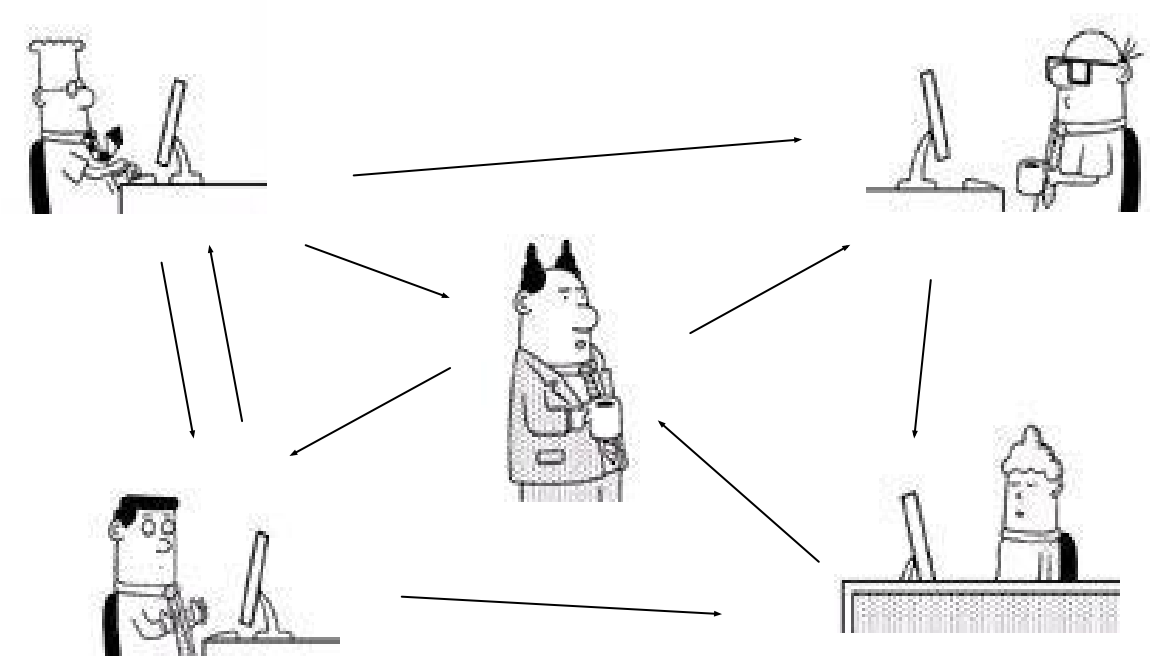
Revision control

INF5750/9750 - Lecture 1 (Part III)

Problem area

- Software projects with multiple developers need to coordinate and synchronize the source code

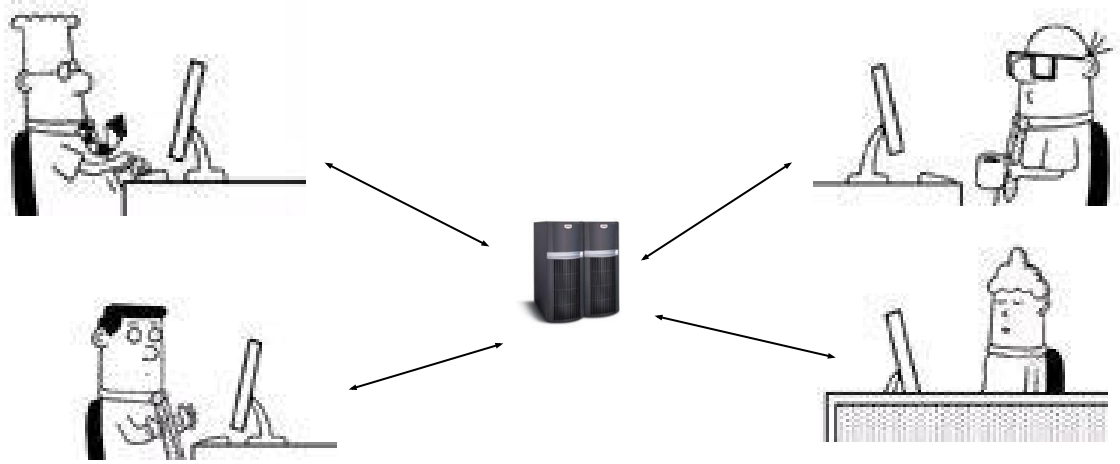


Approaches to version control

- Work on same computer and take turns coding
  - Nah...
- Send files by e-mail or put them online
  - Lots of manual work
- Put files on a shared disk
  - Files get overwritten or deleted and work is lost, lots of direct coordination
- In short: Error prone and inefficient

The preferred solution

- Use a revision control system. RCS - software that allows for multiple developers to work on the same codebase in a coordinated fashion
- History of Revision Control Systems:
  - File versioning tools, e.g. SCCS, RCS
  - Central Style - tree versioning tools. e.g. CVS
  - Central Style 2 - tree versioning tools e.g. SVN
  - Distributed style - tree versioning tools e.g. Bazaar
- Modern DVCS include Git, Mercurial, Bazaar



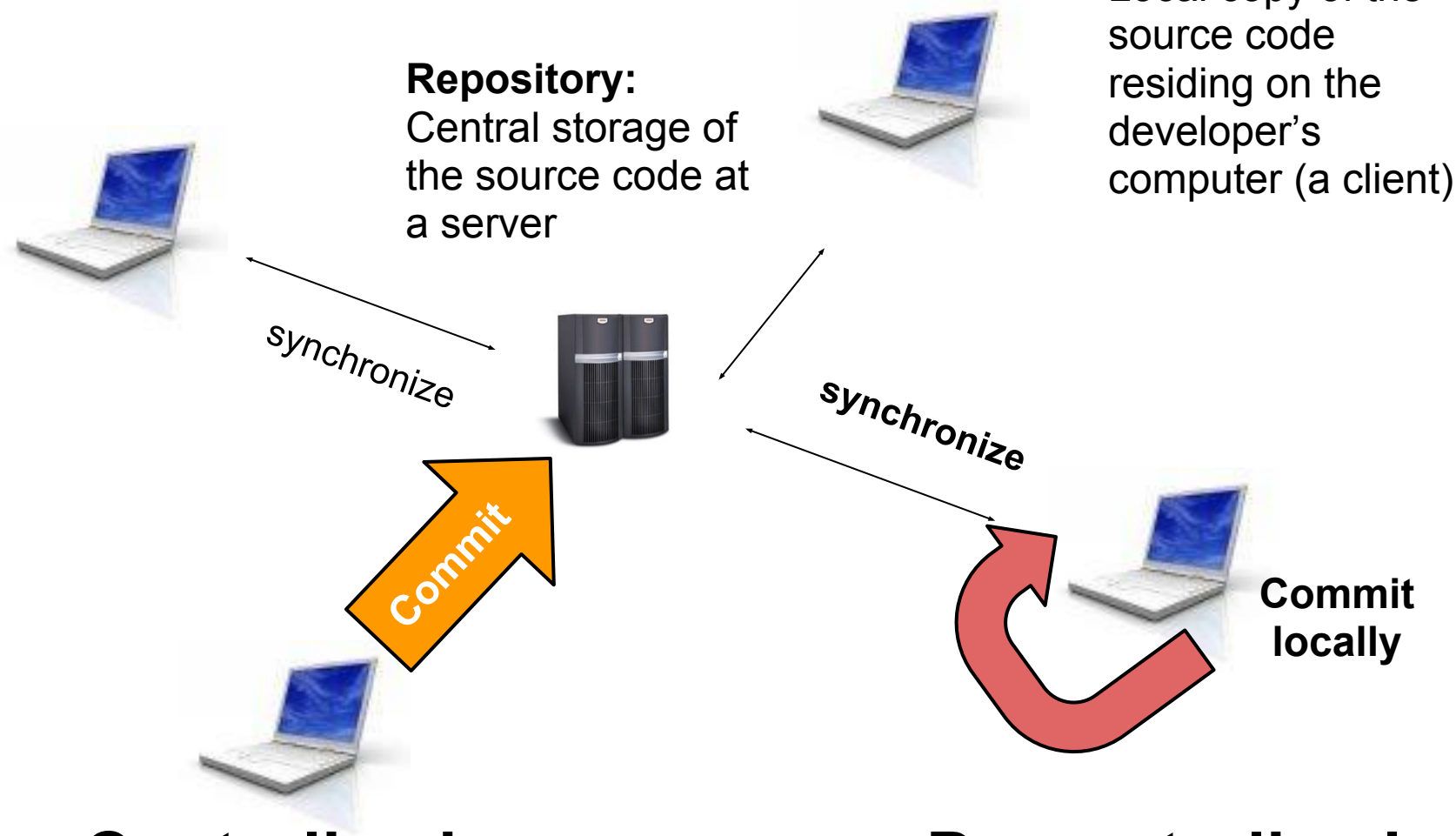
Which system in this course?

- In this course we will be using Bazaar as the version control system
- The remote repository is hosted at launchpad.net
- This is the same system as used in DHIS2 and Ubuntu
- It can be used as a distributed version system or a centralized repository

How it works

Working tree:
Local copy of the source code residing on the developer's computer (a client)

Repository:
Central storage of the source code at a server

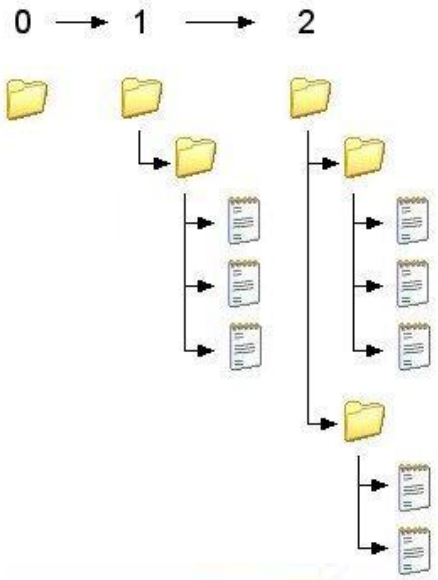


Centralized

De-centralized

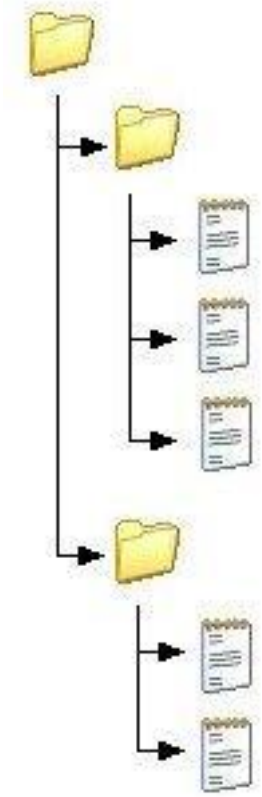
The repository

- Remembers every change ever written to it (called commits)
- You can have a central or local repository.
  - Central = big server in cloud
  - Local = on your harddisk
- Clients can check out an independent, private copy of the filesystem called a *working tree*



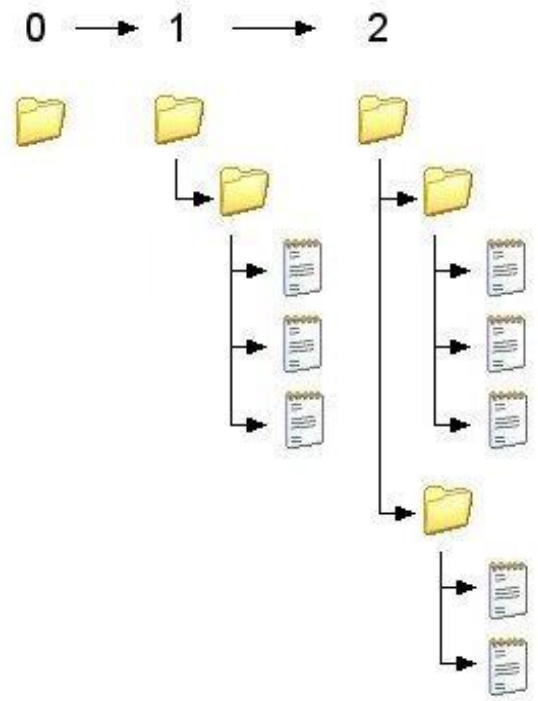
Working tree

- Ordinary directory tree
- Root directory has a hidden administrative .bzr directory, containing your local repository
- Changes are not incorporated or published until you tell it to do so



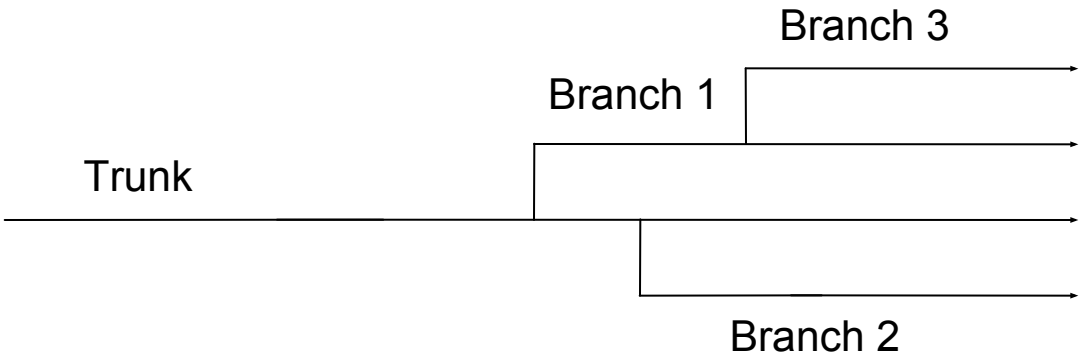
Revisions

- Every commit creates a new *revision*, which is identified by a unique revision id
- A revision identifier refers to a specific state of a branch's history forms a revision history
- Every revision can be checked out independently
- The current revision can be roll-backed to any revision
- Commits are *atomic*



Trunk and Branches

- Trunk is the original main line of development (also a branch, just called trunk for historical reasons)
- A branch is a copy of trunk which exists independently and is maintained separately
- Useful in several situations:
  - Large modifications which takes long time and affects other parts of the system (safety, flexibility, transparency)
  - Different versions for production and development
  - Customised versions for different requirements



Conflicts

- Arises if several developers edit the same part of a file
- Bazaar will try to auto-merge
- If not, you have to resolve conflicts manually

1) Developer A makes a change to Code.java and commits

2) Developer B makes a change to Code.java and tries to commit, but gets an ”out-of-date” warning.

3) Developer B updates his working copy. He will be noticed that Code. java is in a state of conflict.

4) Developer B edits and resolves the conflicts, and commits the file back in the repository

Advantages of RCS

- Concurrent development by multiple developers
- Possible to roll-back to earlier versions if development reaches a dead-end
- Allows for multiple versions (branches) of a system
- Logs useful for finding bugs and monitoring the development process
- Works as back-up

Good practises

- Update, build, test, *then* commit
  - Do not break the checked in copy
- Update out of habit before you start editing
  - Reduce your risk for integration problems
- Commit often
  - Reduce others risk for integration problems
- Check changes (diff) before committing
  - Don’t commit unwanted code in the repo

What to add to the repository

- Source code including tests
- Resources like configuration files

- What *not* to add:
  - Compiled classes / binaries (target folder)
  - IDE project files
  - Third party libraries

- Add sources, not products (generated files)!
- Use a .bzrignore file to tell Bazaar which files to ignore

Work cycle (Centralized way)

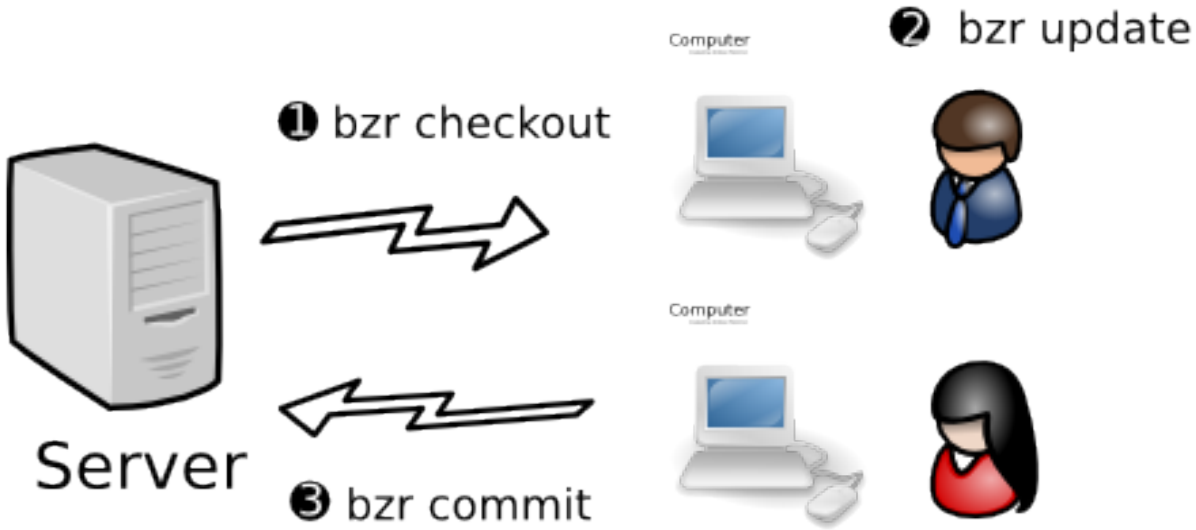
Initial check out:
The developer checks out the source code from the repository

1) Development:
The developer makes changes to the working copy

2) Update:
The developer receives changes made by other developers and synchronizes his local working copy with the repository

Resolve conflicts:
When a developer has made local changes that won't merge nicely with other changes, conflicts must be manually resolved

3) Commit:
The developer makes changes and writes or merges them back into the repository



Repository

Example remote repository locations:

lp:~<user>/<project>/<branch-name>

lp:~<user>/dhis2-academy/<branch-name>

lp:~<user>/+junk/<branch-name>

Starting a new repository

$ bzr init lp:~user/dhis2-academy/branch-name

$ bzr checkout lp:~user/dhis2-academy/branch-name

$ bzr add (adds files)

$ bzr commit -m “My first files”

You can also create a remote repository by doing a local ‘bzr init’ and then a ‘push <remote-repository>’, but you should avoid using push, at least until you know what you’re doing.

‘bzr push’ replaces the central repository with your own, so it can erase what is stored centrally.

We may cover more use of distributed bazaar repositories later in the course if there is interest.

Getting code from an existing repository

$ bzr checkout lp:~user/dhis2-academy/branch-name

(Edit project. Time passes.)

$ bzr add

$ bzr update (download any changed files)

(resolve conflicts either automatically or manually)

$ bzr commit -m “Second commit”

- Now your files are in the central repository
- This is not using the decentralized way to do it
- Instead of bzr checkout, you could use first bzr branch <repo-url> and then bzr bind <repo-url>.

Making changes to a repository

Assume that you've done either "bzr checkout <repository>" or "bzr merge <repository" + "bzr bind <repository>"

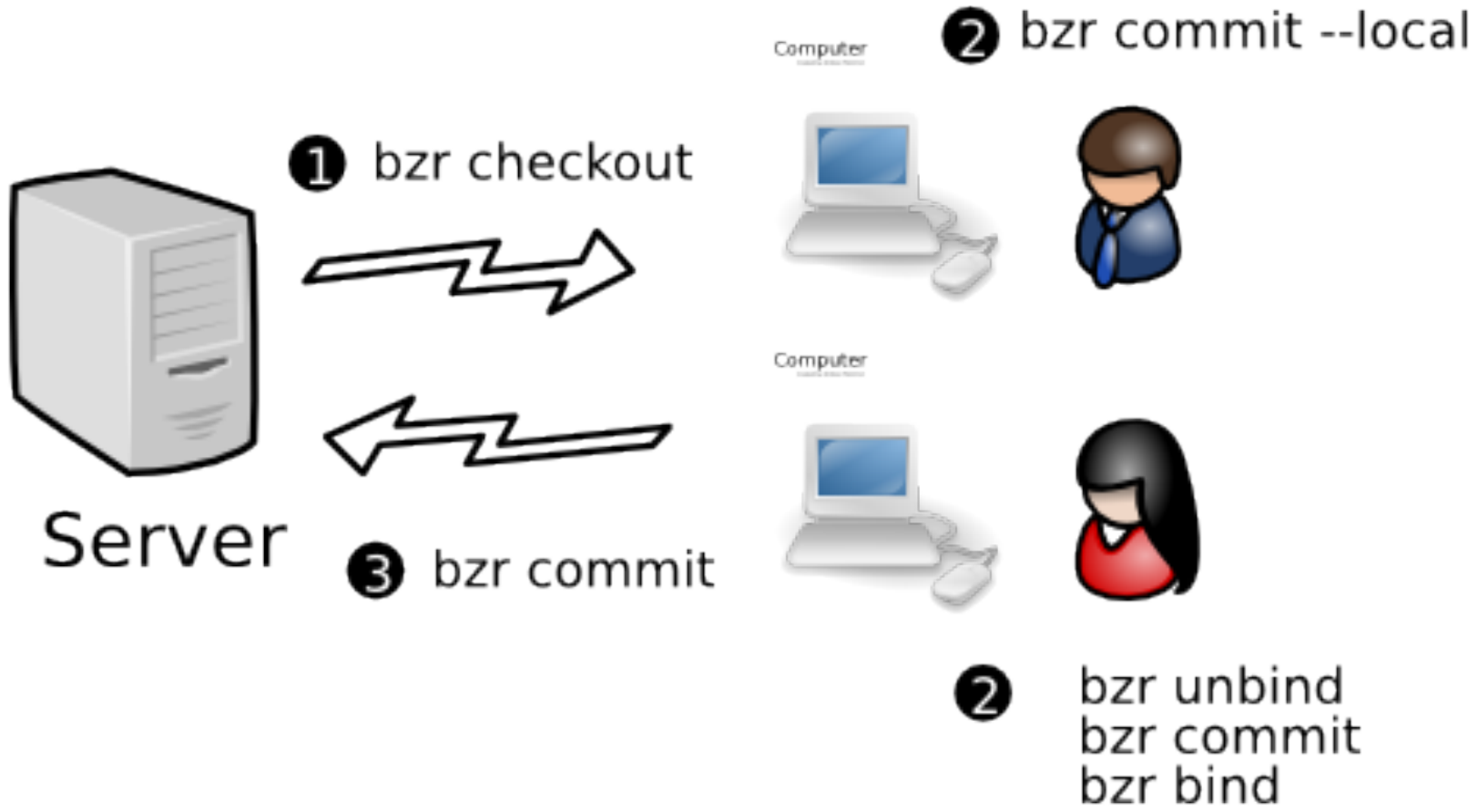
$ bzr update (it'll try to auto-merge changes)

$ bzr add (adds new files)

$ bzr commit -m "My first files"

(it may also discover conflicts and give an error message, forcing you to do bzr update and manually look through files and fix marked changes)

Decentralized - two ways

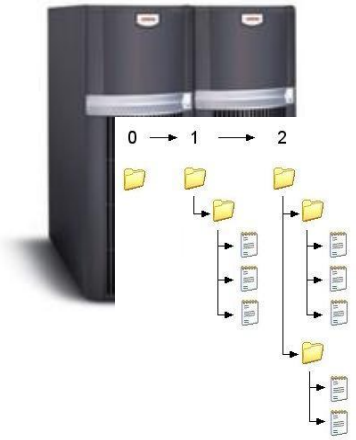


Bind and unbind

Working tree

if bound to central repository

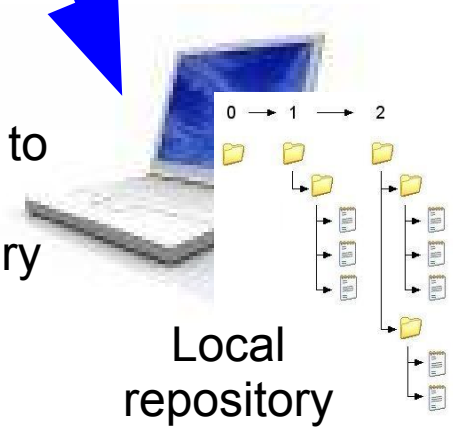
"bzr commit"



Remote repository

$ bzr bind <repository>
or
$ checkout <repository>

if bound to local repository



Local repository

$ bzr unbind
or after
$ branch <repository>

Working decentralized

You can commit locally, if you don’t want to upload to the central repository. For example if your project doesn’t compile, you should never upload to the central repository.

$ bzr commit --local -m “Some revisions”

$ bzr commit --local -m “Some more revisions”

(these will be stored locally)

… then later …

$ bzr commit -m “Stored centrally”

In the above example, you are still bound to the central repository, but you are not using it for all commits.

Creating your own branch (not bound to central repository)

$ bzr branch lp:~user/dhis2-academy/branch-name

- You are now in distributed mode.
- If you run ‘bzr commit’ now, your changes will not be checked into the central repository.
- This is great for checking out projects that you want to play with, but you are not planning to contribute to immediately.
- If you want to contribute some of your source-code, you can bind, merge etc… Or check out with ‘bzr checkout’ in a separate directory, merge your changes in there and then update from the new directory.

Working decentralized

You can unbind from the central repository

$ bzr unbind (now you're on your own)

… edit files …

$ bzr update

$ bzr commit -m "My first files" (these will be stored locally)

…

$ bzr bind <central repository>

$ bzr update

$ bzr commit -m "Stored centrally"

Bazaar offline commands

Add a file to the working copy:

$ bzr add Code.java

Delete a file from the working copy:

$ bzr remove Code.java

Compare working copy with repository on file-level:

$ bzr status

Compare working copy with repository on code-level:

$ bzr diff

Revert a file to the state from last commit

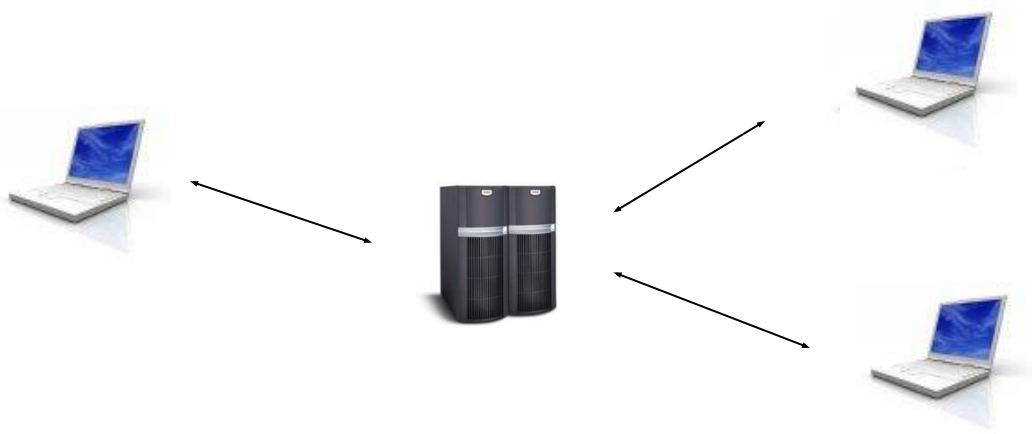
$ bzr revert Code.java

Tell bazaar that you've manually fixed a failed merge

$ bzr resolve Code.java (then do bzr commit afterwards)

Summary

- Revision control systems enable multiple developers to work on the same code base
- Bazaar uses a client/server system with a repository and working copies
- Every commit generates a new revision, which can be checked out independently
- Projects have a main branch, but can have multiple branches



Resources

https://help.launchpad.net/Code

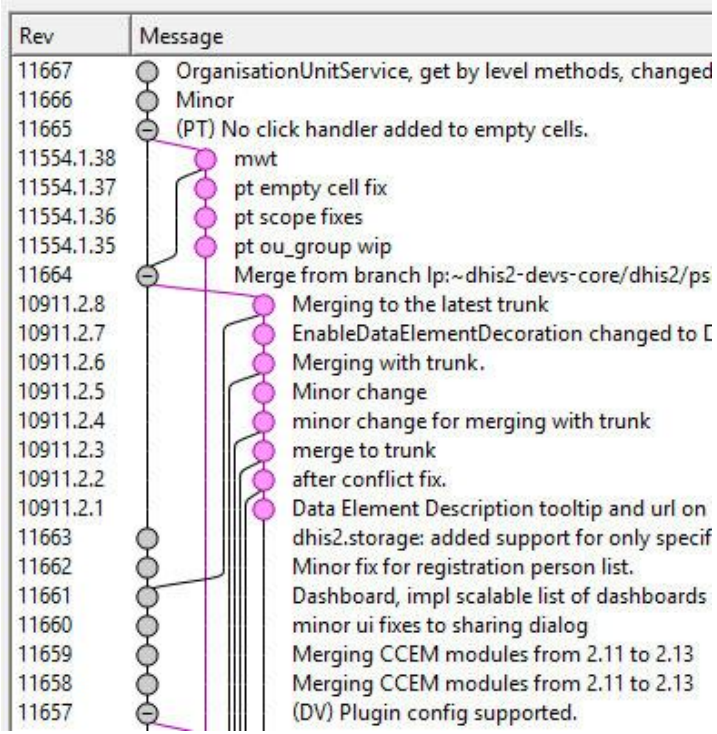
http://doc.bazaar.canonical.com/latest/en/tutorials/using_bazaar_with_launchpad.html

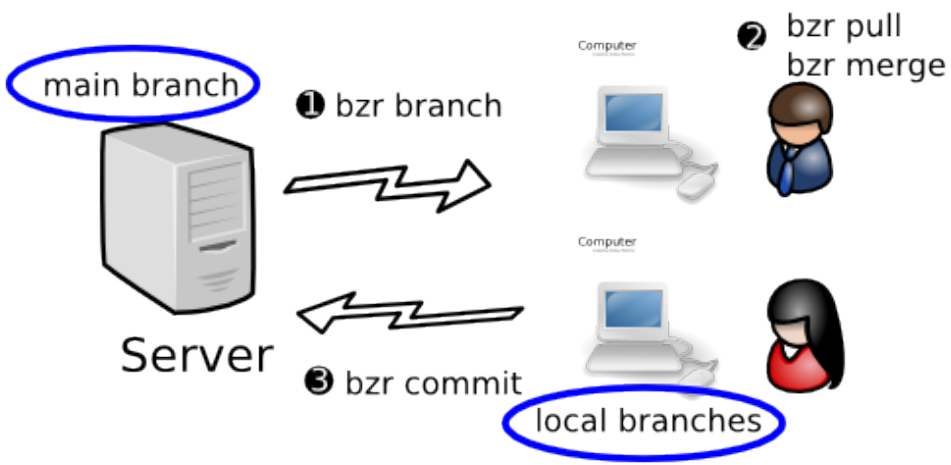
http://doc.bazaar.canonical.com/bzr.2.6/en/tutorials/tutorial.html

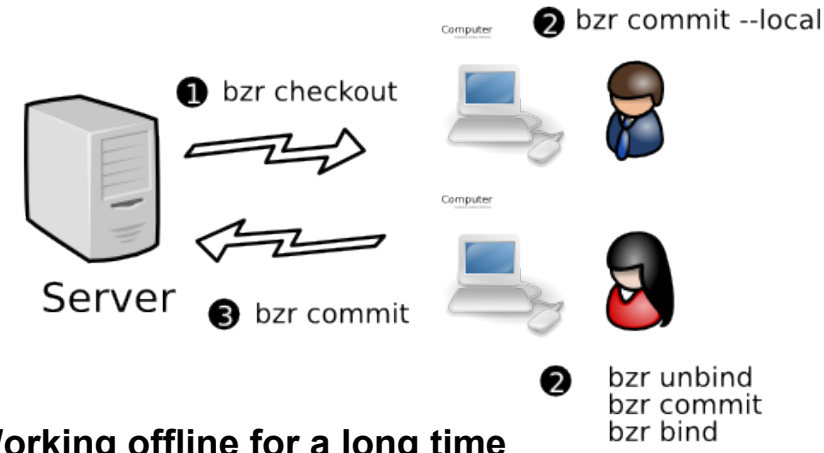
http://doc.bazaar.canonical.com/latest/en/_static/en/bzr-en-quick-reference.

Work cycle - Distributed way

- Several ways to work decentralized
- Create a local branch - “bzr branch …”
- Work locally
- Then pull central changes into your own repository
- Then merge and push your own repository to the cent
- or
- Check out code
- Do local commits using ‘bzr commit --local’
- Run update to get new changes
- Commit into central repository







Working offline for a long time