Source Control
Apr 13, 2009

Dave Eckhardt
Roger Dannenberg
Zach Anderson (S '03)
Agenda

- Malloc
- Source control
Mallocate questions

• “How many buckets should my seglist have?”
• “What should the bucket size limits be?”
  - Factors to be considered:
    - The traces
    - What else?

• Partnership late day policy
  - Alice has 3 late days, Bob has 1
    - They can submit at most 1 day late!
Malloclab questions

- **Private traces – what are they?**
  - Equally as difficult
  - Prevent “coding to the traces”
    - if (size == 448) size = 512;
  - Will we ever get to see them?
    - What does “private” mean?
  - How do I know how well I do?
    - Email. Score is out of 30 points.
Malloclab questions

- **Private traces – what are they?**
  - Not fair!
  - “I want to see every sequence of data my code will run!”
    - Consider glibc malloc...
Source Control - Outline

Motivation
Repository vs. Working Directory
Conflicts and Merging
Branching
Goals

Working together should be easy

Time travel
- Useful for challenging patents
- *Very* useful for reverting from a sleepless hack session

Parallel universes
- Experimental universes
- Product-support universes
Goal: Shared Workspace

Reduce development latency via parallelism

– [But: Brooks, Mythical Man-Month]
Goal: Time Travel

Retrieving old versions should be easy.

Once Upon A Time...

Alice: What happened to the code? It doesn’t work.

Charlie: Oh, I made some changes. My code is 1337!

Alice: Rawr! I want the code from last Tuesday!
Goal: Parallel Universes

Safe process for implementing new features.

- Develop bell in one universe
- Develop whistle in another
- Don't inflict B's core dumps on W
- Eventually produce bell-and-whistle release
How?

*Keep a global repository for the project.*
The Repository

Version / Revision / Configuration
- Contents of some files at a particular point in time
- aka “Snapshot”

Project
- A “sequence” of versions
  • (not really)

Repository
- Directory where projects are stored
The Repository

Stored in group-accessible location
- Old way: file system
- Modern way: “repository server”

Versions in repository visible group-wide
- Whoever has read access
- “Commit access” often separate
How?

Keep a global repository for the project.

*Each user keeps a working directory.*
The Working Directory

Many names ("sandbox")
Where revisions happen
Typically belongs to one user
Versions are checked out to here
New versions are checked in from here
How?

Keep a global repository for the project.
Each user keeps a working directory.

*Concepts of checking out, and checking in*
Checking Out. Checking In.

Checking out

- A version is copied from the repository
  - Typically “Check out the latest”
  - Or: “Revision 3.1.4”, “Yesterday noon”

Work

- Edit, add, remove, rename files

Checking in

- Working directory ⇒ repository *atomically*
- Result: new version
Checking Out. Checking In.

Repository

○

○

v0.1

check out

Working Directory

v0.1 copy
Checking Out. Checking In.

Repository

○
○

v0.1

Working Directory

v0.1 copy

mutate

v0.1++
Checking Out. Checking In.

Repository

○

○

○

| v0.1 |

Working Directory

| v0.2 | v0.1++ |

check in
How?

Keep a global repository for the project.
Each user keeps a working directory.
Concepts of checking out, and checking in

Mechanisms for merging
Conflicts and Merging

Two people check out.

- Both modify foo.c

Each wants to check in a new version.

- Whose is the correct new version?
Conflicts and Merging

Conflict
- Independent changes which “overlap”
- *Textual* overlap detected by revision control
- *Semantic* conflict cannot be

Merge displays conflicting updates per file
Pick which code goes into the new version
- A, B, NOTA
Alice Begins Work

Alice

v0.2

copy

v0.2

fix b#1

Repository

v0.2

Bob
Bob Arrives, Checks Out

Alice
v0.2
copy

Repository
v0.2

Bob
v0.2
copy

v0.2
fix b#1
Alice Commits, Bob Has Coffee

Alice

v0.2

copy

v0.2

fix b#1

Repository

v0.2

Bob

v0.2

copy

v0.3
Bob Fixes Something Too

Alice
- v0.2
- copy
- v0.2
  - fix b#1

Repository
- v0.2
  - v0.2
  - v0.3

Bob
- v0.2
  - copy
  - v0.2
  - fix b#7
Wrong Outcome

Alice
v0.2
copy
v0.2
fix b#1

Repository
v0.2
v0.3

Bob
v0.2
copy
v0.2
fix b#7
“Arguably Less Wrong”

Alice
v0.2 copy
v0.2 fix b#1

Repository
v0.2
v0.3

Bob
v0.2 copy
v0.2 fix b#7

v0.4
Merge, Bob, Merge!

Alice
v0.2
 copy
v0.2
 fix b#1

Repository
v0.2

Bob
v0.2
 copy
v0.2
 fix b#1

v0.3

v0.2
 fix b#7

fix b#7
fix b#1
Committing Genuine Progress

Alice
- v0.2 copy
- v0.2 fix b#1

Repository
- v0.2
- v0.3

Bob
- v0.2 copy
- v0.2 fix b#7
- fix b#1
- fix b#7

v0.4
How?

Keep a global repository for the project.
Each user keeps a working directory.

Concepts of *checking out*, and *checking in*

Mechanisms for *merging*

*Mechanisms for branching*
Branching

A branch is a sequence of versions
   -(not really...)

Changes on one branch don't affect others

Project may contain many branches

Why branch?
   - Implement a new “major” feature
   - Begin a temporary independent sequence of development
Branching

The actual branching and merging take place in a particular user's working directory, but this is what such a sequence would look like to the repository.
Branch Life Cycle

“The Trunk”
- “Release 1.0”, “Release 2.0”, ...

Release 1.0 maintenance branch
- After 1.0: 1.0.1, 1.0.2, ...
- Bug-fix updates as long as 1.0 has users

Internal development branches
- 1.1.1, 1.1.2, ...
- Probably 1.1.1.client, 1.1.1.server
Source Control Opinions

**CVS**
- very widely used
- mature, lots of features
- default behavior often wrong

**SubVersion (svn)**
- SVN > CVS (design)
- SVN > CVS (size)
- Doesn't work in AFS
- Yes, it does
- No, it doesn't?

**Perforce**
- commercial
- reasonable design
- works well
- big server

**BitKeeper**
- Favored by Linus Torvalds
- “Special” license restrictions

**git**
- Favored by Linus
Source Control Opinions

Others

- Mercurial (“hg”)
  - Merge-once branches
- Bazaar (“bzr”)
- Monotone
- arch
- Darcs (“patch algebra”)

Generally

- Promising plans
- Ready yet?