Introduction to Super C-Tau detector software development

Dmitry Maksimov

7 September 2018

Contents

Overview

Workflow in several words

- Get your own copy of the offline code from GitLab
- Develop, test, develop, retest, . . . until you are happy
- Commit your code and push it back to GitLab
- Create a merge request that asks for your code to be added to the main repository
- Continuous integration* runs to test your changes, a human being checks your code — when there are no problems with your code, the merge is accepted and your code has been integrated

*will be available later

Detailed workflow description

Registration

1. Make registration at BINP/GCF cluster

Contacts:

- Dmitry Maksimov <D.A.Maksimov@inp.nsk.su>
- Andrey Suharev <A.M.Suharev@inp.nsk.su>

2. Login to **stark** or **proxima**
   Create ssh keys if absent

$ ssh-keygen

   Accept default answers for all questions

3. Login to https://git.inp.nsk.su/
   Put ssh **public** key to your gitlab account

Repository

Central repository https://git.inp.nsk.su/sctau/aurora

First of all make a fork of Aurora project

Environment setup

Working environment is prepared at `stark` and `proxima`.

```
# Basic setup
# run this after each login
$ setupSCTAU
```

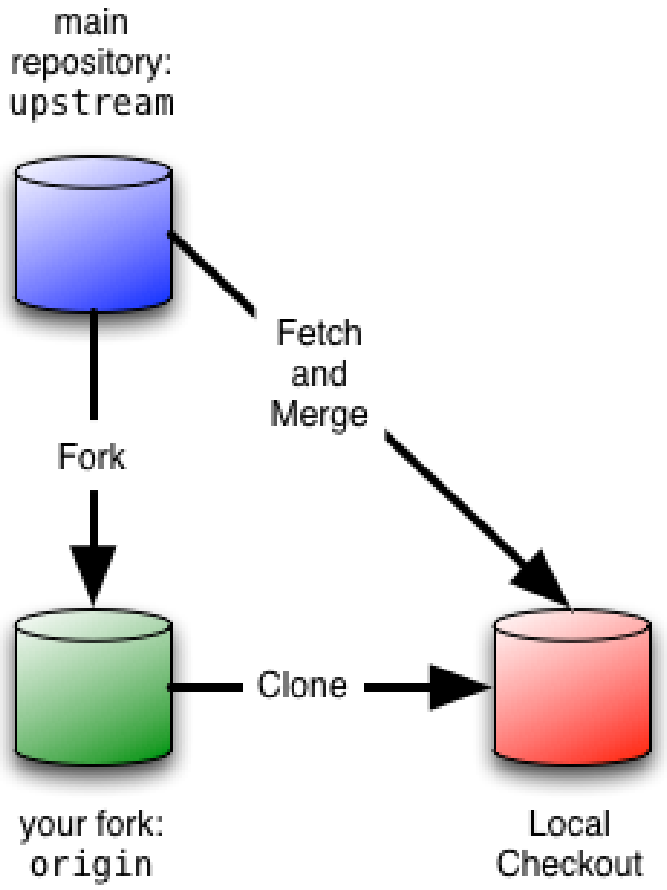
Workdir setup

```
# Create work dir
$ mkdir workarea
$ cd workarea

# Directories for builds and run
$ mkdir build run

# Setup project, release and version
asetup SCTauSim,0.1,latest


# Run primary generators example
cd run
ctaurun GenExamples/evtgen.py
```

Initialize workdir for development

```
$ cd workarea
# Initialize working directory
$ git sctau init-workdir \
>     ssh://git@git.inp.nsk.su/sctau/aurora.git
$ cd aurora

# Get updates for central repository
$ git fetch upstream

# Create working branch
$ git checkout -b MyDevelopmentBranch upstream/0.1 --no-track

# Add some packages
$ git sctau addpkg GenExamples
$ git sctau addpkg G4SimExamples
```

Repository relations



Build and run

```
# Building
$ cd ../build/
$ cmake ../aurora/Projects/WorkDir
$ make

# Make setup to use local (!) packages
$ source x86_64-slc7-gcc7-opt/setup.sh

# Running
$ cd ../run
$ ctaurun.py GenExamples/evtgen.py
```

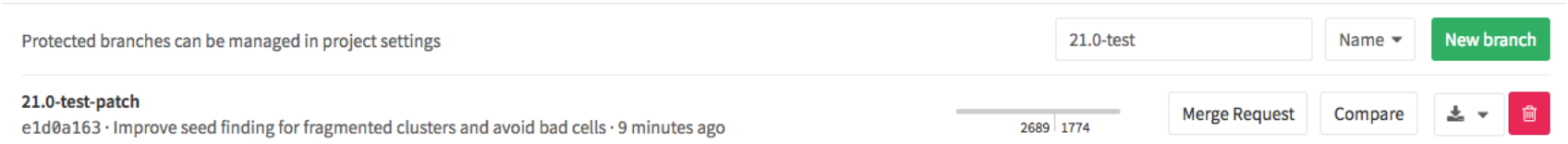
Save and publish results

```
$ git add ...

$ git commit -m '...'

$ git push -u origin MyDevelopmentBranch
# or
$ git push
```

Merge requests

Create **Merge request** from GitLab interface

Go to the branch overview page on your fork by choosing `Repository -> Branches` in the top menu. Then select the `Merge Request` button next to your new branch.



or by link from `git push` command

Merge requests

- Make sure you are merging to the right target branch (this should be the target branch from which you started your topic branch). If GitLab didn't select the correct branch use the `Change Branches` option to correct it.
- Make sure that you enter a good description of your change.

New Merge Request

From **graemes/athena:21.0-test-patch** into **atlas/athena:21.0-tutorial** Change branches

Title: Egamma Improve seed finding for fragmented clusters and avoid bad cells

Start the title with `WIP:` to prevent a **Work In Progress** merge request from being merged before it's ready.

Add description templates to help your contributors communicate effectively!

Description: Write Preview

Add a new EMCrackEtCut property to to the egammaTopoClusterCopier class
and use this to accept clusters in the crack region if they exceed the Et cut.
This improves seed finding for cases of highly fragmented clusters.

Fix ATLASRECTS-3867 where a WARNING was printed, but actually this
happens and is harmless to downgrade to DEBUG,

Fix ATLASRECTS-3860 where LAr properties were mistakenly asked
of tile cells - just veto tile cells at the top of the loop.

Styling with Markdown and slash commands are supported Attach a file

Package components

Package structure

```
<PackageName>
├── <PackageName> ..................... C++ headers
├── src ............................... C++ sources
├── python ............................ python modules
├── doc ............................... documentation
├── data .............................. data files
├── share ............................. data files, scripts, joboptions
├── joboptions ........................ jobs configuration fragments
├── scripts ........................... scripts
├── xml ............................... xml files, Detector geometry for example
└── CMakeLists.txt .... Package building rules description, required
```

CMakeLists.txt Example (1)

```
#################################################################
# Package: EvtGen_i
#################################################################
# Declare the package name:
sctau_subdir(EvtGen_i)

# Declare the package's dependencies:
sctau_depends_on_subdirs( PUBLIC
                          Control/AuroraCommon
                          Control/AuroraAlg
                          External/HepMC
                          External/EvtGen
                          Generation/GenInterfaces
                          PRIVATE
                          Generation/GenTools/EvtGenExternal
                          Tools/PathResolver )
```

CMakeLists.txt Example (2)

```
# Component(s) in the package:
sctau_add_component( EvtGen_i
                     src/*.cpp
                     PUBLIC_HEADERS EvtGen_i
                     PRIVATE_LINK_LIBRARIES AuroraAlg
                       EvtGenExternal EvtGen HepMC
                       PathResolver )

sctau_install_data( share/* )
```

cmake functions (1)

- sctau_subdir — declare package, set its name,
  must be first in CMakeLists.txt
- sctau_depends_on_subdirs — specify dependencies
- sctau_add_library — declare package library
- sctau_add_component — package library plus generate joboptions configurables,
  not linkable to other libraries
- sctau_add_executable — declare executable
- sctau_add_alias — declare short script from set of commands
- sctau_add_test — declare package unit-tests
- sctau_add_dictionary — declare ROOT (Reflex) dictionary as shared libary
- sctau_add_root_dictionary — declare ROOT (CINT) dictionary as C++ source code

cmake functions (2)

- sctau_install_headers — install header files (for pure interface packages)
- sctau_install_python_modules — install python modules
- sctau_install_data — install data files
- sctau_install_joboptions — install job configuration files or fragments
- sctau_install_docs — install documentation
- sctau_install_xmls — install XML files
- sctau_install_scripts — install executable scripts
- sctau_install_runtime — install files required at runtime, but not matching above categories

SCTau software project management

Aurora issues

If you have a work that

- Need to be tracked
- Need to be discussed
- Need to be remembered to return later
- ...

Create an issue!