15-410

"...Goals: Time Travel, Parallel Universes..."

PRCS
Feb. 4, 2004

Dave Eckhardt

Bruce Maggs

Zach Anderson (S '03)

Outline

Motivation

Repository vs. Working Directory

Conflicts and Merging

Branching

PRCS – Project Revision Control System

Goals

Working together should be easy

Time travel

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

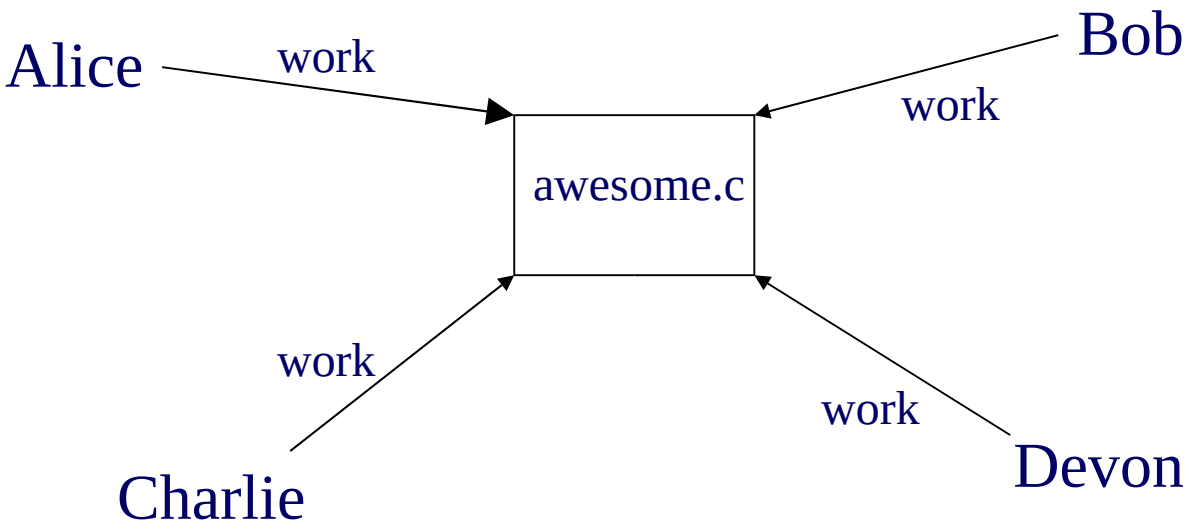
Parallel universes

- 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

- 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 to whole group

"Commit access" often a separate privilege

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”, “Yesterday noon”

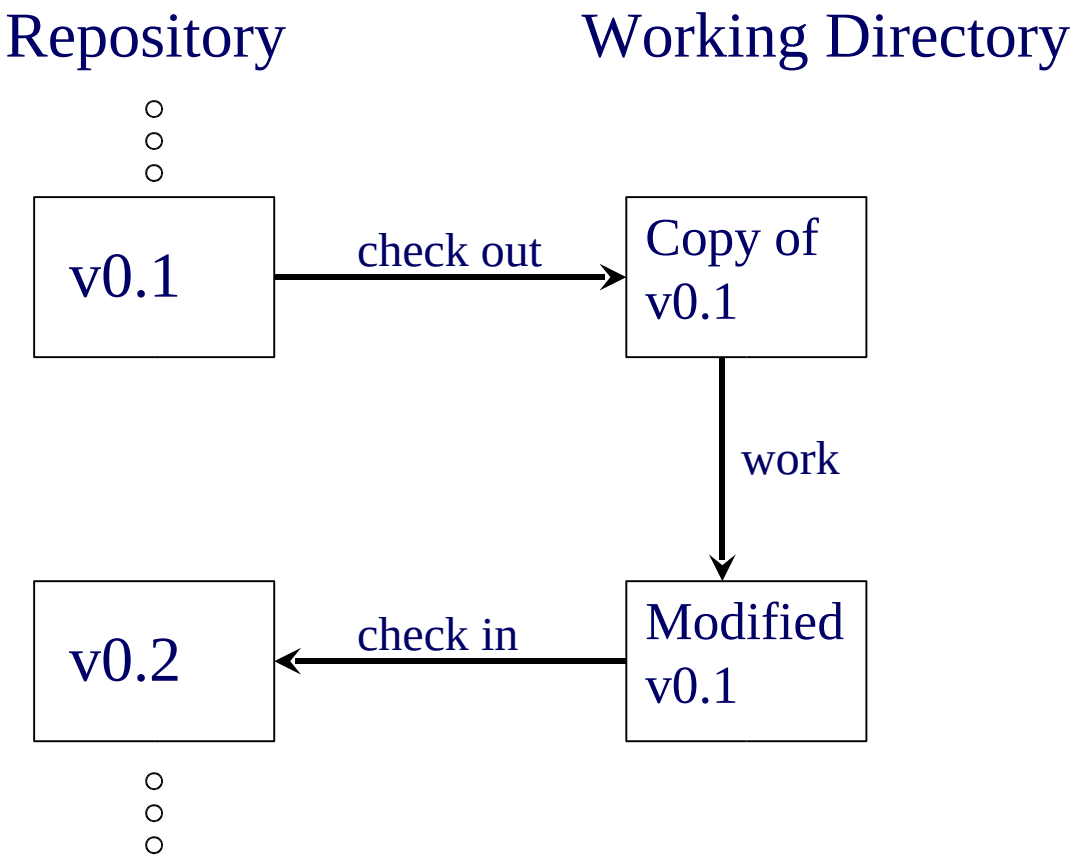
Work

- Edit, add, remove, rename files

Checking in

- Working directory *atomically* $\Rightarrow$ repository
- Result: new version

Checking Out. Checking 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

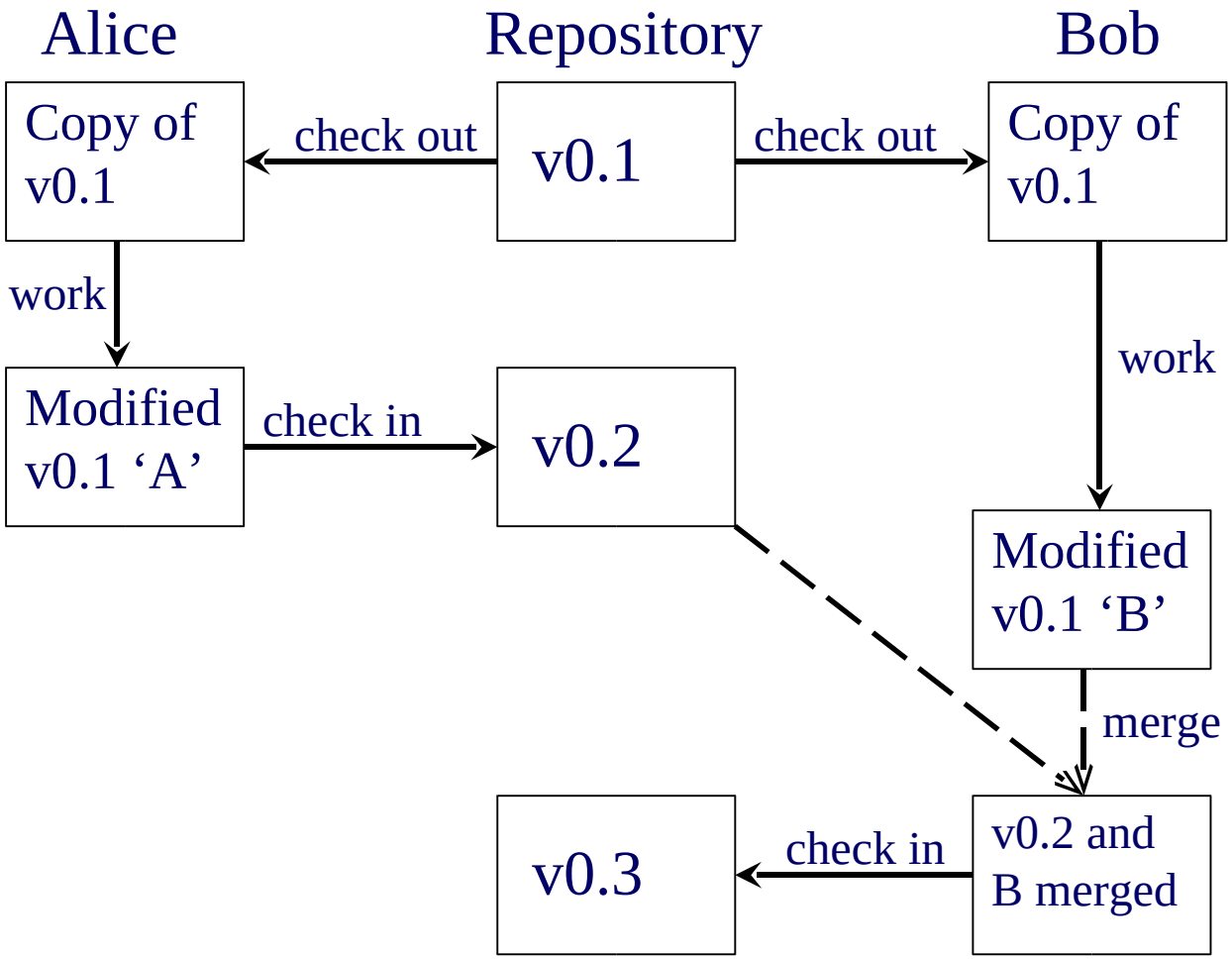
Merge displays conflicting updates per file

Pick which code goes into the new version

- A, B, NOTA

Picture now, example later

Conflicts and Merging



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...)**

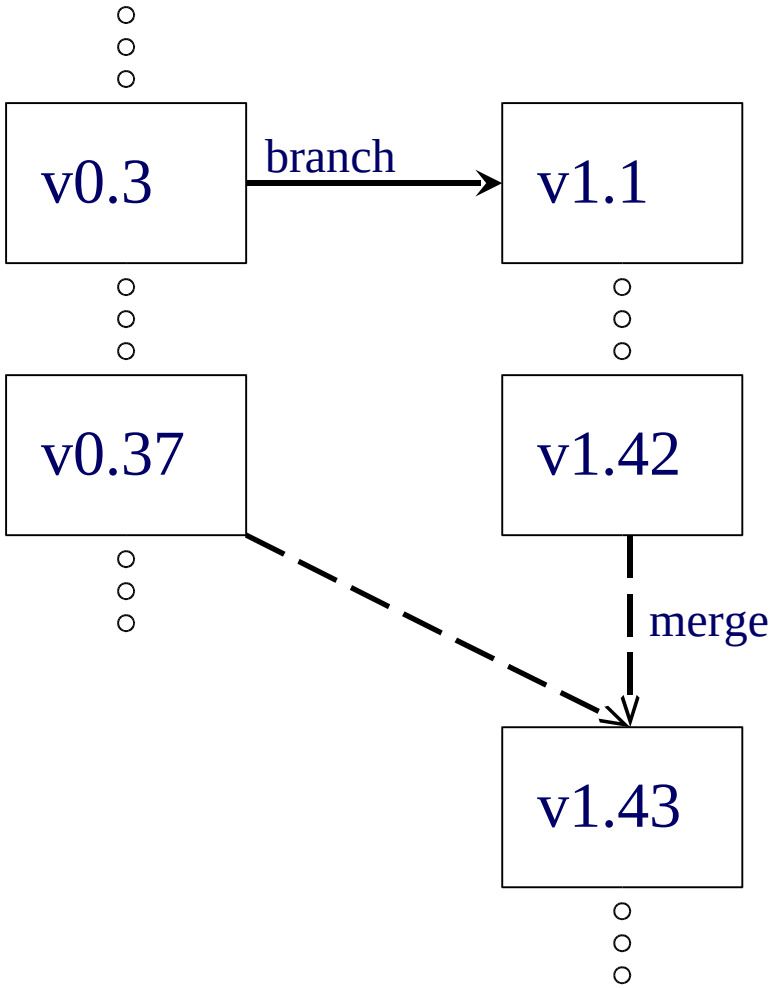
Changes on one branch don't affect others

Project may contain many branches

Why branch?

- **Implement a new “major” feature**
- **Begin an 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

- **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**

Branch Life Cycle

Successful development branch

- Merged back to parent
- No further versions

Unsuccessful development branch

- Some changes pulled out?
- No further versions

Maintenance branch

- “End of Life”: No further versions

Are Branches *Deleted?*

Recall PRCS "data structure"

- Revisions of each file (coded as deltas)
- Revisions of the directory tree

Branch delete

- *Complicated* data structure update
  - [Not a well-tested code path]
- Generally a bad idea
  - History could *always* be useful later...

Source Control Opinions

CVS

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

OpenCM

- security-conscious design
- not widely used

BitKeeper

- Favored by Linus Torvalds
- “Special” license restrictions

SubVersion

- lots of potential
- not ready yet?

PerForce

- commercial
- reasonable design
- works well
- big server

Dave's Raves

CVS

- Commit: atomic if you are careful
- Named snapshots: if you are careful
- Branching: works if you are careful
- *Core operations* require care & expertise!!!

Many commercial products

- Require full-time person, huge machine
- Punitive click-click-click GUI
- Poor understanding of data structure requirements

Recommendation for 15-410

PRCS, Project Revision Control System

- Small “conceptual throw weight”
- Easy to use, state is visible (single text file)
- No bells & whistles

Setting to learn revision control *concepts*

- Quick start when joining research project/job
  - (They will probably not be using PRCS)

Getting Started

Add 410 programs to your path (in bash):

```
$export
  PATH=/afs/cs.cmu.edu/academic/class/1541
  0-s04/bin:$PATH
```

Set environment variables

```
$export
  PRCS_REPOSITORY=/afs/cs.cmu.edu/academic
  /class/15410-s04-users/group-
  99/REPOSITORY
$export PRCS_LOGQUERY=1
```

Creating A New Project

In a working directory:

```
$prcs checkout P
```

- **P is the name of the project**

Creates a file: P.prj

The Project File

```
;; -*- Prcs -*-
(Created-By-Prcs-Version 1 3 0)
(Project-Description "")
(Project-Version P 0 0)
(Parent-Version -*- -*- -*-)
(Version-Log "Empty project.")
(New-Version-Log "")
(Checkin-Time "Wed, 15 Jan 2003 21:38:47 -0500")
(Checkin-Login zra)
(Populate-Ignore ())
(Project-Keywords)
(Files
;; This is a comment.  Fill in files here.
;; For example:  (prcs/checkout.cc ())
)
(Merge-Parents)
(New-Merge-Parents)
```

Description of project.

Make notes about changes before checking in a new version

List of files

Using the Project File

Adding Files

```
$prcs populate P file1 file2 … fileN
```

- To add *every* file in a directory

```
$prcs populate P
```

Removing, renaming files

- See handout

Checking In

Checking in

```
$prcs checkin P
```

- **check in will fail if there are conflicts.**

Conflicts and Merging

Suppose this file is in the repository for project P:

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
        printf("Hello World!\n");
        return 0;
}
```

Conflicts and Merging

Suppose Alice and Charlie check out this version, and make changes:

Alice's Changes

```
#include <stdlib.h>
#include <stdio.h>

#define SUPER 0

int main(void)
{
        /* prints "Hello World"
           to stdout */
        printf("Hello World!\n");
        return SUPER;
}
```

Charlie's Changes

```
#include <stdlib.h>
#include <stdio.h>

int main(void)
{
        /* this, like, says
           hello, and stuff */
        printf("Hello Hercules!\n");
        return 42;
}
```

Conflicts and Merging

Suppose Alice checks in first.

If Charlie wants to check in he must perform a merge

```
$prcs merge
```

- **The default merge option performs a CVS-like merge.**

Conflicts and Merging

The file after a merge

```
#include <stdlib.h>
#include <stdio.h>

#define SUPER 0

int main(void)
{
<<<<<<< 0.2(w)/hello.c Wed, 19 Feb 2003 21:26:36 -0500 zra (P/0_hello.c 1.2 644)
        /* this, like, says hello, and stuff */
        printf("Hello Hercules!");
        return 42;
=======
        /* prints "Hello World" to stdout */
        printf("Hello World!");
        return SUPER;
>>>>>>> 0.3/hello.c Wed, 19 Feb 2003 21:36:53 -0500 zra (P/0_hello.c 1.3 644)
}
```

Conflicts and Merging

Pick/create the desired version

- **Check that into the repository.**

Branching

To create the first version of a new branch:

```
$prcs checkin -rWednesday P
```

To merge with branch X version 37:

```
$prcs merge -rX.37 P
```

Information

To get a version summary about P:

```
$prcs info P
```

- **with version logs:**

```
$prcs info -l P
```

Suggestions

Develop a convention for naming revisions

- Date
- Type of revision(bug-fix, commenting, etc.)
- Short phrase

When to branch?

- Bug fixing?
  - Check out, fix, check in to same branch
- Trying COW fork since regular fork works?
  - Branching probably a good idea.

Summary

We can now:

- Create projects
- Check source in/out
- Merge, and
- Branch

See PRCS documentation:

- Complete list of commands
- Useful options for each command.