CSE 403

Version Control Systems

Announcements

- Deliverables for 11/05 release (due 11:59PM)
  - Documents up to date: req, arch, design, schedule
  - Release notes
  - Status report for the **release** (not due until 11/07) -- content should be part of 11/07 presentation
- Presentations on 11/07: 7 minutes, same logistics as before. Deposit presentations (pdf) in dropbox by 11:59PM on 11/06
- Reading: Integration

Release notes

- A description of what you just released
  - High level, thematic description
  - Major functionality
  - Exceptions of what might not be working
  - How to access the release
- About 1 page

Project Presentations, 11/07 (7 minutes)

- Operations Review by manager or project manager-- similar to weekly status but for the release cycle
  - What did you say you were going to do? (1 minute)
  - What did you do? (1 minute)
  - What are you going to do in the next cycle? (1 minute)
  - What are the issues? (1 minute)
  - Report on a metric -- or if not in place, scale from 1(disaster) - 10 (exceeding) and why (1 minute)
- Demo (1 minute)
- Questions and Answers (1 minute)

Tip of the day
(worse is better)

- Worse is better than better
- Sooner is better than later
- Something is better than nothing
- The only truth is the source code

(Riff on "Worse is better" -- http://www.jwz.org/doc/worse-is-better.html)

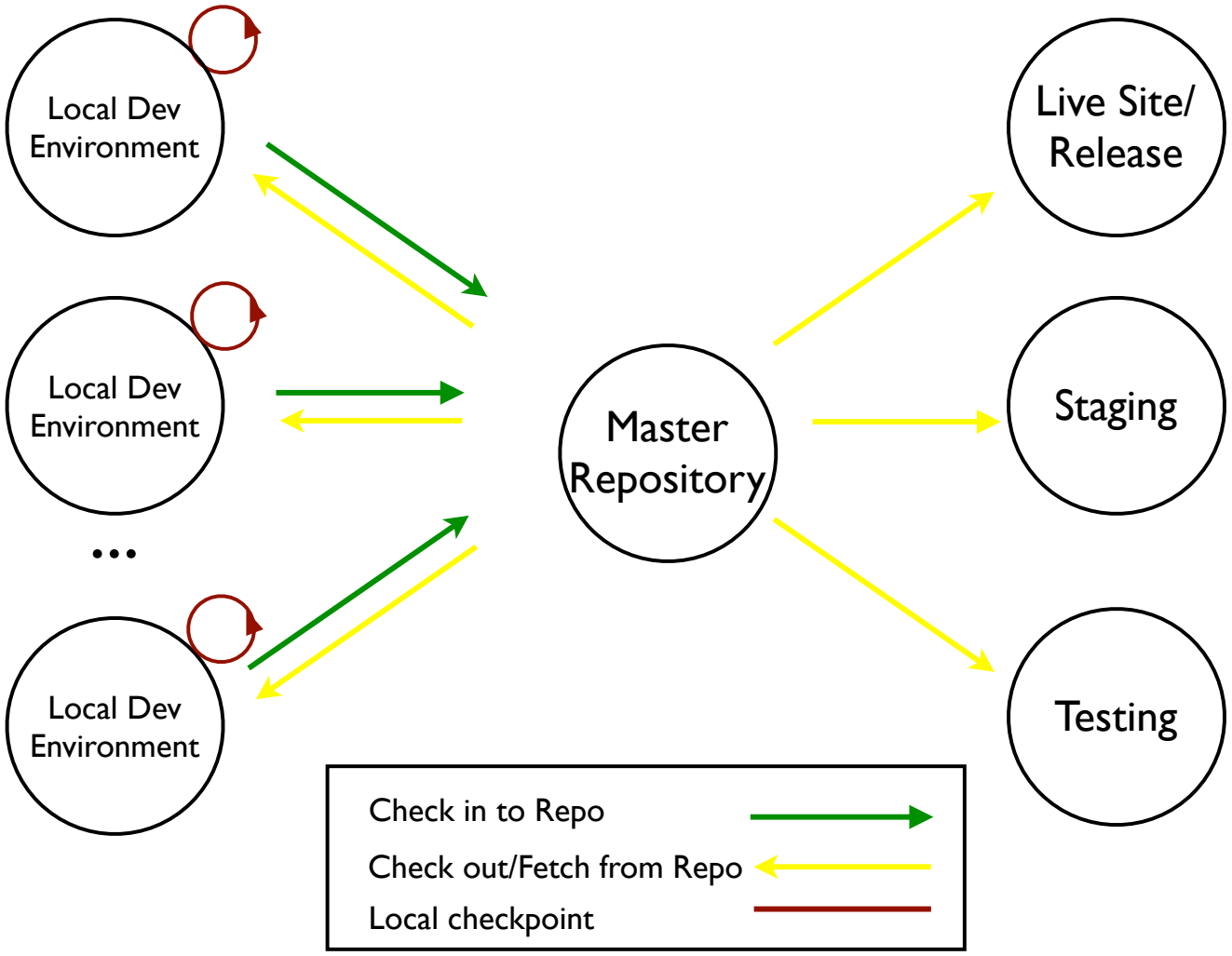
Version control systems

A version control system is a (software) system to help software teams manage their software assets.

Why?

- Keep a history of your work
  - checkpointing, incremental development
  - manage versions (releases)
  - disaster recovery
- Facilitate teams to work together
- Integral component of the software lifecycle

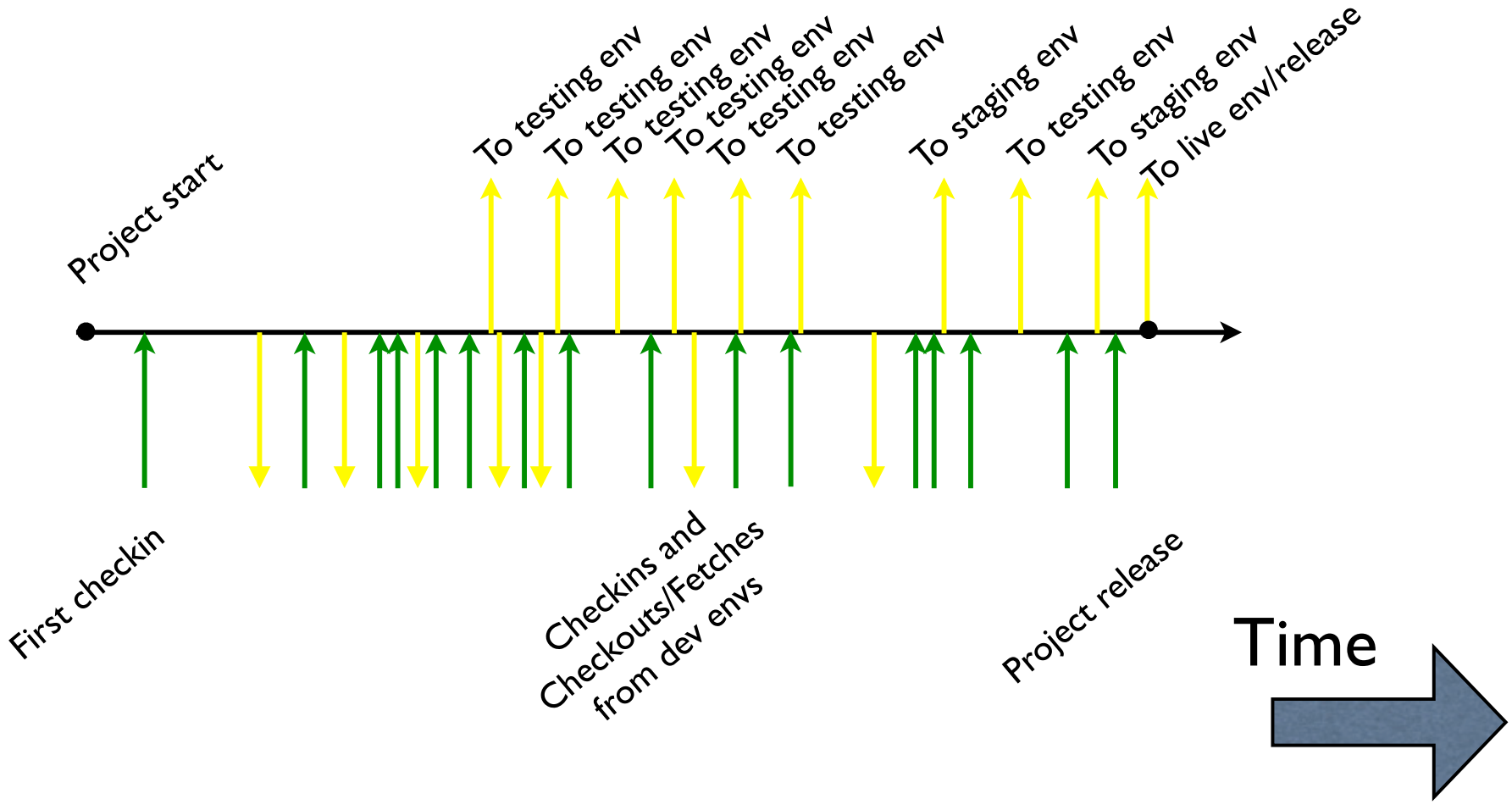
One viewpoint



Mechanics of VCS

- "Checkout" -- retrieve a set of files that comprise the source base
- "Edit"/"Create" -- make changes to a file/create file (granularity of change is typically a file)
- "Fetch" -- update local environment with what's in the repository
- "Checkin" -- push files that have been edited (or created)
- "Merge" -- resolve conflicts when two people are modifying the same file

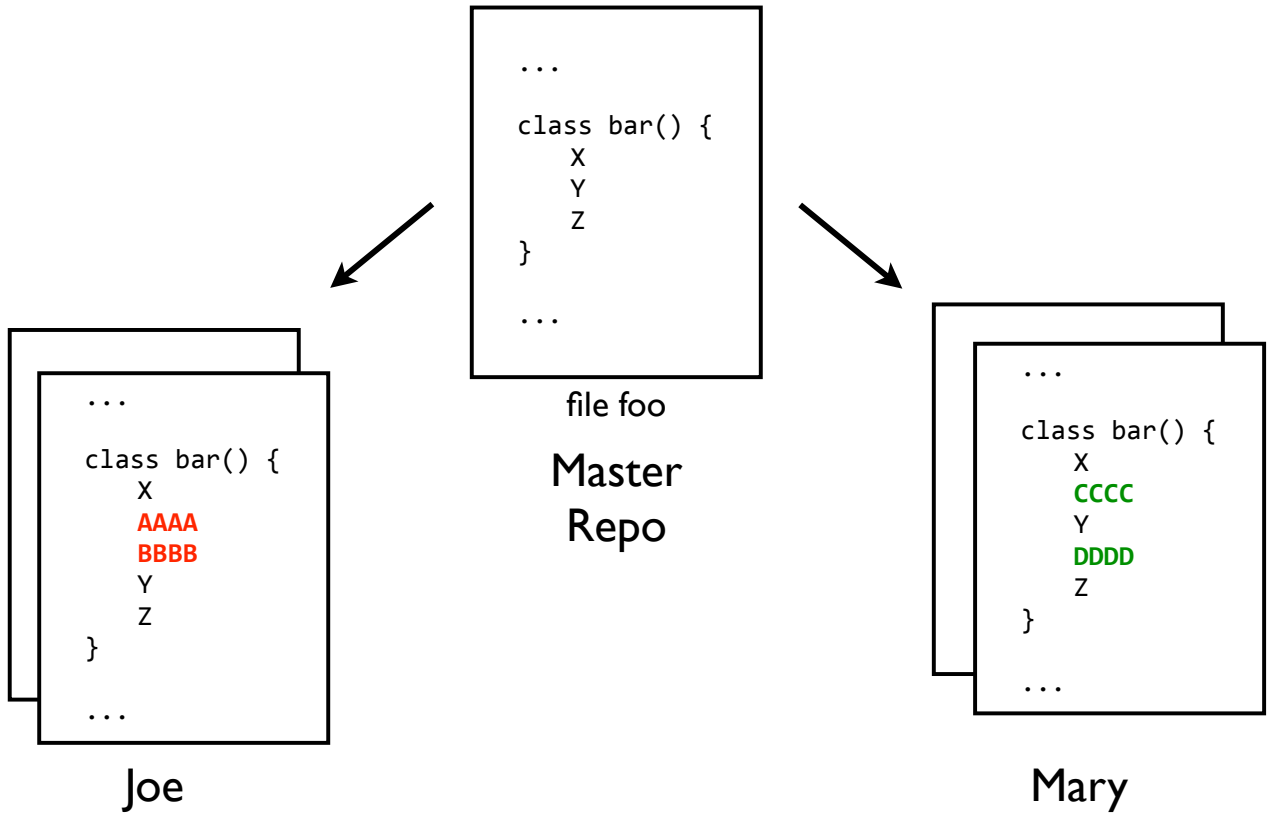
...from a time perspective



Conflicts and Merging

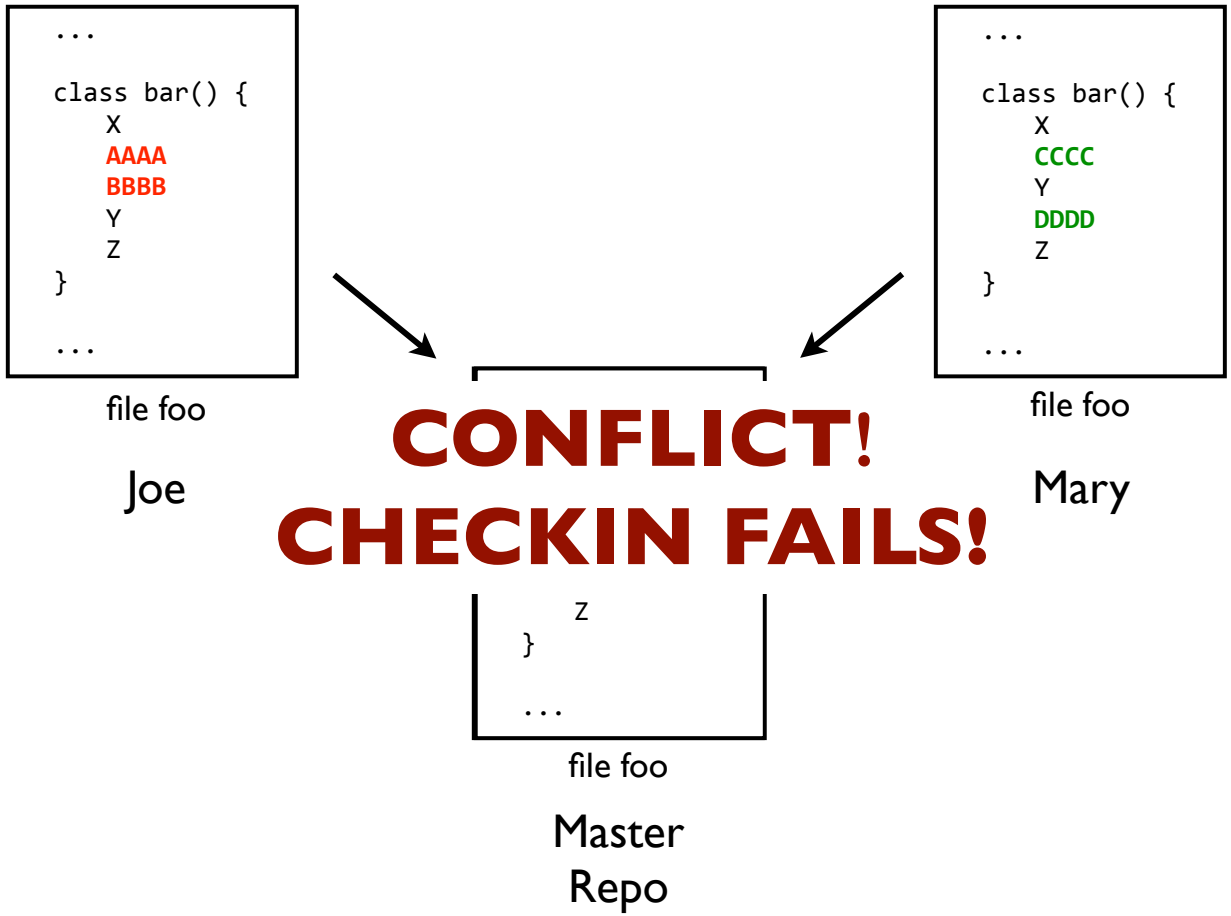
Conflicts and Merging

If two or more people are writing code, code conflicts are possible/likely



Conflicts and Merging

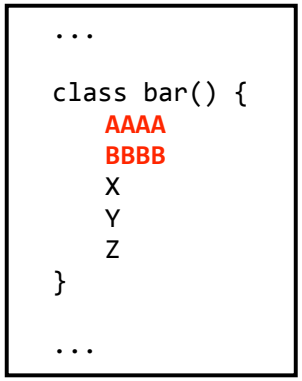
Changes by both Joe and Mary must find their way into the master repository.



Conflicts and Merging

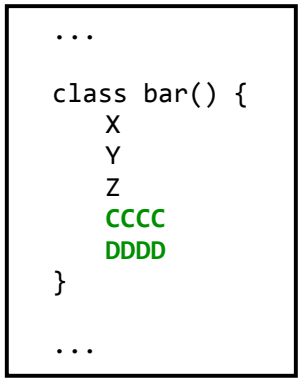
- Mary must first pull in Joe's changes
- VCS detects the TEXTUAL conflicts and prevents Mary's checking
- Mary must resolve the issues and merge the *thr*ee files
- There are tools to help with the merge
- Mary then can checkin her changes

...but semantic conflicts can't be detected



file foo

Joe



file foo

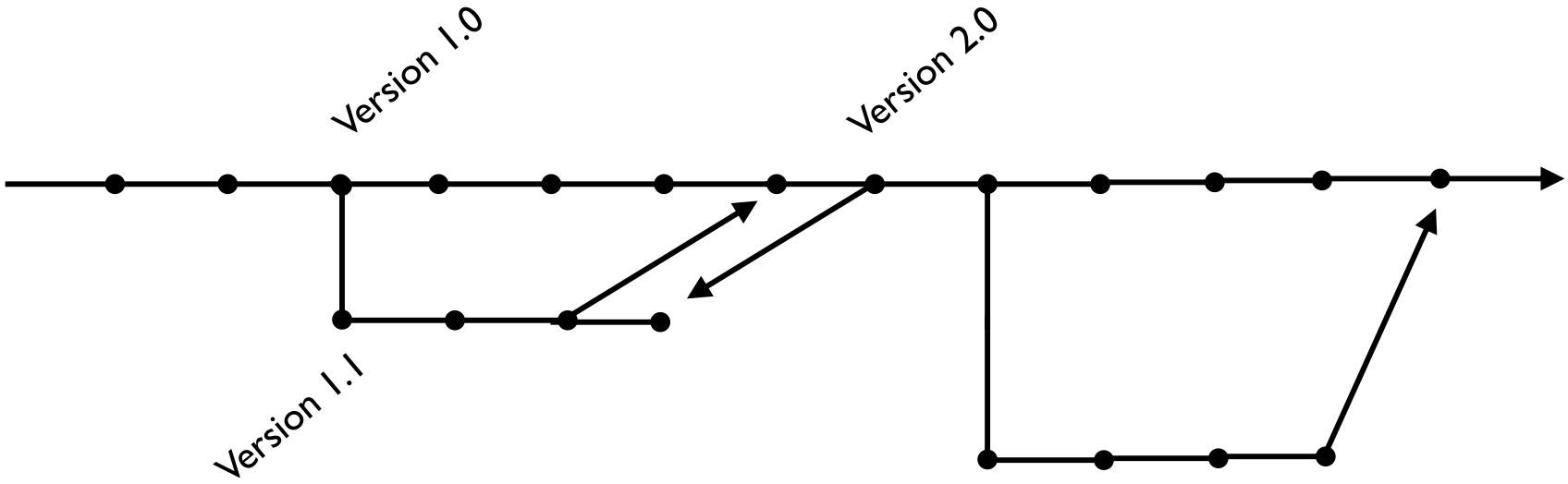
Mary

- No textual conflict -- changes are on different lines
- Mary still needs to pull in Joe's changes before checking in
- But VCS can't know if there is a semantic conflict
- Mary must be careful to make sure stuff still works

Conflicts and Merging

- Conflicts/merging can be painful
- Fetch and checkin(?) often
- Consider the granularity of you checkins
- Partition code intelligently to avoid conflicts
- Don't break the source base when you checkin(?)
- It's better to be Joe than Mary :-)

Branching



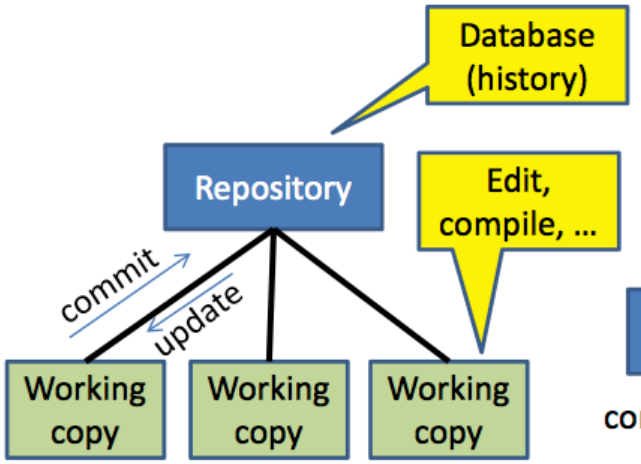
- Branching "forks" the source base
- Creates an "isolated" repository
- Fix bugs in deployed releases
- Allows for development in an isolated repo

Branch strategies

- Branching can be painful, and yet, often it adds value
- Branch strategy 1: Branch as little as possible -- minimizes conflicts across different repos
- Branch strategy 2: Branch liberally -- allows for experimentation, aggressive coding
- Pipelining/parallel releases may require branching
- Might make sense to put "big" features on a branch

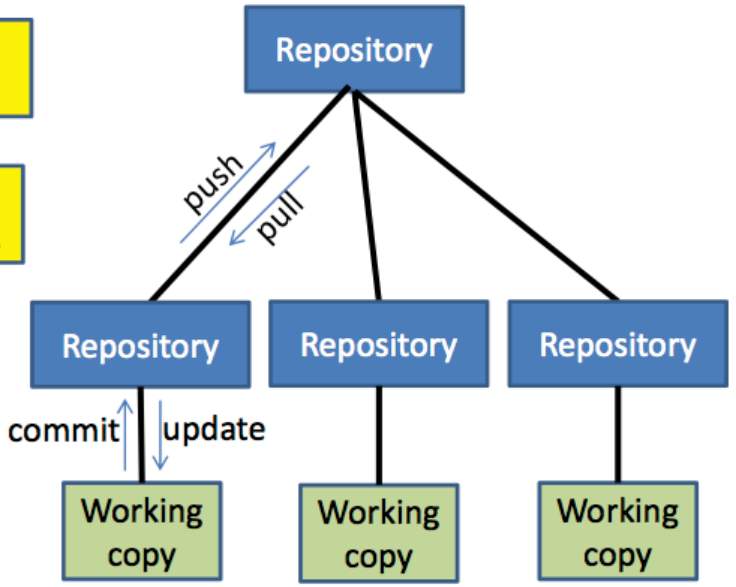
Varieties of version control systems

Centralized VCS



- One repository
- Many working copies

Distributed VCS



- Many repositories
- One working copy per repository

(More complicated topologies are possible)

Centralized vs. Distributed

- Distributed is more powerful
- But with power comes ~~responsibility~~ complexity
- Distributed = centralized + local repository (?)
- Distributed allows for branching off any repo (?)
- Distributed examples: git, mercurial
- Centralized examples: svn, cvs

Advantages of DVCS

- Checkpoint work without publishing to teammates
- Share changes selectively with teammates
- Commit, examine history when not connected to the network
- More accurate history
- More effective merging algorithms
- Flexibility in repository organization and workflow
- Faster performance

What about git/github?

- Distributed, powerful, feature rich, complicated
- Currently popular and in vogue -- encourage massive sharing through "social" aspects in github
- Easy to branch/clone from any repo -- but should you?
- "checkin", "checkout" are generic terms -- git uses it's own slightly different, vocabulary
- Good for your career!

VCS summary

- Why? Software asset management, sharing, integral to software lifecycle
- “checkout,” “checkin,” “edit”, “create,” “fetch,” “merge” are key concepts
- Conflicts and merging -- potential pain
- Branching can aid development/release maintenance
- Distributed vs. Centralized
- git/github is your popular friend (and you are required to use it!)