

Version Control: Mercurial

CS 120 Labs: mid-April, 2010

Allison Mankin
CCv3.0-A license

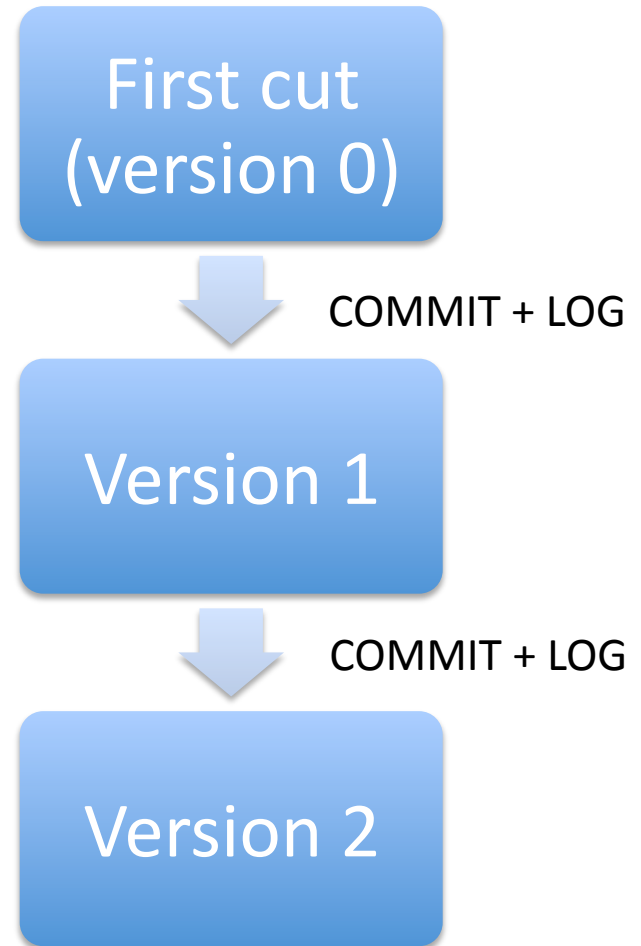
- What is Version Control for
- Basic Actions
- Mercurial basic commands:
hg init, add, commit
- Guidance and help
- Going back to last working version
- Working with team-mates
 - Using undergrad web pages
 - Using BitBucket

What is Version Control For?

Solo Developer

1. Provides a log of code development
2. If/when code goes bad, allows easy return to last good version
3. Allows fixing bug in previous version, updating all subsequent versions

aka Source Code Management
(SCM)



Basic Actions

- Initialize **repository**
- **Commit** version 0 into **repository**
- Always include textual information for the **Log**
- In **centralized** VC, check out code from server. In **de-centralized**, work with code in local directory.
- At a time that you decide on (usually when code meets some testing or functional goals):
 - **Review** your changes, including use of **diff**
 - **Commit** (with **log** information)
 - Arrive at next **version**



mercurial: de-centralized

Mercurial Basics

- Decentralized
 - Differs from cvs, svn
- Your code repository is local, wherever you mkdir it
- Example creates repository and commits one additional version
- Other basics:
 - % **hg log**
 - % **hg diff**

Resources:

[Both of these include quick starts]

<http://mercurial.selenic.com/>

<http://mercurial.selenic.com/guide/>

```
% mkdir pg5repo
% cp pg5/* pg5repo
% hg init p5repo
% cd pg5repo
% make clean (or rm a.out *.o *.~)
% hg add
% hg commit -m "initial commit"
[after commit, work on code]
% hg commit -m "1 ¼ bells – they work"
[or use a file of commit info]
% hg commit -l ./notes.ver1
```

Typing 'hg' with no command lists basic commands with brief description; also try 'hg -v'

% hg

```
Mercurial Distributed SCM

basic commands:

add          add the specified
annotate    show changeset inf
clone        make a copy of an
commit       commit the specifi
diff         diff repository (o
export       dump the header an
forget       forget the specifi
init         create a new repos
log          show revision hist
```

Equivalent of man pages:
'hg help <command>'
(or 'hg help -v <command>')

```
hg commit [OPTION]... [FILE]..
```

```
aliases: ci
```

```
commit the specified files or
```

```
Commit changes to the give
centralized RCS, this oper
way to actively distribute
```

```
If a list of files is omit
committed.
```

```
If you are committing the
or -I/-X filters.
```

```
If no commit message is sp
prompt you for a message.
```

```
See 'hg help dates' for a
```

```
options:
```

```
-A --addremove      mark new/m
--close-branch     mark a bra
```

Using Mercurial Versioning

[In repository directory]

```
% hg commit -m "a whistle – works"
```

[we're at version 2 now]

```
% hg commit -m "another bell – works"
```

[we're at version 3 now]

```
% hg commit -m "new whistle – works"
```

[we're at version 4 now]

[we work on code and try to get new
Bells and whistles working, but can't]

[commit in case want to re-visit broken]

```
% hg commit -m "add a lot – BROKEN"
```

[return to last working]

```
% hg update -r 4
```

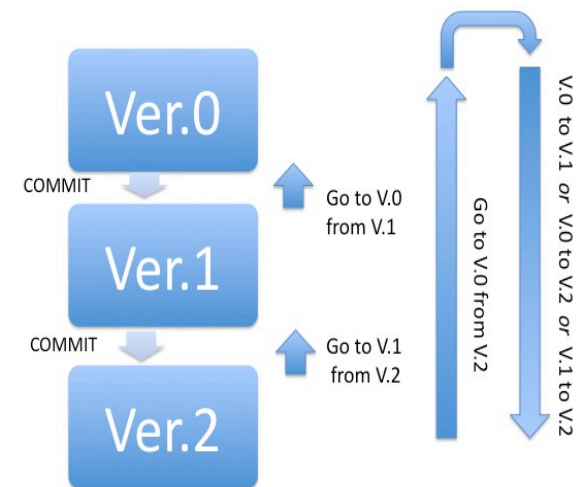
[work with not broken code]

```
% hg diff -r 5 > file [info on breakage]
```

```
% hg commit -m "bells work again, also  
have added bell unit tests"
```

hg update -r <version> -
checks out earlier or later code

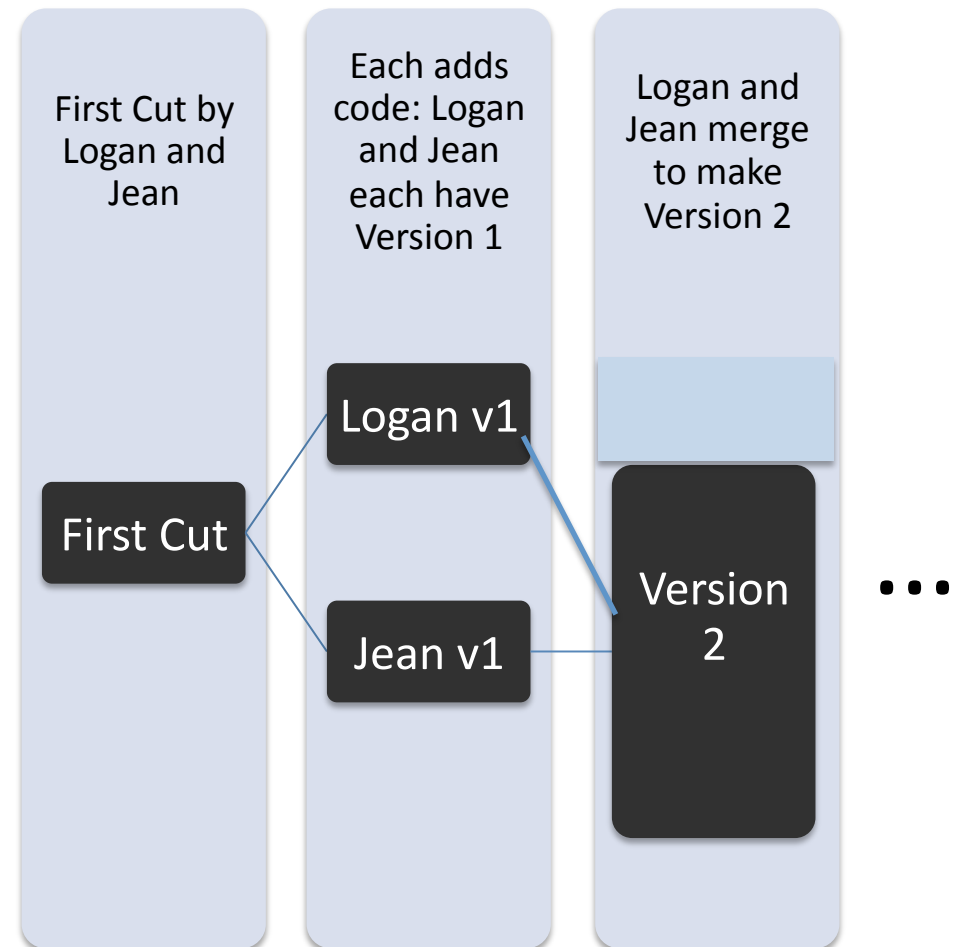
Moving among Versions



Multiple Developers

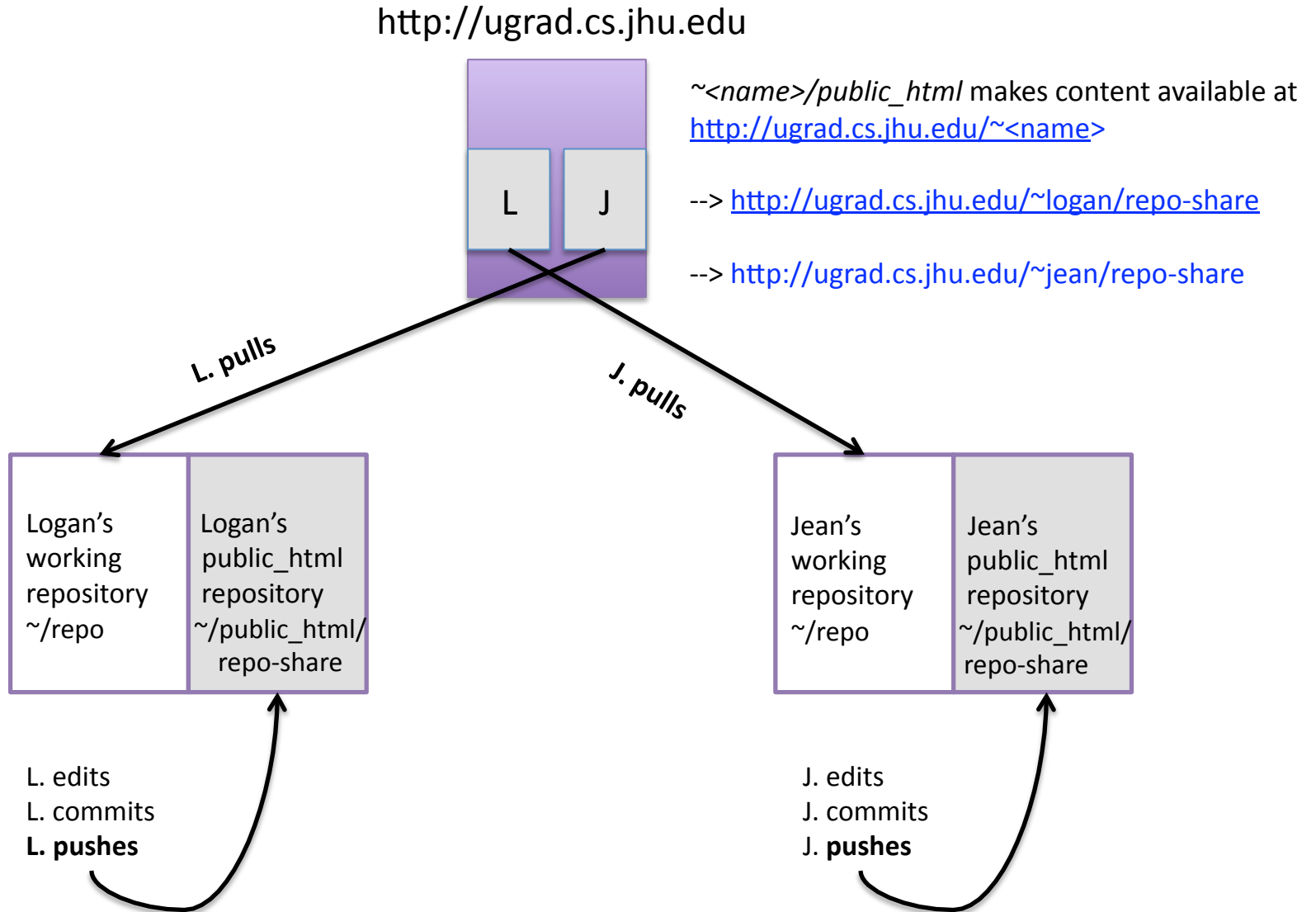
◆ Two Person Team

- Consider case: team's first cut is some common code that compiles and runs, but is incomplete
 1. VC provides same 3 benefits as before
 2. VC supports team members working separately and then re-synchronizing and merging their code



Mercurial sharing via public_html – systems picture

~/public_html holds data for your undergrad web page. If you missed creating this or for more info: <http://www.cs.jhu.edu/webdocs/>



Mercurial sharing via public_html (cont)

Commands to Use and Illustration

Team-mate 1 (ex. Logan)

```
[first time: establish repo on website]  
% hg clone repo ~/public_html/repo-share  
% cd public_html; chmod 705 -R repo-share  
[Logan edits and commits in his repo,  
that is, not in ~/public_html/repo-share]  
% hg commit -m "add bell and whistle to A.cpp"  
[Logan pushes Version 1 from his repo to his  
public_html]  
% hg push ~/public_html/repo-share
```

[Team-mates communicate person2person (chat...)]

```
[Logan wants to sync with Marie]  
% hg pull http://ugrad.cs.jhu.edu/~marie/repo-share  
[synced, but files must be checked out]  
  
% hg update
```

Team-mate 2 (ex. Marie)

```
[first time: establish repo on website]  
% hg clone repo ~/public_html/repo-share
```

```
[Marie wants to sync with Logan]  
% hg pull http://ugrad.cs.jhu.edu/~logan/repo-share  
[synced, but files must be checked out]  
% hg update [ automatically checks out head ]  
[Marie edits and commits in her repo]  
% hg commit -m "new bell and ½ a whistle B.cpp"  
[Maries pushes Version 2 to her public_html]  
% hg push ~/public_html/repo-share
```

Logan and Jean are synced @ Version 2

NOTE: before you can hg pull, you need to be in your hg repository. Error otherwise: no hg repository here

hg clone, push, pull: another view

First Cut by Logan and Jean:

Logan: **clones** to Logan's public_html

Jean: **pulls** their common code into her repository

First Cut

Each adds code: Logan and Jean each have different v1's

Logan **pushes** his to his public_html

Jean **pushes** hers to her public_html

Logan v1

Jean v1

Logan **pulls** Jean's v1 from her website -> result is a v2 in his repository that merges their code

Jean **pulls** Logan's v1 from his website. Now Jean has v2 as well and Logan and Jean have synched code.

Version 2

...

Mercurial (cont) – Sharing via BitBucket

Team-mate 1 (Logan)

*[first time: make sure team-mates have read access.
From your repo directory, populate BB]*

```
% hg push http://bitbucket.org/loganX/repo-share
http authorization required
realm: Bitbucket.org HTTP
user: <enter loganX>
password:
```

[Logan edits and commits in his repo]
% **hg commit** -m "add my bell and whistle"
% **hg push** http://bitbucket.org/loganX/repo-share

[Logan wants to sync with Jean]
% **hg pull** http://bitbucket.org/JeanG/repo-share
[synced, but files must be checked out]
% **hg update**

Team-mate 2 (Jean)

*[first time: make sure team-mates have read access.
From your repo directory, populate BB.
Authorization not shown...]*

```
% hg push http://bitbucket.org/JeanG/repo-share
```

[Jean wants to sync with Logan]
% **hg pull** http://bitbucket.org/loganX/repo-share
[synced, but files must be checked out]
% hg update *[automatically checks out head]*
[Jean edits and commits in her repo]
% **hg commit** -m "new bell and ½ a whistle B.cpp"
% **hg push** http://bitbucket.org/JeanG/repo-share

Logan are Jean are synced @ Version 2

Merging

- System creates merged code from the code that you commit independently (in **hg pull**, the merge is done for you)
- Using SCM encourages modularizing your code well, which will result in few or no conflicts
- When there are conflicts: **hg resolve**

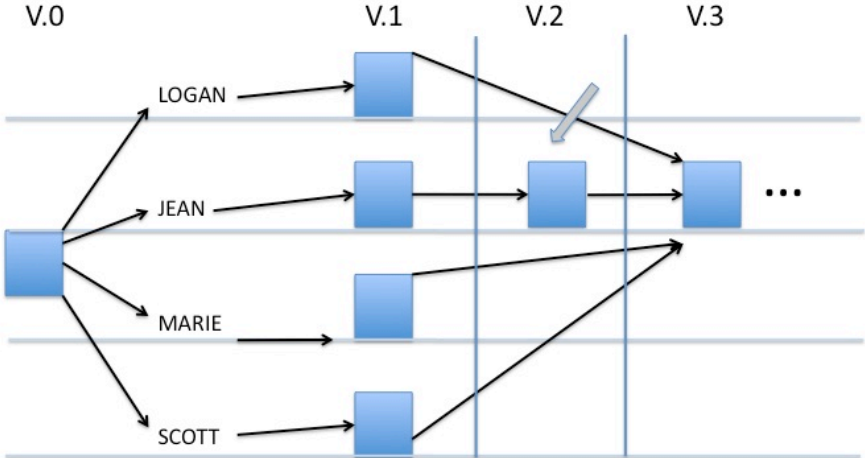
Merging (cont)

- Merge issues depend on nature of the revisions
 - Was the code that changed independent? No issues
 - Two or more may edit in the same file if the design if they are working on independent classes, methods, etc.
 - Examples of changes that cause issues:
 - Two or more edit the same line or lines (stomp on each others' code)
 - Someone edits a declaration that is included by others, e.g. changes arguments, changes the type of a variable...
 - Two or more edit within the same driver. Or this could merge cleanly but result in code that does not work
 - Again: good modularization
 - Add pair programming to use of the version control system
- Finally, when issues do need **hg resolve**: if they are complex, one team member at a time should resolve and communicate to the others. Why?

Finally....

Multiple Developers – Bigger Teams

- Note: can work at independent rates



Multiple Developers – Sub-Team Merges

