Fast, Flexible and Fun: Revision Control with Mercurial

Martin Geisler
⟨mg@aragost.com⟩

IBM Research – Zurich
September 16th, 2010
About the Speaker

Martin Geisler:

➤ core Mercurial developer:
  ▶ reviews patches from the community
  ▶ helps users in our IRC channel
➤ PhD in Computer Science from Aarhus University, DK
  ▶ exchange student at ETH Zürich in 2005
  ▶ visited ZRL for three months in 2008
➤ now working at aragost Trifork, Zürich
OUTLINE

INTRODUCTION

USING MERcurIAL
  Workflows
  Branching
  The Underlying Model
  Using History

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

WRAPPING UP
Outline

Introduction

Using Mercurial
  Workflows
  Branching
  The Underlying Model
  Using History

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

Wrapping Up
What is Mercurial?

Main features:

- fast, distributed revision control system
- robust support for branching and merging
- very flexible and extensible
Who is Using it?

Mercurial is used by:

- Oracle for Java, OpenSolaris, NetBeans, OpenOffice, . . .
- Mozilla for Firefox, Thunderbird, . . .
- Google
- many more . . .
Testimonials

Martin Fowler, 2010:  

http://martinfowler.com/
Testimonials

Joel Spolsky, 2010:  
http://www.joelonsoftware.com/

Mercurial is better than Subversion.  
It is a better way of working on source code with a team. It is a  
better way of working on source code by yourself.  
It is just better.
Centralized Revision Control

Single repository, multiple working copies:

Repository

hello.c
Makefile
Alice

goodbye.c
Makefile
Bob
Centralized Revision Control

Single repository, multiple working copies:

Repository

- Hello.c
- Makefile
- Alice

- Goodbye.c
- Makefile
- Bob

Drawbacks:
- network latency
- single point of failure
- constrained workflow
Distributed Revision Control

Mercurial duplicates the history on many servers:

Alice

Hello.c
Makefile

Bob

goodbye.c
Makefile
Distributed Revision Control

Mercurial duplicates the history on many servers:

Advantages:
- no network latency
- distributed, off-line operations
- no imposed workflow

Drawback(?):
- must synchronize repositories
**Moving Changesets Around**

Pull and merge:

Alice

Bob

Merging:

1. Find common ancestor of $A_2$ and $B_1$:
2. Compute differences between $I$ and $B_1$:
3. Apply them to $A_2$, taking renames into account.
Moving Changesets Around

Pull and merge:

Alice

\[ I \rightarrow A_1 \]

Bob

\[ I \]
Moving Changesets Around

Pull and merge:

Alice

Bob

I

A₁

A₂

Merging:

▶ find common ancestor of A₂ and B₁:

▶ compute differences between I and B₁:

▶ apply them to A₂, taking renames into account
Moving Changesets Around

Pull and merge:

Alice

Bob

I $\rightarrow$ $A_1$ $\rightarrow$ $A_2$

I $\rightarrow$ $B_1$

Merging:

1. Find common ancestor of $A_2$ and $B_1$.
2. Compute differences between $I$ and $B_1$.
3. Apply them to $A_2$, taking renames into account.
Moving Changesets Around

Pull and merge:

Alice

Bob

I → A₁ → A₂

I → B₁

pull

find common ancestor of A₂ and B₁:

compute differences between I and B₁

apply them to A₂, taking renames into account
Moving Changesets Around

Pull and merge:

Alice

\[ I \rightarrow A_1 \rightarrow A_2 \rightarrow A_3 \]

Bob

\[ I \rightarrow B_1 \]

merge
**Moving Changesets Around**

**Pull and merge:**

Alice

\[ I \rightarrow A_1 \rightarrow A_2 \rightarrow A_3 \]

Bob

\[ I \rightarrow B_1 \]

**Merging:**

- find common ancestor of \( A_2 \) and \( B_1 \): \( I \)
- compute differences between \( I \) and \( B_1 \)
- apply them to \( A_2 \), taking renames into account
**Key Mercurial Commands**

Local commands:

- `hg commit`: save a snapshot into the current repository
- `hg update`: checkout revision into working directory
- `hg merge`: join different lines of history
Key Mercurial Commands

Local commands:
- `hg commit`: save a snapshot into the current repository
- `hg update`: checkout revision into working directory
- `hg merge`: join different lines of history

Network commands:
- `hg pull`: retrieve changesets from another repository
- `hg push`: send your changesets to another repository
Outline

Introduction

Using Mercurial

- Workflows
- Branching
- The Underlying Model
- Using History

Cool Extensions

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

Wrapping Up
Outline

Introduction

Using Mercurial

Workflows
Branching
The Underlying Model
Using History

Cool Extensions

Changing History
Talking to Other Systems
Third-Party Tools

Wrapping Up
WORKFLOW IN A TEAM

Mercurial scales from a single team...:

Alice

Bob

Carla

Test

Prod
Workflow Between Company Divisions

... to enterprise-wide development...:
Workflow Between Two Computers

... to working with yourself:

bitbucket.org

Alice’s Desktop

Alice’s Laptop
OUTLINE

INTRODUCTION

USING MERCURIAL
  Workflows
  Branching
  The Underlying Model
  Using History

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

WRAPPING UP
Release Branches
Release Branches

1.0
Release Branches

1.0
Release Branches
Release Branches
Release Branches
Release Branches
Release Branches

1.0

1.0.1

1.0.2

18/42
Release Branches
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

```
$svn log
r10: trunk/ hello.c Makefile
     branches/
     tags/
```

```
r11: trunk/ hello.c Makefile
     branches/
     goodbye/ hello.c Makefile
     tags/
```
Subversion knows nothing about branches!

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

```
r10
trunk/
  hello.c
  Makefile
branches/
tags/

r11
trunk/
  hello.c
  Makefile
branches/
goodbye/
  hello.c
  Makefile
tags/

r12
trunk/
  hello.c
  Makefile
branches/
goodbye/
  goodbye/
  hello.c
  Makefile
tags/
```
Merging Branches in SVN

The support is incomplete and fragile:

- renamed files are not merged correctly
- old clients will not update the merge info
Merging Branches in SVN

The support is incomplete and fragile:

▶ renamed files are not merged correctly
▶ old clients will not update the merge info

From the SVN Book:

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

Using Mercurial
  Workflows
  Branching
  The Underlying Model
  Using History

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

Wrapping Up
**The Underlying Model**

A Mercurial changeset conceptually consist of:

- 0–2 parent changeset IDs:
  - root changeset has no parents
  - normal changesets have one parent
  - merge changesets have two parents

- date, username, commit message
- difference from first parent changeset
- changeset ID is computed as SHA-1 hash of the above
- makes it impossible to inject malicious code on server
Outline

Introduction

Using Mercurial
  Workflows
  Branching
  The Underlying Model
  Using History

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

Wrapping Up
BROWSING THE HISTORY OF A FILE

The `hg annotate` command is invaluable:

- you see when each line was introduced
- you can quickly jump back to earlier versions

History of Mercurial’s README file:

<table>
<thead>
<tr>
<th>3942: Basic install:</th>
</tr>
</thead>
<tbody>
<tr>
<td>445:</td>
</tr>
<tr>
<td>3942: $ make                # see install targets</td>
</tr>
<tr>
<td>3942: $ make install        # do a system-wide install</td>
</tr>
<tr>
<td>3942: $ hg debuginstall     # sanity-check setup</td>
</tr>
<tr>
<td>3942: $ hg                  # see help</td>
</tr>
<tr>
<td>0:</td>
</tr>
<tr>
<td># ...</td>
</tr>
</tbody>
</table>

Better interface in `hg serve`
Ever wondered when a function was introduced?

▶ *hg grep* can help you!

Example: When was *hg forget* introduced?

```bash
% hg grep --all 'def forget' commands.py
commands.py:8902:+:def forget(ui, repo, *pats, **opts):
commands.py:3522:-:def forget(ui, repo, *pats, **opts):
commands.py:814:-:def forget(ui, repo, file1, *files):
commands.py:814:+:def forget(ui, repo, *pats, **opts):
# ...
```
**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:
You’ve found a bug! When was it first introduced? Use `hg bisect` to mark good and bad revisions:
You’ve found a bug! When was it first introduced? Use `hg bisect` to mark good and bad revisions:
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:

```
good        good        bad
```

```
good        good
```

```
test       test
```

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

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

```
good     good     good
        ↓        ↓
bug!     bad
```

```
OUTLINE

INTRODUCTION

USING MERCUrial
  Workflows
  Branching
  The Underlying Model
  Using History

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

WRAPPING UP
Mercurial is Extensible

You can add new functionality to Mercurial:

▶ ships with 30+ extensions
▶ wiki lists 75+ extensions
▶ extensions can change basically everything
▶ helps to keep the core small and focused
Outline

Introduction

Using Mercurial
  Workflows
  Branching
  The Underlying Model
  Using History

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

Wrapping Up
Moving Changesets Around

Tired of all those merges? Use the rebase extension!

- Revision graph:

```
A --> B --> C
```
Moving Changesets Around

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

- Revision graph:

  A → B → C → D → E

Beware: public changes should never be rebased.
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:

\[ A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \]
\[ \rightarrow X \rightarrow Y \rightarrow Z \]

- Merge:

\[ A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \]
\[ \rightarrow M \]
\[ \rightarrow X \rightarrow Y \rightarrow Z \]
Moving Changesets Around

Tired of all those merges? Use the rebase extension!

- Revision graph:
  - A → B → C → D → E
    - X → Y → Z

- Merge:
  - A → B → C → D → E → M
    - X → Y → Z

- Rebase:
  - A → B → C → D → E
    - X → Y → Z → D′ → E′
Moving Changesets Around

Tired of all those merges? Use the rebase extension!

- Revision graph:

  Revision graph:

  - Merge:

  - Rebase:

- Beware: public changes should never be rebased.
**Editing History**

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

- reorder changesets:

  \[ A \rightarrow B \rightarrow C \quad \leadsto \quad \rightarrow A \rightarrow C' \rightarrow B' \\]

- fold changesets:

  \[ A \rightarrow B \rightarrow C \quad \rightarrow A \rightarrow BC \quad \rightarrow \]

- drop changesets:

  \[ A \rightarrow B \rightarrow C \quad \rightarrow A \rightarrow C' \quad \rightarrow \]

- edit changesets:

  \[ A \rightarrow B \rightarrow C \quad \rightarrow A \rightarrow X \rightarrow B' \rightarrow \]
Editing History

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

- reorder changesets:
  
  \[ A \rightarrow B \rightarrow C \]  
  \[ \sim \rightarrow \]  
  \[ A \rightarrow C' \rightarrow B' \]

- fold changesets:
  
  \[ A \rightarrow B \rightarrow C \]  
  \[ \sim \rightarrow \]  
  \[ A \rightarrow BC \]
Inspired by `git rebase -i`, `histedit` lets you

- reorder changesets:
  \[
  A \rightarrow B \rightarrow C \quad \xrightarrow{\sim} \quad A \rightarrow C' \rightarrow B'
  \]

- fold changesets:
  \[
  A \rightarrow B \rightarrow C \quad \xrightarrow{\sim} \quad A \rightarrow BC
  \]

- drop changesets:
  \[
  A \rightarrow B \rightarrow C \quad \xrightarrow{\sim} \quad A \rightarrow C'
  \]
Inspired by `git rebase -i`, `histedit` lets you

- reorder changesets:
  
  \[
  A \rightarrow B \rightarrow C \quad \leadsto \quad A \rightarrow C' \rightarrow B'
  \]

- fold changesets:
  
  \[
  A \rightarrow B \rightarrow C \quad \leadsto \quad A \rightarrow BC
  \]

- drop changesets:
  
  \[
  A \rightarrow B \rightarrow C \quad \leadsto \quad A \rightarrow C'
  \]

- edit changesets:
  
  \[
  A \rightarrow B \rightarrow C \quad \leadsto \quad A \rightarrow X \rightarrow B' \rightarrow C'
  \]
Outline

Introduction

Using Mercurial
  Workflows
  Branching
  The Underlying Model
  Using History

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

Wrapping Up
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 URL
▶ use `hg pull` to convert new SVN revisions
▶ use `hg push` to commit changesets to SVN server

Goal: make `hg` a better Subversion client than `svn`!
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
Outline

Introduction

Using Mercurial
  Workflows
  Branching
  The Underlying Model
  Using History

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

Wrapping Up

aragost Trifork
**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...
Outline

Introduction

Using Mercurial
- Workflows
- Branching
- The Underlying Model
- Using History

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

Wrapping Up
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/

- Getting Started:
  http://mercurial.aragost.com/kick-start/
  http://mercurial.ch/
  http://hginits.com/

- Some free Mercurial hosting sites:
  http://bitbucket.org/
  http://code.google.com/
  http://sourceforge.net/
  http://www.codeplex.com/
Contact

Please get in touch if you have more questions or have considered using Mercurial in your organization:

▶ Email: mg@aragost.com
▶ IRC: mg in #mercurial on irc.freenode.net
Mercurial Contributors

From http://ohloh.net/p/mercurial/map:

![Map of Mercurial contributors around the world]

Showing 50 of 325 contributors.
Mercurial Contributors

From http://ohloh.net/p/mercurial/map:

Thank you!