LEIDEN UNIVERSITY MEDICAL CENTER

Version Control

Jeroen F. J. Laros
Leiden Genome Technology Center
Department of Human Genetics
Center for Human and Clinical Genetics

Life without version control

Usually, a project evolves almost organically.

- Start with one program with some sample input data.
- When the first prototype works, we make a copy.
  - `mkdir old; cp * old`
- We continue developing the program and if something breaks, we compare with old versions.
  - `diff myprogram.c old/old/old/old/myprogram.c`

So, what's wrong with that?

Life without version control: disadvantages

A selection of disadvantages.

- No history (except that `old/old` is older than `old`).
  - Modification dates have no history.
  - Modification dates are lost when we copy a project.
  - Files in a project have different modification dates.
- Virtually impossible to maintain when more than one person works on the project.
- Hard to share.
  - Separate source from data before sharing.
  - Separate source from test programs.

Basic work cycle

If you use it just for yourself, it's very easy:

1. Check out:
   - `svn checkout https://myServer/svn/myProject`
2. Work on your project.
3. Commit your changes:
   - `svn commit`
4. Go to 2.

Notice that we only need to give the location of the project once, we only need one command to synchronise.

History

Some nice advantages are present, even if you work alone.

- On a commit, we give a brief description of the changes.
  - Get a complete change log of each file or the whole project.
    - `svn log`
  - Compare files in different revisions, or whole releases.
    - `svn diff`
- We can work from two locations.
  - `svn update`

Maintenance

If we work with more persons on a project, we can expect problems. Version control systems are designed to deal with this.

- Someone has committed a new version while you were working.
  - `svn update`
  - Non-conflicting changes will be merged with your project.
  - Conflicting changes and resolving options.
    - Keep mine.
    - Use the one from the repository.
    - Resolve by hand.

Maintenance

If these problems are structural, there are solutions to work in parallel:

- Create branches to work independently.
  - Branches can be merged.
- Since every revision has an owner, all changes can be traced back to a person (to blame).
- Tags (also branches) can be created for a release version.
  - Usually you only do bug fixes in these branches.

Sharing

Since our repository is on a server, sharing itself is trivial.

- Separation of code and other stuff is inherent to the setup.
  - Files are added manually (`svn add`). Everything else is not synchronised.
  - Files (executables, generated files, . . .) can be ignored explicitly.
- Each repository has read / write permissions.
  - Username / password to have read access.
  - Username / password to write (public read access).

Integration with other frameworks.

Since we use a standard method of storing code, we can use lots of other tools to integrate with.

- Wikis.
- Forums.
- Bug tracking systems.
- Code reviewing.
- ...

[demo]

Questions?