

G52GRP 2000–2010: Lecture 3

Version Control with Subversion

Henrik Nilsson

University of Nottingham, UK

This Lecture

- Basic models for shared development
- Why use version control systems?
- Subversion
- Using Subversion

Sharing Code and Documents (1)

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

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!

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?

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?

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?
- . . .

Sharing Code and Documents (2)

- A shared repository is a better idea!

Sharing Code and Documents (2)

- A shared repository is a better idea!
 - A UNIX group has been set up for each G52GRP group on the servers. E.g. `gp08-nhn`.

Sharing Code and Documents (2)

- A shared repository is a better idea!
 - A UNIX group has been set up for each G52GRP group on the servers. E.g. `gp08-nhn`.
 - Create a directory owned by your UNIX group in the home directory of one group member and store everything there.

Sharing Code and Documents (2)

- A shared repository is a better idea!
 - A UNIX group has been set up for each G52GRP group on the servers. E.g. `gp08-nhn`.
 - Create a directory owned by your UNIX group in the home directory of one group member and store everything there.
 - Ensure all files and subdirectories are owned by the group (e.g. Set Group ID facility, `chmod s+g . . .`), and everything group readable and writable (`chmod g+rw . . .`, `umask 0007`).

Other Ways To Share and Coordinate (1)

What about external solutions? E.g.

Other Ways To Share and Coordinate (1)

What about external solutions? E.g.

- Google Docs (“No more emailing attachments. Collaborate with others online.”)

Other Ways To Share and Coordinate (1)

What about external solutions? E.g.

- Google Docs (“No more emailing attachments. Collaborate with others online.”)
- Facebook (perhaps mainly for light-weight coordination)

Other Ways To Share and Coordinate (1)

What about external solutions? E.g.

- Google Docs (“No more emailing attachments. Collaborate with others online.”)
- Facebook (perhaps mainly for light-weight coordination)
- Full-featured project hosting, like Source Forge, Google Code, Patch-Tag, ...

-
-
-

Other Ways To Share and Coordinate (2

Issues? Yes!

Other Ways To Share and Coordinate (2)

Issues? Yes!

- All documents and code **must** be backed up on School's servers!

Other Ways To Share and Coordinate (2)

Issues? Yes!

- All documents and code **must** be backed up on School's servers!
Temporary unavailability of external hosting, or external host going out of business (or your machines dying, getting stolen), are not valid extenuating circumstances.

Other Ways To Share and Coordinate (2)

Issues? Yes!

- All documents and code **must** be backed up on School's servers!
Temporary unavailability of external hosting, or external host going out of business (or your machines dying, getting stolen), are not valid extenuating circumstances.
- Many free project hosting services insist on Open Source.

Other Ways To Share and Coordinate (2)

Issues? Yes!

- All documents and code **must** be backed up on School's servers!
Temporary unavailability of external hosting, or external host going out of business (or your machines dying, getting stolen), are not valid extenuating circumstances.
- Many free project hosting services insist on Open Source.
 - May not be appropriate for your projects.

Other Ways To Share and Coordinate (2)

Issues? Yes!

- All documents and code **must** be backed up on School's servers!
Temporary unavailability of external hosting, or external host going out of business (or your machines dying, getting stolen), are not valid extenuating circumstances.
- Many free project hosting services insist on Open Source.
 - May not be appropriate for your projects.
 - External contributions would be an issue.

Why Use Version Control Systems? (1)

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

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?

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”
 - ...

Why Use Version Control Systems? (2)

Version control systems

Why Use Version Control Systems? (2)

Version control systems

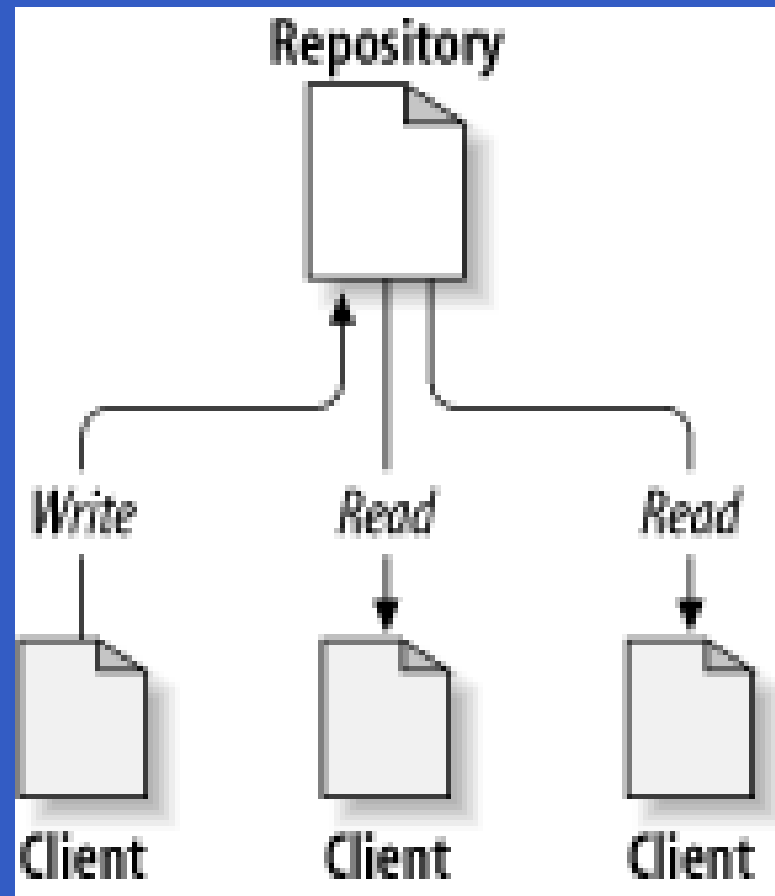
- originally addressed the second problem (hence the name)

Why Use Version Control Systems? (2)

Version control systems

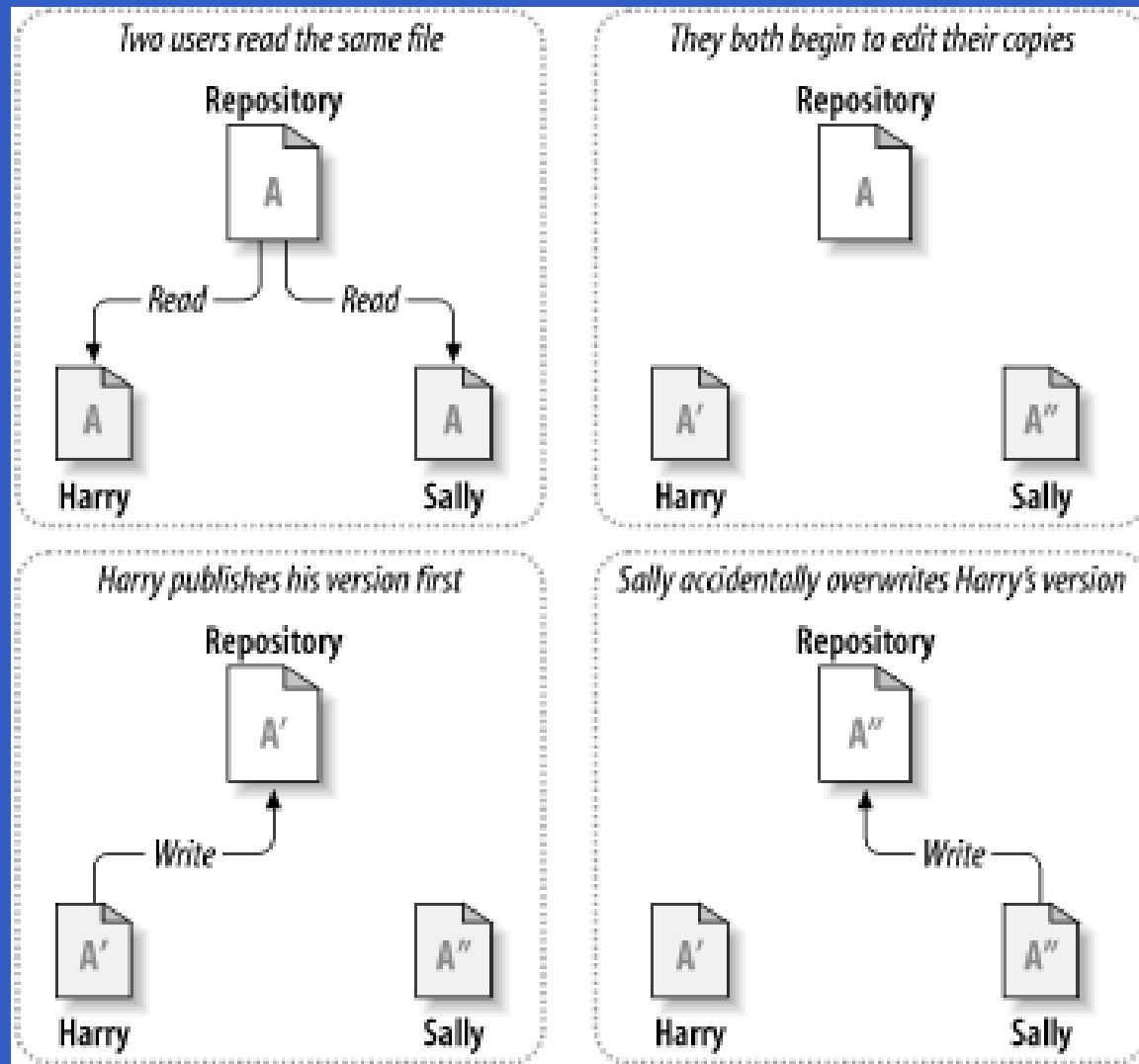
- originally addressed the second problem (hence the name)
- but modern ones also provide very sophisticated support for
 - teams of programmers working on shared source and documentation
 - distributed teams of programmers (over the Internet)

Basic Model

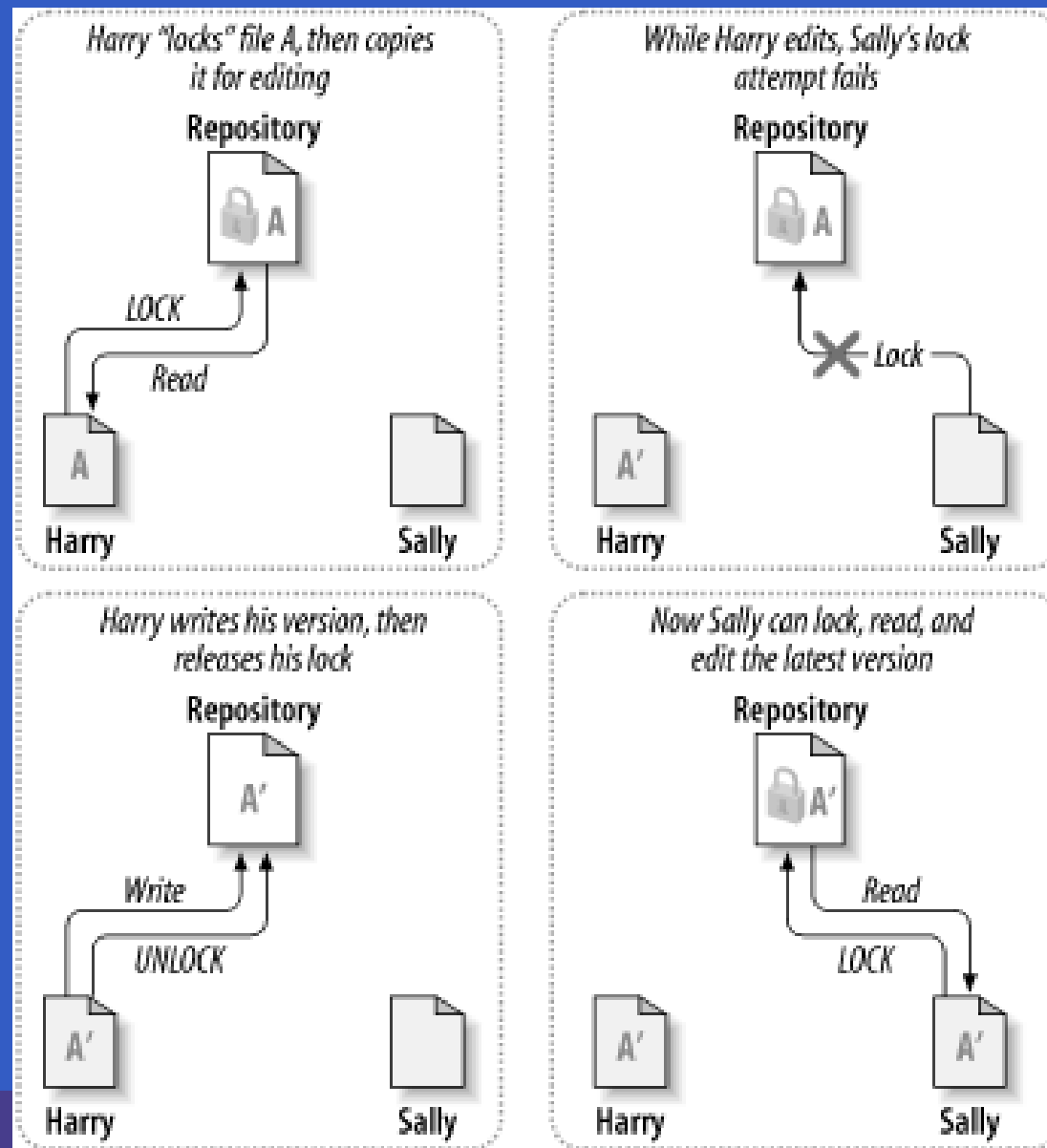


(Pictures from Collins-Sussman, Fitzpatrick, Pilato: Version Control with Subversion.)

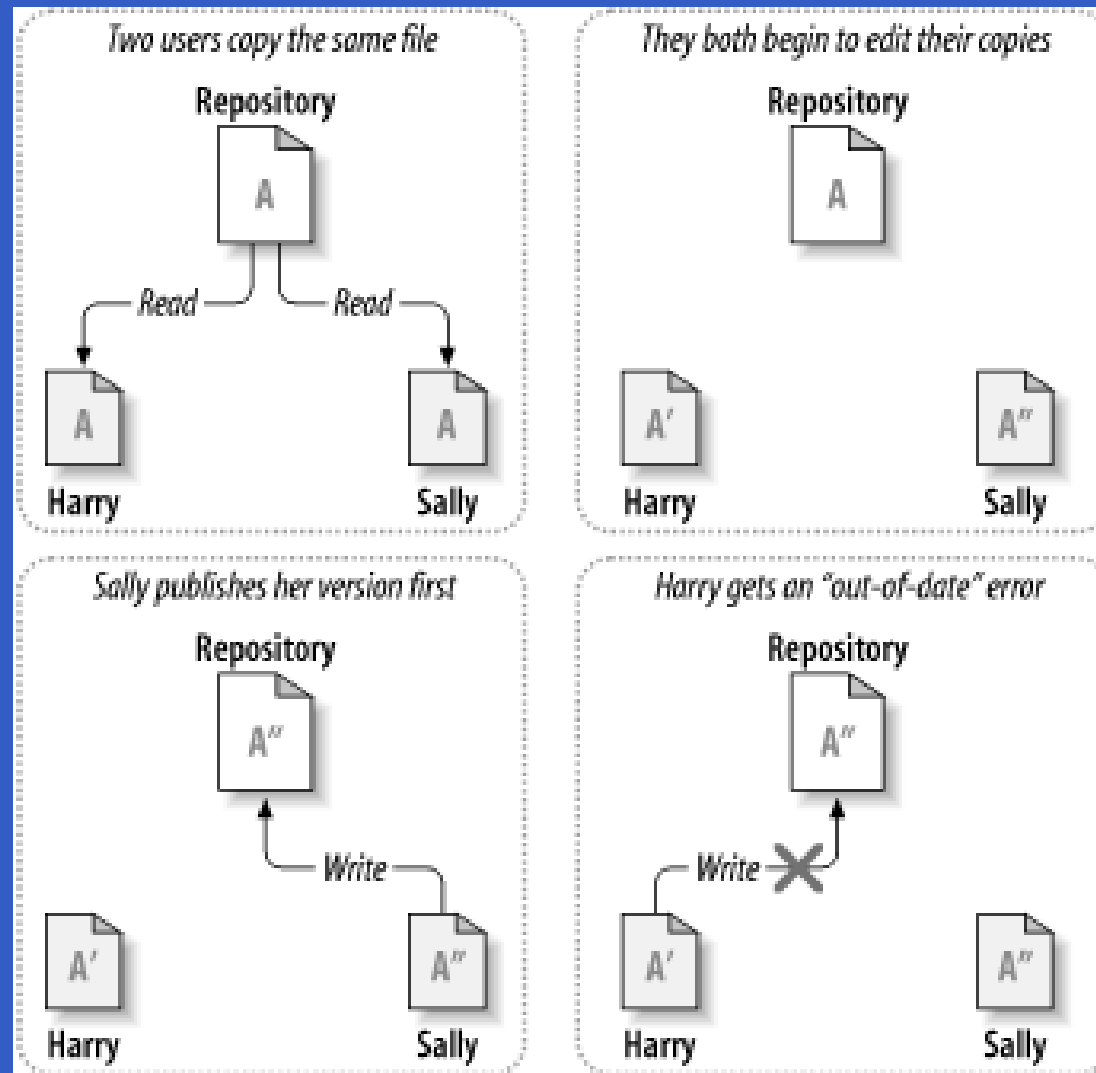
The Problem to Avoid



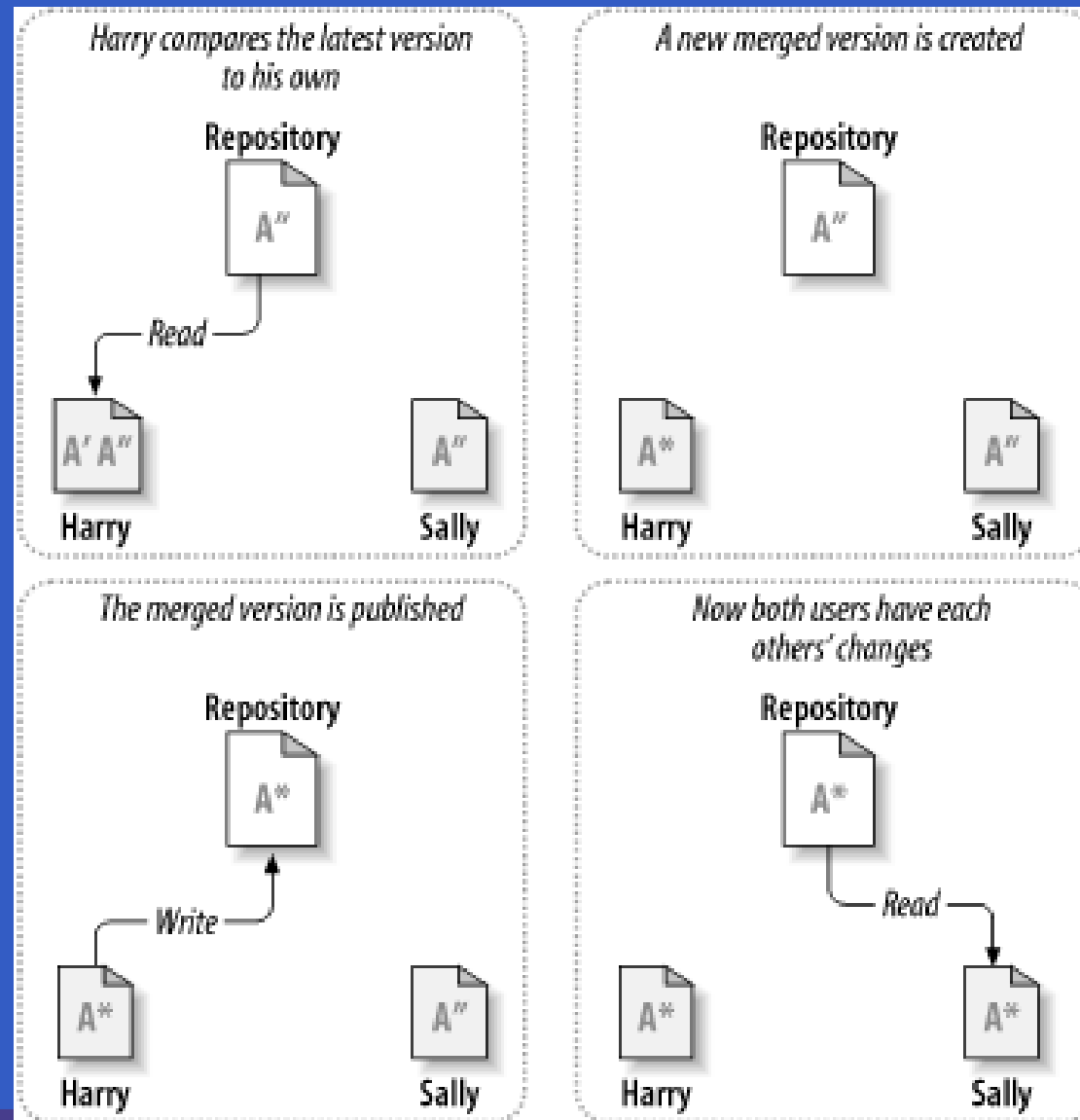
The Lock, Modify, Unlock Model



The Copy, Modify, Merge Model (1)



The Copy, Modify, Merge Model (2)



Version Control (1)

In large-scale software development it is essential to be able to

Version Control (1)

In large-scale software development it is essential to be able to

- have access to source code and documentation as it was at various points in the past

Version Control (1)

In large-scale software development it is essential to be able to

- have access to source code and documentation as it was at various points in the past
- make and maintain releases

Version Control (1)

In large-scale software development it is essential to be able to

- have access to source code and documentation as it was at various points in the past
- make and maintain releases
- allow parts of a big project to evolve separately, without changes constantly getting in other the way of other subteams.

-
-
-

Version Control (2)

How to solve?

Version Control (2)

How to solve?

Frequent copies (“snapshots”) of everything?

Version Control (2)

How to solve?

Frequent copies (“snapshots”) of everything?

But multiple copies is a maintenance nightmare!

-
-
-

Version Control (3)

A version control system provides

Version Control (3)

A version control system provides

- a “time travel” facility: arbitrary earlier versions of the repository can be retrieved

Version Control (3)

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, ...

Version Control (3)

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.

What Is Subversion? (1)

- Free, open-source version control system.

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 meta data such as logs for recording information **about** changes, etc.

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 meta data such as logs for recording information **about** changes, etc.
- Handles both text and binary data

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 meta data such as logs for recording information **about** changes, etc.
- Handles both text and binary data
- Supports concurrent development (the Copy, Modify, Merge model), both locally and remotely (over a network).

What Is Subversion? (2)

- Also does support locking (mainly intended for binary data that cannot easily be merged: images, application-specific binary data like Word documents ...)

What Is Subversion? (2)

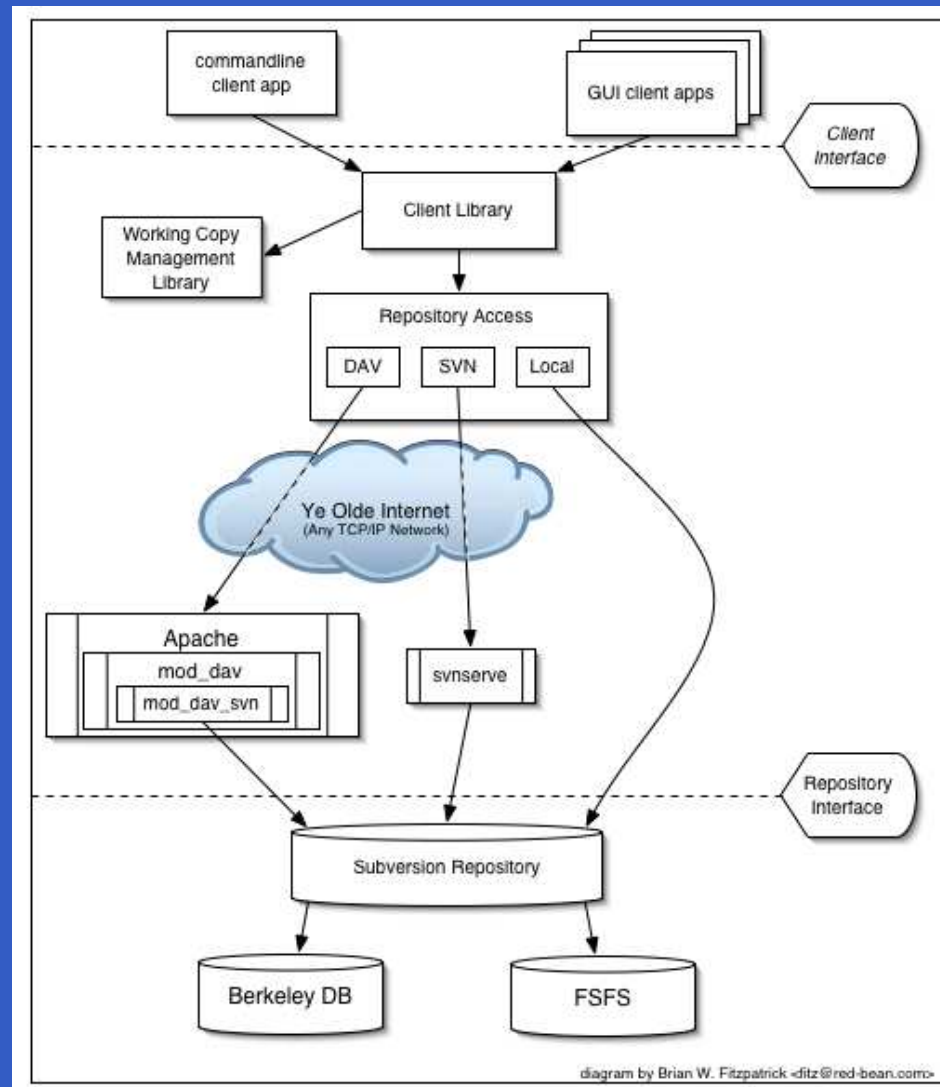
- Also does support locking (mainly intended for binary data that cannot easily be merged: images, application-specific binary data like Word documents ...)
- Other helpful features like
 - file portability (e.g. transparent conversion between CR/LF and LF line ending conventions)
 - automation through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

What Is Subversion? (3)

Main reference (freely available on-line):

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

Architecture of Subversion



-
-
-

Getting Started

Assumptions:

Getting Started

Assumptions:

- The repository is a shared directory owned by the UNIX group of the group and located in the home directory on the School's UNIX servers of one of the group members.

Getting Started

Assumptions:

- The repository is a shared directory owned by the UNIX group of the group and located in the home directory on the School's UNIX servers of one of the group members.
- Access is either

Getting Started

Assumptions:

- The repository is a shared directory owned by the UNIX group of the group and located in the home directory on the School's UNIX servers of one of the group members.
- Access is either
 - local, by logging in to **your own** UNIX account on the servers

Getting Started

Assumptions:

- The repository is a shared directory owned by the UNIX group of the group and located in the home directory on the School's UNIX servers of one of the group members.
- Access is either
 - local, by logging in to **your own** UNIX account on the servers
 - remote over SSH from e.g. own laptop which must have SSH and some Subversion client installed.

Creating a Repository (1)

- Decide which group member is going to host the repository. This person logs in to his or her account on the School's UNIX servers.

Creating a Repository (1)

- Decide which group member is going to host the repository. This person logs in to his or her account on the School's UNIX servers.
- Create the repository, taking care to ensure owner and permission bits are set to enable sharing:

```
marian$ umask 0007
marian$ mkdir repos
marian$ chgrp gp08-nhn repos
marian$ chmod g+s repos
marian$ svnadmin create repos
```

Creating a Repository (2)

Check that the owner and file permissions are as expected:

```
marian$ ls -lg repos
```

```
total 7
```

```
-rw-rw----  1 nhn  gp08-nhn  229 Oct 23 00:11 README.  
drwxrws---  2 nhn  gp08-nhn  512 Oct 23 00:11 conf  
drwxrws---  2 nhn  gp08-nhn  512 Oct 23 00:11 dav  
drwxrws---  5 nhn  gp08-nhn  512 Oct 23 00:11 db  
-r--r----- 1 nhn  gp08-nhn    2 Oct 23 00:11 format  
drwxrws---  2 nhn  gp08-nhn  512 Oct 23 00:11 hooks  
drwxrws---  2 nhn  gp08-nhn  512 Oct 23 00:11 locks
```

Accessing the Repository (1)

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

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

Accessing the Repository (1)

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`

Accessing the Repository (2)

- Local access. Use URL with “file” prefix:

```
marian$ svn list file:///staff/nhn/repos
```

Accessing the Repository (2)

- Local access. Use URL with “file” prefix:

```
marian$ svn list file:///staff/nhn/repos
```

- Remote access, e.g. from home PC. `svn` invoked on local machine, but instructed to talk to remote Subversion server over SSH:

```
isis-12% svn list \  
svn+ssh://marian.cs.nott.ac.uk/staff/nhn/repos
```

(May want to use `ssh-agent` or (on Windows) `pageant` for caching of login credentials.)

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 (that may later be merged back into the main branch)
- **tags**: named “snap shots” of the development, e.g. a stable version.

Initial Repository Structure (2)

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

- **src**: for source code
- **doc**: for documentation
- **meetings**: for agendas, minutes, etc.

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

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

Initial Repository Structure (3)

Let's create this directory structure:

```
marian$ mkdir g52grp
marian$ mkdir g52grp/trunk
marian$ mkdir g52grp/branches
marian$ mkdir g52grp/tags
marian$ mkdir g52grp/trunk/src
marian$ mkdir g52grp/trunk/doc
marian$ mkdir g52grp/trunk/meetings
```

If you have more than one project, you may want to have one directory for each, and then a trunk, branches, and tags in each.

Initial Repository Structure (4)

Let's import this tree into the repository. The tree can also contain initial files, e.g. existing source code, minutes, ...

```
marian$ svn import g52grp file:///staff/nhn/repos \  
-m "Initial import"
```

```
Adding          g52grp/trunk
```

```
Adding          g52grp/trunk/doc
```

```
Adding          g52grp/trunk/src
```

```
Adding          g52grp/trunk/meetings
```

```
Adding          g52grp/branches
```

```
Adding          g52grp/tags
```

Initial Repository Structure (5)

The temporary directory structure `g52grp` can now be removed:

```
marian$ rm -rf g52grp
```

The contents can now be listed, e.g. remotely:

```
isis-17% svn list \  
svn+ssh://marian.cs.nott.ac.uk/staff/nhn/repos/trunk  
doc/  
meetings/  
src/
```

Checking Out a Working Copy

To start working on the project, a working copy of the relevant part of the repository must be checked out:

```
isis-18% svn checkout \  
svn+ssh://marian.cs.nott.ac.uk/staff/nhn/repos/trunk\  
    gp08-nhn  
A    gp08-nhn/doc  
A    gp08-nhn/src  
A    gp08-nhn/meetings  
Checked out revision 1.
```

Adding a File (1)

Let's add a document:

```
isis-19% cd gp08-nhn/meetings
```

```
isis-20% ooffice
```

```
isis-21% ls -l
```

```
-rw-r--r-- 1 henrik henrik 8192 Oct 23 01:45  
minutes-2008-10-23.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
```

```
? minutes-2008-10-23.doc
```

Adding a File (2)

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

```
isis-50% svn add minutes-2008-10-23.doc
A (bin) minutes-2008-10-23.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      minutes-2008-10-23.doc
```

Adding a File (3)

Subversion correctly determined that our document is a *binary* file. It is possible to configure how Subversion handles various kinds of files, but the defaults should work most of the time.

```
isis-52% svn proplist minutes-2008-10-23.doc
```

```
Properties on 'minutes-2008-10-23.doc':
```

```
  svn:mime-type
```

```
isis-53% svn propget svn:mime-type \  
minutes-2008-10-23.doc
```

```
application/octet-stream
```

Adding a Directory

It is equally easy to add directories:

```
isis-54% mkdir Files-2008-10-23
```

```
isis-55% svn add Files-2008-10-23
```

```
A      Files-2008-10-23
```

```
isis-56% emacs Files-2008-10-23/design.txt
```

```
isis-57% svn add Files-2008-10-23/design.txt
```

```
A      Files-2008-10-23/design.txt
```

```
isis-58% svn status
```

```
A      minutes-2008-10-23.doc
```

```
A      Files-2008-10-23
```

```
A      Files-2008-10-23/design.txt
```

Or use `svn mkdir`.

Committing

Time to propagate the changes to the central repository:

```
isis-59% svn commit \  
-m "Minutes and docs for meeting 23 Oct"  
Adding          minutes-2008-10-23.doc  
Adding          meetings/Files-2008-10-23  
Adding          meetings/Files-2008-10-23/design.txt  
Transmitting file data .  
Committed revision 2.
```

Making Changes

Let's assume a few typos in the minutes are fixed.

```
isis-65% svn status
```

```
M      minutes-2008-10-23.doc
```

```
isis-66% svn commit -m "Fixed typos"
```

```
Sending      meetings/minutes-2008-10-23.doc
```

```
Transmitting file data .
```

```
Committed revision 3.
```

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    minutes-2008-10-23.doc
```

```
Status against revision:      3
```

```
isis-70% svn log -r 3 minutes-2008-10-23.doc
```

```
-----  
r3 | nhn | 2008-10-24 09:51:00 +0100 (Fri, 24 Oct 200
```

```
Fixed further typos  
-----
```

Propagating Changes (2)

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

```
isis-71% svn update
```

```
U      minutes-2008-10-23.doc
```

```
Updated to revision 3.
```

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

Conflicts (1)

What if someone else has committed changes before I commit? Conflict! Text files can, however, be merged.

```
isis-92% svn commit
```

```
Sending          Files-2008-10-23/design.txt
```

```
Transmitting file data .svn: Commit failed (details f
```

```
isis-83% svn update
```

```
C    design.txt
```

```
Updated to revision 7.
```

Conflicts (2)

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

```
isis-93% svn resolved design.txt
```

```
Resolved conflicted state of 'design.txt'
```

```
isis-94% svn commit
```

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

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`!**