Comon Git based EPICS Module Development

Bruce Hill
Murali Shankar
Sept 6, 2016

Motivation

- Several groups within SLAC use EPICS, but with different build systems, some using CVS and some using subversion for version control.
- This makes collaboration within the lab difficult.
- For packages such as EPICS base, modules, and extensions, the primary owner of the package is someone in the worldwide EPICS collaboration.
- Most of these collaboration packages are now being hosted on github, with more packages moving there as time goes by.
- This proposal outlines a git based workflow w/ common build support tools that attempts to meet the above needs.

Git Workflow Objectives

- For local development, we need git repo's which any developer can pull or push to so that anyone can create new releases as needed.
- These local SLAC master git repos need to be reachable from our development environments so build scripts can checkout and build releases as needed. For now, these SLAC master repos will be under /afs/slac/g/cd/swe/git/repos/package/epics
- Once we have a gitlab accessible from our development systems, we can move the SLAC master git repos to gitlab.
- Github master branch should not be committed to locally, as it's history is controlled by the package owner.
- To ensure we preserve locally added features/fixes, we need the git repo to have access to the latest versions from CVS and svn, and preferably the full history from each.
- Following common naming and build strategies will facilitate sharing of module development and new features.

Workflow template for EPICS modules

- SLAC master repo directory for EPICS modules:
/afs/slac/g/cd/swe/git/repos/packages/epics/modules
- Github repo added as remote github-origin
- Github master branch mapped to github-master
- A pre-receive hook is enabled for the SLAC repo to block pushing changes to github-master. It can only be updated by pulling from github-origin.

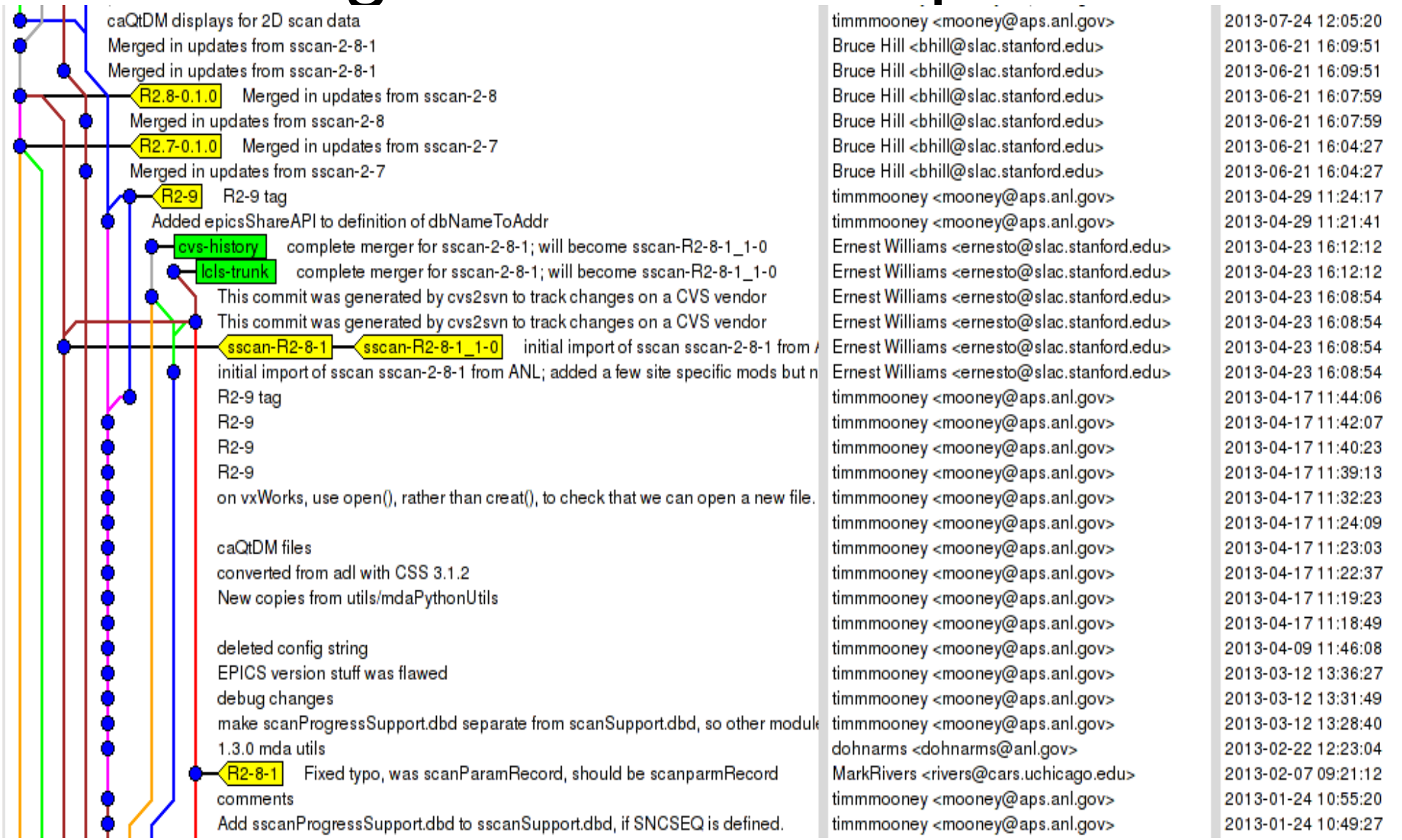
Initial import from CVS and SVN to Git

- Initial import to a dedicated git repo for each CVS and svn module
- CVS imported using cvs2git script in the cvs2svn package. Based on the existing cvs2git support in eco_tools w/ addition of an authors file to provide git style author and email for each commit.
- CVS history exported via eco_tools script to /afs/slac/g/cd/swe/git/repos/package/epics/modules/from-cvs/$MODULE
  - MAIN_TRUNK imported as master branch
  - Other branches imported using branch name
  - Tags imported as is
- SVN imported using “git svn clone” using a new eco_tools script, svnModuleToGit.py
- SVN history git repo: /afs/slac/g/cd/swe/git/repos/package/epics/modules/from-svn/$MODULE
  - trunk/pcds/epics/modules/$MODULE/current imported as master
  - Other branches imported using branch name
  - Tags imported using just the version specific portion. i.e. R2.1-0.1.0

Merging history in git

- The version history from cvs and svn, not that it's in git, can be easily fetched to a git repo like this:
  - git fetch cvs-origin master:cvs-history
  - git fetch svn-origin master:svn-history
- These new branches aren't easily merged w/ the github-master, as they share no common ancestors. Merges create many, many conflicts which must be resolved. If you view the repo via a gui such as gitk, the branches don't join.
- As git allows rewriting history, I fix this by using the GitPython package in a new

Example of gitk for sscan showing lcls-trunk and pcds-trunk



rhel6-x86_64

- We'll need to support building modules for different RedHat releases, which often don't have compatible builds
- Currently PCDS uses:
  - linux-x86_64 # RedHat 5 64 bit builds
  - rhel6-x86_64 # RedHat 6 64 bit builds
  - rhel7-x86_64 # RedHat 7 64 bit builds
- I propose we move to this naming scheme for new base releases and the modules/iocs built for them.
- This can be handled w/ a simple change to $EPICS_BASE/startup/EpicsHostArch

EPICS_BASE and RELEASE.local

- Moving forward we'll need to support building modules for different versions of EPICS_BASE, ideally w/o needing to create new module release tags.
- EPICS V4 has -include $TOP/configure/RELEASE.local in all V4 modules, as do many of the areaDetector modules. Sometimes CONFIG_SITE.local as well. These .local files are not included in the distribution. I propose we follow this model to allow building our modules w/ only the addition of a RELEASE.local and possibly a

Deriving relative EPICS_BASE

- $EPICS_SITE_TOP/$BASE_VERSION/RELEASE.local
  - BASE_VERSION = R3.15.0.1-0.1.0
  - EPICS_MODULES = $EPICS_SITE_TOP/$BASE_VERSION/modules
- Module releases follow the pattern
  - $EPICS_SITE_TOP/$BASE_VERSION/modules/sscan/R2.9.1-0.1.0
- $TOP/configure/RELEASE.local
  - -include $TOP/../../RELEASE.local
  - ASYN_MODULE_VERSION = R4.26-0.1.0
  - ASYN = $EPICS_MODULES/asyn/$ASYN_MODULE_VERSION
- $TOP/configure/CONFIG_SITE.local
  - Optional, add if needed
  - CROSS_COMPILER_TARGET_ARCHS = linuxRT_glibc-x86_64

Release tags

- To facilitate common scripted build support, such as automatic creation of new module versions on demand, I propose a common scheme for release tags.
- All tags will be normalized to the form R2.1.3-0.2.0
  - The first portion, R2.1.3 is based on the collaboration version number. Other variants could include R3.14.12.4, or R3.5
  - The second portion, 0.2.0, represents local modifications
  - Local releases starting w/ 0 indicate little to

Git Workflow

- Local development is done using the SLAC git repos as the master upstream repo.
- Feature development is done on topic branches, which can be kept in the developer's local repo and rebased as desired until pushed to the SLAC repo.
- No branches or other work which has been pushed to the SLAC repo should be rebased.
- No work is pushed to github from our SLAC repo. Instead, users should create their own github forks and push topic branches to their github fork where they can become pull requests.
- New updates from the collaboration are handled by a local developer who pulls github-master directly

The End

- Thanks for your attention
- Comments welcomed