Source control management with Subversion

D. Conan and M. Simatic

Internship course

February 2008

Outline

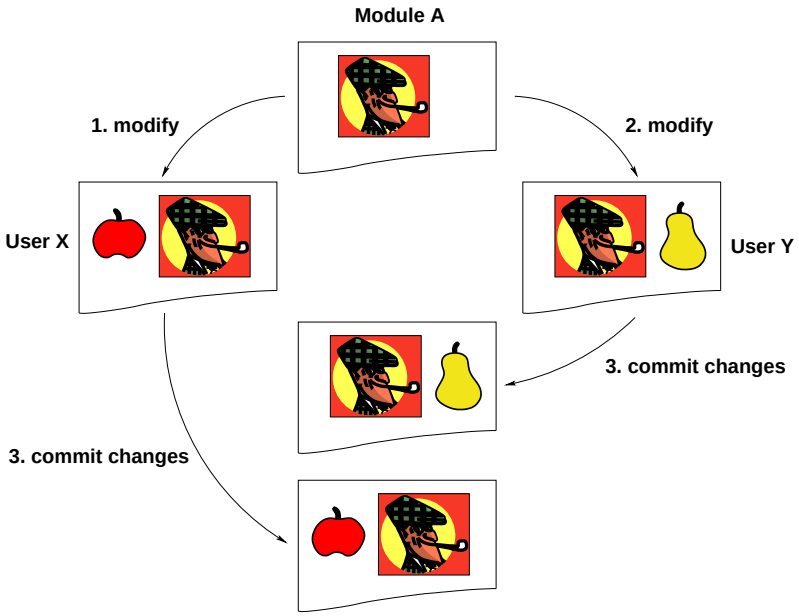
1 Questions récurrentes lors d'un développement de logiciel

- Quels sont les besoins pris en compte par la version 1.0 ?
- Les développeurs travaillent sur la même version
  Comment garantir qu'aucun travail n'est perdu ?
- De quoi est faite la version 1.0 de ce logiciel ?
- Quelles sont les anomalies/demandes de changement prises en compte par la version 2.0 ?
- L'équipe travaille sur la version 2.0. Un client demande une correction sur la version 1.0:
  - Comment développer une version 1.1 ?
  - Comment reporter le travail effectué en 1.1 vers 2.0 ?
- Comment résoudre les problèmes liés au fait que les développeurs sont répartis sur plusieurs sites ?

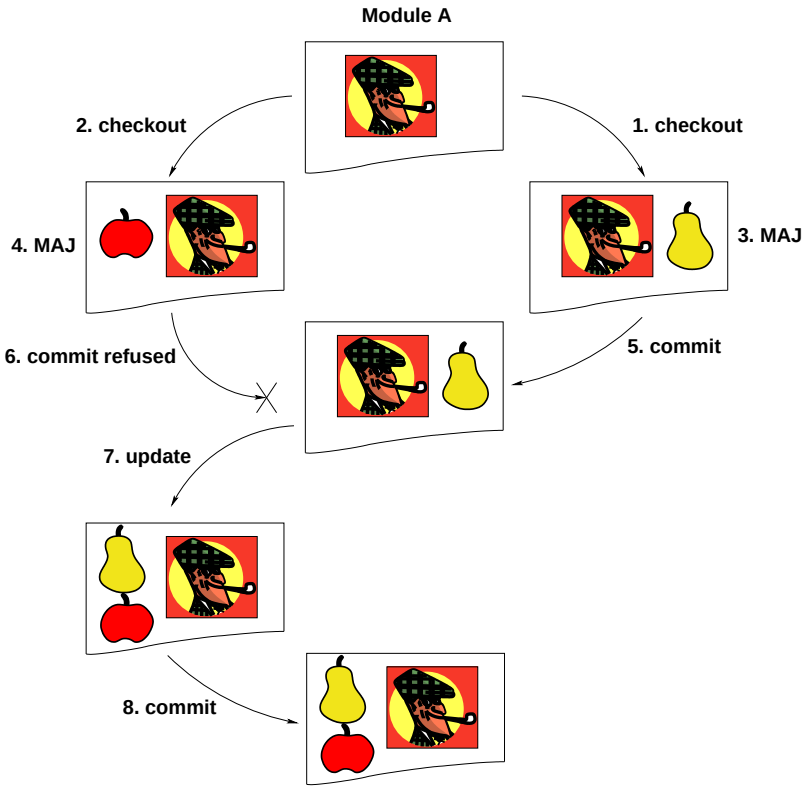
2 Travail en parallèle sur la même version

2.1 Multi-developer—Position du problème

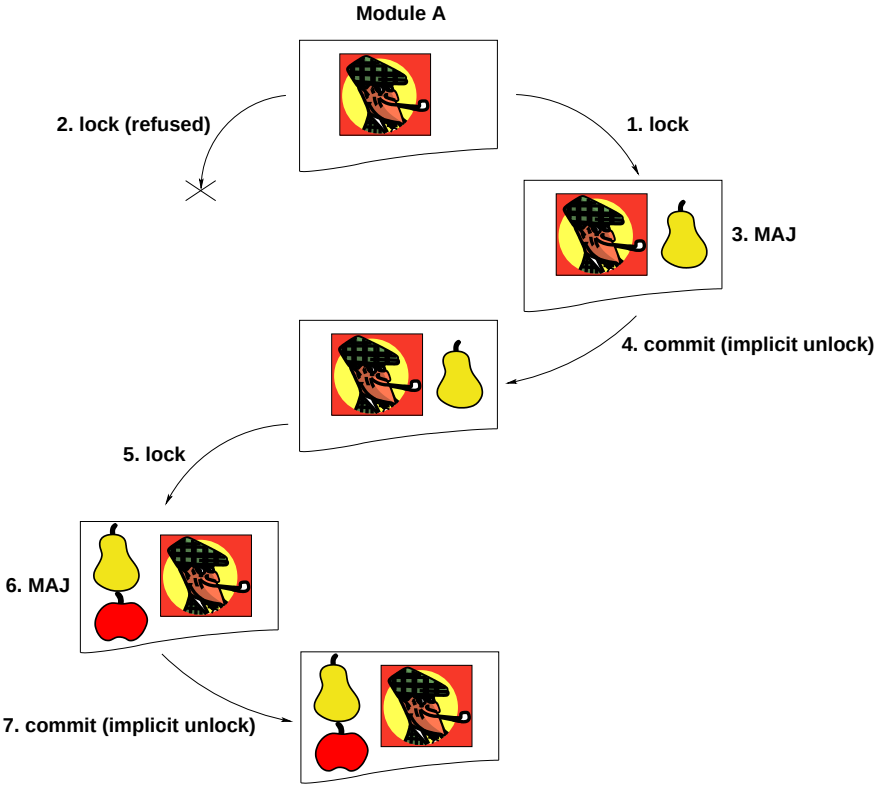
- 2 développeurs peuvent simultanément mettre à jour le même fichier et donc perdre l'une des mises-à-jour



2.2 Multi-developer—Copy-Modify-Merge



2.3 Multi-developer—Lock-Modify-Unlock



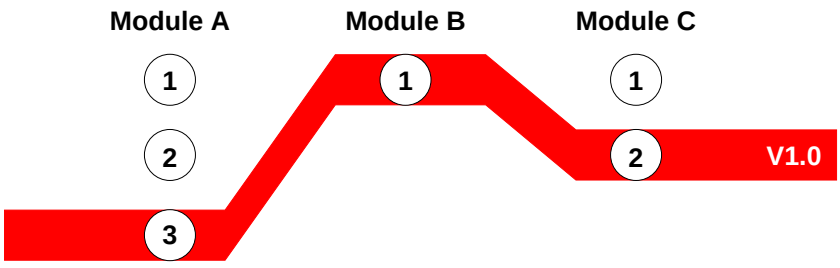
2.4 Discussion

- Le mode Lock-Modify-Unlock n'est imposé que pour certains types de fichiers
- Sinon le choix entre les 2 modes est une question de... choix :-)
- Quelques constantes:
  - Affecter la responsabilité d'un module à une personne
  - Si conflits fréquents sur un module, le scinder en morceaux plus petits
  - La communication/synchronisation entre développeurs est fondamentale

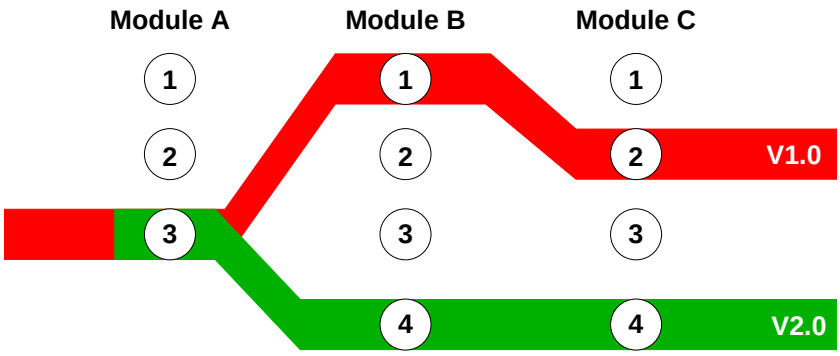
3 Travail en parallèle sur plusieurs versions

3.1 Multiversion—Problem statement

- Développements en parallèle
- Reports d'une version à l'autre

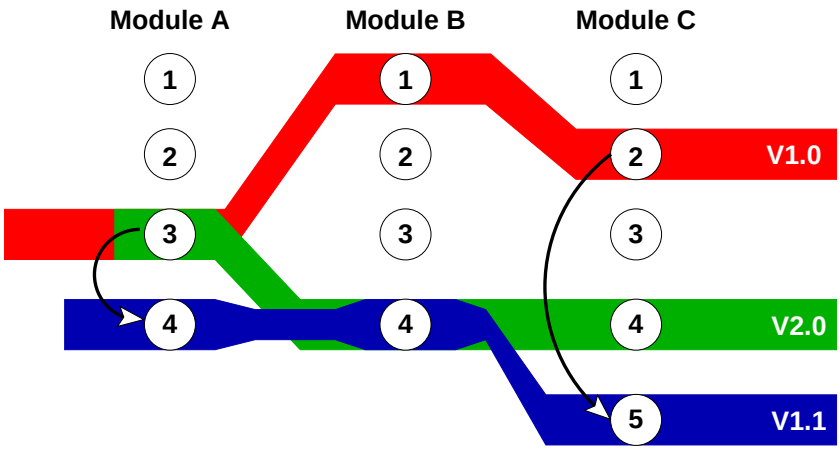


3.1 Multiversion—Problem statement



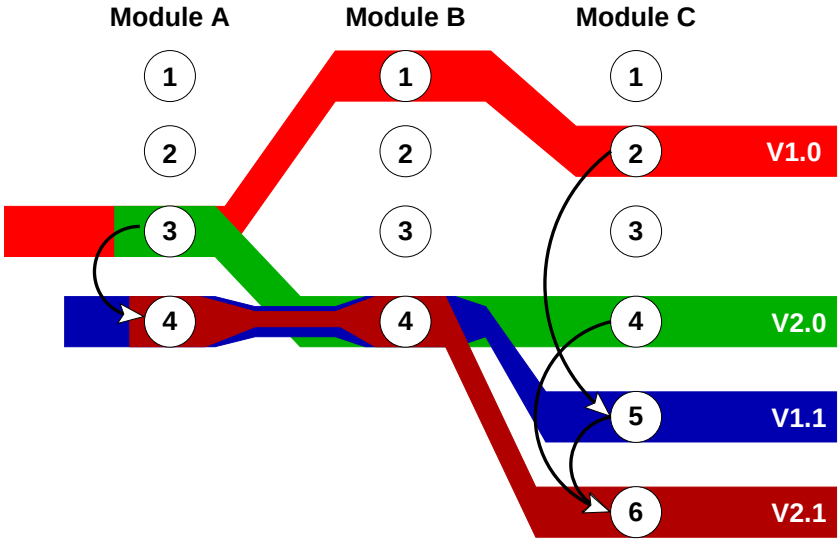
Source control management with Subversion 3 Travail en parallèle sur plusieurs versions

3.1 Multiversion—Problem statement



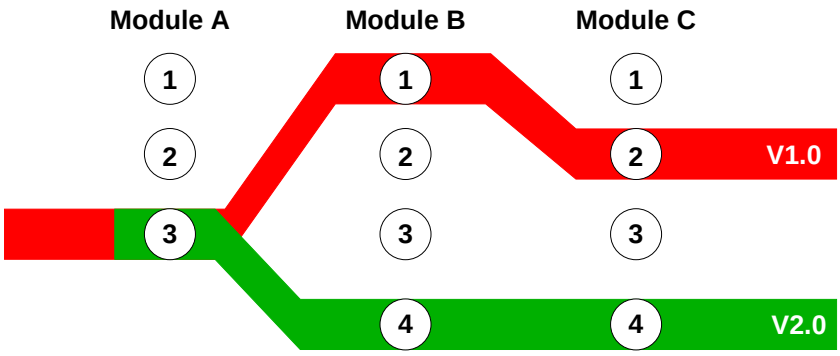
Institut TELECOM, SudParis — D. Conan and M. Simatic — February 2008 — Internship course 10/28

3.1 Multiversion—Problem statement



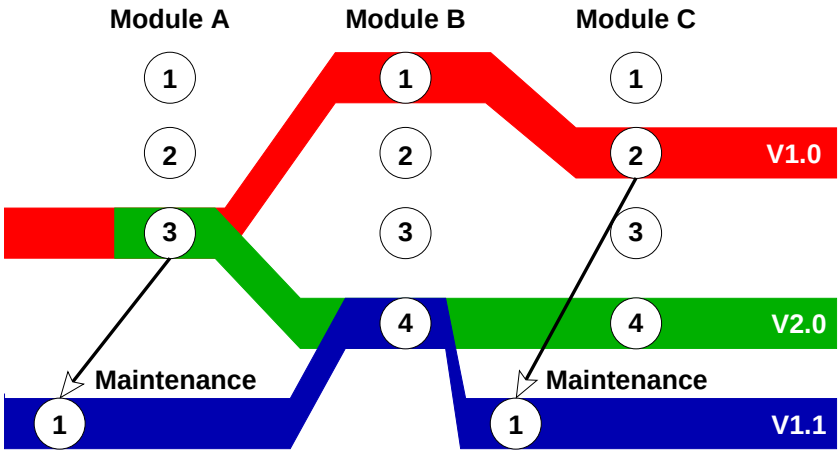
3.2 Multiversion—Éléments de réponse

- Étiquette (“label” ou “tag” sur une version)



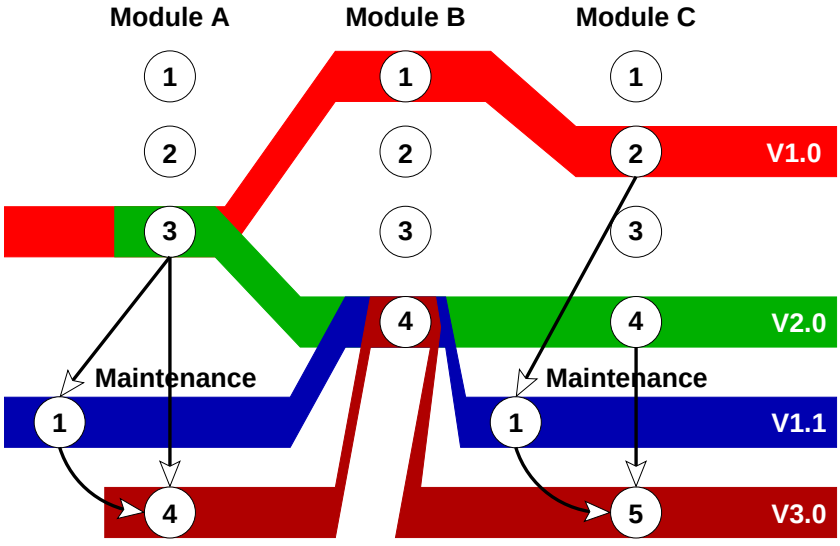
3.2 Multiversion—Éléments de réponse

- Branches



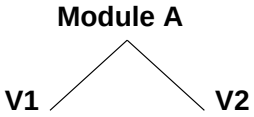
3.2 Multiversion—Éléments de réponse

- Merge

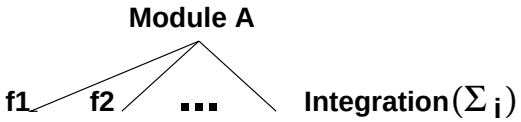


3.3 Multiversion—Discussion

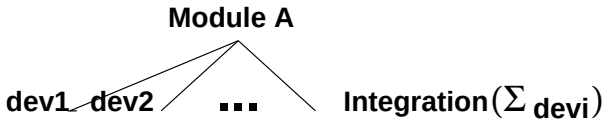
- Why making branches?
  - 1 branch per version?



  - 1 branch per logical function?



  - 1 branch per developper?



- Depending upon the software developed!

4 Presentation of Subversion

4.1 References and principles

- GNU/Linux package or Windows TortoiseSVN (http://tortoisesvn.tigris.org/)
- Documentation
  - Web site: http://subversion.tigris.org/
  - Free download of the svn book [Collins-Sussman et al., 2007] http://svnbook.red-bean.com/, see sanbotest/trunk/Docs
- For those who are accustomed to using CVS, CVS to SVN Crossover Guide: http://svn.collab.net/repos/svn/trunk/doc/user/cvs-crossover-guide.html
- Repository *Vs.* personal working copy, and record/track changes over time
- Enable collaborative editing and sharing of files
  - Copy-Modify-Merge solution
    - Private work area: svn will never incorporate other people's changes, nor make your own changes available to others, until you explicitly tell it to do so
    - You can have multiple working copies of the same project
    - Administrative area in .svn directories which contain meta-data
  - Lock-Modify-Unlock solution: Provided in svn, but not shown here

4.2 Checking out

- Get a working copy
- Main repository access URL
  - http://: Access via WebDAV protocol to svn-aware Apache server
  - https://: Same as http://, but with SSL encryption
  - svn://: Access via custom protocol to an svnserve server
    - *E.g.*, `svn checkout svn://svn.forge.objectweb.org/svnroot/fractal`
  - svn+ssh://: Same as svn://, but through an SSH tunnel
    - *E.g.*, `svn co svn+ssh://conan@picoforge.int-evry.fr/sandbotest`
- Standard directory structure
  - trunk: The "main line" of development
  - branches: Branch copies —*i.e.*, secondary line of developments
  - tags: Tag copies —*i.e.*, previous distinguished revisions
  - sandbox: Developers' sandboxes—*i.e.*, "kind of private" area before contributing to branches or the trunk

4.3 Publishing or committing or checking in

- Report your local changes to the central (public) repository
- Either specify which files to commit
  Or rely on `svn` to know which files and directories need to be committed (using meta-data in the `.svn` directory)
  - Atomic transaction: All the changes or none of them
    - In face of program crashes, systems crashes, network problems...
- *E.g.*, `svn commit -m"message..." File.java`
- Each commit creates a new revision number (which applies to the entire tree)
  - The last revision is called the `HEAD`

4.4 Updating

- Bring your working copy into sync with the latest revision in the central repository
- *E.g.*, `svn update SomeDir`
  - The most common "update codes"
    - `U`: No local changes, so updated with public changes from the repository
    - `G`: Local changes successfully merged with public changes
    - `C`: Conflicts —*i.e.*, local changes overlap public changes
- Attempt to merge the public changes with the local ones
  - If not possible, `svn` leaves it to the user to resolve the conflict
    - `svn` places three extra unversioned files in your working directory
      - ★ `filename.mine`: working file before updating
      - ★ `filename.r`*`OLDREV`*: revision file before local changes (in `.svn`)
      - ★ `filename.r`*`NEWREV`*: `HEAD` revision of the repository
    - `svn` does not allow a commit until these three temporary files are removed
      - ★ Resolve the conflict and next apply `svn resolved filename`
        $\Longrightarrow$ `svn resolved` removes these temporary files

4.5 Adding a file or directory

- Schedule a file or directory to be added *at next commit*
- The unversioned file or directory to add is in versioned tree of files
- *E.g.*, `svn add File.java`, `svn add NewDir`
- `NewDir` $\Longrightarrow$ by default, recursive addition
  - `-N` to only add `NewDir` —*i.e.*, non-recursive
- `svn mkdir NewDir` = (`mkdir NewDir ; svn add NewDir`)

4.6 Deleting or removing a file or a directory

- Schedule a file or directory to be deleted *at next commit*
- Files and directories that have not been committed are immediately removed from the working copy
- Files and directories locally changed are not removed
  - Either commit first or force (option `-force`)
- Of course, nothing is ever totally deleted from the repository
  - Just removed from the `HEAD`
- *E.g.*, `svn delete File.java` or `svn remove File.java`

4.7 Copying a file or a directory

- Create a new item as a duplicate and automatically schedule it for addition
- The new item's history is recorded as having originally come from the original item
- The two items are in the same repository —*i.e.*, no cross-repository copying
- *E.g.*, `svn copy Original.java New.java`

- The easiest way to "tag" a revision = `svn copy` the `HEAD` revision into the directory `tags`
  - *E.g.*, `svn copy trunk tags/NewTag`
- The easiest way to make a branch = `svn copy` the `HEAD` revision into the directory `branches`
  - *E.g.*, `svn copy trunk branches/NewBranch`

4.8 Moving or renaming a file or a directory

- Equivalent to a `svn copy` followed by `svn delete`
  - The new item scheduled for addition as a copy of the original one at next commit
  - The original item is scheduled for removal at next commit
- Only within a single repository —*i.e.*, no cross-repository moving
- *E.g.*, `svn mv OrigDir/OrigFile.java TargetDir/NewName.java`
  or `svn rename OrigDir/OrigFile.java TargetDir/NewName.java`

4.9 Undoing or reverting changes *

- Revert a working file or directory to their pre-modified state
- You don't need to have any network access to the repository
  - Subversion does this by keeping private caches of versioned file inside `.svn`

$\Longrightarrow$ Undoing = retrieving the cache version in `.svn`

- *E.g.*, `svn revert File.java`, `svn revert SomeDir`

- You can revert items mistakenly added or removed
  - *E.g.*, `svn add File.java ; svn revert File.java`
  - *E.g.*, `svn remove File.java ; svn revert File.java`

4.10 Checking the status of a working file

- File AND directories are versioned

- Different status—reminder:
  - Unchanged and current: `svn commit` and `svn update` will do nothing
  - Locally changed and current: `svn commit` will commit local changes and `svn update` will do nothing
  - Unchanged and out-of-date: `svn commit` will do nothing and `svn update` will fold the latest public changes
  - Locally changed and out-of-date: `svn commit` will fail with an "out-of-date" error and `svn update` will attempt to merge the public changes with the local ones
    - If update not possible, `svn` leaves it to the user to resolve the conflict

- *E.g.*, `svn status File.java`
  - The most common status codes
    - `A`: Scheduled for addition
    - `C`: In a state of conflict (update overlap with local changes)
    - `D`: Scheduled for deletion
    - `M`: Locally changed
    - `?`: Item not already versioned (before a `add`)
    - `L`: Item locked (interruption of a `svn` command due to a problem)
  - `-u` to contact the repository and add information about out-of-date items
- You don't need to have any network access to the repository
  - Subversion does this by keeping private caches of versioned file inside `.svn`

4.11 Displaying the differences and Examining histories *

- Displaying the differences
  - Dig into the details of the changes you have made
    - In the unified diff format (Unix `diff` command)
  - *E.g.*, `svn diff File.java` or `svn diff SomeDir`
  - You don't need to have any network access to the repository
    - Subversion does this by keeping private caches of versioned file inside `.svn`
- Examining histories
  - Display the history of changes up to the revision number of the working copy, that might not be up-to-date
    - *E.g.*, `svn log File.java`
      - ★ Who made the changes, at what revision/time/date, and the log message
      - ★ More information with `-v`: copy, move...
  - Any given version: *E.g.*, `svn cat -r` *9999* `File.java`
  - Files in a directory for any given revision: *E.g.*, `svn list -r` *9999* `SomeDir`

4.12 Basic work cycle with Subversion

- Initial checkout: `svn co repositoryName`
- Begin cycle
  - Update your working copy: `svn update`
  - Make changes: `svn add`, `svn delete`, `svn copy`, `svn move`
  - Examine your changes: `svn status`, `svn diff`
  - Possibly undo some changes: `svn revert`
  - Resolve conflicts (Merge other's changes): `svn update`, `svn resolved`
  - Commit your changes: `svn commit`
- End cycle

4.13 More on Subversion *

- Versioning symbolic links
- Importing = Copying an unversioned tree of files into a repository
  - After the import, the original copy is not converted into a working directory
  - To start working, you need to checkout a fresh copy of the tree
  - *E.g.*, `svn import` and then `svn checkout`
- Exporting = Bundle up your files tree without `.svn` directories
  - *E.g.*, `svn export URL`
- Branches: Copying changes between branches, merging a whole branch to another
- In case of problem, unlocking files with `svn cleanup`
- Changing the repository without having a working copy

References

[Collins-Sussman et al., 2007] Collins-Sussman, B., Fitzpatrick, B., and Pilato, C. (2007). *Version Control with Subversion: For Suversion 1.4*. Creative Commons License.