[Software Development]

Development Tools

Davide Balzarotti

Eurecom – Sophia Antipolis, France

Version Control

- Version (revision) control is the process of tracking and recording changes to files
- Most commonly used when a team of people have to simultaneously work on the same documents
- Very useful also for personal development
  - Share configuration files between different locations/computers
  - Personal repository for code and papers
  - History of different versions of the same artifacts
  - Freedom to try many changes and revert to a clean state

Revision Control

- Multi-user Scenario:
  - Alice and Bob are working on a common project
  - Alice changes `file_A`
  - Bob changes `file_A` (at the same time)
  - Problems:
    - How Alice and Bob propagate the changes to each others?
    - How can they merge the changes to the same file?

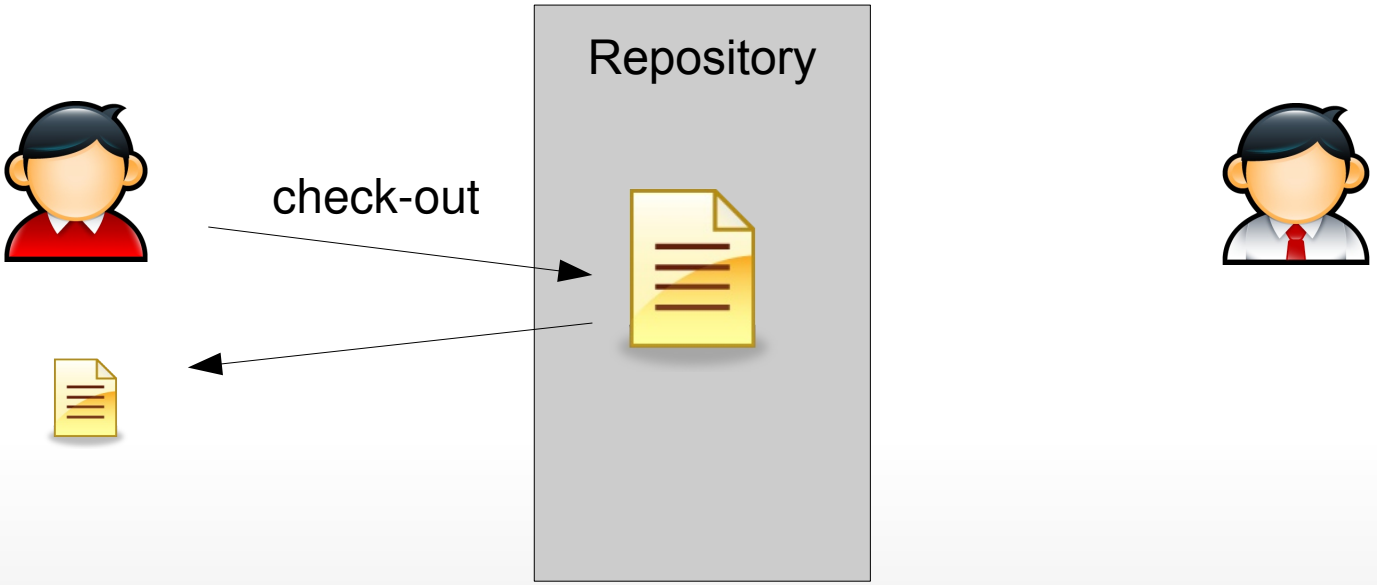
Revision Control

- Multi-user Scenario:
  - Alice and Bob are working on a common project
  - Alice changes `file_A`
  - Bob changes `file_A` (at the same time)
  - Problems:
    - How Alice and Bob propagate the changes to each others?
    - How can they merge the changes to the same file?
- Single-user Scenario:
  - Charlie's program is working just fine
  - Charlie makes a lot of changes overnight
  - Now the program presents some weird behavior
  - Problem:
    - How does he keep track of the previous versions of his files?

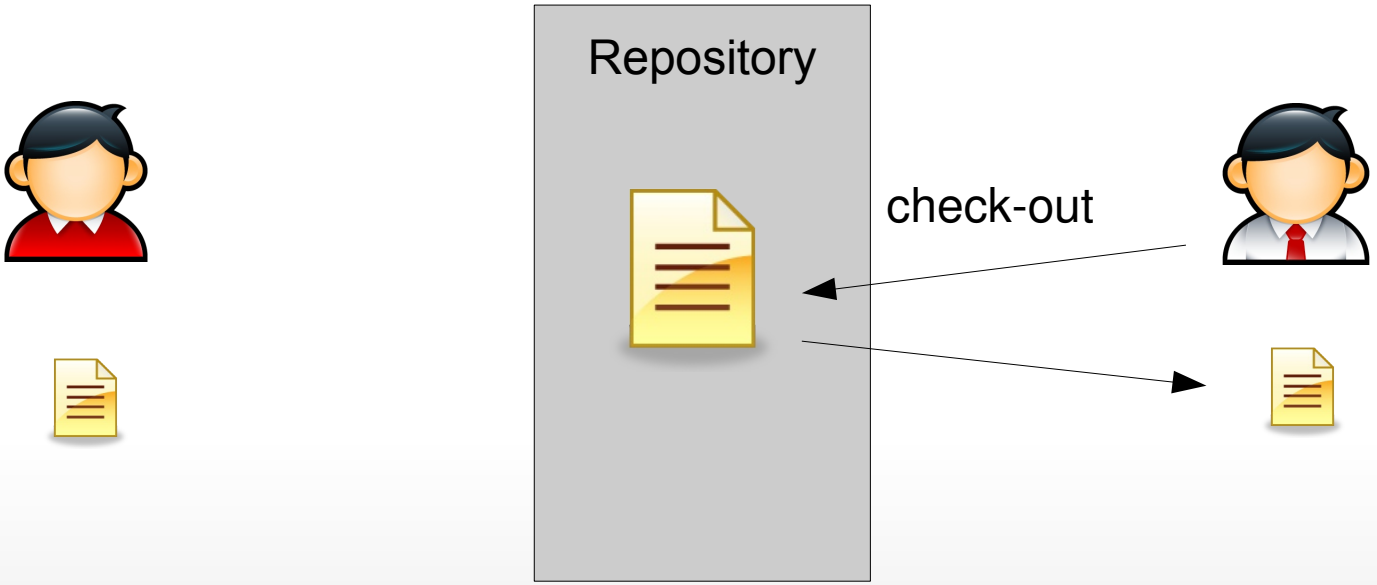
Revision Control System

- Tool that keeps in a repository multiple version of documents
  - Allows “check out” of files into the user filesystem
  - Allows “check in” of new versions of files into the repository
  - Keeps an history of all the changes
  - Often supports multiple project branches
  - Provides a point of coordination between multiple users
- Two main approaches for collaborative work
  - Lock-Modify-Unlock (no parallel work on the same document)
  - Copy-Modify-Merge (parallel work allowed)

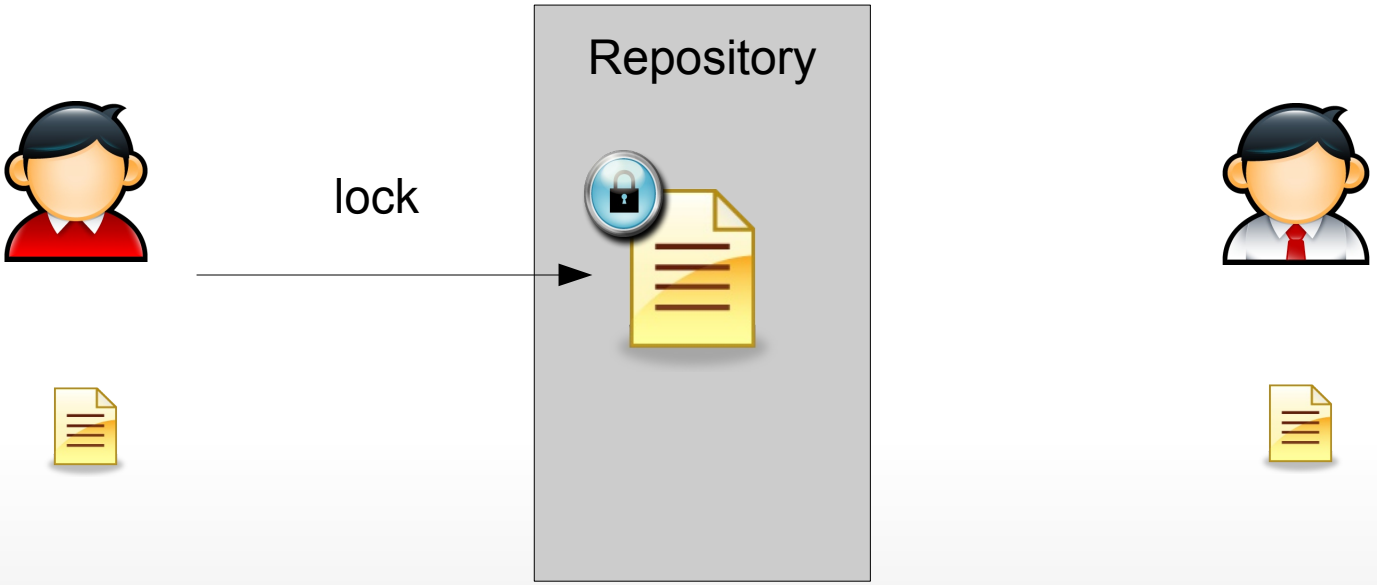
Lock-Modify-Unlock



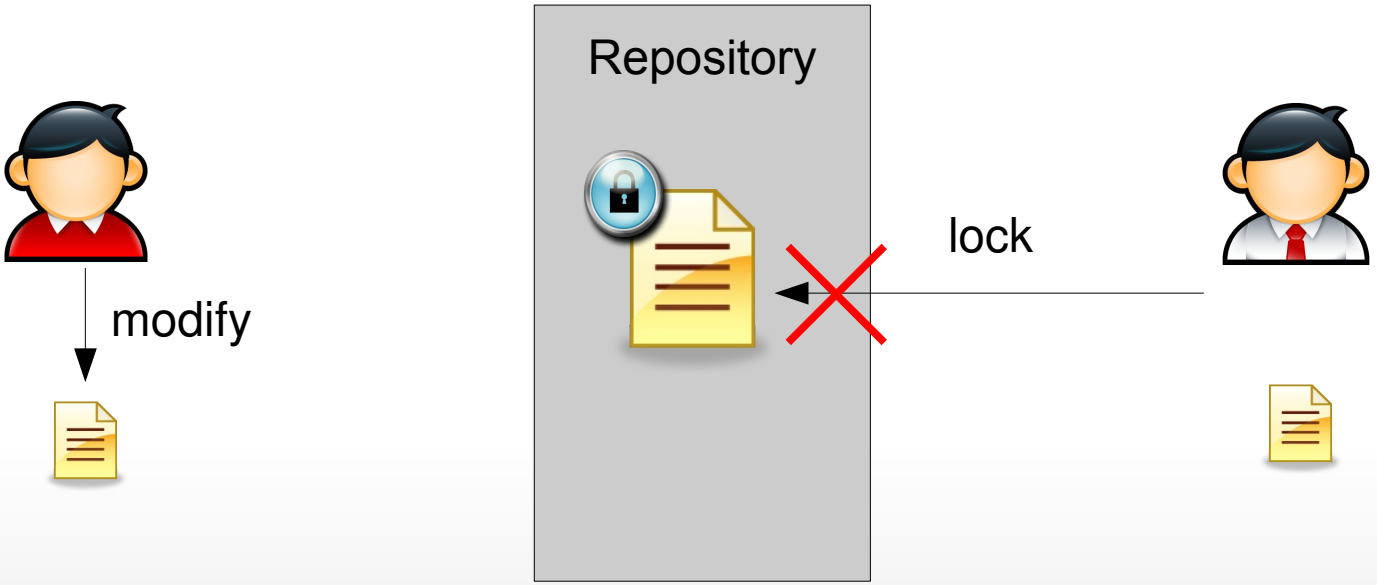
Lock-Modify-Unlock



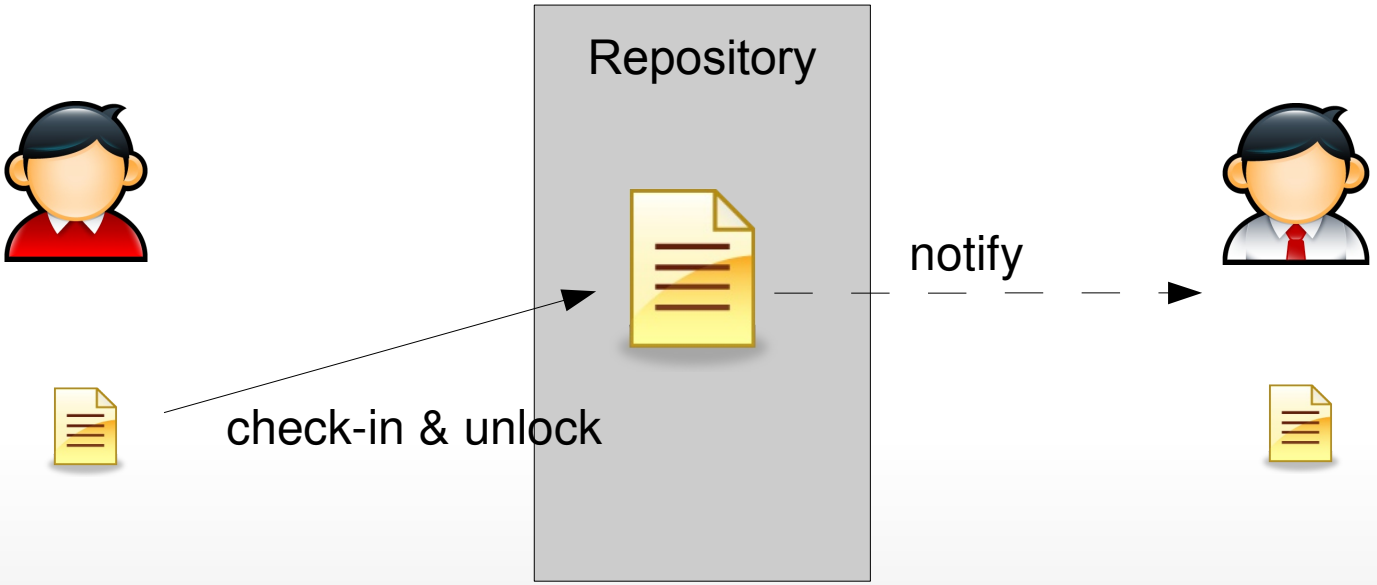
Lock-Modify-Unlock



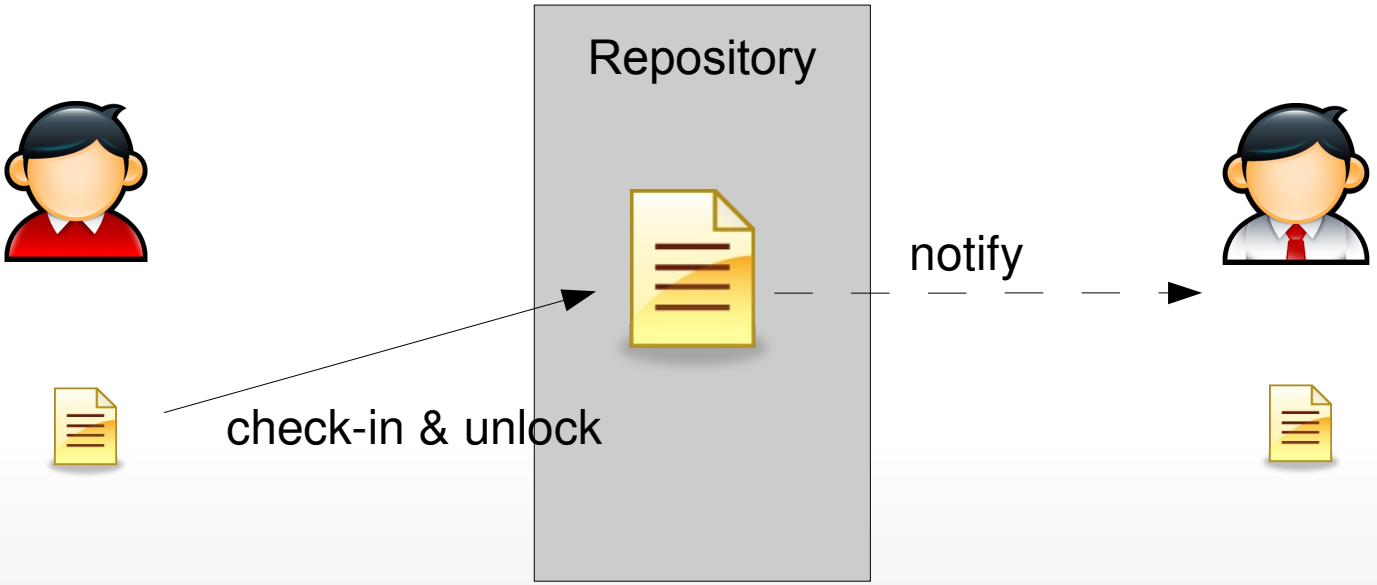
Lock-Modify-Unlock



Lock-Modify-Unlock

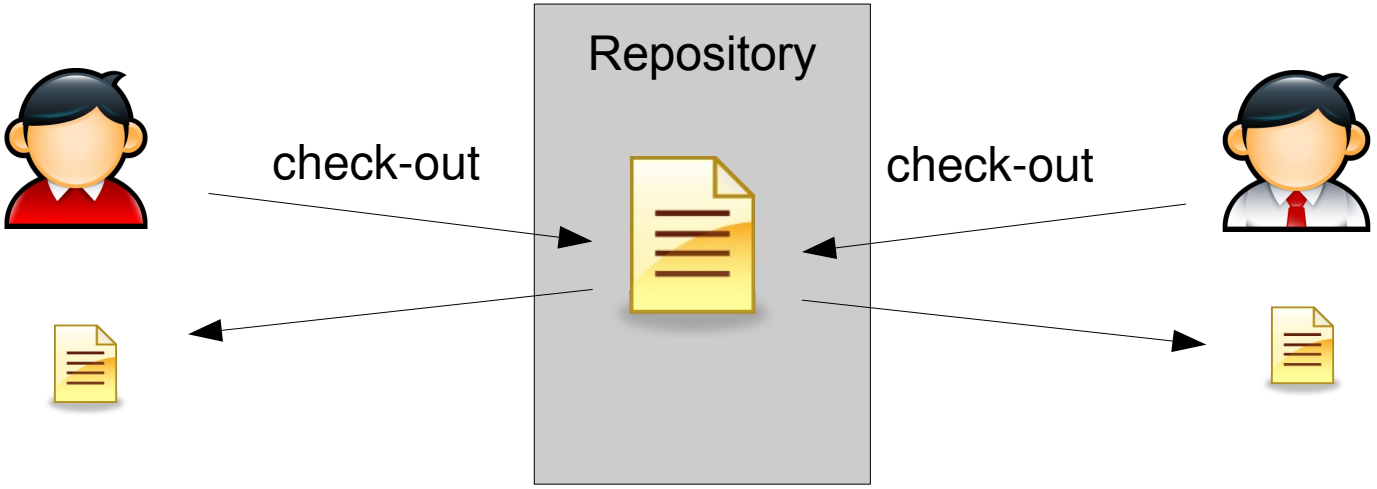


Lock-Modify-Unlock



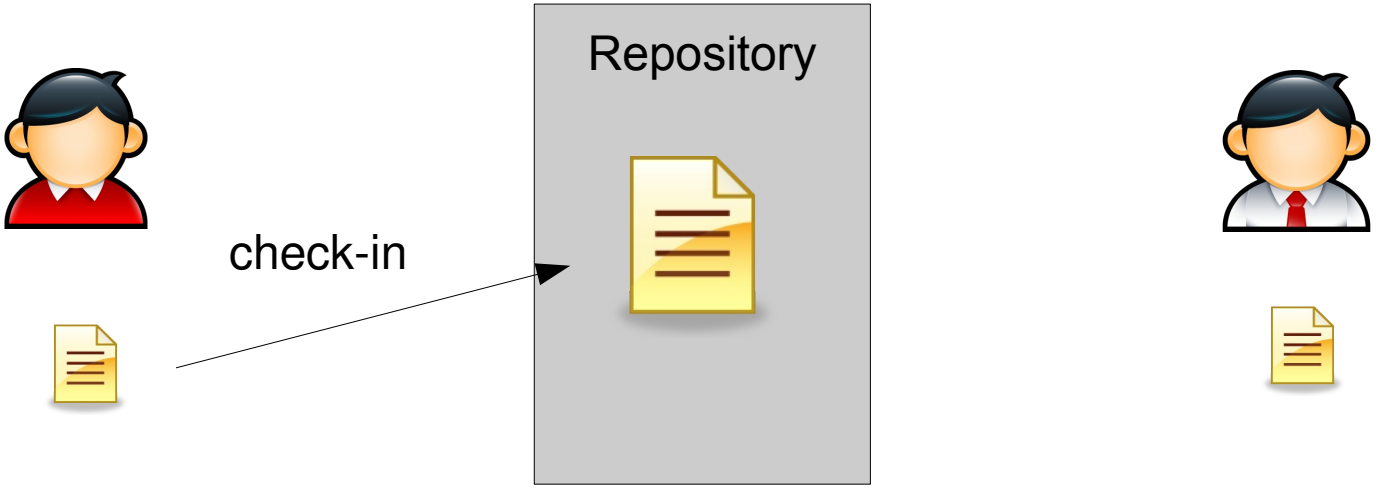
- Advantages:
  - No conflict resolution needed
- Disadvantages:
  - Sequential development
  - Scalability problem when one has to work on many files at the same time

Copy-Modify-Merge

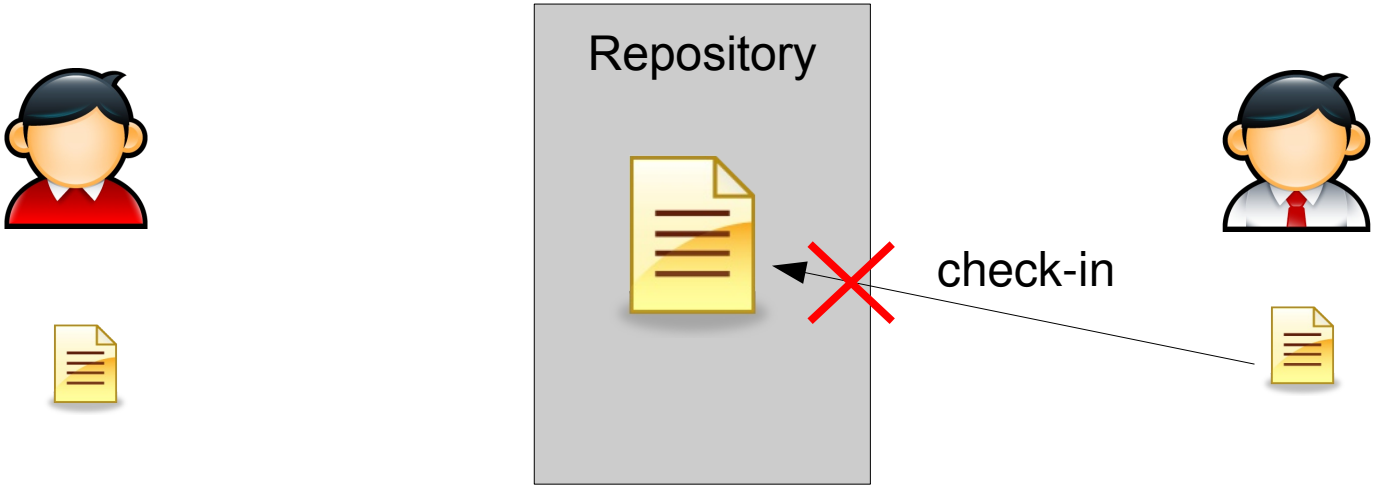


Copy-Modify-Merge

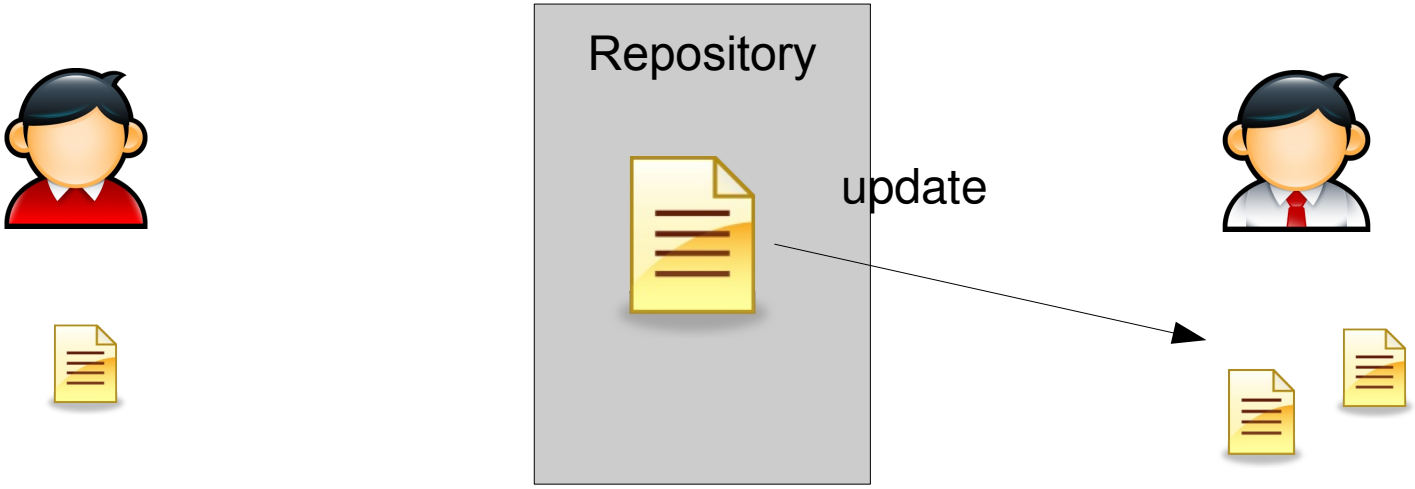
Copy-Modify-Merge



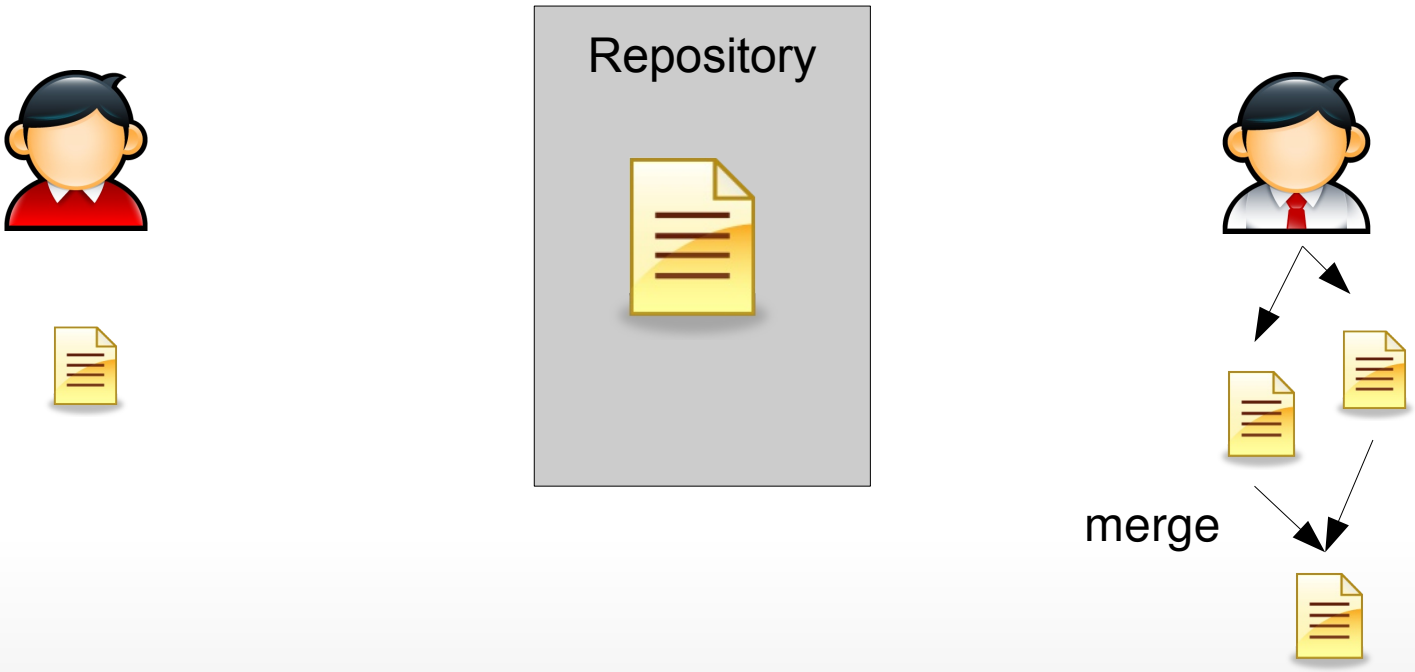
Copy-Modify-Merge



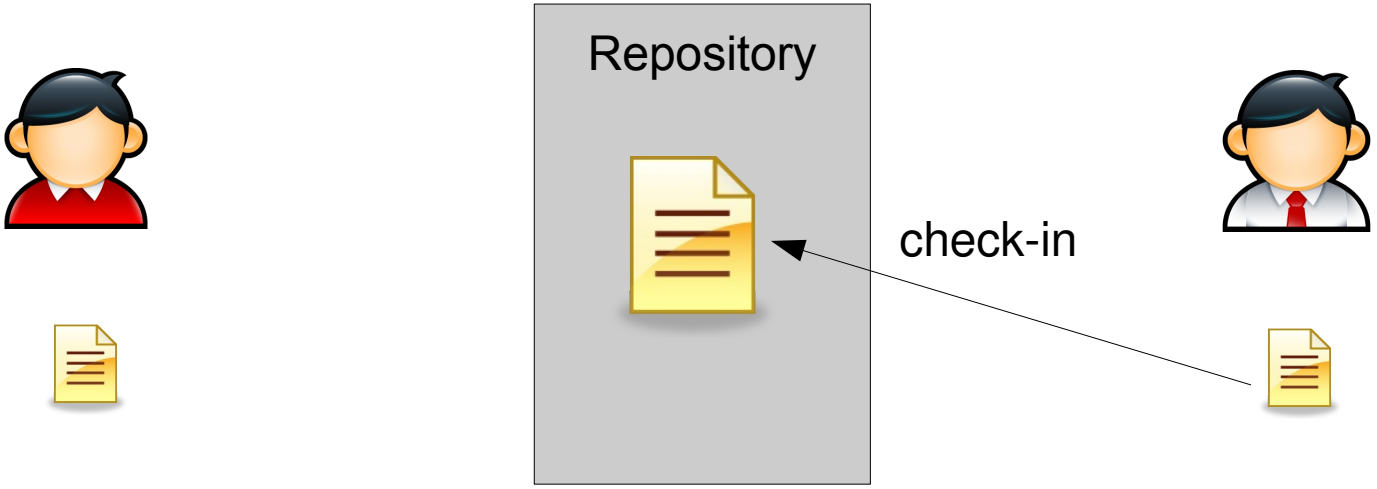
Copy-Modify-Merge



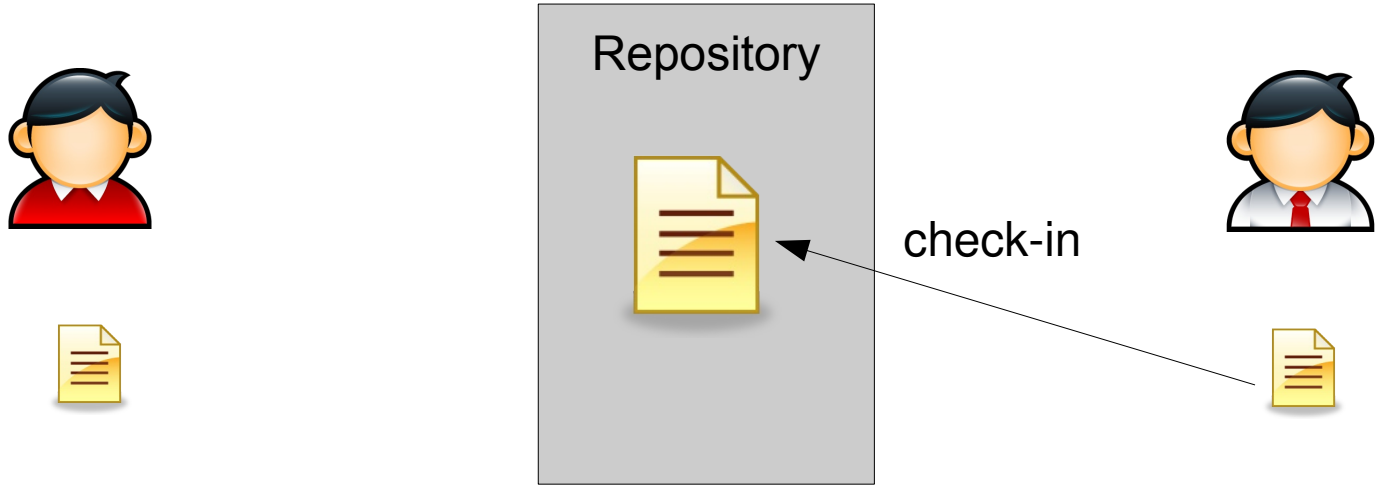
Copy-Modify-Merge



Copy-Modify-Merge



Copy-Modify-Merge

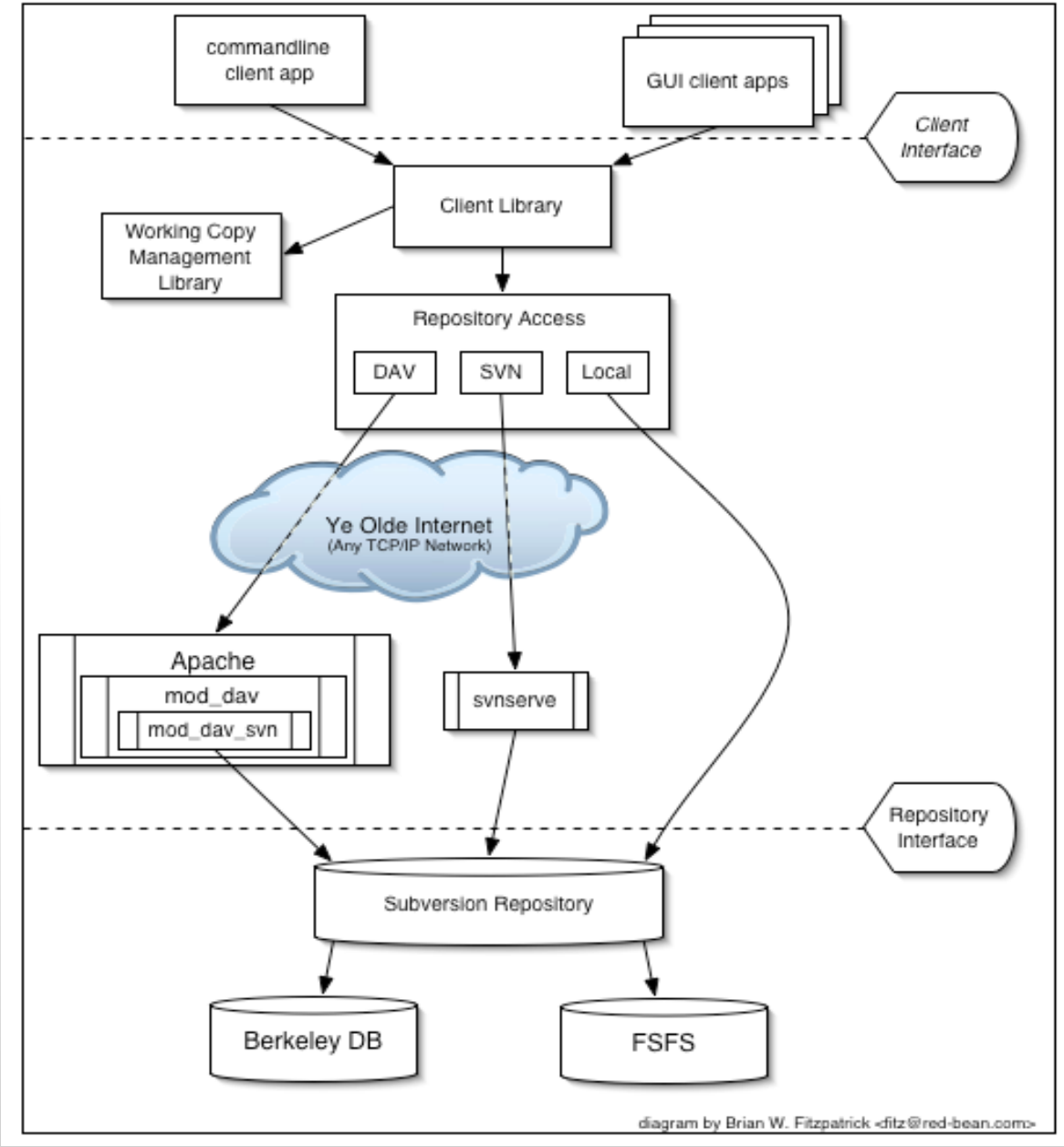


- Advantages:
  - Concurrent development
  - Simple conflicts can be resolved automatically
- Disadvantages:
  - complex conflict resolutions can be hard

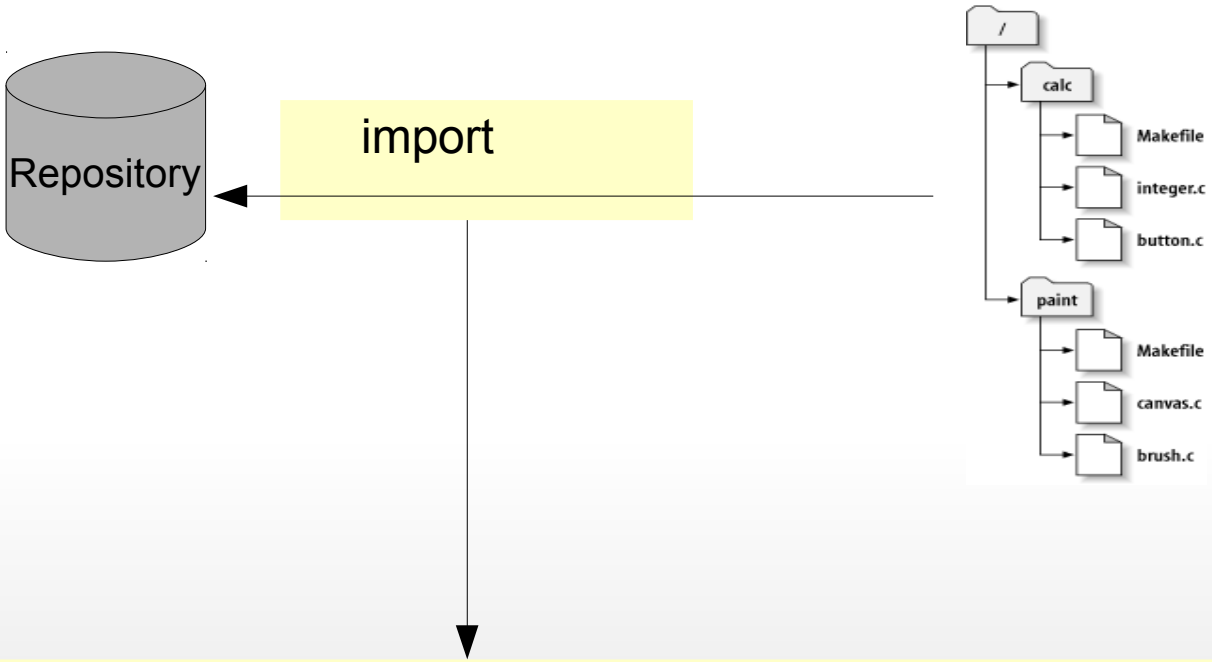
Subversion

- Open source project released in 2001 to replace CVS
  (the tool of choice for version control for many years)
- Can be used to manage any collection of files
  (does not matter if they contain source code or the pictures of your vacation)
- At the core of subversion there is a centralized repository accessible through a traditional client-server architecture
- Subversion adopts a copy-modify-merge approach
  (but it also provides basic locking mechanisms if needed)

The Global Picture



Import a new Project into the Repository

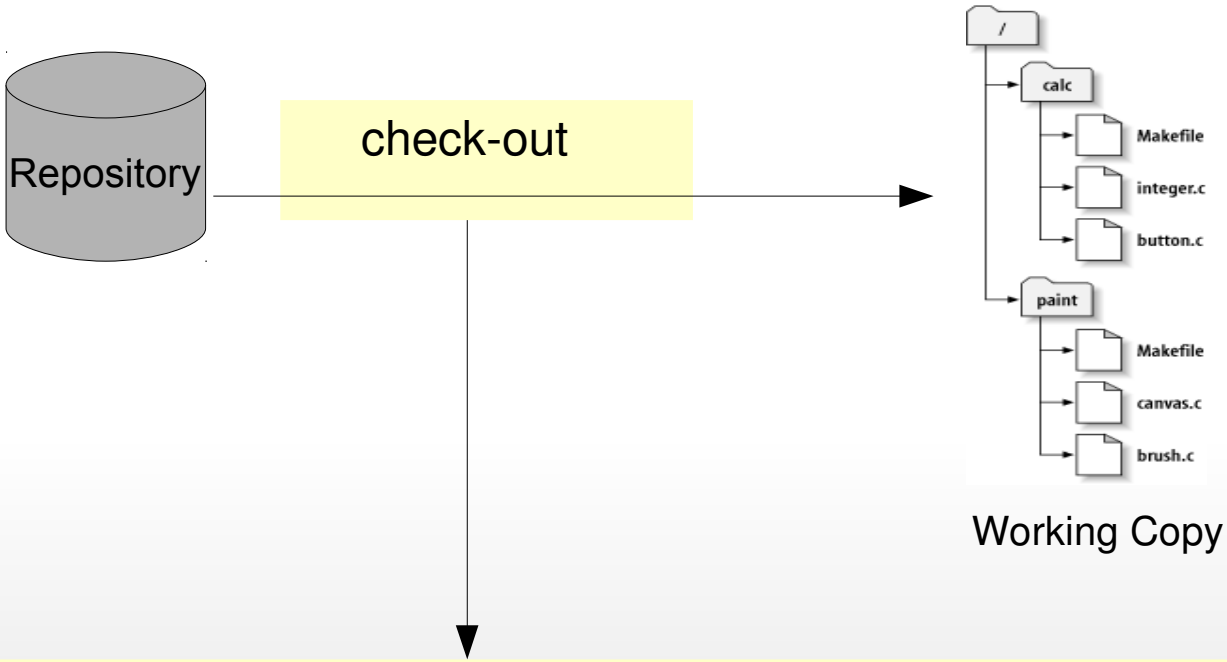



```
> svn inport myproject/  http://svn.my-server.com/repos/project1

file:///        Direct repository access (on local disk)
http://         Access via WebDAV protocol to Subversion-aware Apache server
svn://          Access via custom protocol to an svnserve server
svn+ssh://  Same as svn://, but through an SSH tunnel

NOTE: importing a project does not create a working copy
```

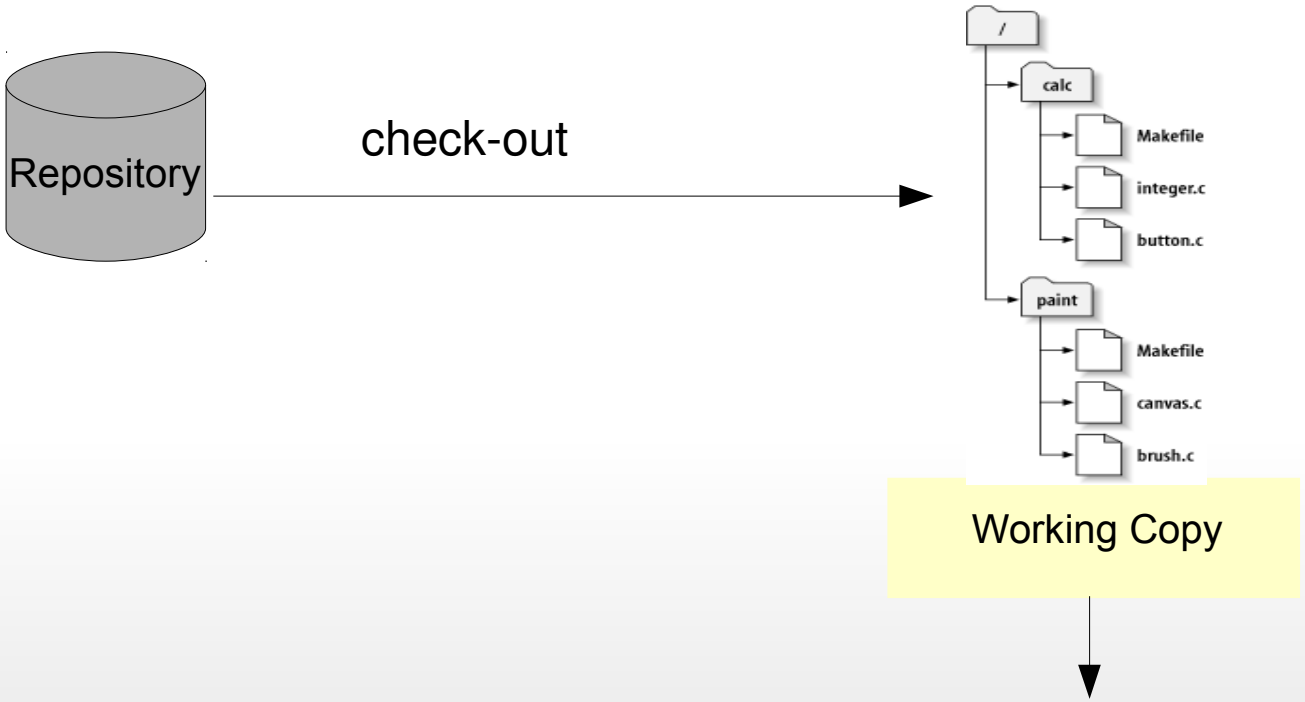
Check-out a Project




```
> svn checkout http://svn.my-server.com/repos/project1

A         project1/file_one.c
A         project1/file_two.c
…
Checked out revision 1
```

Check-out a Project



A working copy usually corresponds to a particular subtree of the repository

Each directory in the working copy contains a subdirectory named `.svn`, also known as the working copy's administrative directory

Information about a Working Copy

- `svn info` provides general info about the current directory (if it is a subversion working copy)

```
balzarot > svn info
Path: .
URL: file:///var/svn/repository/test_project
Repository Root: file:///var/svn/repository
Repository UUID: cb12b695-9261-4517-b769-b34f9168e06e
Revision: 9
Node Kind: directory
Schedule: normal
Last Changed Author: balzarot
Last Changed Rev: 9
Last Changed Date: 2009-12-08 22:55:32 +0100 (Tue, 08 Dec 2009)
```

Modify the Working Copy

- In your working copy you can edit files at will, but you must tell Subversion about everything else that you do
- Any action on the working copy is scheduled to be propagated to the repository at the next commit
  - `svn add foo`

    Schedule file (or directory) *foo* to be added to the repository
  - `svn delete foo`

    Schedule *foo* to be deleted from the repository
  - `svn copy foo bar`

    Copy *foo* to *bar* and schedule bar to be added to the repository
  - `svn move foo bar`

    Schedule *bar* for addition and *foo* for removal
  - `svn mkdir foo`

    Same as mkdir foo; svn add foo

Examine your Changes

- `svn status`

  Overview all file and tree changes you've made on your working copy

```
?   scratch.c             # file is not under version control
A   stuff/loot/bloo.h     # file is scheduled for addition
C   stuff/loot/lump.c     # file has conflicts
D   stuff/fish.c          # file is scheduled for deletion
M   bar.c                 # file has been locally modified
```

- `svn status` does not need to contact the repository (it uses the metadata in the `.svn` directory)
- `-u` – force it to contact the repository and notify of each file that is out of date

Examine your Changes

- `svn diff`

  With no arguments, it compares the files in the working copy against the cached “pristine” copies within the .svn area and prints out all changes in *unified diff* format

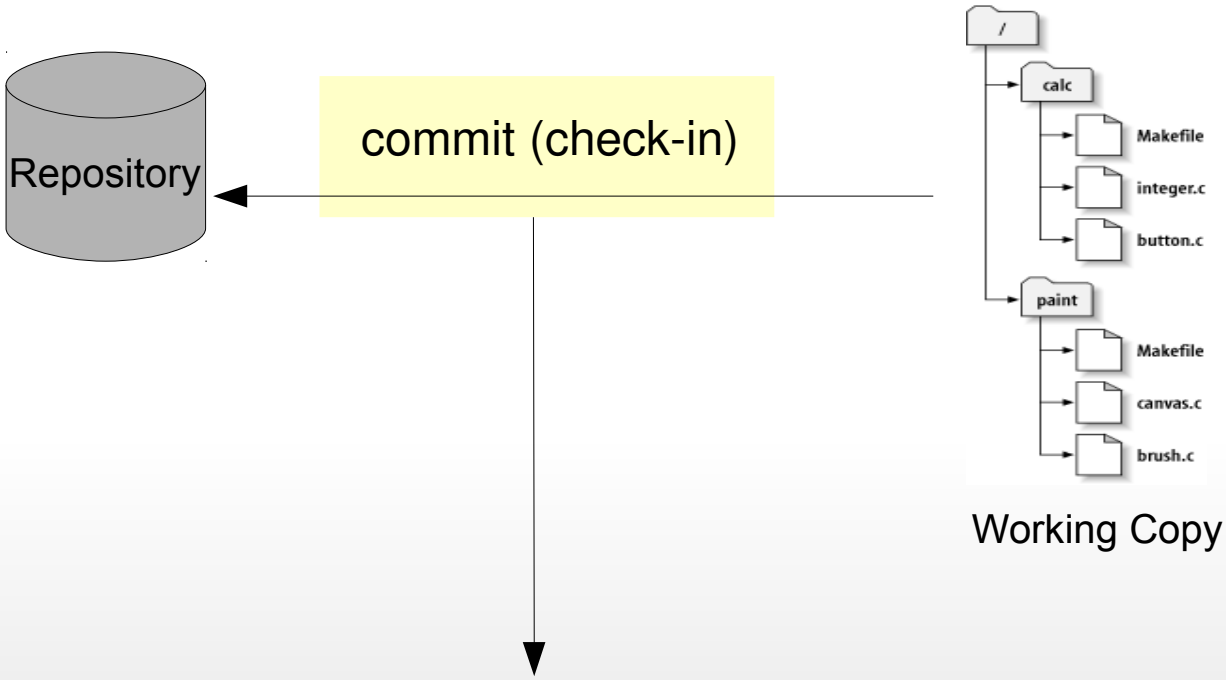
- `svn revert file`

  Reverts the file state to the pristine version from the .svn directory

- `svn blame file`

  Shows the content of the file, annotating each line with its author and last revision number

Committing Changes

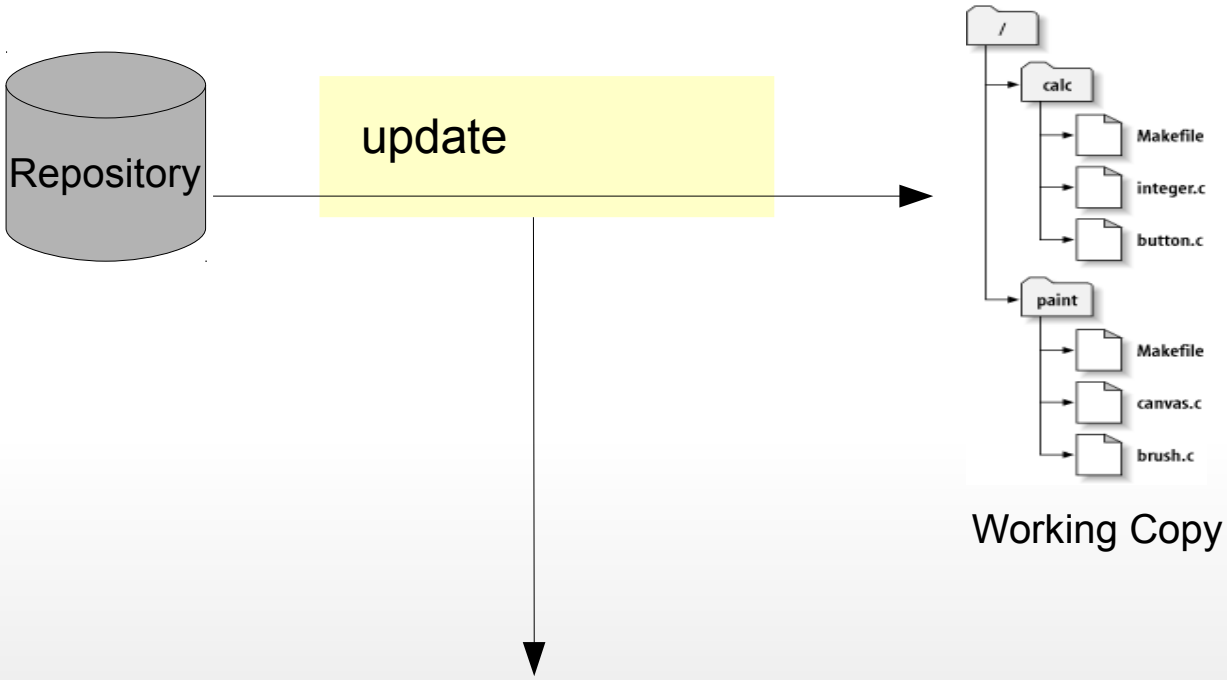



```
> svn commit
Sending        foo.c
Sending        bar.c
Transmitting file data .
Committed revision 2.
```

Atomic Commit

- If one of the file to be committed is not up to date (i.e., there is a newer version in the repository), the commit fails
  - svn commits are atomic: either a commit goes into the repository completely, or not at all
- Each time the repository accepts a commit, it creates a new revision. Each revision is assigned a unique number, one greater than the number of the previous revision
  - Revision N represents the state of the repository after the Nth commit
  - Since revisions do not apply to individual files, revisions N and M of a file (that it, the state of the file as it appears in revisions N and M) do not necessarily differ

Updating the Working Copy




```
> svn update
U   foo.c
G   README
C   bar.c
```

Resolving Conflicts

- **U** `filename` – the file has been updated to the new version available in the repository
- **G** `filename` – the file has been automatically merged with the new version available in the repository
- **C** `bar.c` – conflict!! The automatic merge failed, you have to solve the conflict yourself

- If there is a conflict, SVN won't let you check in until you explicitly say you have resolved it
  - To tell subversion that you fixed the conflict, use

    ```
    svn resolved file
    ```

Checking the History

```
svn log [-r versions] [filename]
```

- Prints a log of all changes for the file filename
  (or the entire project if no file is specified)
- `-r version1:version2` – restrict the log to the changes between version1 and version2
  - Version can be a version *number* or a *{date}*
  - -r 5:{2009-02-17} – between version 5 and the version of February 17th, 2009

Versions

- The -r option can be used in many subversion commands
  - `svn co -r 33 file://....`

    check out version 33 of a project
  - `svn diff -r 4`

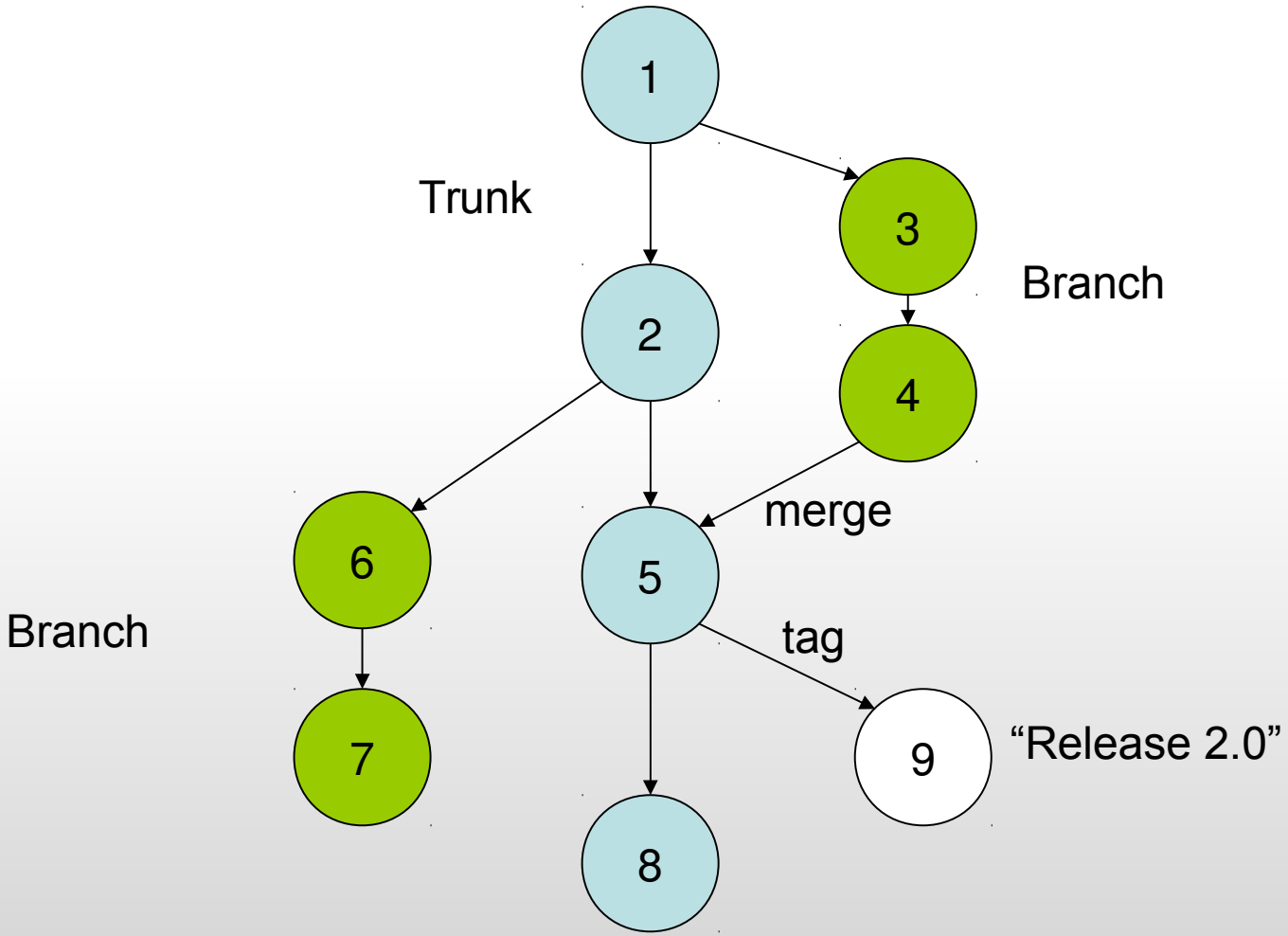
    print the diff between the working copy and version 4 of the repository
  - `svn diff -r 4:5`

    print the diff between version 4 and 5 of the repository (it does not even requires a working copy at all)

Branching and Tagging

- Trunk: main line of development of a project
- Branch: side line of development obtained by copying the Trunk or another Branch
  - When preparing for a release and the work on trunk needs to carry on
  - When a change is too disruptive and it will hinder other developers
- Tag: a labeled snapshot of the Trunk or of a Branch
- It is common to have three subdirs (branches, tags, trunk) to organize the data of a project
- `svn copy` – to create a copy for branching or tagging
- `svn merge` – to merge the changes from one branch to another

Trunk, Tags, and Branches



Versioned Metadata

- Each file and directory has a set of “properties” attached
  - You can invent and store any arbitrary key/value (text or binary) pairs you wish in the file properties
  - Properties are versioned over time, just like file contents
- Properties starting with "`svn:`" are reserved

  (svn:mime-type, svn:executable, svn:keywords, svn:eol-style...)

```
svn propset property-name property-value file
svn propedit property-name file
svn proplist file
svn propdel property-name file
```

- As with file contents, property changes are local to the working copy and are made permanent only by a commit

Revision Control - 3rd Generation

Generation	Networking	Granularity	Concurrency	Examples
First	None	One file at a time	Locks	RCS, SCCS
Second	Centralized	Multi-file	Merge before commit	CVS, Subversion
Third	Distributed	Changesets	Commit before merge	Git, mercurial

- The third generation allows merge and commit operations to be separated
- Most of the basic concepts are still valid, with the addition of a couple of new ones

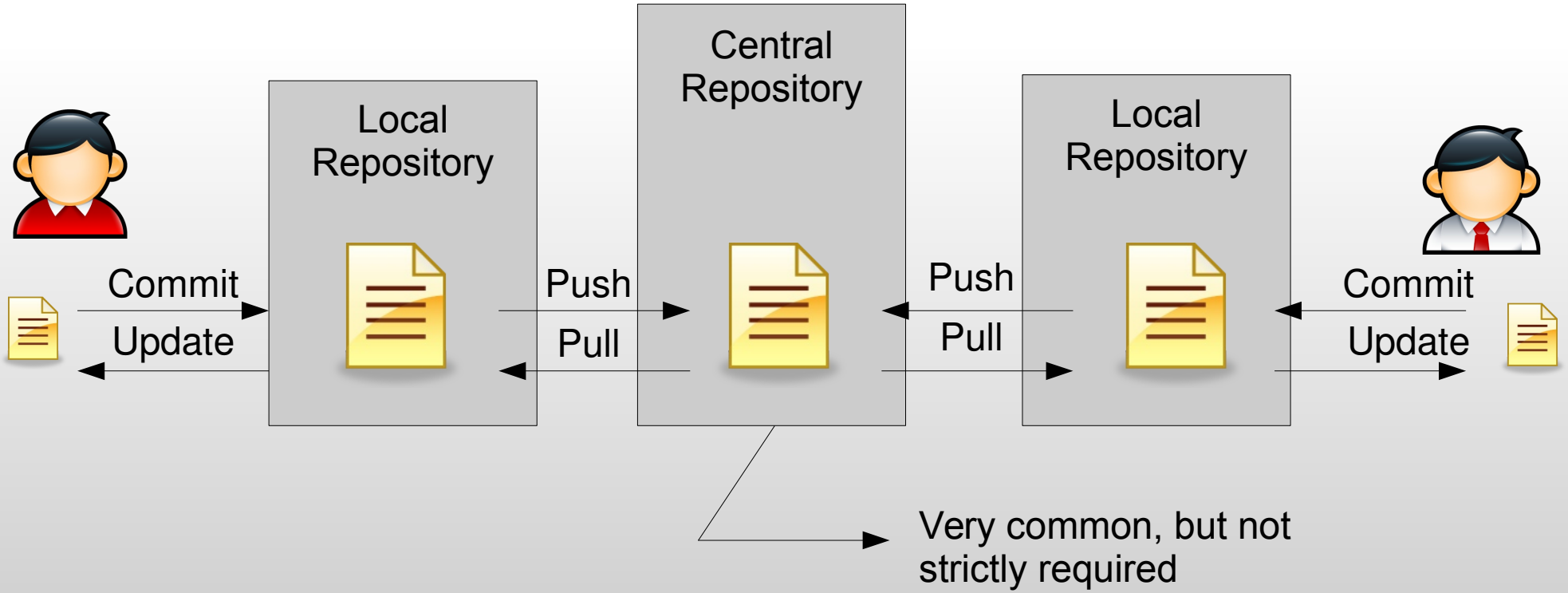
Distributed Versioning Systems

- No single central repository is required
  (but it is often used anyway)
- Each user works directly with his local copy of the repository
- The user choose when the changes are propagated between different repositories

- Advantages
  - Support offline development

Cloning

- The main difference in a distributed version control system is the notion of repository
  - Starting from an initial instance, the repository is then cloned to create multiple copies



Pushing and Pulling

- Users commit their changes to their local repository
  - No need to be connected for that
  - No need to merge other people changes before commit
  - No need to publicly share the changes
    (can be used to version a draft that does not compile)
- Different copies of the repositories are synchronized by performing pulling and pushing operations
  - This may require to merge differences

GIT

- Initially designed by Linus Torvalds for versioning the linux kernel
- First version released in 2005

- Three main concepts:
  - Workspace
    the working space of the user
  - Index
    Staging area between the workspace and the local repository. It keeps track of what needs to be committed
  - Local repository
    A copy of the project repository stored on the user machine

Git 101

- Cloning a repository

```
> git clone http://foo.bar.com:8000/  ./project
Cloning into project ...
```

- Adding something to the staging area

```
> vim fileA.txt
…
> git add fileA.txt
```

- Commit

```
> git commit
```

Git 101

- Push your commit to the rest of the team

```
> git push --all
```

In case of conflicts, the push operation will fail

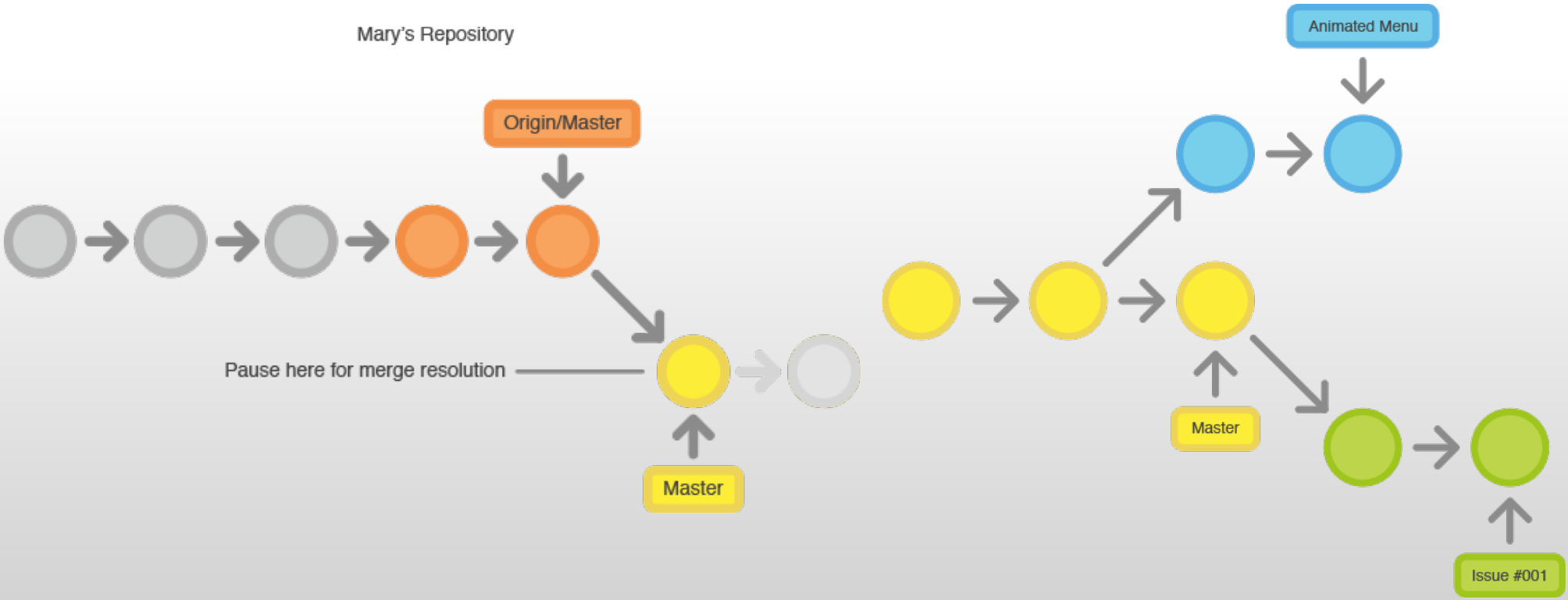
- Getting changes from other users

```
> git pull
```

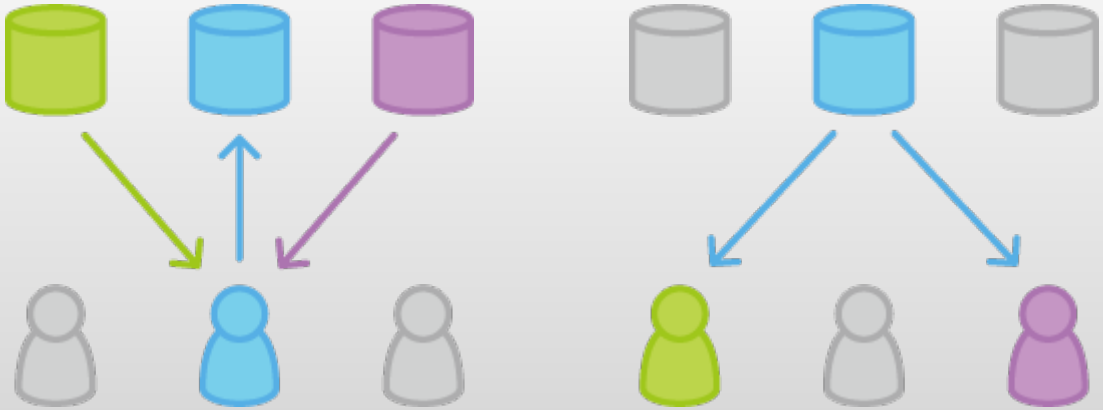
This update both the local repository AND the working copy
(therefore it may require a merge)
To only update the repository you can use the fetch command

Workflows

- A workflow defines a set of collaboration patterns adopted to manage a particular project
- They may be very simple (as in SVN), but GIT also allows for quite complex workflows




v0.1
v0.2
v1.0
Master
Develop
Feature
Feature



Git - SVN

`git init` `git add . ; git commit`	`svnadmin create repo` `svn import` *`URL`*
`git status`	`svn status`
`git {add|rm|mv}` *`file`*	`svn {add|rm|mv}` *`file`*
`git commit -a`	`svn commit`
`git clone` *`URL`*	`svn checkout` *`URL`*
`git pull`	`svn update`
`git tag -a` *`name`*	`svn copy` *`URL1 URL2`*
`git branch` *`branch-name`*	`svn copy` *`URL1 URL2`*

Need GIT for your projects?

- GitHub is a Git repository hosting service
  - Free for public repositories
  - Costs money if you want to have private ones

- Through Eurecom email you can get a free account for five private repositories and the student pack and lots of free tools

  → https://education.github.com/