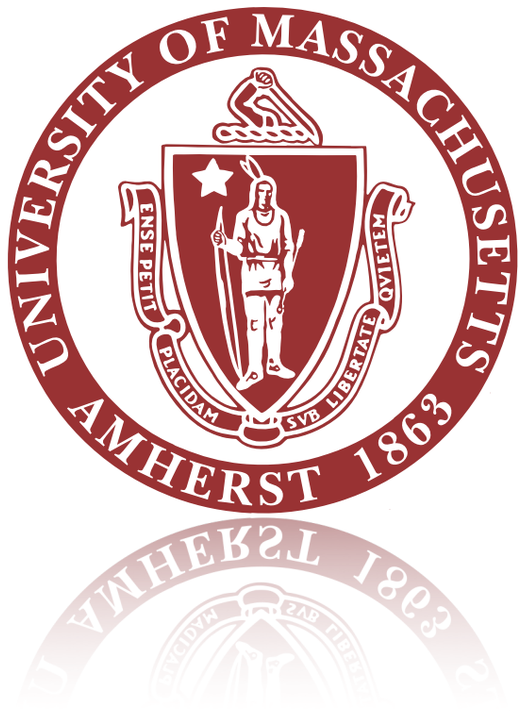


Git for ATLAS offline

Edward Moyse
o.b.o. ATLAS Git study group
with thanks to many others

Introduction

- One of the recommendations of the Software Build and Infrastructure review was the formation of a GIT Study group within ATLAS

> Git is the currently favored source code management system (SCM) at CERN with SVN support currently scheduled to stop around the end of Run-2. It is thus almost mandatory for ATLAS to migrate to git. This opens up the possibility to use a modern collaborative software platform like gitlab (or github) to manage code reviews and code change collection. While git has many advantages over SVN, it potentially represents a significant change in the developer workflow and needs to be carefully prepared. A study group should be tasked with finding a workable and timely migration strategy from SVN to git. This includes evaluating whether migrating from individual versioned packages to the "pull request workflow model" that comes naturally with git/gitlab matches the needs of ATLAS and provides sufficient gains, for instance in managing the forward sweeping of changes in stable releases, to warrant a significant disruption for developers. The role of release coordinators and different release branches should be reviewed in the light of new features and workflows. We note that git's ability to tag the entire state of the repository would largely remove the need for the current tag collector and would simplify the process of building an entire offline release independently. The study group should take advantage of the experience from CMS' migration to git (http://cms-sw.github.io/index.html) and experts with git experience within the collaboration.

- Members:
  - Graeme Stewart, Walter Lampl, Goetz Gaycken, Elmar Ritsch, Emil Obreshkov, Frank Winklmeier, Stefan Stonjek, Edward Moyse
- We've been meeting since mid January, approximately every week / every other week
  - https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/GitStudyGroup

First questions and conclusions

- Is git an acceptable replacement for SVN?

First questions and conclusions

- Is git an acceptable replacement for SVN?
  - **YES!!!**

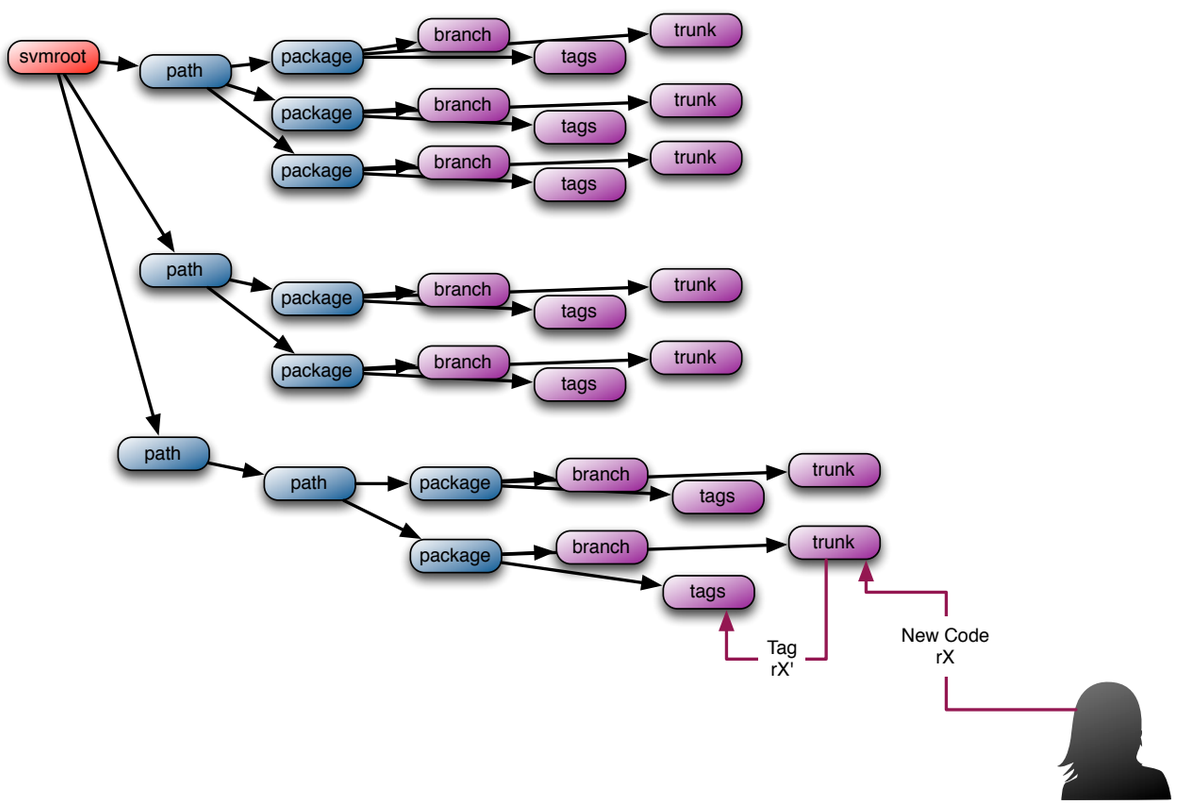
First questions and conclusions

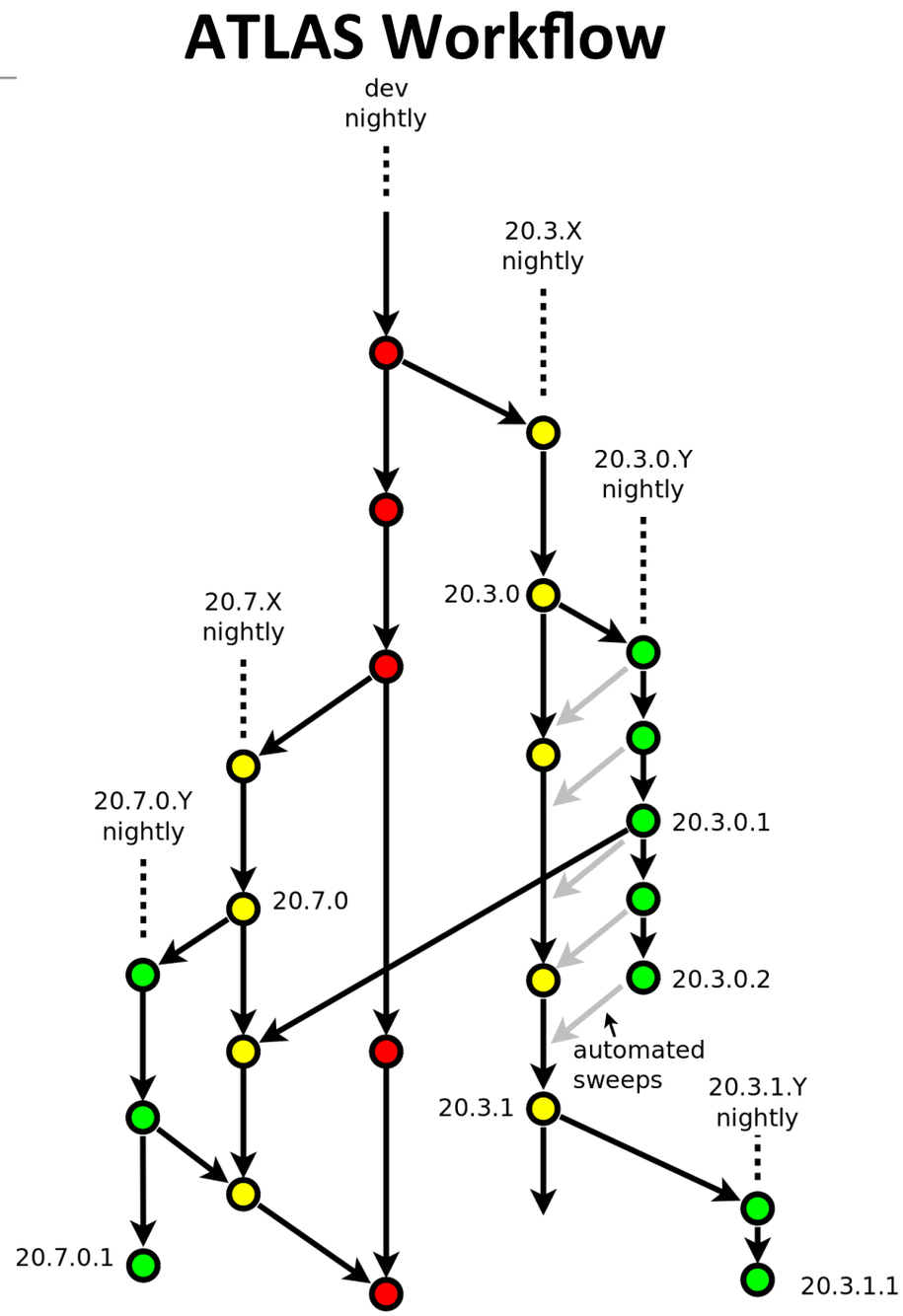
- Is git an acceptable replacement for SVN?
  - **YES!!!**

- Repository per package or for all of ATLAS offline?
  - Unless performance is impacted, one per ATLAS (still discussing … but definitely not per package)
- Github vs Gitlab
  - In the absence of any pressing reason not to, looks like we're drifting toward gitlab
  - Major advantage is CERN support (e.g. e-groups)
  - Possible disadvantage: performance (e-groups?)
- Do we import all of SVN, or just e.g. from 20.1 onwards?
  - Still being discussed. In any case the SVN repository will exist, in read-only mode.
  - Software archeology will be possible

Reminder: ATLAS releases

- ATLAS currently has many releases in use concurrently, and each release has ~2200 packages representing modules of ‘code’
  - This is partially to manage ownership and permissions
  - Cannot build release directly - need Tag Collector





Packages & import strategies

- Oleg has shown ways in which we could reproduce the current system of packages
  - e.g. make git branches representing package tags
    - https://indico.cern.ch/event/506187/
  - But unwieldy - and we (or I think, at least most of us) conclude that when we move to git we should embrace more git-like workflows.
- Do we want to import all of SVN?
  - Can git handle a repository of our size (btw we are comparable to CMS, and smaller than the linux kernel)?
  - Should we use git submodules / subtrees?
- Have two import strategies:
  - atlasgit - Imports offline SW from release 20.1.0 onwards (i.e. truncates history), veto files >100kB
  - aogtfat - full import of all packages and all files

Atlasgit results…

	Time	Imported Tags	Size (du Total)	Size (du Master branch)	time git status
Take only tags in releases + trunk	**r20 minimal**	3h30m	9140	1584MB	670MB	0.69s
Take all tags younger than the oldest tag + trunk	**r20 full**	9h45m	22880	2415MB	670MB	0.97s

ChangeLog files are **big**!

'git log' is a much better idea (provided developers make meaningful commit messages).

File Type	Files	Size (MB)	Average (kB)
.cxx	16673	139	8.4
.py	11656	96	8.2
.h	20318	57	2.8
ChangeLog	2210	28	12.7
.root	605	20	33
.txt	2075	16	7.7
.dat	578	15	26
.xml	1333	12	9.2
.java	839	8	9.5
Total	**67289**	**459**	**6.8**

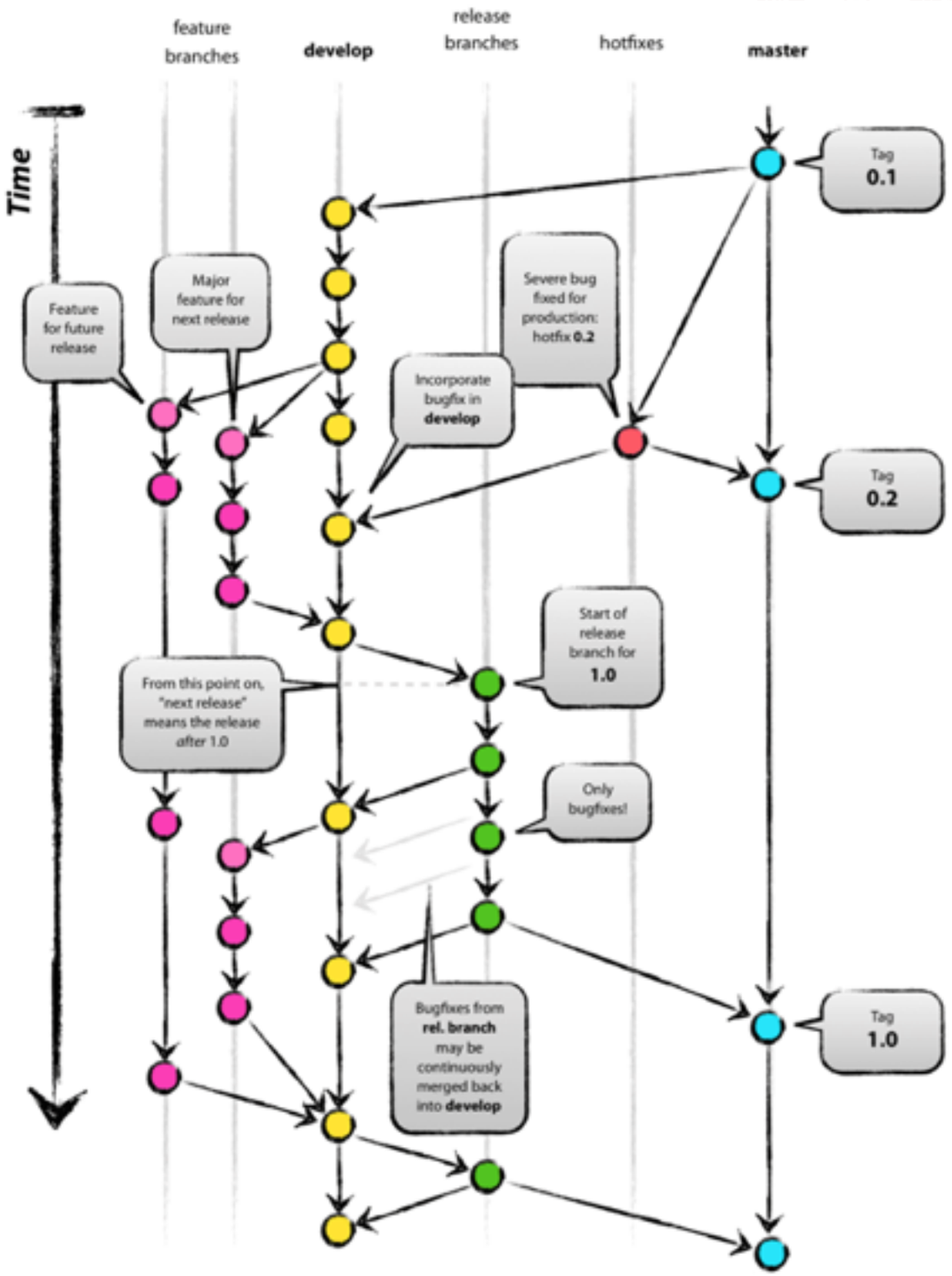
Aogtfat

Slide from Goetz

- https://gitlab.cern.ch/goetz/aogtFat
- Sizes
  - 3 Gb full SVN clone (individual package clones) including full commit history, but extra storage needed for branches/tags which would collect all packages of a release (is about ~ 200kb / releases; not included in these 3Gb)
    - 600 Mb full commit history and including tags for 1000 releases+nightlies. but excluding files >100k.
    - 200 Mb, excluding files >100k, no commit history, only importing releases from 20.1 onwards.
- [After one year of usage assuming that commits are merged in for each nightly as they are (i,e, not squashing them) the 200 Mb will presumably grow to the 600 Mb-1Gb, My estimation: growth of 200kb / nightly in average, and we have O(10-20) release branches]

Git workflows

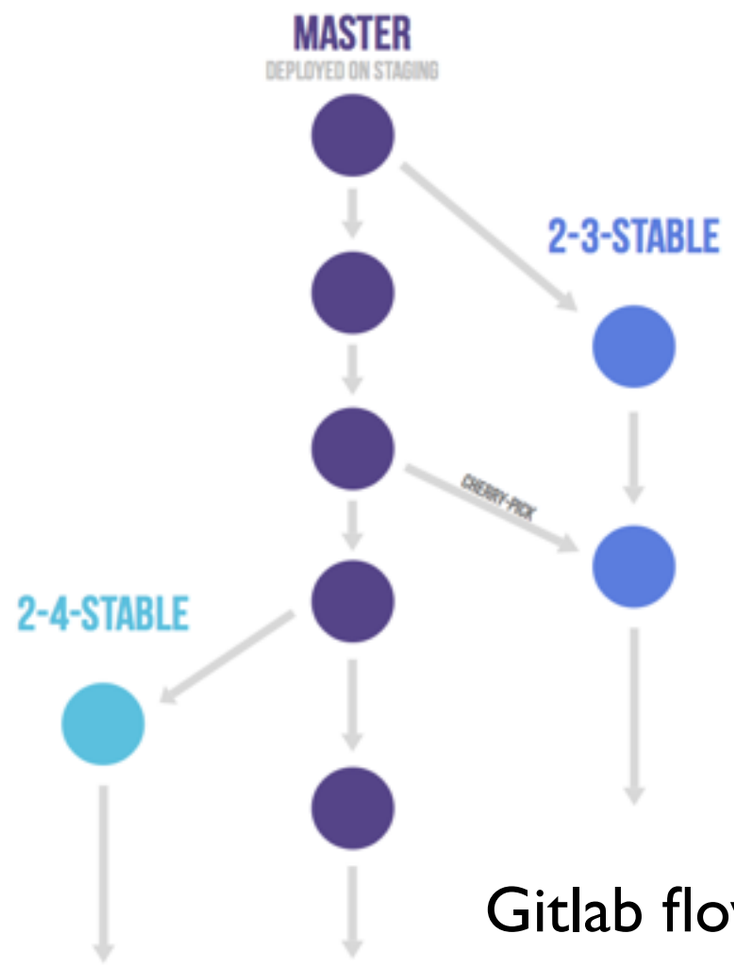
- **Git flow:**
  - Many, many branches
    - **master** branch, **feature** branches, **develop** branch, **release** branches, **hot fix** branches etc
  - Developments happen on **develop** or **feature**, and are eventually merged into **master**
  - Hot fixes are exceptions… e.g. sometimes you urgently need to fix a Tier-0 breaking bug
- **Disadvantages**
  - Complex & error prone - easy to forget to commit to master and/or develop
  - Mostly working on branches other than master, which is not what some tools expect



Git workflows (2)

- Github/Gitlab flow
  - Several variants
  - Key idea in github flow is to only have feature branches
    - Merge often, minimise amount of code lying around
- (Unfortunately) a bit too simple for us … we do need to have (many) releases
  - Can still keep basic concepts though, but extend with releases
- Documentation here:
  - http://docs.gitlab.com/ee/workflow/gitlab_flow.html#introduction

Github flow



Gitlab flow

First thoughts for ATLAS

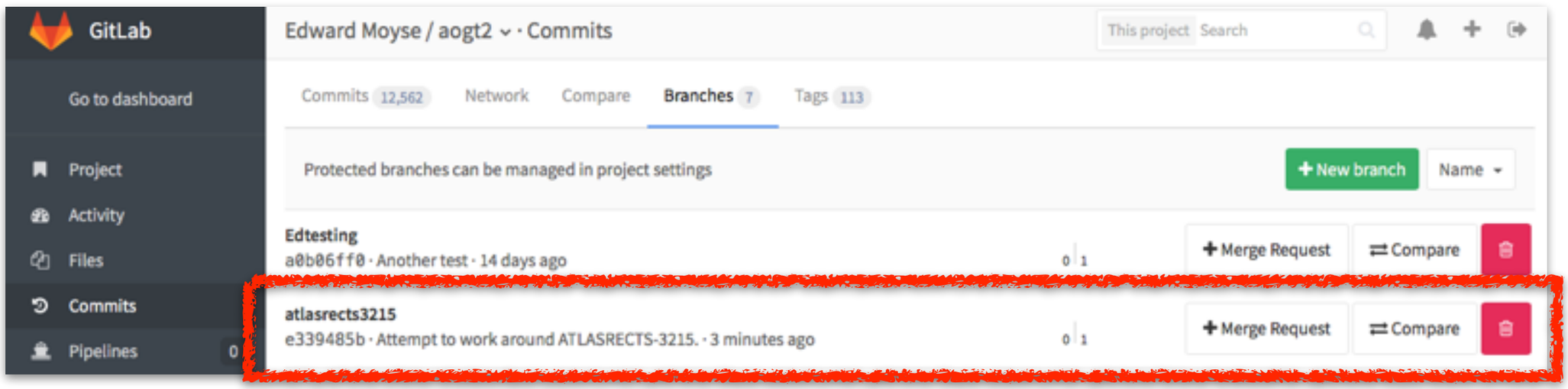
- Have locked core repository, with a limited number of ‘approvers’
  - Branch per release
- Forks
  - Normal developers fork and have their own repo for developments.
    - Branch on these repos for specific feature e.g. fixATLASRECTS6975
  - Groups might also fork, e.g. have a Tracking repo for tracking developments
    - Branch for feature e.g. GPU_tracking_developments
- Merge requests
  - On private repos - up to repo manager
  - On core repo, approvers will examine CI results and accept or reject tags

Use cases:

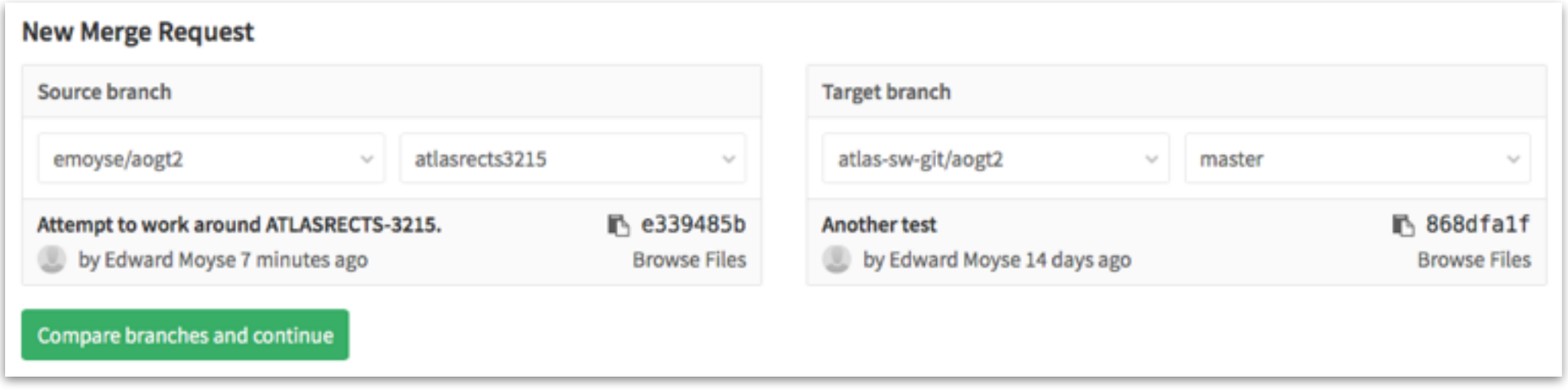
- ‘Hotfix’ to Tier-0
  - We have discovered a bug which deletes every other event if it’s a full moon.
  - Procedure
    - Developer(s) develop patch on private fork (could be personal, could be for a group).
    - After local testing, submit a merge request to core repo’s tier-0 branch
    - ‘Code guardians’ examine fix, and discuss improvements with developers
    - They push more fixes to their branch, and CI tests continue
    - ‘Code guardians’ accept change to the tier-0 release
- ‘Long term development’
  - Procedure
    - Developers create feature branch on their repo
    - Once done, request merge to master on ‘core’ repo
    - Code review, as above…

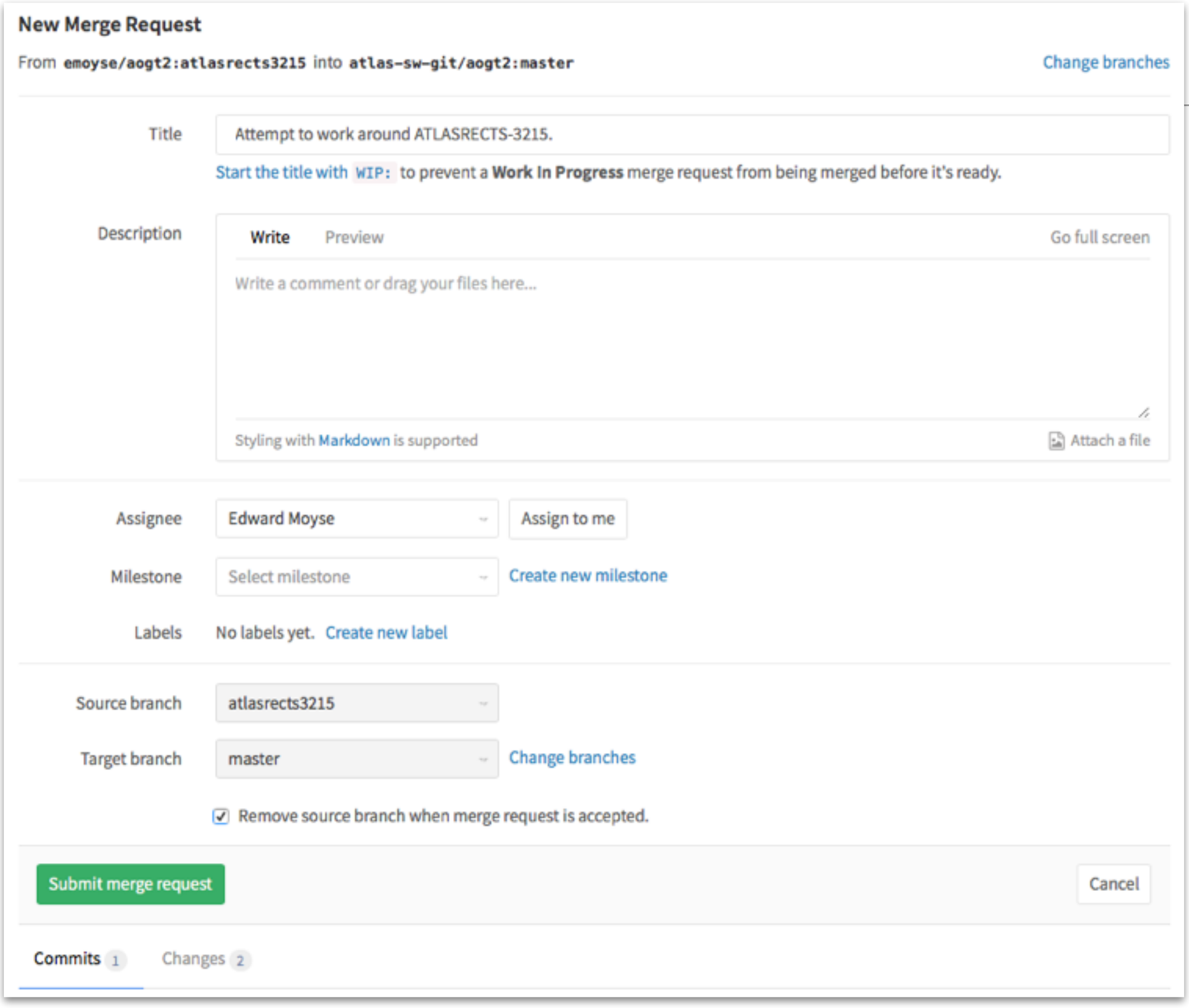
In practice …

- e.g. Fix for bug in reco nightly (in this case, an algorithm is throwing an exception):
  - Developer creates branch and makes a patch in private area.



- When ready (after CI + other tests), can request merge (in this case to master)



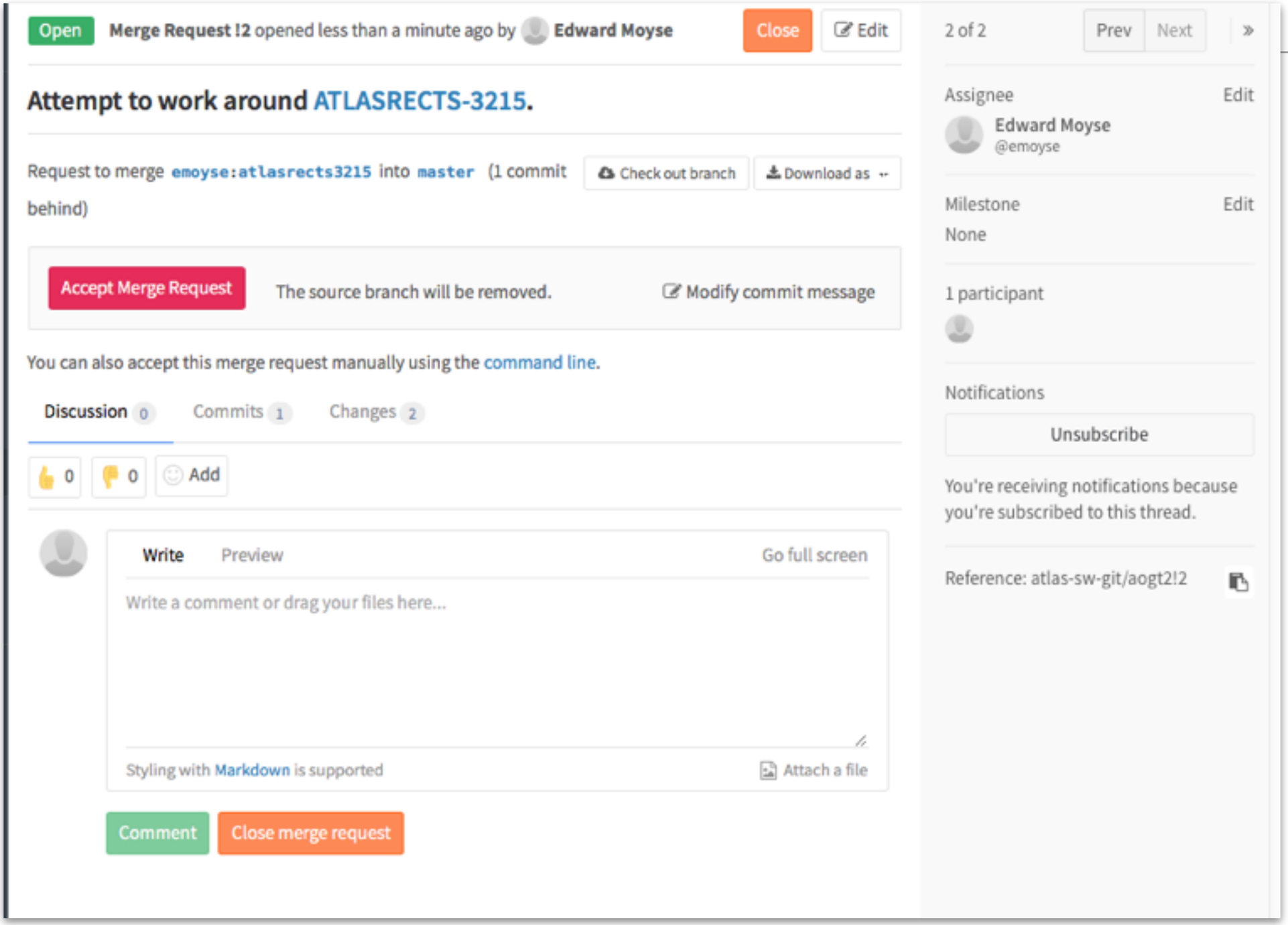


Can flag as WIP (can't be merged)

Can assign, create/ use milestones and add labels

Can auto remove source branch when accepted.

Can examine diff, as a final check it looks ok



Code review discussion

Possible helper tools

- You can checkout the entire repo, but this can become unwieldy, so we investigated using spare-checkout.
  - This takes about 45s on my laptop @ CERN and give:

```
150M    ./.git
184K    ./Tracking
```

  - Bit verbose to do:

```
git init
git config core.sparsecheckout true
echo Tracking/TrkEvent/TrkTrack >> .git/info/sparse-checkout
git remote add origin ssh://git@gitlab.cern.ch:7999/atlas-sw-git/aogt2.git
git  pull origin master
```

  - CMS has e.g. cms-addpkg - probably we'd want something similar to wrap the above
- General feeling is that we need pre-commit (pre-receive) on the master git repo
  - Block e.g. upload of large binary files, misnamed files…
  - Should provide an actual error messages of what part of the code (i.e. what line) is offending and why
  - Include in that error message whom to contact if the pre-commit hook seems to be misbehaving

Misc

- Submodules / subtrees
  - Use for e.g. externals
  - Details TBD
- Rebase
  - Useful tool, but potentially problematic (must not use after commits made public!)
  - Need to discuss recommendations … will likely need to update them as we start using Git
- Large files
  - SVN contained some big files - we really don’t want these in Git
  - LFS looks to be a good solution
  - To be investigated.

TODO and conclusions

- We need to determine structure of imported repo (and which tool will be used)
- We need to start regular tests of standard use-cases
- Once CMake builds from git are available, we can start on CI tests
- Still lots of details to sort, but pretty clear that git will be a very good fit for ATLAS
  - Eagerly looking forward to the move!

Backup

Study group tasks

- Or … things we need to know:
  - Structure of repository
  - Workflow
    - How do we collect SW changes
    - How do we distribute them in releases
    - Do we still need patch releases?
    - etc
- When do we do the migration
  - Spoiler: this isn't going to be answered today.

Some Key questions

- Repository
  - One for the whole of ATLAS offline or one per package?
- Workflows
  - How we envisage this to work? (Life cycle of a patch)
  - What are the sticking points?
  - Do we need integrators or can developers push directly into the main repository?
  - Continuous integration of merge requests
- Branches and Releases
  - We do have to manage multiple releases
  - Is it one branch per release?
  - And how many?
  - Do releases contain all packages, even those nor currently built?
    - If so, how do we describe to the build system which packages are in?
  - How do software changes migrate between branches (a la tag sweeps)
  - Lifecycle of a release
- Package tags
  - So these are gone - how do we succinctly discuss software changes?
- Practical Issues
  - Does gitlab scale well enough (repository size, number of developers)
  - Will it keep scaling for the next 10 years?
  - Namespace for git tags - how to manage?
- Migration
  - Does git only build releases >21?
    - Then we have to manage SVN + git; Would be a real pain!
  - If not, how do we ensure that old releases can be built from git?
  - How is the git migration coupled to the cmake migration?
  - Timescale…?

https://twiki.cern.ch/twiki/bin/viewauth/AtlasComputing/GitStudyGroup

Warning: Intrusive Change

Slide from Graeme

- This is going to be a major and intrusive change in the developer workflow
  - High level concept of development remains
    - Develop, Local test, Submit, Integrate, Build
  - But significant alteration to almost every aspect here:
    - Forking repository (gitlab/github)
    - Checking out *the repository* (or some section thereof)
    - Managing code within this private copy first
    - Committing back to gitlab
    - Making a merge request
    - Request tested and accepted
- In principle this offers many *advantages*, but it is a significant *disruption* to well worn habits

Checkout aogt...

Slide from Graeme

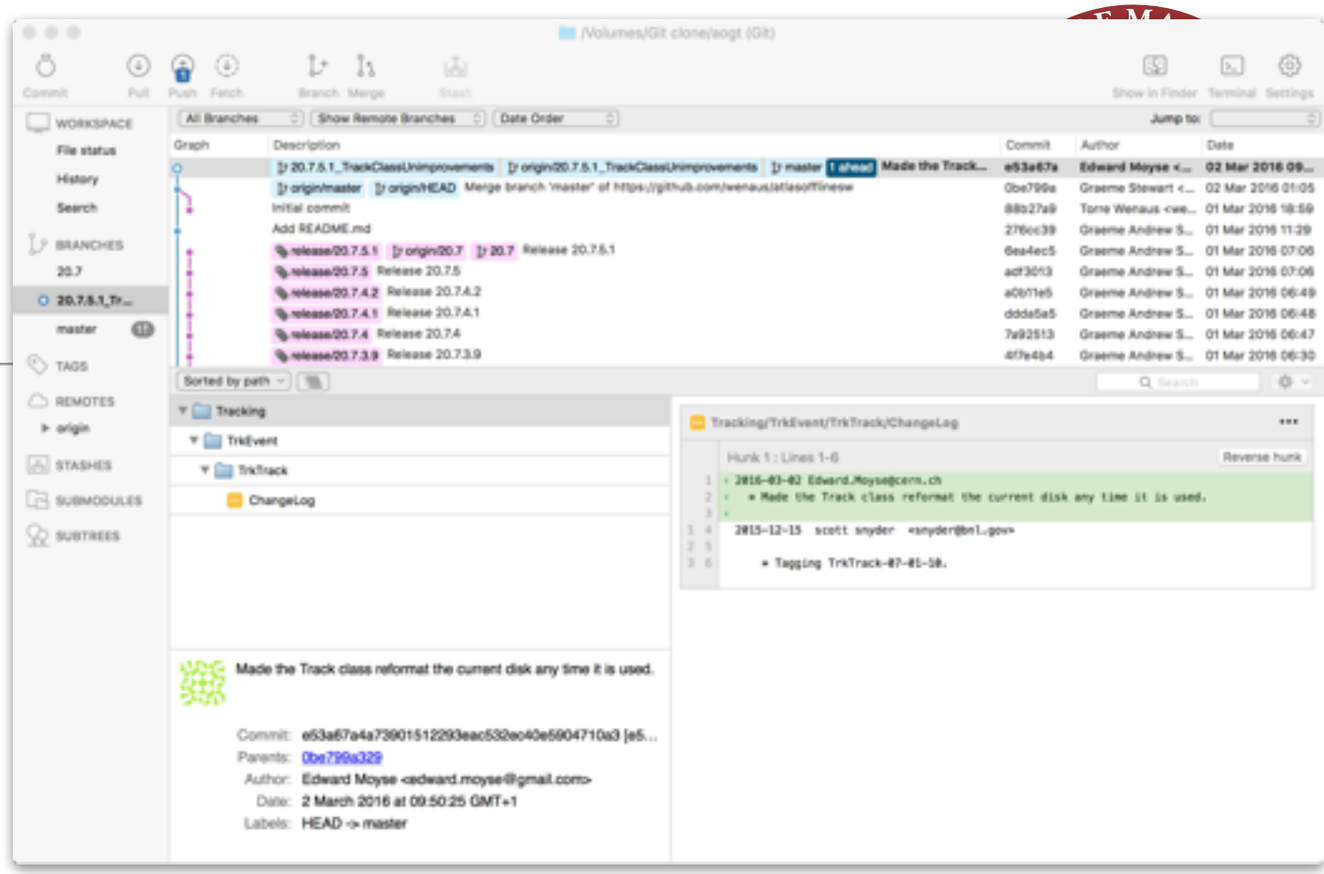
	git clone	git status
Local Mac with SSD	2m27s* or 40s	0.41s
Openstack VM	60s	0.78s
Local server to local disk	33s	0.55s
Local server to AFS	3m3s	3m20s (cold) to 13s (warm)
Glasgow Server (Remote)	40s	0.98s

As an aside: it's also possible to do a shallow checkout and only get a part of the repository … CMS have a tool to do exactly this.
Another aside: SVN takes about 40s to checkout 4 packages (!)

*Salle Dirac wireless!

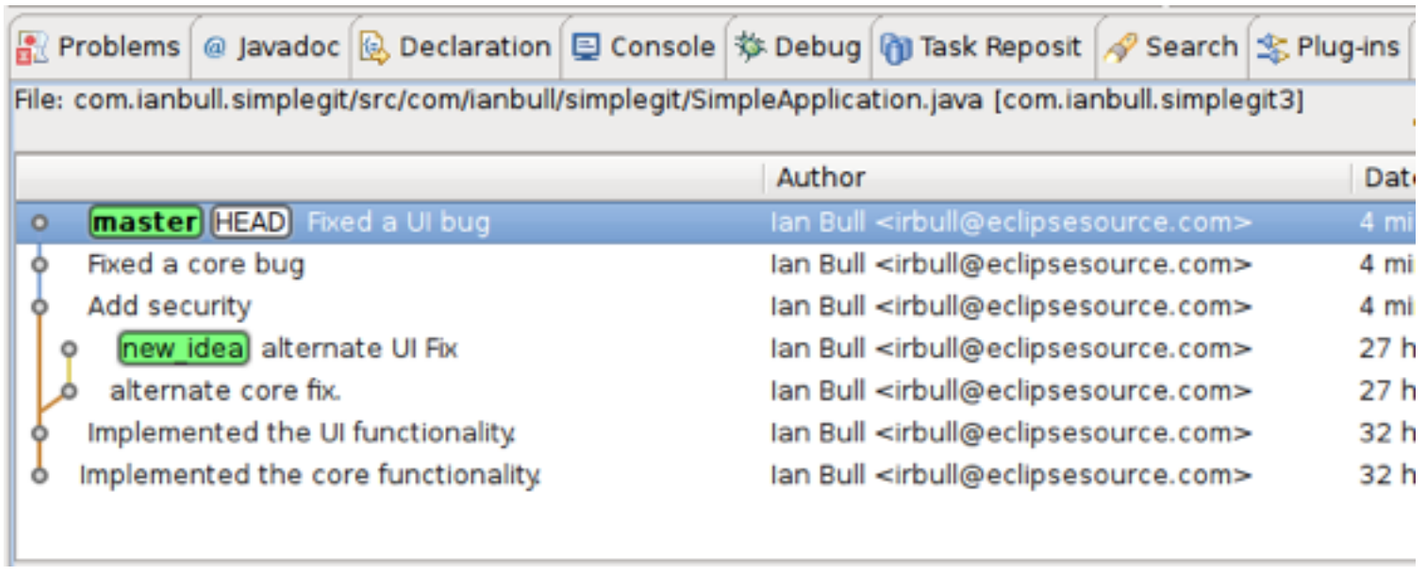
Use tools!

- As well as web interface (and command line, obviously), very nice third party tools exist



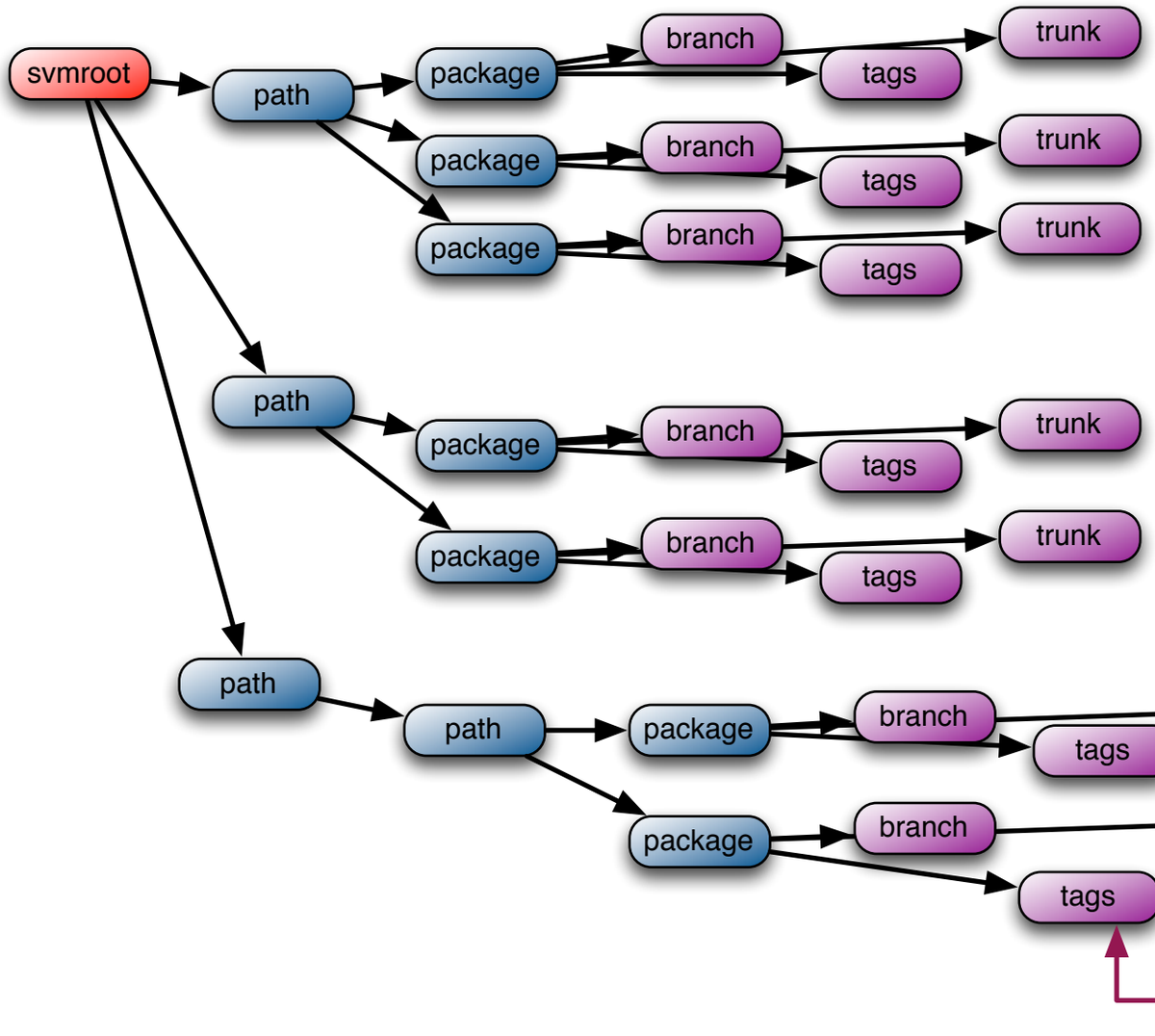
SourceTree - a Windows/OSX Gui

- Also, integration with IDEs



Slide from Graeme

SVN Repository



- Also notice that constructing a release from SVN itself is basically impossible
- Hence we have the Tag Collector

SVN - Pandora's Box...

Slide from Graeme

- SVN contains about 4800 packages
  - But we only have ~2200 in our AtlasOffline release
    - Many things are really not code packages at all
- In our "build" we have ~2GB of files, of which 1.4GB are > 100kB (.db, .root, .xml, …)
  - Trigger/TrigT1/TrigT1TGC - 300MB of text data stored in ".db" files
  - Found ESD and AOD POOL files

Extension	N_files	Total Size	Average Size
.db	3649	715.2MB	196.0kB
.ref	785	223.9MB	285.2kB
.xml	1739	223MB	128kB
.root	1323	197MB	149kB
.py	25328	193MB	7.6kB
.data	70	155MB	2.2MB
.dat	932	149MB	160kB
.cxx	16464	131MB	8kB

How mad are we…?

	Source Lines	Files	Directories
Linux Kernel	13M	42k	3.5k
CMSSW	4.8M	44k	6.4k
ATLAS Offline (trimmed)	4.1M	53k	13k

- Superficially what we are doing does seem to be reasonable
  - Anyway, we do know of techniques to deal a bit better with large repositories
    - Time filtering…
    - Shallow checkouts (caveat - CMS implemented this using AFS)