

Version Control

Lecture 7

2/6/2008

Prof. Hilfinger CS164 Lecture 7

1

Administrivia

- In order to do the homework and turn in the project, you *must* have run set-keys. Do so today!
- Tu 11-12 section is now 12-1 on Wednesday in 310 Soda

2/6/2008

Prof. Hilfinger CS164 Lecture 7

2

The Problem

- Software projects can be large and complex.
- May involve many people, geographically distributed
- May require maintenance of several related versions
 - MacOS vs. Windows vs. GNU Linux
 - Stable release vs. beta release of next version
 - Commercial vs. non-commercial
- May require prototyping potential features while still maintaining existing ones.

2/6/2008

Prof. Hilfinger CS164 Lecture 7

3

Version-Control Systems

- Version-control systems attempt to address these and related problems.
- Allow maintenance and archiving of multiple versions of a piece of software:
 - Saving complete copies of source code
 - Comparing versions
 - Merging changes in several versions
 - Tracking changes

2/6/2008

Prof. Hilfinger CS164 Lecture 7

4

Subversion

- Subversion is an open-source version-control system.
- Successor to CVS
- Provides a simple model: numbered snapshots of directory structures
- Handles local or remote repositories

2/6/2008

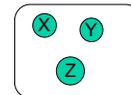
Prof. Hilfinger CS164 Lecture 7

5

Subversion's Model

Repository

User 1



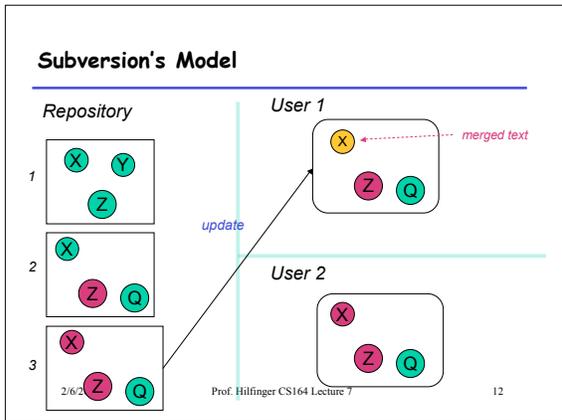
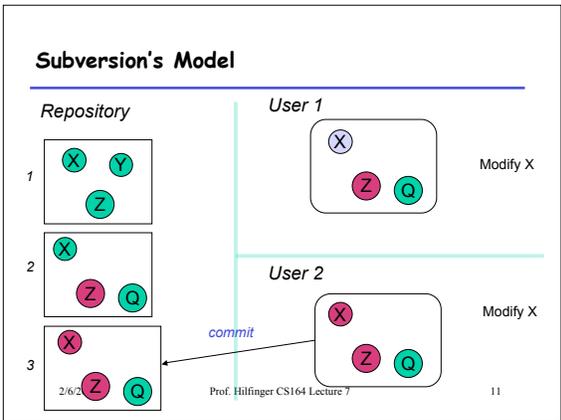
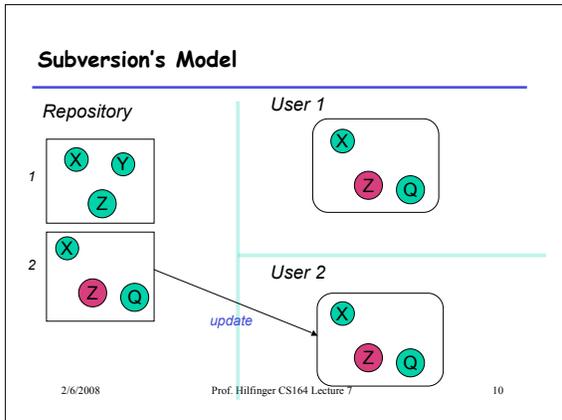
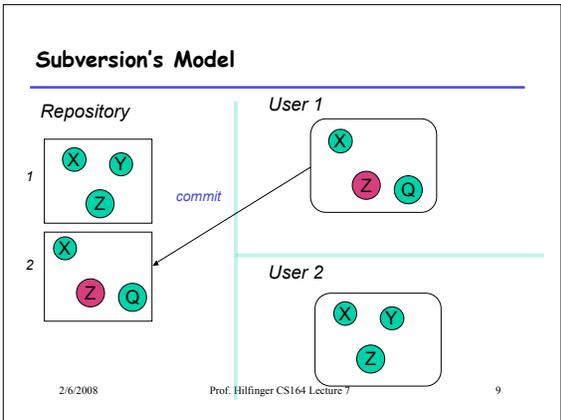
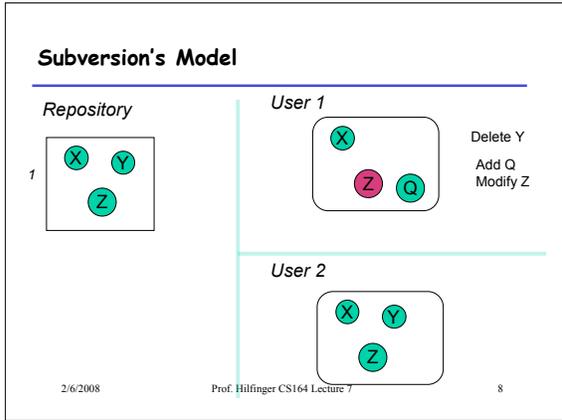
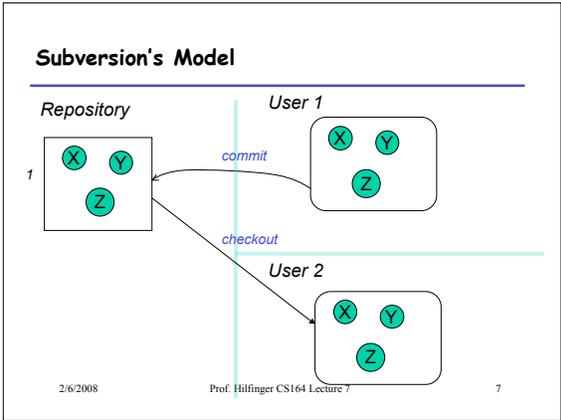
add X
add Y
add Z

User 2

2/6/2008

Prof. Hilfinger CS164 Lecture 7

6



Terminology

- **Repository:** Set of versions
- **Revision:** A snapshot of a particular directory of files
- **Revision number:** A sequence number denoting a particular revision
- **Working copy:** A directory or file initially copied from a revision + administrative data

2/6/2008

Prof. Hilfinger CS164 Lecture 7

13

A Useful Property

- In the previous example, Subversion does not really keep 3 complete copies of the files.
- Instead, it maintains *differences* between versions: if you change little, your revision takes up little space.
- Copying an entire file or directory in the repository is very cheap
 - "Directory foo in revision 110 is the same as directory bar in revision 109"

2/6/2008

Prof. Hilfinger CS164 Lecture 7

14

Some Basic Commands

- We'll be using "ssh tunnels" to access our Subversion repositories.
- We created an ssh key pair for you when you first logged in.
- In the following, we consider login cs164-xx and team Ursa; we'll use nova as a convenient host.

2/6/2008

Prof. Hilfinger CS164 Lecture 7

15

Creating a working copy of a repository

- To get the latest revision:
`svn checkout svn+ssh://cs61b-tj@nova/Ursa`
- Or just one directory:
`svn checkout svn+ssh://cs61b-tj@nova/Ursa/trunk`
- A particular revision:
`svn checkout -r100 svn+ssh://cs61b-tj@nova/Ursa`
- Symbolic revisions:
`svn checkout -rHEAD \`
`svn+ssh://cs61b-tj@nova/Ursa`

2/6/2008

Prof. Hilfinger CS164 Lecture 7

16

Add, Delete, Rename Files, Directories

- When you add or remove a file or directory in a working copy, must inform Subversion of the fact:
 - `svn add NEW-FILE`
 - `svn delete OLD-FILE-OR-DIR`
 - `svn move OLD-PLACE NEW-PLACE`
- These forms *don't* change the repository.
- Must commit changes

2/6/2008

Prof. Hilfinger CS164 Lecture 7

17

Committing Changes

- The command
`svn commit -m "Log message"`
in a working directory will create a new revision in the repository
- New revision differs from previous in the contents of the current directory, which may only be part of the whole tree.
- Message should be informative. Can arrange to use your favorite editor to compose it

2/6/2008

Prof. Hilfinger CS164 Lecture 7

18

Updating

- To get versions of files from most recent revision, do this in directory you want updated
`svn update`
- This will report files Subversion changes, adds, deletes, or *merges*
- Merged files are those modified both by you and (independently) in the repository since you updated/checked out.

2/6/2008

Prof. Hilfinger CS164 Lecture 7

19

Merges and Conflicts

- Reports of changes look like this:
U foo1 *foo1 is updated*
A foo2 *foo2 is new*
D foo3 *foo3 was deleted*
R foo4 *foo4 was deleted, then re-add*
G foo5 *foo5 had mods from you and in repository that did not overlap*
C foo6 *Conflicts: overlapping changes*

2/6/2008

Prof. Hilfinger CS164 Lecture 7

20

Notating Conflicts

- When you have a conflict, you'll find that the resulting file contains *both* overlapping changes:

```
<<<<<<<< .mine
My change
=====
Repository change
>>>>>>>> .r 99 (gives revision #)
```

2/6/2008

Prof. Hilfinger CS164 Lecture 7

21

Resolving Conflicts

- You can either choose to go with the repository version of conflicted file, or yours, or do a custom edit.
- Subversion keeps around your version and the repository version in `foo6.mine`, `foo6.99`
- Personally, I usually just edit the file.
- When conflicts are resolved, use
`svn resolved foo6`
to indicate resolution; then commit.

2/6/2008

Prof. Hilfinger CS164 Lecture 7

22

Branches and Tags

- Suppose Bob wants to make some changes to his project, checking in intermediate steps, but without interfering with partner Mary.
- Good practice is to create a *branch*, a copy of the project files independent of the trunk.
- Copy command does it:
`cd Ursa # Your copy of team's repository`
`svn mkdir branches # If necessary`
`svn copy trunk branches/Bobs-branch`
`svn commit -m "Create Bob's branch"`
`cd branches/Bobs-branch`
and go to work (be sure trunk is committed first).

2/6/2008

Prof. Hilfinger CS164 Lecture 7

23

Branches and Tags

- The use of the `branches` directory is convention; could put it anywhere.
- Again, this copy is cheap in the repository.
- Bob's changes in `branches/Bobs-branch` are completely independent of the trunk.
- Rather elegant idea: no new mechanism!

2/6/2008

Prof. Hilfinger CS164 Lecture 7

24

Tags

- A *tag* is the same as a branch, except that (by convention) we don't usually modify it once it is created.
- Conventional to put it in the `tags` subdirectory, as in the instructions for turning in your project.
- Tags are usually intended as names of particular snapshots of the trunk or some branch (e.g., a release).

2/6/2008

Prof. Hilfinger CS164 Lecture 7

25

Comparing Revisions

- One great feature: ability to compare versions, branches.
- Simple case: what local changes have I made to this working directory?
`svn diff`
- How does this working directory compare to revision 9?
`svn diff -r 9`
- How do revisions 9 and 10 of directory differ?
`svn diff -r 9:10`

2/6/2008

Prof. Hilfinger CS164 Lecture 7

26

More Comparisons

- I'm in branches/Bobs-branch. How does it compare to revision 100 of the trunk?
`svn diff ../trunk@100 .`

2/6/2008

Prof. Hilfinger CS164 Lecture 7

27

Merging

- To *merge* changes between two revisions, *R1* and *R2*, of a file or directory into a working copy means to get the changes that occurred between *R1* and *R2* and make the same changes to the the working copy.
- To merge changes into current working copy:
`svn merge SOURCE1@REV1 SOURCE2@REV2`
where *SOURCE1* and *SOURCE2* are URLs (svn+ssh:...) or working directories and *REV1*, *REV2* are revision numbers.

2/6/2008

Prof. Hilfinger CS164 Lecture 7

28

More Merging

- For short, when sources the same:
`svn -r REV1:REV2 SOURCE`
- To merge in changes that happened between two tagged revisions:
`svn svn+ssh://.../tags/v1 svn+ssh://.../tags/v2 \
branches/Bobs-branch`
- Here we assume we are in *top-level* directory

2/6/2008

Prof. Hilfinger CS164 Lecture 7

29

After Merging

- After merging, as for update, must resolve any conflicts.
- Then we commit the merged version.

2/6/2008

Prof. Hilfinger CS164 Lecture 7

30