The Mercurial SCM

Fast.
Simple.
Distributed.

Bryan O'Sullivan
bos@serpentine.com

About this talk

- Why Google?
- Understanding Mercurial
- Working with other people
- Helping you to work efficiently
- Why Mercurial is fast
- Why your tools are important
- Where Mercurial is going

About Mercurial

- Work began in April 2005
  - Goal: manage Linux-sized trees efficiently
- Now 95% pure Python, 750 lines of C for speed
- Rapid uptake due to speed and usability:
  - **Linux**: video4linux, ALSA, e2fsprogs, ...
  - **System software**: Xen, OpenSolaris, Conary, FreeBSD ports, ...
  - **Other exciting projects**: One Laptop Per Child, Moin Moin, microformats, physics textbooks, ...

Revision control: a crowded field

Accurev

BitKeeper *

CCC/Harvest

ClearCase *

Perforce *

StarTeam *

Surround

Vault

...

Aegis

Arch

Bazaar-NG *

CVS *

Darcs *

git *

PRCS

Subversion *

SVK

Vesta

...

A developer's POV

- I have work to do!
- I want something *simple* that *works*
- My SCM tool should:
  1. Be easy to understand
  2. Help me to work with others
  3. Let me work efficiently

A developer's POV

- My SCM tool should:
  1. ***Be easy to understand***
  2. Help me to work with others
  3. Let me work efficiently

Be easy to understand

- **User quote:** “Mercurial's conceptual model is clean and simple enough to carry around in my head”

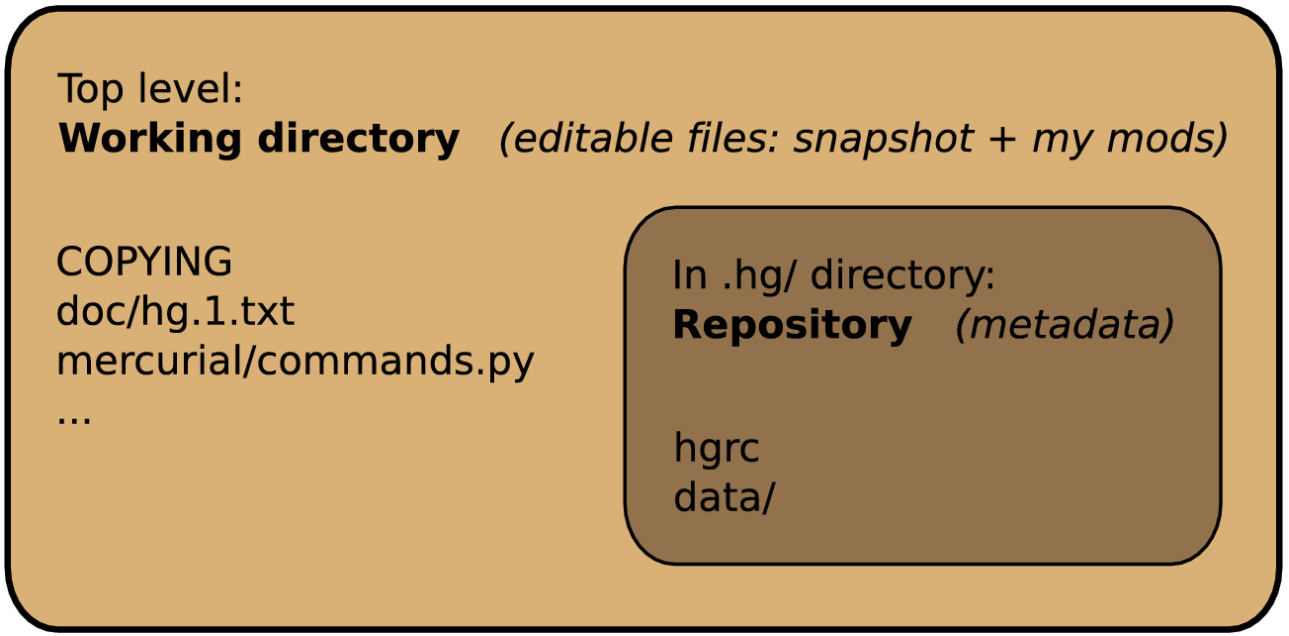
- Let's introduce three concepts:
  - Repository
  - Working directory
  - Changeset

What's a repository?

- Simple
  - A directory containing the history of my project
  - No fancy database, no big server: just a directory
- Lightweight
  - Making a copy (a “clone”) of a repository is cheap
- Everywhere
  - All work happens in repositories
  - Every person works in their own repositories

What's a working directory?

- A snapshot of my work as of some revision
- All files are modifiable (no “hg edit” command)
- My modifications will be saved when I commit



What's *in* a repository?

Mercurial doesn't actually expose these details.

(But they're simple, and it helps to know what's going on.)

- Changelog
  - The history of changes to the repository
- Manifest
  - History of file versions used in each changeset
- Per-file data
  - History of every file that Mercurial tracks

Contrast the repository models

	Traditional SCM	**Mercurial**
Central repo	Exactly one	As many as needed
Bottlenecks	Central server	None
Load mgmt	Expensive or impossible	Mirrors wherever, for free
Distant users	Slow server response	Fast local response
Server failure	Catastrophic	Full backup in every repo
Network connection	Always needed	Fully productive anywhere

What's a changeset?

- A snapshot of the project at a point in time
- It records:
  - Who made the change (the “committer”)
  - A readable description of the change
  - What files were changed, what the changes were
  - What the parent changeset was
- Creating a changeset is called “committing”i t

Micro-tutorial: Hg in 60 seconds

1. Create a repository	`hg init myrepo`
2. Go in there	`cd myrepo`
• Edit a file	`emacs myfile`
3. Tell hg to track the file	`hg add myfile`
4. Now what's happening?	`hg status`
• “File has been added”	*`A myfile`*
5. Record my changes	`hg commit`

A developer's POV

- My SCM tool should:
  1. Be easy to understand
  2. ***Help me to work with others***
  3. Let me work efficiently

Parallel play

- People naturally work in parallel
  - Most revision control tools make this *hard*
- I make some changes
- I go to check them in ("commit" them)
- What if someone else committed first?

I have commitment issues

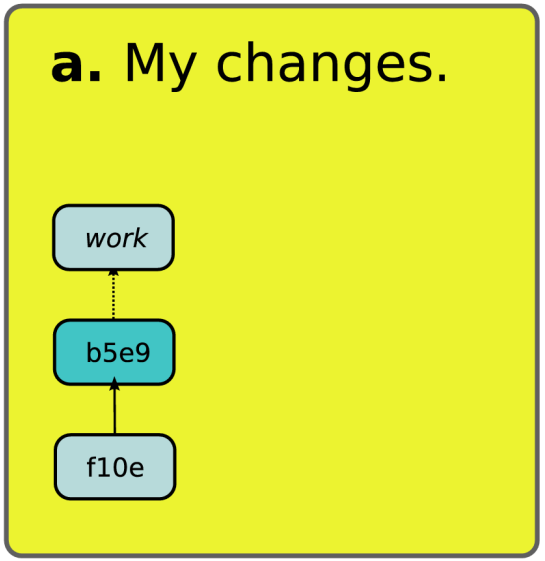
- What if someone else committed before me?
  - Often, I must merge *before* I can commit
- **No permanent record** of my changes yet
  - A mistake during merge can **lose my work**
- *"But this is a branch management policy issue!"*
  - With many tools, default policy is **not safe**

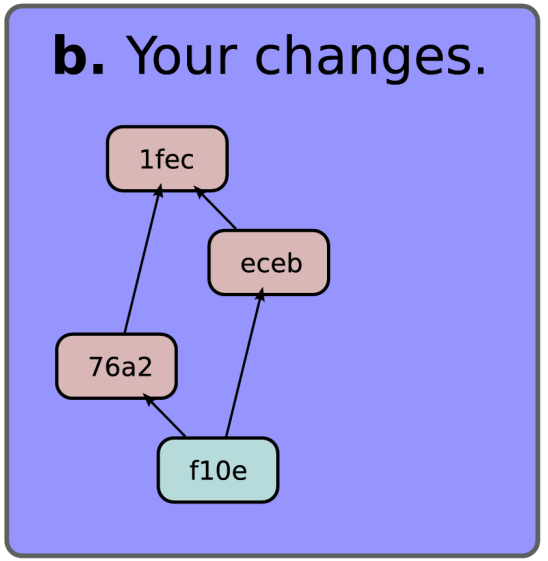
The Hg model: branching

- Remember that a changeset has a parent?
- Two changesets with the same parent make a *branch*
- That's *all* a branch is!
  - Nothing dramatic or complex

Merging in action: 1

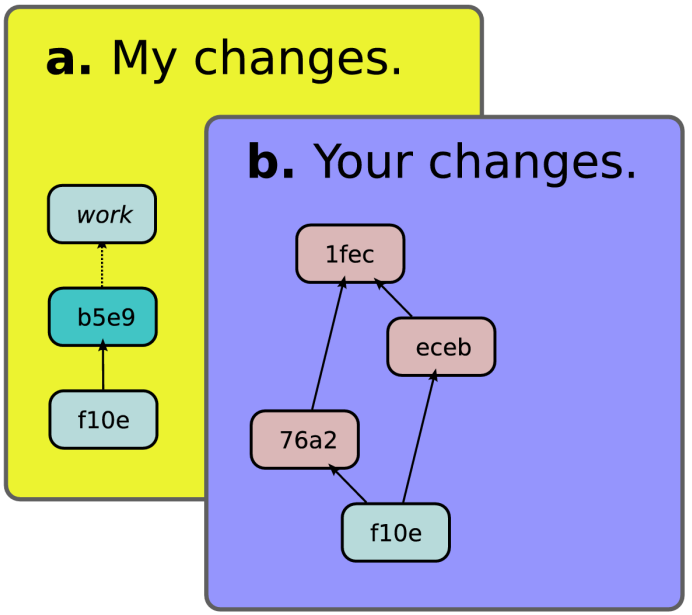
- We have two repositories
- Our changes have a common parent
- My working dir is based on change b5e9

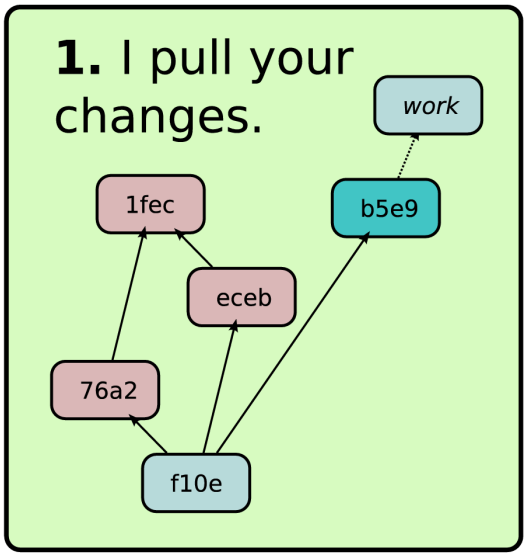




Merging in action: 2

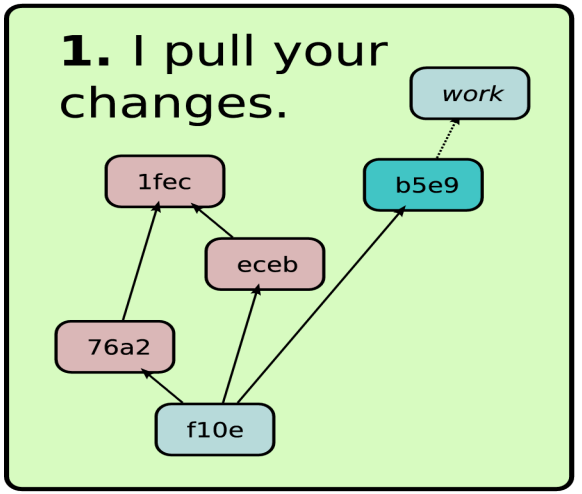
- I fetch your changes using “hg pull”
- My repo now contains both our changes, history
- The pull has *not* affected the working dir

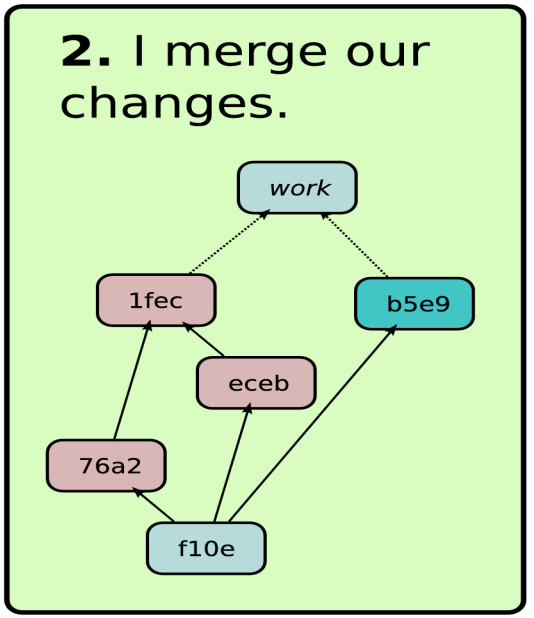




Merging in action: 3

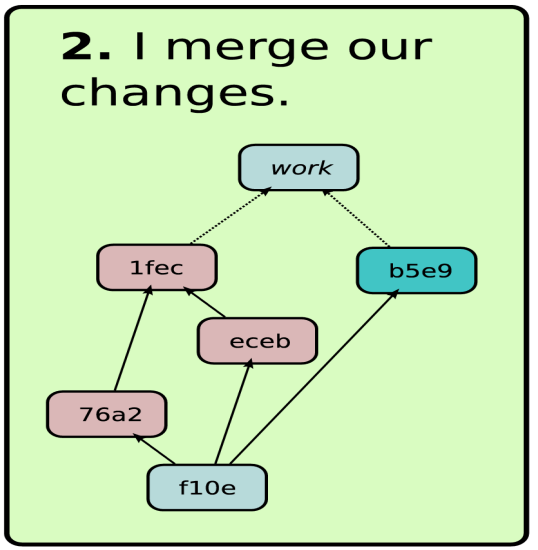
- I run “hg merge”
- The working dir now has two parents
- Contents reflect both my changes and yours

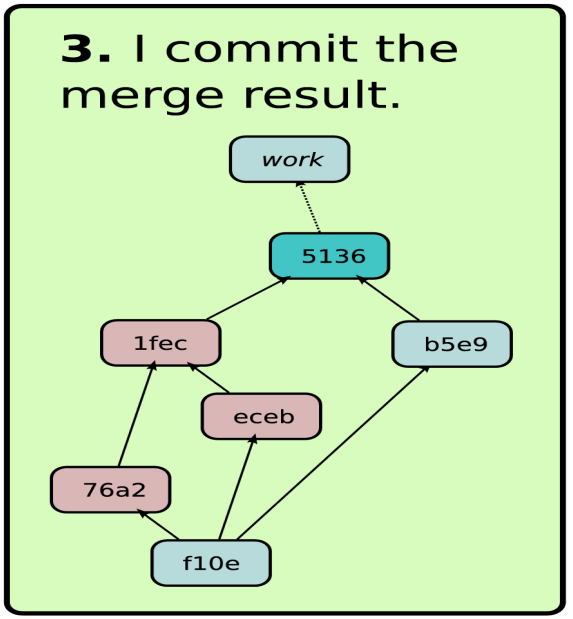




Merging in action: 4

- When I commit, new change has the same parents as working dir had & same contents
- Working dir now has new change as parent





Merging without stress

- What if I make a mistake during a merge?
- My changes are still there; so are yours
- **No work gets lost**
- I simply redo the merge

Sharing is easy

- Built-in web server
  - CGI server for Apache integration
- Use ssh for secure remote access
  - http/https RO now, http**s** RW soon
- Works over network filesystems
- Share work offline using email, USB flash, ...

Sharing is symmetric

- I **clone** a remote repo to get a local copy
- I **pull** new changes from a remote repo
- I **push** my changes to another repo
- After a push, the remote repo is identical to mine

A developer's POV

- My SCM tool should:
  1. Be easy to understand
  2. Help me to work with others
  3. ***Let me work efficiently***

Case study: performance

- `log libsvn_wc/props.c` (~200 revs)
  - Mercurial: 1.2 seconds
  - Subversion: 1.2 seconds — but up to 11 seconds with network burps
- `annotate libsvn_wc/props.c`
  - Mercurial: 1.0 seconds
  - Subversion: 2.4 seconds

- Distributed operation can be fast, lean
  - And Python needn't be slow, either

Case study: space usage

- Unscientific tests on 3-year-old laptop
- Imported Subversion trunk into Mercurial
  - Head is ~1250 files in 25MB
- Subversion working copy: 72MB
- Mercurial repo: 76MB
  - Includes all 15,000 revs and working dir
- Comparable numbers for Linux kernel
  - Plain files 263MB, 28K revs + working dir 556MB

Mercurial makes me more efficient

- Simple concepts let me focus
  - Think less about SCM, more about work
  - *“My tea is still warm”* — no long waits, distraction
- Commits and merges are separate
  - Harder to lose or corrupt work by accident
- Cheap repos let me sandbox my work
  - One repo per task
- Local data lets me work anywhere

Why is Mercurial fast?

- Simple file formats, easy to parse in Python
- O(1) fulltext reconstruction
- Avoid seeks: group related data
  - Splitting revs across files is bad
  - Prefer longer linear reads
  - Access files in consistent, useful order
- Don't read() when you can just stat()
- Network used only when explicitly requested

Mercurial Queues

- Novel approach to patch management
  - Inspired by quilt, but *integrated* (and faster)
- Stack of patches: some applied, some not
- Scaling: ~1550 “mm”p atches atop Linux kernel
  - Push 7 patches/sec on my crufty old laptop
- Applied patches show up as changesets
  - Normal revision control tools work!
  - Use e.g. “annotate” and “bisect” to find bad patches

Mercurial Queues workflow

- Edit the top applied patch, then refresh it
- Got a new subtask? Push a new patch
- Want upstream changes?
  - Pop the stack, pull upstream changes, repush stack
- Conflicts with upstream changes?
  - No problem! Merge quickly with tool's help
- Prototype and refactor freely without leaving history of dead ends, false starts

Why your choice of tools matters

- Your tools shape how *you* work
  - RCS/SCCS: I have to be logged into a host to work
  - CVS/SVN: I have to be online to work
  - Distributed tools: I can work **anywhere, any time**
- Your tools shape how *others* work with you
  - RCS/SCCS: outsiders can't see history
  - CVS/SVN: outsiders can read, but not write
  - Distributed tools: there are **no outsiders**

Distributed tools and Free Software

- Choose the development model you like best
- Every user becomes a potential contributor
  - Mercurial often gets changesets from “outsiders” where a traditional patch would not apply cleanly
- Reconcile more easily after forks

Coming attractions*

- In progress:
  - Support for merging changes across renames
  - Comprehensive user manual
  - Eclipse support
- Want to add:
  - Better GUI interfaces
  - Integration with other popular IDEs

* Based on April 2006 User Survey

User survey quotes: the team

- “The developers are super-helpful.”
- “The Mercurial community is more polite and helpful than most.”
- **“The community is great around Mercurial.”**

User survey quotes: the software

- “I was up and ready to go with Mercurial in less than 5 minutes.”
- “Mercurial was extraordinarily easy to learn.”
- “I used to like CVS a lot. I can't imagine going back. Really.”
- Wow! — “I consider Mercurial the best version control system on earth.”

Thank you!