

Version control with Subversion



SoftEng
<http://softeng.polito.it>

February 2009

What is Subversion

- Free/open-source version control system:
- it manages any collection of files and directories over time in a central repository;
- it remembers every change ever made to your files and directories;
- it can access its repository across networks

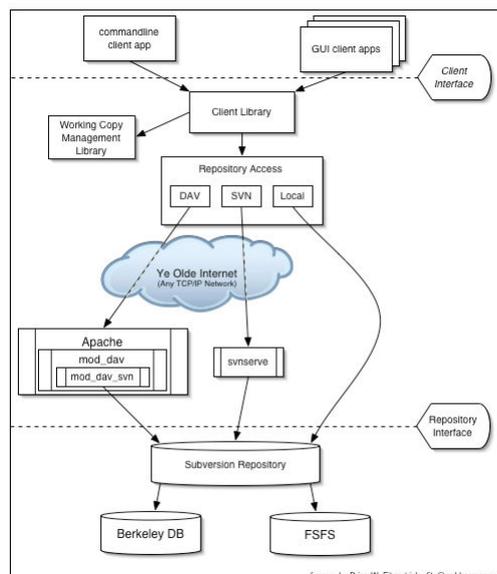
SoftEng
<http://softeng.polito.it>

Features

- Directory versioning and true version history
- Atomic commits
- Metadata versioning
- Several topologies of network access
- Consistent data handling
- Branching and tagging
- Usable by other applications and languages

SoftEng
<http://softeng.polito.it>

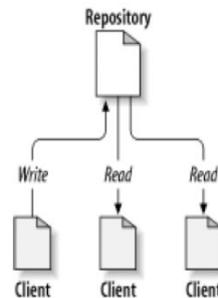
Architecture



SoftEng
<http://softeng.polito.it>

The repository

- Central store of data
- It stores information in the form of a *filesystem*
- Any number of *clients* connect to the repository, and then read or write to these files.
- **\$ svnadmin create /path/to/repos** to create a new repository
- **\$ svn import [PATH] URL** to commit an unversioned file or tree (PATH) into the repository (URL).



SoftEng
http://softeng.polito.it

The working copy

- Ordinary directory tree on your local system, containing a collection of files
- Subversion will never incorporate other people's changes, nor make your own changes available to others, until you explicitly tell it to do so.
- **\$ svn checkout URL repository** to get a working copy
- **\$ svn commit resource path** to publish your changes
- **\$ svn update URL repository** to bring your project up to date

Memento:

“push” action does not cause a “pull”, nor the other way around

SoftEng
http://softeng.polito.it

Revisions

Each time the repository accepts a commit, this creates a new state of the filesystem tree, called a *revision*.

Global revision numbers: each revision is assigned a unique natural number, one greater than the number of the previous revision (the initial revision of a freshly created repository is numbered zero, and consists of nothing but an empty root directory).

Subversion's revision numbers apply to entire trees, not individual files. Each revision number selects an entire tree, a particular state of the repository after some committed change.

So, revision N represents the state of the repository filesystem after the Nth commit. Notice that in general, revisions N and M of a file do not necessarily differ!

SOftEng
<http://softeng.polito.it>

Mixed revisions

Suppose you have a working copy entirely at revision 10. You edit the file `foo.html` and then perform an `svn commit`, which creates revision 15 in the repository.

Therefore the only safe thing the Subversion client can do is mark the one file—`foo.html`—as being at revision 15. The rest of the working copy remains at revision 10. This is a mixed revision.

Only by running `svn update` can the latest changes be downloaded, and the whole working copy be marked as revision 15.

Memento:

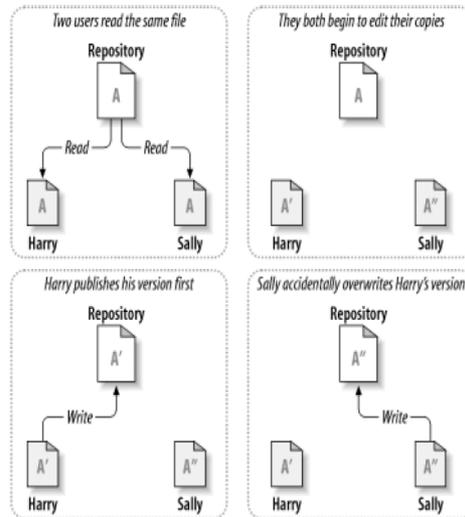
Every time you run `svn commit`, your working copy ends up with some mixture of revisions: the things you just committed are marked as having larger working revisions than everything else.

SOftEng
<http://softeng.polito.it>

Concurrent engineering

All version control systems have to solve the same fundamental problem:

how will the system allow users to share information, but prevent them from accidentally stepping on each other's feet?



SoftEng
<http://softeng.polito.it>

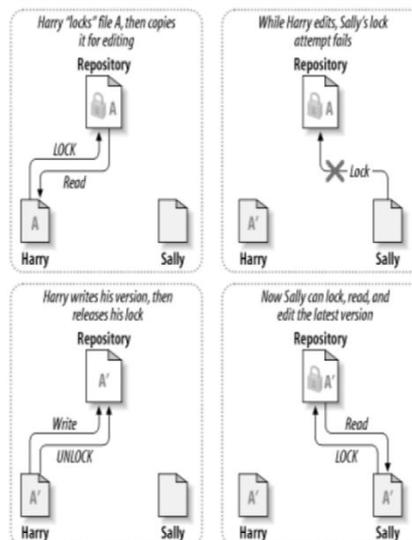
Lock-Modify-Unlock solution

Only one person to change a file at a time; good for editing of binary files, but for others:

Locking may cause administrative problems.

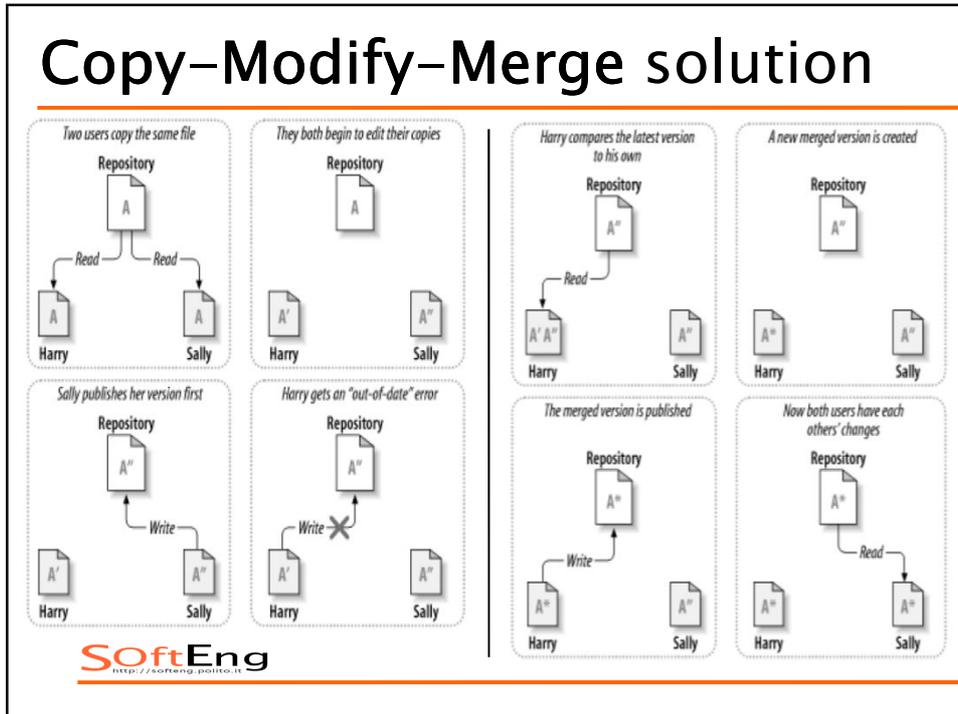
Locking may cause unnecessary serialization.

Locking may create a false sense of security (dependencies among files locked by different persons).



SoftEng
<http://softeng.polito.it>

Copy-Modify-Merge solution



Other commands – 1

svn add	Tell the SVN server a new file or directory is created. Note that the file won't appear in the repository until you do a svn commit
svn delete	Delete a file/directory. When you do an svn commit the file will be deleted from your local sand box immediately as well as from the repository after committing.
svn move SRC DST svn mv SRC DEST or svn rename SRC DEST or svn ren SRC DEST	This command moves a file from one directory to another or renames a file. The file will be moved on your local sand box immediately as well as on the repository after committing.
svn copy SRC DST	Copy a file in a working copy or in the repository. SRC and DST can each be either a working copy (WC) path or URL
svn resolved	Remove «conflicted» state on working copy files or directories. This routine does not semantically resolve conflict markers; it merely removes conflict-related artifact files and allows <i>PATH</i> to be committed again; that is, it tells Subversion that the conflicts have been «resolved»

SOftEng
http://softeng.polito.it

Other commands – 2

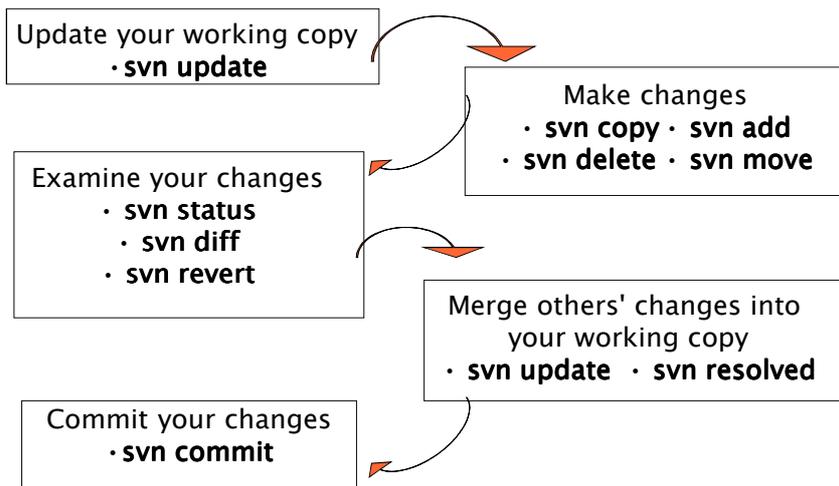
<code>svn status</code> or <code>svn status -u</code>	This command prints the status of working directories and files. If you have made local changes, it'll show your locally modified items. If you use the <code>--verbose</code> switch, it will show revision information on every item. With the <code>--show-updates (-u)</code> switch, it will show any server out-of-date information.
---	--

<code>svn diff [-r N[:M]] [TARGET[@REV]...]</code> <code>svn diff [-r N[:M]] --old OLD-TGT[@OLDREV] [--new NEW-TGT[@NEWREV]] [PATH...]</code> <code>svn diff OLD-URL[@OLDREV] NEW-URL[@NEWREV]</code>	Display the differences between two paths.
---	--

<code>svn revert</code>	Reverts any local changes to a file or directory and resolves any conflicted states. <code>svn revert</code> will not only revert the contents of an item in your working copy, but also any property changes.
-------------------------	--

SoftEng
<http://softeng.polito.it>

Typical work cycle

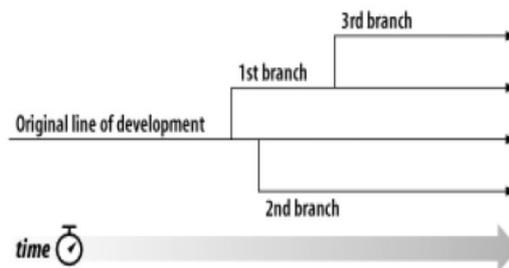


SoftEng
<http://softeng.polito.it>

Branches: general concept

Line of development that exists independently of another line, yet still shares a common history if you look far enough back in time.

A branch always begins life as a copy of something, and moves on from there, generating its own history



SoftEng
<http://softeng.polito.it>

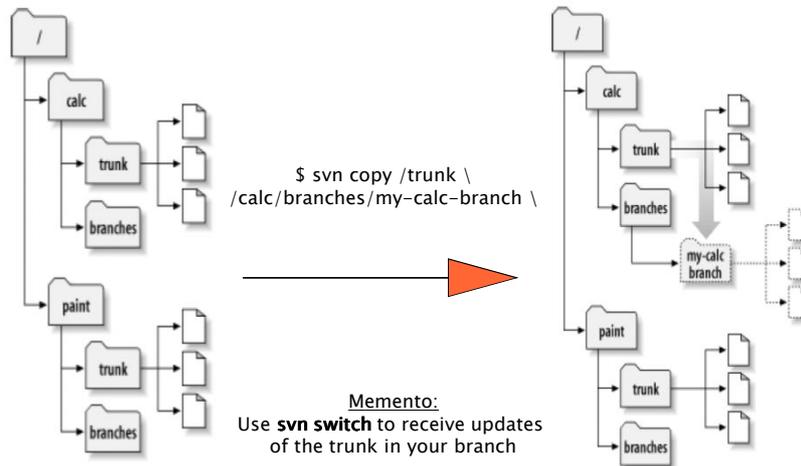
Branches in Subversion

1. Unlike many other version control systems, Subversion's branches exist as *normal filesystem directories* in the repository, not in an extra dimension. These directories just happen to carry some extra historical information.
1. Subversion has no internal concept of a branch—only copies. When you copy a directory, the resulting directory is only a “branch” because you attach that meaning to it. You may think of the directory differently, or treat it differently, but to Subversion it's just an ordinary directory that happens to have been created by copying.

SoftEng
<http://softeng.polito.it>

Branches in Subversion

You create a branch with **svn copy**:



SoftEng
<http://softeng.polito.it>

Merge

When you finish your work in your branch, you need to merge it in the trunk. This is done by **svn merge** command. This command is a very close cousin to the **svn diff** command.

Svn merge, instead of printing the differences to your terminal, however, it applies them directly to your working copy as local modifications. **Svn diff** command ignores ancestry, **svn merge** no.

A better name for the command might have been **svn diffand-apply**, because that's all that happens: two repository trees are compared, and the differences are applied to a working copy.

Conflicts may be produced by **svn merge**: you need to solve them.

SoftEng
<http://softeng.polito.it>

Diff

```
cpu_components.txt r:1091      svn diff -r 1091:1092
                                pc_components.txt

cpu                             Index: pc_components.txt
memory                          =====
keyboard                        - pc_components.txt    (revision 1091)

                                +++ pc_components.txt(revision 1092)

cpu_components.txt r:1092      @@ -1,3 +1,3 @@

cpu                              cpu
memory                          memory
bus                              -keyboard

                                +bus

SoftEng
http://softeng.polito.it
```

Conflicts

For every conflicted file, Subversion places three extra unversioned files in your working copy:

filename.mine : This is your file as it existed in your working copy before you updated your working copy—that is, without conflict markers. This file has only your latest changes in it.

filename.rOLDREV : This is the file that was the BASE revision before you updated your working copy. That is, the file that you checked out before you made your latest edits.

filename.rNEWREV : This is the file that your Subversion client just received from the server when you updated your working copy. This file corresponds to the HEAD revision of the repository.

SoftEng
http://softeng.polito.it

Example

You and Sally both edit file sandwich.txt at the same time. Sally commits her changes, and when you go to update your working copy, you get a conflict and you're going to have to edit sandwich.txt to resolve the conflict. This is what you get in your working copy

```
$ svn update
Conflict discovered in 'sandwich.txt'.
Select: (p) postpone, (df) diff-full, (e) edit,
        (h)elp for more options : p
C sandwich.txt
Updated to revision 2.
$ ls -l
sandwich.txt
sandwich.txt.mine
sandwich.txt.r1
sandwich.txt.r2
```

SoftEng
http://softeng.polito.it

Merging conflicts by hand(1)

Now, let's take a look at the fileSandwich.txt :

```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
<<<<<< .mine
Salami
Mortadella
Prosciutto
=====
Sauerkraut
Grilled Chicken
>>>>>> .r2
Creole Mustard
Bottom piece of bread
```

SoftEng
http://softeng.polito.it



Merging conflicts by hand(2)

The strings of less-than signs, equals signs, and greater-than signs are conflict markers and are not part of the actual data in conflict. **The text between the first two sets of markers is composed of the changes you made in the conflicting area:**

```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
<<<<<<< .mine
Salami
Mortadella
Prosciutto
=====
Sauerkraut
Grilled Chicken
>>>>>>> .r2
Creole Mustard
Bottom piece of bread
```

SoftEng
http://softeng.polito.it

Merging conflicts by hand(3)

The text between the second and third sets of conflict markers is the text from Sally's commit:

```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
<<<<<<< .mine
Salami
Mortadella
Prosciutto
=====
Sauerkraut
Grilled Chicken
>>>>>>> .r2
Creole Mustard
Bottom piece of bread
```

SoftEng
http://softeng.polito.it

Merging conflicts by hand(4)

Usually you won't want to just delete the conflict markers and Sally's changes: this is where you pick up the phone or walk across the office and speak to Sally. Once you've agreed on the changes you will commit, edit your file and remove the conflict markers:

```
Top piece of bread
Mayonnaise
Lettuce
Tomato
Provolone
Salami
Mortadella
Prosciutto
```

Now use `svn resolve`, and you're ready to commit your changes:

```
$ svn resolve --accept working sandwich.txt
Resolved conflicted state of 'sandwich.txt'
$ svn commit -m "Go ahead and use my sandwich, discarding
Sally's edits."
```

SoftEng
<http://softeng.polito.it>

Third party tools

• Clients

- TortoiseSVN
- Subclipse
- AnkhSVN

• Repository browsers

- ViewVC (formerly ViewCVS)
- WebSVN

• Other cool stuff

- svk – distributed VC, with merge tracking, atop Subversion
- cvs2svn –
- Trac – issue tracking system

SoftEng
<http://softeng.polito.it>

Resources

- Subversion community site
 - <http://subversion.tigris.org/>
- *Version Control with Subversion*
 - <http://svnbook.red-bean.com/>
- CollabNet
 - corporate–yet–hands–off sponsorship, support
 - <http://www.collab.net/>
- Subversion hosting
 - http://subversion.tigris.org/project_links.html#hosting