

# Computer Labs: Version Control with Subversion

## 2º MIEIC

Pedro F. Souto (`pfs@fe.up.pt`)

September 16, 2019

# The Problem

```
$edit video_gr.c, make, run, edit, make, run, ...
```

OK! Now that it enters in graphics mode, let's make a backup

```
$copy video_gr.c video_gr.v1.c
```

```
$edit video_gr.c, make, run, edit, make, run, ...
```

OK! Now that it maps graphic memory, let's make another backup

```
$copy video_gr.c video_gr.v2.c
```

```
$edit video_gr.c, make, run, edit, make, run, ...
```

OK! Now that it draws a pixel, let's make another backup

```
$copy video_gr.c video_gr.v3.c
```

```
$edit video_gr.c, make, run, edit, make, run, ...
```

Oops! Does not leave graphics mode, let's retrieve the backup

```
$copy video_gr.v7.c video_grc.
```

Hmm! This is not the version I want. Should it be v3? ... Oops, deleted the last version !@£#%/#\*&\$@

# The Solution? Subversion! (SVN)

- ▶ Subversion is a version control system that is able to:
  - ▶ Keep several versions of an entire (development) directory tree
  - ▶ Restore any of the versions it keeps in a consistent way
- ▶ Furthermore, it:
  - ▶ supports **concurrent access** to the different files or directories in the tree **by several users**;
  - ▶ keeps a log of the changes performed to each file/directory that **can be used to document/keep track of the main changes** between versions
  - ▶ allows to create new **branches**, i.e. to keep track of the evolution of multiple directory trees that have a common ancestor (i.e. a tree of directory trees)

# SVN Key Concepts

**Repository** This is the central store (and the server program) that keeps the **different versions** of the data

- ▶ It usually keeps data for different “projects”
- ▶ We’ll refer to the data of a “project” kept in the repository as ... the repository

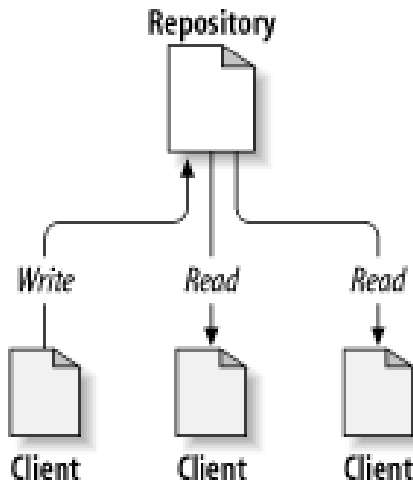
**Working Copy** This is a copy of **one version** of the data of a “project”. The working copy is kept in a client computer, may be the computer that keeps the repository

- ▶ The working copy is a standard directory tree
- ▶ Other programs, like editors and compilers, do not need to be “version-control-aware”

There may be several working copies of a given repository (project), thus supporting the collaboration of multiple programmers in a project

**Revision** A new revision (version) is created by **committing** to the repository the changes done in a working copy

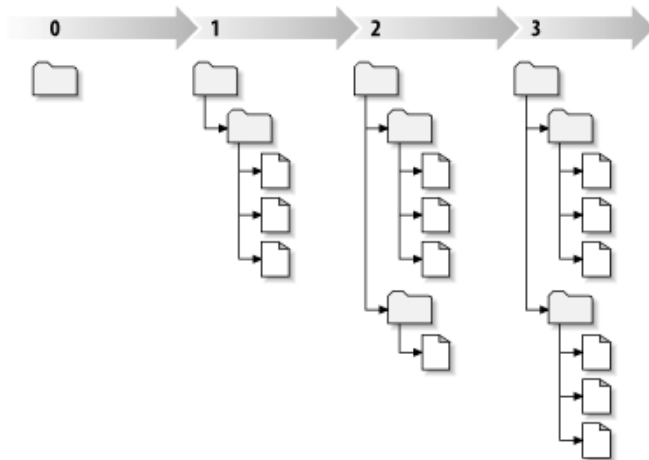
# SVN: The Repository and Working Copies



Source: Ben-Collins-Sussman et al. Version Control with Subversion

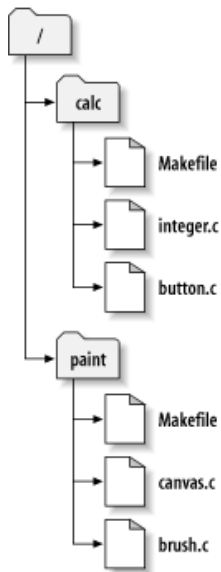
# SVN: Revisions

**Revision** A new revision (version) is created by **committing** to the repository the changes done in a working copy



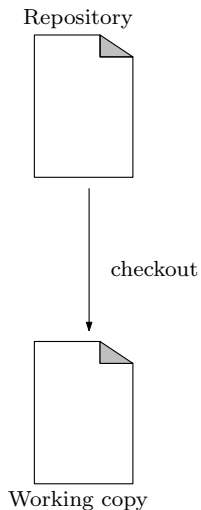
Source: Ben-Collins-Sussman et al. Version Control with Subversion

# SVN: Multiple Projects



# SVN: Basic Usage

- Generate a working copy using **checkout**

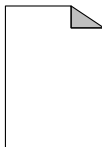




# SVN: Basic Usage

- Change the working copy with your favorite editor

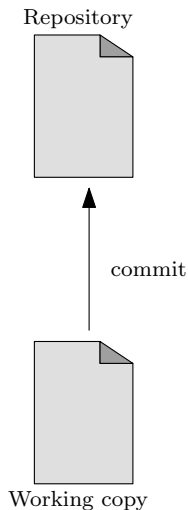
Repository



Working copy

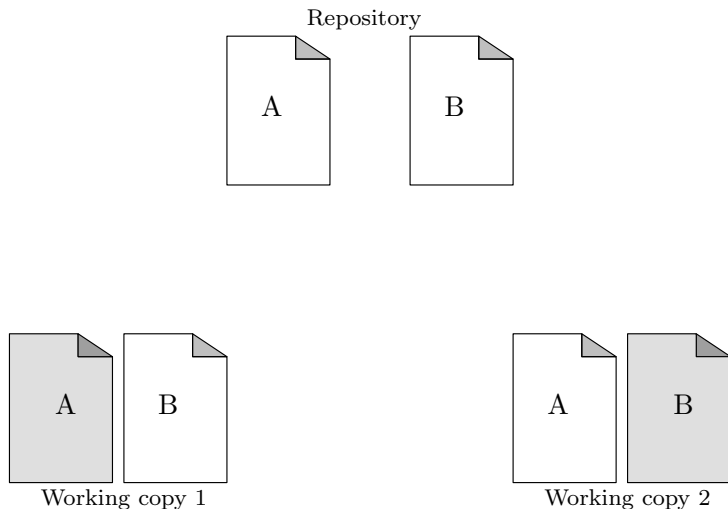
# SVN: Basic Usage

- Publish your changes on the repository with **commit**



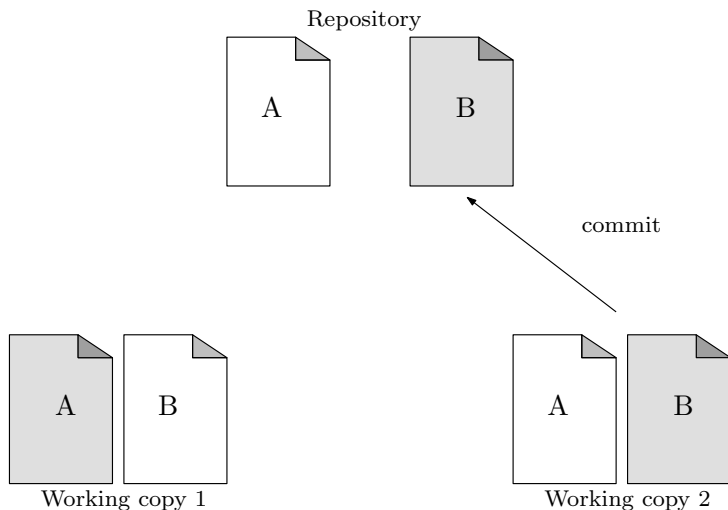
# SVN: Multiple Users

- ▶ Often several users work concurrently on their own working copies, normally on different files



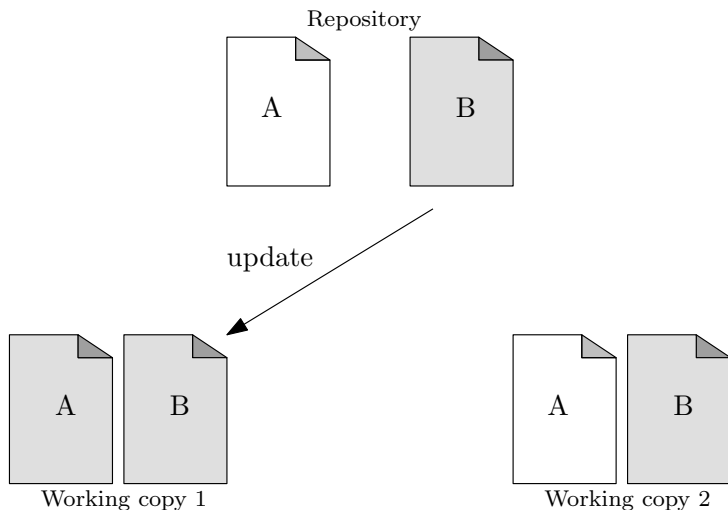
# SVN: Multiple Users

- ▶ When a user commits its changes, the working copies of the remaining users become outdated

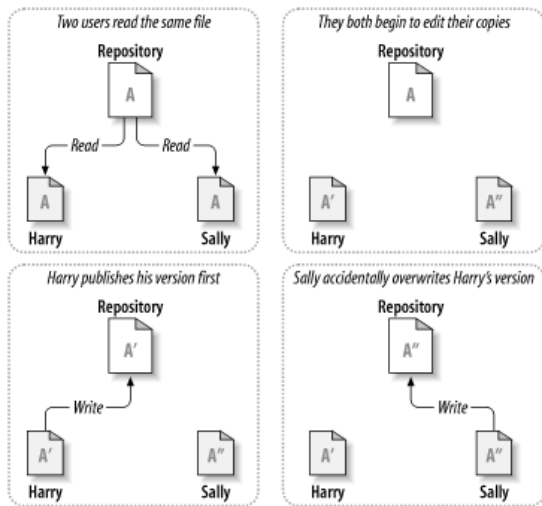


# SVN: Multiple Users

- To bring its working copy in sync with the latest version of the repository a user must **update** it



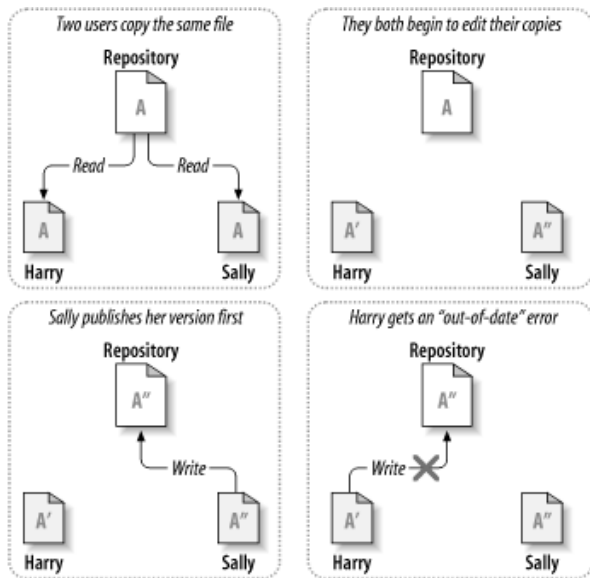
# SVN: Conflicts with Multiple Users



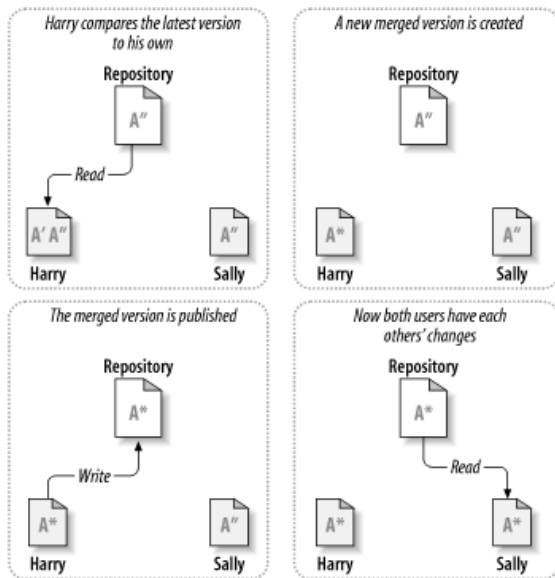
Source: Ben-Collins-Sussman et al. Version Control with Subversion

To address this problem SVN uses a versioning approach known as **Copy-Modify-Merge**

# SVN: Automatic Conflict Detection



# SVN: Manual Conflict Resolution





# SVN: (Some) Useful Commands

**checkout** create a working copy

**mkdir** create a directory

- ▶ Even if you do not have a working copy

**add** add a file/directory to the working copy

**delete** remove a file/directory to the working copy

- ▶ **Be careful**, it may also remove the files in the working copy

**move** rename a file in the repository

**status** list changes made to the working copy

**diff** show differences between the working copy and the repository, or between revisions in the repository

**update** update the working copy

**commit** update the repository

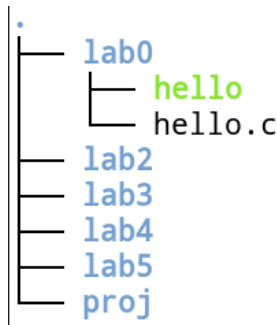
**log** list messages with date and author information attached to revisions and which paths changed in each revision

# SVN and LCOM Labs

- ▶ You must use SVN to submit your work
  - ▶ Both the labs
  - ▶ And the project
- ▶ You'll use the SVN repository available via the Redmine project manager provided by CICA.

# Redmine's SVN Repository Structure

- ▶ There is one SVN repository per Redmine project
- ▶ To facilitate your life, and ours, you must keep it structured as shown in the picture on the right
  - ▶ You can create the structure incrementally
- ▶ You may create subdirectories under the “top level” directories
  - ▶ This is unlikely to help in the labs
  - ▶ But may be useful in the project



# SVN: Advantages

- ▶ It provides automatic backup
- ▶ It makes it easy to restore a previous version
- ▶ It is supported by most IDEs including Eclipse
  - ▶ But you can also use a command line client, even in Minix
- ▶ Users can work on any computer
- ▶ Members of a team can work simultaneously and independently on the same project
- ▶ It logs **who** did/committed **what** and **when**
  - ▶ By using appropriate messages or comments, it is also possible to know **why**
- ▶ It is possible to try a new approach, and continue development on the older one, if you are not happy with the outcome

# Thanks to:

I.e. shamelessly translated material by:

► João Cardoso (jcard@fe.up.pt)

## Further Reading

- ▶ Search the web for the right tutorial for you on SVN
- ▶ *Ben Collins-Sussman et. al* Version Control with Subversion [DRAFT] For Subversion 1.?