In this episode

Automating CMS everyday release integration workflows

Migrating and a large collaboration to a well known VCS (and survive while doing so)

How the two are related: towards a (real) Continuous Integration Workflow
CMS has 10 active platforms

- slc6 gcc434
- slc7 gcc472
- gcc481
- gcc462
- amd64
- slc5
- osx108
CMS has 10 active platforms

ARMv7HL
gcc481
slc6 gcc434
osx107 gcc472
amd64
gcc462
slc5 osx108

https://indico.cern.ch/contributionDisplay.py?contribId=303&sessionId=7&confId=214784
CMS has 10 11 12 active platforms

https://indico.cern.ch/contributionDisplay.py?contribId=303&sessionId=7&confId=214784
Multiple release series

Legacy releases
CMSSW_4_1_X
CMSSW_4_4_X
Multiple release series

Production releases:
CMSSW_5_3_X
CMSSW_6_2_X
Special branches

Specialized integration builds are done for limited periods of time to better check larger feature sets (e.g. multithreaded framework, new db code, detector upgrades, new ROOT)....
While we do our best to keep combinatorics under control, being able to run many different builds in parallel is extremely helpful to facilitate integration of new features and spot real bugs.
Integration workflow

RPM repository

Upload RPMs

Release building
(handled via pkgtools)

Download RPMs

Unit Tests

Download RPMs

Integration tests
(production workflows)

Download RPMs

QA
(static analysers, valgrind, additional tests)
Early solution

- Tie one machine to a given release series / architecture:
  - Ad-hoc configuration $<=$ solved by quattor and puppet
  - Machines do die more often than you might think
  - Long periods of inactivity
  - Dependencies between various tasks stitched together and fragile
  - Burden to maintain
- Does not scale
Improved solution:

In house task queue

- Release building
- Unit Tests
- Integration tests
- QA

Build pool

Build machines become expendable goods
Better solution:

I will not reinvent the (square) wheel.
I will not reinvent the (square) wheel.
I will not reinvent the (square) wheel.
I will not reinvent the (square) wheel.
I will not reinvent the (square) wheel.
I will not reinvent the (square) wheel.
I will not reinvent the (square) wheel.
I will not reinvent the (square) wheel.

Of course writing a scheduling queue in-house is not particularly wise....
Introducing Jenkins

• http://jenkins-ci.org

• JAVA (sigh) based continuous integration system

• Trivial deployment (standalone or servlet)

• Huge community and plugin library

• Mature (2005, was named Hudson at the time), yet still actively developed
acrontab on steroids

<table>
<thead>
<tr>
<th>Name</th>
<th>Last Success</th>
<th>Last Failure</th>
<th>Last Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install IB Releases</td>
<td>18 min - #929</td>
<td>23 days - #676</td>
<td>10 sec</td>
</tr>
<tr>
<td>Install Logs on AFS</td>
<td>59 min - #2237</td>
<td>7 days 2 hr - #2010</td>
<td>1 min 38 sec</td>
</tr>
<tr>
<td>Offsite RPM Repository Backup - Daily</td>
<td>14 hr - #92</td>
<td>1 mo 16 days - #44</td>
<td>12 min</td>
</tr>
<tr>
<td>Offsite RPM Repository Backup - Monthly</td>
<td>9 days 8 hr - #8</td>
<td>15 days - #6</td>
<td>0.66 sec</td>
</tr>
<tr>
<td>RPM Repository Backup</td>
<td>15 hr - #87</td>
<td>1 mo 28 days - #28</td>
<td>10 min</td>
</tr>
<tr>
<td>Update git mirror</td>
<td>15 hr - #107</td>
<td>4 days 15 hr - #103</td>
<td>12 min</td>
</tr>
</tbody>
</table>
**acrontab on steroids**

<table>
<thead>
<tr>
<th>Name</th>
<th>Last Success</th>
<th>Last Failure</th>
<th>Last Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Install IB Releases</td>
<td>18 min - #929</td>
<td>23 days - #676</td>
<td>10 sec</td>
</tr>
<tr>
<td>Install Logs on AFS</td>
<td>59 min - #2237</td>
<td>7 days 2 hr - #2010</td>
<td>1 min 38 sec</td>
</tr>
<tr>
<td>Offsite RPM Repository Backup - Daily</td>
<td>14 hr - #92</td>
<td>1 mo 16 days - #44</td>
<td>12 min</td>
</tr>
<tr>
<td>Offsite RPM Repository Backup - Monthly</td>
<td>9 days 8 hr - #8</td>
<td>15 days - #6</td>
<td>0.66 sec</td>
</tr>
<tr>
<td>RPM Repository Backup</td>
<td>15 hr - #87</td>
<td>1 mo 28 days - #28</td>
<td>10 min</td>
</tr>
<tr>
<td>Update git mirror</td>
<td>15 hr - #107</td>
<td>4 days 15 hr - #103</td>
<td>12 min</td>
</tr>
</tbody>
</table>
Historical information and logs
Jenkins has various ways of specifying dependencies, we use them to define build pipelines.
Towards a better development model

Continuous Integration

Not only Jenkins works for us as a acrontab on steroids it’s one of the key components to pursue “continuous integration”.

HEAD of release branches must be always in a releaseable state (apart from bugs, of course). No more unfinished developments in the HEAD

New developments / bug-fixes need to happen in separate branches and will be merged in the release branch once ready and tested.
git matches extremely well such a development model

- Easy branching
- Easy **merging** of changes
- Distributed by design
Popularity*

* as defined by Google Trends
A Brave New World

Long shut down as an opportunity to align to 2013:

- New Version Control Systems
- New Development Model

The goal is to minimize the potential barrier required for newcomers in the next 5-10 years and to minimize the extra custom layers required.
World leading git service provider

~3M users, 5M repositories, including Linux Kernel, twitter's bootstrap, ruby, node.js, jQuery, redis...last but not least, a ROOT mirror

CMSSW authoritative repository

We minimize our work by relying on features provided by Github (e.g. pull requests, teams) for free

Thanks to git distributed nature, we still maintain complete independence from Github (and CERN). E.g. we can still release CMSSW from a laptop in case of natural disasters, alien invasion, black hole event, govt shutdown...
Still one authoritative repository: https://github.com/cms-sw/cmssw

Includes full per file history of the old CVS repository, at least for files which entered some release. All release tags ported to the new repository.

One branch per active release

CMSSW_7_0_X, CMSSW_6_2_X, CMSSW_5_3_X, CMSSW_4_4_X, etc..

Topic branches

Developers use their private repositories to prepare additions in form of a branch and submit them for integration via the Github “Pull Requests” mechanism.

Topic branches can also be shared among peers to ease collaboration.

Sign-off workflow

Pull Requests are validated by various software area coordinators (RECO, SIM, CORE, HLT, etc) and merged into the main branch. Automated signature handling by a bot.
get official version
(via git cms-addpkg)

develop locally

pull request

via git push my-cmssw

https://github.com/albert/cmssw
Remove friendship relations to removed DaqSource.

No one is assigned

Milestone: CMSSW_7_0_0_pre4

No description given.

4 participants 🐂 🐐 🧣 🧡
An automated bot notices new pull requests and guides developers and software area coordinators through integration of new features.
Developers can give their comments about the proposed updates and coordinators give their sign off via +1 / -1
The bot keeps track of missing signatures and warns release manager when something is ready for final approval.
Impedance mismatch between HEP and git

**Git design assumptions**

- Disk space is cheap
- Developer knows how to build the software from scratch
- Developer will happily have a single release area on his laptop SSD disk

**CMSSW development workflow**

- We tend to work in terms of a base release (hosted centrally) on top of which we checkout a few packages in a so called development area
- We tend to create many development areas, one per “TODO” item
Working around the mismatch -- CMS way

Sparse Checkout

Feature of git allows to have most of the capabilities of single package checkouts back.

Reference Cloning (--reference)

A local mirror of the Github repository reduces to few megabytes the overhead of a given work-area. Objects in the git object-storage which are present in the reference repository will be picked up from there, and not copied in the workarea.
Almost every pull request is tested via Jenkins before it enters the integration build.
Continuous integration

Build Integrating Pull Request #1067
(https://github.com/cms-sw/cmssw/pull/1067)
(Oct 14, 2013 9:56:37 AM)

No changes.

Started by user CMS Build

GNU Compiler Warnings: 0 warnings.
- No warnings since build 869.
- Still 2 days before reaching the previous zero warnings highscore.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PULL_REQUEST</td>
<td>1067</td>
</tr>
<tr>
<td>CMS_GIT_TOOLS_REPO</td>
<td>cms-sw</td>
</tr>
<tr>
<td>CMS_GIT_TOOLS_REF</td>
<td>38fa3403ffce67bfff895ad236ff7816e2dc99cf4</td>
</tr>
<tr>
<td>RELEASE_FORMAT</td>
<td>CMSSW_7_0_X_2013-10-14-0200</td>
</tr>
<tr>
<td>DO_TESTS</td>
<td>checked</td>
</tr>
<tr>
<td>DO_SHORT_MATRIX</td>
<td>checked</td>
</tr>
<tr>
<td>DO_STATIC_CHECKS</td>
<td>unchecked</td>
</tr>
</tbody>
</table>
What’s the status?

**CMSSW development is 100% migrated to git**

We have done a number of releases (both production and development ones) already and we have twice per day integration builds which happily do not compete for CVS locks on AFS anymore.

**More than 300 forks on github, last month at 72 unique committers.**

**Polarized reactions to git and github.**

Some love it and have no problems, some others hate it. Some features are without doubt appreciated by everyone, e.g. the ability to keep code and related integration comments together, code reviews. We should always remember the alternative was not to stay with CVS, but move to SVN or less used alternatives and have the same kind of problems.

**Github support simply excellent.**
Conclusions

Long shutdown is a great opportunity to align to 2013 in terms of build tools.

We have in particular replaced a bunch of in-house code (not too mention an Oracle DB) by selecting Jenkins, git and Github as key components of our new infrastructure.

Replacing the tools alone is not enough, one needs to change the way of thinking as well to take advantage of them as well.
I. A robot may not injure a human being or, through inaction, allow a human being to come to harm.

II. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.

III. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.

The Three Laws will lead to only one logical outcome.
Backup slides
“SVN has been the most pointless project ever started, SVN used to say CVS done right: with that slogan there is nowhere you can go. There is no way to do CVS right.”

Linus Torvalds