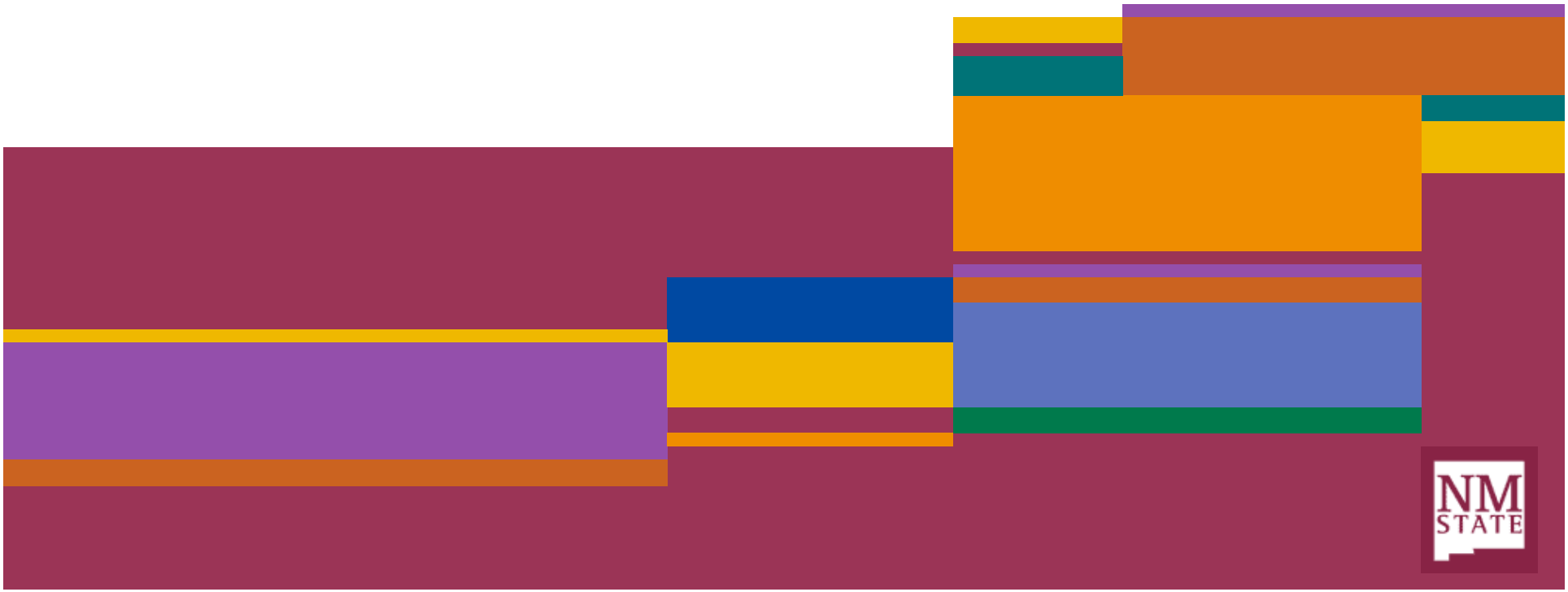
Revision Control: An Introduction to Subversion (svn)



References

- Material for this lecture is drawn from:

  `http://en.wikipedia.org/wiki/Revision_control`

  `http://svnbook.red-bean.com/en/1.5/index.html`

Introduction

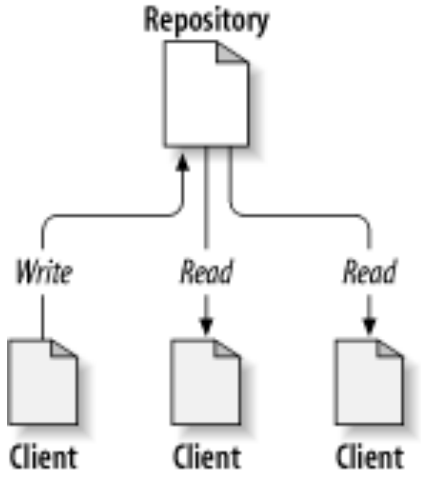
- Revision control is the management of changes (revisions) to source code over time
  - Management includes tracking and controlling changes to source code
  - Allows developer to recover older versions
  - Allows developer to “undo” changes
  - Source code could be C/C++, MATLAB, LaTeX—any text source files
- These names all refer to the same thing:
  - Revision control, Version control, Source control, Source control management (SCM)
- An initial set of source files is “revision 1.” When the first change is made, the resulting set is “revision 2.”
  - Each revision has a time stamp and the person’s name making the revision

Version Control System (VCS)

- A Version Control *System* is a stand-alone application which assists with revision control
- Some popular and free (as in beer) VCSs are:
  - Concurrent Versions Systems (CVS) (centralize SCM)
  - Subversion (svn) (centralize SCM)
  - Mercurial (hg) (distributed SCM)

The Repository

- The repository stores information in the form of a filesystem tree--a typical hierarchy of files and directories
  - The repository remembers every change ever written to it—every change to every file and changes to the directory tree itself (additions and deletions of files)
  - When copying/updating from the repository, we get the latest version of the tree
  - We can also view previous states of the tree: “What did this directory contain last Wednesday?” and “Who was the last person to change this file, and what changes did he/she make?”

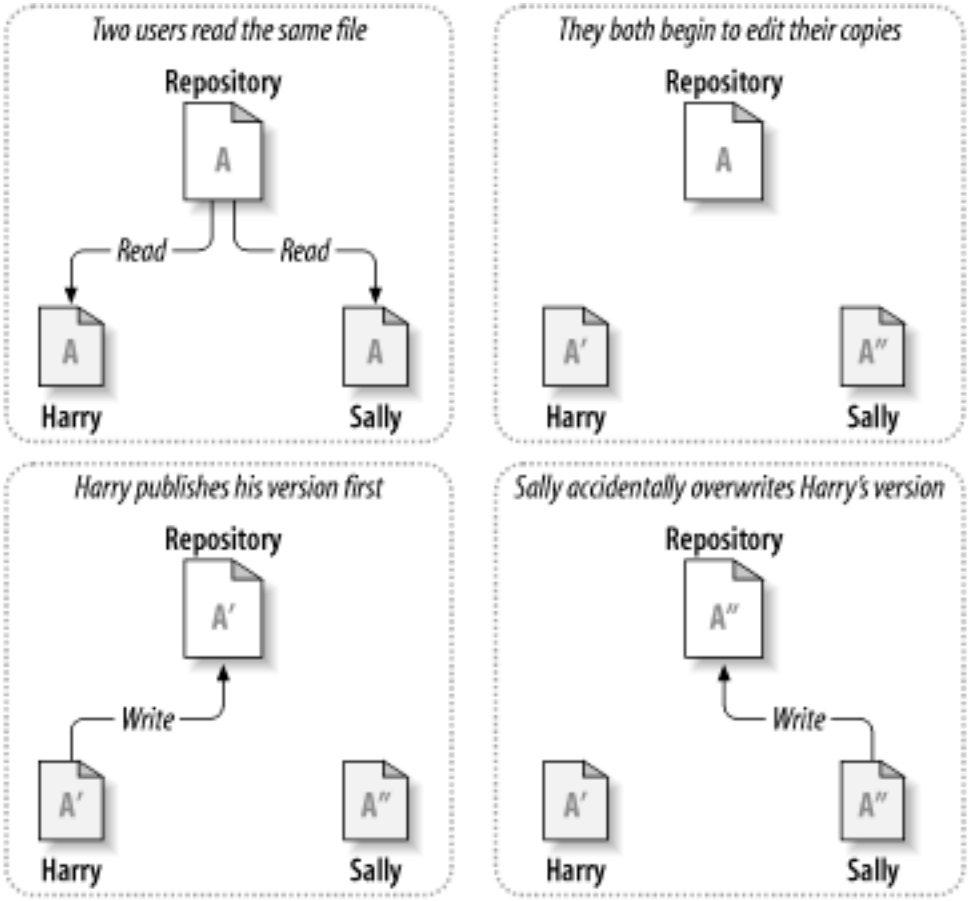


Source Management Models

- VCS uses a centralized model, where all the revision control functions are performed on a shared server
- VCS has to solve a fundamental problem: how will the system allow multiple users to work on a file, without developers overwriting each other's work?
  - If two developers try to change the same file at the same time, without some method of managing access, the developers may end up overwriting each other's work
  - Centralized VCS solves this problem using one of two different "source management models": file locking and version merging

Source Management Models (cont'd)

The problem...

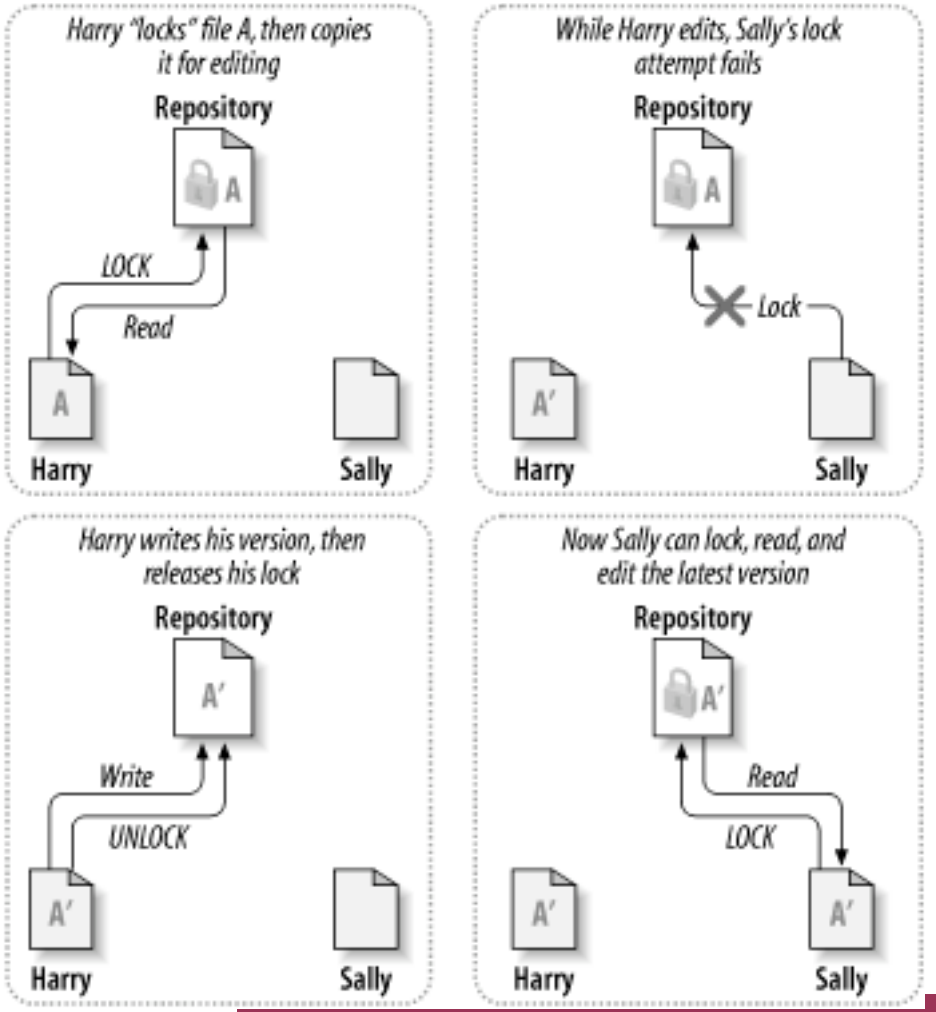


File Locking vs. Version Merging

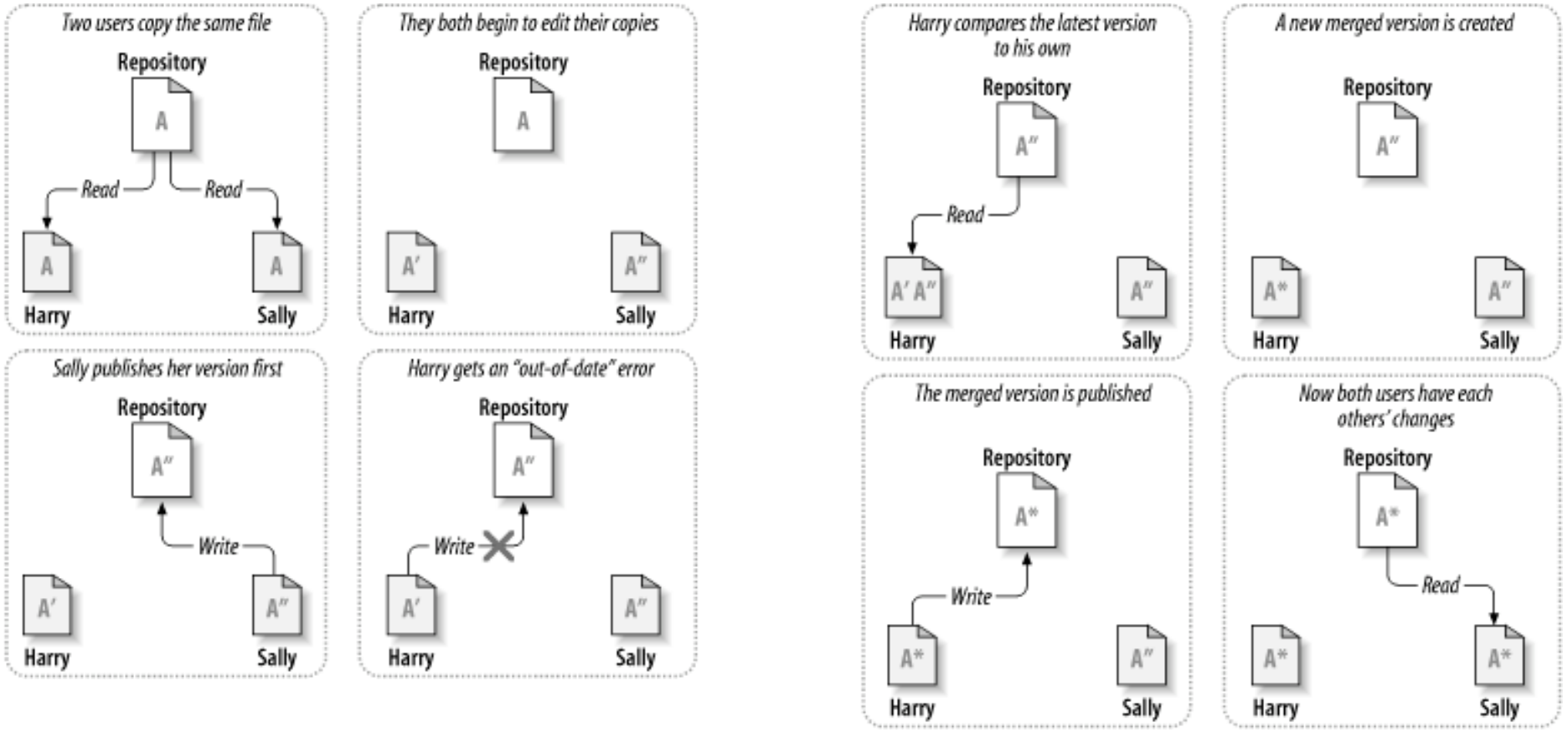
- With “file locking”, the system *locks* the file so that only one at a time person can change a file
  - Once a developer “checks out” a file no one else is allowed to change that file until that developer “checks in” the updated version
- With “version merging” multiple developers may edit the *same file* at the *same time*. Then the system merges changes into the central repository
  - Normally two people are not working on the same part of the source code so merging is usually easy
  - The second developer checking in code, will need to take care with the merge, to make sure that the changes are compatible and that the merge operation does not introduce its own logic errors within the program

Lock-Modify-Unlock

A developer locks the file, modifies the file, and then releases the lock...



Copy-Modify-Merge



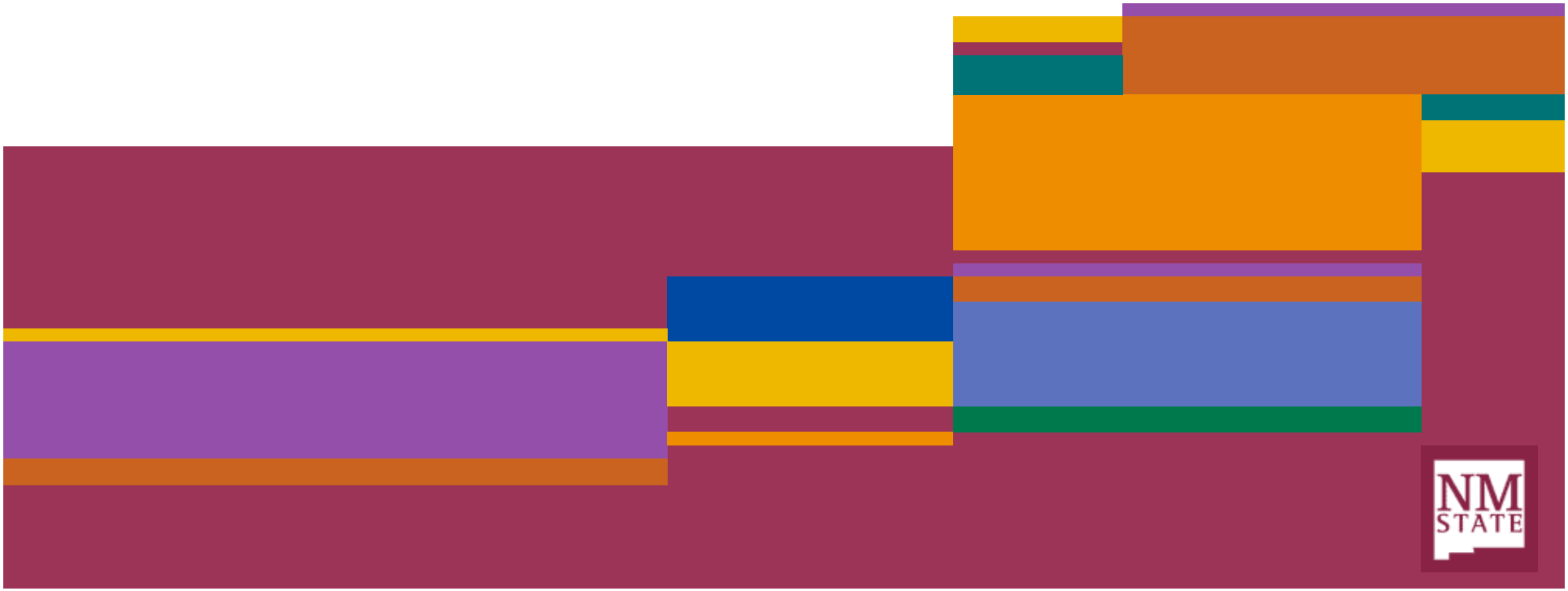
Two developers copy and modify...

...then changes are merged

Solo Developers

- Revision control is very useful even if you are the only developer, i.e. solo developer
  - Allows solo developer to track changes to source codes and go back in time to previous versions of source code
  - Usually no issues with merge if there is only one developer

Subversion (svn)



Subversion (svn)

- To begin using subversion for version control you must download and install the free (as in beer) software

  `http://subversion.apache.org/`

- If you prefer to host your project on the Internet, there are many free SVN hosting services

Example: Unversioned Filesystem Tree

- Projects within “code” directory are unversioned and changes are not yet being tracked
- In this example, we have a project1, project2, project3 residing in a directory called “code”
- Projects normally have three directories
  - “trunk” holds “main line” of development
  - “branches” contain branch or experimental copies (more on this later)
  - “tags” contain tag important copies (more on this later)
- Example source codes
  - project1/trunk/hello.c
  - project2/trunk/hello.m
  - project3/trunk/hello.tex

Creating a Repository

- We begin by creating a repository to track projects

```
mkdir repos
svnadmin create repos/project1
```

- Next, we copy an unversioned filesystem tree into a repository so that we may begin tracking projects. We include a short message describing this action

```
svn import code/project1
file:///Users/pdeleon/Desktop/repos/project1
-m "Initial import"
```

- We need only create and import once

Creating a Repository (cont.)

```
Adding         code/project1
Adding         code/project1/trunk
Adding         code/project1/trunk/hello.c
Adding         code/project1/branches
Adding         code/project1/tags

Committed revision 1.
```

Basic Work Cycle

1. Checkout or update your working copy
2. Make changes
3. Examine your changes
4. Possibly undo changes
5. Merge others' changes and resolve conflicts
6. Commit your changes

- If you are the sole developer, there is unlikely to be a step 5

Checkout Source Files

- Next, we checkout project 1 source files

```
svn checkout file:///Users/pdeleon/Desktop/
repos/project1/trunk project1
```

- A local, “working copy” of project 1 is made for modification since it doesn’t already exist
  - Subversion uses a copy-modify-merge approach instead of lock-modify-unlock

Updating Working Copy

- When working on a team project, you should update your working copy to receive any changes others have made before modifying code

```
svn update project1
```

Commit Changes

- Next, we modify working copy, compile, debug, etc....Once we are done we commit changes to repo

```
svn commit project1
```

  - We are then asked for comments in order to document changes, i.e. log message
- Alternately, we can include a short log message on the commit

```
svn commit –m “Fixed the pointer error.”
project1
```

- After committing changes, be sure to update so that code and logs can be updated

Examine Changes

- After making changes but before committing, it's a good idea to review your changes
  - Discover un-intended changes/problems
  - Review changes so you can create a more accurate log message

```
svn status project1
```

  - Will give status codes for all files you changed
  - ? (file not under version control), A (file scheduled for addition), D (file scheduled for deletion), M (file has local modifications)

```
svn diff project1
```

  - Will tell you exactly how you've modified things

Making Filesystem Changes

- We can modify files but can also add/delete files

`svn add foo`

– Schedule file or directory “foo” to be *added* to the repository. When you next commit, “foo” will become a child of its parent directory

`svn delete foo`

– Schedule file or directory “foo” to be *deleted* from the repository. If “foo” is a file it is immediately deleted from working copy. If “foo” is a directory it will be removed from working copy and repository on next commit.

`svn copy foo bar`

– Create a new item “bar” (duplicate of “foo”) and add “bar” to repository on next commit (copy history is recorded for “bar” having originated from “foo”)

Making Filesystem Changes (cont'd)

`svn move foo bar`

– Schedule "bar" for addition as a copy to "foo" to the repository on next commit and schedule "foo" for removal. Same as copy+delete

`svn mkdir blort`

– A new directory "blort" to created and scheduled for addition to the repository on next commit.

Undoing Changes

```
svn revert foo
```

- Subversion reverts “foo” to its premodified state by overwriting it with a cached “pristine” copy

Examining History

- The subversion repository is like a time machine—there is a record of every change ever committed. You can explore this history in the working copy

```
svn log project1
```

  - Shows log messages with date and author information

```
svn log –r 2:4
```

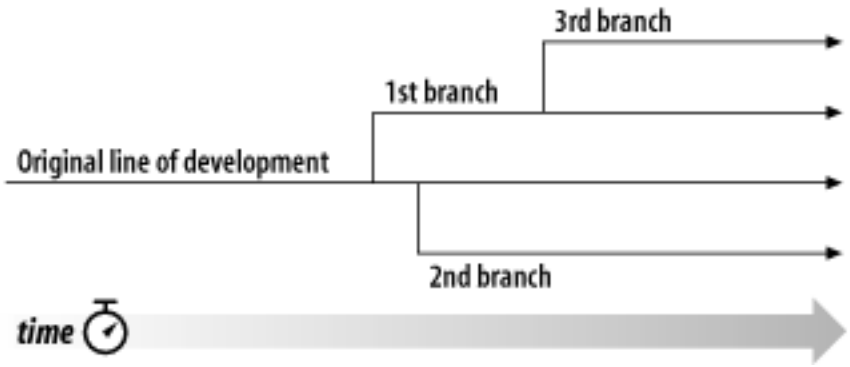
  - Shows log messages for revisions 2--4

```
svn log hello.c
```

  - Shows log history of a single file

Branches

- A branch is a line of development that exists independently of another line
  - Often used for experimental or “side” development



- SVN has tools to create/merge branches from/with main development line or trunk

Tags

- A tag is a snapshot of a project in time
- Although the repository contains snapshots in time, sometimes we want to give more human-friendly names to tags such as “release-1.0”
- In svn we simply “copy” the trunk to the tags directory and include a message

Subversion GUIs

- Many GUIs to subversion exist which allow version control with point and click instead of commands
  - Linux, Mac, Windows: RapidSVN
  - Windows: TortoiseSVN
  - Macintosh: Versions
- Demo of Versions