Version Control Systems

Jan Faigl

Department of Computer Science
Faculty of Electrical Engineering
Czech Technical University in Prague

Lecture 14

B0B36PRP – Procedurální programování

Overview of the Lecture

- Part 1 – Version Control Systems

  Introduction and Terminology

  Version Control Systems

  SVN - Subversion

  Git

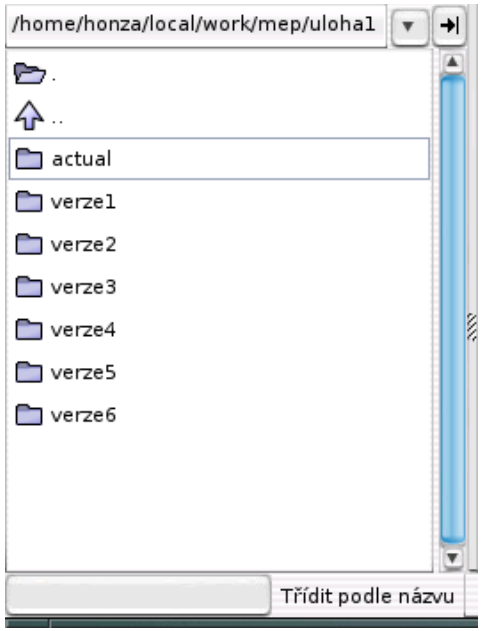
  Versioning

Part I

Part 1 – Version Control Systems (VCSs)

What is Version Control?

- Working on a project or an assignment, we can tend to "backup" our early achievements mostly "just for sure"
  - `hw01`
  - `hw01.backup`
  - `hw01.old`
  - `hw01.old2`
  - `hw01.old3`



- We may try a new approach, e.g., for optional assignment, but we would like to preserve the previous (working) approach
- We may also want to backup the files to avoid file/work lost in a case of hard/solid drive failure *We need to save it to a reliable medium*
- Finally, we need a way how to distribute and communicate our changes to other members of the team

Version Control System

- Version Control System (VCS) is a tool or set of tools that provides **management of changes to files over time**
  - Uniquely identified changes (what)
  - Time stamps of the changes (when)
  - Author of the changes (who)
- VCS can be
  - Manual (by hand) e.g., "save as"
    - Creating multiple copies of files and changes documented in an annotation
    - Backups of the file systems (e.g., snapshots)
    - Files shared between team members
  - Automated version control
    - System or application manages changes
    - Version tracking is managed internally by the system or application
    - It may provide further support for collaboration (team development)

Benefits of Version Control System (VCS)

- VCS provides numerous benefits for both working environment (individual and team)
- Individual benefits
  - Backups with tracking changes
  - Tagging – marking the particular version in time
  - Branching – multiple versions
  - Tracking changes
  - Revert (undo) changes
- Team benefits
  - Working on the same code sources in a team of several developers
  - Merging concurrent changes
  - Support for conflicts resolution when the same file (the same part of the file) has been simultaneously changed by several developers
  - Determine the author and time of the changes

History Overview

- 1972 – Source Code Control System (SCCS) *UNIX*
  - Store changes using deltas
  - Keeps multiple versions of a complete directory
  - Keeps original documents and changes from one version to the next
- 1982 – Revision Control System (RCS) *UNIX*
  - Keeps the current version and applies changes to go back to older versions
  - Single file at a time
- 1986 – Concurrent Versions Systems (CVS)
  - Start as scripts on top of the RCS
  - Handle multiple files at a time
  - Client-Server architecture

Revision Control System (RCS) – Commands

- Create a directory for storing `rcs` files, e.g., `/etc`
- `co -l file` – check out a file and lock it

  Locking by means the file can be checked back in
- `ci file` – check in a revision (put the file under rcs control)
- `rcs -l file` – lock a file already checked out
- `rcsdiff files` – report on differences between files
- `merge files` – merge two files into an original file

  The results has to be checked, it is not a magic!

Revision Control System (RCS) – Example

```
$ mkdir work
$ cd work
$ vim main.sh
$ mkdir RCS
$ ci -u main.sh
RCS/main.sh,v  <--  main.sh
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> My main script
>> ^D
initial revision: 1.1
done
$ ls RCS
main.sh,v
$ echo "echo 'My script'" >> main.sh

$ rcsdiff main.sh
===================================================================
RCS file: RCS/main.sh,v
retrieving revision 1.1
diff -r1.1 main.sh
1a2
> My script

$ci -u main.sh
RCS/main.sh,v  <--  main.sh
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
>> Add the debug message.
>> .
done
```

Terminology – VCS Vocabulary

- Repository – the database storing the files and deltas
- Working (Local) copy of the versioned files
  - An user works with a copy of the versioned files to modify them

We can further distinguish local and working copy of the repository (versioned files) for particular VCS. E.g., subversion in addition to working copy also keeps local copy of the files in the `.svn` directory with the version of the files the developer is currently working on. Git keeps a local copy of the repository in the `.git` directory

- **Trunk** – The primary location for the particular project files in the repository
- **Branch** – A secondary code location (for a variant of the project)
- **Revision** – A version of the a file (or repository)
- **Commit** – Storing a bunch of changes to the repository
- **Revert** – Roll back a commit from the repository
- **Merge** – Pulling changes from one branch into another
- **Conflict** – When a file cannot be merged cleanly (automagicaly)

Repository and Version Control

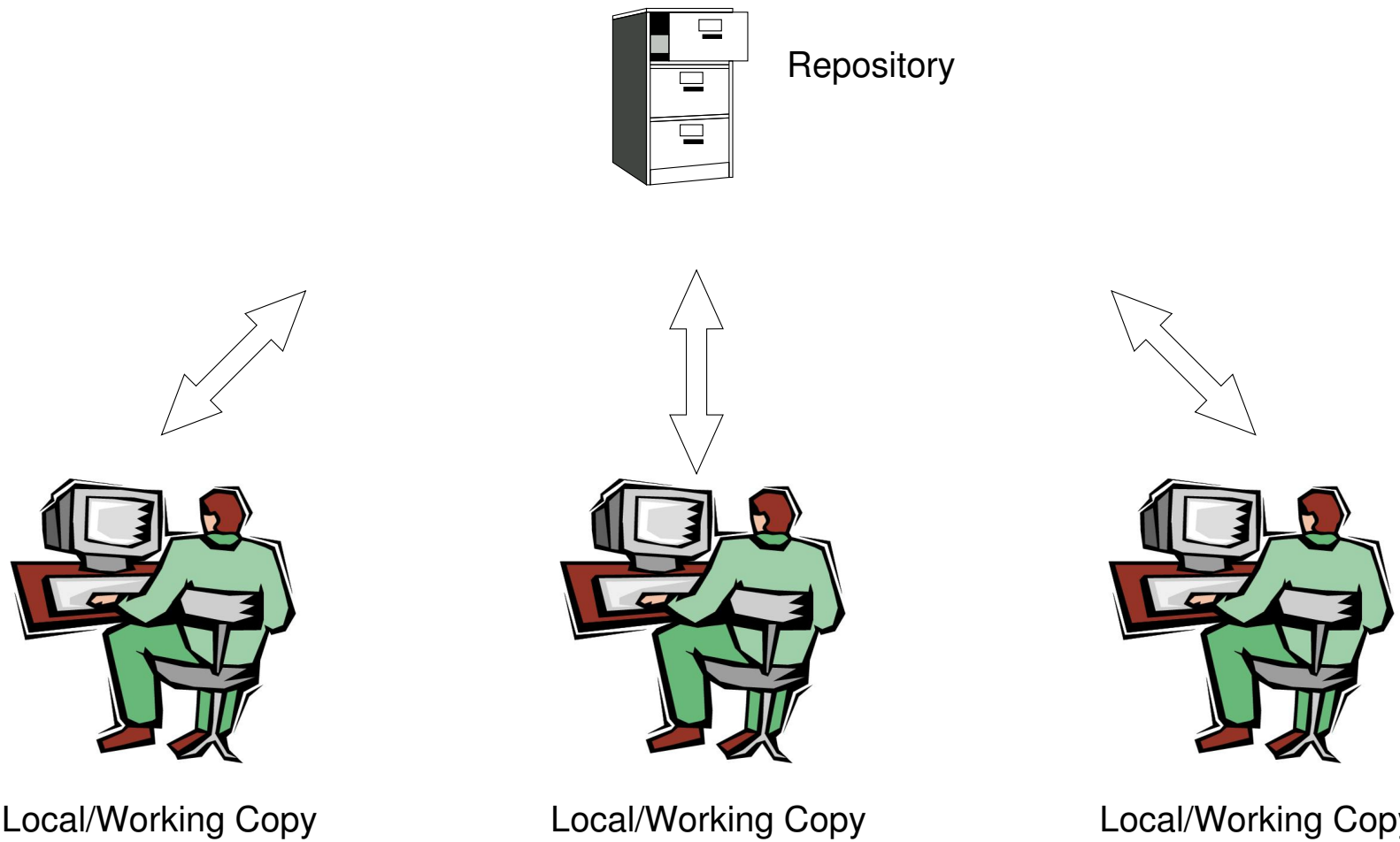
- Version Control System (VCS) is a set of tools (commands) for interaction with the repository and location files (copies of the versioned files)

  Tool is a command or icon or an item in the menu.

- Local command or in the case of the repository also a server service
- **Repository**
  - All changes are stored in the repository

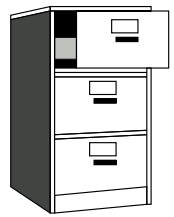
    Usually as deltas, which store differences, and thus save file size
  - Repository can be remote or local

Versioning Files



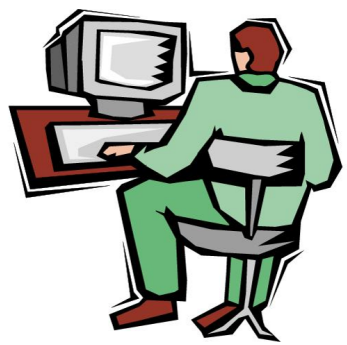
Getting Local/Working Copy – **checkout**

- Create a local copy of the versioned files from the repository
- Directory tree of the local copy usually contains additional files with the information about the versioned files, revisions, and repository, e.g., `.git` or `.svn`
- Then, by modifying checkouted files, we modify the local copies of the particular version of the files



Repository

checkout

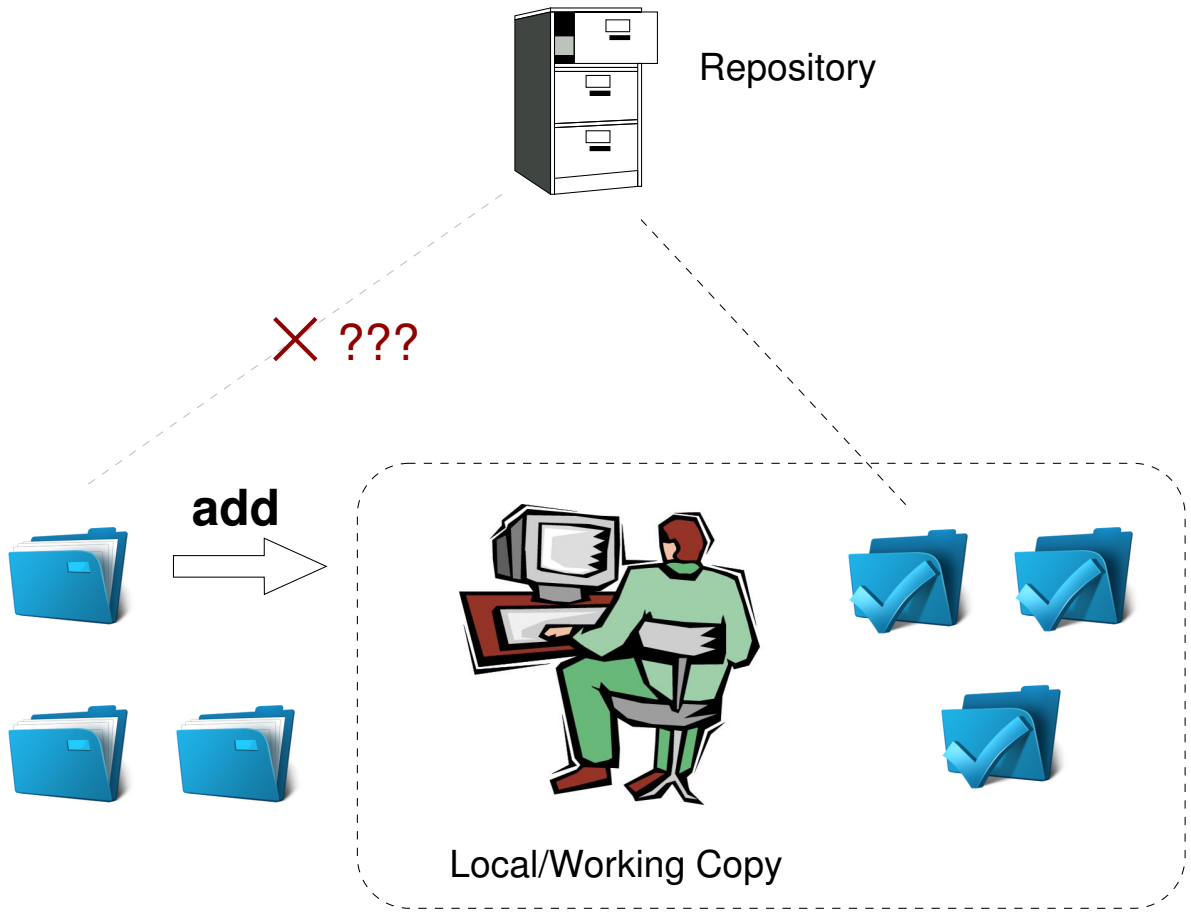


Local/Working Copy

Adding a File to the Version Control – **add**

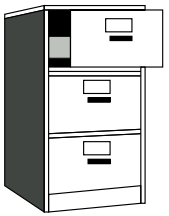
- It is necessary to inform the version control system to track particular files under version control

  Without explicit adding files, the VCS does not know which files we would like to keep under version control and which not.

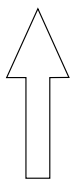


Confirm Changes to the Repository – **commit**

- Request to accept the local modifications as a new revision of the files
- Version control system creates the closest higher version, e.g., with the revision number about one higher
- For the case there is not a newer revision in the repository (according to the local copy of the repository modified locally), changes are propagated to the repository; Otherwise:
  - Update the locally copy of the versioned files to the newer version from the repository
  - If mergers are not handled automagically, it is necessary to handle conflicts



Repository



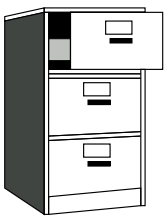
commit

Local/Working Copy

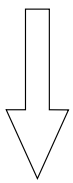
Notice, each commit should be commented by a meaningful, clear, and not obvious comment.

Update the Local Version of the Files from the Repository - **update**

- Update the current local copy of the versioned files from the repository to a newer (or specified) revision from the repository
- If changes of the versioned files is compatible with local modifications, files are automagically merged
- Otherwise it is necessary to manage the conflicts and select the correct version manually



Repository



update

Local/Working Copy

Resolving Conflicts

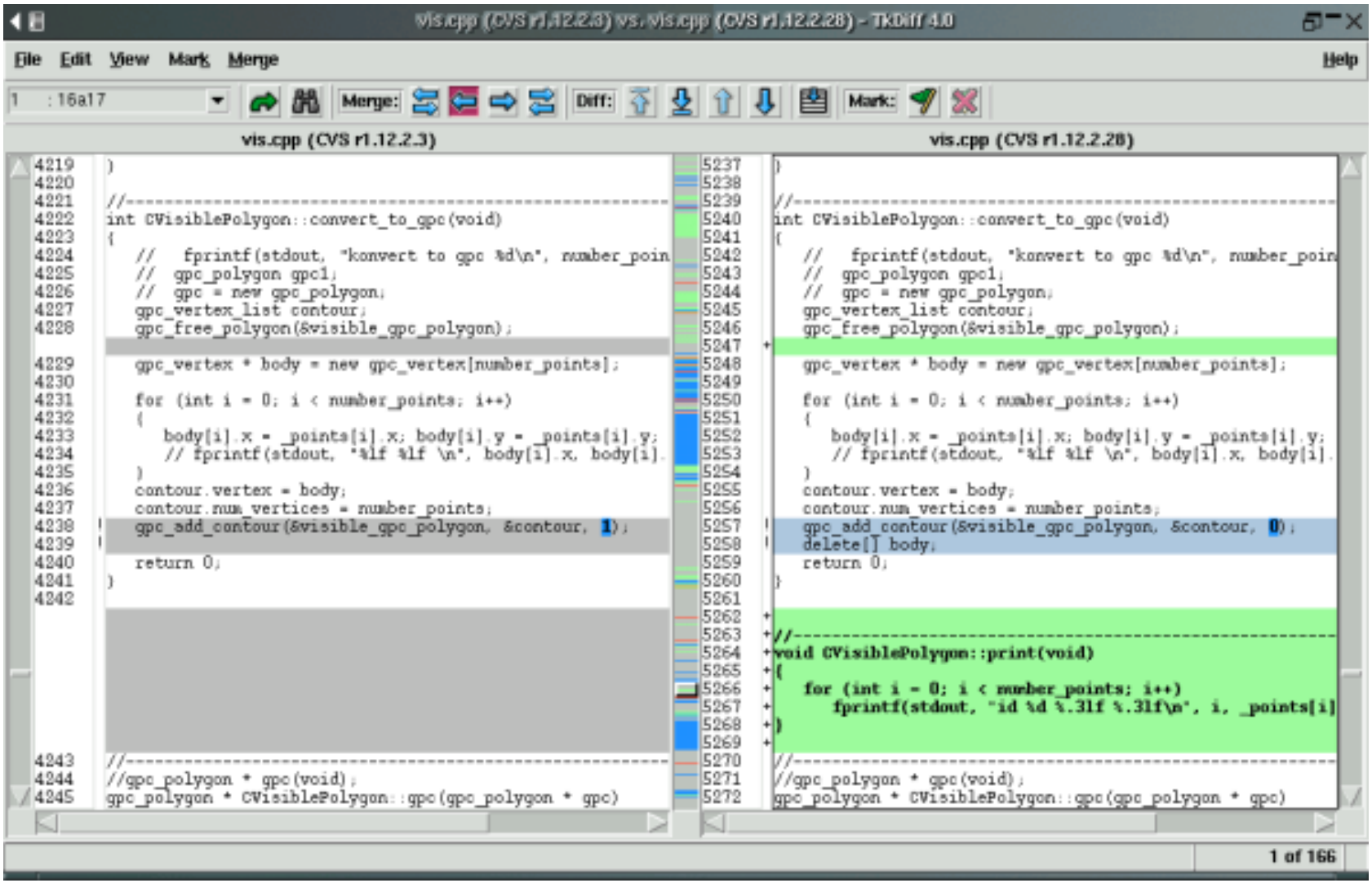
- **Version Control System** (VCS) does not prevent the conflicts, but it provides **tools for resolving the conflicts**
- Conflict is usually caused by simultaneous modification of the same part in the source file
- Conflicts can be avoided by suitable structure of the source files, using modules, and the overall organization of the project files and team work
- Conflicts can be further avoided by specifying access rights to particular files and developers (authorization)

Example of the Merge File with Marked Conflict

- Conflict – incompatible changes of the same lines

```
1169         fprintf(stdout, "%d [%.3lf, %.3lf]\n", i,
1170     }
1171 <<<<<<< vis.cpp
1172     G=12*cities.number;
1173     //G=12.41*4+0.06;
1174 =======
1175     G=12.41*cities.number+0.06;
1176 >>>>>>> 1.12.2.48
1177     separate = false;
1178     return 0;
1179 }
1180
1181 /// ----------------------------------------
1182 int CMap::coords_size(double * min_x, double * m
1183 {
```

Visualization of Differences



Tagging – Point in Time Marking

- VCS keeps the history of the versioned files
- We can label the particular state of the repository in the time by a **tag**, e.g., Release_1.0
- Tag – is a symbolic name for a particular version (state) of the repository
- **HEAD** tag is usually used for the current version of the repository

Branching and Branch Names

- Allows to work in parallel on different ideas / streams / implementations, e.g. for
  - Incremental update to newer techniques and technologies
  - Testing and evaluation of novel approaches before including them into the main product branch

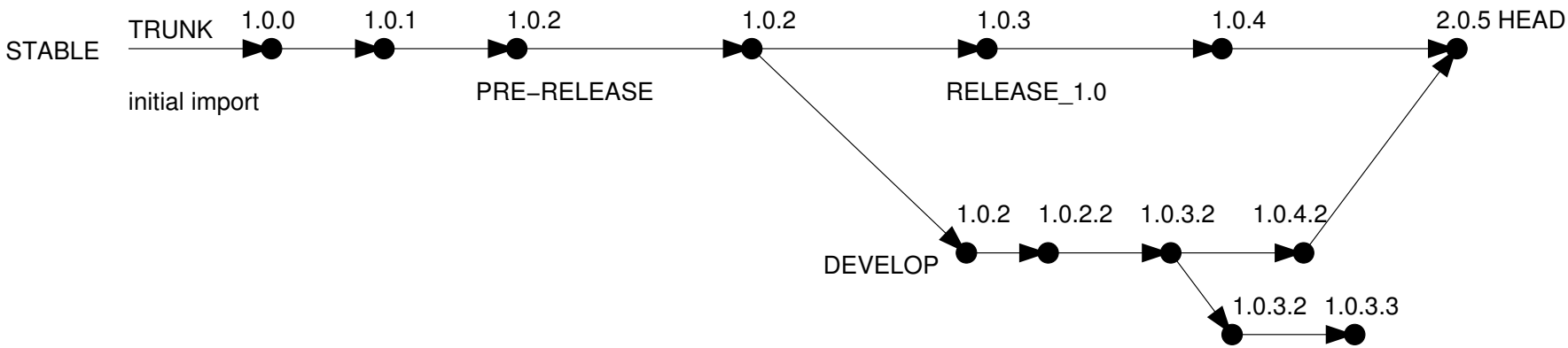
There are common branch names:

- CURRENT, TRUNK - the main development branch
- STABLE - stable development branch

Commit into to the STABLE branch should not disrupt the activities of other developers. E.g., Before merging into the STABLE branch, all changes in API should be propagate to other parts.

Using many branches for the development, branch merge may be the crucial property of the version control system

Example of Branches



Centralized vs. Distributed

Centralized

- Single repository
  - Precisely specified source of record
    Straightforward authorization
    Single point of failure
  - Version ids are usually sequential numbers
    Easy to remember and referenced
    Revision number can be the whole repository
- If repository on the server, it may require network
- Generally less use of branching for experimentation

Distributed

- Every user has a full copy of the repository
  Complicated authorization
  Redundant copies, more robust to failures
  May require unnecessary data space for huge repositories
- Offline work usually possible
  Commit to local repository
- Version ides are usually a GUID (Globally Unique IDentifier)
- More branching and sharing

Example of VCS

- Sometimes may also be called as Source Code Manager (SCM)
- Many VCSs exist as both free/open source and proprietary

  https://en.wikipedia.org/wiki/List_of_version_control_software
- Local only: SCCS (1972), RCS (1982), PVCS[1] (1985), QVCS[1] (1991)
- Client-server: CVS (1986), ClearCase[1] (1992), Perforce[1] (1995), Subversion (2000), Surround SCM[1] (2002), Visual Studio Team Services[1] (2014)
- Distributed: BitKeeper (1998), Darcs (2002), SVK (2003), Bazaar (2005), Mercurial (2005), Git (2005), Plastic SCM[1] (2006), Visual Studio Team Services (2014)[1]

[1] Proprietary

- Free/open-source – **Subversion**, **Git**
- Proprietary – Surround SCM, Plastic SCM

http://www.seapine.com/surround-scm/overview, https://www.plasticscm.com

It is good to known and be aware what systems are available and what are their limitations and features. Knowledge of fundamental principles may help you to make a right choice.

https://en.wikipedia.org/wiki/Comparison_of_version_control_software

Subversion and Git – Main Difference

- **Subversion**
  - Central repository approach – the main repository is the only source and only the main repository has the complete file history
  - Users checkout local copies of the current version
  - It includes authorization to particular directories
  - It revision id is a number for a whole repository
  - Tags and branches are directories (cheap-copy)

    Allows easy and straightforward multiple versions (branches/tags) alongside
- **Git**
  - Distributed repository approach – every checkout of the repository is a full repository with complete history
  - Greater redundancy with higher speed
  - Branching and merging repositories is more heavily used

    Branches and tags are "markers" of the subset of the repository

What the best fits your needs depends on the way how you expect to use it. It also holds for single user usage. Imagine a situation with a single main laptop (do not rely on single HDD/SSD). Or a situation with several workstations and laptops.

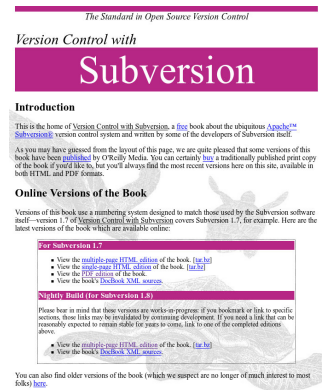
Learn what you need!

Literature

- For both systems Subversion and Git, there are several books also available for download or on-line readings

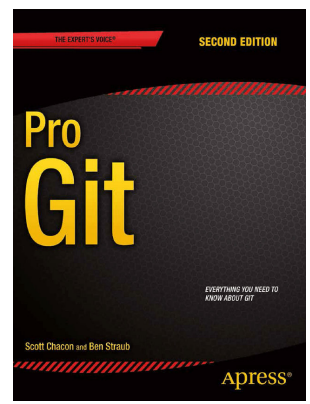
  - **Subversion**

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



  - **Git**

    https://git-scm.com/book/en/v2



Subversion

- http://subversion.apache.org
- Apache Subversion 1.9.5 Release (2016-11-29)
  - Milestone 1 - September 2000,
  - Subversion 0.8 - January 2002,
  - Subversion 0.37 (1.0.0-RC1) - January 2004,
  - Subversion 1.0.0 - February 2004,
  - Subversion 1.1.0 - September 2004,
  - Subversion 1.2.0 - May 2005,
  - Subversion 1.3.0 - January 2006,
  - Subversion 1.4.0 - September 2006,
  - Subversion 1.5.0 - June 2008,
  - Subversion 1.6.0 - March 2009
  - Subversion 1.7.0 - October 2011 (Apache Foundation),
  - Subversion 1.8.0 - June 2012,
  - Subversion 1.9.0 - August, 2015

https://subversion.apache.org/docs/release-notes/release-history.html

SVN – Setting up a repo

- `svnadmin` – administration changes to the SVN repository
- `svn` – for interaction with an SVN repository

  Can be used from other applications / scripts / GUIs or using particular library calls.
- The repository can be setup
  - Locally using local path to the repository

```
svnadmin create /repos/myrepos
svn checkout file:///repos/myrepos my_project
```

    or using ssh account

```
svn checkout svn+ssh://mypc.cvut.cz/repos/myrepos my_project
```

- As a server services using
  - `ssh`
  - `svnserver`
  - `http` and `https` – apache2 `mod_dav_svn_module`.

    Authentication via http(s) seesions, e.g., using LDAP

    Authorization using svn-auth-file

SVN – Commands 1/2

- `svn add files` – schedule files to be added at the next commit
- `svn ci [files]` - commit / check in changed files
- `svn co [files]` – check out
- `svn update [files]` - update local copy to the latest version

  (or specified version using `-r`)
- `svn help [command]` – get help info about a particular command
- `svn status [files]` – get info about the files
- `svn info` – get info about the local the repository and local copy
- `svn diff [files]` – list of changes of the local working files to the local copy
- `svn log [files]` – list commit changes

SVN – Commands 2/2

- `svn revert` *`files`* – restore working copy to the repo's version
- `svn merge` *`source path`* – merge changes
- `svn resolve` *`source path`* – resolve merging conflicts
- `svn resolved` *`files`* – mark the files as conflicts resolved

E.g., after manual editing or using other tools

- Further commands are, e.g., `blame`, `changelist`, `mkdir`, `ls`, `mv`, `lock`/`unlock`, `propset`, etc.
- A file can be removed from the versioning by `svn rm` *`files`*
  - The previous versions of the file are kept in the repository as a part of the history
  - The real deletion of the file is not possible (straightforwardly)

Obliterate feature is planned for Subversion vers. 2.0?

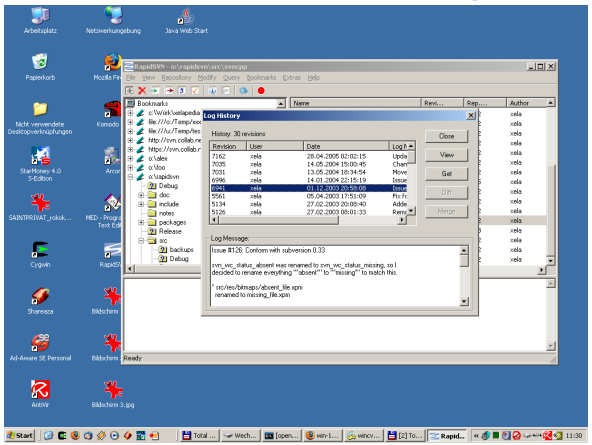
`https://subversion.apache.org/roadmap.html`

Subversion – Example

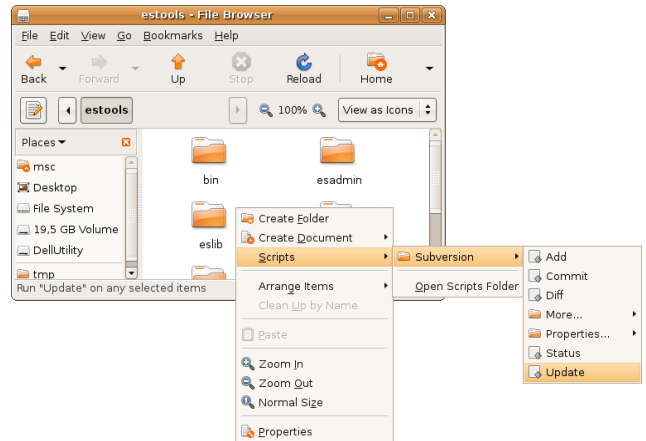
```
mkdir ~/svn
% svnadmin create ~/svn/my_project
% svn co file:///$HOME/svn/my_project
Checked out revision 0.
% cd my_project
% vim main.c
% svn add main.c
A         main.c

% svn ci -m "Add main program"
Adding         main.c
Transmitting file data .done
Committing transaction...
Committed revision 1.

% svn info
Path: .
Working Copy Root Path: /home/jf/my_project
URL: file:///home/jf/svn/my_project
Relative URL: ^/
Repository Root: file:///home/jf/svn/my_project
Repository UUID: 72237e9d-24c5-e611-beef-9c5c8e834429
Revision: 0
Node Kind: directory
Schedule: normal
Last Changed Rev: 0
Last Changed Date: 2016-12-18 14:19:33 +0100 (Sun, 18 Dec 2016)

% svn up
Updating '.':
At revision 1.
```

Subversion – Shell and IDE Integration – Examples

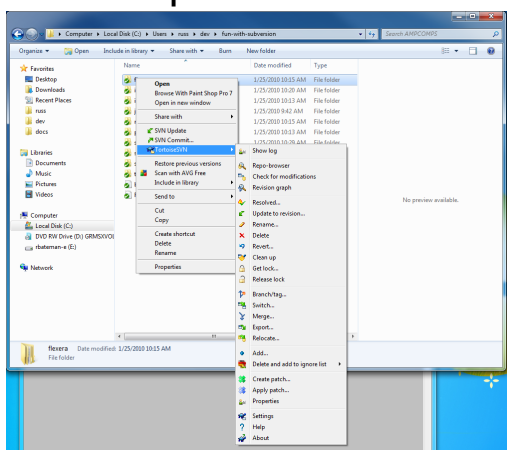
https://en.wikipedia.org/wiki/Comparison_of_Subversion_clients



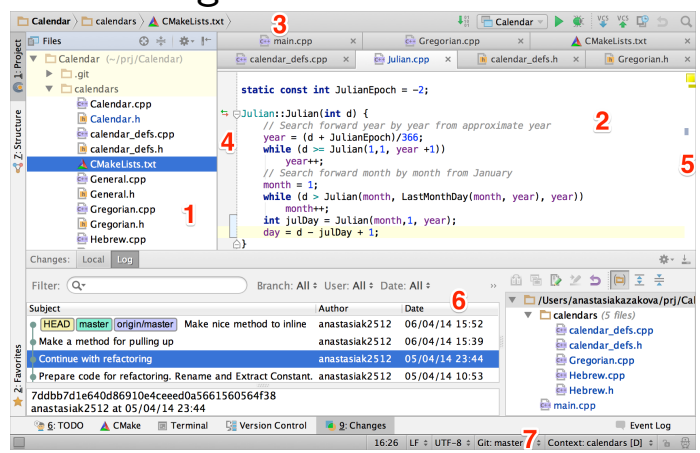
RapidSVN



Nautilus Integration



TortoiseSVN



CLion

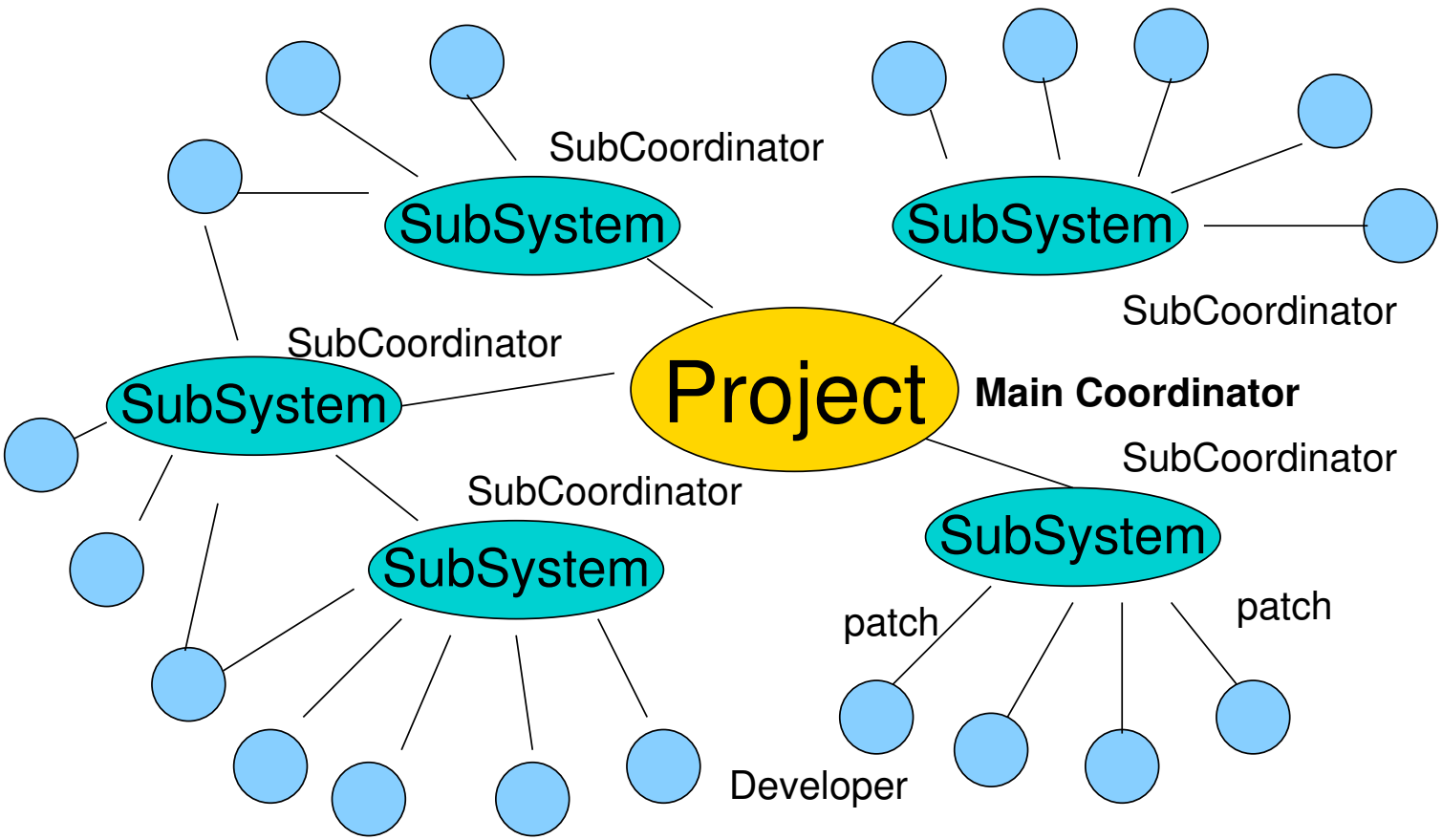
https://tortoisesvn.net/ExplorerIntegration.html

https://www.jetbrains.com/help/clion/2016.1/quick-start-guide.html

Distributed Version Control System (DVCS)

- DVCS does not necessarily have a central repository
- Each developer keeps its own *local* repository
- It is usually very often to use branches
- The final version of the project is a compilation of particular branches of individual developers
- Beside Git, there are several another systems:
  - Bazaar – **bzr**
  - Monotone - http://monotone.ca.
  - SVK – based on Subversion http://elixus.org/
  - Darcs (**darcs**) – David's Advanced Revision Control System http://darcs.net *Written in Haskell*
  - Mercurial – http://www.selenic.com/mercurial/wiki
  - BitKeeper - http://www.bitkeeper.com.
  - Perforce, Plastic SCM – *proprietary software*
  - Git – **git** – created for developing the Linux kernel http://git-scm.com

Git – Development Process of the Linux Kernel Model with a Huge Number of Developers



Git – Properties and Features

- Local repository allows versioning without network connection

  The central repository is substituted by a responsible developer
- Commit only adds the changes to the local repository therefore it is necessary to propagate the changes to the upstream using git push
- Can be efficient for large projects *But it may also not be suitable*
- Files are stored as objects in a database (INDEX)

  SHA1 fingerprints as file identifiers
- Low-level operations on top of the database are encapsulated by more user-friendly interface
- Support development a high usage of **branches**
- Support for applying path sets , e.g., delivered by e-mails
- **Tags** and **Branches** are marked points/states of the repository
- Suitability of the Git deployment depends on the project and model of the development

Git – Usage

- `git clone` *`path to git repository`* – create a copy of the repository (in `.git` directory)
- `git remote` – setup of the repository following (`git fetch`)
- `git help` *`command`* – get help info about a particular command
- `git add`, `git status`, `git log`, `git merge`, `git rm` – commands for developing and local versioning
- `git checkout` *`files`* – update the files from the repository
- `git branch` *`branch name`* – initial a new branch based on the current revision

- `git pull` – update local repository with new revision at the remote repository
- `git push` – propagate local repository to a remote repository

Git – SVN Crash Course

http://git-scm.com/course/svn.html

`git init`	`svnadmin create repo`
`git clone url`	`svn chekout url`
`git add file`	`svn add file`
`git commit -a`	`svn commit`
`git pull`	`svn update`
`git status`	`svn status`
`git log`	`svn log`
`git rm file`	`svn rm file`
`git mv file`	`svn mv file`
`git tag -a name`	`svn copy repo/trunk repo/tags/name`
`git branch branch`	`svn copy repo/trunk repo/branches/branch`
`git checkout branch`	`svn switch repo/branches/branch`

Git – Example

```
% mkdir my_project
% cd my_project
% git init
Initialized empty Git repository in ~/my_project/.git/
% git init
% vim main.c
% git add main.c
% git st
On branch master

Initial commit

Changes to be committed:
  (use "git rm --cached <file>..." to unstage)

    new file:   main.c

% git ci -m "Add main program"
[master (root-commit) ab2afdf] Add main program
 1 file changed, 7 insertions(+)
    create mode 100644 main.c
% git st
On branch master
nothing to commit, working tree clean
% git log
commit ab2afdfc60e7702f1452288c83f97e6a6926e53c
Author: Jan Faigl <faiglj@fel.cvut.cz>
Date:   Sun Dec 18 17:35:23 2016 +0100

    Add main program
```

FEL, GitLab

https://gitlab.fel.cvut.cz

- You can use the provided space for versioning sources of your semester projects and assignments
- After the cloning the repository to your local repository

  You can `push` your changes in the local repository and `pull` modifications from the repository, e.g., made by other developers
- You can also control access to your repositories and share them with other FEL users

  Collaboration with other students on the project

- You need to create your private/public ssh-key to access to the GitLab
- Using server based git repository, you can combine local versioning with server based backup

Wrap-Up – What You Can Put under Version Control?

- Source codes of your programs
- Versioning of the Third-party libraries

  Even though it make more sense to version source files, i.e., text files, you can also versioning binary files, but you cannot expect a straightforward diff.
- Versioning documents (text/binary)
  - File and Directory Layout for Storing a Scientific Paper in Subversion

  http://blog.plesslweb.ch/post/6628076310/file-and-directory-layout-for-storing-a-scientific
- You should definitely put sources of your diploma or bachelor thesis under version control

  Also as a sort of backup

  Even you will use it only for your thesis, TEX or LATEX should be your option.
- Repository and version control as an additional "backuping"

  Repository on the server may usually be located on backuped and reliable disk system.
- Versioning can be used as a tool for sharing files

  Be aware that files are persistent in the repository!

Summary of the Lecture

Topics Discussed

- An overview of history of VCSs
- Fundamental concepts and terminology
- Brief overview of existing VCSs
- Centralized and Distributed VCSs
  - Subversion – commands and basic usage
  - Git – commands and basic usage
- FEL GitLab
- Next: Exam!