Observations

- **Copy-modify-merge** allows users to work in parallel
  - Most concurrent changes do not overlap
  - Consistency amongst files is explicitly managed (i.e., no false sense of security)
- This model assumes files are line-based text files
  - Assumes changes can (usually) be merged

More observations

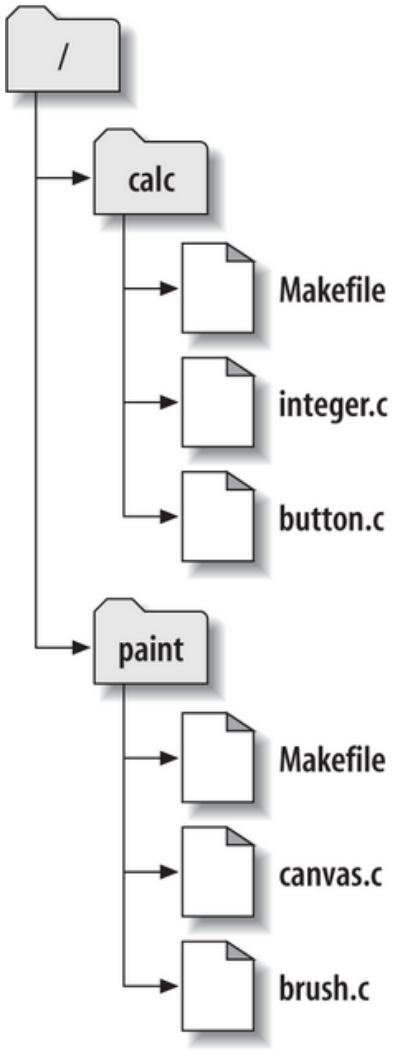
- For binary files, very difficult (if not impossible) to merge conflicting changes
  - JPEG files
  - Object files and executable images
- However, we may still want to keep such items in the repository
  - Also: there exist "text" versions of some image formats (PNG, SVG)
- While Subversion is primarily copy-modify-merge, it still does use locks on occasion
  - These support the Subversion system, not clients
  - Helps prevent "race conditions" involving simultaneous client use

Basic cycle of use with Subversion

1. **`checkout`** a local copy of a project from a repository
2. in your local copy of the project (which is a directory)
   - edit files or
   - create files to **`add`** to the project or
   - do both
3. possibly **`update`** your local copy
   - this picks up changes made by team members / project participants since your last update
4. build / run / test / view / render / read / <fill-in-verb> your work
5. if not yet ready to **`commit`** changes, go to step 2
6. **`commit`** your changes to the repository
7. go to step 2

Basic Concepts: Working Copy

- A working copy is an ordinary directory on your local system
- These are the files you edit
- When changes are complete, you publish your changes by writing to the repository
- Challenges:
  - Files already changed? must merge!
  - Metadata in account used by svn programs (".svn" directory)
  - Subversion repository contains files for several projects, while clients usually work on a subtree



Basic Concepts: Working Copy

- Obtaining a working copy means "checking out" some subtree of the repository
  - This is usually done **only once** per working copy
  - The term "checking out" implies exclusive access, but it is a usage holdover from old code-management tools
- Repository access methods differ
  - direct access via local disk (file:///) **(ugh!)**
  - custom protocol via SSH tunnel (svn+ssh://)
  - http or https
  - (original access method for working copy is stored as metadata in an .svn directory in that working copy)

Basic Concept: Working Copy

- Example: get working copy of the "calc" project from svn.example.com

```
$ svn checkout http://svn.example.com/repos/calc
A calc
A calc/Makefile
A calc/integer.c
A calc/button.c

$ ls -A calc
Makefile integer.c button.c .svn/
```

Basic Concepts: Commit

- Suppose you wish to make a change to button.c
  - You edit the file using your normal workflow
  - Time and date on edited file will be more recent than time and date of file in repository
  - Changes are published by committing your changed file to the repository
- Note: access method and repository location are already stored as metadata!
  - Do not need to specify location of repository
  - Do not need to specify access method
  - (But you might need to provide a password.)

```
$ svn commit -m "fixed a bug" button.c
Sending button.c
Transmitting file data .
Committed revision 57.
```

Basic Concepts: Update

- What if Sally was also working on the project?
  - Let's assume she was not working on button.c
- She can ask Subversion to bring her working copy "up to date"
  - Subversion only updates files that have been changed
  - Principle: Update often if working with a group on a project that uses a repository!

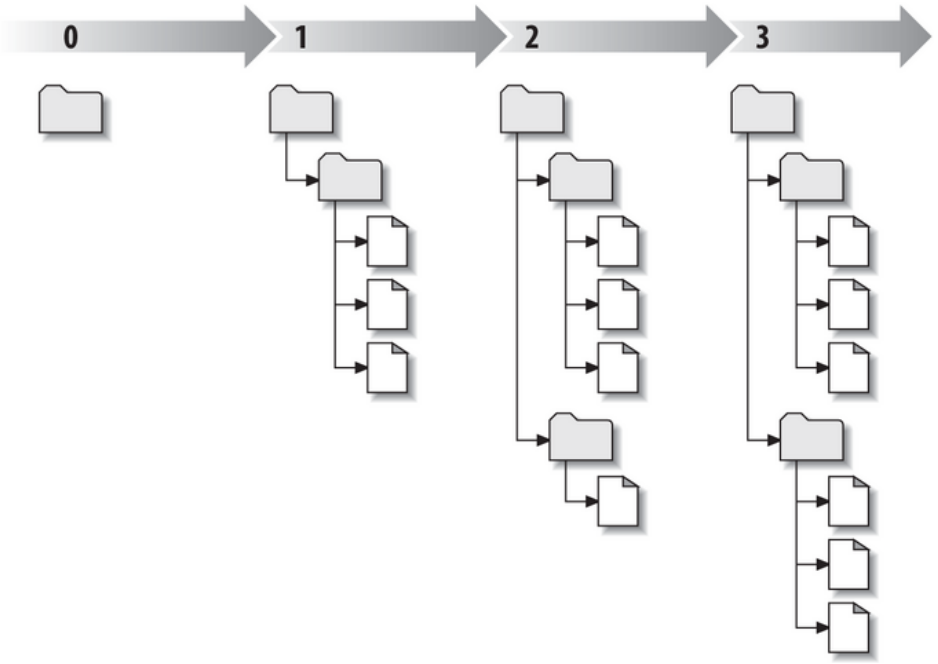
```
$ pwd
/home/sally/calc
$ ls -A
.svn/ Makefile integer.c button.c
$ svn update
U button.c
```

Basic Concepts: Revision

- Each svn **commit** operation
  - can publish changes to any number of files
  - can publish changes to any number of directories
  - does so as a **single, atomic transaction**
- Before a **commit**, you can work on your working copy (i.e., local copy) as you normally would to:
  - change files
  - add files
  - add directories, etc.
- You then can commit at a point you believe (or the group believes) is appropriate
  - Note: adding and deleting files are done using different svn commands
  - is it not enough to use **rm** to delete a file from the repository!
- Each time repository accepts a commit:
  - Creates new state for filesystem tree
  - Each revision is assigned a new number
  - Revision numbers reflect repository state, not just file state

Basic Concept: Repository

- The diagram is a visualization of repository changes in time
- Revisions are numbered from zero, and increase from left to right
- Revisions here denote successive commits
- Each revision number has a filesystem tree hanging below it
  - Each tree is a "snapshot" of the way the repository looked after a commit



Basic Concept: File state

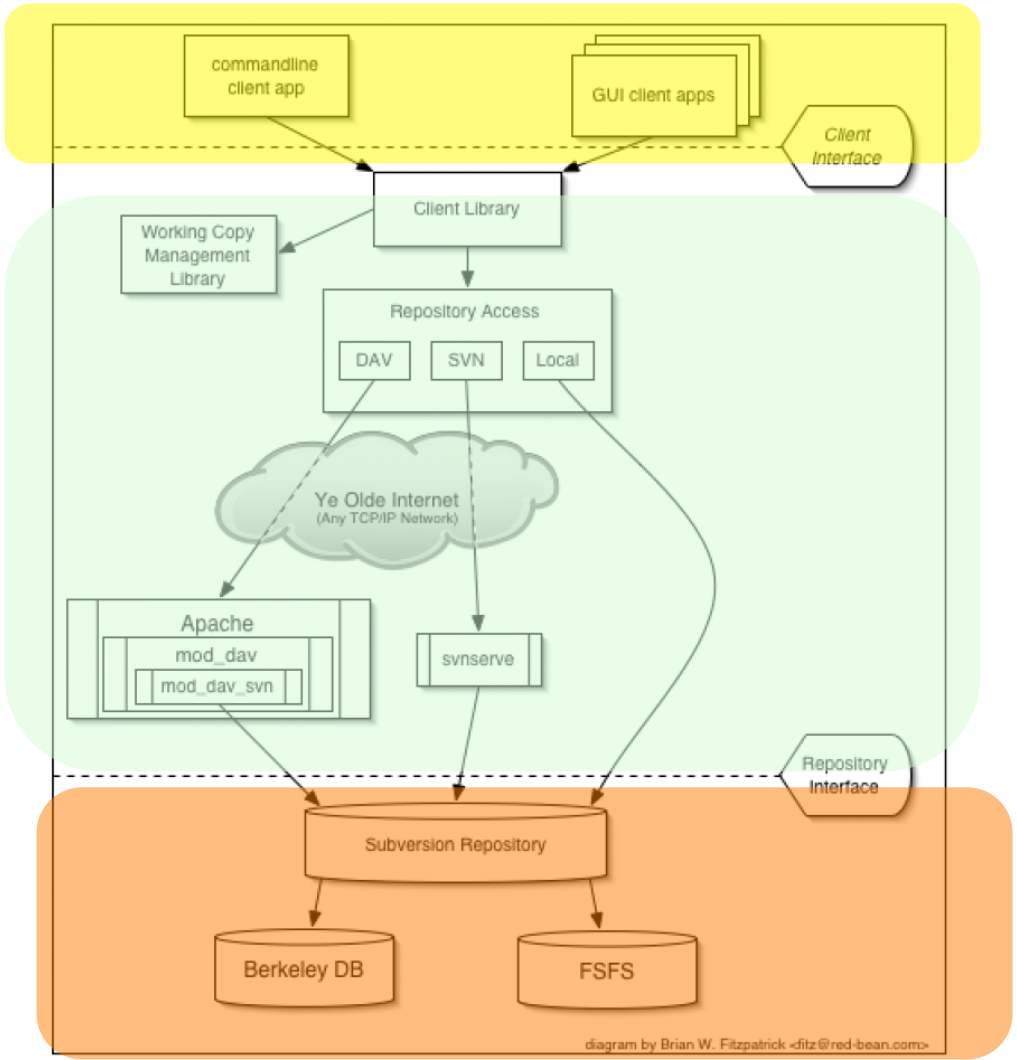
- Working directories can be in one of four states

1. **Locally unchanged** and **Current**
2. **Locally changed** and **Current**
3. **Locally unchanged** and **Out-of-date**
4. **Locally changed** and **Out-of-date**

Basic Concept: File state

1. Locally unchanged and Current
   - no changes to the file in the repository since you updated/added it
2. Locally changed and Current
   - your local copy has changed, but the repository copy is the same as when you last updated/added it
3. Locally unchanged and Out-of-date
   - the copy in the repository has changed, but your copy has not, so an svn update command will work to get you a new copy
4. Locally changed and Out-of-date
   - the file has changed in **both** your local working copy and the repository
   - if you try to do svn commit, you will get an out-of-date error
   - you must update first, and if subversion can't resolve the merge (i.e., if there is a conflict) then **you** will have to help

Subversion tool architecture



Using Subversion

- **You need a Subversion client**
- In this course we use a command-line client
  - `svn` (actually `/usr/bin/svn`)
  - provide `svn` with commands and arguments
- Note other client possibilities
  - Subversion functionality might be built into IDE
  - Subversion functionality built into tool
  - Stand-alone GUI for interacting with repository
- Also: web access
  - There are some browser-based interfaces for accessing web-enabled repositories

Using Subversion

- **Need a repository**
- **Need access to that repository**
- UVic Software Engineering hosts a repository for you:
  - this is on a per-course basis
  - `svn.seng.uvic.ca/svn/seng265/yourlogin`
  - replace "`yourlogin`" with your login ID
  - later can use other, properly configured projects in the same repository for group work
- Other possibilities you might encounter:
  - `sourceforge.net`-like services
  - administering your own server

(a wee word…)

- For this course, use the seng265 repository
- Each of your assignments and labs …
  - … will be "subprojects" within your larger "project"
  - … accessible to the lab instructors and administrators for help when debugging (but accessible to no one else!)
  - … can be accessed remotely by you
- Later in the course:
  - We'll say something about Subversion server administration
  - Also give you advice on how to use Subversion with other projects in other courses