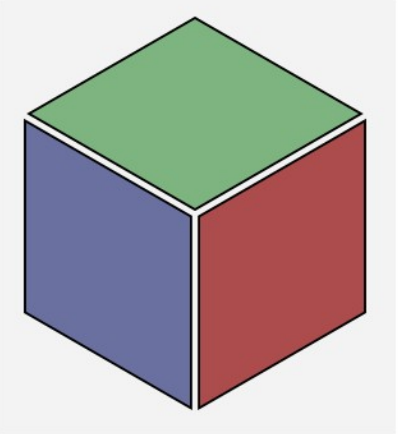
Introduction to Subversion

Getting started with svn

Matteo Vescovi

19/02/2010



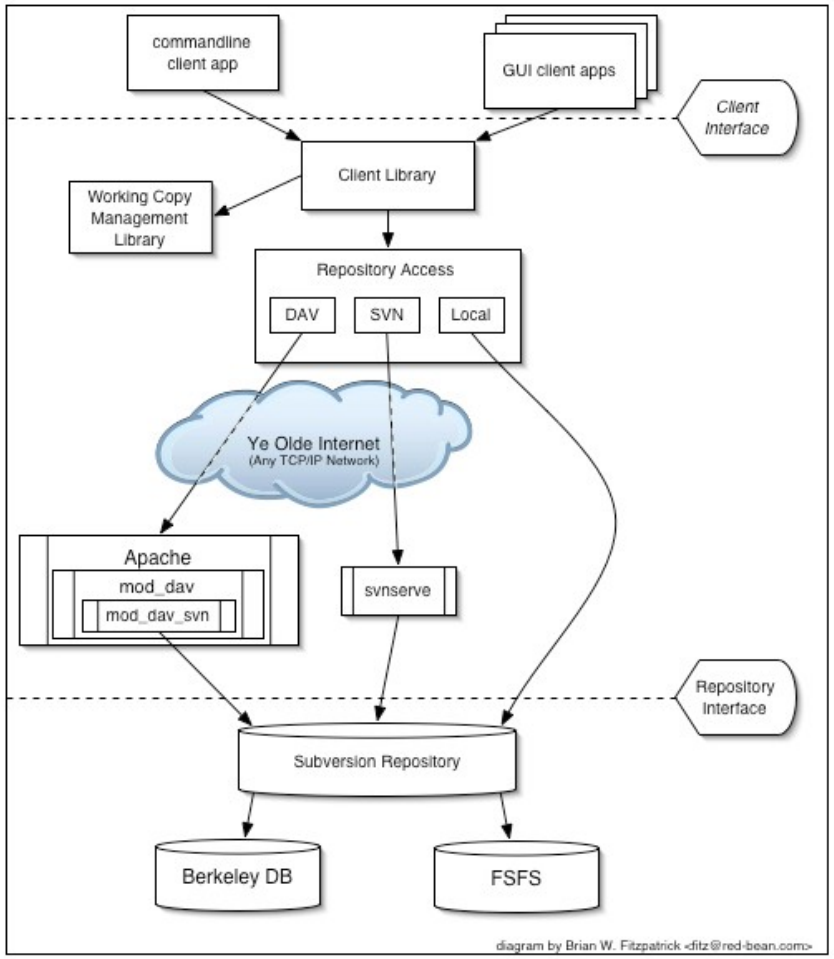
Agenda

- A little bit of theory
  - Overview of Subversion
  - Subversion approach to Version Control
- Using Subversion
  - Typical subversion usage and workflow
  - Examples using mock repository
- Branching and merging
  - Creating branches, keeping branches in sync, merging back
  - Backing out merges
  - Resurrecting deleted items
  - Tagging

What is subversion?

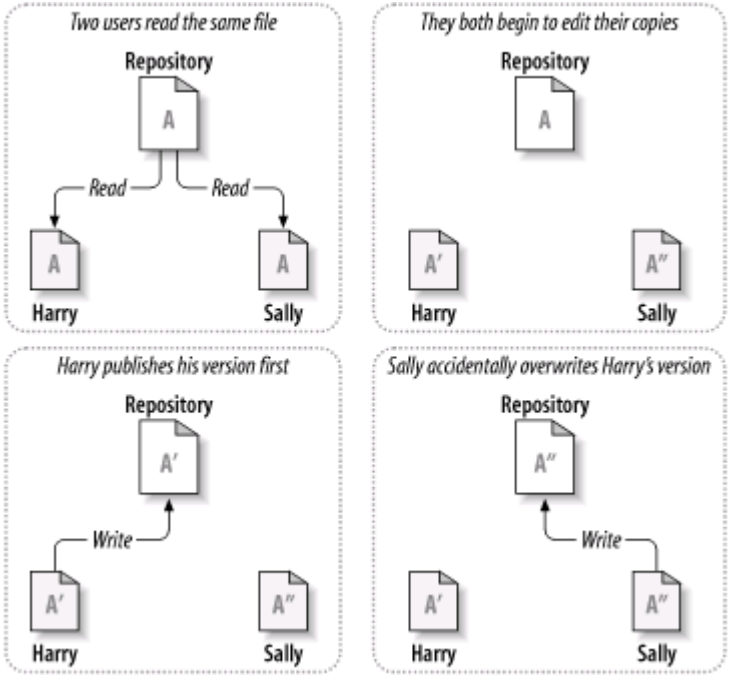
- Subversion is a free/open source version control system. That is, Subversion manages files and directories, and the changes made to them, over time.
- Some version control systems are also software configuration management (SCM) systems. These systems are specifically tailored to manage trees of source code and have many features that are specific to software development—such as natively understanding programming languages, or supplying tools for building software. Subversion, however, is not one of these systems.

Subversion architecture



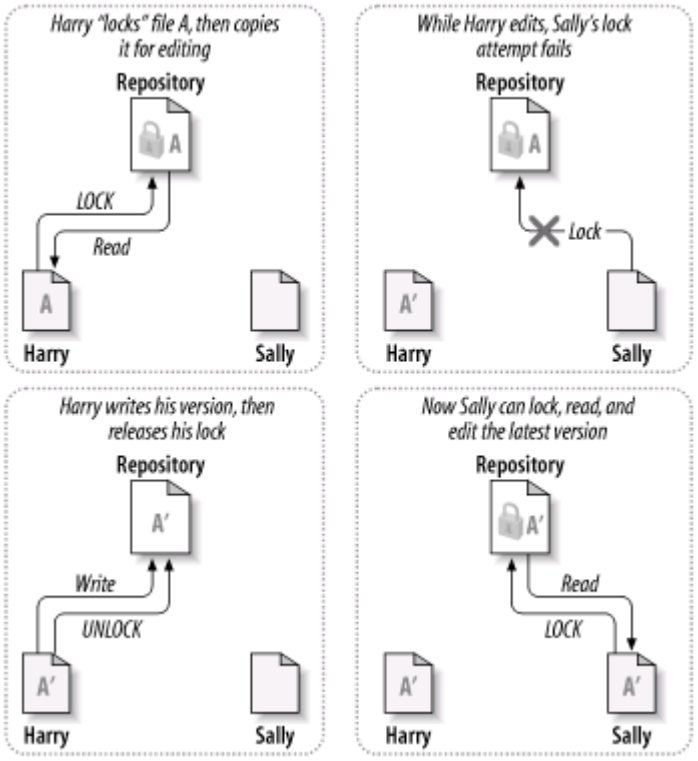
- Repository can use DB or FS as back-end
- SVN supports various protocols (file://, http://, https://, svn://, etc.)
- Multiple language bindings are available
- Users interact with subversion through a client application
  - CLI client: svn
  - GUI clients (on Windows: TortoiseSVN, cross-platform: RapidSVN)

Fundamental problem VCS solve



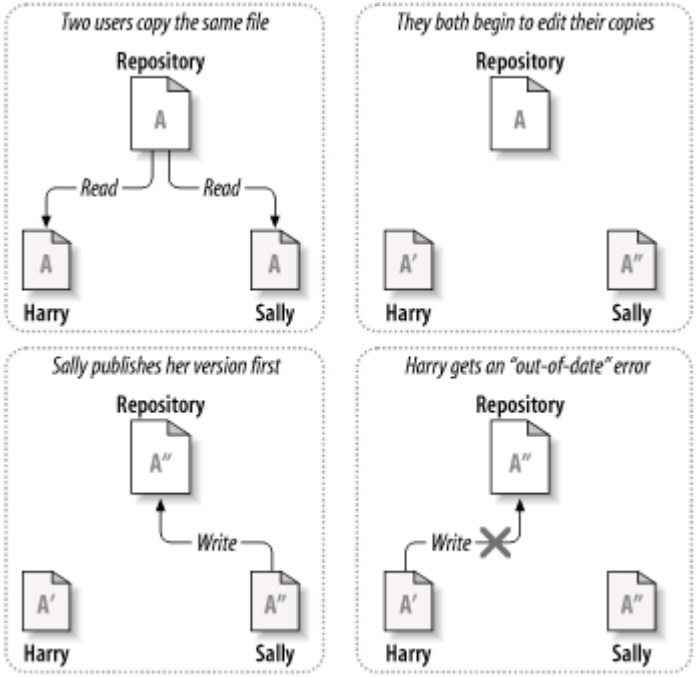
- Version Control Systems solve the following fundamental problem:
  - Harry and Sally decide to edit the same file A
  - Harry and Sally concurrently make changes to A, resulting in A' and A''
  - Harry commits A' to repository before Sally
  - Sally commits A'' to repository, overwriting Harry's changes (A' is lost, overwritten by A'')

Lock-Modify-Unlock solution



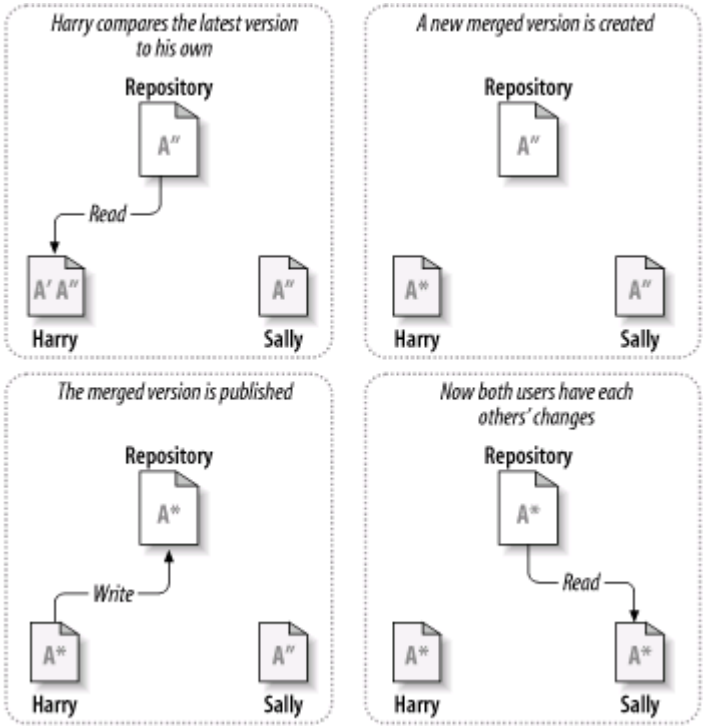
- Harry must "lock" a file before he can begin making changes to it.
- If Harry has locked a file, Sally cannot also lock it, and therefore cannot make any changes to that file.
- All Sally can do is read the file and wait for Harry to finish his changes and release his lock.
- After Harry unlocks the file, Sally can take her turn by locking and editing the file.

Copy-Modify-Merge solution



- Harry and Sally each create working copies of the same project, copied from the repository.
- They work concurrently and make changes to the same file A within their copies.
- Sally saves her changes to the repository first. When Harry attempts to save his changes later, the repository informs him that his file A is *out of date* (A in the repository has changed since he last copied it)

Copy-Modify-Merge solution (2)



- Harry asks his client to *merge* any new changes from the repository into his working copy of file A.
- If Sally's changes don't overlap with Harry's own, changes are integrated automatically. Harry can commits his working copy back to the repository.
- If Sally's changes *do* overlap with Harry's changes, there is a *conflict*. Changes must be integrated manually before committing.

Obtaining a working copy of the repository

- A Subversion working copy is an ordinary directory tree on your local system.
- A working copy also contains some extra files, created and maintained by Subversion, to help it carry out these commands. In particular, each directory in your working copy contains a subdirectory named .svn, also known as the working copy's *administrative directory*.
- To get a working copy, you *check out* some subtree of the repository.

```
$ svn co http://svn.apache.org/repos/asf/subversion/trunk subversion
```

Pushing and pulling changes from repository

- Your working copy is your own private work area: Subversion will never incorporate other people's changes, nor make your own changes available to others, until you explicitly tell it to do so.
- To push your changes to the repository, you can use Subversion's **svn commit** command:

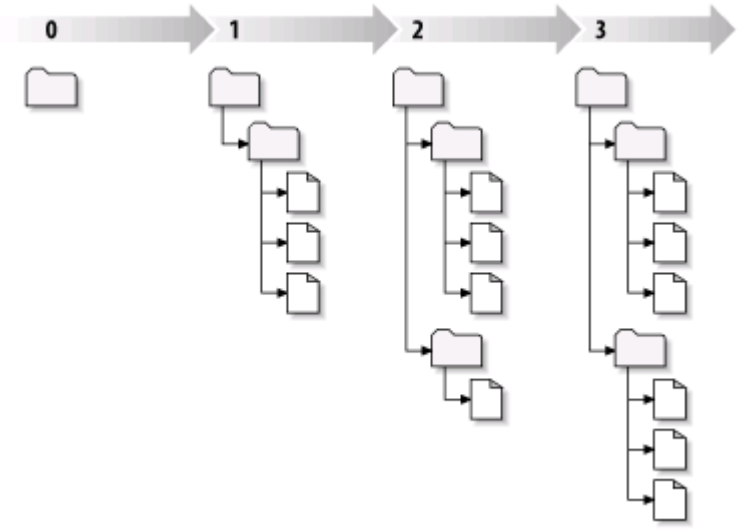
```
$ svn commit README -m “Fixed a typo in README.”
```

- To pull changes from the repository into your working copy, use **svn update** command:

```
$ svn update
```

Revisions

- An **svn commit** operation publishes changes to any number of files and directories as a single atomic transaction.
- Each time the repository accepts a commit, this creates a new state of the filesystem tree, called a *revision*.



How working copies track the repository

Working copy status	svn commit	svn update
Unchanged, and current	Do nothing	Do nothing
Locally changed, and current	Push changes to repository	Do nothing
Unchanged, and out of date	Do nothing	Pull changes from repository
Locally changed, and out of date	Fail with an “out-of-date” error	Pull changes and merge them with local changes. If automatic merge does not succeed, leave it to the user to resolve the conflict.

- Updates and commits are separate. Mixed revisions are normal and useful in a local copy, but have limitations (e.g. deleting an out-of-date file or directory)

Basic work cycle

- In case of doubt
  - **svn help**
- Update your working copy
  - **svn checkout**
  - **svn update**
- Make changes
  - **svn add**
  - **svn delete**
  - **svn copy**
  - **svn move**
  - **svn mkdir**
- Examine your changes
  - **svn status**
  - **svn diff**
- Undo changes
  - **svn revert**
- Resolve conflicts (merge others' changes)
  - **svn update**
  - **svn resolve**
- Commit your changes
  - **svn commit**

Hands-on practice and examples on tutorial repository

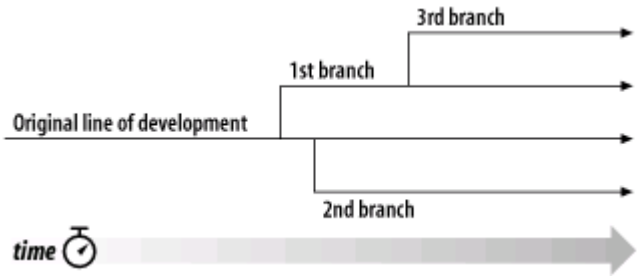
More basic commands

- Examining history
  - **svn log**
  - **svn diff**
  - **svn cat**
  - **svn ls**
- Cleaning up
  - **svn cleanup**
  - rm –rf ; rd /s/q
- Other useful commands
  - **svn annotate**
  - **svn blame**
  - **svn praise**
  - **svn info**

Hands-on practice and examples on tutorial repository

Branching and merging

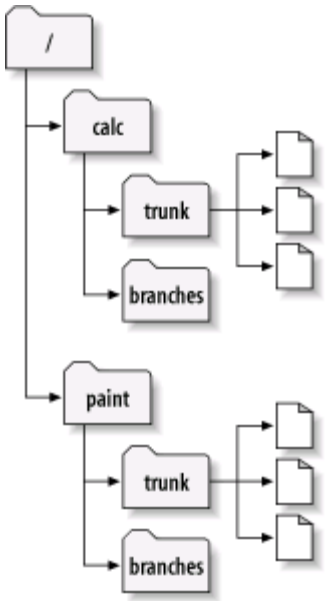
- Subversion handles branches differently:
  - First, Subversion has no internal concept of a branch—it knows only how to make copies. When you copy a directory, the resultant directory is only a “branch” because *you* attach that meaning to it
  - Second, because of this copy mechanism, Subversion's branches exist as *normal filesystem directories* in the repository. This is different from other version control systems, where branches are typically defined by adding extra-dimensional “labels” to collections of files

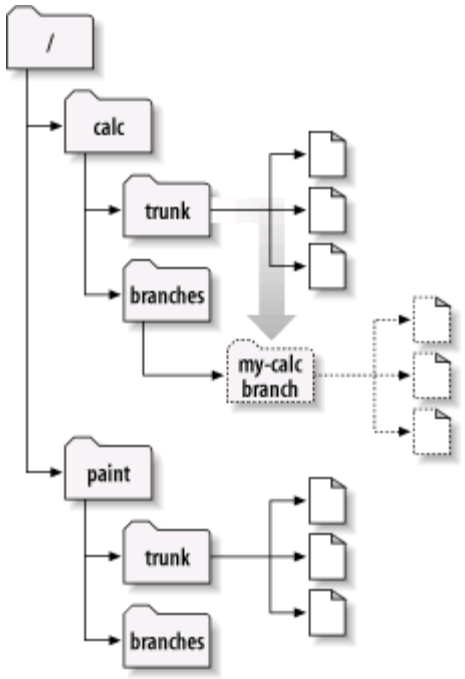


Creating a branch

- To create a branch you make a copy of the project in the repository using the **svn copy** command.

```
$ svn copy http://svn.example.com/repos/calc/trunk \
http://svn.example.com/repos/calc/branches/my-calc-branch \
 -m "Creating a private branch of /calc/trunk"
```





Basic merging

- Replicating changes from one branch to another is performed using various invocations of the **svn merge** command.
- Frequently keeping your branch in sync with trunk helps prevent conflicts when it comes time for you to fold your changes back into the trunk.
  - First ensure that your branch working copy is “clean” (has no local changes) and then run:

```
$ svn merge http://svn.example.com/repos/calc/trunk
```

  - **svn status** and **svn diff** show what changes were merged. Build and test your working copy. Once satisfied, **svn commit** the changes to your branch.

Merging back to trunk

- When you're ready to merge your branch changes back to the trunk:
  - Bring your branch in sync with the trunk again
  - Obtain a “clean” and up-to-date working copy of trunk

```
$ svn merge --reintegrate
http://svn.example.com/repos/calc/branches/my-calc-branch
```

  - Build, test, verify your changes, then commit
  - Delete the branch

```
$ svn delete http://svn.example.com/repos/calc/branches/my-
calc-branch \ -m "Remove my-calc-branch."
```

Roll back changes

- An extremely common use for **svn merge** is to roll back a change that has already been committed.
- All you need to do is to specify a *reverse* difference.

```
$ svn merge –r 303:302 http://svn.example.com/repos/calc/trunk
```

- By using the -r option, you can ask **svn merge** to apply a changeset, or a whole range of changesets, to your working copy. In our case of undoing a change, we're asking **svn merge** to apply changeset #303 to our working copy *backward*.

Resurrecting deleted items

- The first step is to define exactly *which* item you're trying to resurrect.
  - Every object in the repository exists in a sort of two-dimensional coordinate system. The first coordinate is a particular revision tree, and the second coordinate is a path within that tree.
    - A good strategy is to run **svn log --verbose** in a directory that used to contain your deleted item.
- Copy the exact revision and path “coordinate pair” from the repository to your working copy:

```
$ svn copy http://svn.example.com/repos/calc/trunk/real.c@807
./real.c
```

```
$ svn cat http://svn.example.com/repos/calc/trunk/real.c@807 >
./real.c && svn add real.c
```

Tags

- A *tag is* a “snapshot” in time of the project.
- To create a tag, we use **svn copy**:

```
$ svn copy http://svn.example.com/repos/calc/trunk \
 http://svn.example.com/repos/calc/tags/release-1.0 \
 -m "Tagging the 1.0 release of the 'calc' project."
```

- Subversion sees no difference between a tag and a branch. Both are just ordinary directories that are created by copying. Just as with branches, the only reason a copied directory is a “tag” is because *users* have decided to treat it that way.

References

- The red-bean subversion book:
  - Version control with Subversion
  - Freely available on-line
  - http://svnbook.red-bean.com/
- Feel free to try things out in the local subversion repository:
  - http://kangaroo/repos/