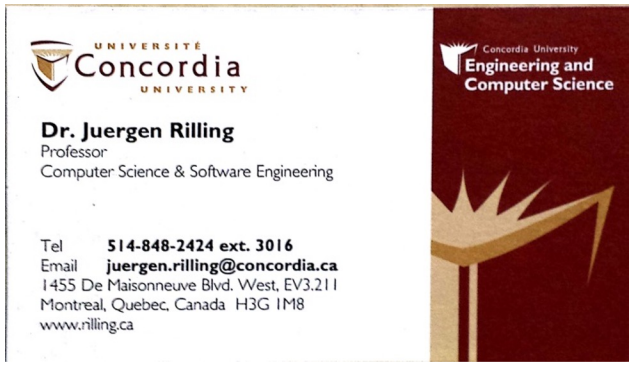
SOEN 6431
SOFTWARE MAINTENANCE AND Program Comprehension

Dr. Juergen Rilling



Week 3
Issue Trackers and Versioning Systems

The last 15 years: Globalization of the software Industry

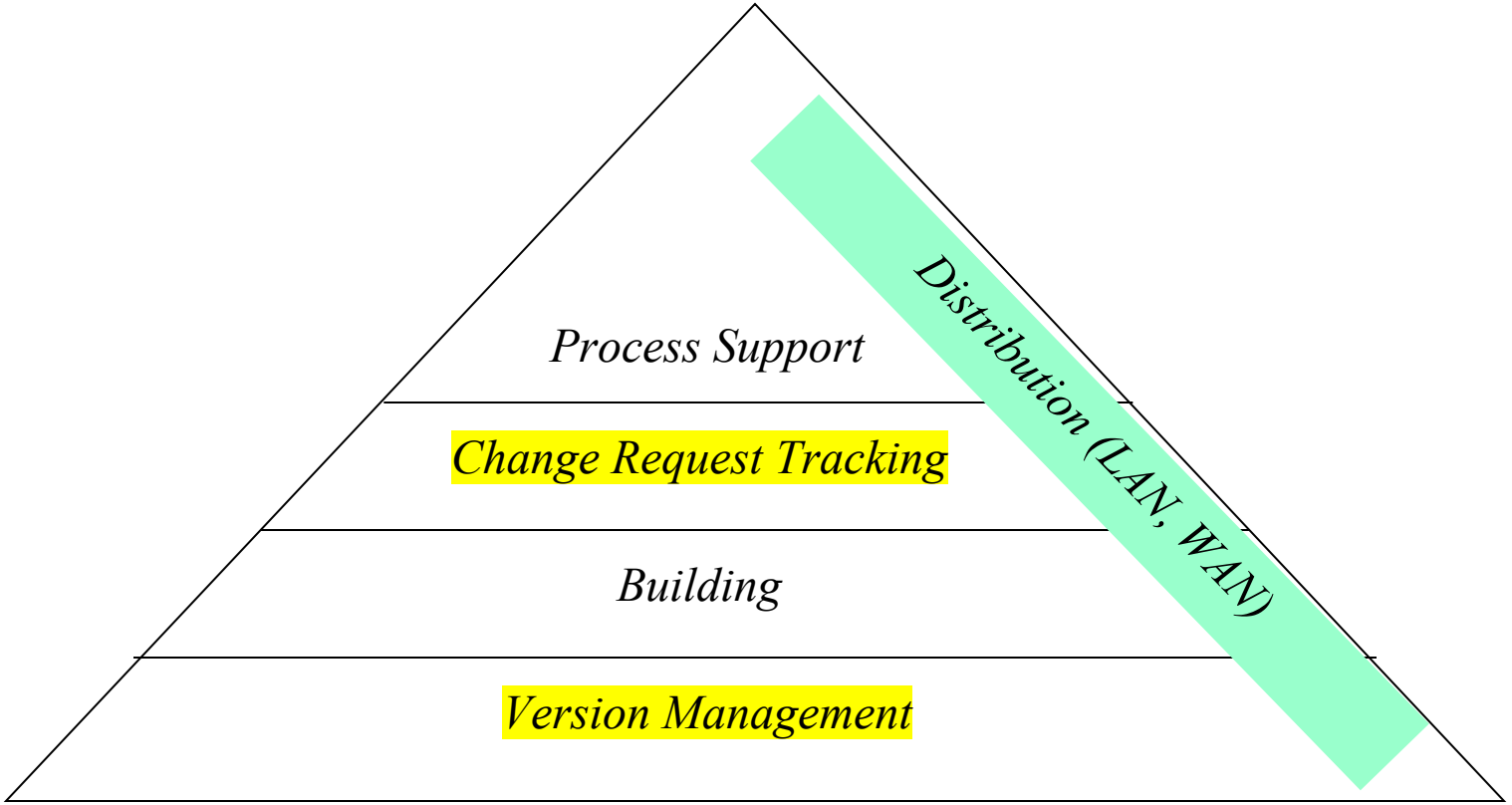
became standard practice:

- Software development adopts to cultural and social changes caused by
  - Introduction of the Internet as an enabling technology, with software being shared and published on the Internet.
  - Heterogeneous software ecosystems, multi programming languages, operating systems, hardware were created.
  - Outsourcing of parts of the software development/testing
  - New software development process, and technologies being available, e.g. open source movement, e-mail, wikis, etc.
- Knowledge resources (people and artifacts) are no longer available "on-site" – rather distributed across organization boundaries.

Software repositories

- Enabling technology to facilitate
  - Collaborative software development
  - Given many agile development processes – these repositories become often the sole form of system “documentation” being available.
  - Persistency and management of information across organization boundaries

Source Code Management (SCM) Requirements Space



Source Code Management Space

Version Management

- Traceable version history
- branch & merge
- stable workspaces for developers and integrators
- control over incorporation of concurrent changes

Configurations Management

- build anywhere, anytime, with correct components
- several builds running in parallel
- each build can run itself in parallel

Change Request Tracking

- link between change requests, solutions, releases
- state changes (tested, accepted, released, etc.)
- accurate status reports

SCM Requirements Space Cont.

Process Support

- process support for submit/pickup/build&test/accept
- parallel work is the rule
- change set tracking
- dependency tracking

Distribution

- distributed teams need to be coordinated.
- in case of time zone differences: difficult to do SCM informally.
- databases must be connected and up-to-date.

The change management process

```
Request change by completing a change request form
Analyze change request
if change is valid  then
    Assess how change might be implemented
    Assess change cost
   Submit request to change control board
   if change is accepted  then
       repeat
         make changes to software
         submit changed software for quality approval
      until  software quality is adequate
    create new system version
 else
    reject change request
else
    reject change request
```

Change request form

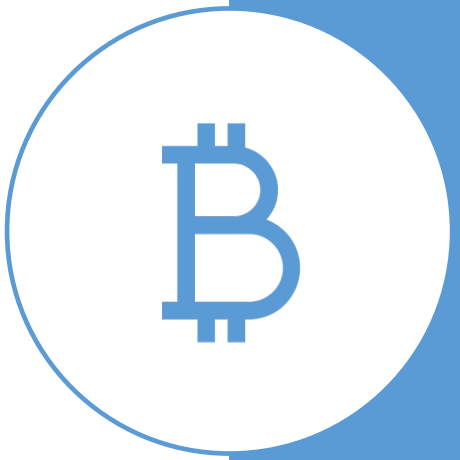
The definition of a change request form is part of the
CM planning process.

This form records the change proposed, requestor of change, the reason why change was suggested and the urgency of change(from requestor of the change).

It also records change evaluation, impact analysis,
change cost and recommendations (System
maintenance staff).

Defect Information

- Where Found
  - product, release, version, hardware, os, drivers, general area
- Who Found It
  - customer, internal, when
- Description of the Defect
  - summary, description, how to reproduce, associated data
  - links to related defects or features
- Triage
  - severity, likelihood → priority
- Audit Trail
  - all changes to the defect data, by whom, when
- State
  - state, owner



Change request form

Change Request Form

Project: Proteus/PCL-T ools **Number:** 23/02
Change r equester: I. Sommerville **Date:** 1/12/02
Requested change: When a component is selected from the structure, display the name of the file where it is stored.

Change analyser: G. Dean **Analysis date:** 10/12/02
Components affected: Display-Icon.Select, Display-Icon.Display

Associated components: FileT able

Change assessment: Relatively simple to implement as a file name table is available. Requires the design and implementation of a display field. No changes to associated components are required.

Change priority: Low

Change implementation:
Estimated effort: 0.5 days

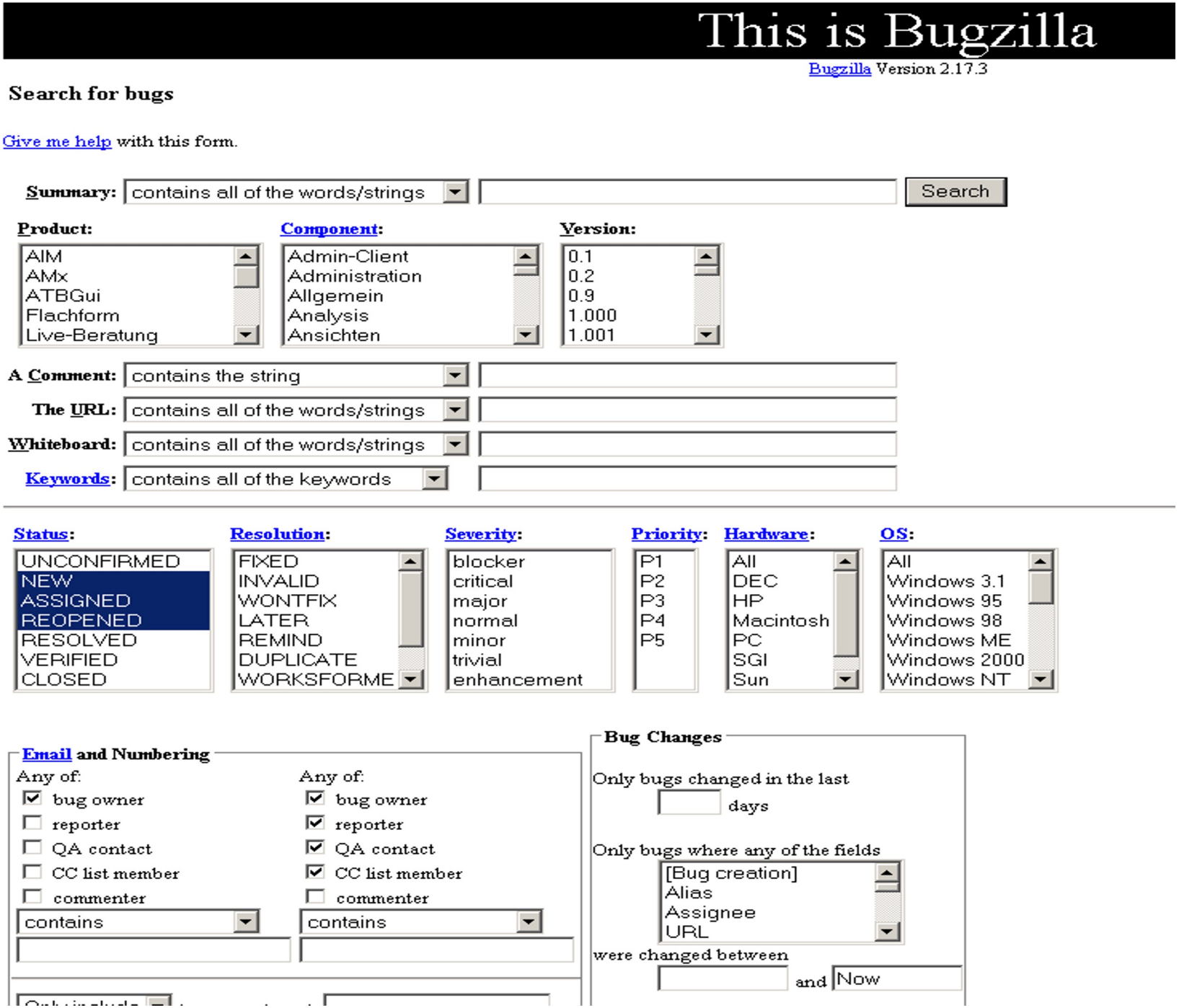
Date to CCB: 15/12/02 **CCB decision date:** 1/2/03
CCB decision: Accept change. Change to be implemented in Release 2.1.
Change implementor: **Date of change:**
Date submitted to QA: **QA decision:**
Date submitted to CM:
Comments

Change tracking tools

A major problem in change management is tracking change status.

Change tracking tools keep track the status of each change request and automatically ensure that change requests are sent to the right people at the right time.

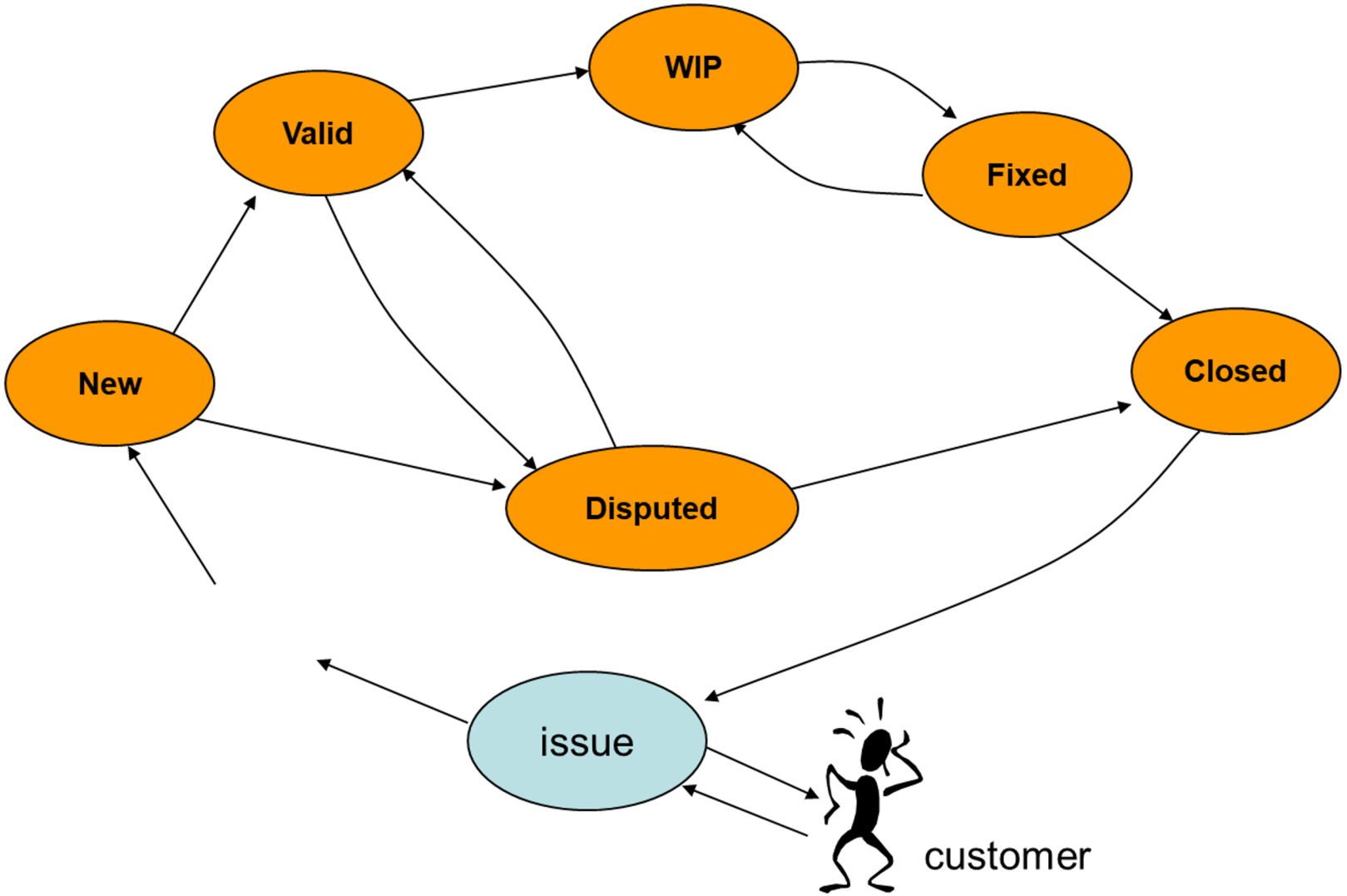
Integrated with E-mail systems allowing electronic change request distribution.


This is Bugzilla
Bugzilla Version 2.17.3
Search for bugs
Give me help with this form.
Summary: contains all of the words/strings
Search
Product:
AIM
AMx
ATBGui
Flachform
Live-Beratung
Component:
Admin-Client
Administration
Allgemein
Analysis
Ansichten
Version:
0.1
0.2
0.9
1.000
1.001
A Comment: contains the string
The URL: contains all of the words/strings
Whiteboard: contains all of the words/strings
Keywords: contains all of the keywords
Status:
UNCONFIRMED
NEW
ASSIGNED
REOPENED
RESOLVED
VERIFIED
CLOSED
Resolution:
FIXED
INVALID
WONTFIX
LATER
REMIND
DUPLICATE
WORKSFORME
Severity:
blocker
critical
major
normal
minor
trivial
enhancement
Priority:
P1
P2
P3
P4
P5
Hardware:
All
DEC
HP
Macintosh
PC
SGI
Sun
OS:
All
Windows 3.1
Windows 95
Windows 98
Windows ME
Windows 2000
Windows NT
Email and Numbering
Any of:
bug owner
reporter
QA contact
CC list member
commenter
contains
Any of:
bug owner
reporter
QA contact
CC list member
commenter
contains
Bug Changes
Only bugs changed in the last
days
Only bugs where any of the fields
[Bug creation]
Alias
Assignee
URL
were changed between
and Now

Guidelines for writing a good bug report

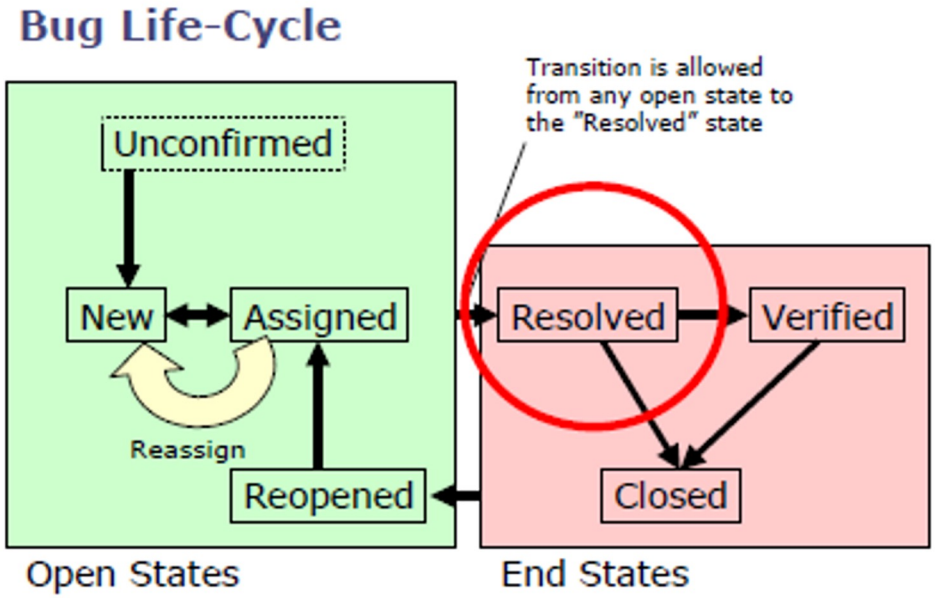
https://www.youtube.com/watch?v=Fkf565-pePs

Defect Workflow



Resolution

- The resolution field indicates what happened to this bug.
- Only bugs in"Resolved" state will be marked with one of the resolutions.
- All bugs which are in one of the "Open" states have no associated resolution.



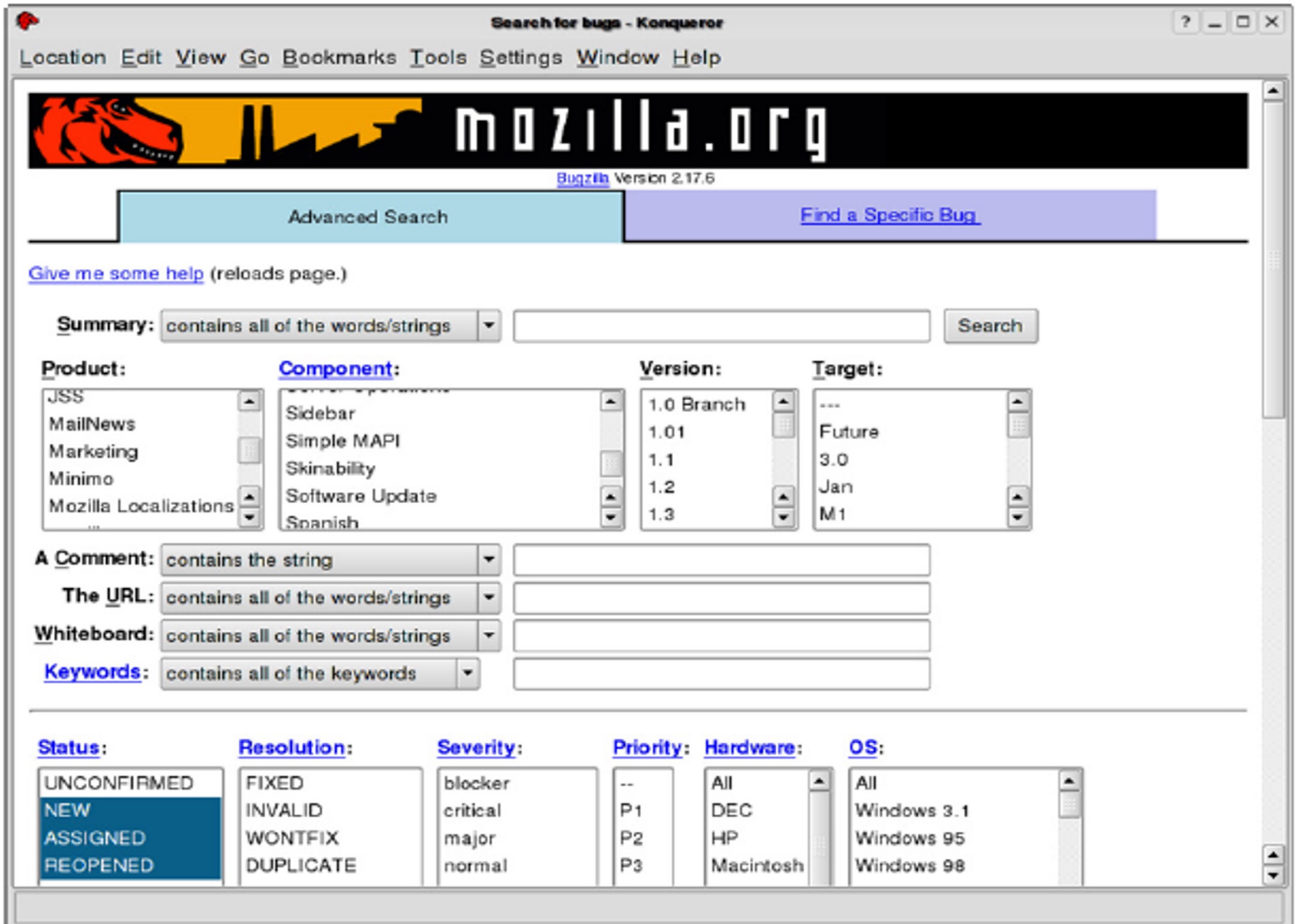
Resolution

- **FIXED** - A fix for this bug is checked into the tree and tested.
- **INVALID** - The problem described is not a bug.
- **WONTFIX** - The problem described is a bug which will never be fixed.
- **LATER** - The problem described is a bug which will not be fixed in this version of the product.
- **REMIND** - The problem described is a bug which will probably not be fixed in this version of the product, but might still be.
- **DUPLICATE** - The problem is a duplicate of an existing bug. Marking a bug duplicate requires the bug number of the duplicate and that number will be placed in the bug description.
- **WORKSFORME** - All attempts at reproducing this bug were futile, reading the code produces no clues as to why this behavior would occur. If more information appears later, please re-assign the bug, for now, file it.

Bug Status – End States

- RESOLVED - A resolution has been made, and it is awaiting verification by the QA.
  - From here bugs are either re-opened and become REOPENED, are marked VERIFIED, or are closed for good and marked CLOSED.
- VERIFIED- QA has looked at the bug and the resolution and agrees that the appropriate action has been taken.
  - Bugs remain in this state until the product they were reported against actually ships, at which point they become CLOSED.
- CLOSED - The bug is considered dead, the resolution is correct, and the product the bug has been reported against is terminated or shipped.
  - Any zombie bugs who choose to walk the earth again must do so by becoming REOPENED. This state is rarely ever used.
- NOTE: Resolution values can only be specified for bugs being in one of the end states!

Querying the issue tracker



Bugzilla list

This is Bugzilla

Bugzilla Version 2.17.3

Bug List

Sun Jun 22 23:28:47 MEST 2003

24 hours in a day..24 beers in a case.....a coincidence???

17 bugs found.

ID	Open Date	Last Changed Date	Sev	Pri	Plt	Assignee	Reporter	Status	Resolution	Product	Comp	Vers	Summary
6	2002-03-05	2003-05-22	nor	P2	Oth	scholze@atb-bremen.de	scholze@atb-bremen.de	ASSI		LVS-Neu	Server P	1.46	edi_bestand
765	2003-02-03	2003-05-22	nor	P2	PC	scholze@atb-bremen.de	scholze@atb-bremen.de	ASSI		LVS-Neu	Ansichte	1.59	Behälterstatistik
794	2003-03-19	Tue 12:28	enh	P2	PC	wuerthele@atb-bremen.de	scholze@atb-bremen.de	ASSI		ATBGui	Allgemei	unspe	Druckroutine verbessern (Style Report)
795	2003-03-19	2003-03-19	enh	P2	PC	scholze@atb-bremen.de	scholze@atb-bremen.de	NEW		ATBGui	Allgemei	unspe	XML Format für Statistiken
796	2003-03-19	2003-05-13	enh	P2	PC	wuerthele@atb-bremen.de	scholze@atb-bremen.de	ASSI		ATBGui	Allgemei	unspe	Export Funktionalität verbessern (Style Report)
797	2003-03-19	2003-05-13	enh	P2	PC	wuerthele@atb-bremen.de	scholze@atb-bremen.de	ASSI		ATBGui	Allgemei	unspe	Benutzeroption: Sprache
798	2003-03-19	2003-05-12	enh	P2	PC	wuerthele@atb-bremen.de	scholze@atb-bremen.de	ASSI		ATBGui	Allgemei	unspe	Benutzeroption: Farben
799	2003-03-19	2003-05-13	enh	P2	PC	wuerthele@atb-bremen.de	scholze@atb-bremen.de	ASSI		ATBGui	Allgemei	unspe	Benutzeroption: Font
800	2003-03-19	2003-03-19	enh	P2	PC	scholze@atb-bremen.de	scholze@atb-bremen.de	NEW		ATBGui	Allgemei	unspe	Trennung Oberfläche Logik
801	2003-03-19	2003-03-19	enh	P2	PC	scholze@atb-bremen.de	scholze@atb-bremen.de	ASSI		ATBGui	Allgemei	unspe	Suchfenster aus KLVS
802	2003-03-19	2003-05-12	enh	P2	PC	scholze@atb-bremen.de	scholze@atb-bremen.de	ASSI		ATBGui	SurfaceT	unspe	Toolbar verbessern
804	2003-03-19	2003-03-19	enh	P2	PC	scholze@atb-bremen.de	scholze@atb-bremen.de	NEW		ATBGui	Allgemei	unspe	Java Webstart Support
806	2003-03-19	2003-03-19	enh	P2	PC	campos@atb-bremen.de	scholze@atb-bremen.de	NEW		ATBGui	Allgemei	unspe	Layout Manager aus Pick
807	2003-03-19	2003-03-19	enh	P2	PC	scholze@atb-bremen.de	scholze@atb-bremen.de	NEW		ATBGui	Allgemei	unspe	Logik auf Server auslagern
810	2003-03-19	2003-03-19	enh	P2	PC	scholze@atb-bremen.de	scholze@atb-bremen.de	NEW		ATBGui	Allgemei	unspe	Code-Review
812	2003-03-19	2003-03-20	enh	P2	PC	scholze@atb-bremen.de	scholze@atb-bremen.de	ASSI		ATBGui	Allgemei	unspe	Log4J integrieren
864	2003-05-07	2003-05-07	enh	P2	PC	scholze@atb-bremen.de	scholze@atb-bremen.de	NEW		ATBGui	IfrmUebe	unspe	DataBinding

17 bugs found.

Long Format | CSV | Change Columns | Change Several Bugs at Once | Send Mail to Bug Owners | Edit this Query

Actions: New | Query | Find bug # | Reports | My Requests | My Votes

Edit prefs, parameters, users, products, flags, groups, keywords | Sanity check |
Log out scholze@atb-bremen.de

Preset Queries: My Bugs | AIM: Offene Fehler | AMx: Offene Fehler | ATBGui: Offene Fehler | Flachform: Offene Fehler | Live-Beratung: Offene Fehler | LVS: alle nicht geschlossenen | Prefas: Offene Fehler | STUTE-online-SHOP: Offene Fehler | teamhos: Offene Fehler

Advanced Features

Voting

- Voting allows users to be given a pot of votes which they can allocate to bugs, to indicate that they'd like them fixed.
- This allows developers to gauge user need for a particular enhancement or bugfix.
- By allowing bugs with a certain number of votes to automatically move from "UNCONFIRMED" to "NEW", users of the bug system can help high-priority bugs garther attention so they don't sit for a long time awaiting triage.

Version Control Systems

Problem 1: working solo

How do you keep track of changes to your program?

Problem 1: working solo

- Option 1: Don't bother
  - Hope you get it right the first time
  - Hope you can remember what changes you made if you didn't
  - (You probably won't get it right)
  - (Or remember)
  - (You will end up rewriting code)
  - (I do!)

Working solo (cont.)

- Option 2: Periodically save “backups”
  - Save snapshots of your program in another directory or under a different name
    - E.g. Main.1.java, Main.2.java
    - Or save it in a directory by date
  - Problems:
    - Totally ad hoc
    - Only the programmer knows how to interpret the names
    - Hard to pick a version to go back to
    - Prone to error
    - No tools to help you

Solving problem 1

When you get something working, commit the changes

Tools allow you to revert to a previous version

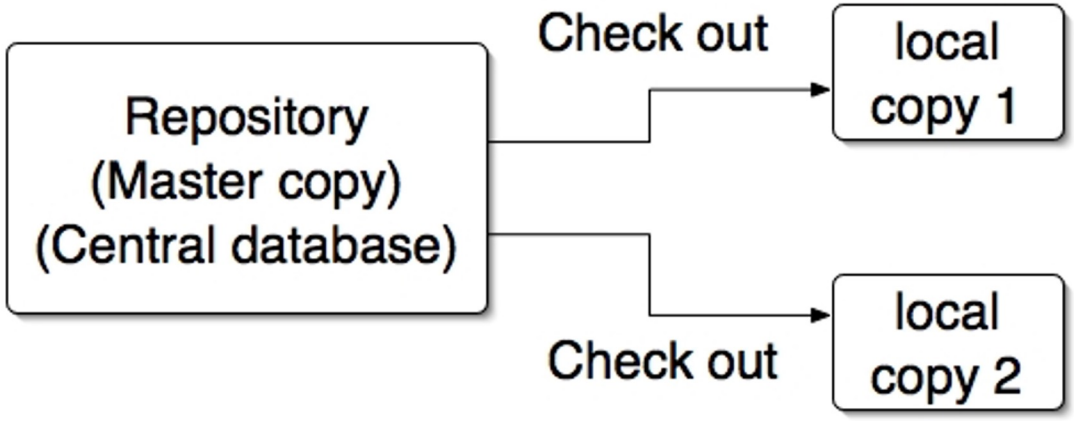
Write good log messages so that you don't have to remember what changed in each version

Problem 2: moving around

- After a hard day of work in the lab, you want to go home and do some work at home in the evening.
- How do you know which files to copy to your home machine?
  - Copy everything
    - potentially slow
    - might overwrite something you did at home, but forgot to copy from home to school
- Try to remember what changed
  - highly likely to get it wrong

Solution: version control

- Keep code in a central location ("repository")
  - This is the master copy
  - Never directly modify this directory
- Create a local copy of the repository in your account at school, on your machine at home, on your laptop...



Storing changes

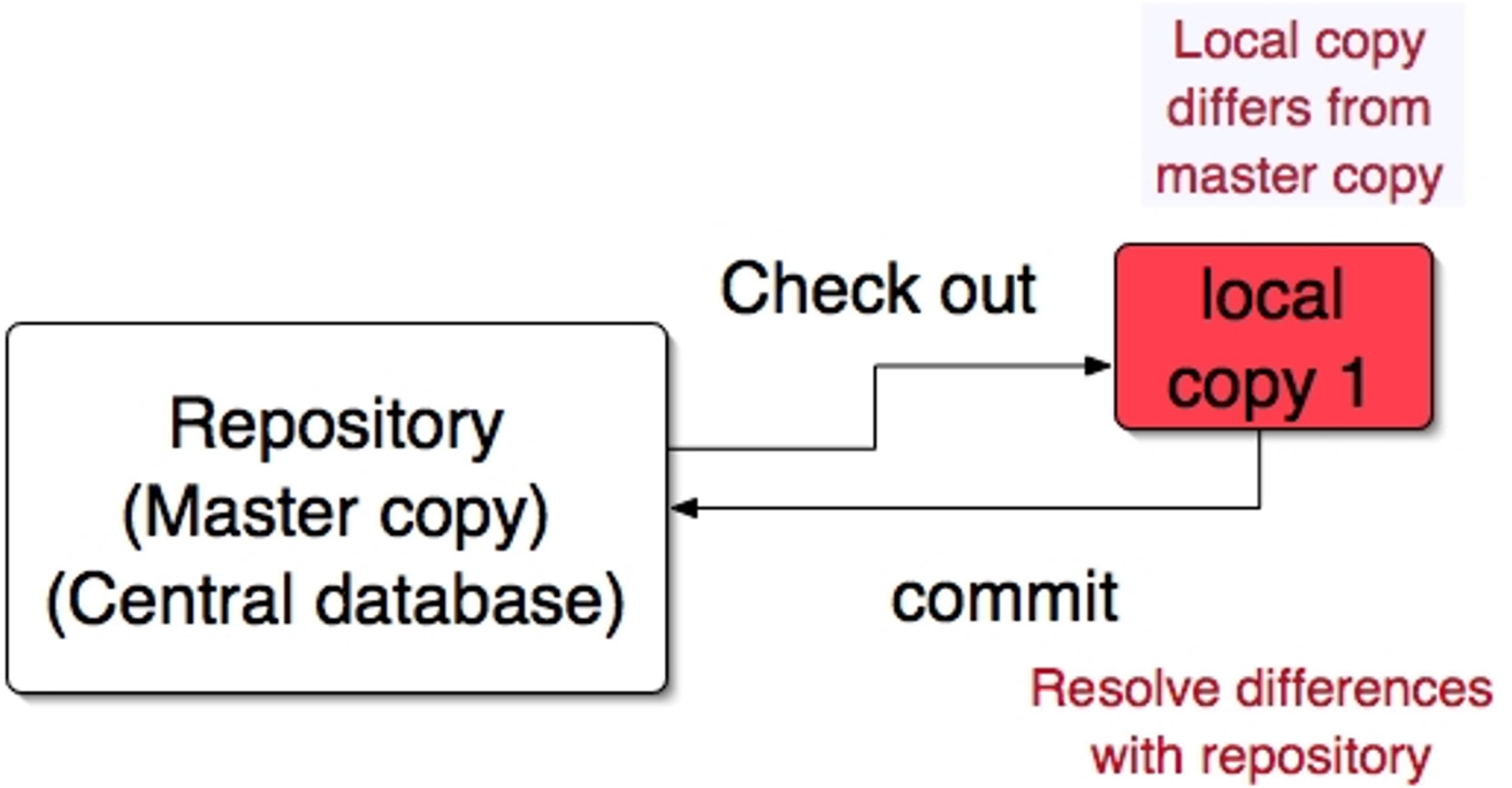
Storing entire copy of a file on every change would require an enormous amount of disk space

Version control systems store incremental differences

The incremental differences allow the system to reconstruct previous versions

Commit

When the local copy changes, “commit” the changes to the repository



Problem 3: working in a team

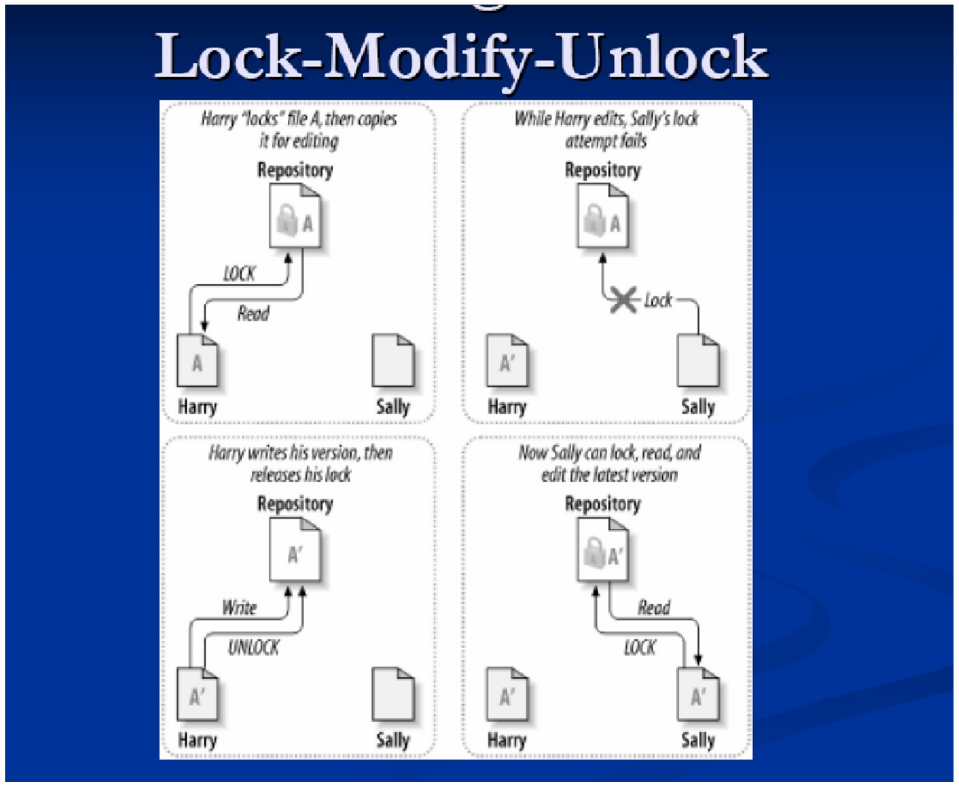
- How do you coordinate who has the authority to change a file?
- Worry about it after the fact
  - “Hey, why is this broken?”
  - “My changes got overwritten!”
  - “You weren't supposed to change that file.”
- Exchange email
  - “Okay, I'm going to work on A.java, so don't touch it.”

Managing concurrency

What if two (or more) people want to edit the same file at the same time?

Pessimistic Concurrency

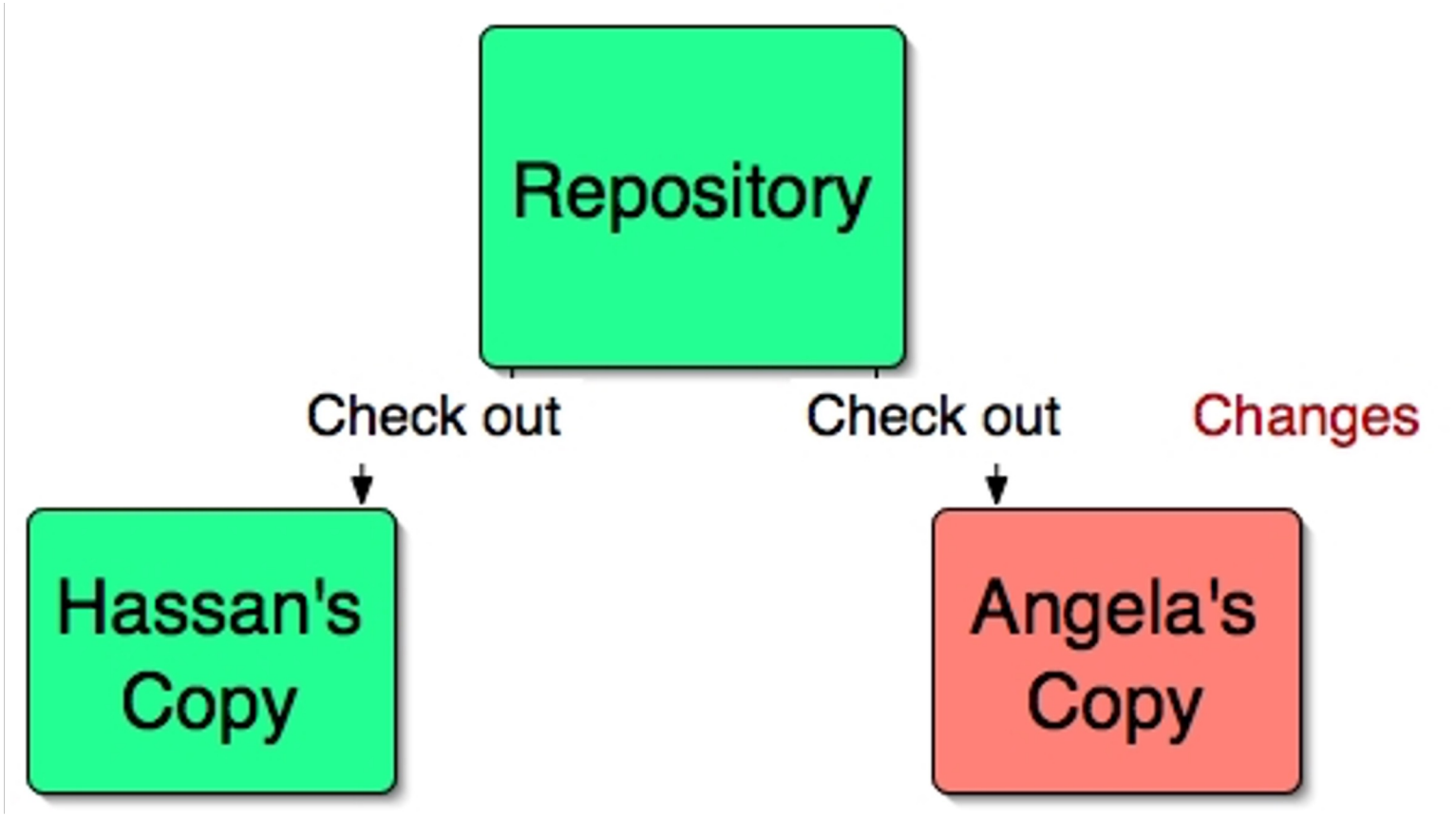
- **Option 1: prevent it**
  - Only allow one writeable copy of the file
  - Pessimistic concurrency Microsoft Visual SourceSafe (originally)



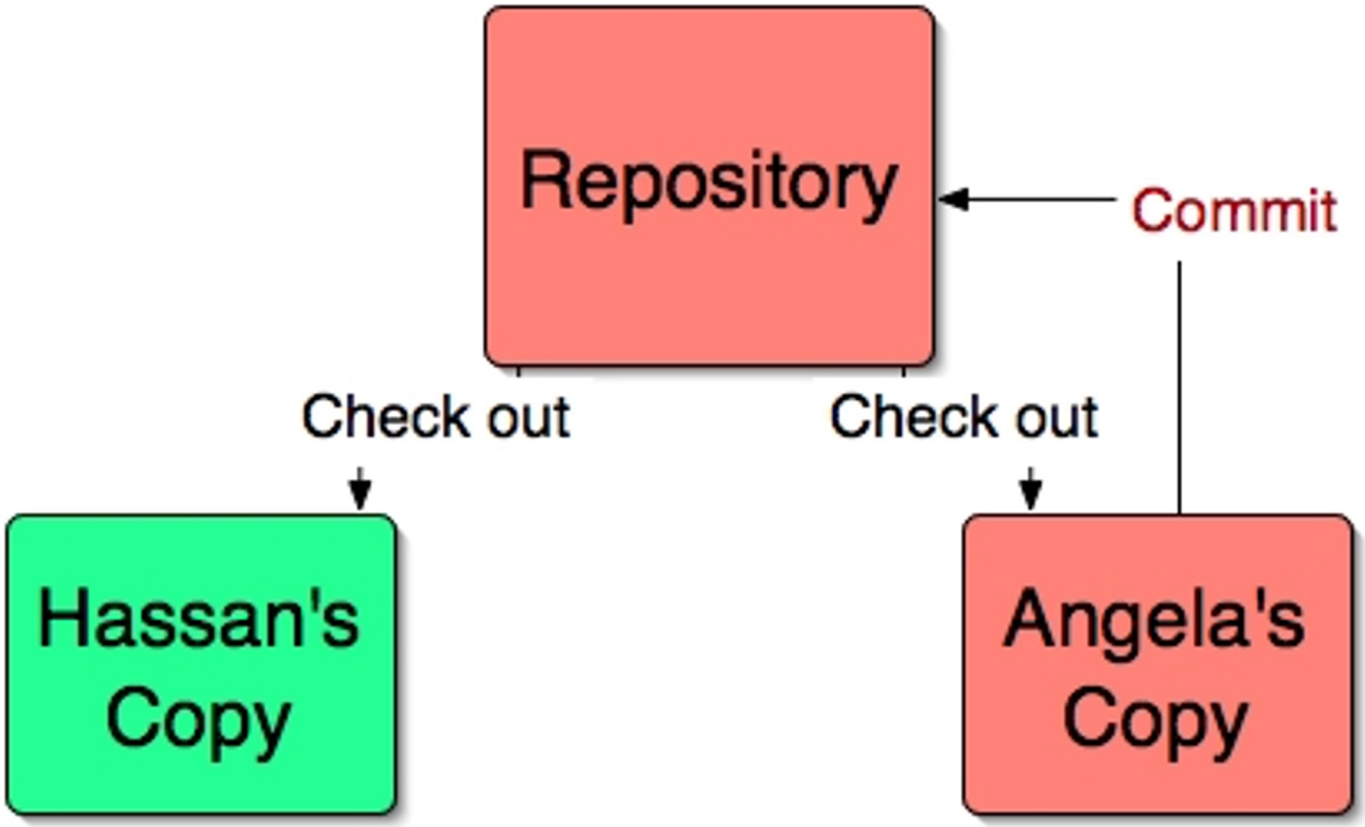
Optimistic concurrency

- Option 2: patch up afterward
  - Optimistic concurrency
  - "Easier to get forgiveness than permission"
  - CVS, Perforce, Subversion

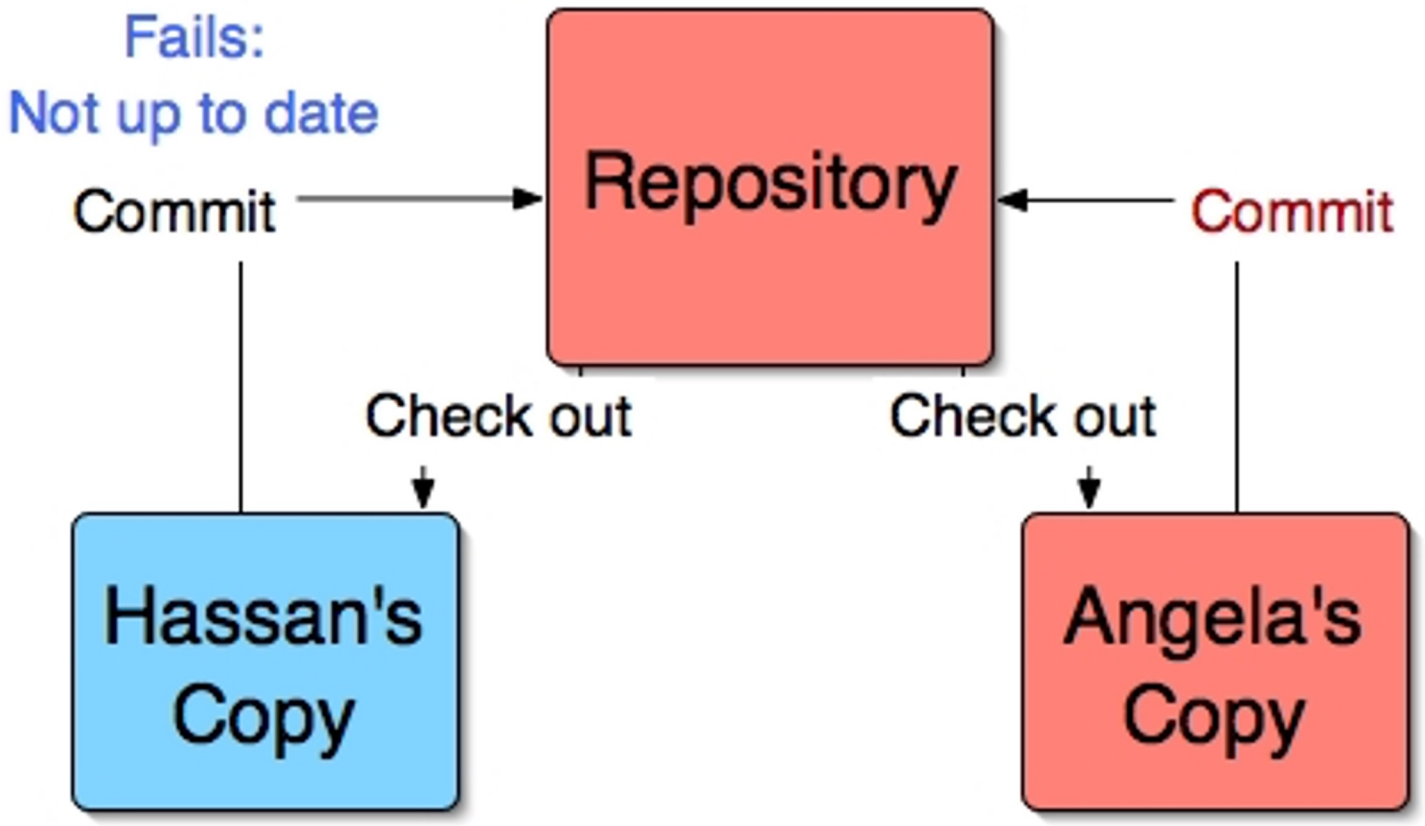
Optimistic concurrency: example



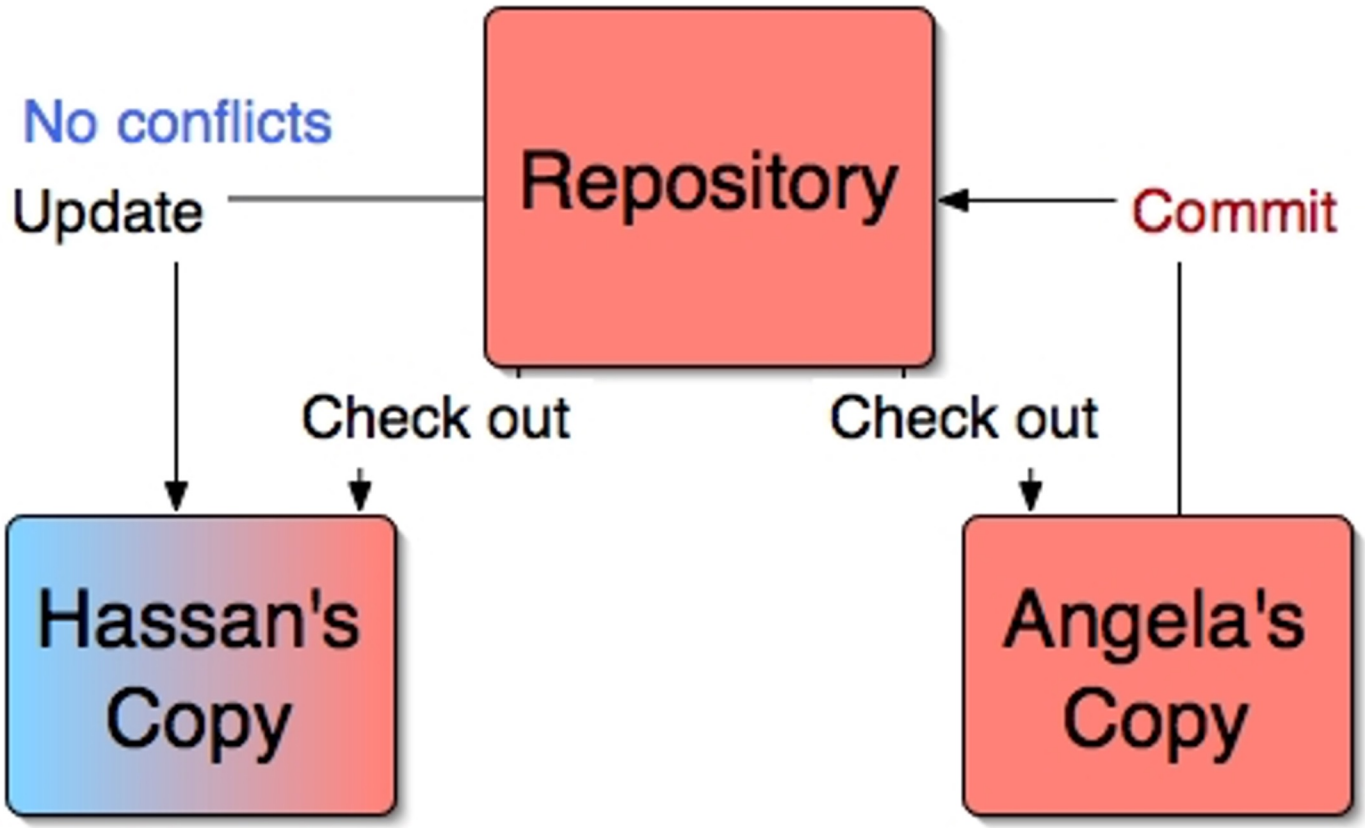
Angela commits changes



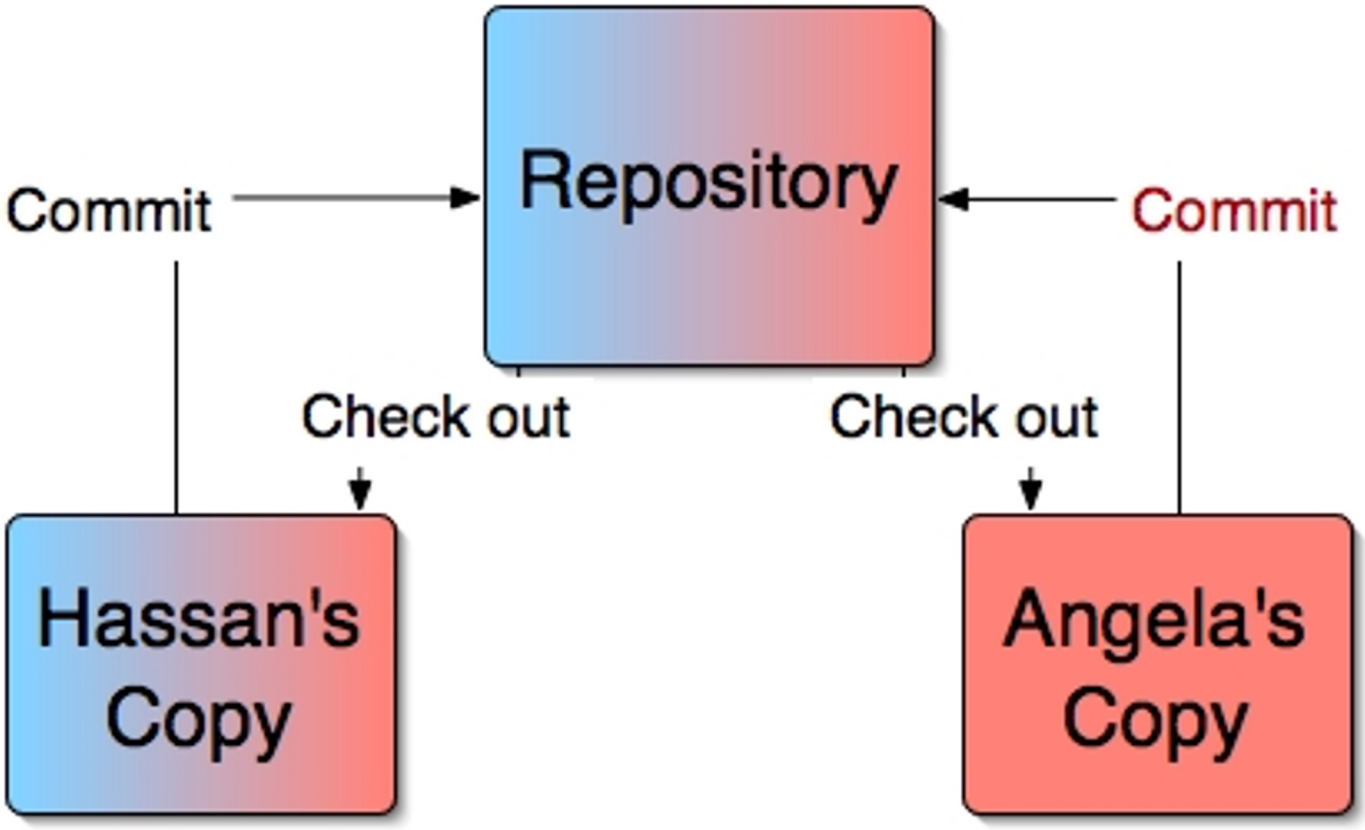
Hassan tries to commit changes



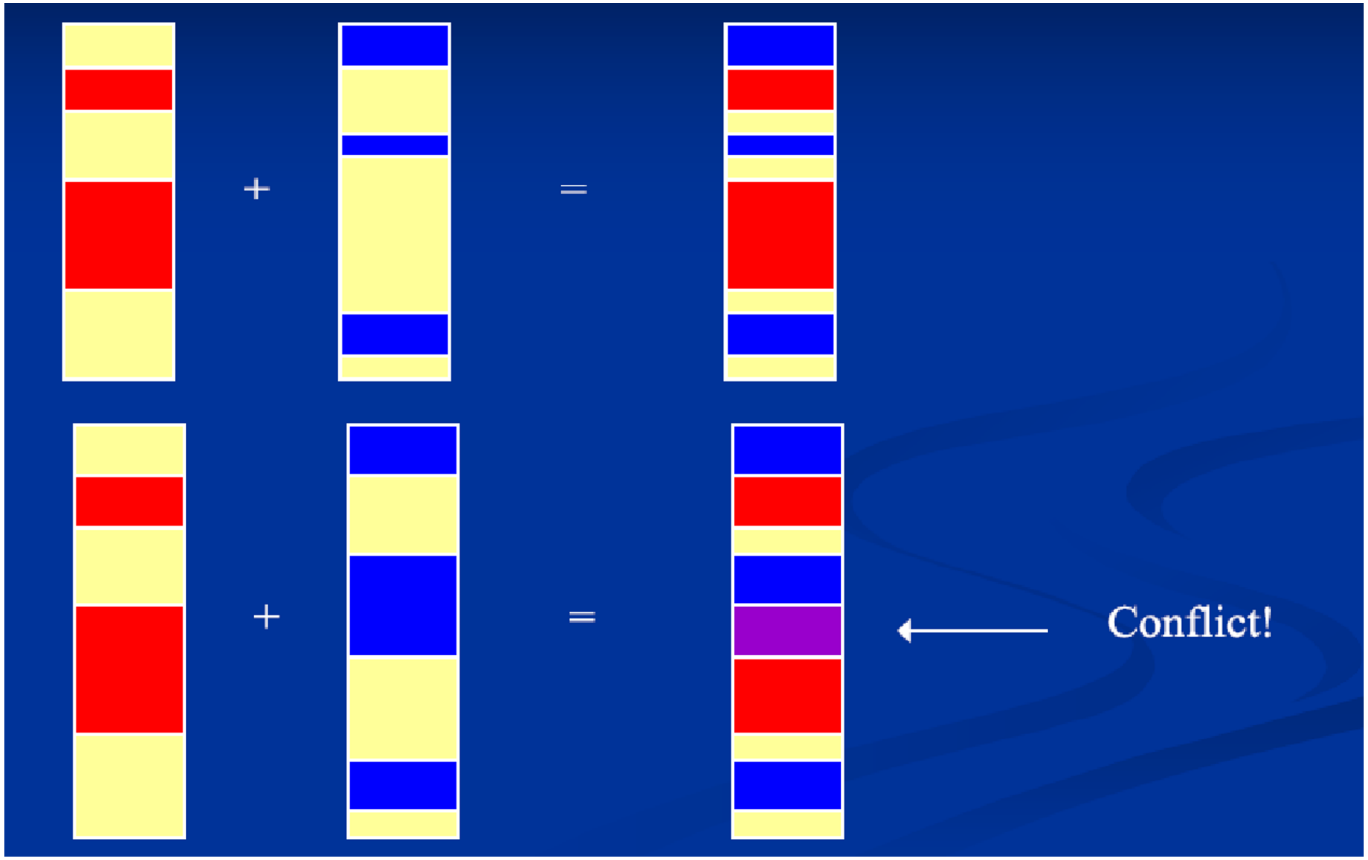
Hassan updates his version



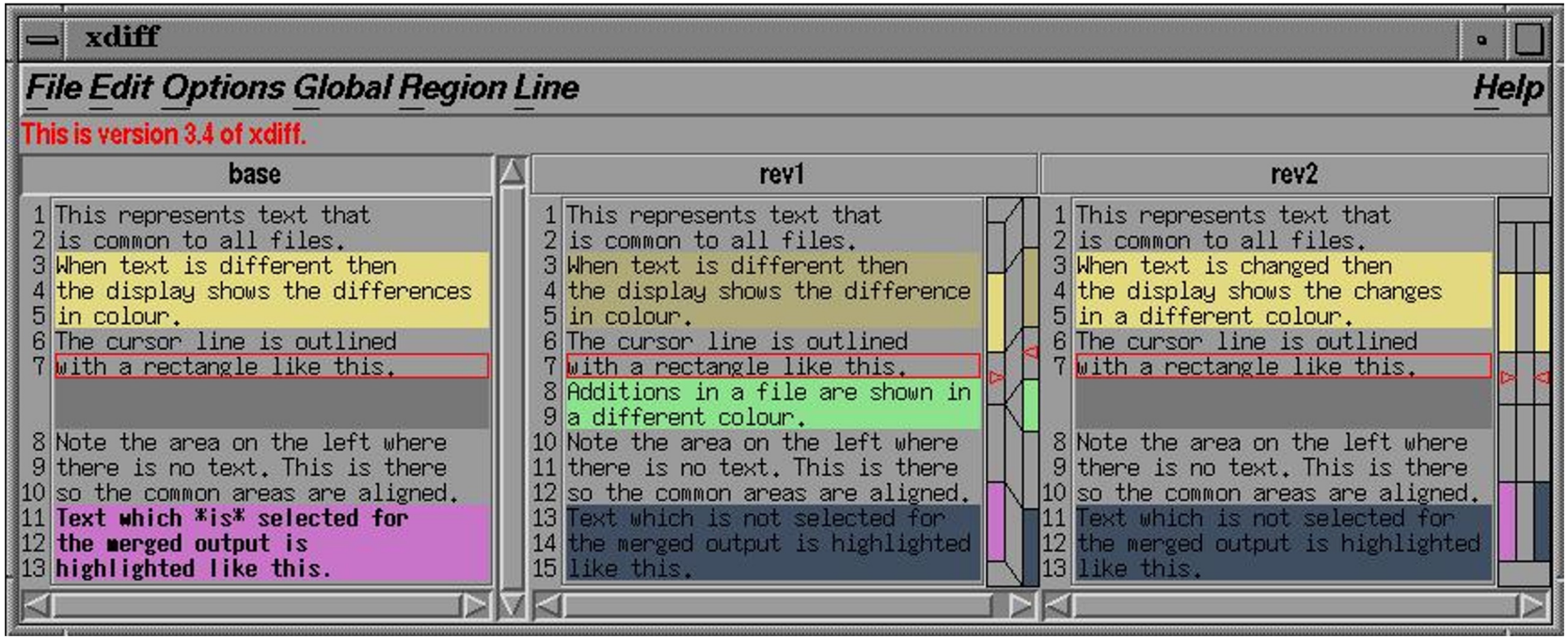
Hassan now commits



Merging



Merging Detailed view



More Uses of Version Control

Version control is not just useful for collaborative working, essential for quality source code development

Often want to undo changes to a file

- start work, realize it's the wrong approach, want to get back to starting point
- like "undo" in an editor…
- keep the whole history of every file and a *changelog*

Also want to be able to see who changed what, when

- The best way to find out how something works is often to ask the person who wrote it

Video: Distributed Version Control

Comparison

Centralized

- Server with database
- Clients have a working version
- Examples
  - CVS
  - Subversion
  - Visual Source Safe
- Challenges
  - Multi-developer conflicts
  - Client/server communication

Distributed

- Authoritative server by convention only
- Every working checkout is a repository
- Get version control even when detached
- Backups are trivial
- Reduced merging (initially)

A Brief History of Git

Linus uses BitKeeper to manage Linux code

Ran into BitKeeper licensing issue

- Liked functionality
- Looked at CVS as how not to do things

April 5, 2005 - Linus sends out email showing first version

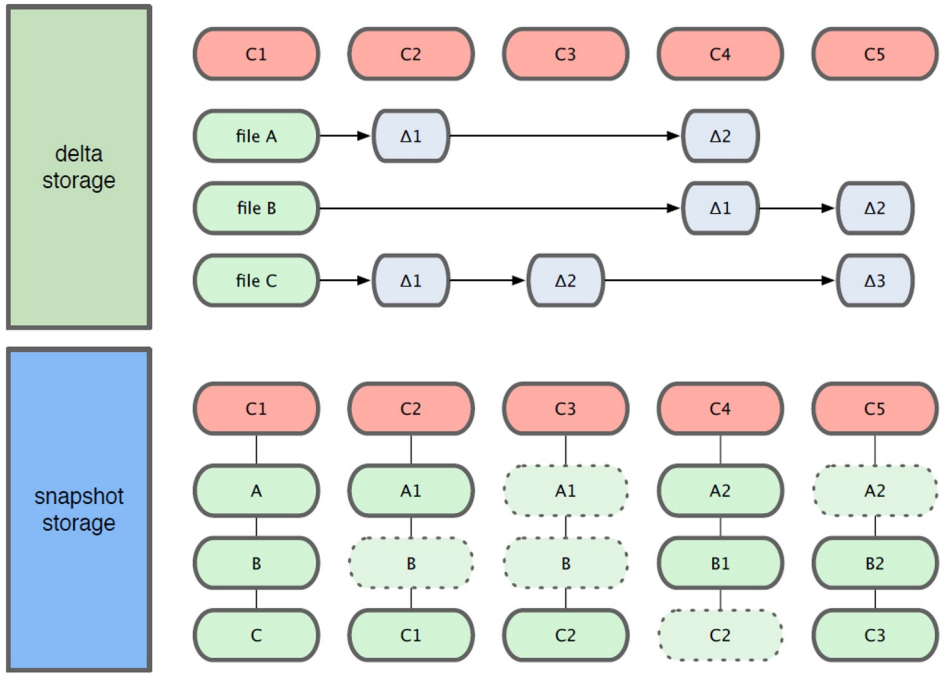
June 15, 2005 - Git used for Linux version control

Getting Started

- A basic workflow
  - (Possible init or clone) Init a repo
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

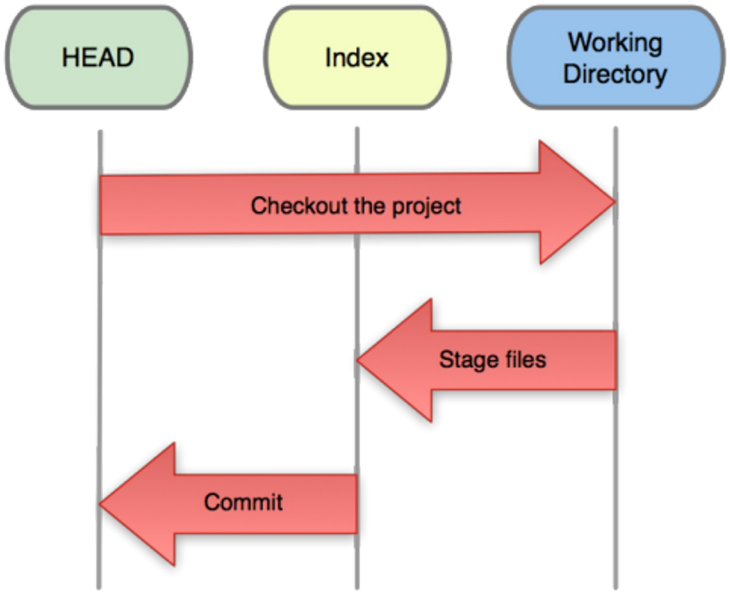
Getting Started

Git use snapshot storage



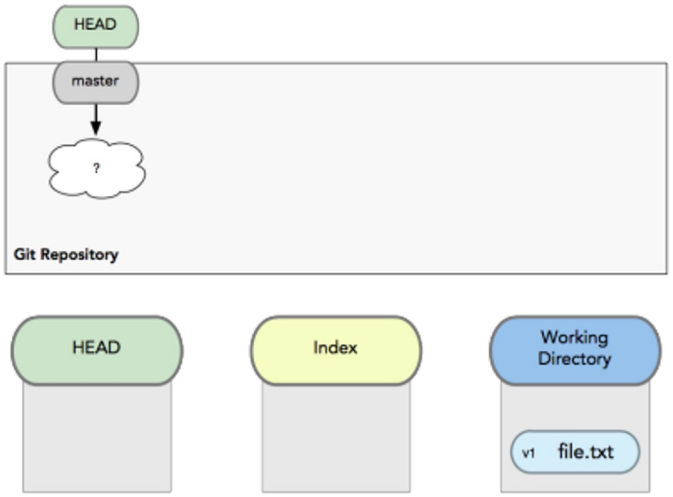
Getting Started

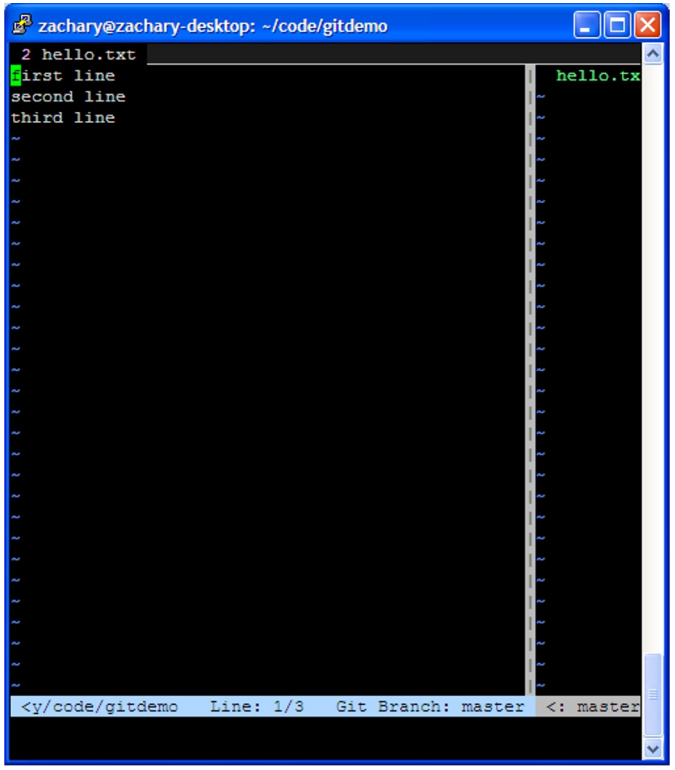
- Three trees of Git
  - The HEAD
    - last commit snapshot, next parent
  - Index
    - Proposed next commit snapshot
  - Working directory
    - Sandbox



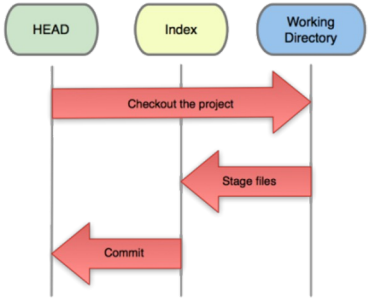
Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes



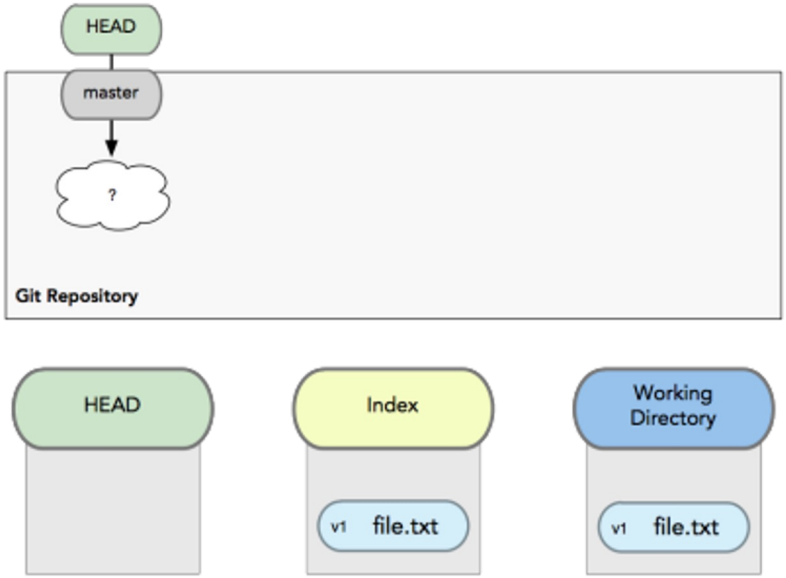


Getting Started



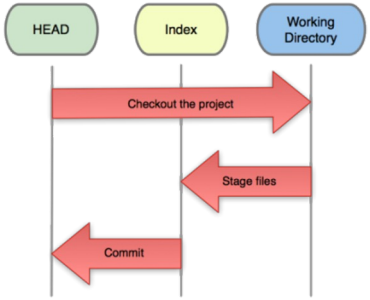
- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes

- Git add filename




```
zachary@zachary-desktop:~/code/gitdemo$ git status
# On branch master
# Changes not staged for commit:
#   (use "git add <file>..." to update what will be committed)
#   (use "git checkout -- <file>..." to discard changes in working directory)
#
#       modified:   hello.txt
#
no changes added to commit (use "git add" and/or "git commit -a")
```

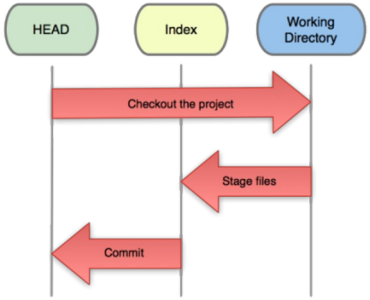
Getting Started



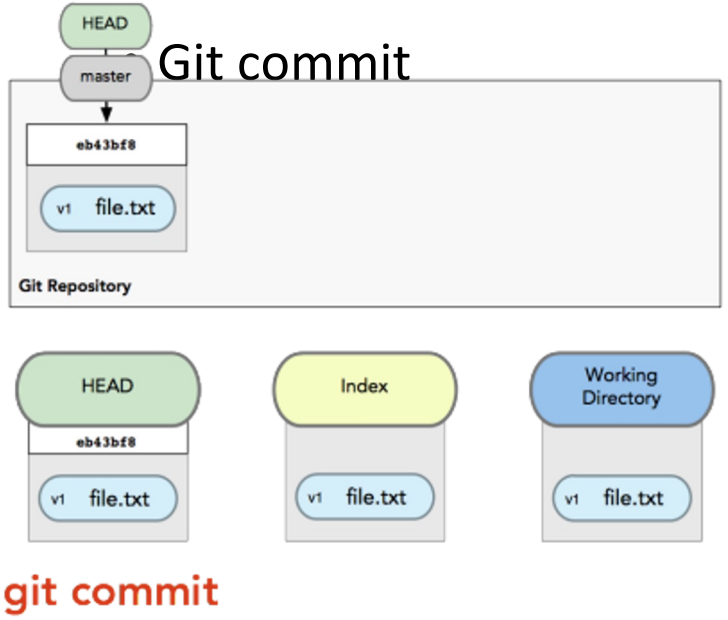
- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes


```
zachary@zachary-desktop:~/code/gitdemo$ git add hello.txt
zachary@zachary-desktop:~/code/gitdemo$ git status
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#       modified:   hello.txt
#
```

Getting Started



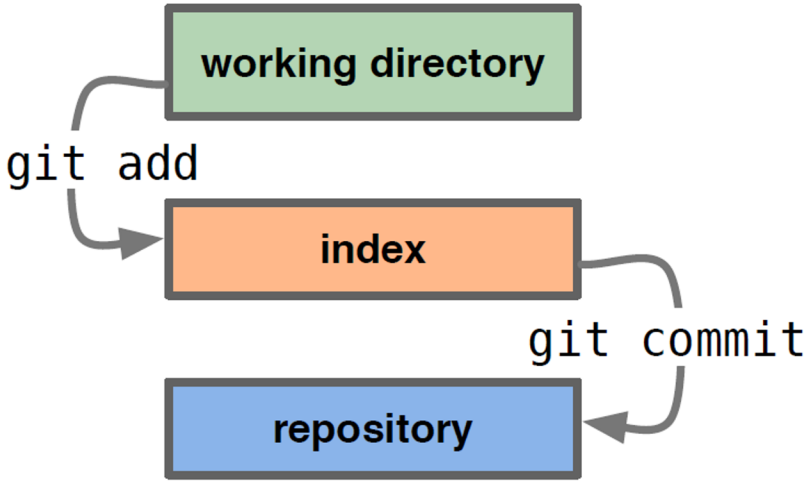
- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes




```
# Please enter the commit message for your changes. Lines starting
# with '#' will be ignored, and an empty message aborts the commit.
# On branch master
# Changes to be committed:
#   (use "git reset HEAD <file>..." to unstage)
#
#   modified:   hello.txt
#
```

Getting Started

- A basic workflow
  - Edit files
  - Stage the changes
  - Review your changes
  - Commit the changes



Basic workflow – short video:
https://www.youtube.com/watch?v=eDRt9wI15mI