Software Engineering
The University of Auckland

Quality Assurance Version Control

Part II - Lecture 7

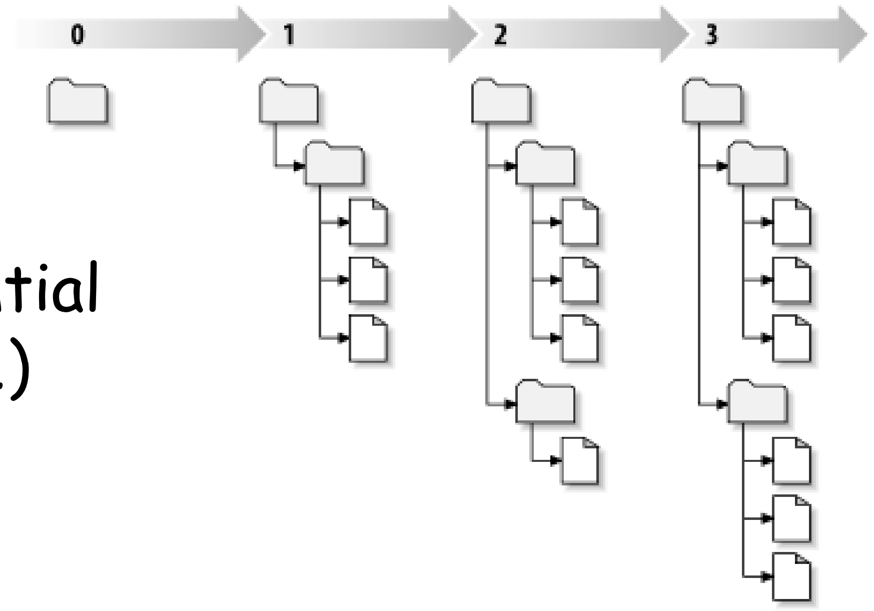
Today's Outline

- Subversion (SVN)
- Version Control Best Practices
- Distributed Version Control

Subversion (SVN)

Subversion (SVN)

- Centralized open-source VCS; started in 2000
- Developed as replacement for the Concurrent Versions System (CVS) which started 1986
- Subversion filesystem versions files and folders ("three dimensional" filesystem)
- Each change creates new revision of the whole file/folder structure
- Revision names are sequential natural numbers (0, 1, 2, …)



Subversion Features

SE Software Engineering The University of Auckland

- Supports **merging** (recommended) as well as **locking**
- Changes are **transactions**
  - Atomic: completely or not at all
    Change is either committed and becomes the latest revision, or is aborted
  - Interrupted commits do not corrupt repository
- **Complete file/folder structure** is versioned, including renames and file metadata
- Delta encoding and merge algorithms work also with **binary data**
- Costs are proportional to change size, not data size
- Works with HTTP server: WebDAV/DeltaV protocol makes it possible to read repository with just a web browser

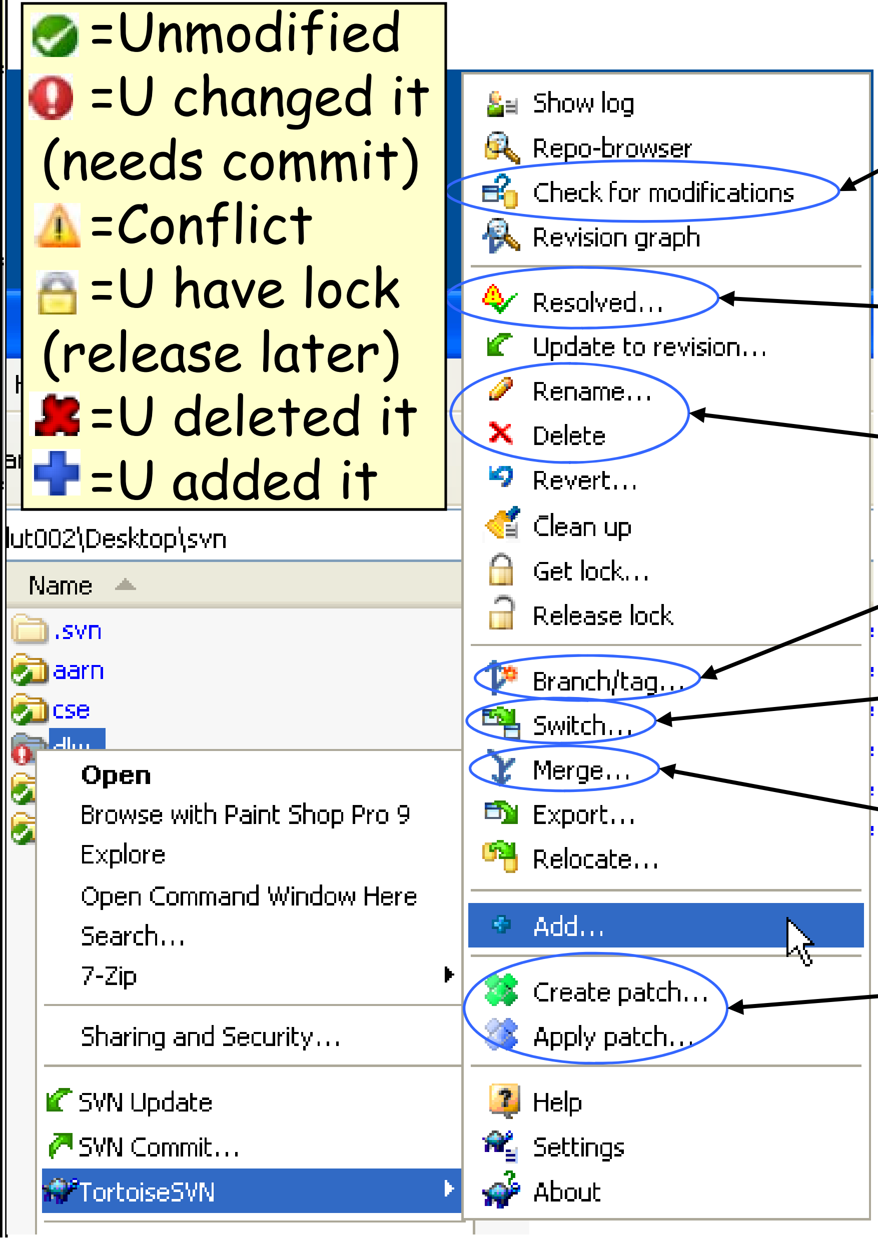
Basic SVN Operations

- **Checkout**: create a working copy of a repository
  - Choose local folder for working copy
  - Enter the URL of the repository
  - Choose the revision to check out (HEAD revision is latest one)

- **Update**: update your working copy to the latest revision
  - If no newer revision exists: no effect
  - If you have changed your working copy:
    latest revision is automatically merged into your working copy
  - Textual merging conflicts have to be resolved manually

- **Commit**: write local changes to the repository
  - Fails if your local revision is out of date; update first
  - Creates a new revision on success

Software Engineering
The University of Auckland



Check if somebody else has modified files or has acquired lock; also for stealing locks

Tell SVN that conflict in files has been resolved

Use these instead of normal ones!!! Also updates version info.

Create cheap copy of a folder.

Switch to the version in a cheap copy (like updating to it).

Merge revision range of branch into other branch.

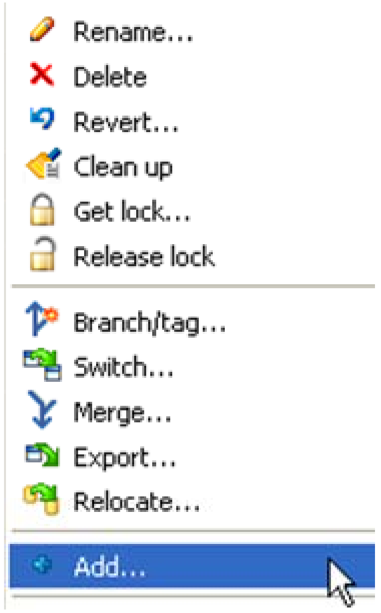
Creating a file containing the local changes or use it to update working copy.

Right-click & drag
= copy/move & update version info

Add, Delete, Rename, Revert

Add file/folder to the repo

- All new files/folders need to be added explicitly to the repo
- Only add source files (e.g. not `.class` files)
  - Other files are generated from them
  - Take up space and are hard to merge

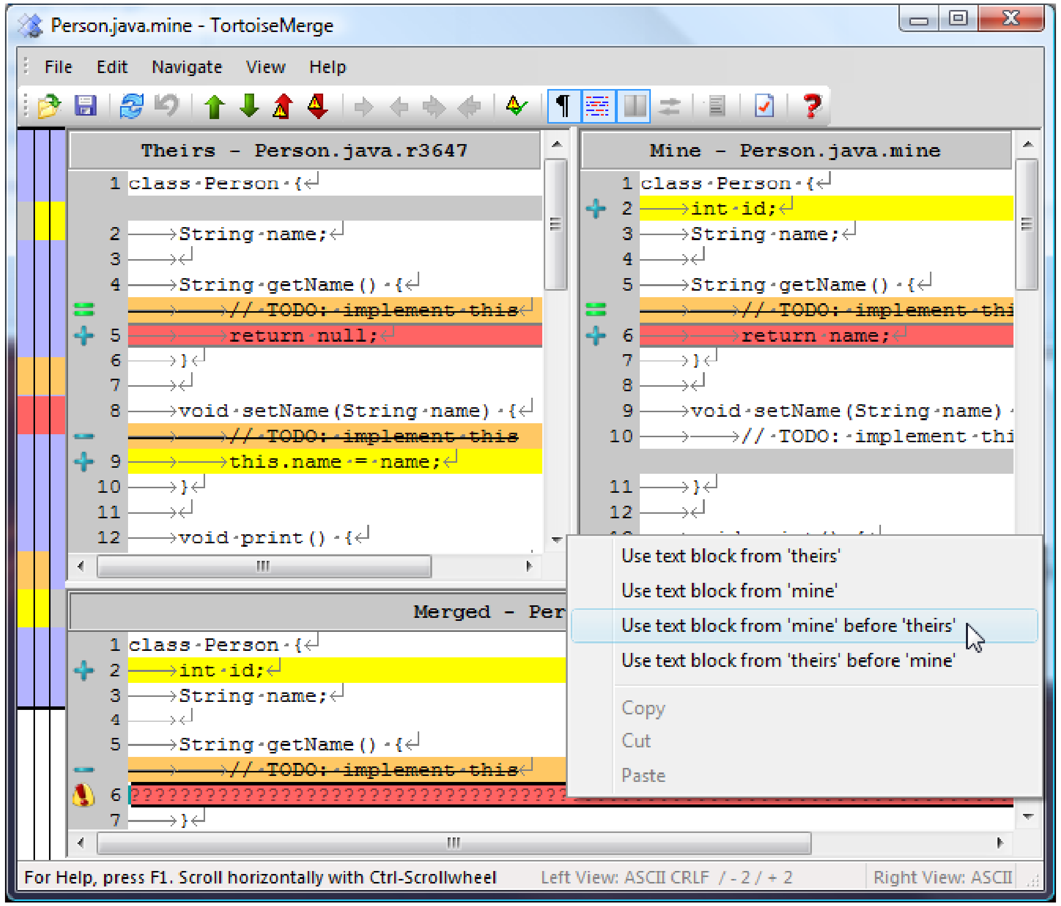


For **deleting** and **renaming** files/folder in the repo use the SVN commands (don't delete/rename directly)

Revert local changes in a file/folder if you want to go back to the last version you got from the repo (i.e. throw away local modifications)

Resolving Conflicts

- After updating SVN might tell you someone committed a change that conflicts with your local changes
- Resolving the conflict means deciding how to merge the conflicting changes
- Supported by editor that shows conflicting changes and gives options to resolve it (e.g. use only one of two changes)
- When conflict is resolved, you must tell SVN


Person.java.mine - TortoiseMerge
File
Edit
Navigate
View
Help
Use text block from 'theirs'
Use text block from 'mine'
Use text block from 'mine' before 'theirs'
Use text block from 'theirs' before 'mine'
Copy
Cut
Paste

Theirs - Person.java.r3647

```
class Person {

    String name;

    String getName() {
        // TODO: implement this
        return null;
    }

    void setName(String name) {
        // TODO: implement this
        this.name = name;
    }

    void print() {
```

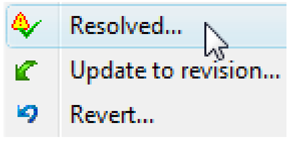
Mine - Person.java.mine

```
class Person {
    int id;
    String name;

    String getName() {
        // TODO: implement thi
        return name;
    }

    void setName(String name)
        // TODO: implement thi

    }

```

Merged - Per

```
class Person {
    int id;
    String name;

    String getName() {
        // TODO: implement this
???????????????????????????????????
    }
```

For Help, press F1. Scroll horizontally with Ctrl-Scrollwheel   Left View: ASCII CRLF / - 2 / + 2   Right View: ASCII

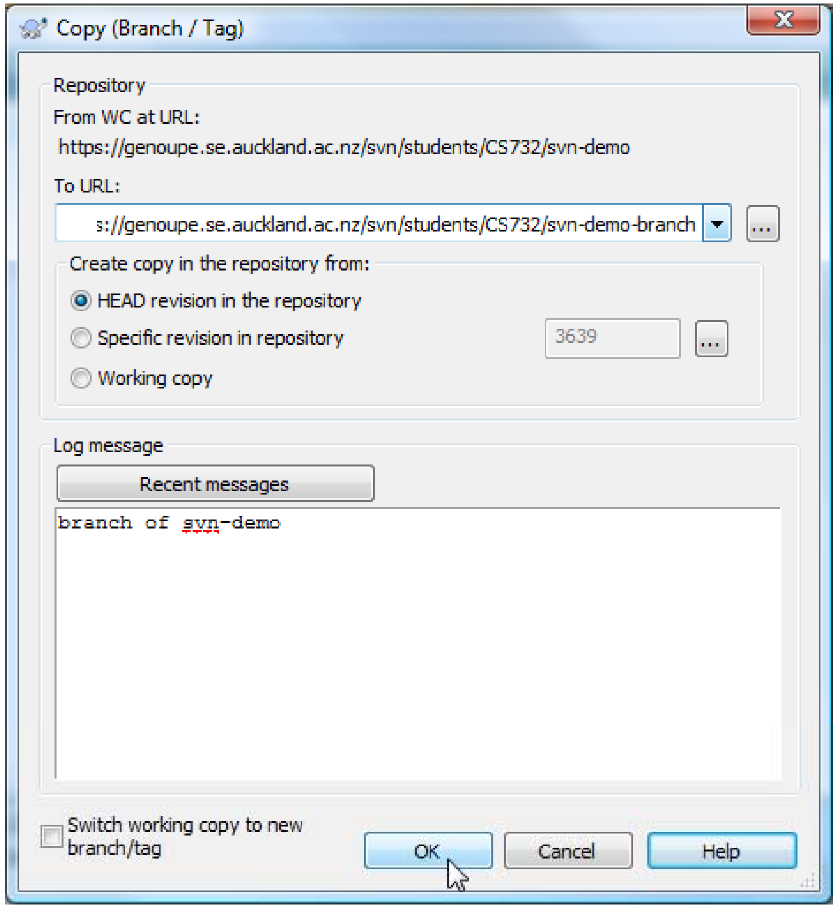


Branching / Tagging

- Creates a copy of a folder in your repository
- **Branch**: the copy will be used for further development
- **Tag**: the copy is just for archival and will remain unchanged

How to do it:

1. Select folder to copy from (right-click on it, use menu)
2. In the dialog: select new folder to copy to
3. Select revision of that folder (usually HEAD)
4. Enter log message
5. Update parent folder of branch in the local working copy



Copy (Branch / Tag)

Repository

From WC at URL:

https://genoupe.se.auckland.ac.nz/svn/students/CS732/svn-demo

To URL:

s://genoupe.se.auckland.ac.nz/svn/students/CS732/svn-demo-branch

Create copy in the repository from:

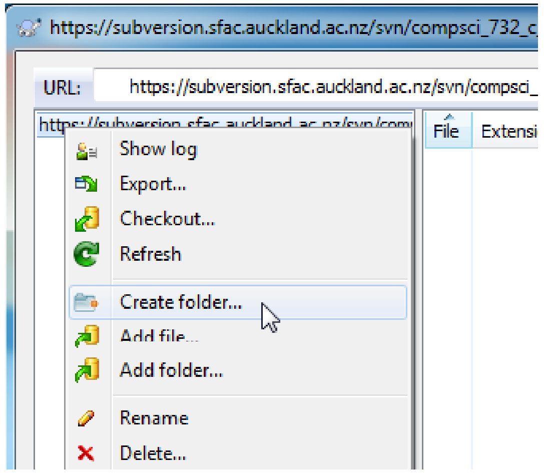
- [✓ HEAD revision in the repository]
- Specific revision in repository 3639
- Working copy

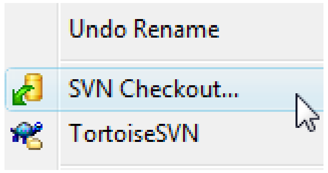
Log message

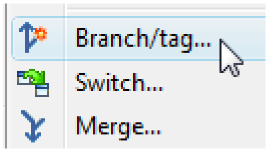
```
branch of svn-demo
```

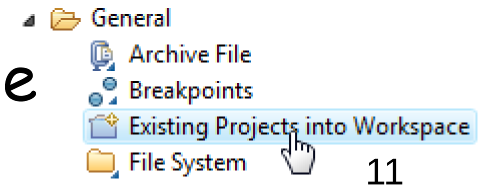
Setting Up a Project in a SVN Repository

1. Use repository explorer to connect to repo and use context menu to create a folder for your project
2. Checkout your own local working copy of that folder (use above URL + your folder name)
3. Add your files, and/or create a branch of existing resources in the repo if your project is based on them
4. Open the project in Eclipse
   - Choose your folder as workspace
   - Import the project into the workspace


https://subversion.sfac.auckland.ac.nz/svn/compsci_732_c
URL: https://subversion.sfac.auckland.ac.nz/svn/compsci_
Show log
Export...
Checkout...
Refresh
Create folder...
Add file...
Add folder...
Rename
Delete...
File
Extensi

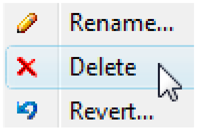


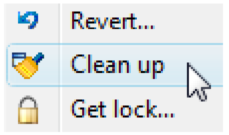




Subversion Tips

1. Don't forget to **add** your files/folders to the repo
2. Delete and rename only using **SVN operations**
3. If two SVN clients are running at the same time, there might be errors like "working copy locked"
4. If something is wrong with working copy, use **cleanup** command
5. If nothing else helps, delete local working copy and check out a new one





Various other clients available,
e.g. Subclipse plugin for Eclipse

Version Control
Best Practices

1. One Change at a Time

Complete **one change** at a time and commit it

- If you committing several changes together you cannot undo/redo them individually
  - Sometimes individual changes are needed
  - Sometimes individual changes need to be excluded

- Continuous integration (see also XP practice)
  - If you make several changes conflicts are much more likely
  - Merging simple changes is much easier

- If you don't commit and your hard disk crashes…
  - Your repository is your backup system
  - Even if the repo is destroyed, other developers will probably have their own local copy

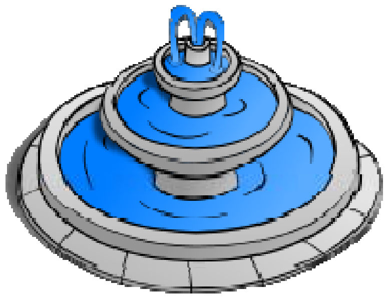
2. Don't Break the Build

- Only commit changes that preserve system integrity
  - No "breaking changes" that make compilation or tests fail
- Test-driven development (see also XP practice):
  - Write tests for every change
  - Run tests before committing (at least some of them)
- Think of others:
  - All other developers will download your changes
  - Any problem that was introduced will suddenly be everyone's problem

3. Only the Source

Commit only **source files**

- I.e. files that are actually necessary for your software (including documentation)
- Not generated files (e.g. **`.class, .exe`**)
- Not temporary files (e.g. irrelevant data or log files)



Why?

- Unnecessary files waste space
(other people need to download them when checking out / updating)
- Most binary files are unmergeable
(easily lead to conflicts that can't be resolved manually)

4. Use the Log

Write a log entry for each change

- What has been changed and why
- Like a short blog post (Twitter style or more)
- Easier to find good and bad changes

Revision	Time	Author	Description
4	1am	CodeCowboy	Added the files
5	1pm	CodeCowboy	More code
6	2pm	CodeCowboy	Minor change

Revision	Time	Author	Description
4	1am	CodeSheriff	Added files from our old repo at http…
5	1pm	CodeSheriff	Added Order.sort() for sorting OrderItems
6	2pm	CodeSheriff	Bugfix for #67: initialized variable

5. Communicate

Communicate with the other developers

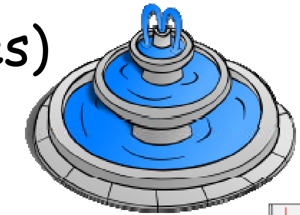
- Before changing existing code
  - See who else is working on it / has worked on it
  - Ask that person about your change before committing (possibly show them a patch)

- Before starting something new
  - Discuss with co-developers and agree on a design
  - Make design proposal, point out design alternatives

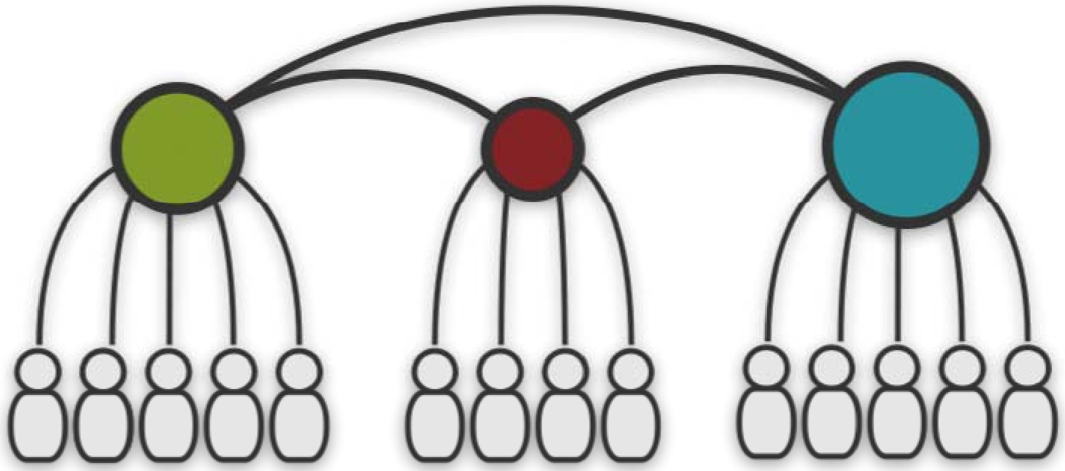
- Always follow the project guidelines & specifications

Version Control
Best Practices (Overview)

1. Complete **one change** at a time and commit it
   - If you committing several changes together you cannot undo/redo them individually
   - If you don't commit and your hard disk crashes…
2. Only commit changes that preserve system **integrity**
   - No "breaking changes" that make compilation or tests fail
3. Commit only **source files** (e.g. not **`.class`** files)
4. Write a **log entry** for each change
   - What has been changed and why
5. **Communicate** with the other developers
   - See who else is working on a part before changing it
   - Discuss and agree on a design
   - Follow the project guidelines & specifications



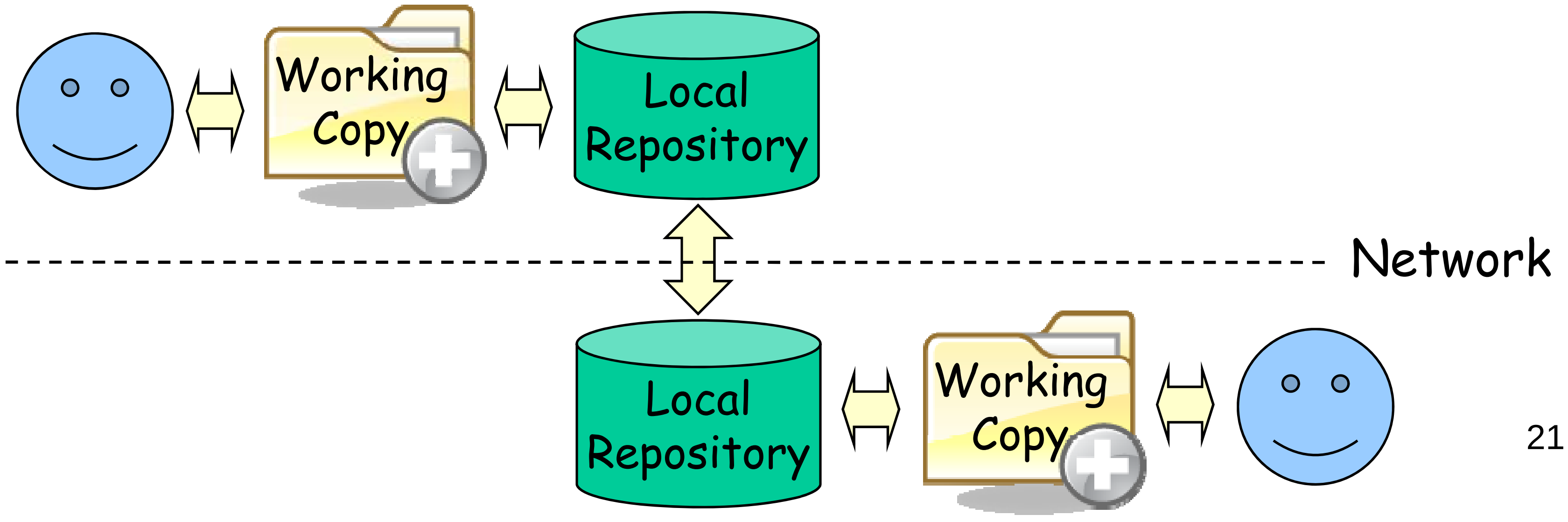
Distributed Version Control



Distributed Version Control

Every developer has their own local repository (a.k.a. "decentralized version control")

1. Developers work on their working copy
2. Developers commit changes of the working copy to their own local repository first
3. Changes can be exchanged between repositories ("pushed" and "pulled")



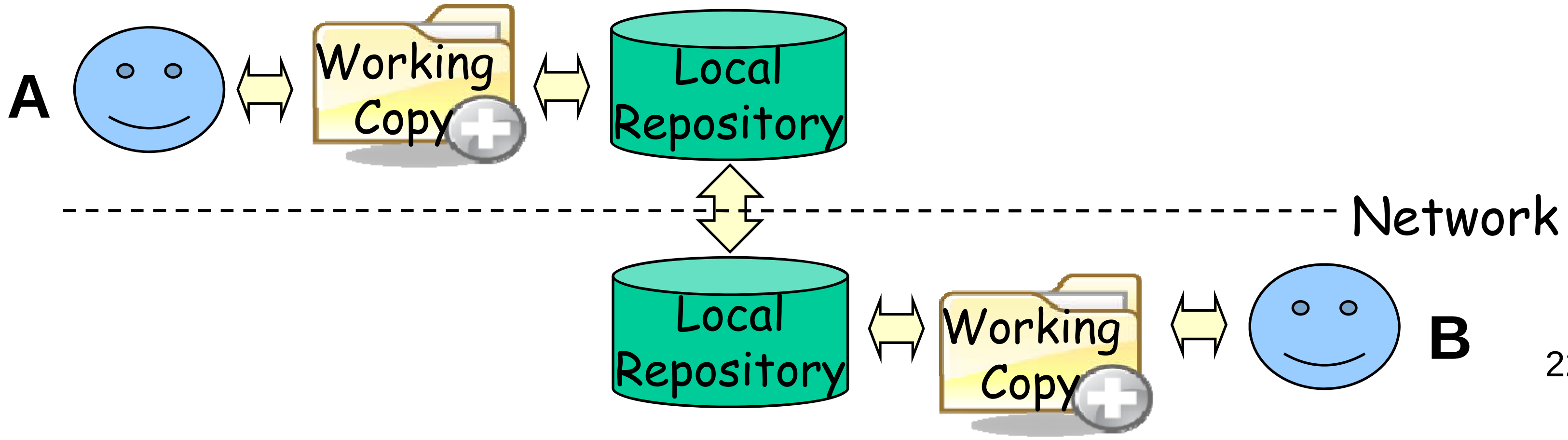
Push and Pull

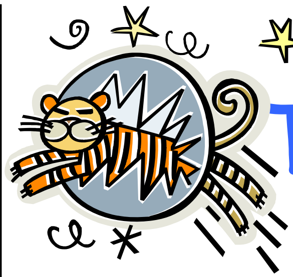
Push

- Once developers have committed versions on their local repository, they can push them to another repo
- Versions are pushed from local branches into corresponding remote branches
- Like "commit" from one repo to another

Pull

- Latest versions are pulled from remote branches and put into the corresponding local branches
- Like "update" from one repo to another





Today's Summary

- **Subversion (SVN)** is a popular centralized VCS
  - Supports merging and locking
  - Checkout, commit, update
- **Version control best practices** are good for harmony & success: One Change at a Time, Don't Break the Build, Only the Source, Use the Log, Communicate
- **Distributed version control systems** give every developer their own repository (version control can be done locally)

References:

B. Collins-Sussman, B.W. Fitzpatrick, C.M. Pilato. Version Control with Subversion. 2008. http://svnbook.red-bean.com/

TortoiseSVN Manual: http://tortoisesvn.net/support

Quiz

Software Engineering
The University of Auckland

1. What are the steps of working with a SVN repository?

2. Explain 3 best practices of version control and describe what could happen if they are not followed.

3. What is the main characteristic of distributed version control systems?