Managing Changes with CVS

COMP 3111/H tutorial

What is version control?

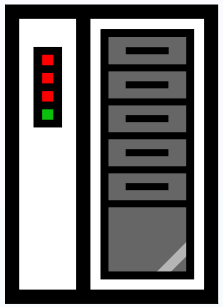
- A version control system (also known as a revision control system) is a repository of files, often the files for the source code of computer programs, with monitored access
- Every change made to the source is tracked, along with who made the change, why they made it, and references to problems fixed, or enhancements introduced

What is CVS?

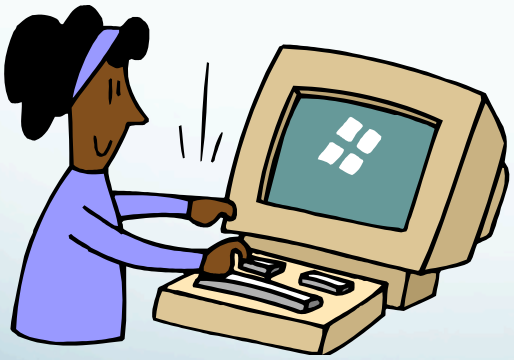
- CVS = Concurrent Versions System
- Advantages
  - Open source
  - Client and servers available for just about any platform OS X, Windows, and Linux

What does it look like?

The “Repository”



All Versions of all the files



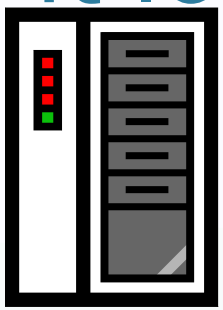
Alice’s Local Version

Joe’s Local Version

Jane’s Local Version

What does it look like?

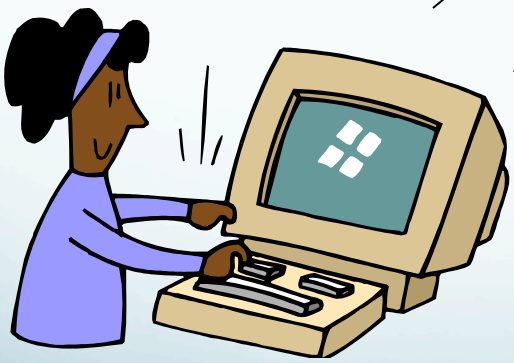
The "Repository"



All Versions of all the files

Add a new file (or directory) that everyone can see

"OK" or "File already exists"



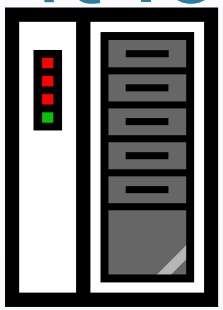
Alice's Local Version

Joe's Local Version

Jane's Local Version

What does it look like?

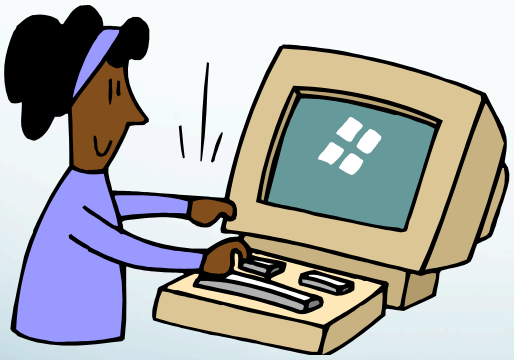
The “Repository”



All Versions of all the files

Commit my changes so everyone can see them

“OK” or “Your files are out of date”



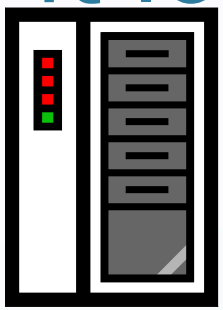
Alice’s Local Version

Joe’s Local Version

Jane’s Local Version

What does it look like?

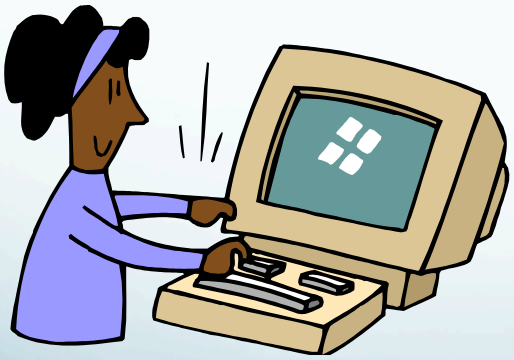
The “Repository”



All Versions of all the files

Update my local files to reflect all changes

“OK” or “Your files conflict”



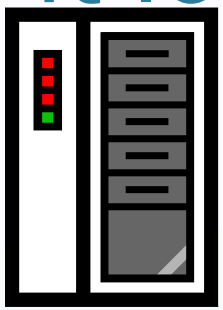
Alice’s Local Version

Joe’s Local Version

Jane’s Local Version

What does it look like?

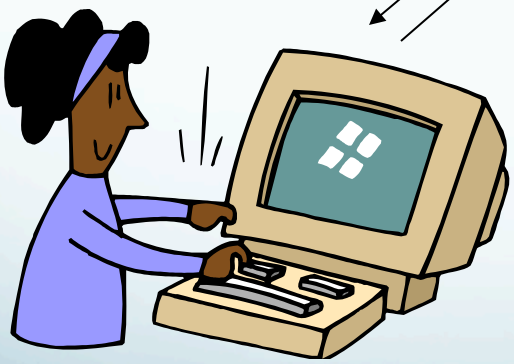
The “Repository”



All Versions of all the files

Call what I have
“Release 1.0”

“OK”



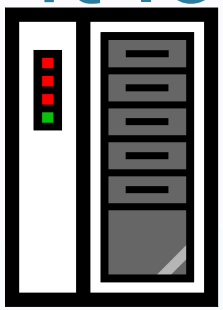
Alice’s Local Version

Joe’s Local Version

Jane’s Local Version

What does it look like?

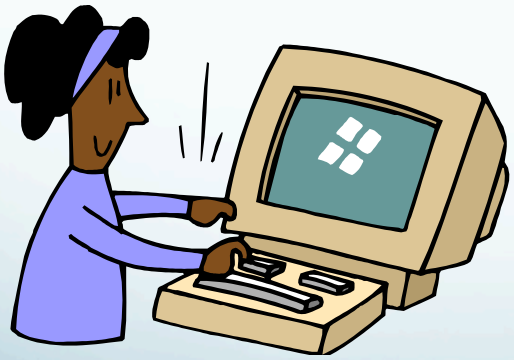
The “Repository”



All Versions of all the files

Create a separate version: “Release 1.0.joe” for local testing

“OK”



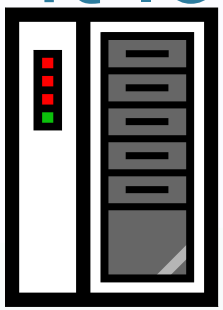
Alice’s Local Version

Joe’s Local Version

Jane’s Local Version

What does it look like?

The “Repository”

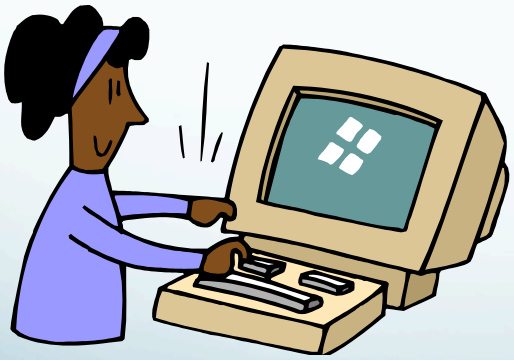


All Versions of all the files

Six Months Later

Merge my special version back into the main product

“OK”



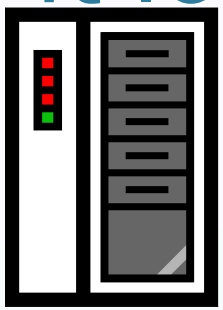
Alice’s Local Version

Joe’s Local Version

Jane’s Local Version

What does it look like?

The “Repository”



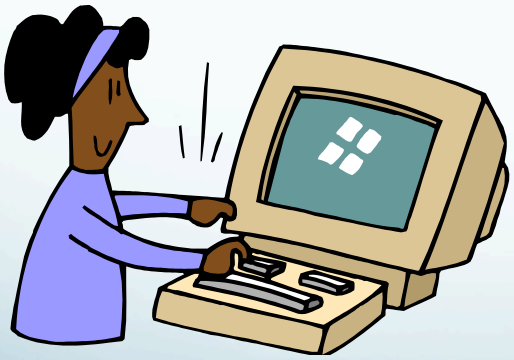
All Versions of all the files

One Year Later

Release 1.5-Joe causes injuries!

Give me “Release 1.0”

“OK”



Alice’s Local Version

Joe’s Local Version

Jane’s Local Version

CVS The Easy Way

- In Linux or Unix-like OS
  - Usually accessible via command line
  - Run “cvs”
- Windows
  - Use cygwin OR minGW
  - Use GUI-based CVS clients (e.g. WinCVS or Eclipse)

Setting up CVSROOT

- For personal usage
  - Create an empty directory (e.g. /homes/cspeter/MYCVS)
  - Create a subdirectory "CVSROOT"
- For group usage
  - Make sure that the directory is owned by your group
  - A group stick bit should be added (It is a security measure to avoid deletion of critical content in Unix-like OS)
  - Example:
    - chown cspeter mycvsgroup /homes/cspeter/MYCVS
    - chmod g+s /homes/cspeter/MYCVS

Terminology (part 1)

- Repository - area on the server where the files are stored
- Checkout - act of getting a local a copy of the latest version of the code from the repository
- Commit - saving the changes to your local file(s) to the cvs repository
- Update - getting code changes that have been committed since you checked out the project
- Merge - combining changes between two versions of the same file

Terminology (part 2)

- History - shows a list of commit messages, times and, who committed for a particular file
- Revision - cvs assigned version number for a file
- Tagging - a way to mark a group of files and revisions
- Branching - a way to work on multiple copies of your application

Repository

- The CVS server manages a *repository* containing a number of *modules*
- A module contains directories and *versioned files*. A versioned file represents the history of single file and contains a number of *revisions*
- Whenever a change is made to the file, a new revision is created
- CVS also records the user who created the revision, their comment for the revision, and a revision number. Revision numbers are automatically updated; in most cases, the first revision of a file is 1.1, the second 1.2, and so on.

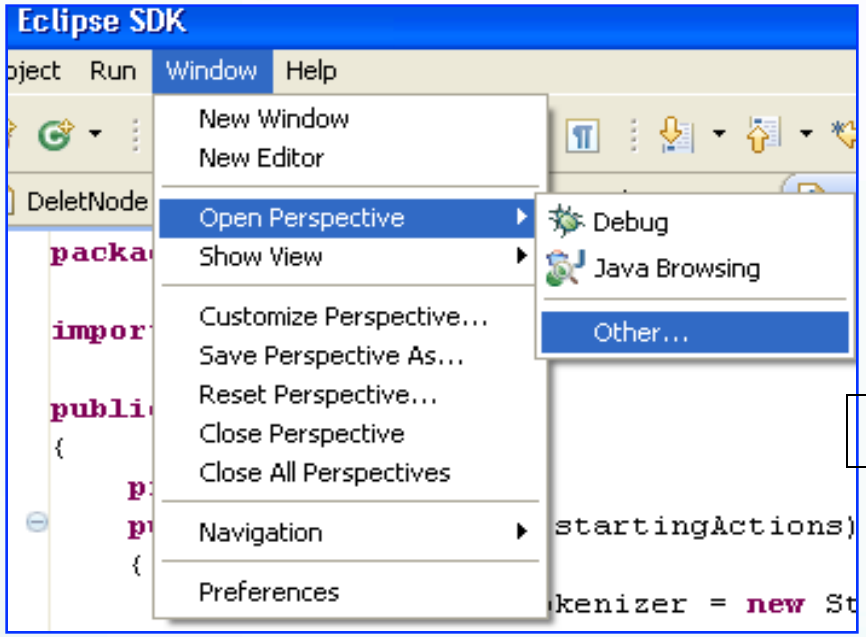
Check out, Synchronise, update and commit

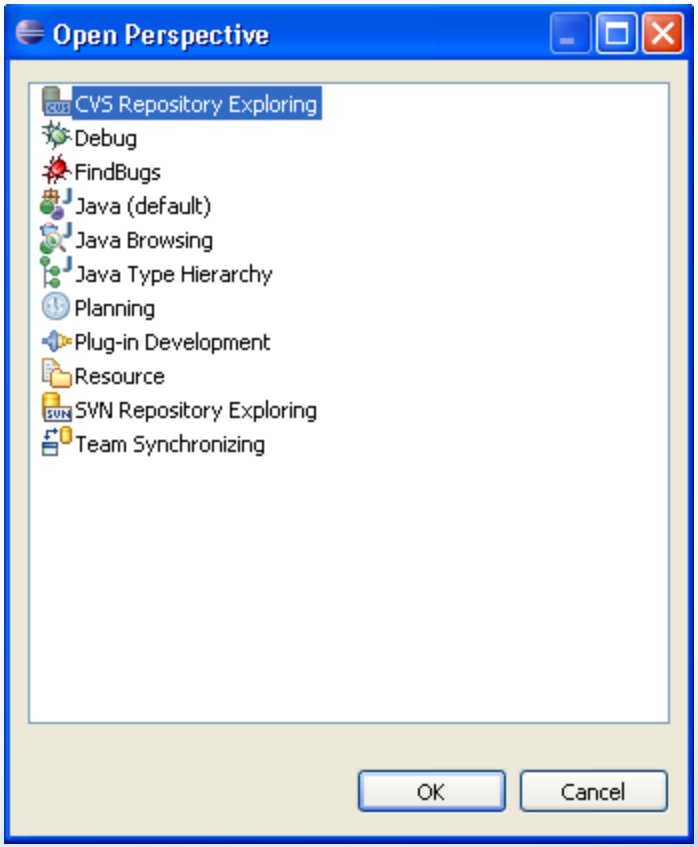
- *Check out* modules after connecting to the repository. Copy the latest module contents from CVS to local machine, giving a working copy of the module
- Use CVS client to *synchronise* local working copy with the repository
- The client can *update* local working copy with some or all of the incoming changes. If you have write permission for the module, you can also *commit* (*check in*) some or all of the outgoing changes.

Using CVS in Eclipse

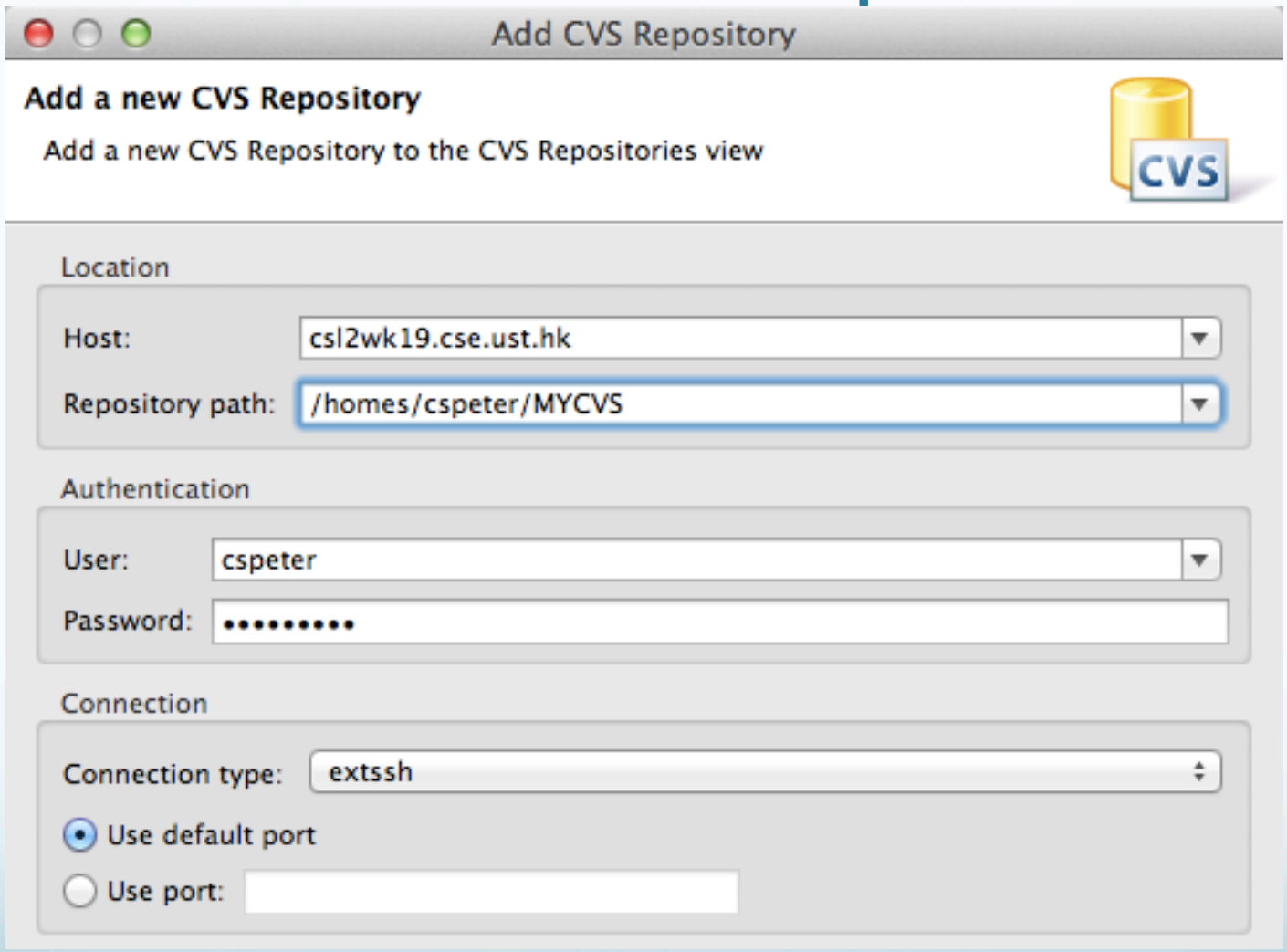
- Open Eclipse
- Window>Open Perspective>Other>CVS Repository Exploring
- Add CVS Repository
  - Host: csl2wk01.cse.ust.hk – csl2wk40.cse.ust.hk
  - Repository Path: (More details in the lab)
    - Any directory with group access right and a subdirectory of "CVSROOT"
    - Example: /project/comp211_stu/cs211g302
  - Connection type: extssh

CVS in eclipse

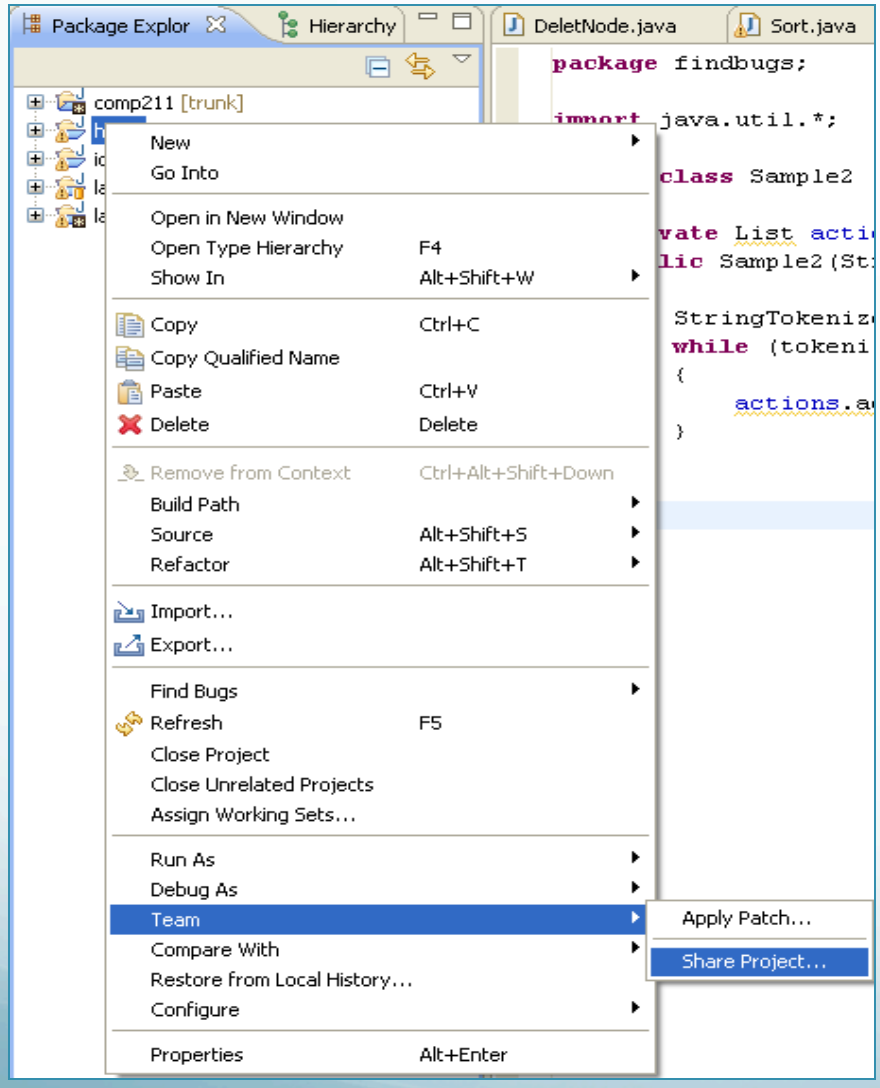


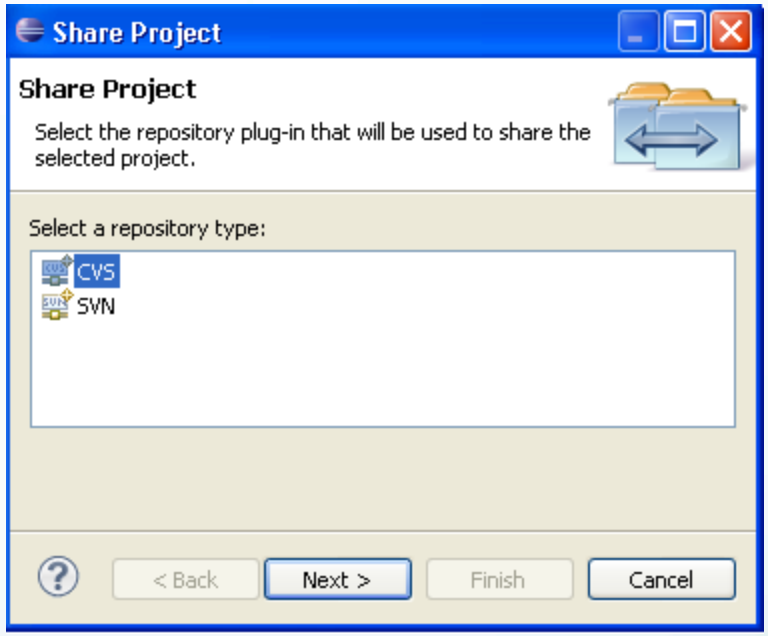


CVS in eclipse

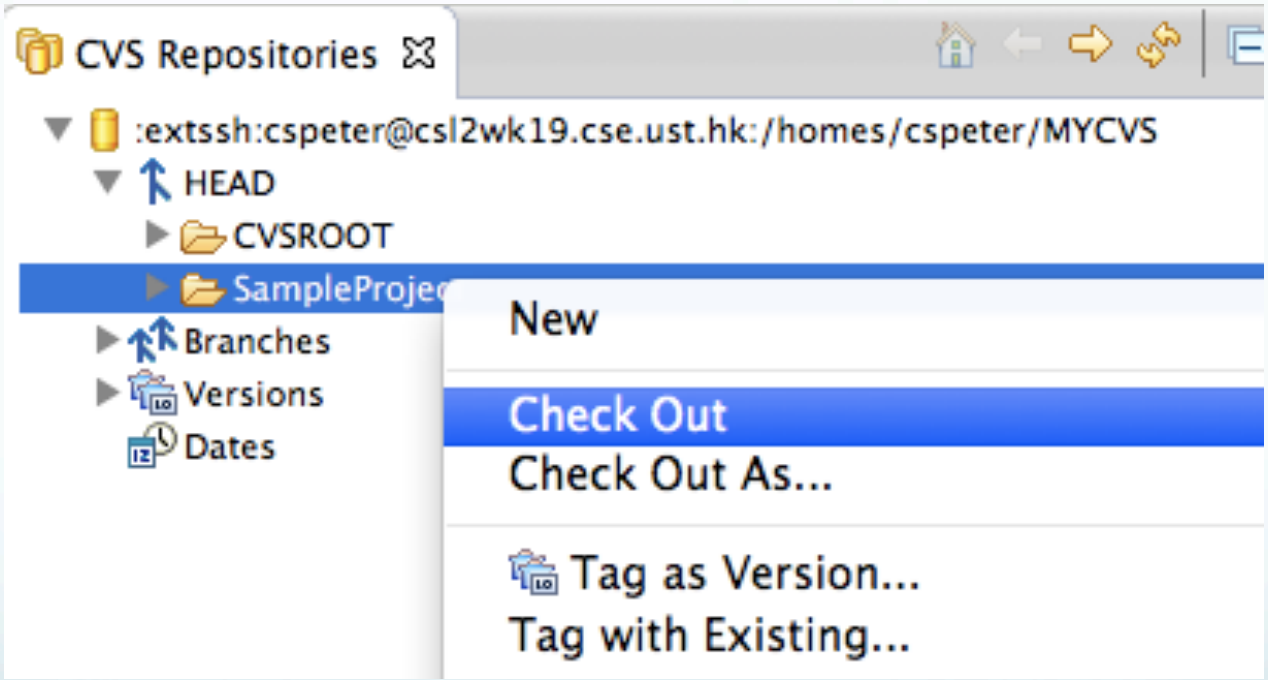


Import (for project owner)





Check out (for other team members)



Example: Conflicts

- Conflicts arise when developers A and B checkout a lcoal copy and edit the same file (e.g. HelloWorld.java)
- Both A and B are working on the local copy with an empty implementation of method "void sayHello()"
- Based on the local copy, A implements "sayHello" and commits to the CVS repository
- Based on the local copy, B implements "sayHello" and commits to the CVS repository

Merging conflicts

- When a conflict is detected for a file, latter developer must merge the conflicting changes before the file is committed.
- Conflict handling
  - CVS can automatically merge changes which affect different lines in the file
  - If the changes affect the same lines, a manual merge must be performed

Tags

- When developing, often reach milestones such as:
  - "Basic system working, all tests pass"
  - "Before refactoring account processing"
  - "After refactoring account processing"
  - "Release 1.4"
  - "Fixed defect D113”
- Marking these milestones is essential for tracking changes and ensuring that we can revert code to a previous state if required

Tags

- In CVS, mark milestones by tagging the module. The tag captures a single revision for each file in the module:

Branches and the HEAD

- Branches are used to isolate streams of development. Unless otherwise specified, checkouts and commits operate on a default branch called HEAD (a.k.a. trunk)
- Branches must always start from a tag, called the root tag
- The most common use for branches is to manage parallel development of releases.
  - For example, released version 1.3.0 and development for 2.0.0 has started on the trunk. If a user reports a critical bug in 1.3.0, we can create a branch using RELEASE1-3-0 as the root tag.
  - When we start working in the branch, the module contents will be exactly the same as they were for release 1.3.0. Our updates and commits will act on the branch only.