

# FAST, FLEXIBLE AND FUN: REVISION CONTROL WITH MERCURIAL

Martin Geisler  
<mg@lazybytes.net>

JAOO Geek Night in Aarhus  
August 25, 2009

# OUTLINE

## INTRODUCTION

- Basic Concepts
- Historic Context
- Subversion in Detail

## USING MERCURIAL

- The Underlying Model
- Workflows
- Using History

## COOL EXTENSIONS

- Changing History
- Talking to Other Systems
- Third-Party Tools

## WRAPPING UP

- Alternatives
- Unsolved Problems
- Conclusion

# OUTLINE

## INTRODUCTION

- Basic Concepts
- Historic Context
- Subversion in Detail

## USING MERCURIAL

- The Underlying Model
- Workflows
- Using History

## COOL EXTENSIONS

- Changing History
- Talking to Other Systems
- Third-Party Tools

## WRAPPING UP

- Alternatives
- Unsolved Problems
- Conclusion

# FEATURES IN VERSION CONTROL SYSTEMS

Universal features:

- ▶ let you save changes
- ▶ browser the history



# FEATURES IN VERSION CONTROL SYSTEMS

Universal features:

- ▶ let you save changes
- ▶ browser the history

Common features:

- ▶ collaboration
- ▶ branches
- ▶ tags

# FEATURES IN VERSION CONTROL SYSTEMS

Universal features:

- ▶ let you save changes
- ▶ browser the history

Common features:

- ▶ collaboration
- ▶ branches
- ▶ tags

Not so common features:

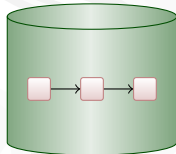
- ▶ speed!
- ▶ much more, as you will see...

# THE REPOSITORY

The database that holds your history:

- ▶ series of project-wide snapshots
- ▶ called changesets or revisions in Mercurial
- ▶ changesets encapsulate a delta between two snapshots
- ▶ each changeset has a parent changeset

Repository



# THE WORKING DIRECTORY

Holds your files:

Alice

hello.c  
Makefile

Bob

hello.c  
goodbye.c  
Makefile

- ▶ you edit files in the working directory
- ▶ reflects a changeset (the parent changeset)
- ▶ also called a working copy



# BRANCHES

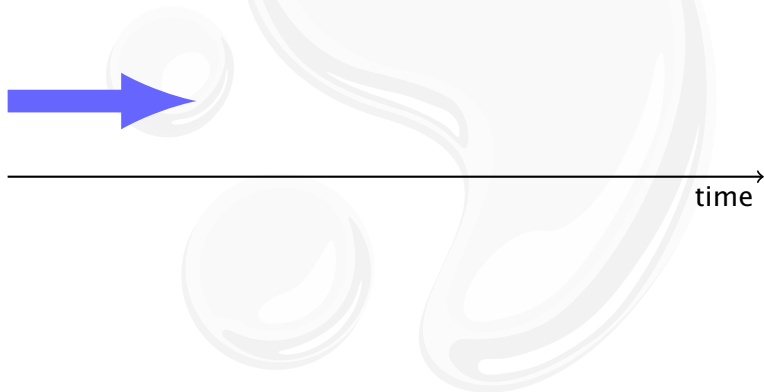
A key concept:

- ▶ parallel lines of development
- ▶ used to track releases
- ▶ used to isolate disruptive changes

# BRANCHES

A key concept:

- ▶ parallel lines of development
- ▶ used to track releases
- ▶ used to isolate disruptive changes



# BRANCHES

A key concept:

- ▶ parallel lines of development
- ▶ used to track releases
- ▶ used to isolate disruptive changes



# BRANCHES

A key concept:

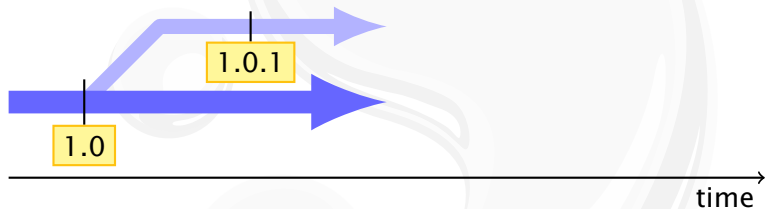
- ▶ parallel lines of development
- ▶ used to track releases
- ▶ used to isolate disruptive changes



# BRANCHES

A key concept:

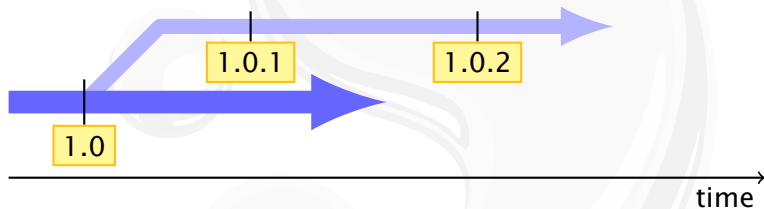
- ▶ parallel lines of development
- ▶ used to track releases
- ▶ used to isolate disruptive changes



# BRANCHES

A key concept:

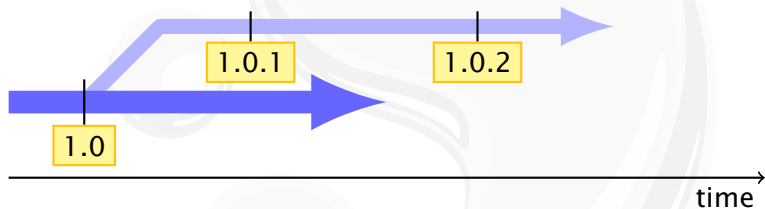
- ▶ parallel lines of development
- ▶ used to track releases
- ▶ used to isolate disruptive changes



# MERGING

The opposite of branching:

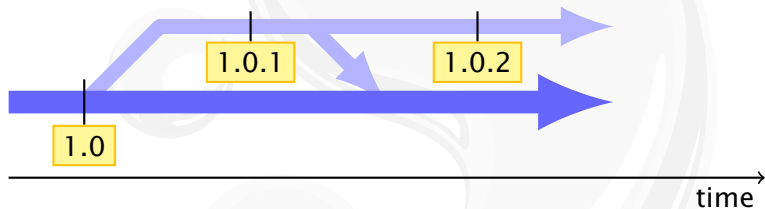
- ▶ combines two branches
- ▶ used to merge back bugfixes
- ▶ used to integrate feature branches



# MERGING

The opposite of branching:

- ▶ combines two branches
- ▶ used to merge back bugfixes
- ▶ used to integrate feature branches

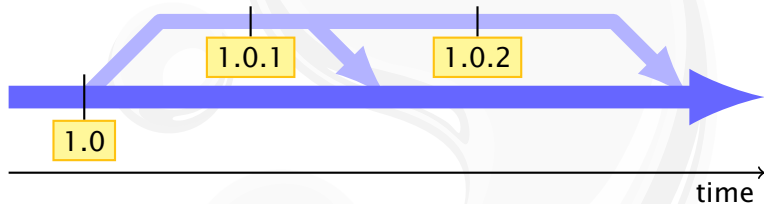




# MERGING

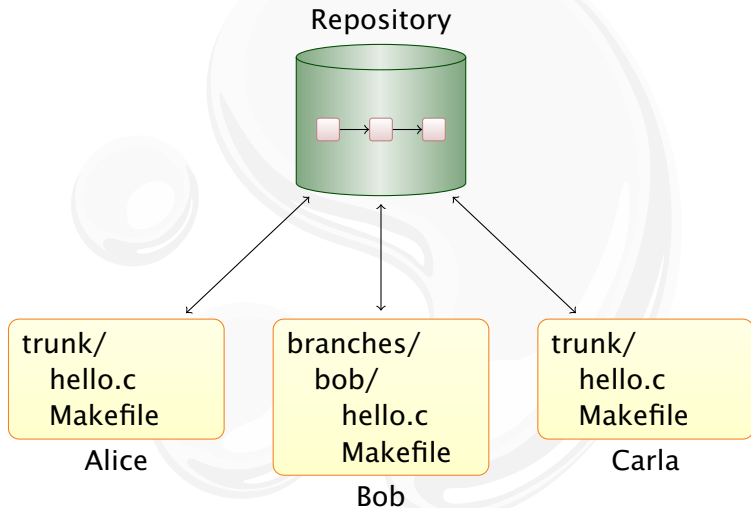
The opposite of branching:

- ▶ combines two branches
- ▶ used to merge back bugfixes
- ▶ used to integrate feature branches



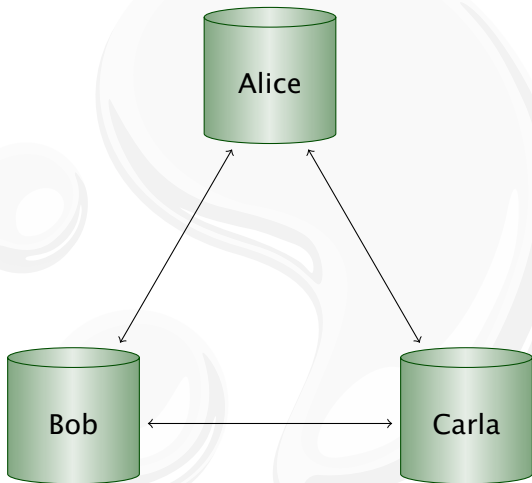
# CENTRALIZED REVISION CONTROL

Subversion use a single server:



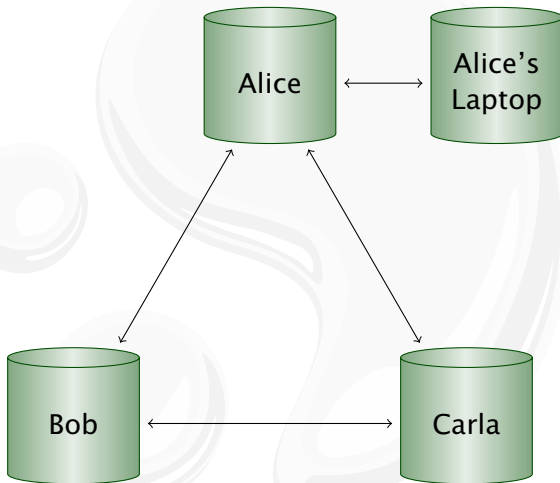
# DISTRIBUTED REVISION CONTROL

Mercurial duplicates the history on many servers:



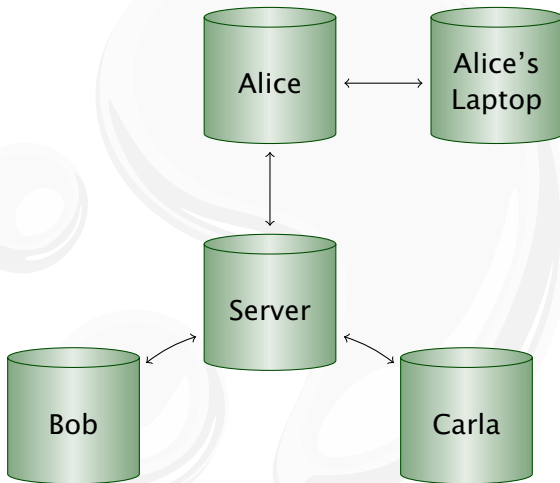
# DISTRIBUTED REVISION CONTROL

Mercurial duplicates the history on many servers:



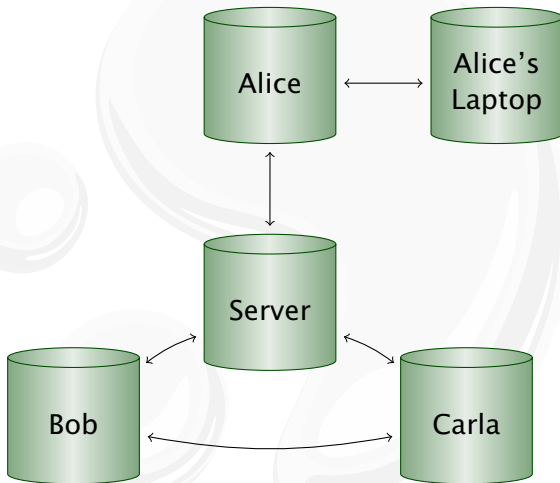
# DISTRIBUTED REVISION CONTROL

Mercurial duplicates the history on many servers:



# DISTRIBUTED REVISION CONTROL

Mercurial duplicates the history on many servers:



# REVISION CONTROL SYSTEM (RCS)

Version control from the 1980s:

- ▶ store changes (date, commit message)
- ▶ browse history

# REVISION CONTROL SYSTEM (RCS)

Version control from the 1980s:

- ▶ store changes (date, commit message)
- ▶ browse history

hello.c:

Makefile:



# REVISION CONTROL SYSTEM (RCS)

Version control from the 1980s:

- ▶ store changes (date, commit message)
- ▶ browse history

hello.c: 

Makefile: 

# REVISION CONTROL SYSTEM (RCS)

Version control from the 1980s:

- ▶ store changes (date, commit message)
- ▶ browse history

hello.c: 

Makefile: 

# REVISION CONTROL SYSTEM (RCS)

Version control from the 1980s:

- ▶ store changes (date, commit message)
- ▶ browse history


hello.c: 

Makefile: 

# REVISION CONTROL SYSTEM (RCS)

Version control from the 1980s:

- ▶ store changes (date, commit message)
- ▶ browse history


hello.c: 

Makefile: 

# REVISION CONTROL SYSTEM (RCS)

Version control from the 1980s:

- ▶ store changes (date, commit message)
- ▶ browse history

hello.c: 

Makefile: 

# CONCURRENT VERSIONS SYSTEM (CVS)

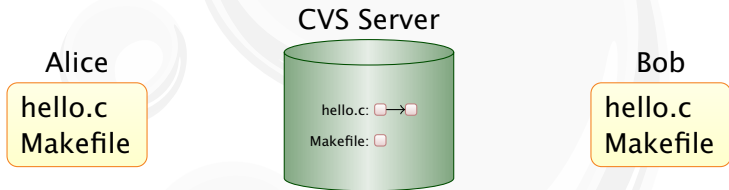
Version control in the 1990s:

- ▶ collaboration over the network
- ▶ several working copies
- ▶ one central repository

# CONCURRENT VERSIONS SYSTEM (CVS)

Version control in the 1990s:

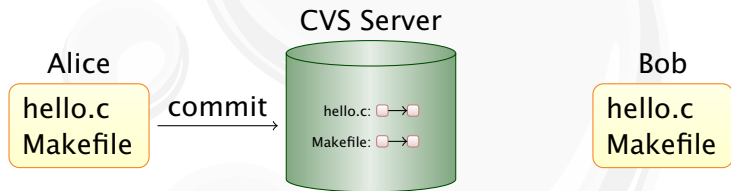
- ▶ collaboration over the network
- ▶ several working copies
- ▶ one central repository



# CONCURRENT VERSIONS SYSTEM (CVS)

Version control in the 1990s:

- ▶ collaboration over the network
- ▶ several working copies
- ▶ one central repository

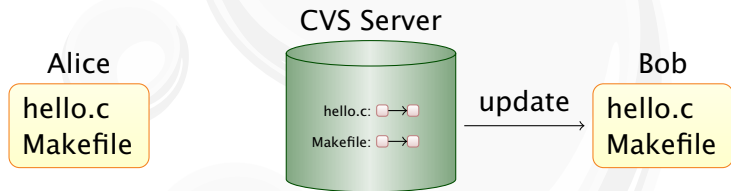




# CONCURRENT VERSIONS SYSTEM (CVS)

Version control in the 1990s:

- ▶ collaboration over the network
- ▶ several working copies
- ▶ one central repository



## SO, EVERYTHING IS GOOD?

CVS has many limitations:

- ▶ no atomic commits
- ▶ no rename information
- ▶ branches? I hope you have a CVS guru at hand...

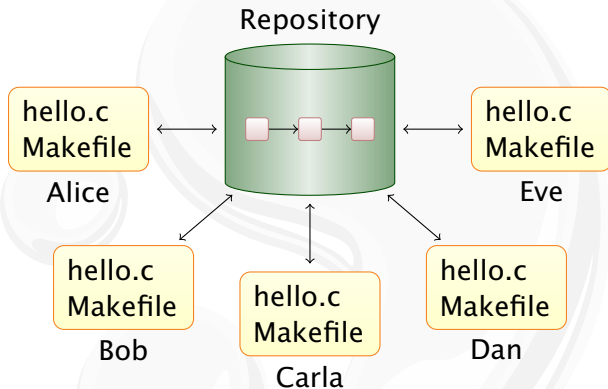
## SUBVERSION (SVN)

Version control from this millennia — “*CVS done right*”:

- ▶ atomic commits
- ▶ tracks renames
- ▶ supports some operations without network access
- ▶ clean and simple design compared to CVS

# MERGING IN SUBVERSION

SVN is a centralized system:



- ▶ merging takes place in client
- ▶ merging takes place on server(!)

## SVN MERGE IN CLIENT

When you do `svn update`, you merge:

- ▶ this can of course result in conflicts
- ▶ you must resolve the merge before you go on
- ▶ might be difficult to handle everything at once

## SVN MERGE IN CLIENT

When you do `svn update`, you merge:

- ▶ this can of course result in conflicts
- ▶ you must resolve the merge before you go on
- ▶ might be difficult to handle everything at once

What if you want to abandon a merge?

- ▶ you can revert files with conflicts to their old state
- ▶ but other files may have been updated
- ▶ beware of mixed revisions...

## MIXED REVISIONS?!

SVN lets you work with an **inconsistent** working copy:

- ▶ Alice and Bob have revision 1
- ▶ Alice changes `hello.c` → revision 2
- ▶ Bob changes `Makefile` → **revision 3**

## MIXED REVISIONS?!

SVN lets you work with an **inconsistent** working copy:

- ▶ Alice and Bob have revision 1
- ▶ Alice changes `hello.c` → revision 2
- ▶ Bob changes `Makefile` → **revision 3**

They both have mixed revisions in their working copies:

hello.c (r2)  
Makefile (r1)

Alice

hello.c (r1)  
Makefile (r3)

Bob

Difficult for Carla to reproduce either working copy.



## SERVER-SIDE MERGES

Mixed revisions are a result of allowing server-side merges

- ▶ can we forbid server-side merges?
- ▶ not really — it would ruin branches in SVN

## BRANCHES IN SVN

Subversion knows nothing about branches!

- ▶ but SVN has a cheap copy mechanism
- ▶ used for tags and branches

## BRANCHES IN SVN

Subversion knows nothing about branches!

- ▶ but SVN has a cheap copy mechanism
- ▶ used for tags and branches

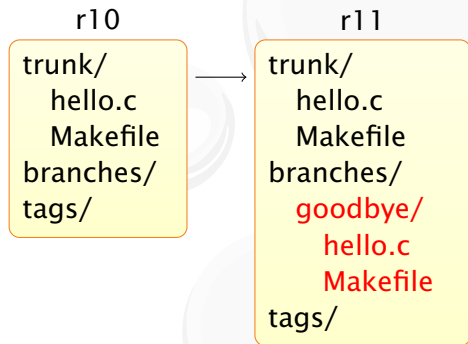
r10

```
trunk/  
  hello.c  
  Makefile  
branches/  
tags/
```

## BRANCHES IN SVN

Subversion knows nothing about branches!

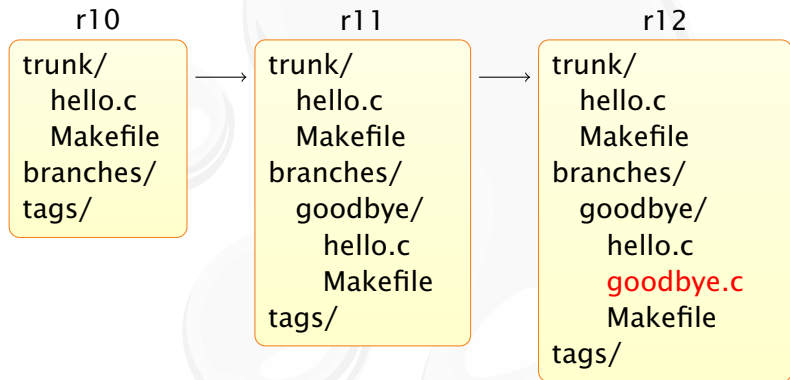
- ▶ but SVN has a cheap copy mechanism
- ▶ used for tags and branches



## BRANCHES IN SVN

Subversion knows nothing about branches!

- ▶ but SVN has a cheap copy mechanism
- ▶ used for tags and branches



## MERGING BRANCHES IN SVN

Branches are only interesting if you can merge them:

- ▶ before SVN 1.5: no built-in support for merging(!)
- ▶ SVN 1.5 and later: tracks info needed for merging

## MERGING BRANCHES IN SVN

Branches are only interesting if you can merge them:

- ▶ before SVN 1.5: no built-in support for merging(!)
- ▶ SVN 1.5 and later: tracks info needed for merging

The bottom line is that Subversion's merge-tracking feature has an **extremely complex** internal implementation, and the `svn:mergeinfo` property is the only window the user has into the machinery. Because the feature is **relatively new**, a numbers of edge cases and possible unexpected behaviors may pop up.

—*Version Control with Subversion*

(Mercurial has robust built-in support for merging branches.)

# OUTLINE

## INTRODUCTION

- Basic Concepts
- Historic Context
- Subversion in Detail

## USING MERCURIAL

- The Underlying Model
- Workflows
- Using History

## COOL EXTENSIONS

- Changing History
- Talking to Other Systems
- Third-Party Tools

## WRAPPING UP

- Alternatives
- Unsolved Problems
- Conclusion





**Live Demo!**

## THE UNDERLYING MODEL

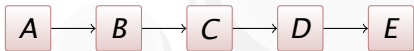
A Mercurial changeset conceptually consist of:

- ▶ 0-2 parent changeset IDs:
  - ▶ root changeset has no parents
  - ▶ normal commits have one parent
  - ▶ merge changesets have two parents
- ▶ date, username, commit message
- ▶ difference from first parent
- ▶ changeset ID is computed as SHA-1 hash of the above

# IMMUTABLE HISTORY

SHA-1 hashes as changeset IDs have some consequences:

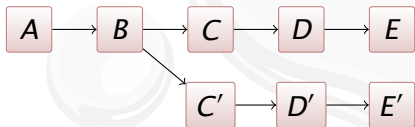
- ▶ a changeset ID is a hash of the entire history
- ▶ changing history changes subsequent changesets
- ▶ history is immutable, you can only make new history:



# IMMUTABLE HISTORY

SHA-1 hashes as changeset IDs have some consequences:

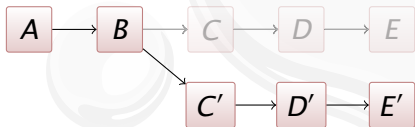
- ▶ a changeset ID is a hash of the entire history
- ▶ changing history changes subsequent changesets
- ▶ history is immutable, you can only make new history:



# IMMUTABLE HISTORY

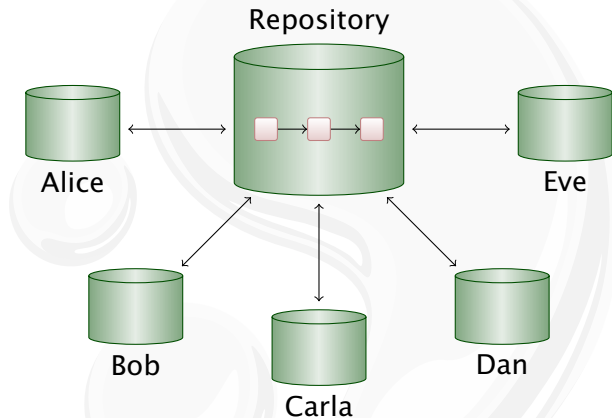
SHA-1 hashes as changeset IDs have some consequences:

- ▶ a changeset ID is a hash of the entire history
- ▶ changing history changes subsequent changesets
- ▶ history is immutable, you can only make new history:



# CENTRAL WORKFLOW

Everybody has write access to a central repository:



# PULL WORKFLOW

People have read-only access (e.g., `hg serve`):

Alice

0

Bob

0

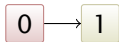
Carla

0

# PULL WORKFLOW

People have read-only access (e.g., `hg serve`):

Alice



Bob



Carla





# PULL WORKFLOW

People have read-only access (e.g., `hg serve`):

Alice



Bob



Carla



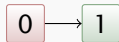
# PULL WORKFLOW

People have read-only access (e.g., `hg serve`):

Alice



Bob

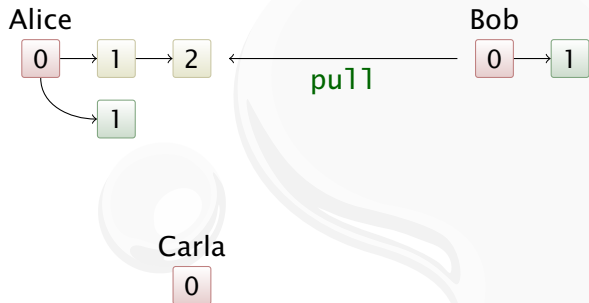


Carla



## PULL WORKFLOW

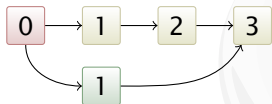
People have read-only access (e.g., `hg serve`):



## PULL WORKFLOW

People have read-only access (e.g., `hg serve`):

Alice



Bob



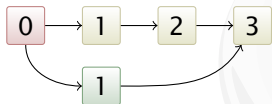
Carla



## PULL WORKFLOW

People have read-only access (e.g., `hg serve`):

Alice



Bob



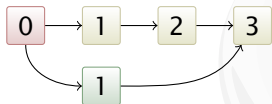
Carla



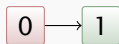
## PULL WORKFLOW

People have read-only access (e.g., `hg serve`):

Alice



Bob

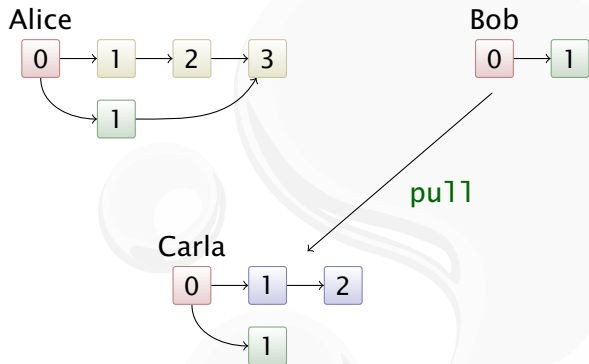


Carla



## PULL WORKFLOW

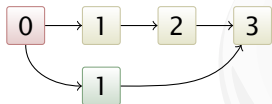
People have read-only access (e.g., `hg serve`):



## PULL WORKFLOW

People have read-only access (e.g., `hg serve`):

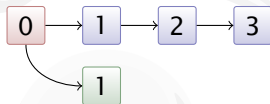
Alice



Bob



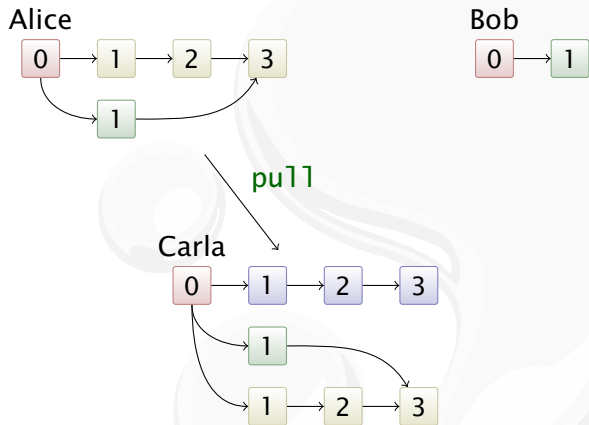
Carla





## PULL WORKFLOW

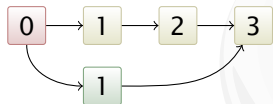
People have read-only access (e.g., `hg serve`):



## PULL WORKFLOW

People have read-only access (e.g., `hg serve`):

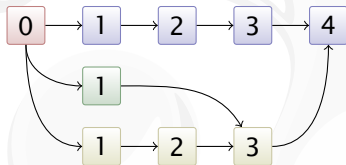
Alice



Bob



Carla



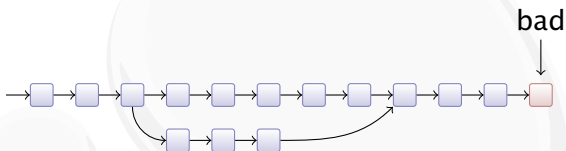
## EXPLORING HISTORY

Everything is local and fast:

- ▶ `hg log` can search in commit messages and usernames
- ▶ `hg grep` searches in file content
- ▶ `hg bisect` makes a binary search on the revision graph

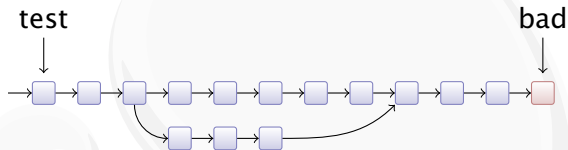
## REVISION GRAPH BISECTION

You've found a bug! When was it first introduced?  
Use `hg bisect` to mark good and bad revisions:



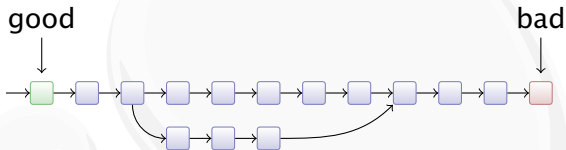
## REVISION GRAPH BISECTION

You've found a bug! When was it first introduced?  
Use `hg bisect` to mark good and bad revisions:



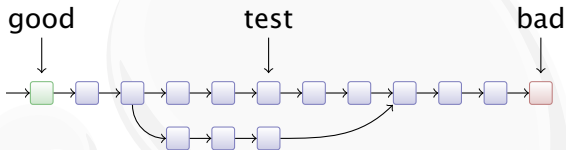
## REVISION GRAPH BISECTION

You've found a bug! When was it first introduced?  
Use `hg bisect` to mark good and bad revisions:



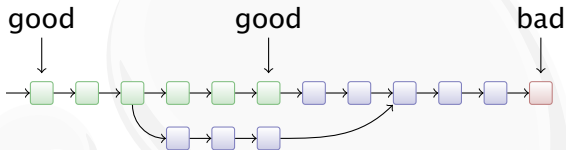
## REVISION GRAPH BISECTION

You've found a bug! When was it first introduced?  
Use `hg bisect` to mark good and bad revisions:



## REVISION GRAPH BISECTION

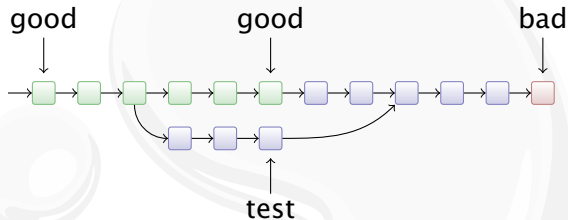
You've found a bug! When was it first introduced?  
Use `hg bisect` to mark good and bad revisions:





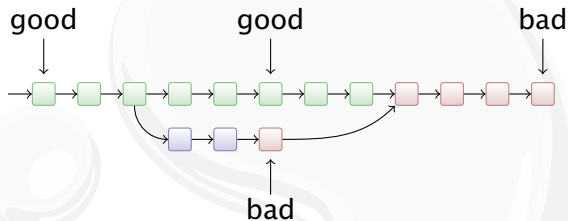
## REVISION GRAPH BISECTION

You've found a bug! When was it first introduced?  
Use `hg bisect` to mark good and bad revisions:



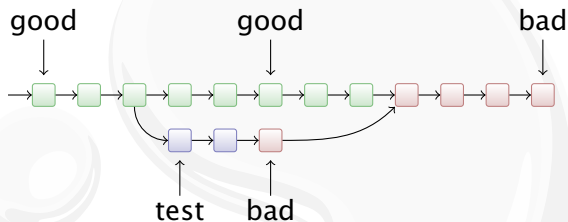
## REVISION GRAPH BISECTION

You've found a bug! When was it first introduced?  
Use `hg bisect` to mark good and bad revisions:



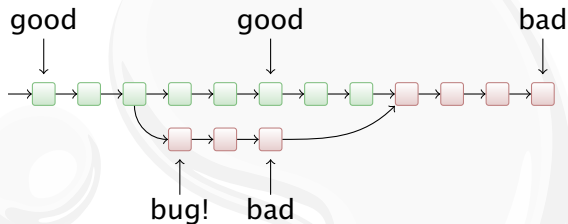
## REVISION GRAPH BISECTION

You've found a bug! When was it first introduced?  
Use `hg bisect` to mark good and bad revisions:



## REVISION GRAPH BISECTION

You've found a bug! When was it first introduced?  
Use `hg bisect` to mark good and bad revisions:



# OUTLINE

## INTRODUCTION

- Basic Concepts
- Historic Context
- Subversion in Detail

## USING MERCURIAL

- The Underlying Model
- Workflows
- Using History

## COOL EXTENSIONS

- Changing History
- Talking to Other Systems
- Third-Party Tools

## WRAPPING UP

- Alternatives
- Unsolved Problems
- Conclusion

## MERCURIAL IS EXTENSIBLE

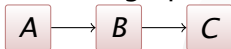
You can add new functionality to Mercurial:

- ▶ ships with 30 extensions
- ▶ wiki lists 57 extensions
- ▶ extensions can change basically everything
- ▶ helps to keep the core small and focused

## MOVING CHANGESETS AROUND

Tired of all those merges? Use the **rebase** extension!

- ▶ Revision graph:



# MOVING CHANGESETS AROUND

Tired of all those merges? Use the **rebase** extension!

- ▶ Revision graph:

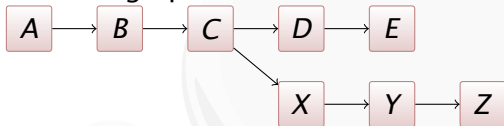




# MOVING CHANGESETS AROUND

Tired of all those merges? Use the **rebase** extension!

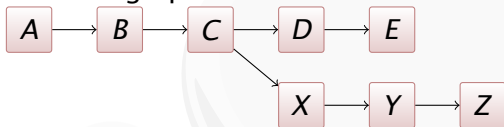
- ▶ Revision graph:



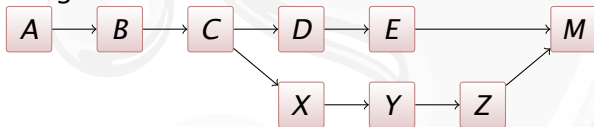
# MOVING CHANGESETS AROUND

Tired of all those merges? Use the **rebase** extension!

- ▶ Revision graph:



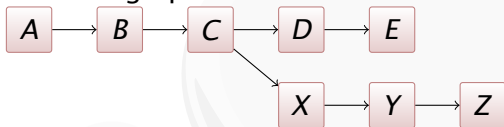
- ▶ Merge:



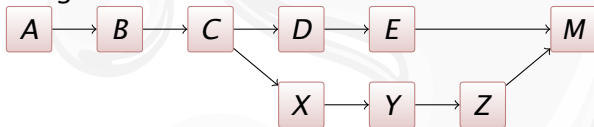
# MOVING CHANGESETS AROUND

Tired of all those merges? Use the **rebase** extension!

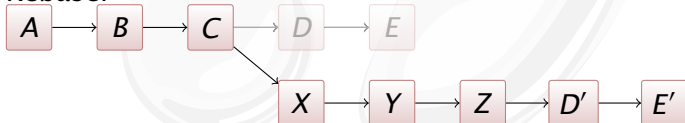
- ▶ Revision graph:



- ▶ Merge:



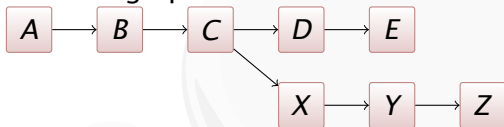
- ▶ Rebase:



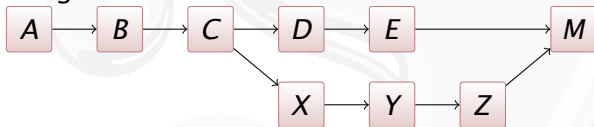
# MOVING CHANGESETS AROUND

Tired of all those merges? Use the **rebase** extension!

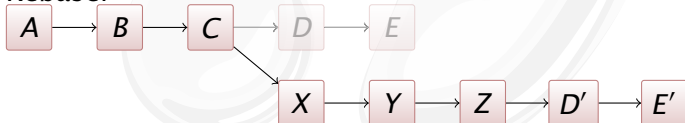
- ▶ Revision graph:



- ▶ Merge:



- ▶ Rebase:



- ▶ Beware: public changes should never be rebased.

## MAINTAINING PATCH SERIES

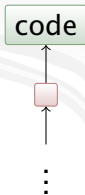
The `mq` extension makes it easy to maintain a patch series:



Works nicely for local modification for upstream sources.

## MAINTAINING PATCH SERIES

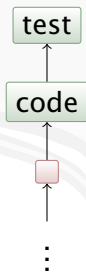
The `mq` extension makes it easy to maintain a patch series:



Works nicely for local modification for upstream sources.

## MAINTAINING PATCH SERIES

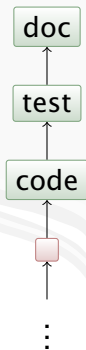
The `mq` extension makes it easy to maintain a patch series:



Works nicely for local modification for upstream sources.

## MAINTAINING PATCH SERIES

The `mq` extension makes it easy to maintain a patch series:

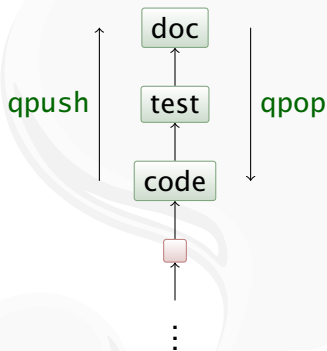


Works nicely for local modification for upstream sources.



## MAINTAINING PATCH SERIES

The `mq` extension makes it easy to maintain a patch series:



Works nicely for local modification for upstream sources.

# EDITING HISTORY

Inspired by `git rebase -i`, `histedit` lets you

- ▶ reorder changesets:



# EDITING HISTORY

Inspired by `git rebase -i`, `histedit` lets you

- ▶ reorder changesets:



- ▶ fold changesets:



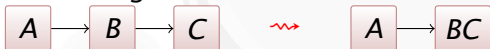
# EDITING HISTORY

Inspired by `git rebase -i`, `histedit` lets you

- ▶ reorder changesets:



- ▶ fold changesets:



- ▶ drop changesets:



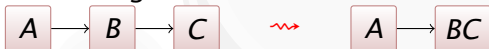
# EDITING HISTORY

Inspired by `git rebase -i`, `histedit` lets you

- ▶ reorder changesets:



- ▶ fold changesets:



- ▶ drop changesets:



- ▶ edit changesets:



# MIGRATING HISTORY

The **convert** extension can import history:

- ▶ CVS, SVN, Git, Bazaar, Darcs, . . .
- ▶ incremental conversion
- ▶ many options for fiddling with branches, authors, . . .

## MIGRATING HISTORY

The `convert` extension can import history:

- ▶ CVS, SVN, Git, Bazaar, Darcs, ...
- ▶ incremental conversion
- ▶ many options for fiddling with branches, authors, ...

Interestingly, `convert` can import from Mercurial:

- ▶ `--filemap` lets you exclude and rename files
- ▶ `--branchmap` lets you rename branches

## INTERFACING WITH SUBVERSION

The `hgsubversion` extension let's you:

- ▶ use `hg clone` on a SVN URLs
- ▶ use `hg pull` to convert new SVN revisions
- ▶ use `hg push` to commit changesets to SVN server

The extension is being used, but please note:

- ▶ it is not yet feature complete
- ▶ requires standard trunk/, branches/ and tags/ layout

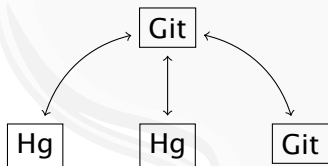




## INTERFACING WITH GIT

Need to work on a Git repository? Try **hg-git**!

- ▶ Mercurial extension: you get the nice **hg** command line
- ▶ round-tripping: changeset hashes are preserved

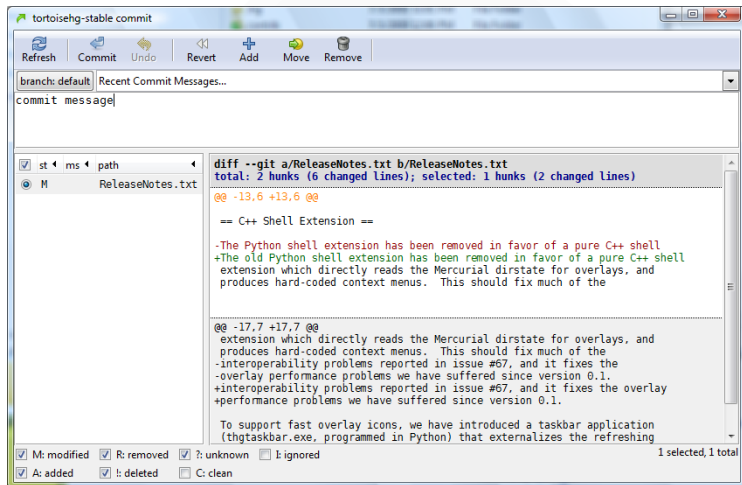


## THIRD-PARTY TOOLS

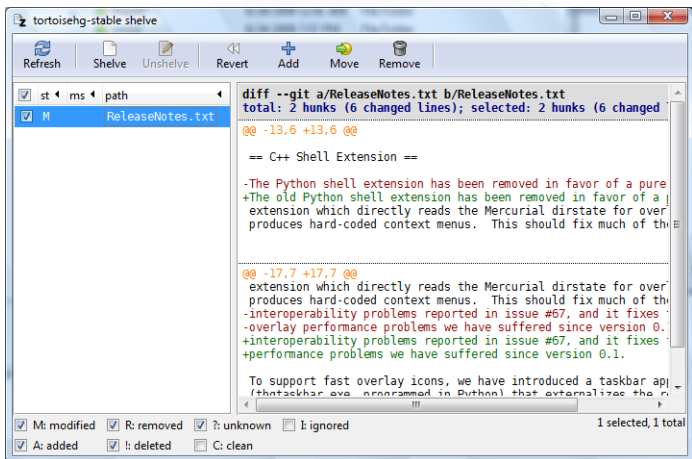
Tools with Mercurial support:

- ▶ Shell integration: TortoiseHg (Windows, Mac, Linux)
- ▶ IDEs: Eclipse, NetBeans, IntelliJ, Visual Studio, Emacs...
- ▶ Project Support: Trac, JIRA, Maven, Hudson, BuildBot...
- ▶ Ant tasks

# TORTOISEHG SCREENSHOTS



# TORTOISEHG SCREENSHOTS



# TORTOISEHG SCREENSHOTS

DataMining - tortoisehg-stable

New Search Stop

hgtk@3023

| Graph | Rev  | Summary  | User        |
|-------|------|--|-------------|
| ●     | 3023 | hgtk: fix early forking problems by not using fork | Steve Borho |
| ●     | 2897 | hgtk: introduce --nofork global option             | Steve Borho |
| ●     | 2890 | hgtk: fix indexing errors in nofork hack           | Steve Borho |
| ●     | 2846 | hgtk: improve argument detection before forking    | Steve Borho |
| ●     | 2832 | hgtk: enable forking behavior for GUI commands     | Steve Borho |

| Line | Rev  | Source   |
|------|------|--|
| 5    | 1725 | # Copyright (C) 2008-9 Steve Borho <steve@borho.org> |
| 6    | 1725 | # Copyright (C) 2008 TK Soh <teekaysoh@gmail.com>    |
| 7    | 1725 | #  |
| 8    | 1725 |  |
| 9    | 2340 | import os  |
| 10   | 2340 | import sys   |
| 11   | 2340 |  |
| 12   | 2340 | if hasattr(sys, "frozen"):                           |

Done

# OUTLINE

## INTRODUCTION

- Basic Concepts
- Historic Context
- Subversion in Detail

## USING MERCURIAL

- The Underlying Model
- Workflows
- Using History

## COOL EXTENSIONS

- Changing History
- Talking to Other Systems
- Third-Party Tools

## WRAPPING UP

- Alternatives
- Unsolved Problems
- Conclusion

# GIT



The version control system by Linus Torvalds:

- ▶ distributed, fast, scalable
- ▶ used by Linux Kernel, Perl, and many others
- ▶ written in C, shell scripts(!) and a little Perl
- ▶ models history the same way as Mercurial

# GIT DIFFERENCES



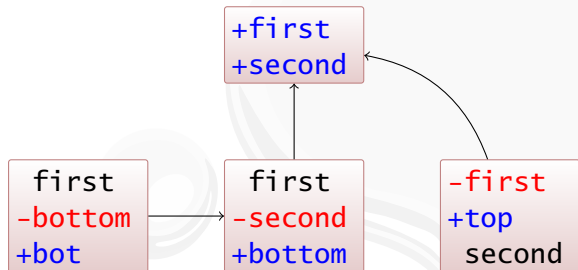
Git is different in a number of ways:

- ▶ quite verbose compared to Mercurial
- ▶ core Mercurial never rewrites history, Git has it built-in
- ▶ changes are added to an “index” before commit:

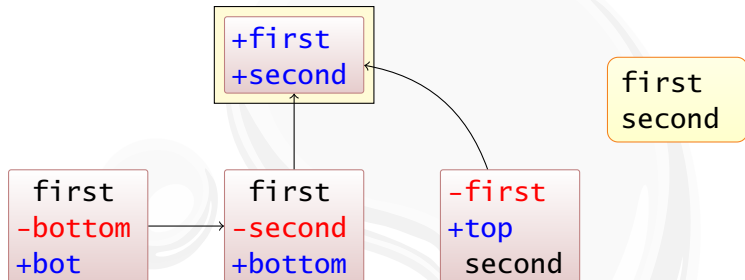
```
% git add .
% git commit -m 'Initial import.'
[master (root-commit) ba41f97] Initial import.
 2 files changed, 7 insertions(+), 0 deletions(-)
 create mode 100644 Makefile
 create mode 100644 hello.c
% echo '/* The End */' >> hello.c
% git add hello.c
% git commit -m 'Added comment.'
[master 2489734] Added comment.
 1 files changed, 1 insertions(+), 0 deletions(-)
```



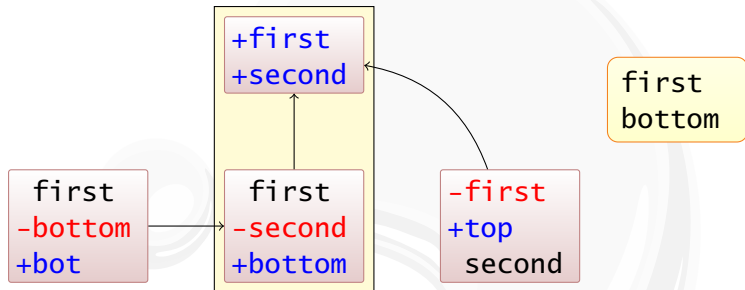
Repository consist of **unordered** changesets:



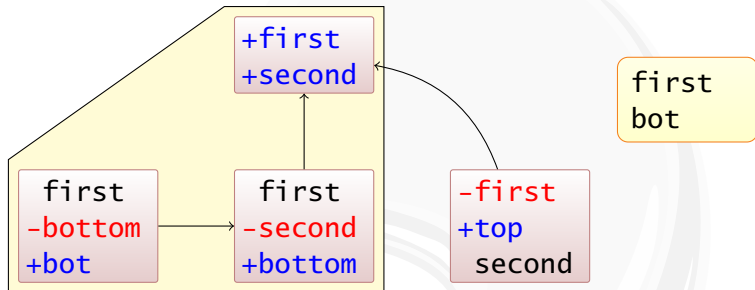
Repository consist of **unordered** changesets:



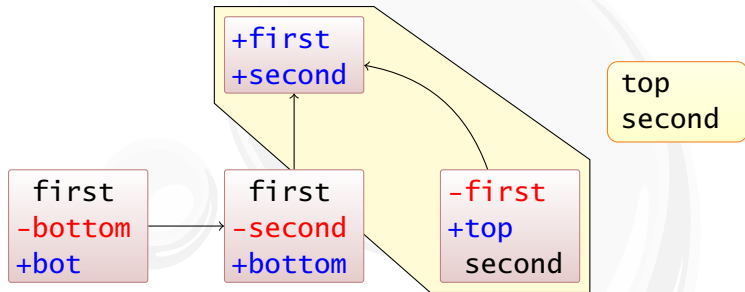
Repository consist of **unordered** changesets:



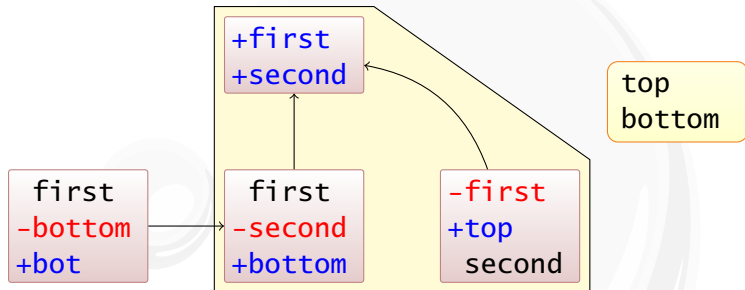
Repository consist of **unordered** changesets:



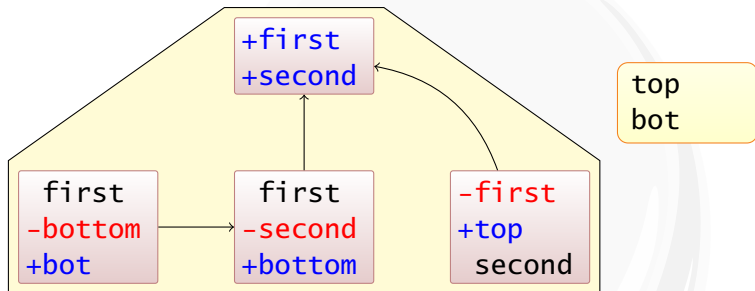
Repository consist of **unordered** changesets:



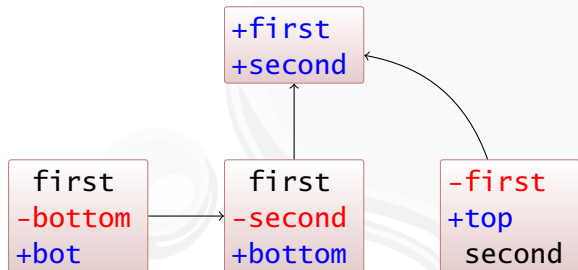
Repository consist of **unordered** changesets:



Repository consist of **unordered** changesets:



Repository consist of **unordered** changesets:



Why isn't everybody using Darcs?

- ▶ determining if a change commutes with another is slow
- ▶ merges could take hours or days with older versions



## UNSOLVED PROBLEMS

DVCSs generally have problems with

- ▶ large binary files
- ▶ retrieving only a sub-tree (narrow clones)
- ▶ retrieving only recent history (shallow clones)

Some more Mercurial specific problems are:

- ▶ it is fairly easy to create conflicts in `.hgtags` files
- ▶ filename encodings across different filesystems

## MERCURIAL IN A NUTSHELL

Mercurial changes the way you develop:

- ▶ simple yet strong model for **both** branching and merging
- ▶ power tool instead of necessary evil
- ▶ light-weight and snappy

## MORE INFORMATION

- ▶ Mercurial homepage:  
<http://mercurial.selenic.com/>
- ▶ *Mercurial: The Definitive Guide*:  
<http://hgbook.red-bean.com/>
- ▶ Some free Mercurial hosting sites:  
<http://www.bitbucket.org/>  
<http://code.google.com/>  
<http://sourceforge.net/>

## MORE INFORMATION

- ▶ Mercurial homepage:  
<http://mercurial.selenic.com/>
- ▶ *Mercurial: The Definitive Guide*:  
<http://hgbook.red-bean.com/>
- ▶ Some free Mercurial hosting sites:  
<http://www.bitbucket.org/>  
<http://code.google.com/>  
<http://sourceforge.net/>

**Thank you!**