

VERSION CONTROL SYSTEMS

Anton Kulchitsky

2007

Outline

1. The reason to use them
2. How they work
3. Subversion at ARCS in details
4. Practical example of software development in a group
5. Exercises

Why use it?

1. **Reversion:** If you make a change, and discover it's not viable, how can you revert to a code version that is known good?
2. **Change/Bug Tracking:** You know your code has changed; do you know *who, when and why*? Sometimes: when and where the new bug was introduced?
3. **Branches:** How to introduce a completely new feature or concept and not mess up the working code?
4. **Merging branches:** If I divided the code, how to *merge* new code with good days old code and not mess up?

What we expect from the system then?

1. **Reversion:** It automatically revert the code to the date or named tag if things went wrong or we want to fork from there.
2. **Change/Bug Tracking:** System automatically generates who, when and what in the form we requested.
3. **Branches:** We can branch code as much as we can from the point we like and without worry of merging in the future.
4. **Merging branches:** System also cares about all merges that can be done automatically. It should even understand programming languages to recognize when changes even in the same file do not conflict with each other!

What else?

The system is more reliable, easy to backup, saver to use and harder to destroy. It also improve self-organization. It helps developers to communicate and discuss issues. It can prevent a conflict access to the same document. It can be integrated into IDE or editor. It can cook and wash your dishes after the meal...

Distributed vs. Centralized VC Systems

Distributed (more discuss if left some time):

1. No canonical, reference copy of the code base exists by default; only working copies.
2. Common operations such as commits, viewing history, and reverting changes are fast, because there is no need to communicate with a central server.
3. Each working copy is effectively a remote backup of the code base and change history, providing natural security against data loss.

Subversion concepts

- There is a *repository* that saves initial state of the project, all changes to it, branches, merges and logs.
- There are *working copies* of the code where some state of the code is saved as it is desired to be seen by developers. Developers work on the code in their working copy. One developer can have many working copies.
- To determine what changes are made in the working copy compare to the repository, the developer request a *status* of his working copy.

- To check a detailed difference between any file into the working copy and the file into the repository, or some older version of the file, the developer uses *diff*.
- To make her working copy up-to-date with the repository and to receive last changes from other members of the team or that were made from other working copies, the developer *updates* her working copy before making any changes into the repository.
- The developer *reverts* changes if necessary.
- To submit her changes into the repository, the developer *commits* her changes.

- To make a branch the developer *copies* the trunk to branches and *commits* this copying.
- The developer later can *merge* a branch back into the trunk.

Creation a repository at ARSC

It can be a bit complicated procedure. Though let us practice. We will do this at \$ARCHIVE directory.

1. Create a repository:

```
svnadmin create $ARCHIVE/svntest -fsfs
```

2. Change a permission of \$ARCHIVE so anyone can see permitted directories inside:

```
chmod a+x $ARCHIVE
```

3. Change group of your repository so you group can access it (additional permission actions are requeired):

```
chgrp -R faccamp $ARCHIVE/svntest
```

4. Change permissions of the repository so your group would have permissions same with you (on same files):

```
for perm in 4 6 7;
do find $ARCHIVE/svntest -perm ${perm}00
    -exec chmod ${perm}${perm}0 {} \;
;done
```

5. Set a special bit for all directories so all new subdirectories and files will automatically belong to appropriate group:

```
find $ARCHIVE/svntest -type d chmod g+s {} \;
```

Developers from the group are ready to work with the repository now. Let us make a practical task. We will develop an application together now using Subversion for version control.

[... Practice time...]

“Always Use Source Code Control”

From “The pragmatic programmer” by Andrew Hunt and David Thomas, 1999, an excellent advanced programming book:

“Always. Even if you are a single-person team on a one-week project. Even if it’s a “throw-away” prototype. Even if the stuff you’re working on isn’t source code. Make sure that *everything* is under the source code control — documentation, phone number list, memos to vendors, makefiles, build and release procedures, that little shell script that burns the CD master — everything. ... Even if we’re not working on a project, our day-to-day work is secured in a repository.”

“But my team isn’t using source code control”

From “The pragmatic programmer” by Andrew Hunt and David Thomas, 1999, an excellent advanced programming book:

“Shame on them! Sounds like an opportunity to do some evangelizing! However, while you wait for them to see the light, perhaps you should implement your own private source control.”

Recommended literature

1. Version Control with Subversion,
<http://svnbook.red-bean.com/>
2. Open Source Development with CVS,
<http://cvsbook.red-bean.com/>