MATH 676

–

Finite element methods in scientific computing

Wolfgang Bangerth, Texas A&M University

Lecture 32.5:

Learning to use modern tools, part 5a:

Version control systems (VCSs)

Subversion

Rationale

Version control systems were invented a long time ago to:

- Keep a *history* of changes
- Keep a record of *why* a change was made
- Allow *undoing* a change
- Allow going back to a *defined state in the past*

- As a sort of backup

Later extensions:

- Facilitate collaboration
- Track authorship

Current state

Today, there are essentially two open source systems left:*

- Subversion
- Git

Despite differences, their design shares many commonalities.

(* There are many other open source systems, but they are no longer widely used. There are also many commercial systems.)

The general idea of VCSs

Using subversion (svn) as an example:

- There is a central location where *svn* stores all files of your project
- Anyone with permission can get a copy of these files onto their local drive
- You can modify your local copy
- When done, you upload your version to the central location
- The VCS now stores both old and new versions

Note 1: VCSs always store *all* versions of your project!

Note 2: In reality, project files may be stored in a database instead of files; only *diffs* between versions are stored.

The general idea of VCSs

Using subversion (svn) as an example:

- There is a central location where *svn* stores all files of your project (the "repository")
- Anyone with permission can get a copy of these files onto their local drive ("checking out" a "working copy")
- You can modify your local copy
- When done, you upload your version to the central location (you "commit" your version)
- The VCS now stores both old and new versions

Note 1: VCSs always store *all* versions of your project!

Note 2: In reality, project files may be stored in a database instead of files; only *diffs* between versions are stored.

Version numbers

Using subversion (svn) as an example:

- The repository contains all versions of your project
- Every commit increases the version number by one
- Every commit has an author, a date, and a message

- We can search for commits by author, date and message

- We can check out a particular version to a working copy
- We can update a working copy to a particular version

- We can update a working copy to the current HEAD

Collaborative work

Using subversion (svn) as an example:

- Checking out
- Editing
- Checking in (committing)

- Viewing the history of a file
- Viewing who changed what

- Conflicts

...let's see how this works in practice...

Branching and merging

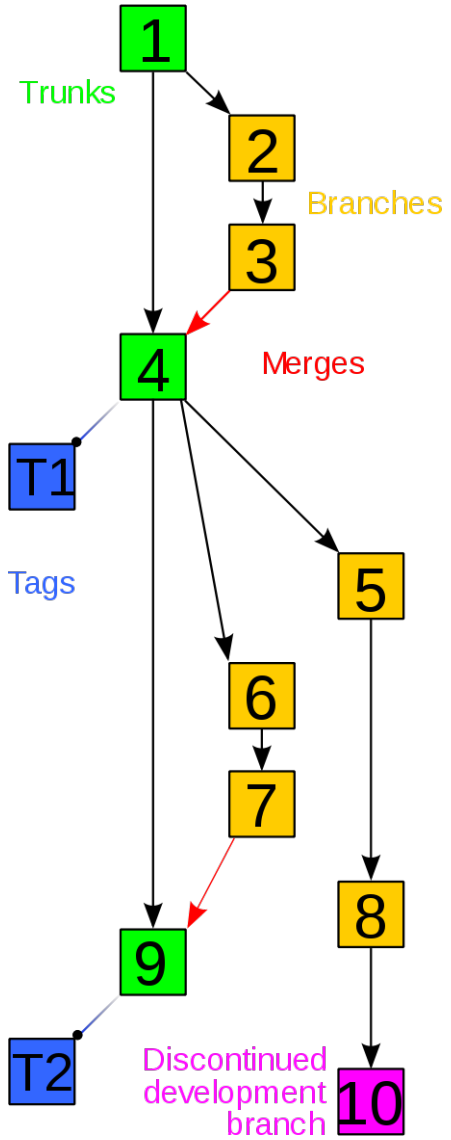
Using subversion (svn) as an example:

- A branch is simply a copy of the main development directory in the repository
- We can *merge* changes that have been made on mainline to the branch
- We can *merge the branch back* to mainline

...let's see how that works in practice...

Mainline, branches and merges

Mainline, branches, HEAD and tags are often visualized as a growing tree:



Note: Revisions are sequentially numbered and can be individually addressed.

(E.g.: “The error was introduced in r32985.”)

Collaborating with others

If you are writing software or papers with others:

- Check out a working copy from the repository
- Edit it (fix bugs, implement features, write text, …)

- Recall that you're working with others:
  - Test your implementation!
  - Document it!
  - Proof read your text!

- If you have write access:
  - Commit your changes
  - Commit all related changes as one revision
  - Include a meaningful commit message

  - Do not include unrelated changes; commit separately

Collaborating with others

If you are writing software or papers with others:

- Check out a working copy from the repository
- Edit it (fix bugs, implement features, write text, …)

- Recall that you're working with others:
  - Test your implementation!
  - Document it!
  - Proof read your text!

- If you do not (yet) have write access:
  - Send a complete patch to someone who does
  - Include a meaningful description
  - Ask them to commit it on your behalf

  - Repeat, after a few times you will get write access :-)

Summary

Do use VCSs!

- For small projects:
  - allows you to work on different machines
  - allows to go back to the "state before the bug"

- For larger projects:
  - preserves history of code (including metadata)
  - allows collaboration
  - allows attribution of authorship

Note: *All* professionally developed software today uses VCSs. Learn how they work by using them!

MATH 676

–

Finite element methods in scientific computing

Wolfgang Bangerth, Texas A&M University