Configuration Management With CVS

References: (https://www.cvshome.org)

- **“CVS for New Users” by Bob Arnson**
- **“Introduction to CVS” by Jim Blandy**
- **“Version Management with CVS” by Per Caderqvist et al.**

1.1 What is Configuration Management?

Traditional definition: Management of source code for software development, especially in the context of:

- Large code/artifact bases
- Multiple developers
- Multiple versions
- Desire to be able to back out of changes

Configuration refers to the way different versions of different files are put together for different system versions

Traditional definition also known as "revision control"

1.2 Revision Control: The Problem

Take a software project where:

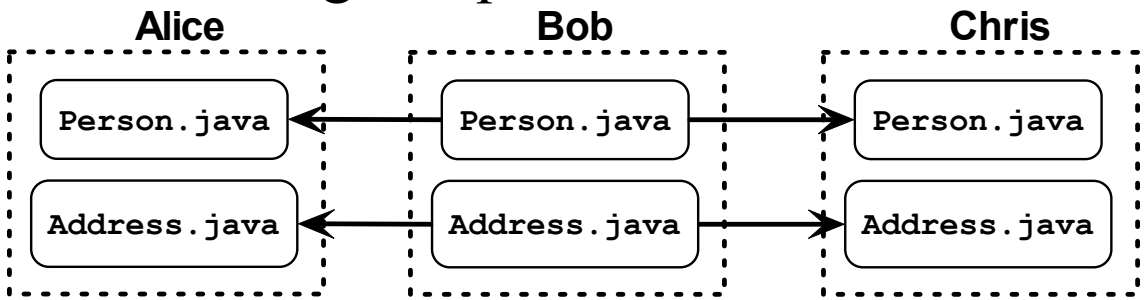
- There are several files with several people working on those files
- You want one final version of everything

Possible problems:

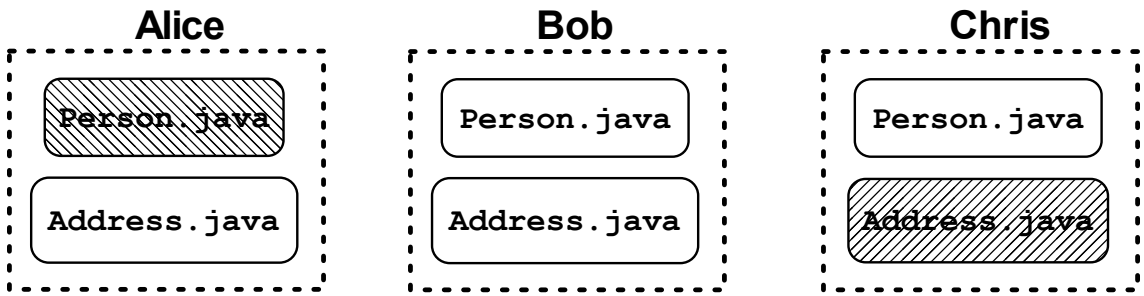
- Different people having different versions of files
- Inconsistent changes by different people
- Changes getting lost

Example 1: Conflicting Changes

1. Bob has the "master copy"
2. Alice and Chris get copies



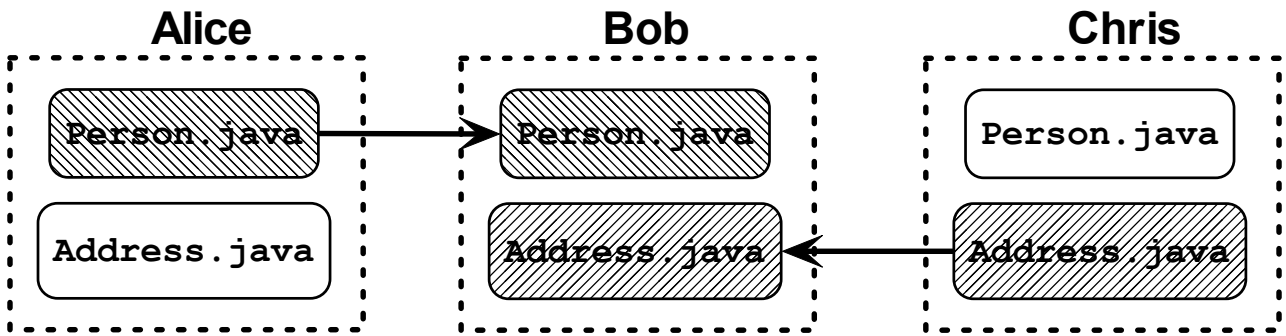
3. Alice changes **`Person.java`** and Chris changes **`Address.java`**



— Everything still works for them

Example 1: Conflicting Changes …

4. Alice and Chris send their copies to Bob



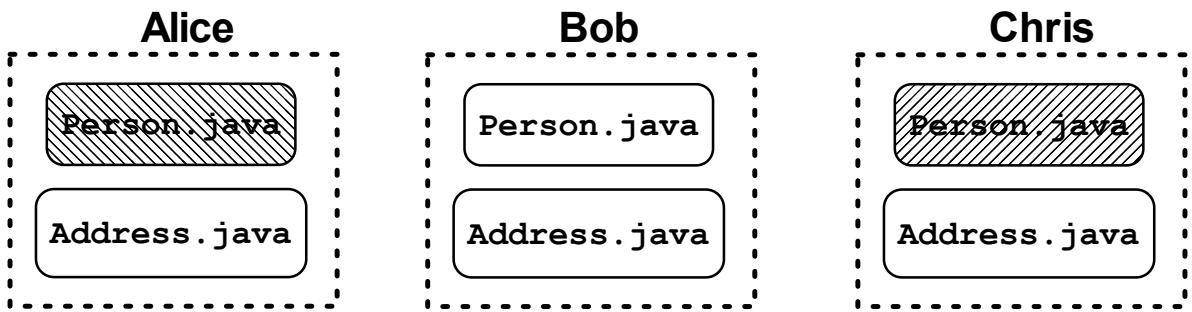
   — Bob can't compile the program anymore

5. Now there is no working master copy

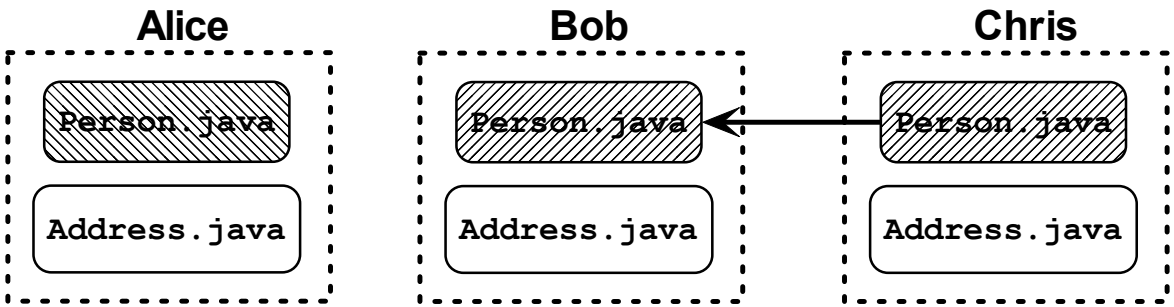
   — Have to figure out manually how to reconcile changes

Example 2: Changes Getting Lost

1. Alice and Chris both change **`Person.java`**

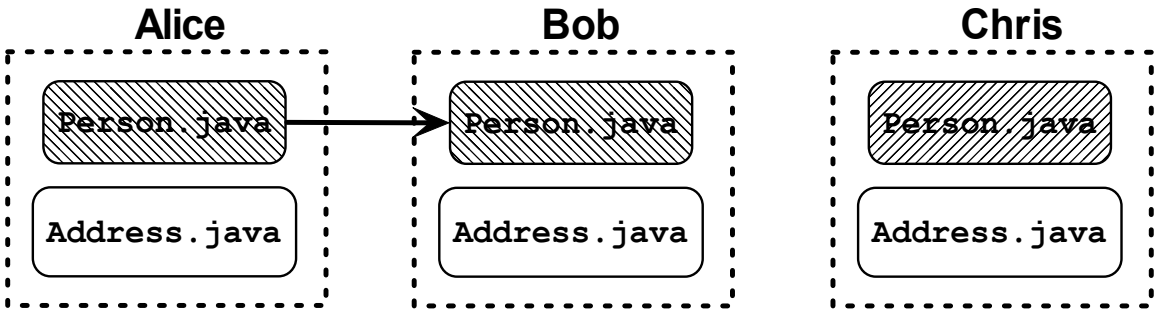


2. Chris sends his changes to Bob

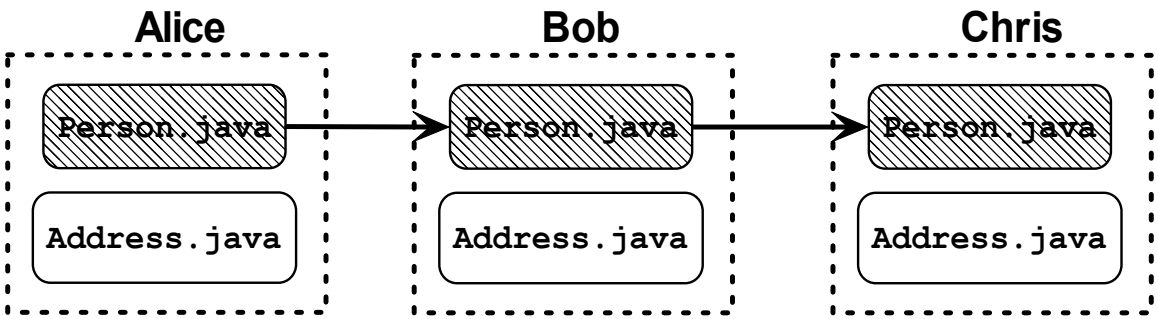


Example 2: Changes Getting Lost …

3. Later, Alice sends her changes to Bob



4. Finally, Chris gets Alice's changes from Bob



5. Now no one has Chris's original changes!

1.3 CVS Terminologies

***Revision*:** a version of a given file

- File goes through a revision every time someone edits and saves it
- File has a revision history

***Release*:** major publicized version of a system

- Release 1.0, 1.2, 2.0, etc.

***Version*:** used informally for both of the above

***Repository*:** storage space for revisions of files

- May also contain information about what revisions of what files go in what releases

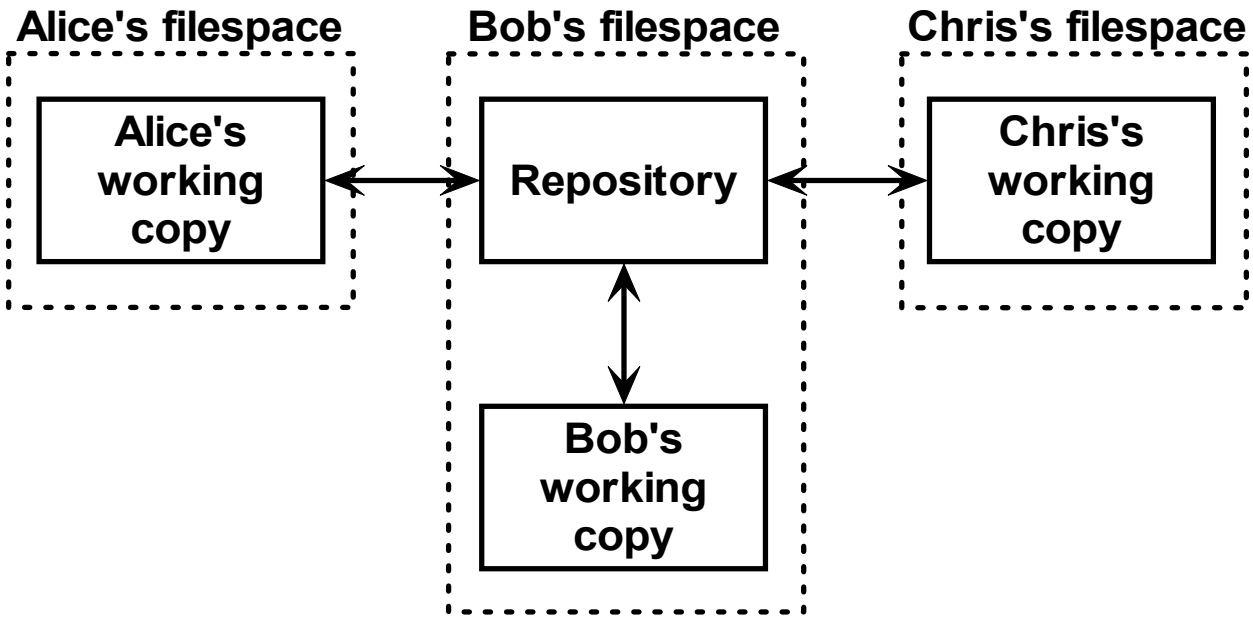
1.4 The CVS Repository

Repository lives on someone's filespace

- You never access any of the files in the repository directly
- Instead, use CVS commands to get your own copy of the files into a ***working directory***
- When you finish making changes, you check (commit) them back to the repository
- Contains information such as:
  - —Changes made
  - —What exactly was changed
  - —When changes were made
  - —Other information, such as comments

The CVS Repository …

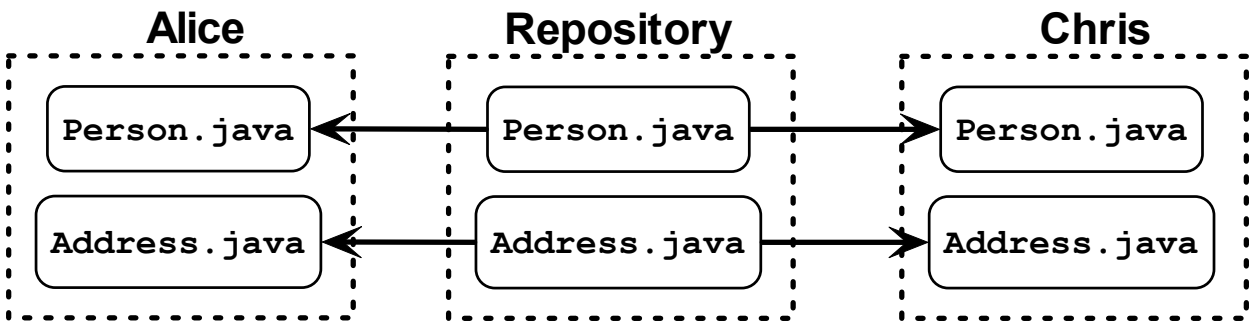
- Developers (including the repository owner) have working copies



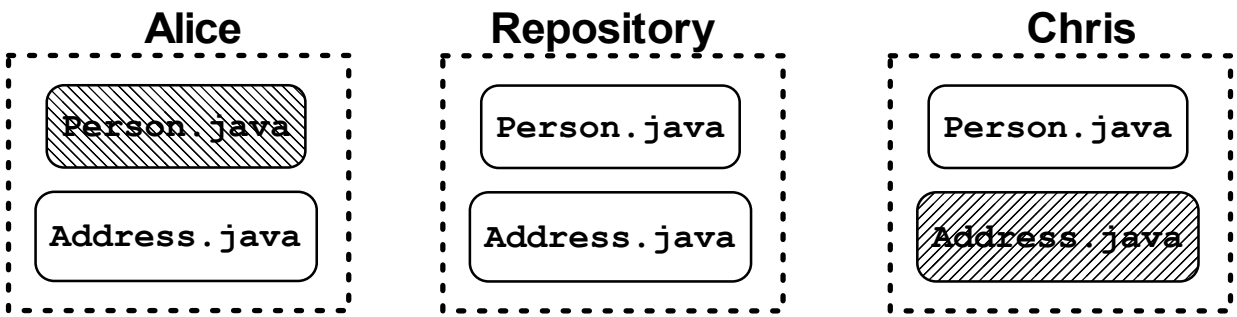
- All developers must have write access on Unix
- Networked version of CVS must exist

Example 1: Conflicting Changes - with CVS

1. Alice and Chris both check out all files

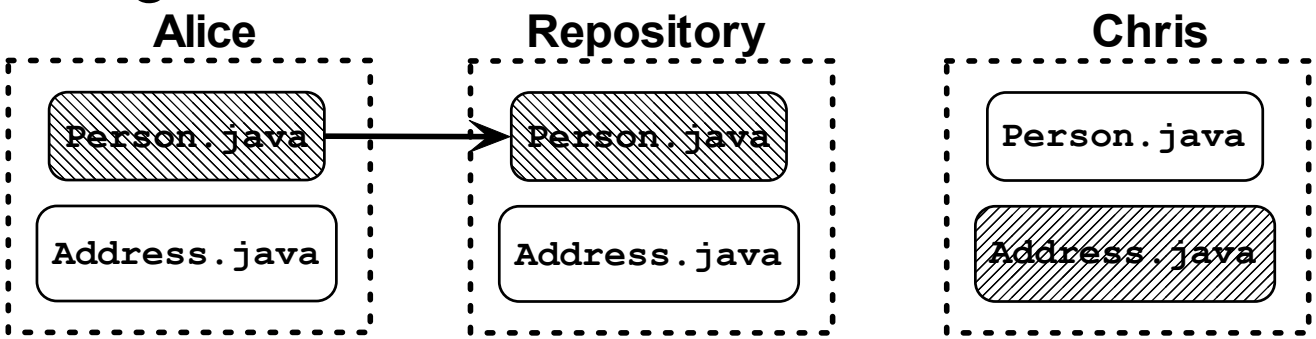


2. Alice changes **`Person.java`** and Chris changes **`Address.java`**

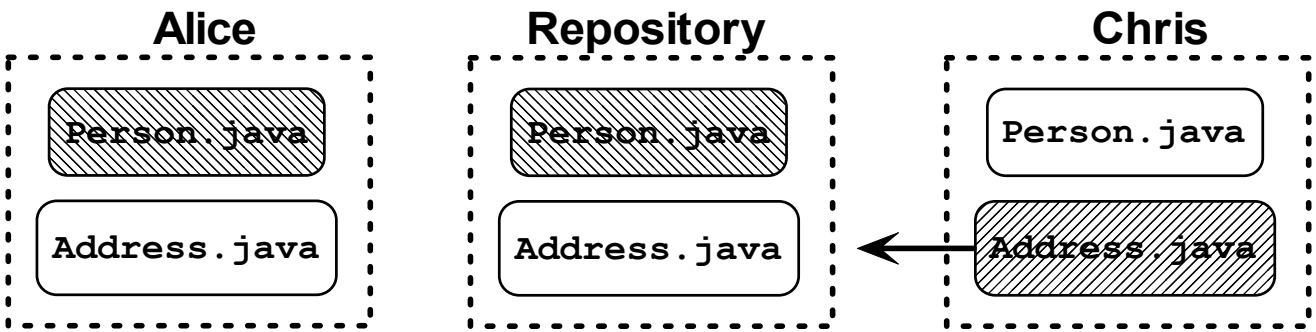


Example 1: Conflicting Changes - with CVS…

3. Alice makes sure her copy works, then ***commit*** her changes.

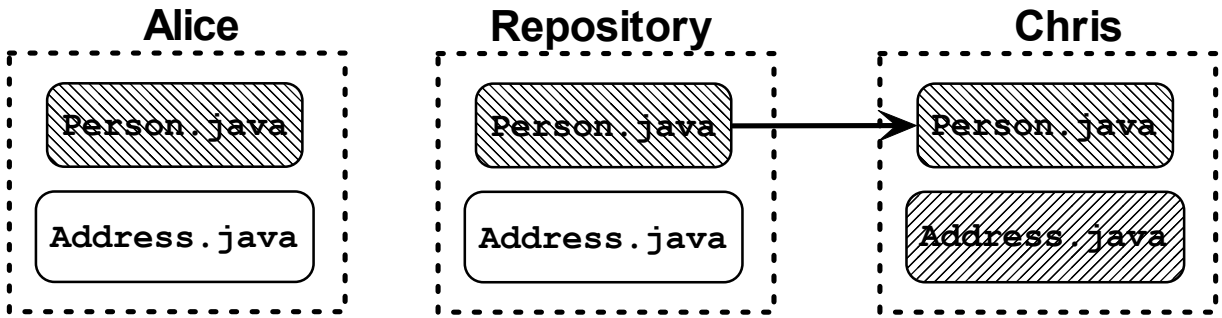


4. Chris in now blocked from committing his changes

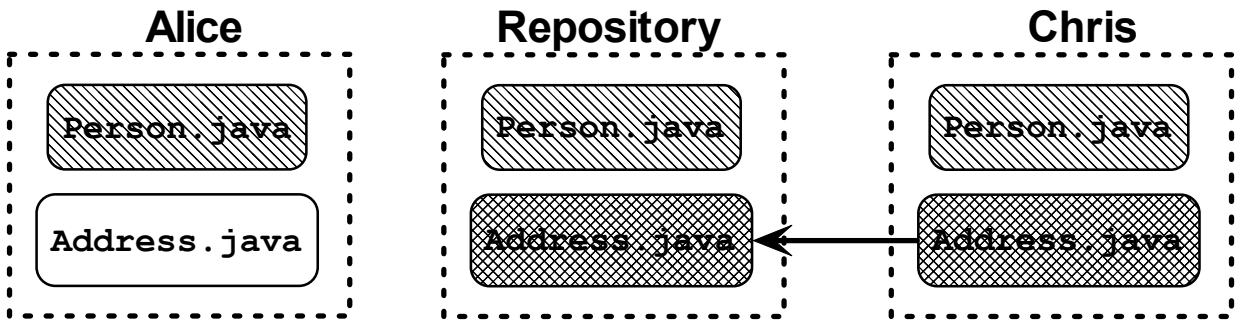


Example 1: Conflicting Changes - with CVS…

5. He must ***update*** his files first



6. After he has made sure his copy works, he can commit his changes

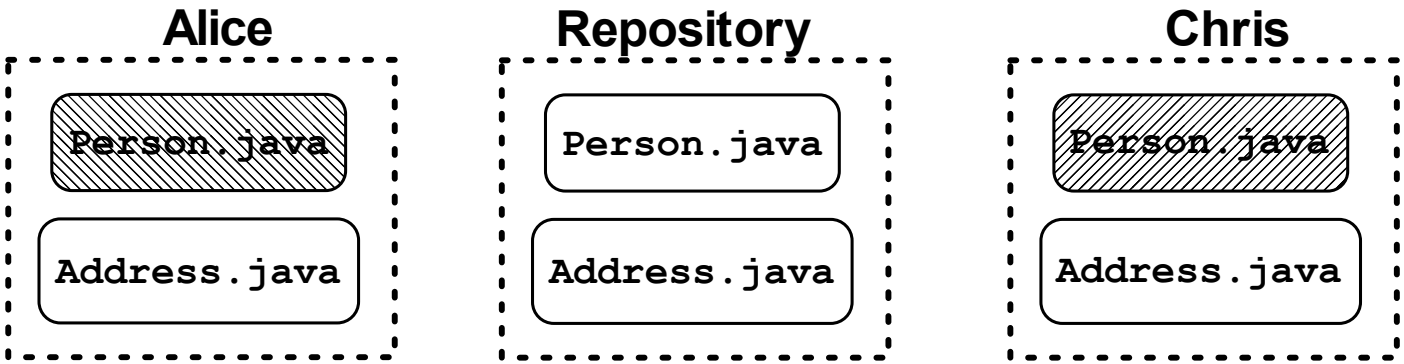


— Chris does not have to know which files to update

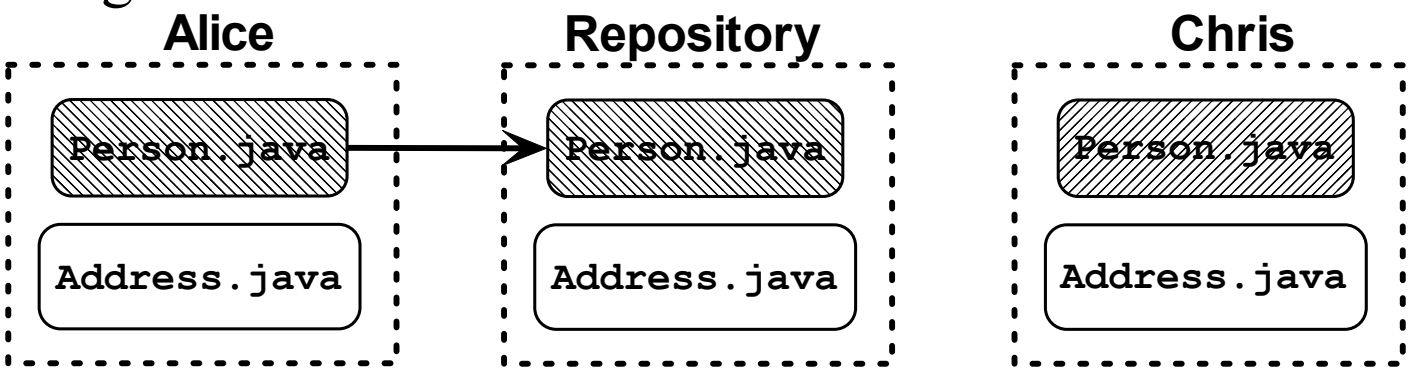
— CVS knows not to clobber his new version

Example 2: Changes Getting Lost - with CVS

1. Alice and Chris both changes **`Person.java`**

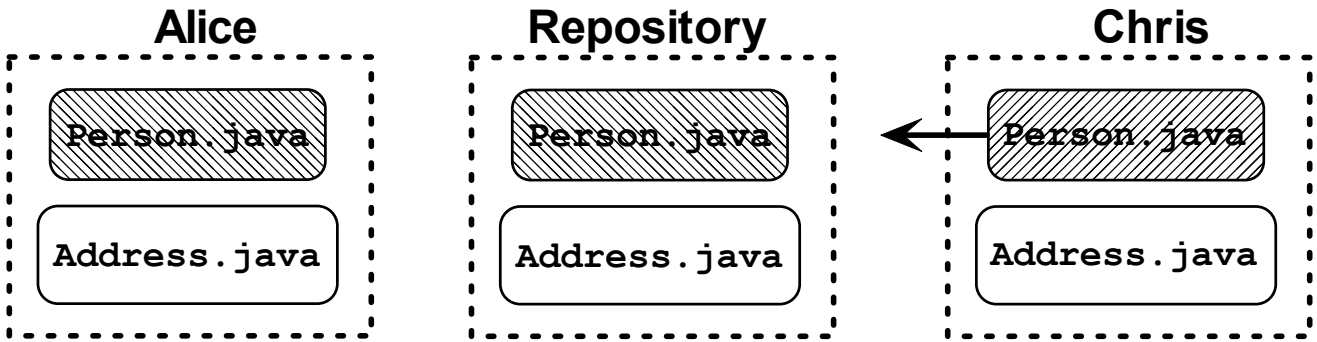


2. Alice makes sure her copy works, then ***commit*** her changes.

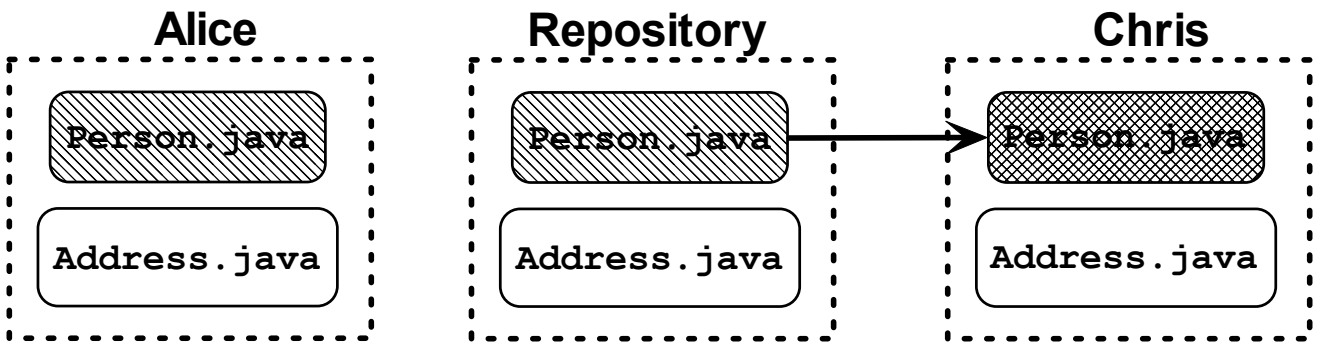


Example 2: Changes Getting Lost - with CVS…

3. Chris in now blocked from committing his changes

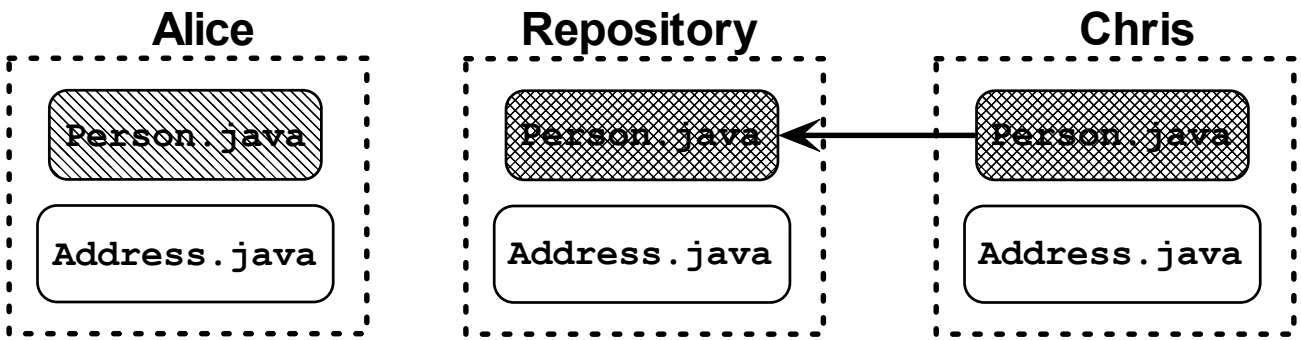


4. He must ***update*** his files first. Chris gets a revision that has both changes present and clearly annotated

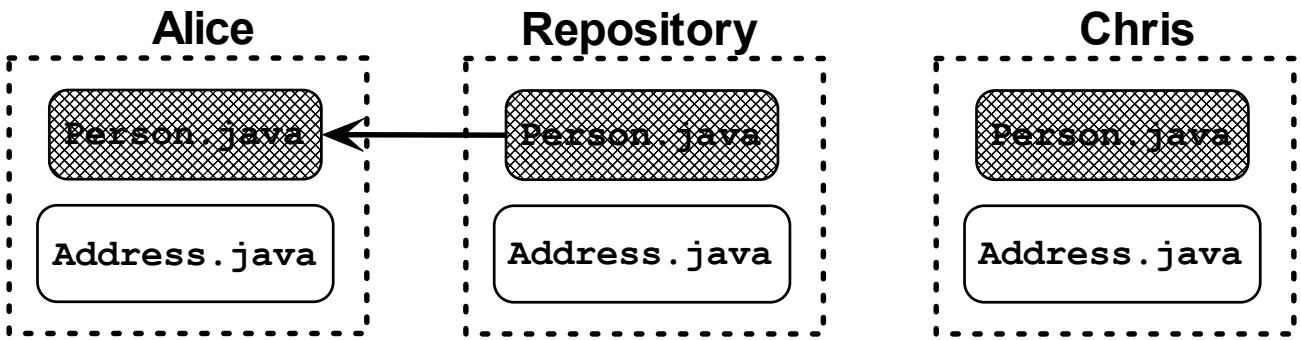


Example 2: Changes Getting Lost - with CVS…

5. Chris must resolve the conflict and remove annotation first before committing his changes



6. Alice's next ***update*** will get the new changes



1.5 Using CVS: Steps for the repository owner

1. Decide where the repository is going to be
   - e.g. **`/gaul/special/cs212/<group-name>/PROJECT`**
2. Set **CVSROOT** environment variable
   - For csh/tcsh:

     `setenv CVSROOT /gaul/special/cs212/<group-`
   - For sh/bash:

     `CVSROOT=/gaul/special/cs212/<group-name>/PROJECT`

     `export CVSROOT`
3. Create repository

   `cvs init`
4. Make repository accessible

   `chgrp -R <group-name> $CVSROOT`

   `ROOT`

Using CVS: Setting Up Initial Version for the Group (By one member)

1. Choose a name for the project
   - e.g. **`cs212project`**
2. Go to the directory with the initial version of the code
3. Issue command:

```
cvs import -m "CS212b Project" cs212project <group-name> start
```

   - The thing in the double-quotes is a readable description
   - Every checkout from now will use **`cs212project`** as the root directory of the system
   - **`<group-name>`** is the "vendor tag"
   - **`start`** is a "release tag"

Using CVS: Setting up your working copy (by all project members)

1. Set **CVSROOT** environment variable, just as the owner did (see Step 2 of the repository owner)
2. Go to the directory (***working directory***) where you want directory in
3. Issue command:

   ```
   cvs checkout cs212project
   ```

4. You will get all the files
   - Working copy
   - You can make changes
5. You also get a **CVS** directory
   - Administrative files

Using CVS: Other commands

- Say "**`cvs commit`**" to commit your changes
  - —This will first invoke an editor for adding comments
  - —To avoid invoking an editor, use:
    - "**`cvs commit -m "message"`**"
  - —It will figure out which files to commit
  - —It will block your commit if you need to update

- Say "**`cvs update`**" to get the latest update
  - —It will tell you which files were updated
  - —It will warn you if there are conflicting changes

Using CVS: Other commands …

- If you have a new file to add, say "**`cvs add <fname>`**"
  - —It has not added it to the repository yet
  - —"**`cvs commit`**" will commit the add
- Similarly, "**`cvs remove <fname>`**" will flag the file for removal
  - —"**`cvs commit`**" will commit the removal
- "**`cvs diff`**": shows the differences between your file and what is in the repository, or between versions
- "**`cvs export`**": like "**`cvs checkout`**", but without admin directories (i.e. for giving a copy to the client)
- "**`cvs history`**": brief history
- "**`cvs log <fname>`**": exhaustive history of ***fname***