

G52GRP 2012–2013: Lecture 3

Project Site and Version Control with Subversion

Henrik Nilsson

University of Nottingham, UK

G52GRP 2012–2013: Lecture 3 – p.1/41

Sharing Code and Documents (1)

- Passing copies from person to person using e.g. e-mail or USB memory sticks?

Might work for a single document where people “take turns”, but otherwise recipe for disaster!

- Who’s got the latest version?
- Who’s got the right to edit?
- How to ensure that everyone sees up-to-date versions of everything?
- ...

G52GRP 2012–2013: Lecture 3 – p.3/41

This Lecture

- Indefero project site
- Why use version control systems?
- Subversion
- Using Subversion

G52GRP 2012–2013: Lecture 3 – p.2/41

Sharing Code and Documents (2)

- A shared repository is a better idea!
 - A School-hosted Indefero project has been set up for each group. Sign in with your CS credentials at:
`https://code.cs.nott.ac.uk`
 - Provides additional project management features beyond a shared repository.
 - The designated `gp12-xxx` Indefero project should (in most cases) be your central repository for code and documentation.
- Additionally, a Unix group has been created for each group on the School’s Linux servers.

G52GRP 2012–2013: Lecture 3 – p.4/41

Indefero Features (1)

- Project Management:
 - Project description (using Markdown syntax).
 - Access rights (who can access and change what: defaults should work for most groups).
 - Detailed configuration of many of the other features (e.g. “tags”, notification e-mails).
- Source code repository:
 - Version control using Subversion.
 - Not just for code, but can also be used for reports, design documents, and more.

G52GRP 2012–2013: Lecture 3 – p.5/41

Indefero Features (2)

- Documentation
 - Detailed project description (including aims, vision), agendas, meeting minutes, design documents, QA plans, ...
 - Hyperlinks between documents
 - Markdown syntax
- Issue tracking
- Downloads
 - Various releases
 - Source, binaries for various platforms, ...

G52GRP 2012–2013: Lecture 3 – p.6/41

Group Project Site Deliverable

- Group Project Site: first “deliverable” of G52GRP
- Due 2 November
- Designed to get your project site off the ground and ensure everyone understands the basics of Indefero and Subversion
- Nominate a Project Site Master ASAP!
- See the Group Project Handbook for further details.

G52GRP 2012–2013: Lecture 3 – p.7/41

Other Ways To Share & Coordinate (1)

What about other (possibly external) solutions?
For example:

- GitHub, Gitorious, Bitbucket, ...
- Google Docs
- Dropbox
- Facebook
- Google+
- ...

G52GRP 2012–2013: Lecture 3 – p.8/41

Other Ways To Share & Coordinate (2)

OK, and may even be needed, but:

- Group Project Site deliverable must still be done as specified.
- All documents and code **must** be backed up on School servers!
Temporary unavailability of external hosting, or external host going out of business (or your own, private machines dying, getting stolen, ...), are **not** valid extenuating circumstances.
- You may have to copy certain data across to the project site for submission purposes.

G52GRP 2012-2013: Lecture 3 - p.9/41

Other Ways To Share & Coordinate (4)



G52GRP 2012-2013: Lecture 3 - p.11/41

Other Ways To Share & Coordinate (3)

Additionally:

- Social networking sites like Facebook were designed for ... social networking, not software development.
- Consequently, lack key features like version control and issue tracking.
- Grops that **did** use Facebook reported that the social aspects were a constant source of distraction.

G52GRP 2012-2013: Lecture 3 - p.10/41

Why Use Version Control Systems? (1)

OK, doc and code shared. Problem solved? No ...

- If a team of people involved, how to coordinate the work on the shared source code and documentation?
- As the source and documentation evolves, how to
 - keep track of changes
 - keep track of consistent configurations
 - insulate against “work in progress”
 - ...

G52GRP 2012-2013: Lecture 3 - p.12/41

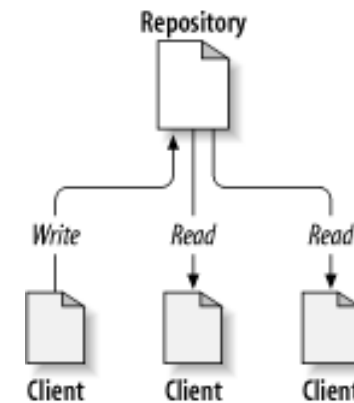
Why Use Version Control Systems? (2)

Version control systems

- originally addressed the second problem (hence the name)
- but modern ones also provide very sophisticated support for
 - teams of programmers working on shared source and documentation
 - distributed teams of programmers (over the Internet)

G52GRP 2012-2013: Lecture 3 - p.13/41

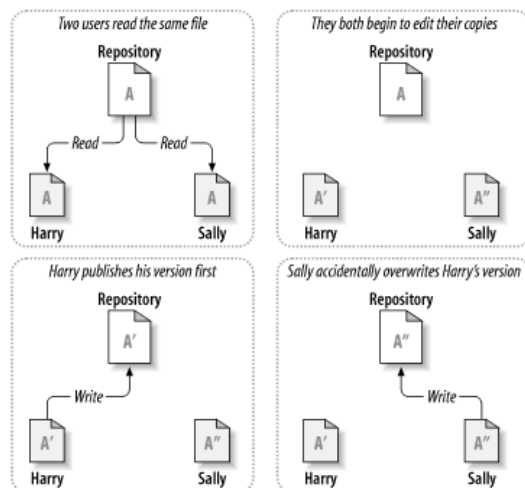
Basic Model



(Pictures from Collins-Sussman, Fitzpatrick, Pilato: Version Control with Subversion.)

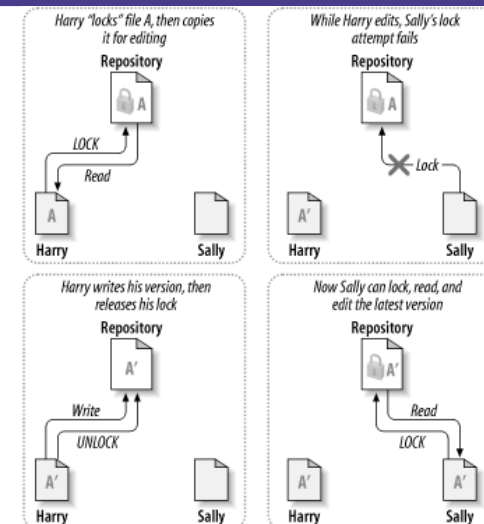
G52GRP 2012-2013: Lecture 3 - p.14/41

The Problem to Avoid



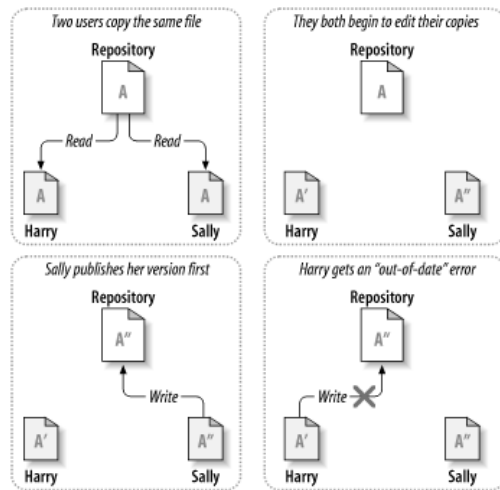
G52GRP 2012-2013: Lecture 3 - p.15/41

The Lock, Modify, Unlock Model



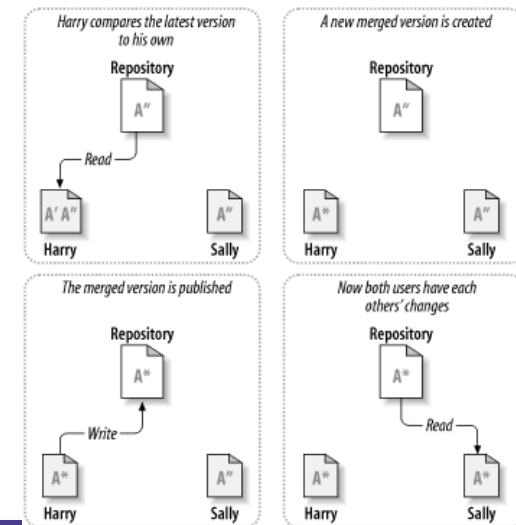
G52GRP 2012-2013: Lecture 3 - p.16/41

The Copy, Modify, Merge Model (1)



GS2GRP 2012-2013: Lecture 3 - p.17/41

The Copy, Modify, Merge Model (2)



GS2GRP 2012-2013: Lecture 3 - p.18/41

Version Control

A version control system provides

- a “time travel” facility: arbitrary earlier versions of the repository can be retrieved
- facilities for supporting parallel, non-interfering development, e.g. through what **looks** like separate copies, ...
- ... while **maximizing sharing** and **facilitating reintegration** of lines of development.

GS2GRP 2012-2013: Lecture 3 - p.19/41

What Is Subversion? (1)

- Free, open-source version control system.
- Manages files **and directories**, allowing older versions of (a part of) a file hierarchy to be retrieved at any point in time, pinpointing changes, keeping track of metadata such as logs for information **about** changes, etc.
- Handles **both** text and binary data (like Word documents, images)
- Supports concurrent development (the Copy, Modify, Merge model), both locally and remotely (over a network).

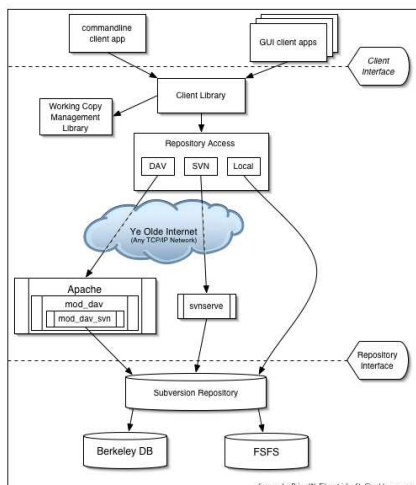
GS2GRP 2012-2013: Lecture 3 - p.20/41

What Is Subversion? (2)

- Also does support locking (mainly intended for binary data that cannot easily be merged: images, Word documents, other application-specific binary data, . . .)
- Other helpful features like
 - file portability (e.g. transparent conversion between CR/LF and LF line ending conventions)
 - automation through hooks (e.g. sending e-mail after changes committed)

Main reference (freely available on-line) G52GRP 2012-2013: Lecture 3 – p.21/41

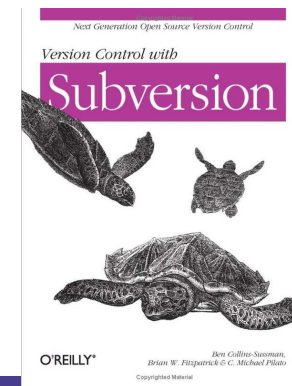
Architecture of Subversion



What Is Subversion? (3)

Main reference (freely available on-line):

Collins-Sussman, Fitzpatrick, Pilato:
Version Control with Subversion



Getting Started

- Use either the command-line interface, or a Subversion client like Tortoise (Windows, shell extension), or from within IDEs like NetBeans, Eclipse (with Subclipse plugin).
- Command-line interface used in the following: principles remain the same regardless of access mode.
- Check out a copy of the repository, e.g.:

```
svn checkout \  
https://code.cs.nott.ac.uk/svn/gp12-nhn
```
- Change directory: `cd gp12-nhn.`

Accessing the Repository

The command-line Subversion client is called `svn`. It has many subcommands, e.g.:

- `svn list`
- `svn add`
- `svn copy`
- `svn commit`

It is always possible to get help, including on specific subcommands:

- `svn help`
- `svn help copy`

G52GRP 2012–2013: Lecture 3 – p.25/41

Initial Repository Structure (1)

Let's populate the repository with some initial structure. The Subversion book recommends three main directories for each project:

- **trunk**: for the main development
- **branches**: for branched-off developments (may later be merged back into main branch)
- **tags**: named “snap shots” of the development; often a “release”.

Branches and Tags are created by copying: `svn copy`

G52GRP 2012–2013: Lecture 3 – p.26/41

Initial Repository Structure (2)

Under trunk we might want to have subdirectories for subprojects, e.g.:

- **src**: for source code
- **doc**: for documentation

Subversion does not make any particular assumptions: the directory structure can be what you like.

AND! It is easy to change the structure later by simply moving around files and directories.

G52GRP 2012–2013: Lecture 3 – p.27/41

Initial Repository Structure (3)

Let's create this directory structure:

```
marian$ mkdir trunk
marian$ mkdir branches
marian$ mkdir tags
marian$ mkdir trunk/src
marian$ mkdir trunk/doc
```

G52GRP 2012–2013: Lecture 3 – p.28/41

Initial Repository Structure (4)

Let's tell subversion all these directories have been added:

```
marian$ svn add trunk
marian$ svn add branches
marian$ svn add tags
marian$ svn add trunk/src
marian$ svn add trunk/doc
```

Then, let's commit to the central repository:

```
marian$ svn commit
```

G52GRP 2012-2013: Lecture 3 – p.29/41

Adding a File (1)

Let's add a document:

```
isis-19% cd trunk/doc
isis-20% ooffice
isis-21% ls -l
-rw-r--r-- 1 henrik henrik 8192 Oct 23 01:45
  design.doc
```

The location of the repository is stored with the working copy, so Subversion commands can now be given without giving the repository URL:

```
isis-49% svn status
?      design.doc
```

G52GRP 2012-2013: Lecture 3 – p.31/41

Checking Out a Working Copy

Now *other* people can check out a working copy of the relevant part of the repository:

```
svn checkout \
https://code.cs.nott.ac.uk/svn/gp12-nhn/trunk
A      trunk/doc
A      trunk/src
Checked out revision 1.
```

G52GRP 2012-2013: Lecture 3 – p.30/41

Adding a File (2)

The status “?” indicates something which is unknown to Subversion. We need to tell Subversion about it:

```
isis-50% svn add design.doc
A (bin) design.doc
```

Important! The new document is now added to the *local working copy*. But it (and other changes) will not be propagated to the central repository until we explicitly perform a **commit**.

```
isis-51% svn status
A      design.doc
```

G52GRP 2012-2013: Lecture 3 – p.32/41

Adding a File (3)

Subversion correctly determined that our document is a **binary** file. (Could have been configured otherwise.)

```
isis-52% svn proplist design.doc
Properties on 'design.doc':
  svn:mime-type
isis-53% svn propget svn:mime-type \
  design.doc
application/octet-stream
```

G52GRP 2012-2013: Lecture 3 – p.33/41

Committing

Time to propagate the changes to the central repository:

```
isis-59% svn commit \
  -m "System design and ideas from meeting 23 Oct"
Adding      doc/design.doc
Adding      doc/Meeting-2012-10-23
Adding      doc/Meeting-2012-10-23/ideas.txt
Transmitting file data .
Committed revision 2.
```

G52GRP 2012-2013: Lecture 3 – p.35/41

Adding a Directory

It is equally easy to add directories:

```
isis-54% mkdir Meeting-2012-10-23
isis-55% svn add Meeting-2012-10-23
A      Meeting-2012-10-23
isis-56% emacs Meeting-2012-10-23/ideas.txt
isis-57% svn add Meeting-2012-10-23/ideas.txt
A      Meeting-2012-10-23/ideas.txt
isis-58% svn status
A      design.doc
A      Meeting-2012-10-23
A      Meeting-2012-10-23/ideas.txt
```

Or use `svn mkdir`.

G52GRP 2012-2013: Lecture 3 – p.34/41

Making Changes

Let's assume a few typos in the design are fixed:

```
isis-65% svn status
M      design.doc
isis-66% svn commit -m "Fixed typos"
Sending      doc/design.doc
Transmitting file data .
Committed revision 3.
```

G52GRP 2012-2013: Lecture 3 – p.36/41

Propagating Changes (1)

Assume someone else makes changes and commits. We can check the status against the repository and get log entries

```
isis-70% svn status -u
      *      2  design.doc
Status against revision:      3
isis-70% svn log -r 3 design.doc
-----
r3 | nhn | 2012-10-24 09:51:00 +0100 (Wed, 24 Oct 2012)

Fixed typos
-----
```

G52GRP 2012-2013: Lecture 3 – p.37/41

Conflicts (1)

What if someone else has committed changes before I commit?

Conflict! Text files can, however, be merged.

```
isis-92% svn commit
Sending      Meeting-2012-10-23/ideas.txt
Transmitting file data .svn: Commit failed (details fo]
isis-83% svn update
C    ideas.txt
Updated to revision 7.
```

G52GRP 2012-2013: Lecture 3 – p.39/41

Propagating Changes (2)

Let's bring our working copy up-to-date:

```
isis-71% svn update
U    design.doc
Updated to revision 3.
```

In general, it is good practice to bring everything up-to-date before starting to make any changes. Minimizes the risk of conflicts.

G52GRP 2012-2013: Lecture 3 – p.38/41

Conflicts (2)

The differences are marked in the file. Edit as necessary. Then:

```
isis-93% svn resolved idead.txt
Resolved conflicted state of 'ideas.txt'
isis-94% svn commit
```

(Newer versions has a more sophisticated svn resolve command.)

G52GRP 2012-2013: Lecture 3 – p.40/41

Other useful Subversion commands

Some other commands:

- `svn delete`
- `svn copy`
- `svn diff`
- `svn revert`
- `svn lock`
- `svn unlock`

Be sure to read at least the introductory chapters of the Subversion book (very accessible) and do use `svn help!`