# Source Control

Zach Anderson
zra@andrew.cmu.edu

# Outline

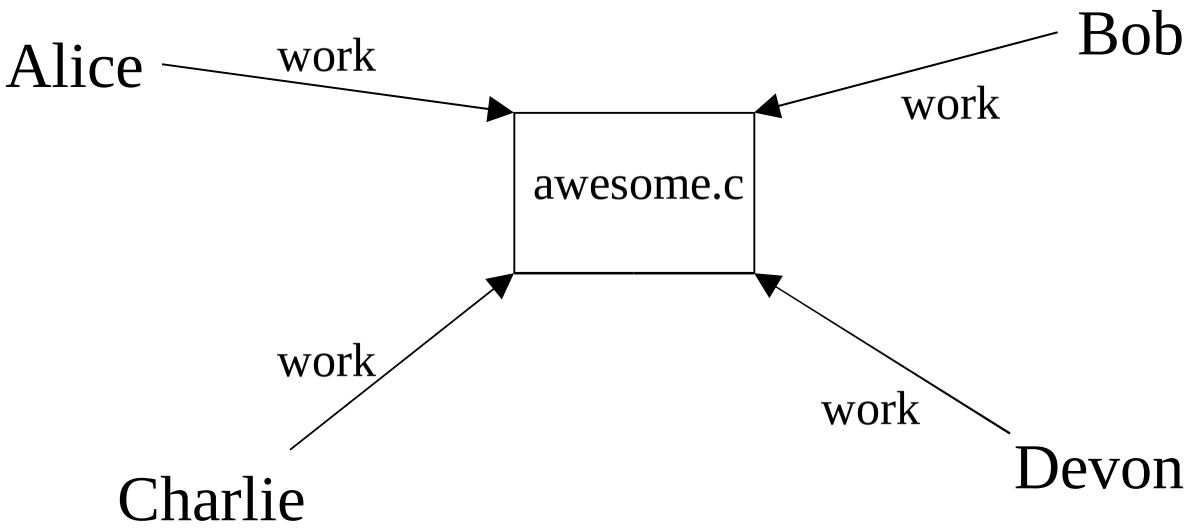
- Motivation
- Repository vs. Working Directory
- Conflicts and Merging
- Branching
- The Project Revision Control System

# What do we want?

- Working together should be easy.
- Going back in time.
- New features, and parallel universes.

# What do we want?

- A group of people should be able to work on the same project at the same time without difficulty.



# What do we want?

- 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!

# What do we want?

- There should be a safe process for implementing new features.
- The development of bell B, and unrelated whistle W should not interfere with each other.

# How?

- *Keep a global repository for the project.*

# The Repository

- Version
  - The state of a set of files at a particular point in time.
- Project
  - A sequence of versions.
- Repository
  - The directory where projects are stored.

# The Repository

- The repository is in some well known location.
- Versions in the repository are visible to all users who work on a particular project.

# How?

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

# The Working Directory

- This is where revision takes place.
- It may belong to a particular user.
- Versions are *checked out* to here.
- New versions are *checked in* from here.

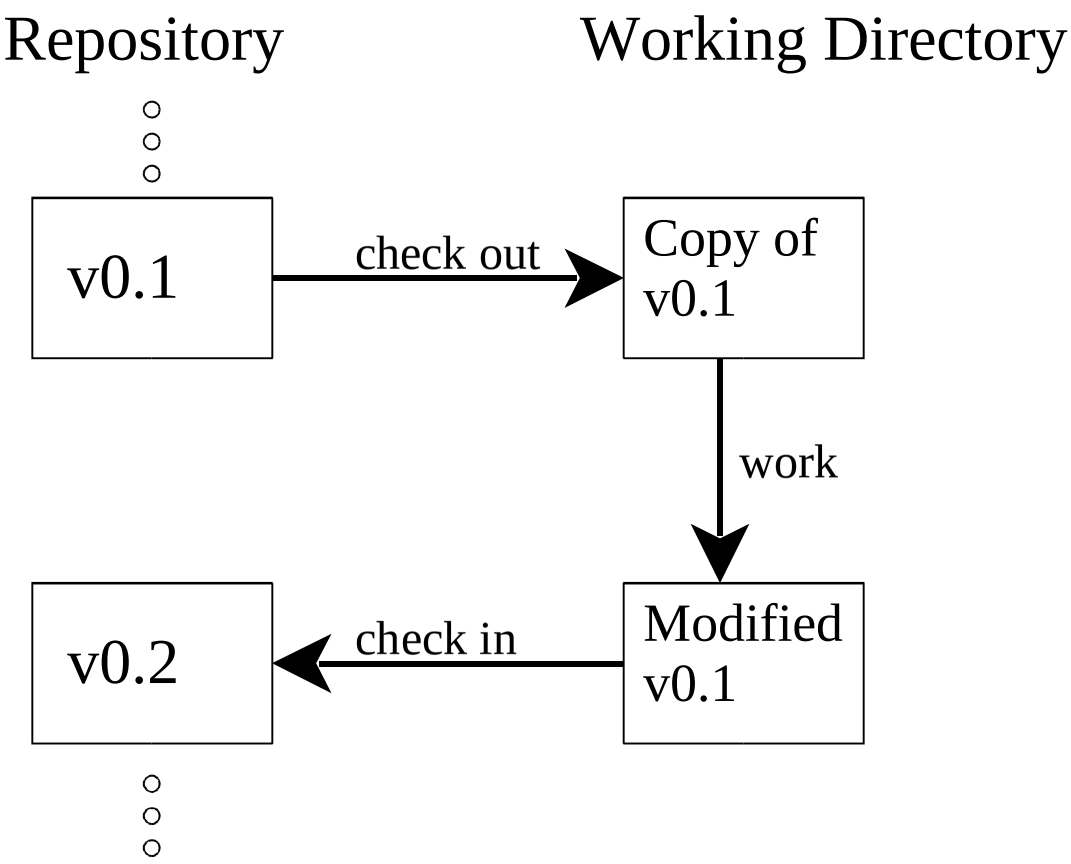
# How?

- 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.
- Work
  - edit, add, remove, rename files.
- Checking in
  - contents of working directory copied to repository.
  - 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**.*

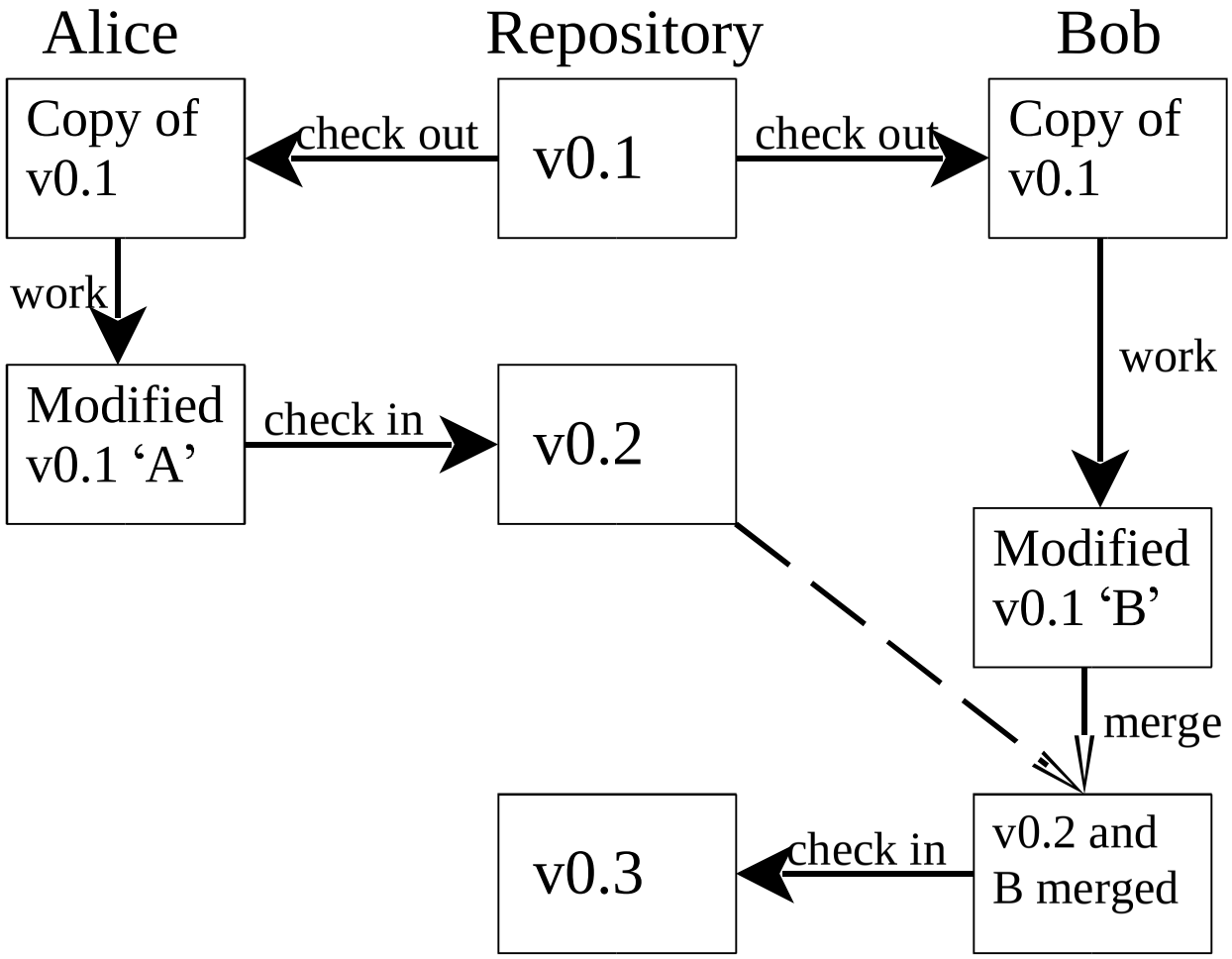
# Conflicts and Merging

- Two people check out.
- They modify intersecting sets of files.
- They each want to check in a new version.
- Whose is the correct new version?

# Conflicts and Merging

- A merge highlights differences between versions of the same files.
- Pick which code goes into the new version.
- Diagram first. Concrete example a little later.

# Conflicts and Merging



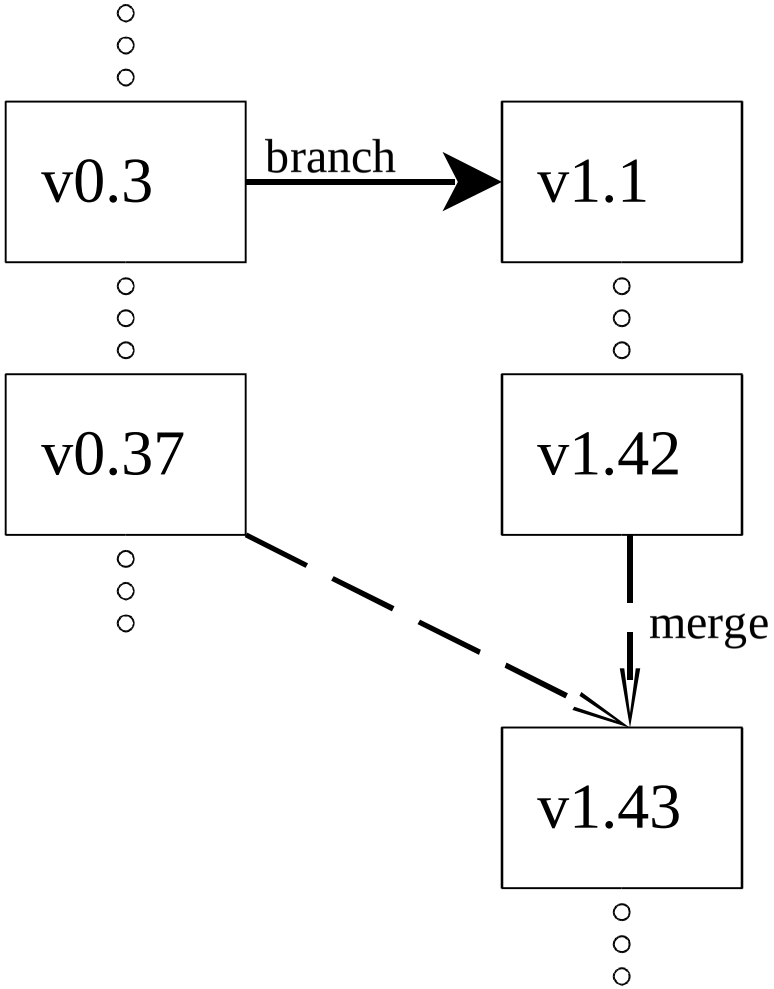
# How?

- 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.
- One project may contain several branches.
- Why branch?
  - Implementing new major features.
  - Beginning an independent sequence of developments.
- Merging branches is useful.

# 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.

# Source Control Software

- CVS
  - very widely used
  - mature, lots of features
  - branching awkward
- OpenCM
  - security conscience design
  - not widely used
- BitKeeper
  - Favored by Linus Torvalds
  - Unusual licensing restrictions
- SubVersion
  - lots of potential
  - not ready yet
- PerForce
  - commercial
  - conceptually reasonable design
  - good reputation

# Source Control Software

- We recommend PRCS (for 15−412).
  - Small size
  - Easy to use
  - Straightforward organization
- Opportunity to learn revision control concepts
  - Quick start when joining research project/job
  - They will probably not be using PRCS

# Getting Started

- Add the location of prcs to your path(in bash):
  - `$export PATH=/afs/andrew/scs/cs/15-412b/prcs/bin:$PATH`
- Create a directory for the repository.
  - `$mkdir /afs/cs.cmu.edu/academic/class/15412-s03-users/group-99/PRCS_FILES`
- Set the environment variable PRCS_REPOSITORY to that directory.
  - `$export PRCS_REPOSITORY=/afs/cs.cmu.edu/academic/class/15412-s03-users/group-99/PRCS_FILES`

# 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 Files
  - Remove the file from the list of files in P.prj.
- Renaming Files
  - Rename the file. Change the file name in the entry in P.prj

# 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!");
        return 0;
}
```

# Conflicts and Merging

- Suppose that 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!");
        return SUPER;
}
```

**Charlie's Changes**

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

int main(void)
{
        /* this, like, says
           hello, and stuff */
        printf("Hello Hercules!");
        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

- The idea, now, is to pick the desired version, and check that into the repository.

# Branching

- To create the first version of a new branch:
  - `$prcs checkin -rNewBranch P`
- To merge with branch X version 37:
  - `$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
  - the date
  - type of revision(bug−fix, commenting, etc.)
  - short phrase change summary
- When to branch?
  - Bug fixing? Probably not. Check out, fix, check in to same branch should be sufficient.
  - Trying a different approach? Implementing a new feature? Attempting COW Fork after 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.