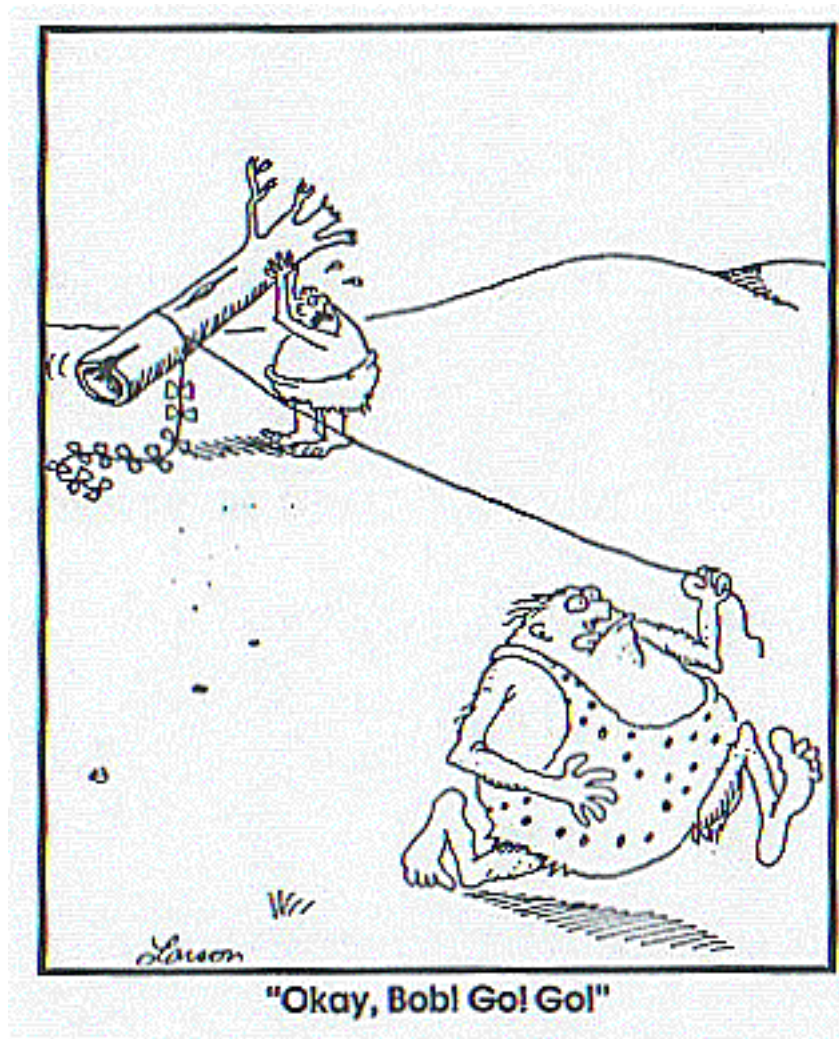
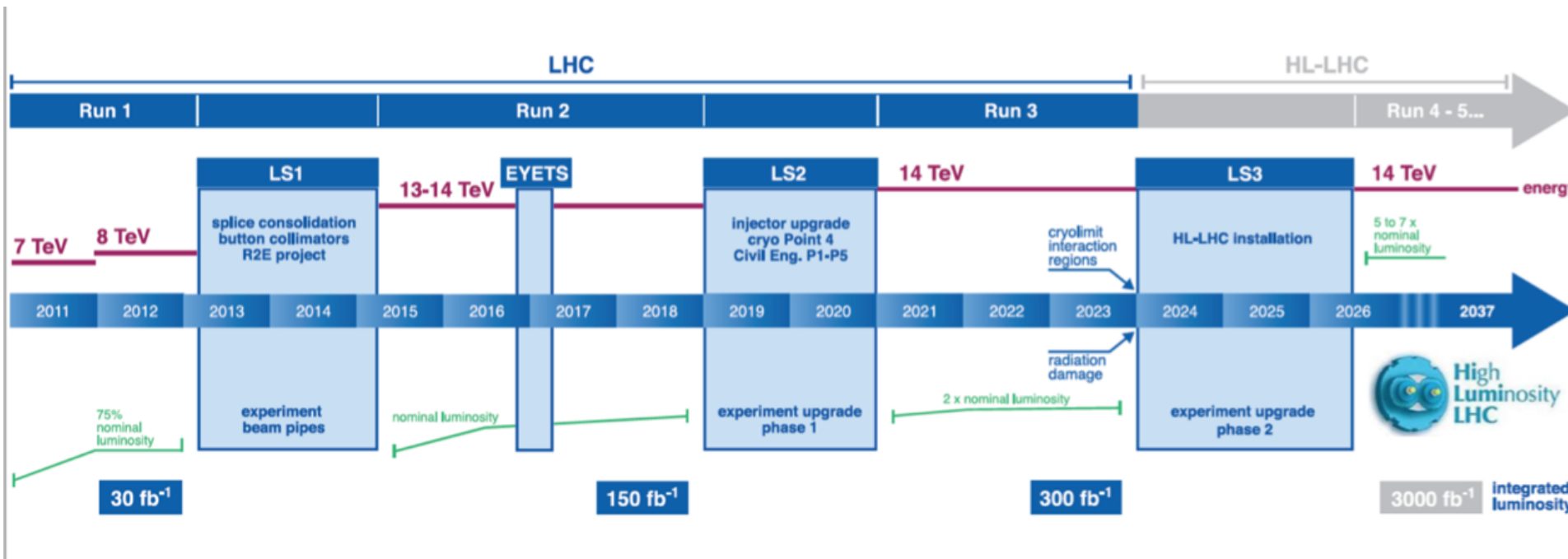


Big things are different from small things



The life time of HEP software

Software is a long-term commitment



Many releases of the software are needed over its lifetime to fix bugs, add new features, support new platforms etc

Can't technology save us?

We've built a series of ever-larger tools to handle large code projects:

Git for controlling and versioning code

Tools for building “releases” of systems

Tools for “configuration management”



More

But we struggle against three forces:

- We're always building bigger & more difficult systems
- We're always building bigger & more difficult collaborations
- And we're the same old people

Net effect: We're always pushing the boundary of what we can do

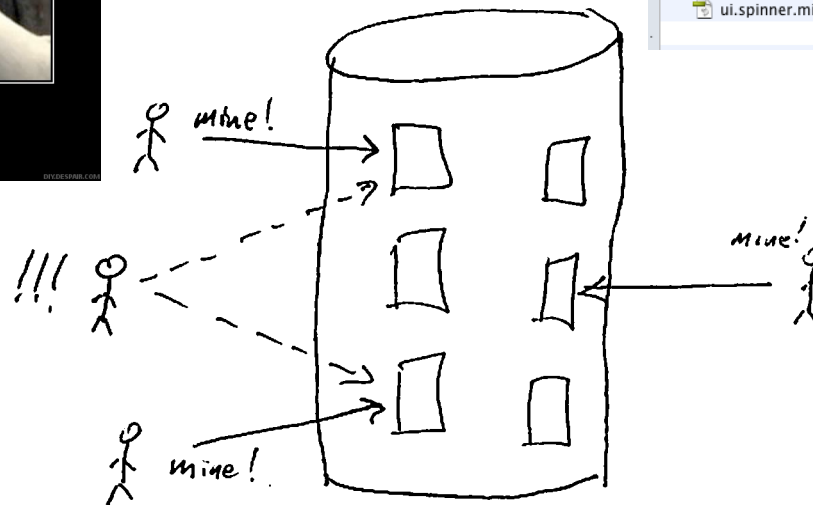
Stupidity got us into this mess; why can't it get us out? - Will Rogers

How we got here:

First, you just wrote a big program
But soon it was so big you wanted help
So you broke it into pieces/files/modules
But how do you share work on those?



Name	Date Modified	Size
images	Today, 7:31 AM	--
jquery-1.6.4.min.js	Today, 7:31 AM	94 KB
jquery-ui-1.8.16.custom.css	Today, 7:31 AM	37 KB
jquery-ui-1.8.16.custom.min.js	Today, 7:31 AM	213 KB
jquery.webforms2.js	Today, 7:31 AM	25 KB
modernizr-1.7.js	Today, 7:31 AM	16 KB
placeholder-jquery-min.js	Today, 7:31 AM	4 KB
placeholder-min.js	Today, 7:31 AM	4 KB
ui.spinner.css	Today, 7:31 AM	4 KB
ui.spinner.min.js	Today, 7:31 AM	12 KB



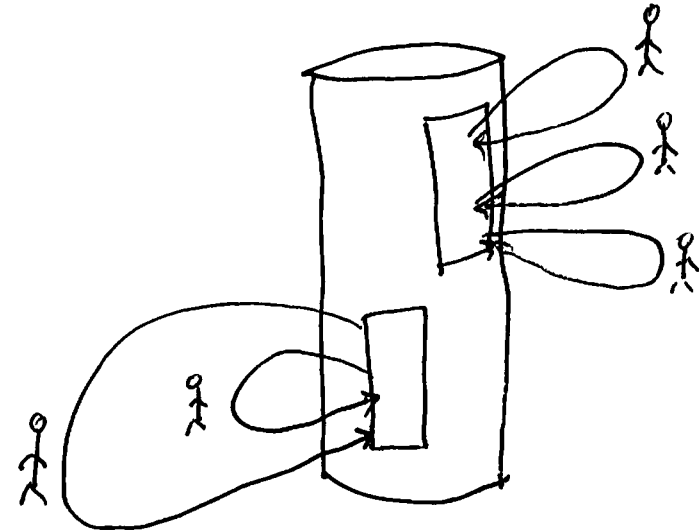
Version Control Systems (Hg, SVN, Git)

As systems & collaborations grow, efficiency goes down

“Version” idea: Track changes from one version to next

More

Anybody can get a specific set of source



Big advantage: checkout is not exclusive

- More than one developer can have the same file checked out
- Developers can control their own use of the code for read, write
- Changes can come from multiple sources
- Tool handles (most) of the conflict resolution

Scaling is still an issue

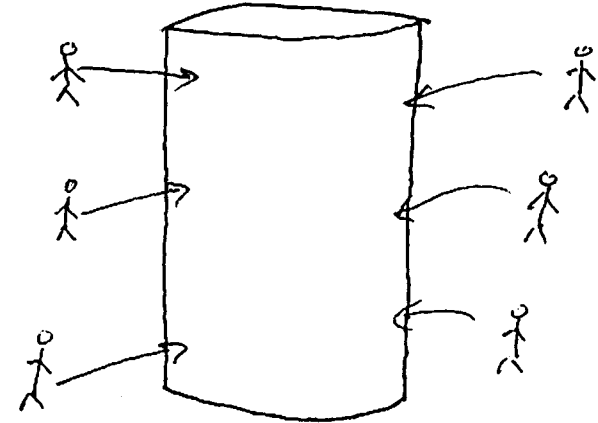
Everybody is sharing a single repository

Every commit is immediately visible to everybody else

More

Development stands on shifting sand

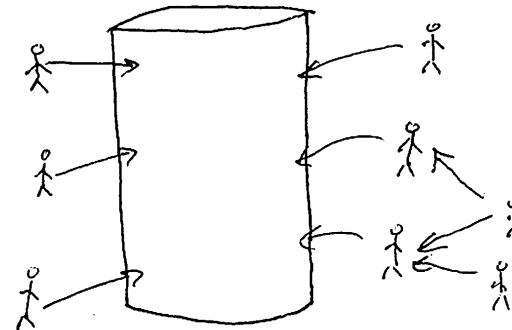
Detailed records, but little understanding



Workarounds!

External record keeping tools

Package Coordinators



Issue with this arise at large & small level

At the level of developers and contributions, needed way to manage this

- Both tools and procedures

We'll be discussing & exercising git as typical tool

Individual collaborations have their own ways of sharing info



More

At the collaboration leveled, need procedures to ensure it all works

- “Nightly builds”

Now common in HEP - Gives early feedback on consistency problems

- “Continuous Integration”, including automated testing

Only works when people actually integrate early and often

- Reduces problems, but integration is still a lot of work

When Boeing wanted to design the 747, they had two choices:

1. Hire “SuperEngineer”, who could do it alone
2. Hire 7,200 engineers and organize them to cooperate

Which did they choose?

Why?

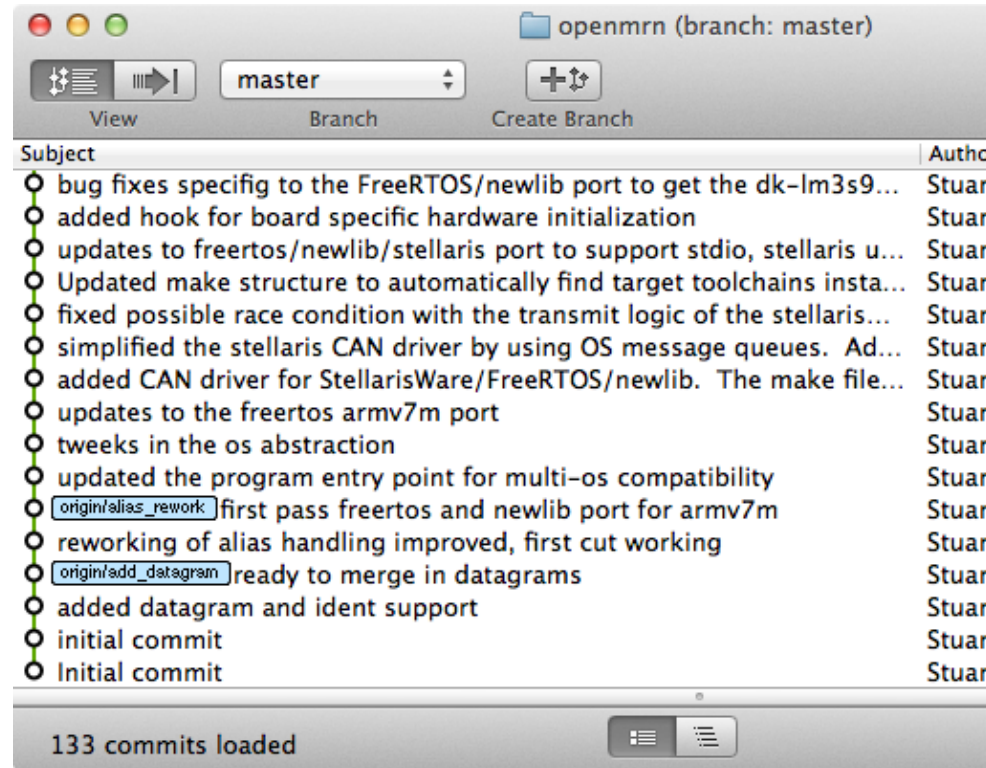
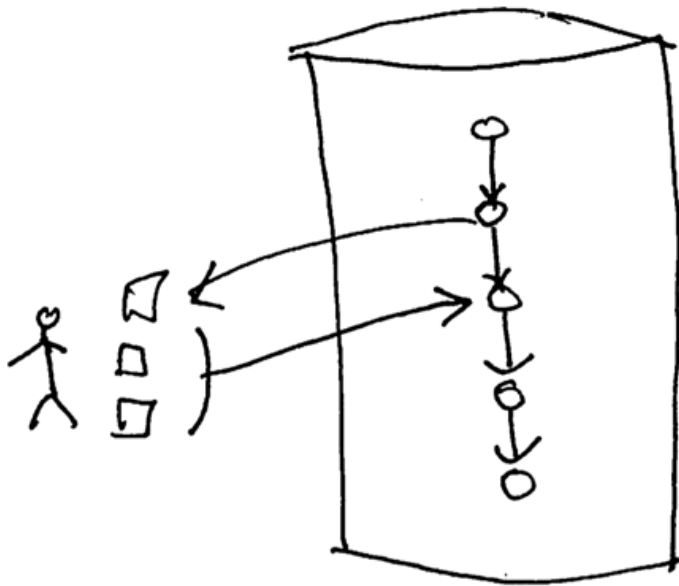
What can we learn from this?





At first, Git looks like a simple file system...

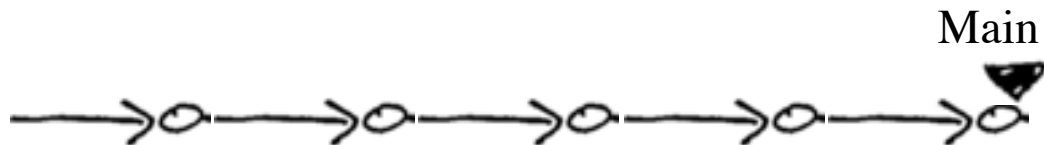
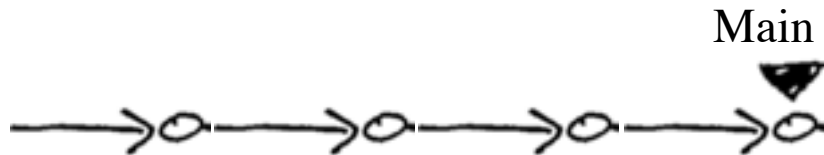
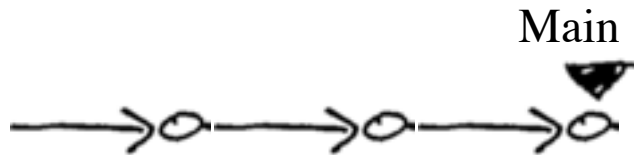
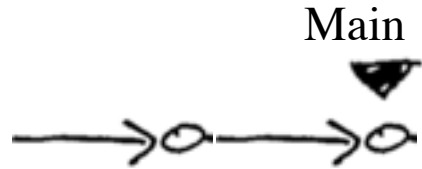
You bring out a copy, work on it, and commit
Git repository contains all that history

[More](#)


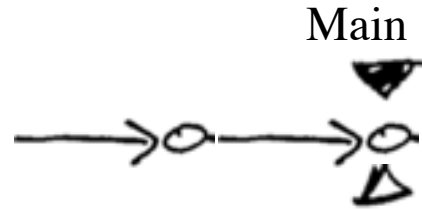
“Scratchpad” idea lets you control what you commit: **Shaping the story**

[More](#)

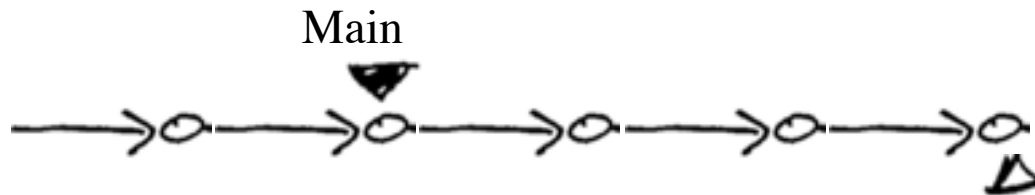
Committing to the Main Branch



Committing on a Branch and Merging to Main



WorkBranch



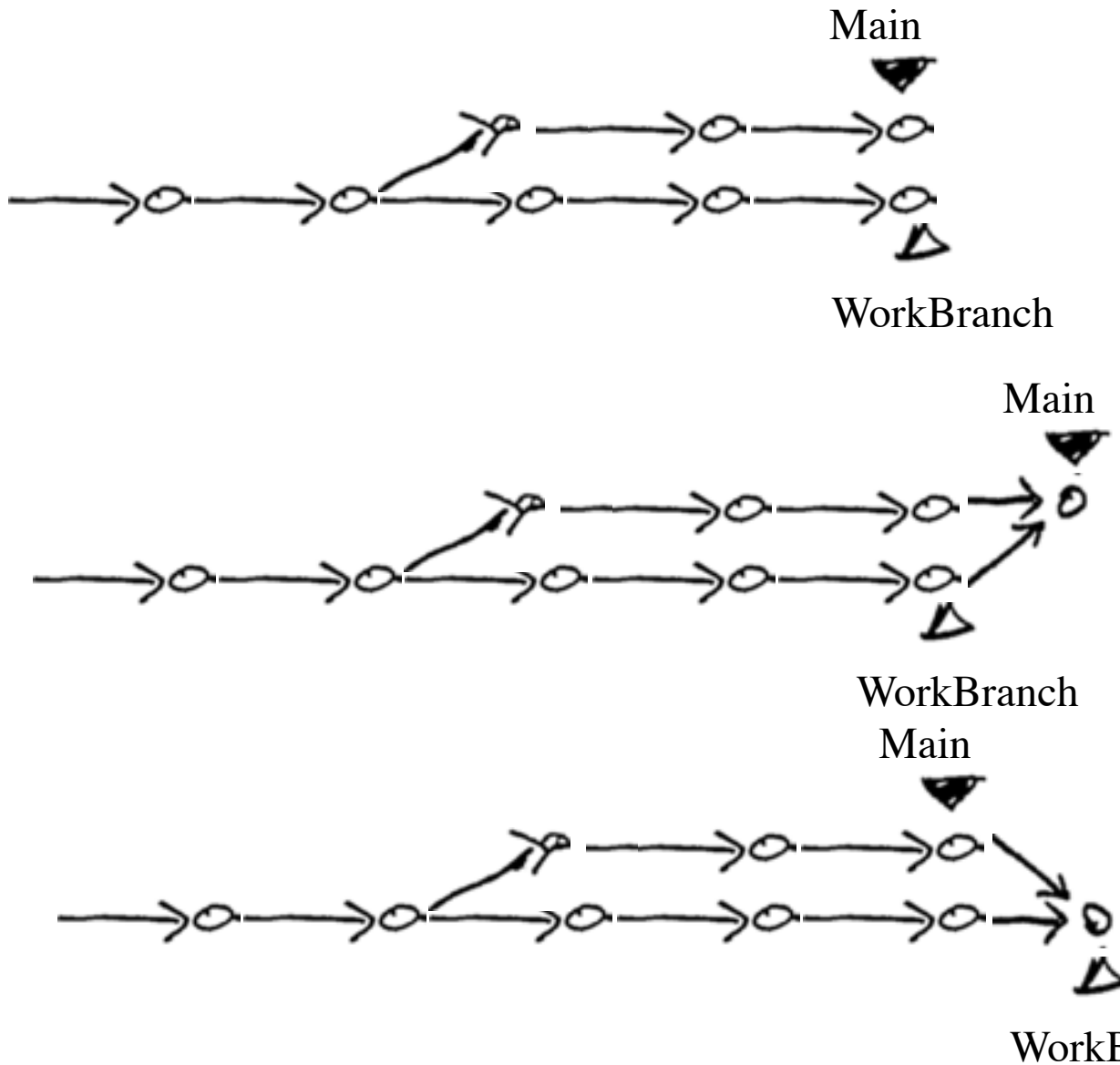
WorkBranch



WorkBranch

“Fast forward” form of merge

Committing on a Branch and Merging to Main



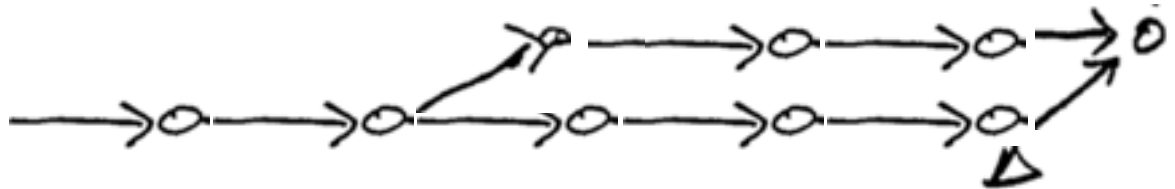
Key concept:
merge commits

Committing on a Branch and Merging to Main



Branch

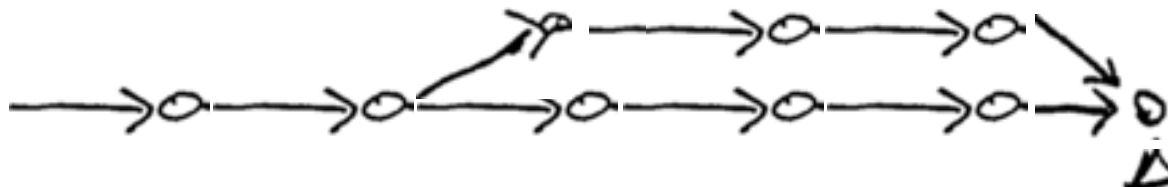
Main



WorkBranch

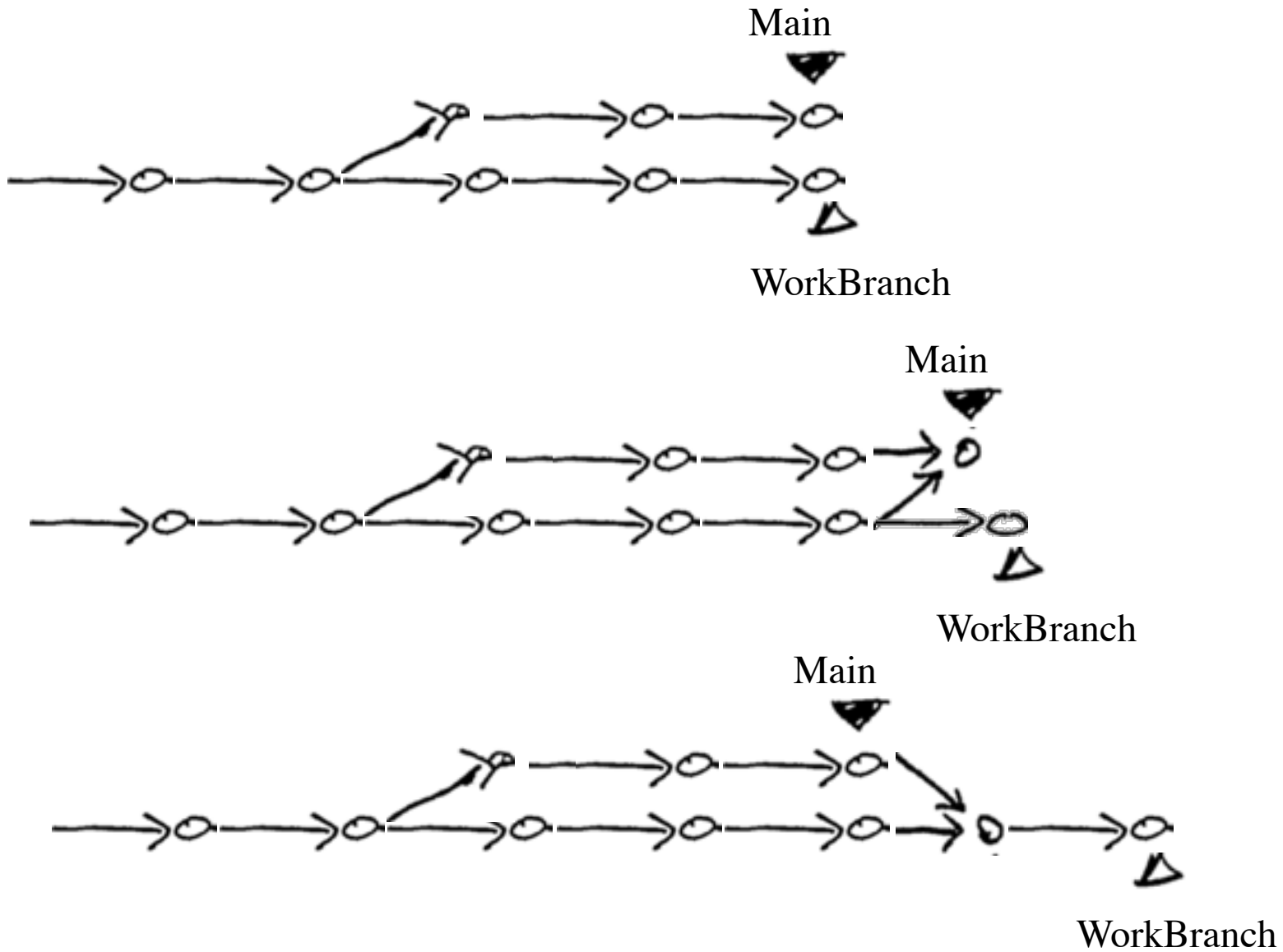
Main

Key concept:
merge commits



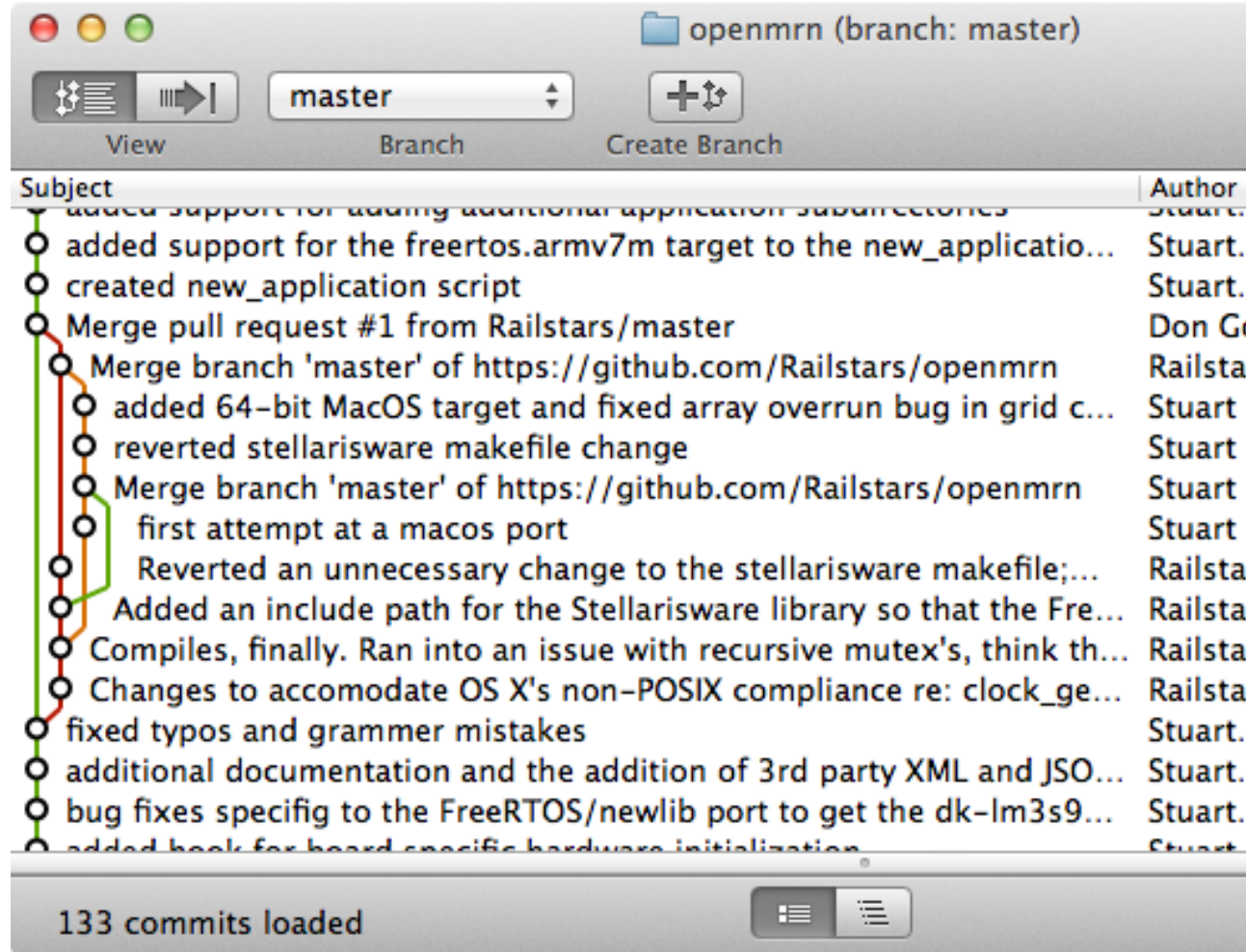
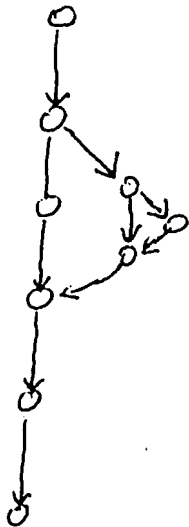
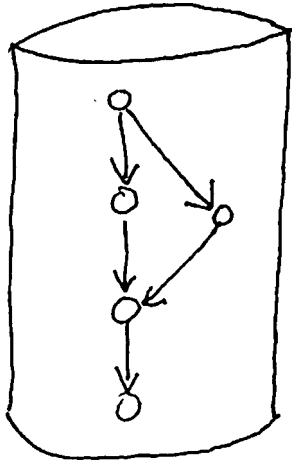
WorkBranch

Committing on a Branch and Merging to Main



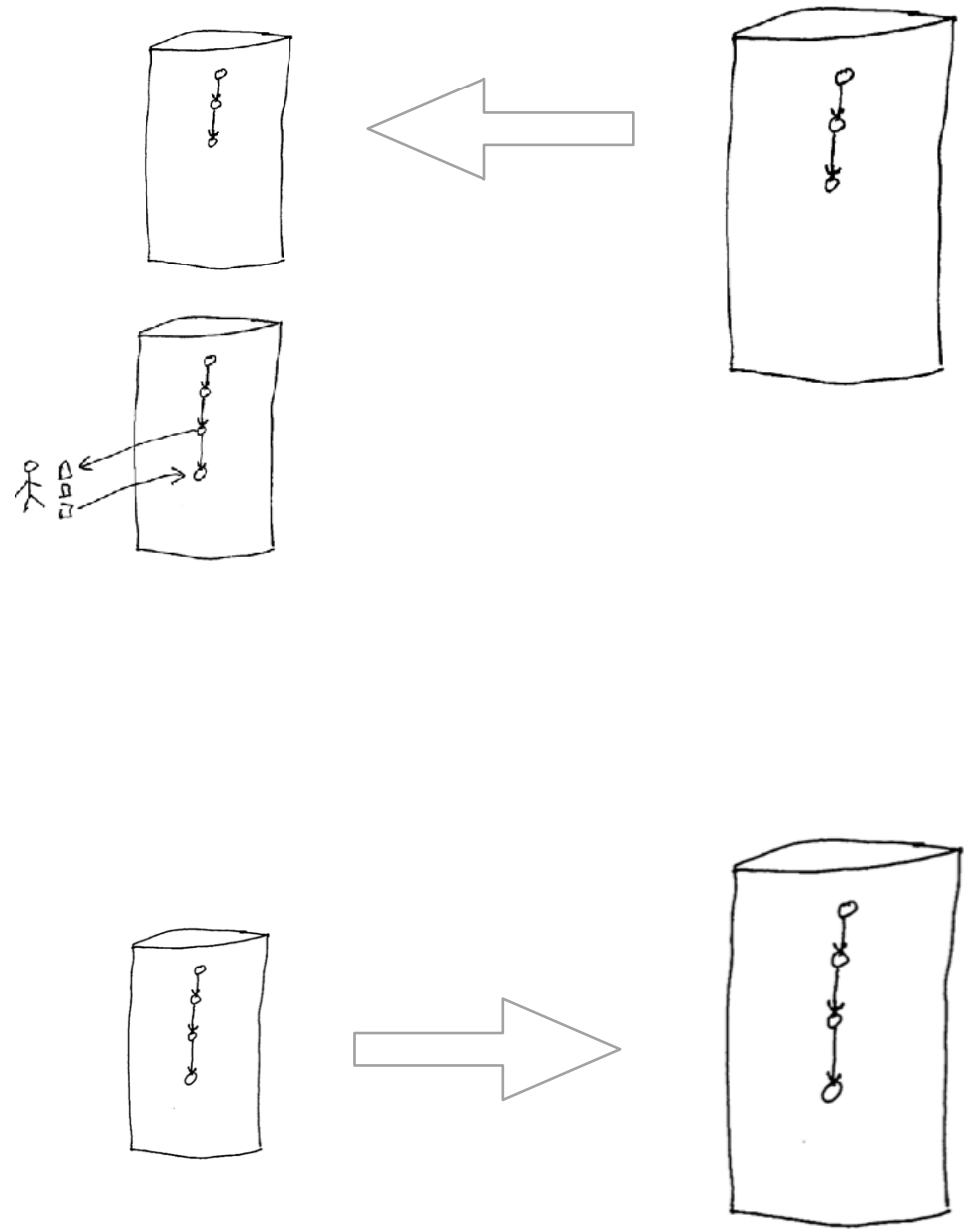
Merging

Because Git focused on commits, not on single file versions,
 you get powerful merging



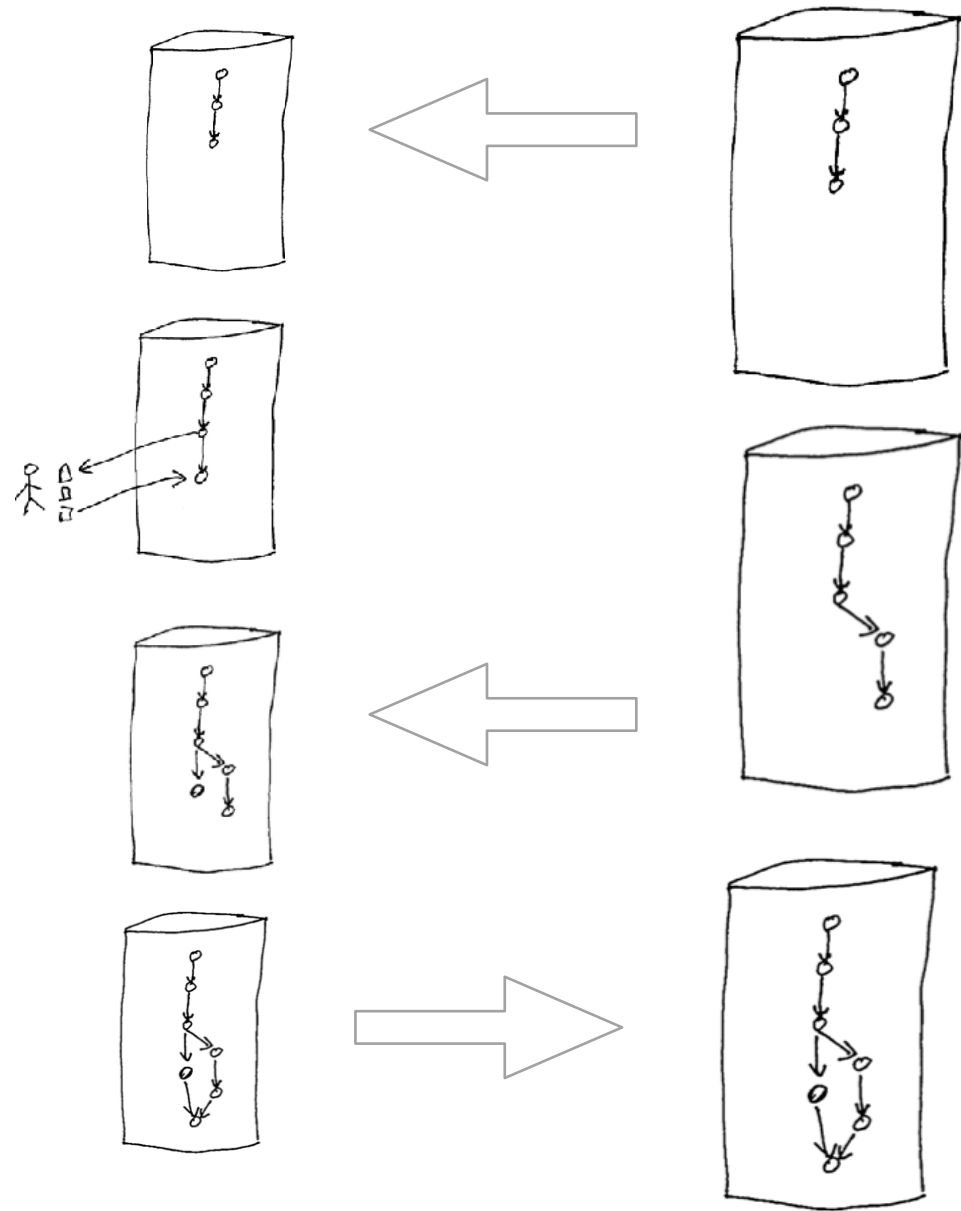
Multiple repositories with easy transfer of commits between

More

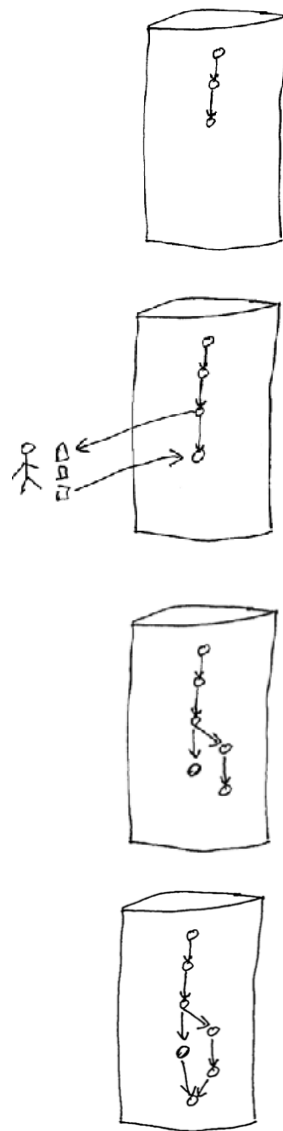


More than just mirroring

More



More than just mirroring



THIS IS GIT. IT TRACKS COLLABORATIVE WORK ON PROJECTS THROUGH A BEAUTIFUL DISTRIBUTED GRAPH THEORY TREE MODEL.

COOL. HOW DO WE USE IT?

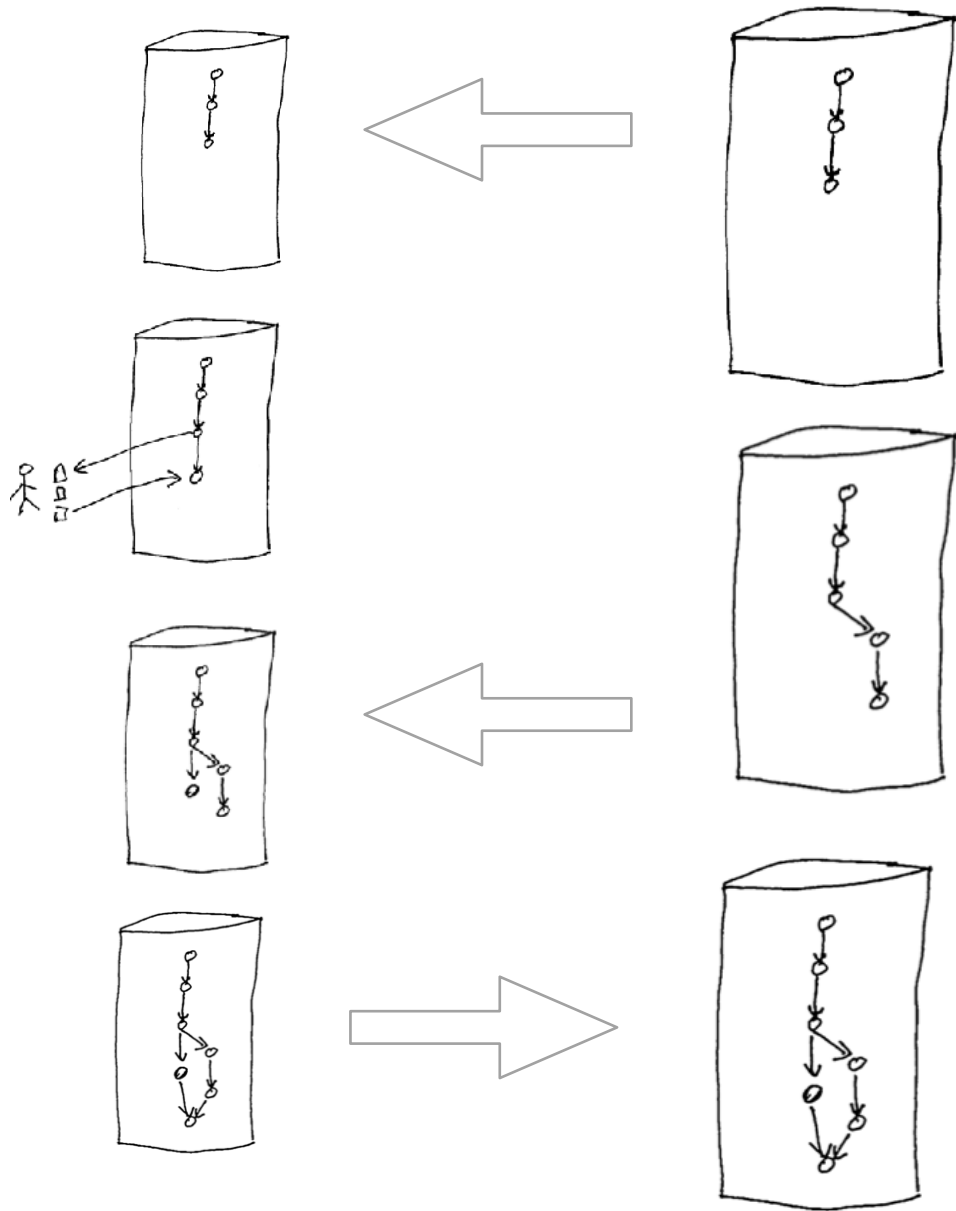
NO IDEA. JUST MEMORIZE THESE SHELL COMMANDS AND TYPE THEM TO SYNC UP. IF YOU GET ERRORS, SAVE YOUR WORK ELSEWHERE, DELETE THE PROJECT, AND DOWNLOAD A FRESH COPY.



More

More than just mirroring

More

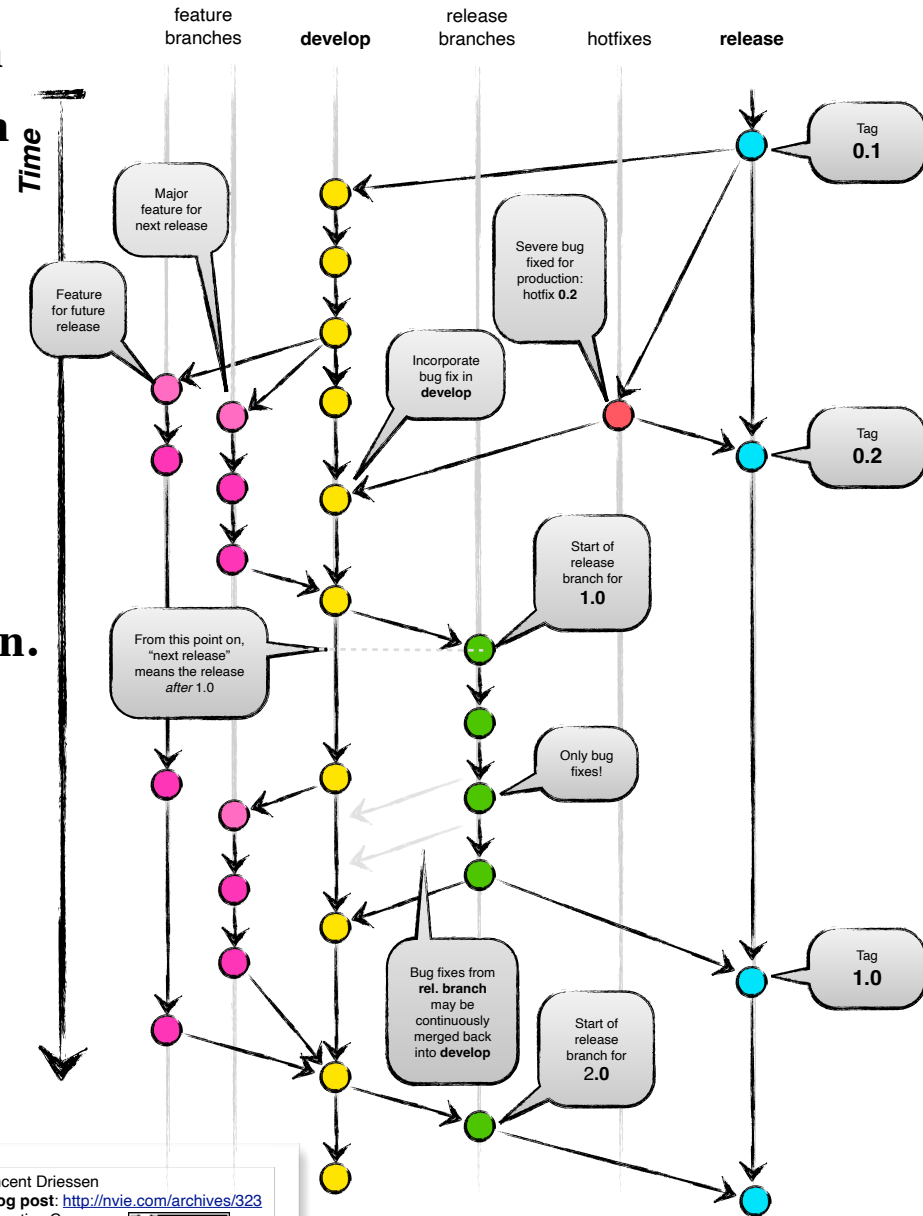


Branches are key

- Develop on a separate branch
- Future Big Feature on branch
- And another one for ll work
- Pays off for bug fix!
- Git merge to get fix across
- Feature done, merges in
- New branch holds release
- and its inevitable fixes
- until merge and release main.
- Meanwhile, work proceeds
- And the process repeats

Keys: cheap branches,
reliable merges

Gives understandable story?



Author: Vincent Driessen
 Original blog post: <http://nvie.com/archives/323>
 License: Creative Commons 

Using all that history:

My feature broke between 0.1 and 1.0

Which commit broke it?

“git bisect” works through the graph

Was it in 0.2? No?

Was it in merge before the release branch? Yes

....

I found a bug in a specific commit SHA

Which releases does it affect?

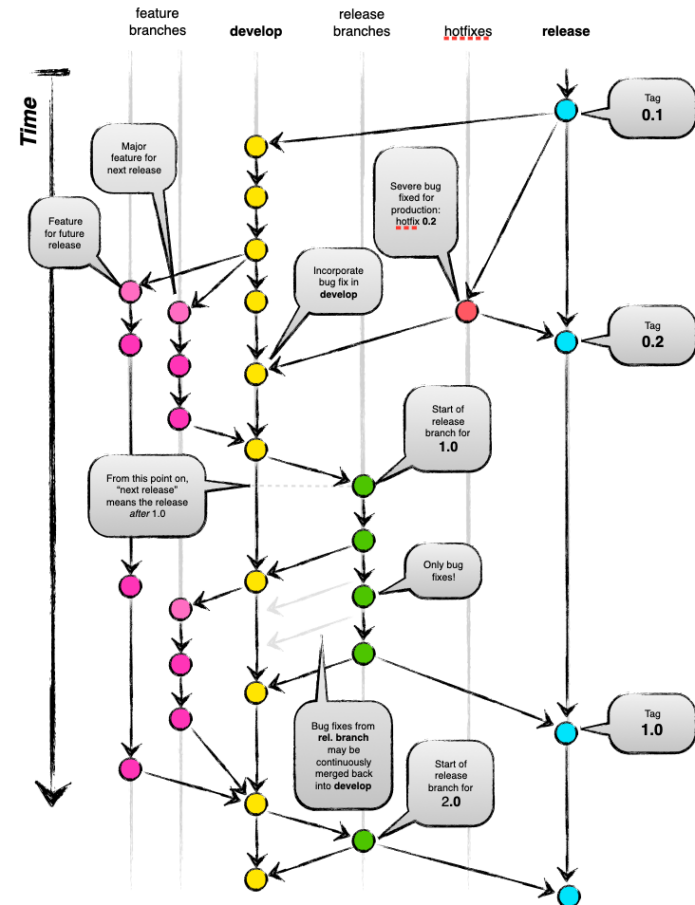
What’s not affected?

“git diff tag1.0...SHA” to see if included

“git log” and “git revlog” explore history

Graphical representations can help a lot

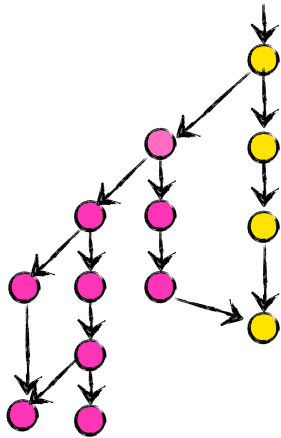
gitk, gitg tools



Complex! Linear history in repository would resolve these much easier

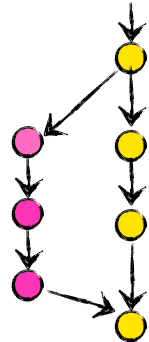
Git Rebase: An Editor for the Story

Finished difficult development task, after several dead ends, lots of little bits of progress & dead ends

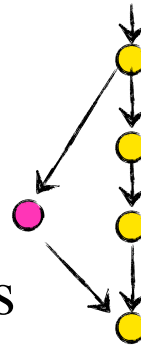


More

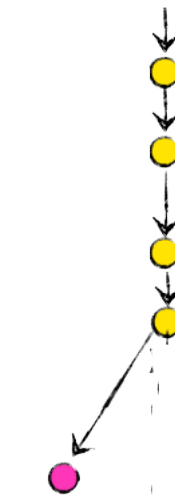
Deleting only gets you so far



“Squashing” commits

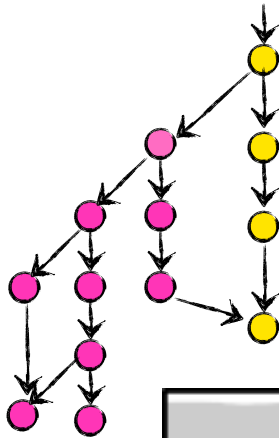


“Rebase” operation



Fast-forward merge

Git Rebase: An Editor for the Story



Finished difficult development task, after several dead ends, lots of little bits of progress & dead ends

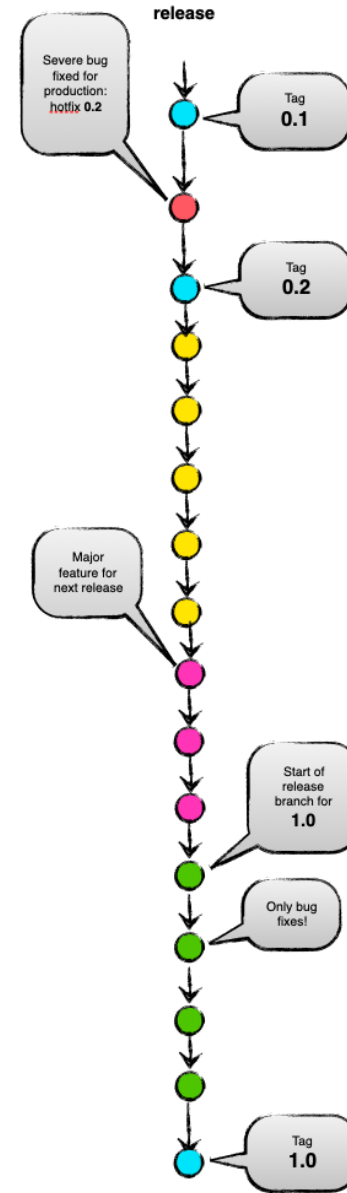
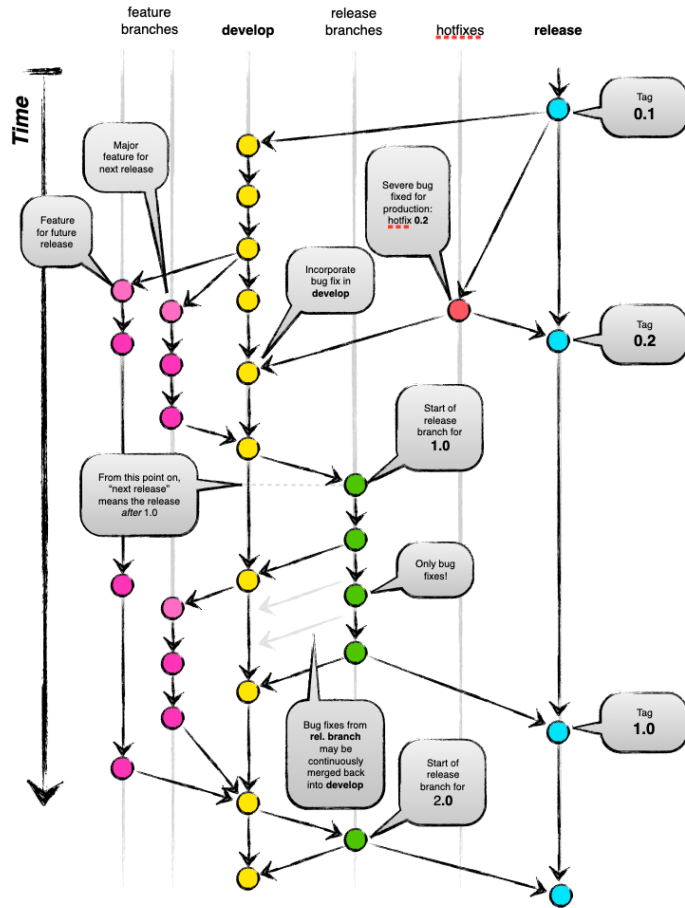
More

	COMMENT	DATE
○	CREATED MAIN LOOP & TIMING CONTROL	14 HOURS AGO
○	ENABLED CONFIG FILE PARSING	9 HOURS AGO
○	MISC BUGFIXES	5 HOURS AGO
○	CODE ADDITIONS/EDITS	4 HOURS AGO
○	MORE CODE	4 HOURS AGO
○	HERE HAVE CODE	4 HOURS AGO
○	AAAAAAAAA	3 HOURS AGO
○	ADKFJSLKDFJSDKLFJ	3 HOURS AGO
○	MY HANDS ARE TYPING WORDS	2 HOURS AGO
○	HAAAAAAAAAANDS	2 HOURS AGO

AS A PROJECT DRAGS ON, MY GIT COMMIT MESSAGES GET LESS AND LESS INFORMATIVE.

Linear history:

Using rebase and fast-forward merges:



You want me to trust how many people?

How do you give 6,000 people access to a central repository?

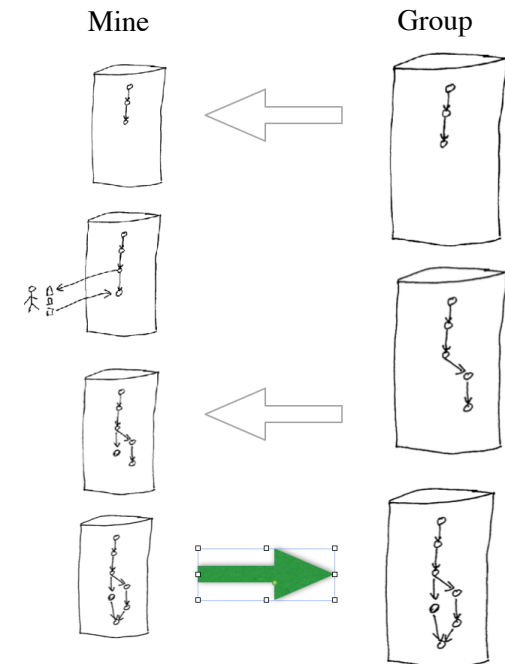
Use a distributed repository and “pull requests”

Git-based developers have a full local repository

Commits have full context

“Push” moves all that to target

A “pull request” sends all that to somebody at the target, who can accept or not



When accepted, the merge is completed & both repositories in sync

(Pull requests rarely rejected outright - usually it’s “fix these things and resend”)

Strong tools exist to make pull requests easy: CI test results, etc automated

More

Life Cycle of a Pull Request

Bob is working on his laptop, and commits another change locally:

```
% git commit -m"Cover rest of classes" help/en/html/tools  
[ctc-tools 79c28b4c93] Cover rest of classes  
1 file changed, 14 insertions(+)
```

Life Cycle of a Pull Request


Bob is working on his laptop, and commits another change locally:




```
% git commit -m"Cover rest of classes" help/en/html/tools
[ctc-tools 79c28b4c93] Cover rest of classes
1 file changed, 14 insertions(+)
```







He's ready for that work to be reviewed, and wants to move it to a repository that's always online:

```
% git push
Counting objects: 8, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 1.07 KiB | 0 bytes/s, done.
Total 8 (delta 6), reused 0 (delta 0)
remote: Resolving deltas: 100% (6/6), completed with 6 local objects.
To https://github.com/bobjacobsen/JMRI.git
3d35322e43..79c28b4c93 ctc-tools -> ctc-tools
```






Life Cycle of a Pull Request

 **bobjacobsen / JMRI**
 forked from JMRI/JMRI



 Unwatch ▾ 1
  Star 0
  Fork 90

 Code
  Pull requests 0
  Projects 0
  Wiki
  Settings
  Insights ▾



<http://jmri.org> Edit
 Add topics


 40,216 commits
  181 branches
  185 releases
  53 contributors

Your recently pushed branches:

 **ctc-tools** (7 minutes ago)  Compare & pull request

Branch: **master** ▾
 New pull request
 Create new file
 Upload files
 Find file
 Clone or download ▾

This branch is 3 commits ahead of JMRI:master.  Pull request  Compare


 **bobjacobsen** Merge branch 'master' of <https://github.com/JMRI/JMRI>
 Latest commit 87241d2 18 minutes ago



Open a pull request

Create a new pull request by comparing changes across two branches. If you need to, you can also [compare across forks](#).

...




✓ **Able to merge.** These branches can be automatically merged.



Update CTC tools based on user feedback

Write Preview


AA B i “ <> 🔗 ⋮ ⋮ ⋮ ↶ @ 🌟

- Better handling of timing
 - Locks can handle multiple segments
 - Improved documentation
- 

Attach files by dragging & dropping or [selecting them](#).

Allow edits from maintainers. [Learn more](#)



Reviewers 

No reviews—request one

Assignees 

No one—assign yourself

Labels 

None yet

Projects 

None yet

Milestone 

No milestone











↻ 15 commits


📄 20 files changed

💬 0 commit comments

👤 1 contributor

📅 Commits on Jul 08, 2017

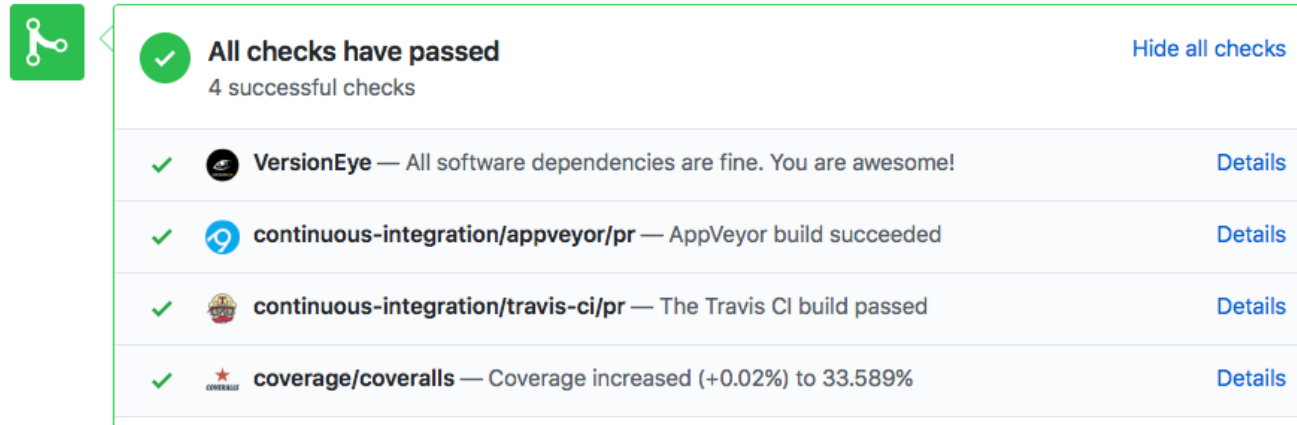
	 bobjacobsen	Merge branch 'master' into ctc-tools	d8b4fbf
	 bobjacobsen	Merge branch 'sensor-scripts' into ctc-tools	8dFd297
	 bobjacobsen	current sequences	d3cf209
	 bobjacobsen	log lock fails	acc23cd
	 bobjacobsen	sequencing and comments	26f3506



Life Cycle of a Pull Request

Once created:

Continuous integration tests are run




A screenshot of a pull request interface showing the status of continuous integration (CI) checks. At the top, a green checkmark icon is followed by the text "All checks have passed" and "4 successful checks". To the right of this text is a link that says "Hide all checks". Below this, there is a list of five individual checks, each with a green checkmark icon and a "Details" link to its right:

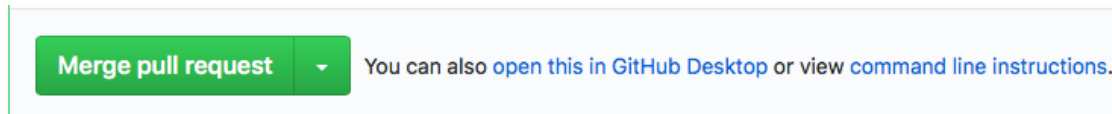
- VersionEye** — All software dependencies are fine. You are awesome!
- continuous-integration/appveyor/pr** — AppVeyor build succeeded
- continuous-integration/travis-ci/pr** — The Travis CI build passed
- coverage/coveralls** — Coverage increased (+0.02%) to 33.589%

Reviews happen

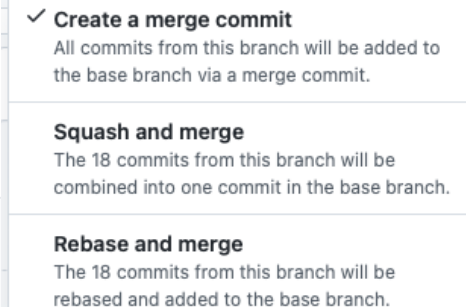
Merge checks are done

 **This branch has no conflicts with the base branch**
 Merging can be performed automatically.

And finally, somebody with authorization can click this:



A screenshot of a pull request interface showing a green button labeled "Merge pull request" with a dropdown arrow. To the right of the button, there is text that says "You can also [open this in GitHub Desktop](#) or view [command line instructions](#)."



A screenshot of a dropdown menu showing three options for merging a pull request:

- ✓ Create a merge commit**
All commits from this branch will be added to the base branch via a merge commit.
- Squash and merge**
The 18 commits from this branch will be combined into one commit in the base branch.
- Rebase and merge**
The 18 commits from this branch will be rebased and added to the base branch.

to complete the merge onto the desired branch in the main repository.

Three choices for merging PRs:

Merge Commit

- **Contains the entire development history from the merged branch**
- **Usually a merge commit, sometimes fast-forward**
- **For many, this is the default approach**

Squash and Merge

- **Merges entire change as a single commit**
- **Usually a merge commit, sometimes fast-forward**
- **Contains the entire change in a single commit**
 - **Optionally, a more comprehensive, holistic comment**

Rebase

- **Puts a single commit on the end**
- **Always a fast-forward commit**
- **Contains the entire change in a single commit**
 - **Optionally, a more comprehensive, holistic comment**

How do you use this all?

Individually:

Use it to work independently

Both of others, and of yourself!

Collaborate on intermediate results

Clean branches easy to share: “Try bobj/FixIssue10343”

Shape your work result to make it understandable

Comments, squashing, comments, rebasing as tools

Integrate early and often!

Pull “main” and make sure work is still OK

For a collaboration project:

Help people work at the scales they need to

Individually, in small groups, large groups, ...

Control how code is added/updated

Shaping contents of common development, releases

Make the contents understandable

Tags, known branching / linear history

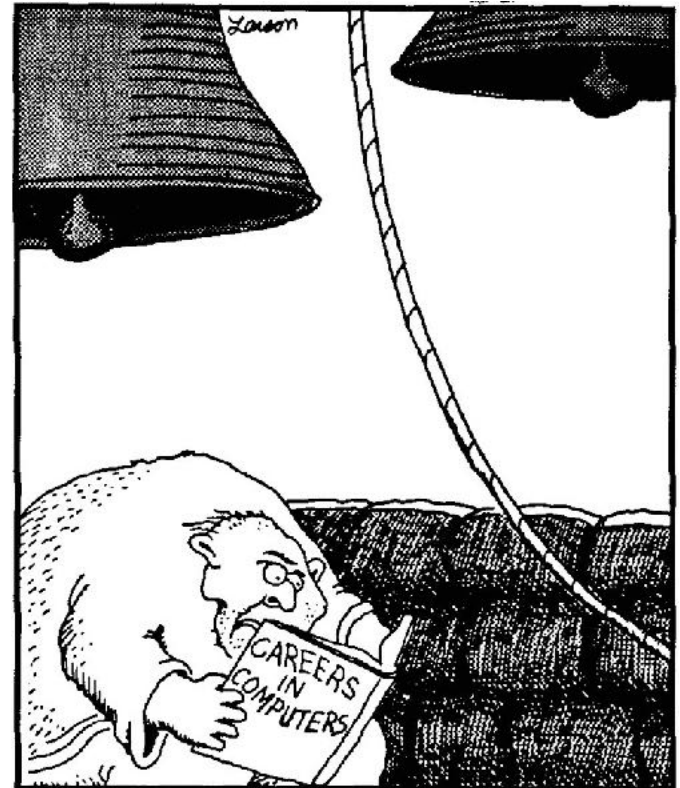
Series summary

Software engineering is the art of building complex computer systems

It's ideas and techniques spring from our need to handle size & complexity

As you do your own work & develop your own skills, consider:

- How your effort effects or contributes to things 10X, 100X, 1000X larger
- How you'll do things different/better when it's your problem



Questions? jacobsen@berkeley.edu

Exercises

Test Frameworks

Memory Issues

Code Management



Instructions to get started on Indico (Tools & Techniques E1)

<https://indico.cern.ch/event/1376644/contributions/5785497/>

More

You'll work in pairs. Try to find somebody with complementary skills!

Learn about each topic, spend more time on the ones that interest you.

Speed is not the issue: no reward for first done, no complaint about last.

Think about what you're doing: There are larger lessons to be found!