A beginner's guide to the Batch Submission and Execution (BSE) Farm

This document briefly lists the features of the Physics departmental cluster and gives some basic UNIX commands to allow you to get started using it. Those with a knowledge of UNIX will probably want to read the first section and skim the rest: real beginners will probably want to skim the first section and then come back to it.

Contents

1 Hardware

1.1 Nodes

The cluster is made up of a front end server (calgary.phy.bris.ac.uk) which is used for batch job preparation, debugging and submission to the PBS

(Portable Batch System) batch queueing system for eventual running on one of the worker nodes (abbreviated WN).

calgary and its WN run 64-bit Scientific Linux v5.5 = approximately RedHat Enterprise Linux (RHEL) 5.5. The WN are quad-core 2.8GHz nodes with 4GB RAM; one core is reserved for interactive use, the other 3 are for batch processing.

When using the farm, log into calgary, and you may run short CPU-intensive jobs interactively on calgary. To run/submit batch jobs, submit them via PBS to the WN (instructions are in Section 6 of this document).

You can see queued and running jobs by looking at

`http://calgary/cgi-bin/pbswebmon.py`

1.2 Disks

Your home directory is NFS-mounted from a PP Linux server (calgary). There is a quota on your home directory; it's not for Large Software or Data.

For large data, use /data/cluster (automounted to the WN), or any of /data/sda1, sdb1 ... sdi1 - these are large RAID partions. Automount = just use them (ls or cd) and they'll auto-mount from server.

Your home directory (the directory you start in when you log in) is /usersc/<your user name> although the system puts 'automount' in front of it. Home directory space is intended only for personal setup and config, and has quotas imposed; therefore you are encouraged to keep data on /data/cluster or /data/sda....i1. Just make a working directory, i.e. mkdir /data/cluster/<your user name>, to store your data.

The data disks are **NOT** backed up. Only /usersc is backed up each evening.

2 Basic UNIX commands

2.1 Files

In UNIX, files store data and directories contain files. Examples of some basic file- and directory-manipulating commands are listed below.

Command	Effect
`ls`	List the files in the current working directory
`ls -al`	Get a long listing of all files
`ls dir`	List files in the directory dir
`rm file1 [file2 file3...]`	Delete one or more files
`cp file1 file2`	Copy file1 to file2
`cp file1 file2 ... newdir`	Copy files 1, 2... to directory newdir
`mv file1 file2`	Rename file1 to file2
`mv file1 file ... newdir`	Move files 1, 2...to directory newdir
`cat file1`	Print file1 on the terminal
`cat file1 file2... > newfile`	Join files 1, 2... together and put the result in newfile.
`cd dir`	Change the current working directory to be dir. `cd ..` to go back up one level.
`mkdir dir`	Make new directory dir
`rmdir dir`	Delete the (empty) directory dir
`less file`	View file on the terminal, moving around with cursor keys

The `man` command gives access to on-line documentation. E.g. `man ls` would tell you about all the options for the ls command. `man -k wombats` searches for all commands relating to wombats.

2.2 Networking - from Windows

The `slogin` and `ssh` commands allow you to log in to other machines in the local cluster to which you have been allowed login access. (Generally login access is fairly restricted.) You do not need to specify the domain name (.phy.bris.ac.uk) part of the internet address.

```
calgary> slogin capitolhill

[ capitolhill will ask for your password ]

capitolhill>
```

Log out with `logout` or `exit` as usual.

`ssh` allows you to execute a single command or series of commands on a remote machine without having to log in; so

```
calgary> ssh capitolhill ps -e -u fred
```

allows a user on calgary to list fred's processes on capitolhill.

Transferring files can be done in two ways. `scp` is easiest for transferring files between two machines on the BSE farm (though it's not usually necessary to do this since all disks are shared). E.g.

```
calgary> scp capitolhill:/var/tmp/file1 file2
```

copies the remote file from capitolhill to file2 on your local machine, while

```
calgary> scp localfile1 capitolhill:/var/tmp/some/dir
```

copies a file local to calgary to capitolhill. For file transfer outside the cluster, use `sftp`:

```
sftp norman.bris.ac.uk
```

You will need to supply a user name and password to the remote host, unless you have been told to use 'anonymous' ftp, in which case give 'ftp' or 'anonymous' as the user name and your e-mail address as the password. Once connected, you can use a limited set of file management commands: `dir` or `ls` to get a directory, `cd` to change remote directory, and `get` and `put` to transfer files from and to the remote machine respectively. `mget` and `mput` allow you to use wildcard characters (e.g. `mget *` to get all files in a remote directory). Before you transfer binary files (anything other than plain text) be sure to use the command `bin` to specify that the program should not try to translate any characters, otherwise the files will be corrupted.

2.3 Windows

Use **PuTTy**. This can be obtained from `\\physics\dfs`, look for putty.exe

Start up putty.exe. The PuTTY install may put it in `C:\Program Files\Putty\putty.exe` A desktop shortcut might be handy.

The window "PuTTy Configuration" should appear. Type "calgary" in the box "Host name (or IP address)" and select the "SSH" button. Then "Open". If accessing the server from off-campus, you will need to use the full host name "calgary.phy.bris.ac.uk".

The first time you do this, a "PuTTy security alert" will appear and tell you that it's a new host that PuTTy doesn't know about. It is safe to select "Yes" to the query so you don't get this alert every time.

A window on calgary will appear, so you can login and begin working. PuTTy will allow you to perform any line-based Linux function such as navigating the directory structure and listing files.

To use emacs, RooT or any Linux graphical application, you also need to run eXceed on the Windows host. If it is installed, then from the Start menu select Hummingbird Connectivity, then eXceed, then eXceed. Once it is running, there's an entry on the task bar but no other windows appear.

If eXceed is not installed, you need to contact a Windows sysadmin person to install it. Email to support@phy.bris.ac.uk with your request.

on Windows, a program called the Xming X server can perform the same function as eXceed and includes PuTTY within its programming. It can be installed from

`http://sourceforge.net/projects/xming`

It's also a good idea to install the xming-fonts installer. A good description of using the program it creates - Xlaunch - is here:

`http://www.linux.com/feature/118106`

More info on Xming:

`http://freedesktop.org/wiki/Xming`

The X server uses a program called XLaunch to access calgary. There are

five windows to navigate before reaching the calgary command line. The only one needing editing is the second and third ones (the first, fourth and fifth windows you can just click next).

In the second window, select 'start a program' instead of 'start no client'. A new window will come up called 'Start Program'. Under 'Run Remote' choose the 'Using Putty(plink.exe)' option and fill in the text boxes as follows:

```
Connect To Computer:  calgary.phy.bris.ac.uk

Login as user:  your username

Password:  your password
```

Once eXceed (or Xming) is running, you also need to tell calgary where to send the graphical output :

```
export DISPLAY=remotehostname:0
```

where remotehostname is the name of the machine from which you're running PuTTY/eXceed. Unfortunately that's tricky to automate as the OS doesn't set REMOTEHOST for bash users.

You can test whether you have things setup properly by typing a graphical command, e.g. "emacs &". If it's working, you'll get a query from eXceed asking you whether you want to allow calgary to send windows to your PC. Answer "Yes" and a window should appear. Xming does not require this step and should configure your graphical settings by default.

2.4 Important configuration files

There are several files in your home directory that you may need to edit in order to customise the way you access the system. (Editors and some editing How-To is in Section 3.) Filenames of configuration files in UNIX start with a dot (.); such files are hidden from `ls` unless you give it the argument `-a`:

```
> ls -a
.  .Xauthority .bashrc .xinitrc
..  .bash_profile .logout .xsession
```

Files "." and ".." are the current directory, and directory one level up. (Remember that command cd .. changes your directory to one level up.)

The .bash_profile is executed once, when you login - only when you start a 'login shell'. (As differentiated for instance from a non-interactive login shell where you use ssh to execute a single command on a remote machine.)

Your .bashrc file contains commands which are run whenever you start a new shell (when you log in for the first time, open a new window, run any command - a subshell is spawned to run the new command - and so on).

========

NOTE (very important): If your .bashrc OUTPUTS ANYTHING then you WILL not get any job output back from the WorkerNodes. Known issue:

http://www.openssh.org/faq.html

sftp and/or scp may fail at connection time if you have shell initialization (.profile, .bashrc, .cshrc, etc) which produces output for non-interactive sessions. This output confuses the sftp/scp client. You can verify if your shell is doing this by executing:

ssh yourhost /bin/true

If the above command produces any output, then you need to modify your shell initialization.

The solution is to put whatever you want in .bash_profile, *not* .bashrc. .bash_profile is executed *once* per login. .bashrc is executed for *every* command you run. So .bashrc should be short!

========

Some environment variable settings belong in .bash_profile; aliases belong in .bashrc. Some environment settings might need to go into .bashrc.

For example, you may want to add aliases to your .bashrc for commands you use often:

```
alias la="/bin/ls -aF" (F shows if it's a directory or plain file)
alias ll="/bin/ls -alF"
alias lt="/bin/ls -altF" (This shows sorted by most recent modify)
```

To use a CMSSW package, these lines are needed in your .bash_profile:

```
export VO_CMS_SW_DIR=/software/cms
```

```
export SCRAM_ARCH=slc4_ia32_gcc345
.  $VO_CMS_SW_DIR/cmsset_default.sh
```

Setting environment variables is probably best done in .bashrc. The command to do set an environment variable is `VARIABLE=value`, eg `EDITOR=emacs` This sets the variable `EDITOR` to `emacs` You can see what environment variables you currently have set with command `env`.

Important environment variables include:

Name	Effect
DISPLAY	The name of the X display that windowing commands will use by default. If you get 'Can't open display' errors, check this. DISPLAY should be a fully qualified host name followed by :0.0; `> echo $DISPLAY` `oddjob.phy.bris.ac.uk:0.0`
EDITOR	The name of your default editor.
PATH	A colon-separated list of directories to search for commands. The default path is quite minimal.
LD_LIBRARY_PATH	As PATH, but for libraries.
MANPATH	As PATH, but for manual pages.

Your .xsession file gives the commands that will be executed when you start an X-windows session from an X-terminal. You may wish to add commands to this to change your background, open more windows at startup, change your window manager and etc.

3 Text and document processing

3.1 Editors

Various applications are available to edit text on the system, including (in order of sophistication) `pico`, `vi`, and `emacs`.

`emacs` is the recommended text editor; it is powerful and widely used. You can start it with the name of a file to edit:

```
calgary> emacs guide.tex &
calgary>
```

Emacs is a full-screen editor. You can move around with the cursor keys, page up/page down keys, mouse and insert and delete text in the obvious way. Emacs is partially menu-driven; use the mouse to select items from the drop-down menus at the top of the window.

Some useful emacs keystrokes are:

Key combination	Effect
control-X control-C	Quit emacs, prompting to save any changed files
control-X control-S	Save the file currently being edited to disk
control-X control-F	Open a new file
control-A	Move to start of line
control-E	Move to end of line
control-K	Delete from current cursor position to end of line, or (if at end of line already) close up.
control-S	Search forwards
control-S control-S	Repeat previous forward search
control-R	Search backwards
control-_	Undo previous action
control-space	Set mark
control-W	Cut text between cursor and mark
control-Y	Paste previously cut text at cursor
ESC <	Go to top of document
ESC >	Go to end of document
ESC q	Word-wrap current paragraph

Emacs has its own extensive on-line documentation; see the 'Help' menu.

Note that it's not efficient to start a new version of emacs whenever you want to edit a new file. Instead use control-X control-F to open a new file. Use the 'Buffers' menu to switch between files.

Emacs is configurable; you may want to copy someone else's .emacs file and customise it.

3.2 Miscellaneous

Miscellaneous text utilities include `ispell` (spell checker), `wc` (word/line counter), `grep` (search text file for specified text). Use `man wc, man grep` and so on to see the online manual pages for these commands.

4 Programming

The Fortran compiler is `f77`. The C compiler is `cc`. Most of the GNU software such as `gcc, g++, g77` etc are available.

The only shells available are `sh, tcsh` and `bash`. `bash` is the default. You should never need to use `tcsh`. Extensive shell-script programming in `tcsh` is generally considered a bad idea. `perl` is available.

Use of the `make` utility is recommended for anything other than the simplest programming tasks.

5 Jobs and processes

Each program currently executing on a given workstation is called a process, and has a numeric process id (pid). You can get a list of the processes you are currently running with `ps -e -u <your user name>` or list all current processes with `ps -ef`. `top` lists the processes that are currently using the most resources. If your CPU-intensive job on calgary requires no input from you then please use `renice` to flag it as non-interactive priority. Find the pid of your process with `top`, and then `renice 19 <pid>`. This will speed up interactive processes (xterms, shells, etc) while allowing your job to run at full speed when there are no interactions currently happening.

Sensible use of the available CPU and memory is important in order to make sure that the system works well for everyone. Before running a program that is likely to consume large amounts of either, use `top` to check what processes are already running and how much free memory the system has. If in any doubt, run your job on one of the WN using PBS for submission.

6 Portable Batch System (PBS)

6.1 Job Queues

There are three queues: `short`,`medium` and `long`.

Your job will be started provided:

- Free processors are available - each job gets one dedicated processor
- The system has not exceeded the maximum of allowed jobs
- The user has not exceeded the maximum number of jobs allowed per user within the whole system
- All higher priority jobs have been processed
- The queue has not exceeded the maximum number of jobs allowed per user for that queue
- The user has not exceeded the maximum number of jobs per user for that queue
- The job is considered by the scheduler to be the most eligible job to run within the queue

PBS chooses which queue a job will run under according to the maximum CPU time requested by the job; if a CPU time is not requested the default is medium.

Queue	CPU Time (h:m:s)	walltime
short	00:20:00	00:30:00
medium	48:00:00	72:00:00
long	180:00:00	200:00:00

If you think your jobs will run longer than 180 hours cputime (or 200 hours walltime) we can setup an extra-long queue! It is intended that the bulk of "production" work will be run in queue medium; this is the default queue.

6.2 The short queue

It has a CPU limit of 20 minutes and has a higher priority than medium or long queues. Also one jobslot is always reserved for short jobs.

To submit a job to the express queue:

qsub -q short <my.script>

where <my.script> is the file containing the batch job.

More qsub options may be specified if necessary. See 6.4 Job Submission for details.

6.3 Job Submission

The PBS command qsub is used to submit jobs. At its simplist:

qsub <scriptname>

will submit script <scriptname> to the bulk queue. When the qsub command has been obeyed PBS returns the jobnumber which has been given to the batch job; this can then be used with the `qstat` command to monitor the job's progress.

If no CPU time limit is given on a directive in the script, the job will run in the medium queue with a 48-hour cputime limit.

The job on the WN runs in your home directory (which is automounted to the WN). If in your job you want to access files in subdirectories you must use the `cd` command within your script, or give absolute path names explicitly such as: `/users/<you>/CMSSW_1_4_8/`

The **qsub** command has a number of options which can be given either on the command line or as PBS directives at the beginning of the job script. The man page for **qsub** (`man qsub`) gives full details, but below we give the most useful ones.

6.3.1 Jobname

By default the jobname is the same as the local name of the script file. This can be changed by using the `-N` qsub option or perhaps more usefully by putting a PBS directive at the beginning of the script file.

#PBS -N jobname

6.3.2 CPU Time Limit

This is specified as one of the possible reources for the `-l` option.

qsub -l cput=n myscript

sets a limit of n seconds.

qsub -l cput=3:00 myscript

sets a limit of 3 minutes

qsub -l cput=6:00:00 myscript

sets a limit of 6 hours.

The equivalent PBS directive to qsub -l cput=3:00 is #PBS -l cput=3:00

6.3.3 Restartable Job

A restartable job will restart if the batch WN crashes or if the system administrator forces your jobs off the machine.

The option **-r n** says that the job is not restartable. The option **-r y** says that the job can be restarted after a break.

The equivalent PBS directives are #PBS -r n and #PBS -r y

The default is that jobs are restartable.

6.3.4 Job Output

By default the standard output stream is returned to the directory from which the qsub command was obeyed as <jobname>.o<jobnumber>. The standard error stream is similarly returned as <jobname>.e<jobnumber>. You can reroute that standard output stream by using the **-o** option, eg **-o** resultsdir/jobout. Similarly you can reroute that standard error stream by using the **-e** option.

The **-j** option allows you to merge the two output streams together.

-j oe directs that the two streams are merged as standard output.

-j eo directs that the two streams are merged as standard error.

6.3.5 Example Job Script

We use an editor to create a file pbs-job.sh containing:

#PBS -N testjob1
#PBS -l cput=3:30

```
#PBS -j eo
cd subdir
# run my program in my chosen subdirectory subdir
cmsRun Zen.cfg
```

We submit it using:

```
qsub pbs-job.sh
```

This job will be called testjob1, have a CPU limit of 3 minutes and 30 seconds, and the two output streams will be merged as standard output. The output will be returned as testjob1.o<jobnumber>.

Note that the PBS directives must be at the beginning of the file, before any executable line.

If an option is present as both a directive and on a command line, the command line takes precedence. This allows you to override values set in the job script for individual runs.

6.4 Monitoring Batch Jobs

The command **qstat** allows you to check what is happening to your batch jobs. The man page for qstat gives full details of all the options but we just give the most useful ones here. You do not need to know which host your job is executing on, PBS handles that itself.

The command **xpbs &** provides a graphical interface to information using X Windows. It can be rather daunting to use to start with but does have its own extensive help information.

6.4.1 Individual Jobs

It is worth noting the jobnumber assigned by PBS to the job when it is first submitted. Then you can query the state of that individual job:

qstat <jobnumber> gives basic details about the job <jobnumber>

qstat -f <jobnumber> gives full details of that job

qstat -s <jobnumber> says why a job is not running, or, if it has started, when it started.

6.4.2 All Your Jobs

qstat -n shows all jobs and what WN they are running on.

qstat -u <userid> gives information about all jobs under that userid, whether queued or running. You can also see which queues they are in.

qstat -r -u <userid> lists the running jobs for that userid.

6.4.3 Queue Details

The four execution queues are L, M ,S for the bulk routing queue and E for the express routing queue.

qstat <queuename> lists all the jobs in that queue.

qstat -q gives the CPU limit and total job limit for each queue. (Adding a queue name to this command limits the information on that queue).

qstat -f -Q <queuename> gives full information about that queue.

qstat -B -f gives totals and defaults for the whole server.

6.4.4 Cancelling Batch Jobs

Cancelling is done with the **qdel** command. **qdel** <jobnumber> cancels your job with job number <jobnumber>. If it is running the job output is returned as normal. If it is still queued no output is returned.

6.4.5 Altering and Ordering Batch Jobs

qalter <attributes> <jobnumber> modifies the given attributes of the job specified by <jobnumber>. The attributes **-e** path, **-N** jobname, **-o** path and **-l cput**=time can be altered by this command.

qorder <jobnumber1> <jobnumber2> exchanges the order of the two batch jobs in a queue. This cannot be used once a job is running and queue limits cannot be exceeded.

6.5 Job Runtime Environment

6.5.1 Shell Strategy

The free shell stategy used on the Linux farm aims to duplicate the shell choice that would be made if the script was run interactively (ie your default shell would be used unless the shell is specified in the job script). You may override these choices should you wish to use the batch submission option **-s**.

6.5.2 Work Directory ($WORKDIR)

When your batch job starts it will be allocated a work directory on the batch WN. The location of this directory can be obtained by the environment variable $WORKDIR. To use this, **cd $WORKDIR** at the beginning of your job script.

When the batch job finishes, all files remaining in $WORKDIR will be cleaned up. Any that you need must be copied to more permanent storage.

7 Using CMSSW on calgary

7.1 Installing a CMSSW version

CMSSW is a framework designed by physicists on the CMS detector project at the LHC in CERN. Within its structure are many programs and subroutines designed to help simulate fake data before the collider comes online and to study both this simulated data and the real data as it filters in.

To use CMS Software you must first set your environnment and install it.

As detailed earlier, the current setup requires the following environment lines to be added to your .bashrc file

```
export VO_CMS_SW_DIR=/software/cms
export SCRAM_ARCH=slc4_ia32_gcc345
.  $VO_CMS_SW_DIR/cmsset_default.sh
```

Alternatively these commands can be run separately on the server prior to installing the software.

In addition, the line:

```
eval ‘scramv1 runtime -sh‘
```

(Note the back-ticks, not apostrophes!) must be used each time CMSSW is used to set the environment for installation and running of the programs within. An example is below.

The CMS software is installed using the SCRAM arch, as detailed on the CMS Twiki, which will be essential viewing for anyone with even a remote interest in CMSSW: `https://twiki.cern.ch/twiki/bin/view/CMS`

A list of available software versions can be obtained by typing:

```
scramv1 list CMSSW
```

This will display those CMSSW versions currently available on CMSSW - project students using the framework should consult their supervisor about which version is most suited to their work.At the time of writing CMSSW_1_6_9 was the latest version available although 2_0 and higher are coming into use at CERN.

Once a software version is chosen, it can be installed using

```
scramv1 project CMSSW CMSSW_1_X_X
```

where X_X is the project number required, eg. CMSSW_1_6_9. This will copy the neccessary files to your user area and will remain for future use.

To access some of the programs after install and on future logons simply change to the new directory and set your environment:

```
cd CMSSW_1_X_X/src
eval ‘scramv1 runtime -sh‘
project CMSSW
```

Subroutines and programs can now be installed and/or used.

7.2 Subroutines of CMSSW

7.2.1 ROOT

For instance root, the particle physics analysis package, can be loaded by typing `root` or `root -l`

This brings up the program without the startup logo flashing up on screen. To load a root file:

```
root nameoffile.root
```

Root files are produced by many of the CMSSW programs and have excellent documentation at `http://root.cern.ch` They can store histograms and graphs of data as well as stacking many examples of these in 'trees' of directories.

When root loads, the Unix command line disappears and a root one replaces it. (You are now interacting with the root program, not the Unix shell.) To exit this at any time type `.q`

To look directly at the graphs in root files you can use:

```
new TBrowser
```

A new window will then appear in which you can navigate the directories much like in Windows.

Histograms can be created in root to be plotted with data, using for example:

```
TH1F *nameofhistogram = new TH1F("nameofhistogram", "Title of Histogram",
100, 0, 5);
```

where TH1F is the histogram type, and the three numbers at the end are the number of bins, and maximum and minimum value of the x axis respectively.

A histogram such as this can then be filled using `nameofhistogram->Fill(values);`

where the values are specified by the user. A Random gaussian distribution of 10,000 data points is given by :

```
nameofhistogram->FillRandom("gaus", 10000);
```

The histogram can be plotted using: `nameofhistogram->Draw()`

root will automatically draw the histogram on a new 'canvas' c1 which can be divided using `c1->Divide(1,2);` where the two numbers are the xy dimensions of the grid. You can switch between these divisions using `c1->cd(#);` where # is the number of the panel.

Labels, headings, axes and much more can all be altered either graphically or by command line; much more detail is available online at the root.cern.ch website.

7.2.2 PYTHIA

PYTHIA is one of the montecarlo random number generators of data and can be installed using CVS, which installs subpackages from the CERN server.

To login remotely to CVS, use these commands (from `https://twiki.cern.ch/twiki/bin/view/CMS/WorkBookComputingConcepts`):

```
cmscvsroot CMSSW (sets CVS root to CMSSW)
echo $CVSROOT (checks environmental variable, you should see :pserver
anonymous@...)
cvs login (logs in, will prompt for password on first attempt, the
password is 98passwd)
```

Then to install PYTHIA:

```
cvs co -r CMSSW_1_X_X GeneratorInterface/Pythia6Interface
```

pythia, like much of the programs in CMSSW, runs via configuration cards that tell the program what various constants and variables are. Example cards and variables can be found in the PYTHIA manual at `http://home.thep.lu.se/~torbjorn/Pythia.html`

Version 6.4 is the version used by CMSSW versions on calgary and this manual should be used at present - not the more recent version 8.1.

Configuration cards are text files with commands with a .cfg suffix and they are run by PYTHIA using the command

```
cmsRun nameofcard.cfg
```

This command can later be used to run much more complicated cards that

run the simulated montecarlo data through simulation and reconstruction of the detectors of CMS, as detailed in
`https://twiki.cern.ch/twiki/bin/view/CMS/WorkBookEntireRecoChain`
Both processes output the data as large root files. Whilst PYTHIA is quick to produce pure MC data, the longer reconstruction processes can take many hours to do small numbers of events and it is recommended to use calgary's dedicated WN for this processing.

7.2.3 EDAnalyser packages

Data produced in root files is studied using EDAnalyser C++ packages, written by the user to pick out useful data.

To make an EDAnalyser:

```
cd CMSSW_1_X_X/src
mkdir adir
mkedanlzr nameofedanalyser
```

This will create the skeleton form of an EDAnalyser package in the directory adir. An EDAnalyser has two main components:

- a .cc file in the 'CMSSW_1_X_X/src/adir/nameofedanalyser/src' directory in which all the analysis code is written;
- a .cfg file in the 'CMSSW_1_X_X/src/adir/nameofedanalyser/test' directory that runs the actual analysis (using the command cmsRun name.cfg as in the PYTHIA case).

In addition the build file in the 'nameofedanalyser' package will need some initialising code to ensure all modules are loaded.

Examples of how to use analysers can be found at
`https://twiki.cern.ch/twiki/bin/view/CMS/WorkBookWriteFrameworkModule`
and examples of different actual packages for various CMSSW version are available on the LXR: `http://cmslxr.fnal.gov/lxr/`

Much of the work to be done is in the .cc file, which will require understanding of c++. The hardest thing is to find the correct labels to load data types from root files. A few common ones are listed below. Note that these need the correct include file at the start of the .cc file to function correctly.

```
//get electrons from the event
#include "DataFormats/EgammaCandidates/interface/PixelMatchGsfElectron.h"

edm::Handle<PixelMatchGsfElectronCollection> gsfElectrons;
iEvent.getByLabel("pixelMatchGsfElectrons",gsfElectrons);

//get jets from event
#include "DataFormats/JetReco/interface/CaloJetCollection.h"
#include "DataFormats/JetReco/interface/Jet.h"

Handle<CaloJetCollection> jets;
iEvent.getByLabel("iterativeCone5CaloJets", jets);

//get tracks from event
#include "DataFormats/TrackReco/interface/Track.h"

edm::Handle<TrackCollection> tracks;
iEvent.getByLabel("ctfWithMaterialTracks",tracks);

//get HepMC virtual particles from event
#include "SimDataFormats/HepMCProduct/interface/HepMCProduct.h"

edm::Handle<edm::HepMCProduct> hepMC;
iEvent.getByLabel("source",hepMC);

//get missing transverse energy - MET - from event
#include "DataFormats/METReco/interface/CaloMETCollection.h"

Handle<CaloMETCollection> calo;
iEvent.getByLabel("met", calo);

//get b tag information
#include "DataFormats/BTauReco/interface/JetTag.h"

edm::Handle<reco::JetTagCollection> bTagHandle;
iEvent.getByLabel("trackCountingHighEffJetTags", bTagHandle);
const reco::JetTagCollection &bTags = *(bTagHandle.product());
```

These data types can then be referred to in the main body code, for instance you could run through all HepMC particles in the file:

```
for ( HepMC::GenEvent::particle_const_iterator
mcIter=myGenEvent->particles_begin(); mcIter != myGenEvent->particles_end();
mcIter++ ) {
body of loop
```

}

(The `for` line has been wrapped for legibility)

Within that loop you could, for instance, fill histogram h1 (which you had earlier declared) with the transverse energy of all HepMC particles that were electrons (particle id = 11 from the monte carlo rulebook):
`http://pdg.lbl.gov/mc_particle_id_contents.html`

```
if((*mcIter)->pdg_id()==11)  h1->Fill(mcIter.Et());
```

All of the data types listed above work in similar ways although some research of the CMS Twiki will be needed to get the exact parameters that can be referenced for each (or deduced from the root files). By combining a number of loops, collecting data and cutting out those particles and events that don't match researched criteria, the user can create a completed Analyser package.

To run a completed EDAnalyser change to the /nameofedanalyser directory (`cd nameofedanalyser` or `cd /path/to/nameofedanalyser` and use:

```
scramv1 b
```

(Note: after the first build you can use scramv1 b -f to build faster. In addition if the build doesn't appear to have worked you can use `scramv1 b clean` to remove all traces of previous installs of the analyser and then can build again.)

Once the project is built you can then change to the /test directory and run your configuration file using `cmsRun name.cfg`.