Table of Contents

ECDF

ECDF is the Edinburgh Compute and Data Facility. It is a University wide resource, available to all research groups. This is a short introduction to getting your LHCb jobs running on the cluster and accessing your data. A link to **ECDF** home page.

Getting an account

You can request an account online or by sending an email to Science Support: science.support@edNOSPAMPLEASE.ac.uk.

Logging in

First of all, you need to request a user account on the machine. It should be the same username/password as EASE. Email ecdf-systems-team AT lists.ed.ac.uk.

```
ssh username@eddie.ecdf.ed.ac.uk
```

You will see that you have a home area at `/exports/home/username`. There is a `ppe` group work directory at `/exports/work/ppe/` which has more space available for us to use.

Quick .bashrc

Just a quick reference to a complete .bashrc on ECDF which sets up an environment to build and run RapidFit:

```
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
# Source LHCb on EDDIE
source /exports/work/ppe/lhcb/lhcb-soft/scripts/local-setup.sh
# Setup Ganga
export X509_CERT_DIR='/exports/work/middleware/WN/etc/grid-security/certificates'
GangaEnv 5.4.3
# Setup Root
export ROOT_PATH=/exports/work/ppe/sw/cern/root_v5.25.02_gcc41
cd $ROOT_PATH &&  source bin/thisroot.sh
cd
```

Sourcing the LHCb environment

For the moment, you need to do something like this:

```
source /exports/work/ppe/lhcb/lhcb-soft/scripts/local-setup.sh
```

Then you can set your application environment in the usual way:

```
setenvGauss v30r3
```

- Before using **getpack**, you will need to follow the instructions for a local installation of the LHCB software (to setup a ssh-key; bear in mind you will also need to have a .hepix directory in your ecdf-home area).

After that, you can get the applications and build your own algorithms, e.g.,

```
getpack Sim/Gauss v30r3
source Sim/Gauss/v30r3/cmt/setup.sh
```

You all know the rest....

32bit compatibility libraries

At the moment (August 2007) the LHCb software is still not fully supported on 64bit platforms (like ECDF). Therefore, before you build any software, you should ensure that you change your `LD_LIBRARY_PATH` by doing this (bash or csh):

```
export LD_LIBRARY_PATH=/exports/work/ppe/lib32:$LD_LIBRARY_PATH
setenv LD_LIBRARY_PATH /exports/work/ppe/lib32:${LD_LIBRARY_PATH}
```

Submitting Jobs

ECDF uses the Sun Grid Engine (SGE) batch system. It has commands like `qsub` and `qstat` which you will be familiar with. To get this environment,

```
module load sge
```

You can then submit jobs doing something like:

```
qsub -l h_rt=HH:MM:SS jobscript.sh
```

where HH:MM:SS is the time that you expect your job to run for. If you don't specify `-l h_rt HH:MM:SS` then your job will be added to the 30min queue. There are four different run time limits for various portions of ECDF. These are currently set to 30 minutes (default), 6 hours, 24 hours, 48 hours. You should get into the habit of specifying your required run time when you submit your job.

```
qstat -u username
```

gives you a list of your jobs and their status.

- More detailed information can be found in the SGE user's guide.
- ECDF system statistics

Test first on interactive node

Before submitting your job to the batch queue, you should test that it works first of all by first logging into one of the interactive machines:

```
qlogin
```

You can then proceed as normal.

Job output

By default, the stdout and stderr will be written to your home area (/exports/home/username/) as jobscript.csh.oJOBID and jobscript.csh.eJOBID. You can custmize the output location by using the -o and -e options when submitting the job using qstat.

Any other output from the job will be written to the $TMPDIR on the node where the job runs. You need to copy this back to your home or work area once the job is complete. See the example script for an idea of what to do.

Array jobs

If you want to submit a large number of jobs, each of which changes a parameter, then you can use the concept of an array of jobs. For example:

```
qsub -t 2-20:2 jobscript.csh
```

Submits 10 jobs, the first with 2 as a parameter, then 4, 6...20. You can use the shell variable $SGE_TASK_ID to get hold of this number and use it in your script. Again, see the example.

Example script

```
#!/bin/csh
# Greig A Cowan, August 2007

# Set output location
set OUTDIR=/exports/work/gridpp/gcowan1/Gauss
#$ -o /exports/home/gcowan1/work/Gauss
#$ -e /exports/home/gcowan1/work/Gauss

# Source the environment
source /exports/work/ppe/lhcb/lhcb-soft/scripts/local-setup.csh
setenvGauss v30r3
source Sim/Gauss/v30r3/cmt/setup.csh

# Set up LD so that it gets the correct libs
setenv LD_LIBRARY_PATH ${HOME}/temp/lib32:${LD_LIBRARY_PATH}

# Options and Executable
set MYOPTS=myGauss.opts
set OPTS=~/cmtuser/Gauss_v30r3/Sim/Gauss/v30r3/options/${MYOPTS}
set EXE=~/cmtuser/Gauss_v30r3/Sim/Gauss/v30r3/slc4_ia32_gcc34/Gauss.exe

# Change to the job working directory and get a local copy of the options file
cd ${TMPDIR}
cp -r ${OPTS} .

# Need to change this for each run to get independent events
set EVTNUM=${SGE_TASK_ID}
echo "GaussGen.FirstEventNumber = ${EVTNUM};" >> ${MYOPTS}

# Run the job
${EXE} ${MYOPTS}

# Rename the output so that we don't overwrite files
mv ${TMPDIR}/GaussHistos.root ${OUTDIR}/GaussHistos-${EVTNUM}-${JOB_ID}.root
mv ${TMPDIR}/GaussMonitor.root ${OUTDIR}/GaussMonitor-${EVTNUM}-${JOB_ID}.root

# Tidy up
rm *.sim
```

Root

Working

Root can be set within an ECDF (a.k.a Eddie) session by adding the following lines to your .bash_profile script:

```
#  Root section of .bashrc on ECDF
#
export ROOT_PATH=/exports/work/ppe/sw/cern/root_v5.25.02_gcc41
cd $ROOT_PATH &&  source bin/thisroot.sh
```

```
cd ~
# the last line returns the user to their standard login directory asif they never ran off to get
```

Leave root to configure itself which is neater in many ways to appending environmental variables yourself imo

Old_Config

I leave in an old configuration (below) used by some others in the group, but I had issues with it...

```
#....................................
#Path to LHCb-PPE group software
export LHCBPPE=/exports/work/ppe/lhcb

#....................................
#set ROOT
export ROOTSYS=$LHCBPPE/cern/root/pro
export PATH=$ROOTSYS/bin:$PATH
export LD_LIBRARY_PATH=$ROOTSYS/lib/root:$LD_LIBRARY_PATH
```

It is also possible to set the Root enviroment on ECDF using:

```
 module add root
```

Where the above add module line could aslo be added to your .bash_profile script.

Using Ganga on ECDF

Ganga SGE setup

Ganga supports SGE as a backend, in the same way that it supports LSF at CERN, so if you are used to using Ganga for configuring your job environment and options, then you can continue to use it while submitting jobs to ECDF. If you use Ganga, you don't have to worry about all of the SGE scripts, qsub and qstat commands.

You'll need some entries in your bashrc to find grid libraries:

```
export X509_CERT_DIR='/exports/work/middleware/WN/etc/grid-security/certificates'
```

Ganga 5.X

All the configuration is managed centrally using the ganga ini and ganga_utils. You need nothing in your gangarc except the setup and configuration of ganga_utils. If you're running with DPM you will need to add X509_USER_PROXY to your bashrc (even if you use tcshell).

To use root in ganga you need to add:

```
location = /exports/work/ppe/sw/cern/root_v5.25.02_gcc41
```

...in the [ROOT] section of the .gangarc on ECDF.

Ganga 4.X

Before using Ganga 4.X (which isn't managed centrally) you need to first setup your .gangarc file to be able to submit to SGE with some special options that we need. Add this stanza to the bottom of your .gangarc file on the ECDF frontend nodes:

```
[SGE]
kill_str = qdel %s
submit_str = cd %s;qsub -cwd -l h_vmem=2500M -V %s %s %s %s
preexecute = os.chdir(os.environ["TMPDIR"])
        os.environ["PATH"]+=":."
        os.environ["DCACHE_REPLY"]="".join(("eddie",os.uname()[1][4:],".ecdf.ed.ac.uk"))
        os.environ["DCACHE_CLIENT_ACTIVE"]="1"
```

If you're running with dCache data or DPM data you will need extra bits and pieces here, check out the individual pages for more details.

Ganga local filesystem setup

The file system in use on eddie is very sensitive to large numbers of small files in your home directory. Hence it is adviseable to place all output files in the shared filespace.

You will need to change the job workspace location to a new directory under /exports/work/physics_ifp_ppe, either by making a softlink in your gangadir/workspace to point there, or be editing your gangarc

```
[FileWorkspace]
...
topdir=/exports/work/physics_ifp_ppe/scratch/<username>/ganga_workspace
```

You must change the outputdata location for files like .sim .digi .dst to somewhere with more space, i.e. the scratch space under /exports/work/physics_ifp_ppe/scratch which is also given here.

```
[LHCb]
...
DataOutput=/exports/work/physics_ifp_ppe/scratch/<username>/ganga_outputdata
```

You can check these in your ganga session by typing:

```
In [34]:config['FileWorkspace']
Out[34]: [FileWorkspace]
*    topdir = '/exports/work/physics_ifp_ppe/scratch/rlambert/ganga_workspace'
     splittree = 0


In [35]:config['LHCb']
Out[35]: [LHCb]
*    DiracTopDir = '/afs/cern.ch/lhcb/software/releases/DIRAC/DIRAC_v2r15'
     DiracLoggerLevel = 'ERROR'
     copy_cmd = '/bin/cp'
*    DataOutput = '/exports/work/physics_ifp_ppe/scratch/rlambert/ganga_outputdata'
     mkdir_cmd = '/bin/mkdir'
     maximum_cache_age = 10080
*    LocalSite = 'CERN'
*    SEProtocol = 'castor'
```

If you like, you can also change the location of your job repository to point to shared filespace, or use a remote repository to reduce the number of files in the filesystem.

Ganga Quickstart

To start Ganga, first source the LHCb environment (see above) and then use:

```
GangaEnv
```

to pick your Ganga version. You can then start up ganga with:

```
ganga
```

To submit a test "Hello, World!" Executable() job you can do something like:

```
j=Job( backend=SGE())
j.submit()
j.peek('stdout')
```

As you will have worked out from the options added to the .gangarc file, you can use Ganga to submit DaVinci jobs that need access to the files stored on the LHCbEdinburghGroupDcache.

Submitting to different queues

On LSF you would type:

```
j=Job( backend=LSF())
j.backend.queue='8nh'
```

To submit to an 8nh queue. There is no implimentation of this in the Ganga backend. Instead you must modify the configuration to submit to different queues.

Add the string: -l h_rt=hh:mm:ss to the submission string, where hh:mm:ss is the expected length of the job.

Set a default length in your gangarc:

```
[SGE]
...
submit_str = cd %s;qsub -cwd -l h_vmem=2500M -l h_rt=4:00:00 -V %s %s %s %s
```

Or set it directly in ganga on the command line before submitting your job.

```
config['SGE']['submit_str'] = 'cd %s;qsub -cwd -l h_vmem=2500M -l h_rt=8:00:00 -V %s %s %s %s'
```

Submitting ROOT jobs from Ganga

I would recommend using the Root() application type:

```
j=Job(application=Root(), splitter=ArgSplitter(), merger=RootMerger())
j.submit()
```

You should first of all set the path in the [ROOT] section of .gangarc. This will gaurantee that Ganga picks up the correct version of ROOT to run your jobs. Probably best not to have $ROOTSYS set in your environment in addition to this.

```
[ROOT]
path = /exports/work/ppe/sw/builds/root
```

If instead you want to submit a precompiled excecutable with root, make sure you build, run and merge with the same version of root. For example version 5.12:

```
[bashrc]
export ROOTSYS='/exports/work/physics_ifp_ppe/sw/builds/root5_12/root'
PATH=${PATH}':'${ROOTSYS}'/bin'
LD_LIBRARY_PATH='/exports/work/ppe/lib32:'${LD_LIBRARY_PATH}':'${ROOTSYS}':'${R
OOTSYS}'/lib:'${ROOTSYS}'/include:/usr/lib'

[gangarc]
[ROOT]
#  Location of ROOT
```

```
location = /exports/work/physics_ifp_ppe/sw/builds/root5_12/root

#  Set to a specific ROOT version. Will override other options.
path = /exports/work/physics_ifp_ppe/sw/builds/root5_12/root

version = 5.12.00g
```

/Custom_binary

If you want to run RapidFit or some custom_binary on ECDF you want to use ganga in the standard way

I use my own branch of RapidFit with some code yet to be tested/pushed to the trunk.

0) Setup the environment and ganga to know where a compatible root is on ECDF

1) In order to use this I first grab a copy of my branch and compile it on ECDF in the standard way

2) Start Ganga

3) setup your job as:

```
j=Job(application=Executable(exe='/full/path/to/fitting',args=['-f','some_example.xml','-events',
```

here there are 6 args to be passed to the binary this is the same as './fitting -f some_example.xml -events 100 -repeats 2'

You need to provide the input to this now:

```
j.inputsandbox=[File('/path/to/some_example.xml')]
```

Notice that the main program just looks for the file some_example.xml as if it is in the same directory as the binary and ganga takes care of providing it when the job runs

For this job there are outputs:

```
j.outputsandbox=['pullPlots.root']
```

If 2 more options have been passed in the job decleration i.e. '--doPulls','some_output_filename.root' you would have to tell ganga you want 'some_output_filename.root' back with the outputsandbox mechanism.

4) Run on ECDF:

```
j.backend=SGE()
j.submit()
```

To check your job is running on ECDF:

```
qstat
```

The Columns of interest are generally:

state: r for running qw for submitted in the que (waiting) anything else usually indicates a failiure of some sort slots ja-task-ID: number of slots requested on the ECDF cluster (max 8) 1 slot correponds to 1CPU and 2Gb memory per job with max of 16Gb

(requesting more than 16Gb will cause a job to hang, although if a job is mutlithreaded then you can request more than 8CPUs but if your doing that your beyond this 'tutorial')

5) Wait, go away and come back to some nice pull plots (providing you got your xml right)

Current Issues

- Using setenvGauss in a bash script does not work. This is OK if you use csh. Strange.
- Using Gauss v30r3 you need to get a copy of a couple of libraries that are not install on ECDF. You then need to modify your LD_LIBRARY_PATH before compiling the code.

- Ganga's ability to cleanup after a jobs(job_num).remove() is a bit poor I also run from the command line for i in $(qstat | grep job_num | awk -F" " '{print $1}'); do qdel $i; done; assuming you've edited your ganga config to give jobs sensible names on ECDF

-- GreigCowan - 03 Aug 2007

This topic: Main > LHCbEdinburghGroupECDF
Topic revision: r16 - 2010-11-22 - RobCurrie