An Introduction to Mercurial Version Control Software

CS595, IIT
[Doc Updated by H. Zhang] Oct, 2010

Satish Balay
balay@mcs.anl.gov

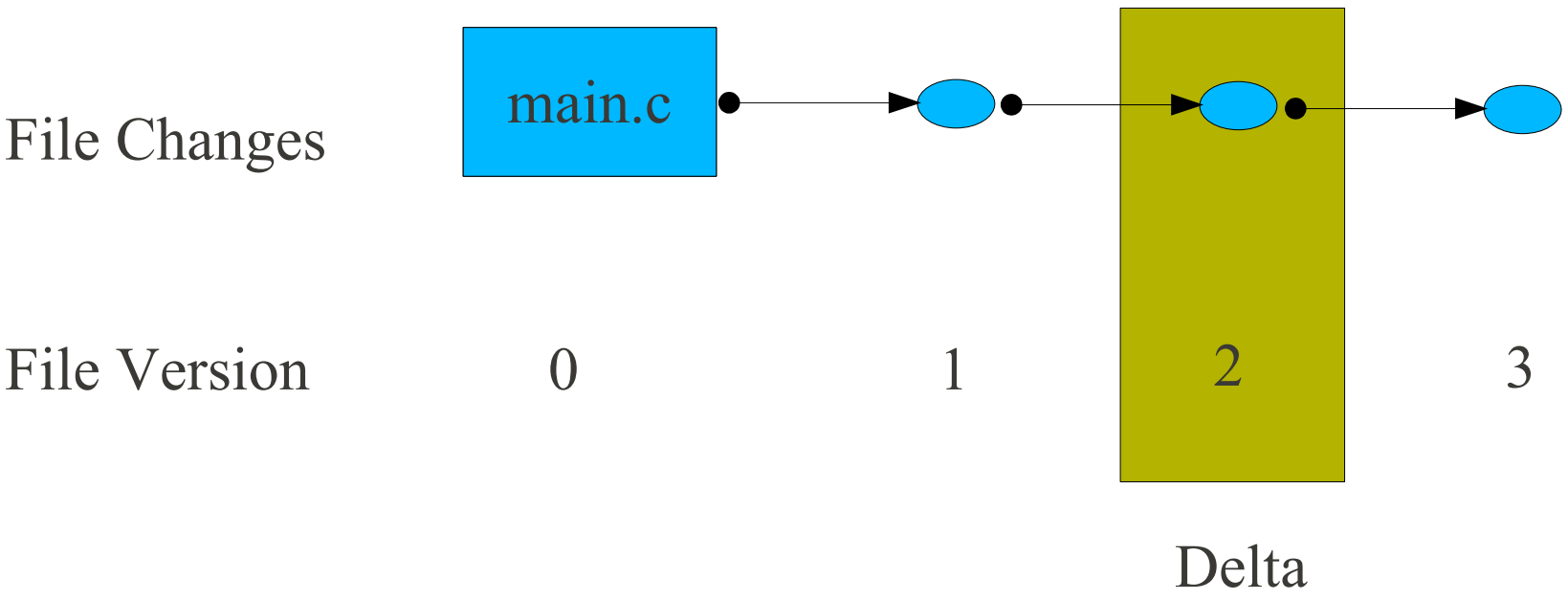
Outline

- Why use version control?
- Simple example of revisioning
- Mercurial introduction
  - Local usage
  - Remote usage
  - Normal user workflow
  - Organizing repositories [clones]
- Further Information
- [Demo]

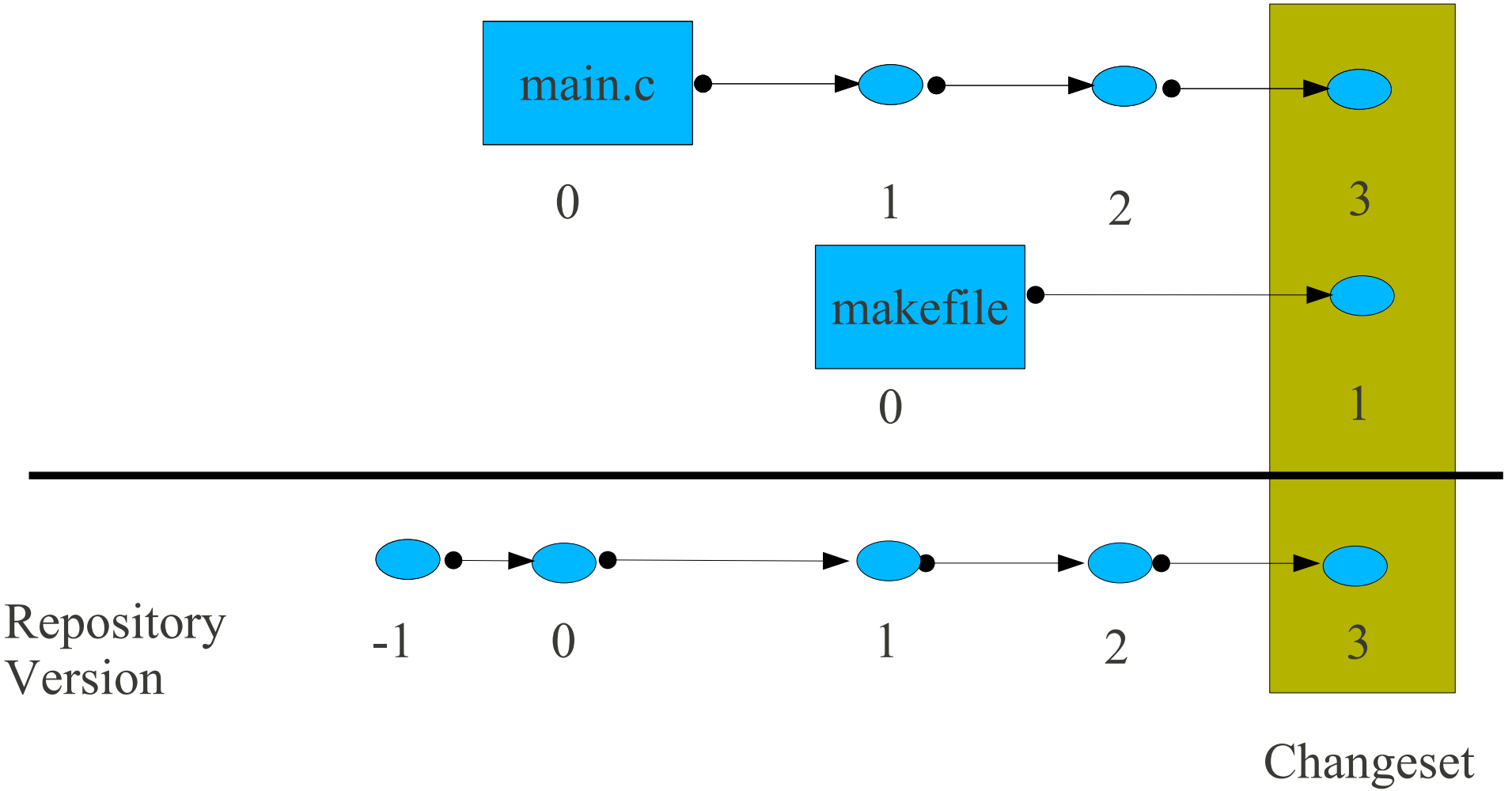
What do we use Version Control for?

- Keep track of changes to files
- Enable multiple users editing files simultaneously
- Go back and check old changes:
  * what was the change
  * when was the change made
  * who made the change
  * why was the change made
- Manage branches [release versions vs development]

Simple Example of Revisioning

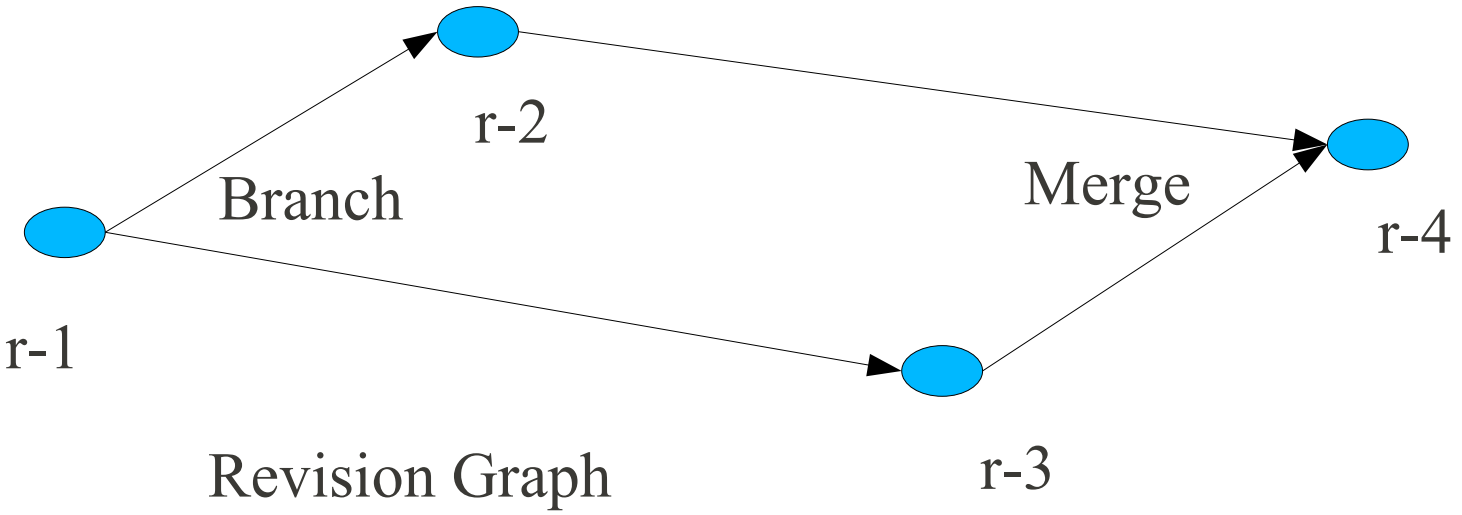


Simple Example Cont.



Concurrent Changes to a File by Multiple Users & Subsequent Merge of Changes

Initial file	UserA edit	UserB edit	Merge edits by both users
Line1	Line1	Line1	Line1
Line2	**UserA**	Line2	**UserA**
Line3	Line2	Line3	Line2
Line4	Line3	**UserB**	Line3
	Line4	Line4	**UserB**
			Line4



Revision Graph

Merge tools:
- kdiff3
- meld

Merge types:
- 2-way
- 3-way

Some Definitions

- Delta: a single change [to a file]
- Changeset: a collection of deltas [perhaps to multiple files] that are collectively tracked. This captures a snapshot of the current state of the files [as a revision]
- Branch: Concurrent development paths for the same sources
- Merge: Joining changes done in multiple branches into a single path.
- Repository: collection of files we intend to keep track of. This includes the revision graph/history [or metadata]
- Version [or Source] Control Tool: Enables us to keep track of all the changes [to all files] in a repository

Version Control Tools

- File Level Control
  - SCCS
  - RCS
- Centralized [or Client/Server]
  - CVS
  - Subversion
- Distributed
  - **Mercurial**
  - Git

http://en.wikipedia.org/wiki/List_of_revision_control_software

Mercurial

- Distributed version control tool.
- Open Source [GPL]
- Started by Matt Mackall in 2005, has many contributors
- Written in python
- Works on Linux, Windows, OS X and other systems.
- Reasonably efficient [handles 10,000+ changesets in PETSc]
- Active mailing list: mercurial@selenic.com
- http://www.selenic.com/mercurial

Usage: Creating a Repository

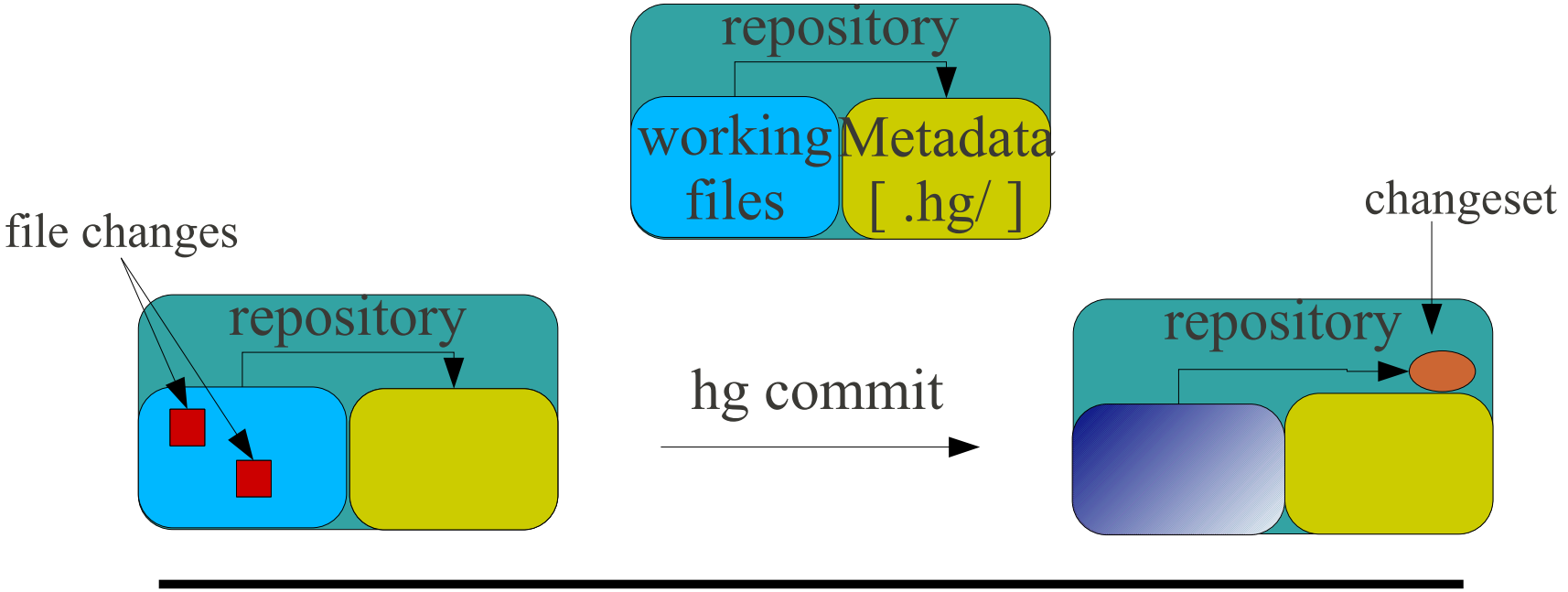
- *mkdir project*
- *cd project*
- ***hg init***
- Initializes the directory 'project' as a mercurial repo.
- It is currently an empty repository [i.e no files added]
- All 'hg' commands are invoked inside the repository
- All commands are in the form 'hg command'. For example : *hg help*
- Stores metadata in the subdirectory *project/.hg*

Usage: Adding/Modifying Files

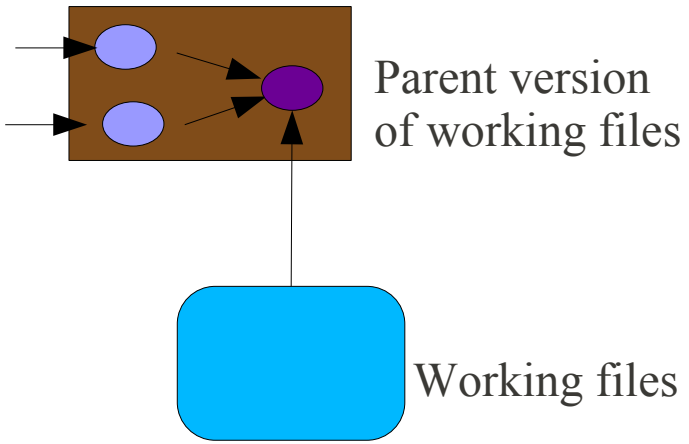
- *cd project*
- *touch main.c* [create or edit a file]
- ***hg add main.c***
- ***hg commit***
- *emacs main.c* [edits to file]
- ***hg commit*** [recommend alternative: *hg qct*]

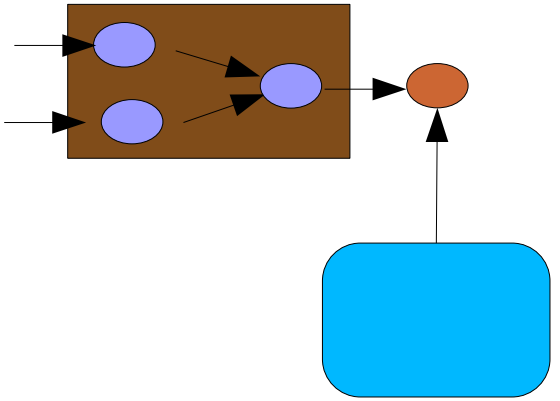
- 'add' indicates the file is now part of the repository.
- 'commit' creates a changeset for the current changes. [prompts the user to enter comments]
- Note: use ***hg commit -A*** to add/commit all new files

Illustration of Changes



Revision Graph View [Metadata]





Repository Data vs Working Files

- Repository data is the revision history and graph of all the changes. Its stored in project/.hg directory
- Working files are the reset of the files in project. User edits the working files.
- *hg tip* [show the tip revision of the repository graph]
- *hg parent* [show the parent revision of the working dir]

Note: Working files can correspond to any revision of the repository. So one has to be careful about this point [and not assume the parent is always the tip revision]

- *hg update REV* [update working copy to REV version]

Checking Status/History

- hg status [list the current modified, unknown files]
- hg diff [list the diff of the changed files in patch form]
- hg log [list the revision history of all changes]

- hg view [extension: GUI tool to check changeset graph]
- hg qct [external: GUI tool to check and commit changes]

Note: So far we have covered local operations on the repository

Distributed Model

- Information flows between repositories as changesets.
- Each operation is between two repositories [metadata].
- ***hg clone /home/balay/repoA repoB***
- *cd repoB* [Local repository to invoke commands]
- ***hg pull*** *[repoA]* [get remote changesets and apply locally]
- ***hg push*** *[repoA]* [apply local changesets to the remote repo]

Notes:

- Every clone repository has complete revision history [metadata].
- Peer to peer: all copies of repositories are equivalent.
- One can switch roles of repoA & repoB.
- Remote operations are between repositories [as opposed to local operations – discussed in the previous slides]

URLs/ Communication Mechanism

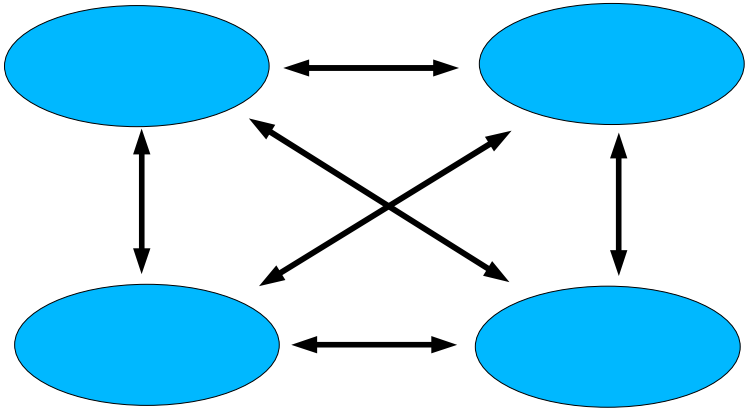
hg help pull [documentation of urls]

- /home/balay/petsc-dev
- ssh://petsc@petsc.cs.iit.edu//hg/petsc/petsc-dev
- http://petsc.cs.iit.edu/petsc/petsc-dev [readonly]
- http-old://www.mcs.anl.gov/~petsc/project [readonly]
- https:// [read/write support]

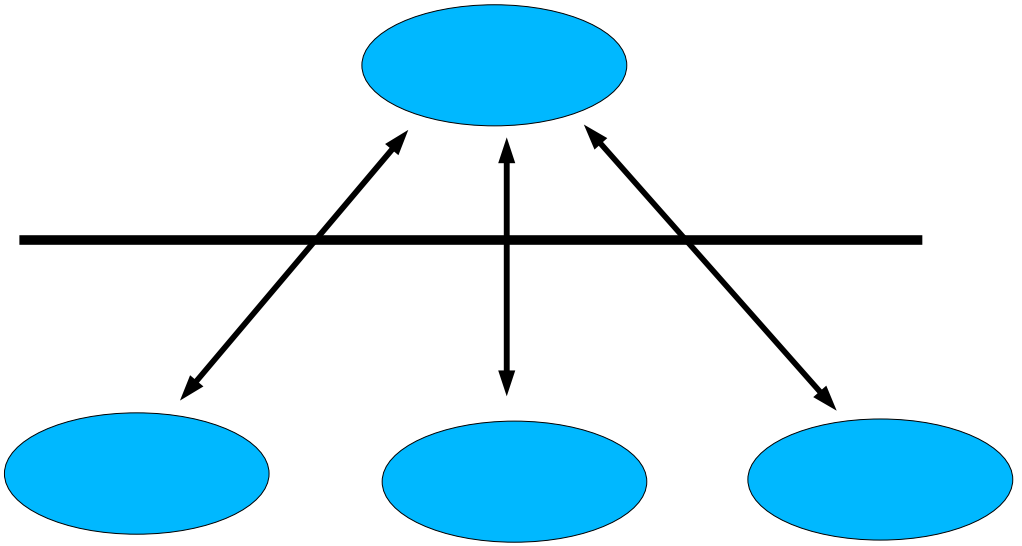
Notes:

- 'hg clone' stores the URL for remote repository [in .hg/hgrc]. When push/pull operations are invoked, default URL is used.
- Require [remote] read access for pull, write access for push.
- Email changesets via 'hg bundle' [if necessary].

Organizing Repositories [clones]



Any to Any

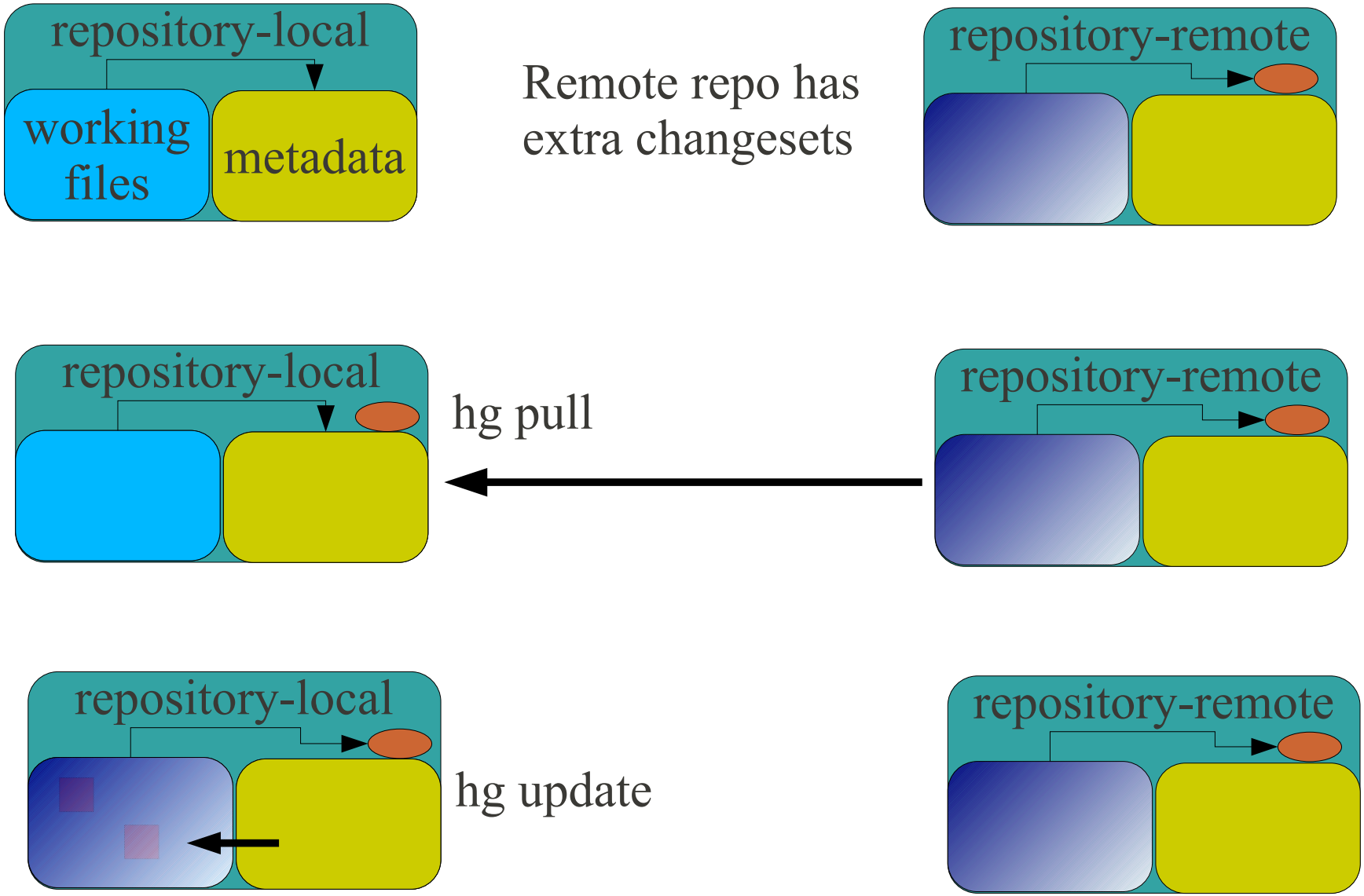


Shared Common

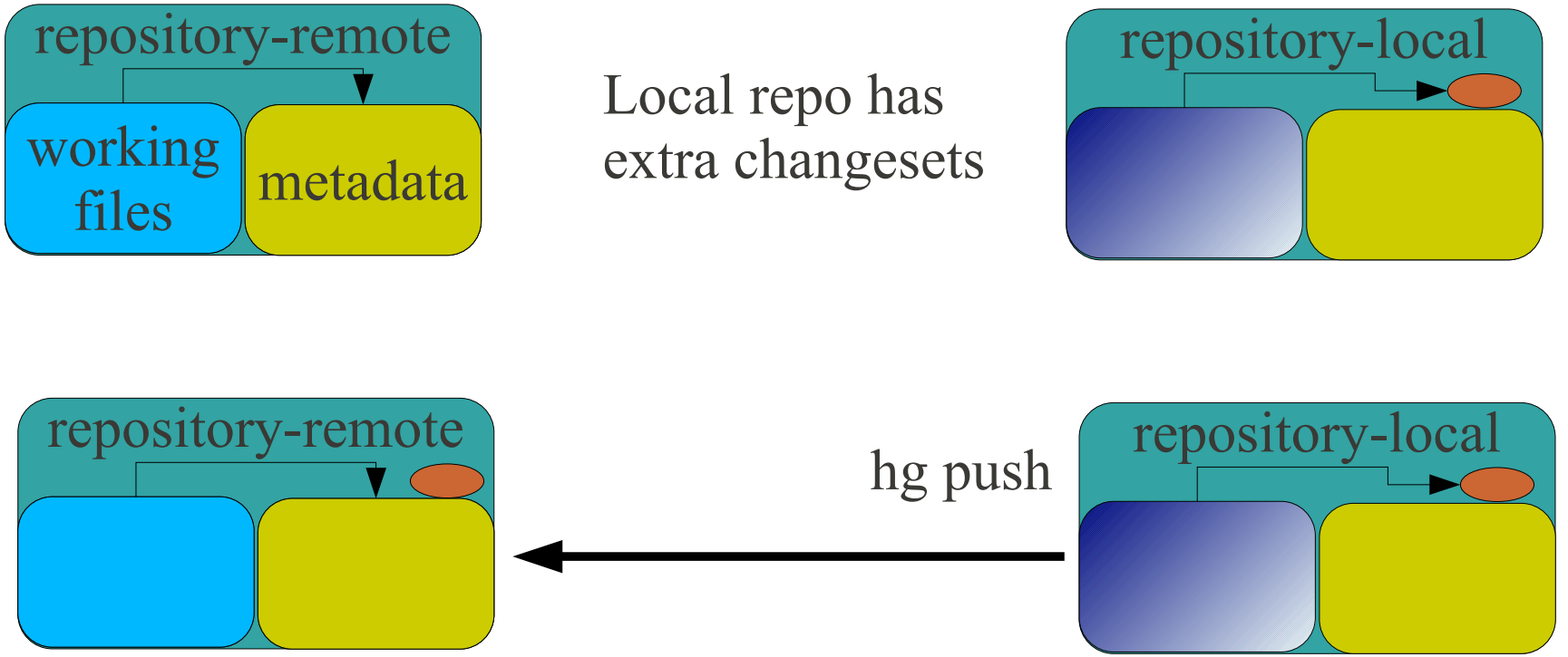
Methods of communicating changes
•clone/push/pull [changesets]
•import/export [email patch]
•bundle/unbundle [email changesets]

The relations are not hardcoded

Syncing Two Repositories with Changesets to Remote Repository

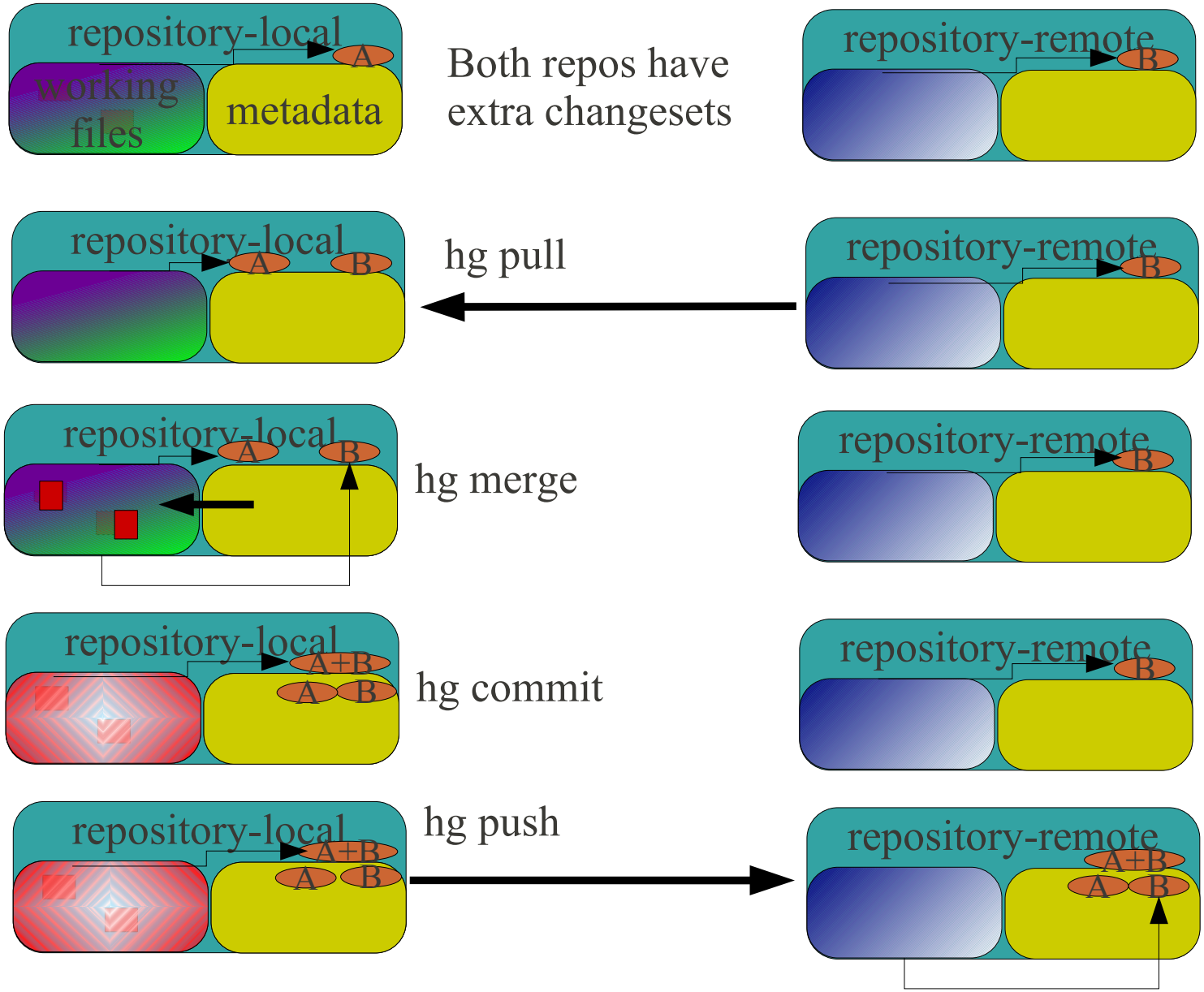


Syncing Two Repositories with Changesets to Local Repository

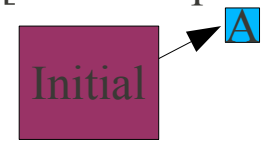


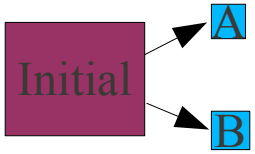
Updating Working copy of remote is not done.

Syncing Two Repositories with Changesets to both Repositories

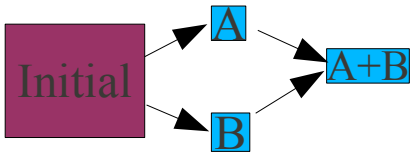


Revision Graph Change [Local Repository]





Normally working files have 1 parent, but in the merge process, they have 2 parents.



A: local repo changeset
B: remote repo changeset
A+B: merge changeset

Normal User Work Flow: [Making local changes and updating Shared Common Repository]

- <make changes to working files>
- *hg commit* [commit local changes]
- *hg pull* [check & obtain remote changes]
- *hg merge* [if remote changes exist, auto merge. Or a manual merge via external tool like kdiff3]
- *hg commit* [commit the merge changeset]
- *hg push* [push local changesets + merge changesets to the remote repository]

Note: Merge is always done in the local repository

[hence the order: pull, merge, push]

Qct: Graphical Commit Tool

Helps reviewing changes before commiting.

- *hg qct*

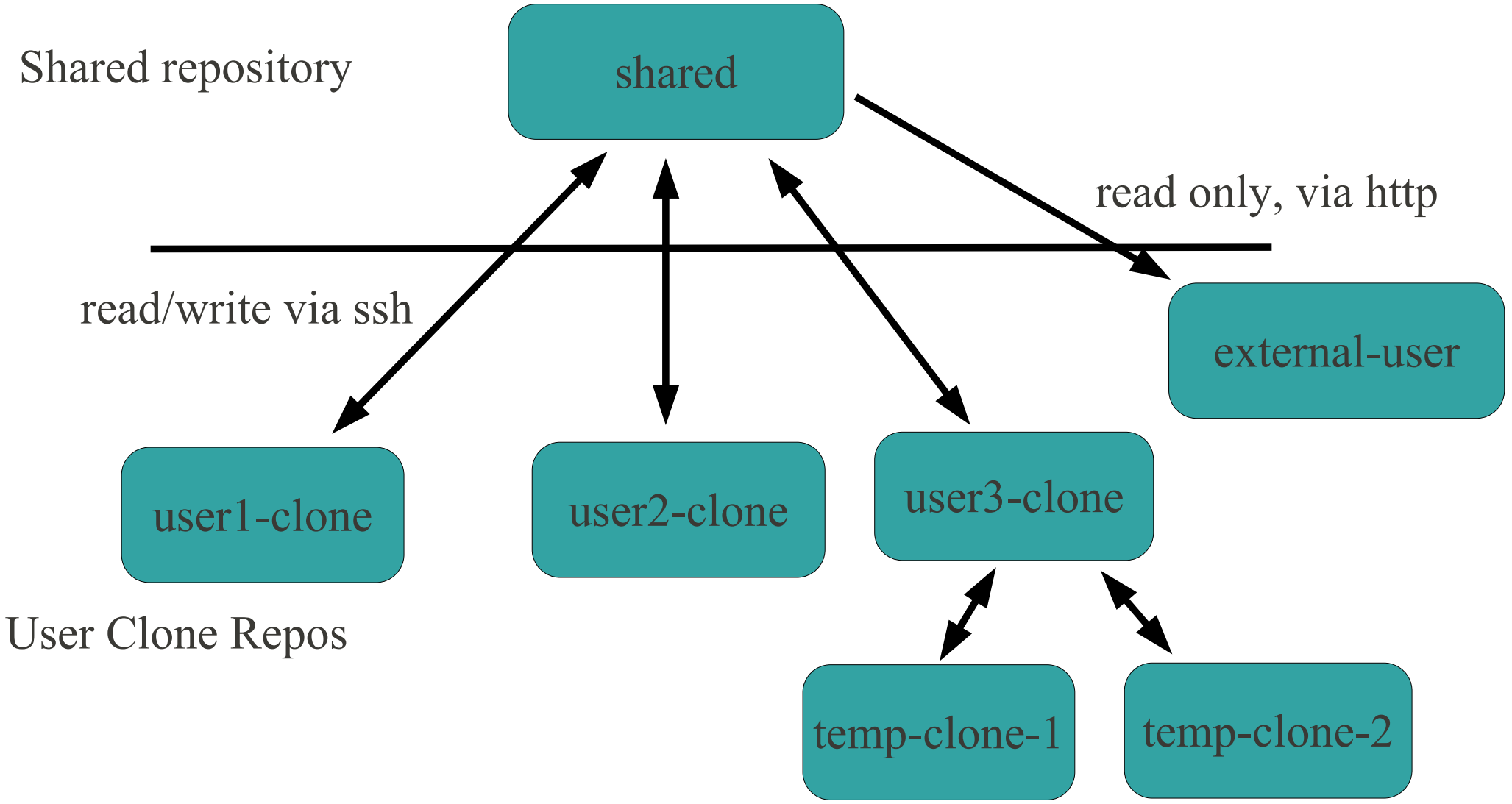
Equivalent to the following normal work flow

- *hg status*
- *hg diff*
- *hg commit*

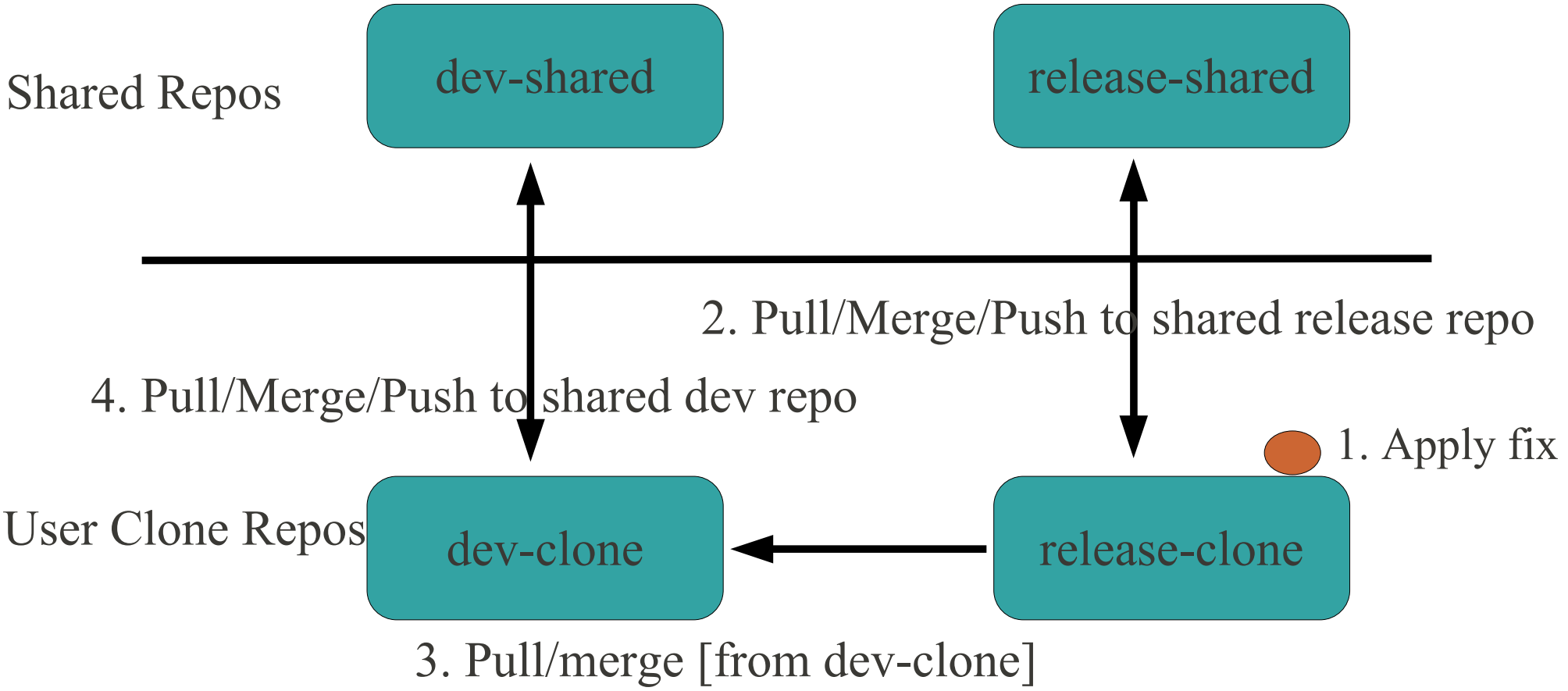
External tool from http://qct.sourceforge.net/

It requires PyQt4.

Multiple Users: Communicating Changesets using a Shared Repository



Managing Patches to Release Versions



Browsing changes

- *hg view*
- *hg log*
- *hg annotate filename [REV]*
- *hg update [REV]*
- *hg serv* [starts a web server]
- Use a web browser to browse changes

Mercurial at ada.cs.iit.edu

- mercurial 1.6.2 is installed on the linux machine

Further Information

- http://www.selenic.com/mercurial/
- http://hgbook.red-bean.com/hgbook.html
- mercurial@selenic.com