Lecture 5: Version Control

Administrivia

- Everyone should now be registered electronically using the link on our webpage. If you haven't, do so today!
- I'd like to have teams formed by next Wednesday at the latest. Preferred size is 3, but *must* be > 1.

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
  - Commericial vs. non-commercial
- May require prototyping potential features while still maintaining existing ones.

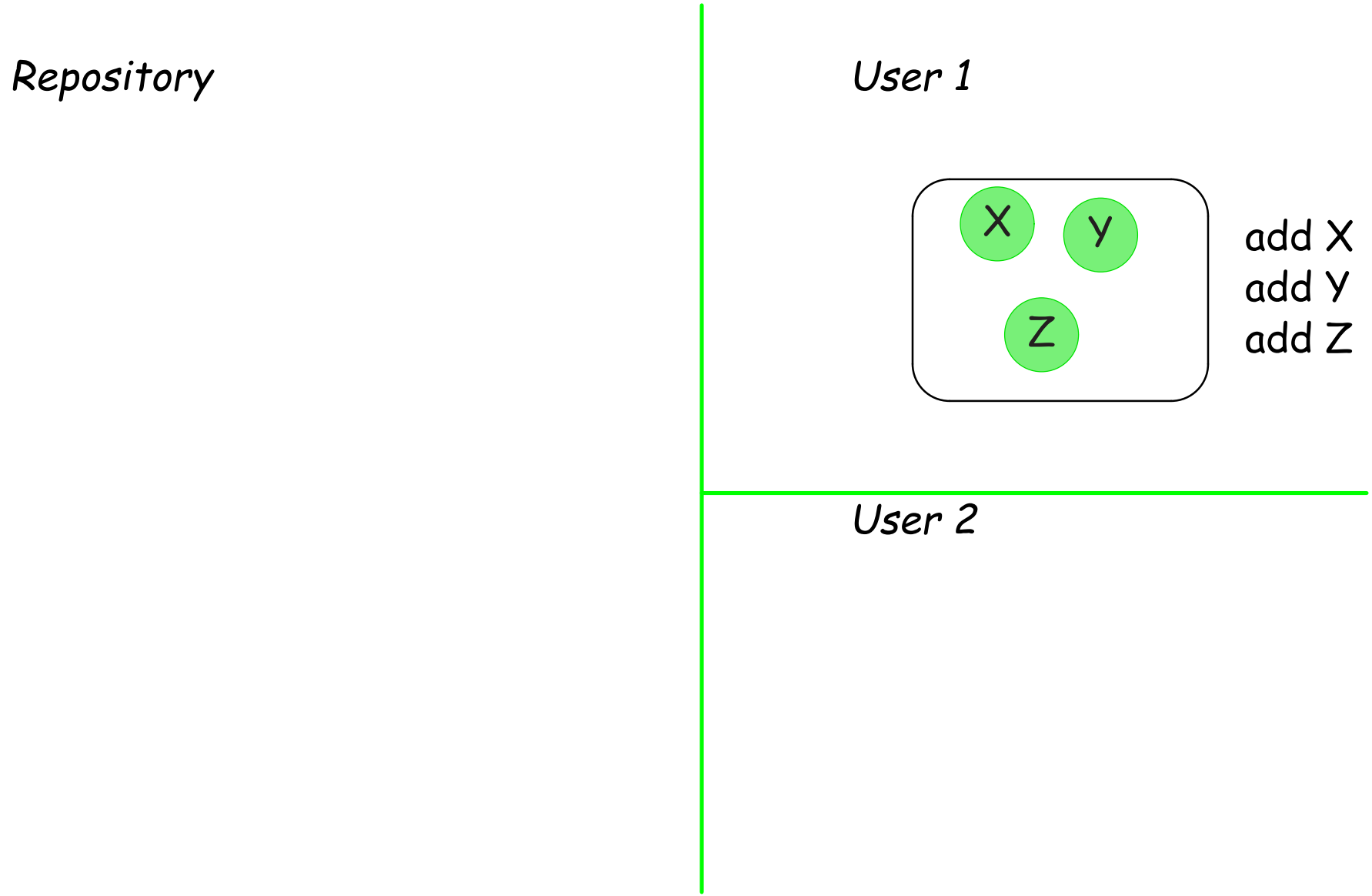
Version-Control Systems

- 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

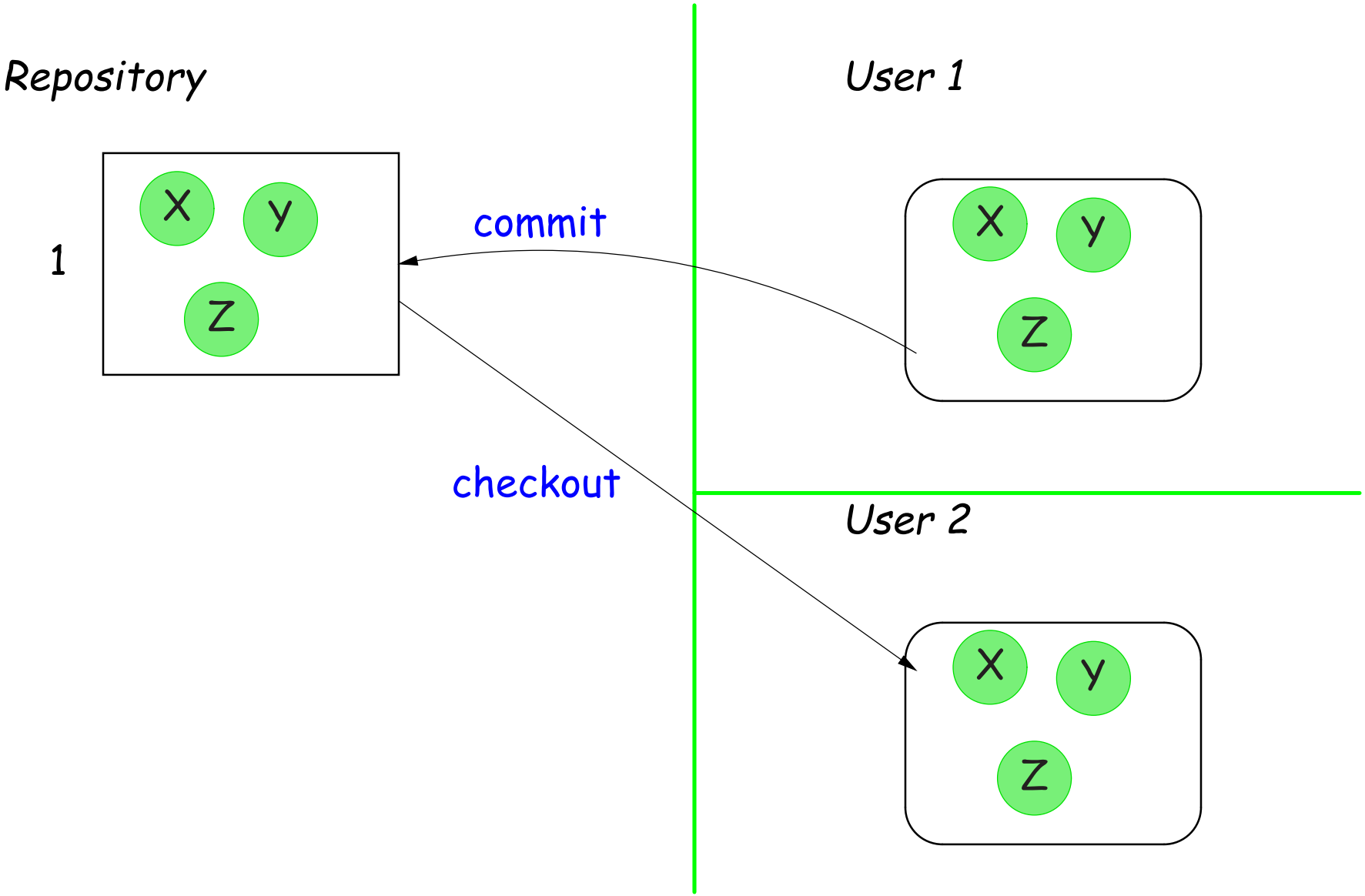
Subversion

- 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

Subversion's Model



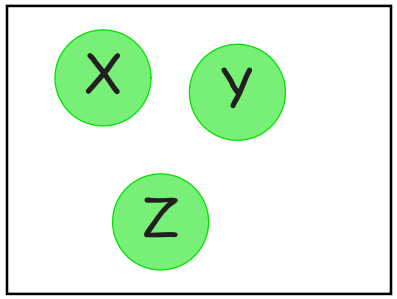
Subversion's Model



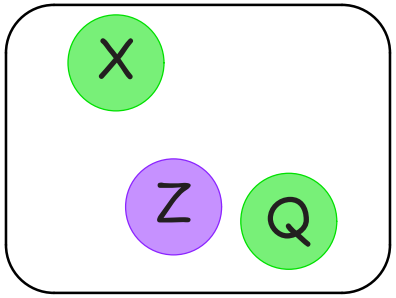
Subversion's Model

Repository

1

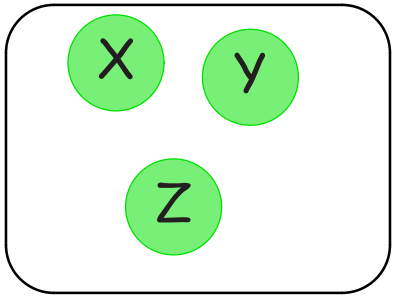


User 1

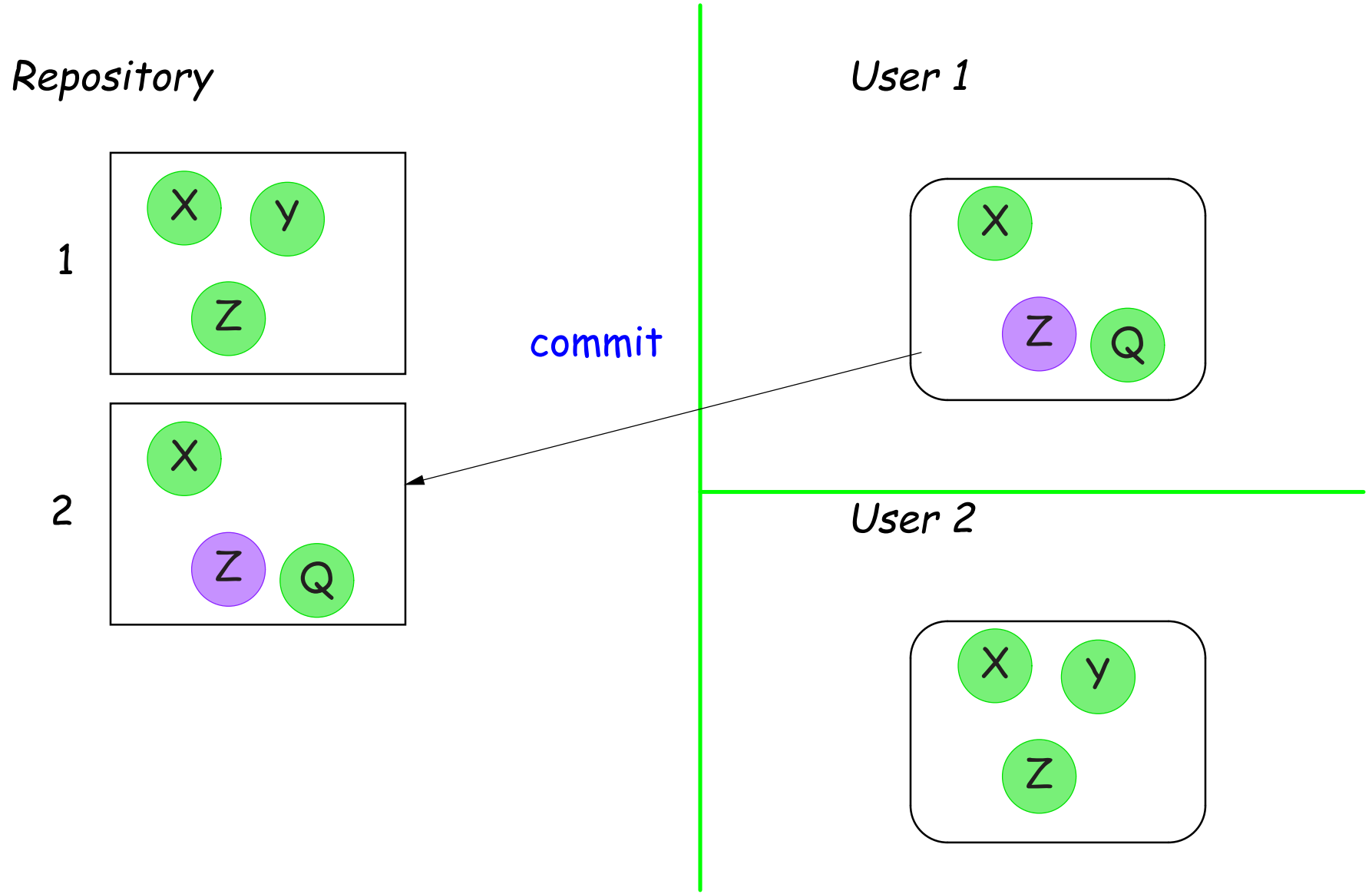


delete y
add Q
modify Z

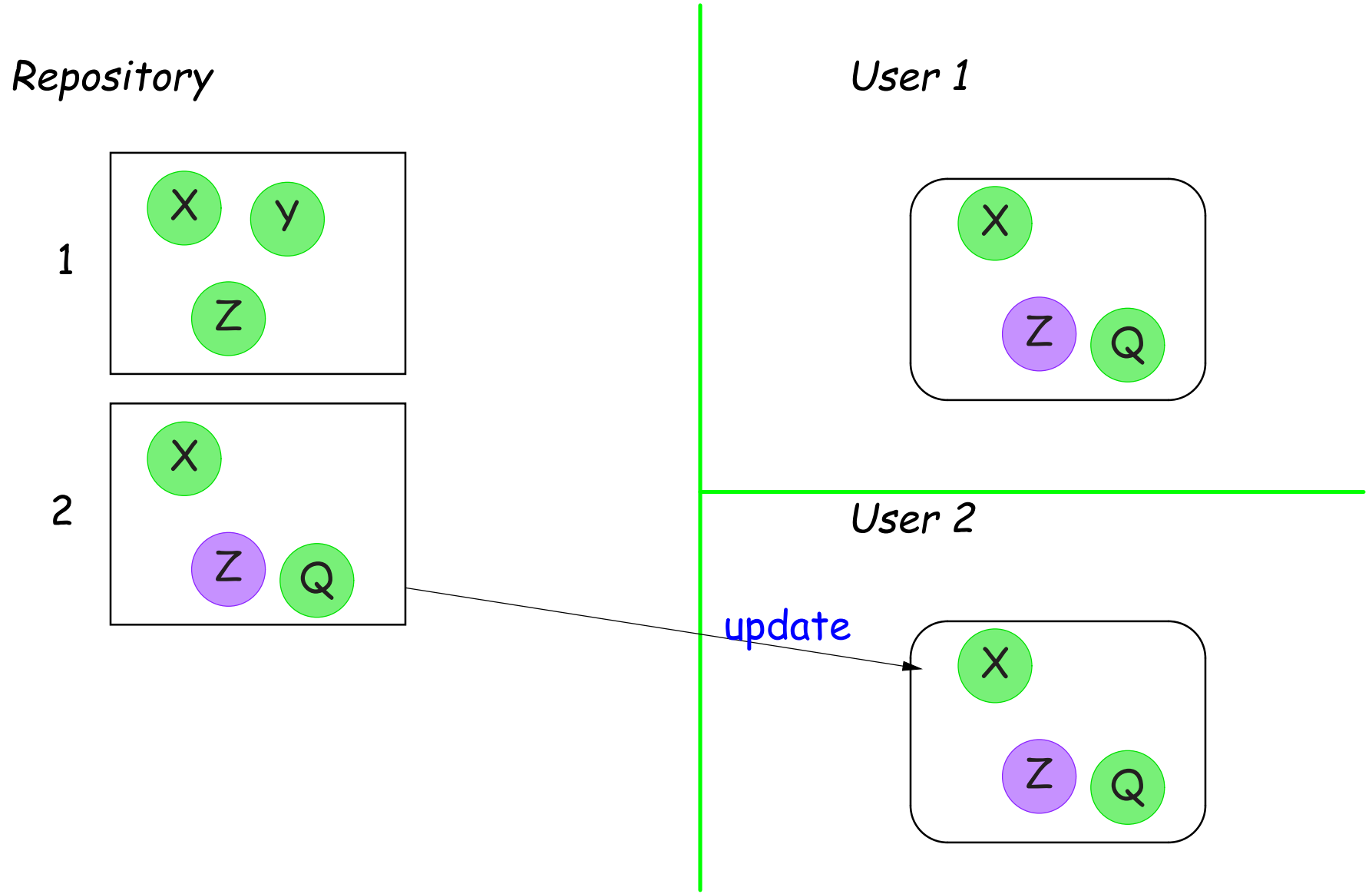
User 2



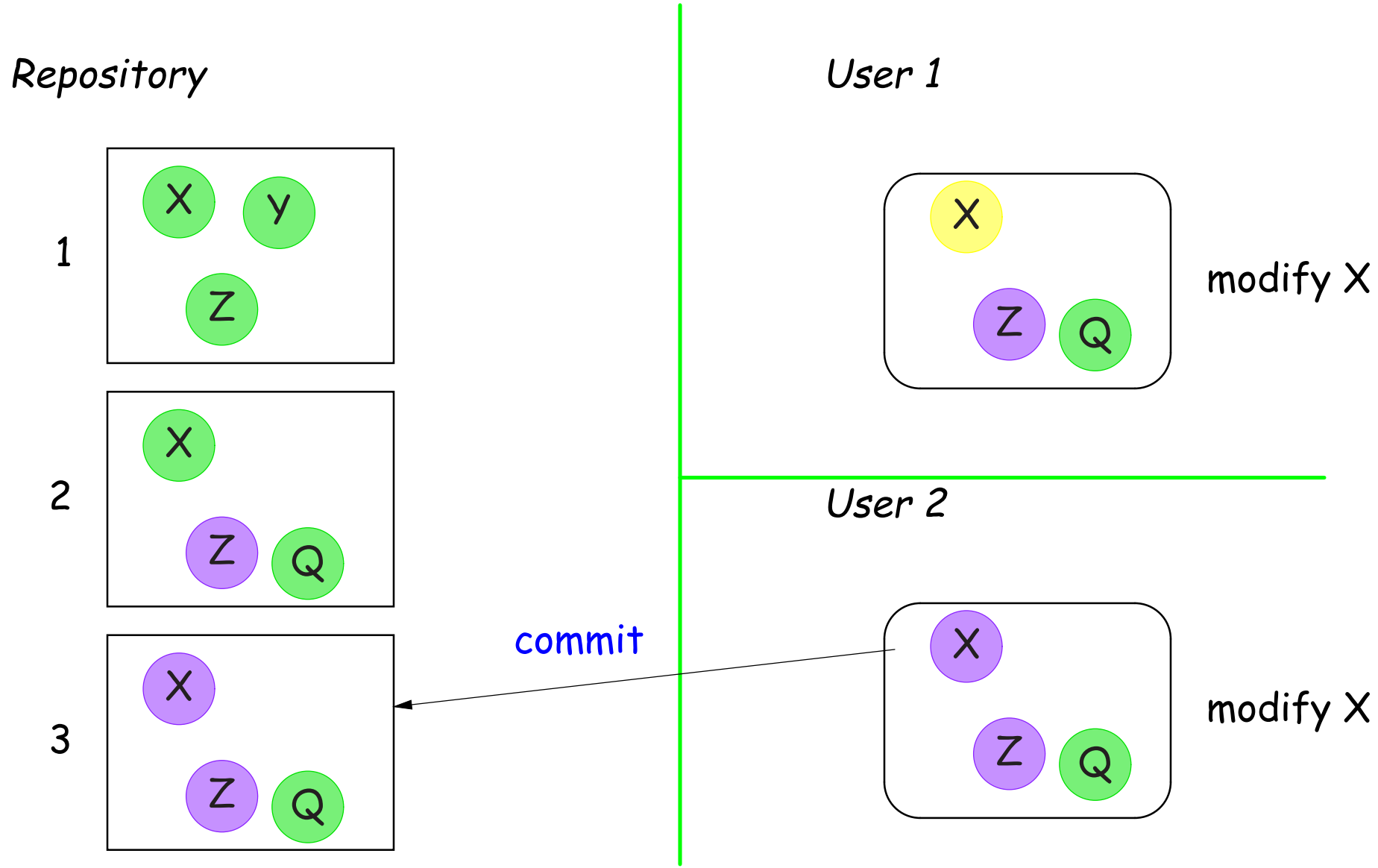
Subversion's Model



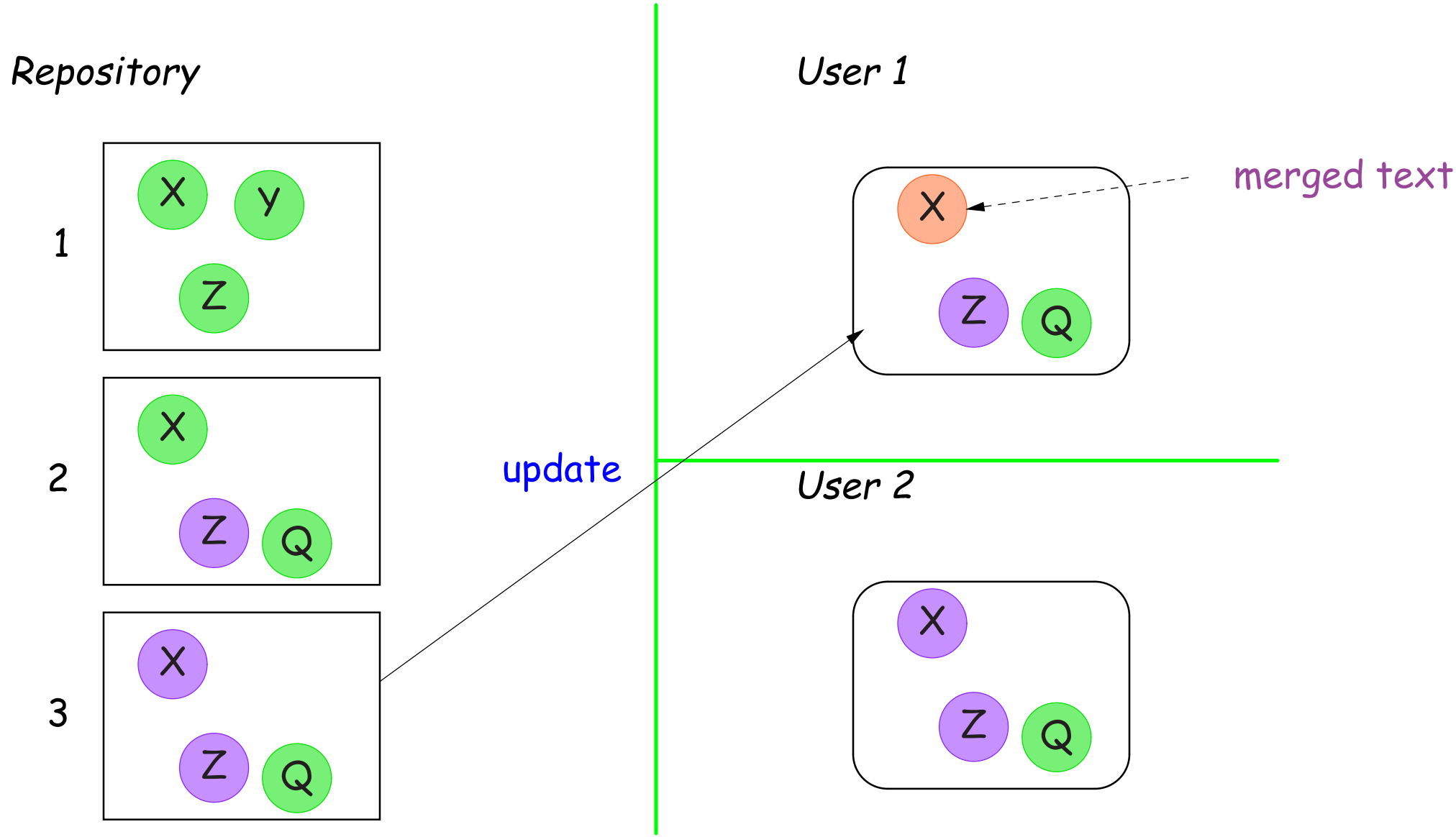
Subversion's Model



Subversion's Model



Subversion's Model



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

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"

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 torus as a convenient host.

Creating a working copy of a repository

- To get the latest revision of projects:

```
svn co svn+ssh://cs61b-ta@torus/Ursa mydir
```

- Or just one directory:

```
svn co svn+ssh://cs61b-ta@torus/Ursa/proj1 mydir
```

- A particular revision:

```
svn co -r100 svn+ssh://cs61b-ta@torus/Ursa/proj1 old1
```

Some useful (local) abbreviations

- On instructional accounts, I have defined a few shortcuts:

  ```
  $MYREPOS = svn+ssh://cs164-ta@torus.cs.berkeley/cs164-xx

  $STAFFREPOS = svn+ssh://cs164-ta@torus.cs.berkeley/staff

  $TEAMREPOS = svn+ssh://cs164-ta@torus.cs.berkeley/URSA
  ```

  I'll use these from now on.

- (For those of you with Bash shells at home, you can introduce these definitions by adding

  ```
  MYREPOS=svn+ssh://cs164-ta@torus.cs.berkeley/cs164-xx
  ```

  etc. to your `.bashrc` file.)

Abbreviated commands

- To get the latest revision of projects:

```
svn co $TEAMREPOS myteamdir
```

- Or just one directory:

```
svn co $TEAMREPOS/proj1 myproj1teamdir
```

- A particular revision:

```
svn co -r100 $TEAMREPOS/proj1 old1
```

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, just your personal working directory.
- Must commit changes to change repository.

Reverting

- Before committing, can undo adds, removes, modifications.
- The command

  ```
  $ svn revert FILE
  ```

  undoes changes to FILE.
- Reverting a modification or delete restores file.
- Reverting an add removes FILE from version control without deleting the file.

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. If you leave off the -m, will call your favorite editor, which we suggest, because...
- Log messages should be accurate and informative. They are displayed by svn log -v, and can help both you and others looking at changes understand why they happened and where things changed.

Example of log message (from GDB project)

As shown by svn log -v:

```
------------------------------------------------------------------------
r156209 | brobecke | 2009-11-17 10:39:33 -0800 (Tue, 17 Nov 2009) | 10 lines
Changed paths:
   M /trunk/gdb/gdb-head/gdb/ChangeLog.GNAT
   M /trunk/gdb/gdb-head/gdb/ada-lang.c
   M /trunk/gdb/gdb-head/gdb/breakpoint.c

Missing second location when breaking on inlined function.

        * breakpoint.c (expand_line_sal_maybe): Adjust adjust the SAL
        past the function prologue in the case where we were given only
        one SAL.
        * ada-lang.c (adjust_sal_past_prologue): Delete.
        (ada_finish_decode_line_1): Remove call to adjust_sal_past_prologue,
        already taken care of by expand_line_sal_maybe.
        (ada_sals_for_line): Likewise.
        Fixes IB17-025.  Also related to IB15-007.
```

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.

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-added
G foo5 foo5     had mods from you and in repository
                that did not overlap
C foo6          conflicts:  overlapping changes since you
                updated/checked out.
```

Notating Conflicts

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

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

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
  ```

  or the more modern and preferred form:

  ```
  svn resolve --accept=working
  ```

  to indicate resolution; then commit.
- Actually, recent versions of svn will prompt you for resolutions as you update.

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 (main development) version.
- Copy command does it:

```
svn mkdir $TEAMREPOS/branches  # If not already done
svn copy $TEAMREPOS/proj1 $TEAMREPOS/branches/Bobs-proj1
svn co $TEAMREPOS/branches/Bobs-proj1 somedir
```

  and go to work (be sure proj1 is committed first).
- 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-proj1 are completely independent of the trunk (main) version.
- Rather elegant idea: no new mechanism!

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).

Typical examples of turning in work

- You've completed hw1 and want to hand it in. Currently, it's in $MYREPOS/hw1 (and, of course, committed). Use

```
$ svn copy $MYREPOS/hw1 $MYREPOS/tags/hw1-N
```

  Or (using UNIX shorthand):

```
$ svn copy $MYREPOS/{hw1,tags/hw1-N}
# For a team:
$ svn copy $TEAMREPOS/{proj1,tags/proj1-N}
```

  where N is a unique number.

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 this directory differ?

```
svn diff -r 9:10
```

- How does Bobs-proj1 compare to revision 100 of the trunk?

```
svn diff $TEAMREPOS/branches/Bobs-proj1 $TEAMREPOS/proj1@100
```

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 (*without* committing them).
- After merging, as for update, must resolve any conflicts (then commit the merged version).
- To merge changes into current working directory (assuming you are in that directory at the moment):

  ```
  svn merge SOURCE1@REV1 SOURCE2@REV2
  ```

  where SOURCE1 and SOURCE2 are URLs (svn+ssh:...) or working directories and REV1, REV2 are revision numbers.
- For short, when sources the same:

  ```
  svn -r REV1:REV2 SOURCE
  ```

- To merge in changes that happened between two tagged revisions into current working directory:

  ```
  svn merge $TEAMREPOS/tags/{v1,v2}
  ```

Getting Information

- The command svn status is your friend. Identifies
  - changes, additions, deletions that have not been committed;
  - files, directories that have not been added
  - things you've messed up.
- To list what's in a repository directory:

```
svn ls $TEAMREPOS/tags
```

- To list revisions of a file or directory:

```
svn log FILEORDIR
svn log -v FILEORDIR # For details
```

Final thought

- If you commit early and often, system is quite forgiving. You can re-construct previous states. You can freely clean things up by deleting and checking out again.
- BUT for this to work, you must commit regularly and must make sure that everything you want is under version control (svn status)