from CVS, Subversion and other distributed VCS
 - It is possible to track e.g. CVS or Subversion repositories using Git
- What about the size of the repository?
 - Not really a problem, *if* the VCS has smart repository format
 - Some support submodules
- Example: Mozilla project¹
 - Size of checkout: 350 MB
 - Original CVS: 2.7 GB

¹http://keithp.com/blogs/Repository_Formats_Matter/ 

Distributed version control

Interoperability, repository size

- Good interoperability
 - Import from CVS, Subversion and other distributed VCS
 - It is possible to track e.g. CVS or Subversion repositories using Git
- What about the size of the repository?
 - Not really a problem, *if* the VCS has smart repository format
 - Some support submodules
- Example: Mozilla project¹
 - Size of checkout: 350 MB
 - Original CVS: 2.7 GB
 - Conversion to Subversion: 8.2 GB

¹http://keithp.com/blogs/Repository_Formats_Matter/

Distributed version control

Interoperability, repository size

- Good interoperability
 - Import from CVS, Subversion and other distributed VCS
 - It is possible to track e.g. CVS or Subversion repositories using Git
- What about the size of the repository?
 - Not really a problem, *if* the VCS has smart repository format
 - Some support submodules
- Example: Mozilla project¹
 - Size of checkout: 350 MB
 - Original CVS: 2.7 GB
 - Conversion to Subversion: 8.2 GB
 - Conversion to Git: 450 MB
 - And this contains the whole history from 1998!

¹http://keithp.com/blogs/Repository_Formats_Matter/

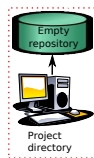
They're not perfect

Or where Subversion beats Git

- Single repository, which contains everything *for sure*
- Access control, as fine-grained as required
- Shorter revision numbers
 - Subversion starts numbering from 1
 - Git uses SHA-1 hashes for identifying commits
- If one has a centralized system working well enough, one shouldn't break it

Usage examples (with Git terminology)

- Start the project
 - Initialize an empty repository

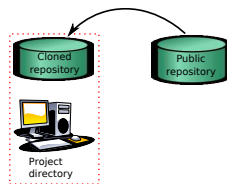


Example command

```
$ mkdir project
$ cd project
$ git init
Initialized empty Git repository
in .git/
```

Usage examples (with Git terminology)

- Start the project
 - **I**nitialize an empty repository
 - **C**lone the public repository

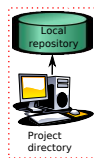


Example command

```
$ git clone  
http://www.kernel.org/pub/  
scm/linux/kernel/git/  
torvalds/linux-2.6.git
```

Usage examples (with Git terminology)

- Start the project
 - **I**nitialize an empty repository
 - **C**lone the public repository
- Edit code, **c**ommit

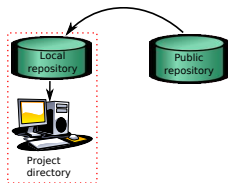


Example command

```
$ git add foo.cc foo.h
$ git commit
    or
$ git commit -a
```

Usage examples (with Git terminology)

- Start the project
 - **Initialize** an empty repository
 - **Clone** the public repository
- Edit code, **commit**
- **Fetch** or **pull** new commits from the public repository, or from someone else

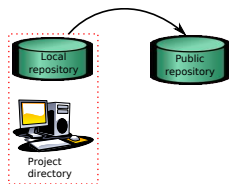


Example command

```
$ git fetch  
or  
$ git pull
```


Usage examples (with Git terminology)

- Start the project
 - **Initialize** an empty repository
 - **Clone** the public repository
- Edit code, **commit**
- **Fetch** or **pull** new commits from the public repository, or from someone else
- **Push** the commits to the public repository



Example command

```
$ git push origin  
or  
$ git push  
ssh://account@host/repository
```

Summary

- Distributed VCS are flexible and enable several development models
- Branching and merging are easy
- Some are very efficient both in speed and in space
- They're not superior in general
 - E.g. if one only want's to track individual files, CVS might be still ok
- Further information
 - <http://git.or.cz/>
 - <http://www.selenic.com/mercurial/>
 - Wikipedia, Google, etc.

