Version Control
(Správa verzí)

http://d3s.mff.cuni.cz

Pavel Parízek
parizek@d3s.mff.cuni.cz
What is it good for?

- Keeping history of system evolution
  - Tracking progress

- Allowing concurrent work on the system
  - Teams of developers
  - Possible conflicts

- Easy reverting to a previous version
  - Safer experimentation
Typical architecture

Source code repository (versioned sources)

Working copy

Working copy

synchronization
Basic usage scenario

1. check-out or update

2. modify & test

3. add & check-in

Source code repository

Working copy
Categories of versioning systems

- Centralized
  - CVS: Concurrent Versioning System
    - The “classic” system
  - SVN: Subversion
    - Currently still used by many open-source projects
      - [http://apache.org/index.html#projects-list](http://apache.org/index.html#projects-list)

- Distributed
  - Git, Mercurial, Bazaar
Branches and merging

- Bug fix in released version
- Main line of development (master branch)
- Concurrent development (experimenting)
- Branching
- Merging

Time & software versions
Conflicts

- Options
  - Postpone resolving
  - Choose version
  - External merge tool
  - and many others

- Conflict markers
  - <<<<<<<<< and >>>>>>>>> in source file

- Three variants of the source file created
Tree conflicts

• Typical cause
  ▪ Renamed files and directories
  ▪ Deleted files

• Solution
  ▪ Make proper changes in the working copy
  ▪ Use patches created with the `diff` command
  ▪ Commit when everything is in a clean state
• Snapshot with a human-friendly name

• Logical copy of the whole source tree
Best practices: synchronizing developers

- Software developed in large teams
  - People may not be always able to coordinate efficiently

- Solution: Copy-Modify-Merge
  - Concurrent modification of source files
  - Resolving conflicts when they happen

- Alternative: Lock-Modify-Unlock
  - The old classic approach (“before internet”)
  - Does not scale well (exclusive access)
  - Not very robust (people forget to unlock)
Best practices: branches and merging

- Use branches for experimental features
- Create special branch for each feature
- Separate release and development branches
  - Propagating bugfixes from development to stable

- Merge often and synchronize with trunk
  - Lower chance of ugly conflicts occurring
  - Smaller conflicts are easier to resolve
  - Commit often ➔ others will have to merge
• Patterns for Managing Source Code Branches
  - [https://martinfowler.com/articles/branching-patterns.html](https://martinfowler.com/articles/branching-patterns.html)
Important features

- Whole source tree versioned
  - Integer numbers (1, 2, 3, ...)
- Mixed versions in the working copy
- Atomic commits
- Versioning for files and directories
  - Operations: move, rename, delete
- Support for binary files
- Disconnected operations
- Metadata in “.svn” directories
Locations

- Repository
  - Remote server (svn+ssh://<network url>)
  - Local directory (file://<absolute path>)

- Working copy
  - Local directory on your computer
Basic commands

- **Help:** `svn help <command>`
- **Create new repository:** `svnadmin create`
- **Create new working copy:** `svn checkout`
- **Update working copy:** `svn update`
- **List modified and new files:** `svn status -v`
- **Show differences between repository and working copy (two versions):** `svn diff -r<version>`
  - Use `svn diff -r<v1>:<v2>` to see differences between two specific versions
- **Add new files into repository:** `svn add`
- **Commit changes:** `svn commit -m “...”`
- **Display information about file:** `svn info`
Few more useful commands

- Undo changes in working copy: `svn revert`
- See full history of a given file: `svn log`

- Importing whole unversioned tree into repository: `svn import <dir> <repo>`
- Exporting content of the repository without metadata: `svn export`
Managing files and directories

- Commands
  - `svn add <path>`
  - `svn delete <path>`
  - `svn copy <path1> <path2>`
  - `svn move <path1> <path2>`
  - `svn mkdir <path>`

- Path
  - In your local working copy
  - Repository (auto-commit)
Branching and merging – commands

- Create new branch
  - `svn copy <main line repo path> <branch repo path>`

- Print differences
  - `svn diff <main line repo path> <branch repo path>`

- Make your branch up-to-date (sync merge)
  - `svn merge <main line repo path>`
  - `svn merge ^/<main line repo dir>`

- Merge branch into the main line (trunk)
  - `svn merge --reintegrate ^/<branch repo dir>`

- Preview
  - `svn merge <repo path> --dry-run`
Undoing committed modifications

- Merge negative version range into local working copy
  - `svn merge <repo path> -r <v1>:<v2>`
  - Note: v1 > v2
- Commit everything
Cherrypicking

- Merge specific change into your branch
  - `svn merge -c <version> <repo path>`
- Commit your branch
Standard repository layout

/trunk
/branches
/tags

/project1/trunk
/project1/branches/feature1
/project1/tags
/project2/trunk
/project2/branches
/project2/tags/R_1_0
/project2/tags/R_1_2_1
Revision keywords

- HEAD
  - Latest version in the repository

- BASE
  - Revision number in the working copy (before modifications)

- COMMITTED
  - The latest revision in which the item changed and was committed (not larger than BASE)

- PREV
  - Equal to COMMITTED-1
Properties

- Standard name-value pairs
- Many internal system properties
  - `svn:ignore`, `svn:eol-style`, ...

Setting property value

- `svn propset <name> <value> <path>`
- `svn propset <name> -F <file> <path>`

Other commands

- `svn proplist`
- `svn propget`
- `svn propedit`
Locks

- Still needed to work with binary files
  - Merge not supported for concurrent modifications

- Locking
  - `svn lock`

- Unlocking
  - `svn commit`
  - `svn unlock`

- Inspecting
  - `svn info`
GUI clients for SVN

- Tortoise SVN (Windows)
  - http://www.tortoisesvn.net

- Eclipse IDE

- Other
  - kdesvn (Linux)
  - svnx (Mac OS)
Links

- http://subversion.apache.org

- SVN Book
  - http://svnbook.red-bean.com