Table of Contents

Ganga Tutorial 1

We start with a very simple problem which doesn't require knowledge about any other LHCb software - factorizing prime numbers. Be careful what you copy and paste from this wiki as python doesn't like random white space (it's best if you don't copy and paste anything).

Ganga Setup

First, set the environment for Ganga in a fresh terminal:

```
SetupProject Ganga
```

which gives you the latest version (5.4.3 at time of writing). If you have last used `ganga` before version `5.4` you may need to do

```
ganga -g
```

to update your `.gangarc` file to include some of the newer options.

For this tutorial, we need to add an extra Ganga configuration file, so depending on your shell do

```
setenv GANGA_CONFIG_PATH ${GANGA_CONFIG_PATH}:/afs/cern.ch/user/j/jwilliam/public/GangaTutorial/T
```

or for bash-like shells

```
export GANGA_CONFIG_PATH=${GANGA_CONFIG_PATH}:/afs/cern.ch/user/j/jwilliam/public/GangaTutorial/T
```

If you don't have access to `/afs/cern.ch/user/j/jwilliam/public/GangaTutorial` (e.g. you're using a machine that doesn't have afs or if something is wrong w/ my afs account), then you can do everything in this tutorial except submitting to the grid by simply doing this instead:

```
setenv GANGA_CONFIG_PATH ${GANGA_CONFIG_PATH}:GangaTutorial/Tutorial.ini
```

or for bash-like shells

```
export GANGA_CONFIG_PATH=${GANGA_CONFIG_PATH}:GangaTutorial/Tutorial.ini
```

You will not need to do either of these for standard LHCb running.

Now start an interactive session:

```
ganga
```

```
In [1]:
```

You should now see the Ganga prompt! Check to make sure that the application for this tutorial was loaded (we need `PrimeFactorizer`):

```
In [1]:plugins('applications')
```

Ignore the warning if you don't have a valid Grid proxy (you should only see this once; we'll create the proxy when we need it below). You can check which plugins are available to you in each category in your current Ganga session using `plugins`. Try using the `help` utility to see if you can figure out how to list all of the

available plugins in all categories:

```
In [2]:help(plugins)
```

This runs `less`, so type `q` to exit. Ganga provides `help` information on just about every object, method, etc. Try this first if you get stuck.

A Local GangaTutorial

If you would like to run the files locally, copy them to a directory you own and change the contents of the `Tutorial.ini` file.

```
cp -R /afs/cern.ch/user/j/jwilliam/public/GangaTutorial ~/public/
```

The `Tutorial.ini` file should be edited to look something like this.

```
[Configuration]
RUNTIME_PATH = /afs/cern.ch/user/a/auser/public/GangaTutorial
```

Now add your `Tutorial.ini` file to `GANGA_CONFIG_PATH`.

```
setenv GANGA_CONFIG_PATH ${GANGA_CONFIG_PATH}:/afs/cern.ch/user/a/auser/public/GangaTutorial/Tuto
```

or for bash-like shells

```
export GANGA_CONFIG_PATH=${GANGA_CONFIG_PATH}:/afs/cern.ch/user/a/auser/public/GangaTutorial/Tuto
```

Prime Number Factorization

In this tutorial, our task is to find the prime factors of a given integer. Finding very large prime factors requires a lot of CPU time. This tutorial provides code that can factorize any number whose prime factors are among the first 15 million known prime numbers. We have 15 tables of 1 million prime numbers each and we can scan the table in search of the factors. The python modules we will use (which are already written for you) include a collection of prime number tables called a `PrimeTableDataset` which are used by the `PrimeFactorizer` application.

Running a Factorization Job using Ganga

Let's start with a small example. The goal is to find the prime factors of the integer 1925. For such a small number, we (clearly) only need the first prime number data table (recall that each table contains 1 million prime numbers). At the Ganga prompt, type the following:

```
In [1]: j = Job()
In [2]: j.application = PrimeFactorizer(number=1925)
In [3]: j.inputdata = PrimeTableDataset(table_id_lower=1, table_id_upper=1)
```

At this point, we've created a `Job` object but we haven't run anything yet. We're free to edit its attributes as much as we like prior to submitting the job. For actual LHCb jobs, the `application` might be `DaVinci` while the `inputdata` could be a list of LHCb data files. The idea and most of the syntax are the same though as in this simple example. To see all of the job's attributes, do

```
In [4]:j
```

Notice that the backend is set to Local (which is the default value since we didn't specify where we wanted the job to run). This means that the job will run in the background on the local machine.

OK, let's submit the job:

```
In [5]: j.submit()
```

We can check the status of the job by doing

```
In [6]:j.status
```

This will either be submitted, running or completed. If the job hasn't finished yet, wait for a few seconds and check again (for such a small number, the job should finish very quickly).

We can see what files were output by the job by doing

```
In [7]:j.peek()
```

All Ganga jobs return the standard output and error in the files stdout and stderr. This job has also produced the file factors-1925.dat. We can view the contents of this file using

```
In [8]:j.peek('factors-1925.dat')
```

which opens the file using less (use standard less commands to scroll etc., type q to quit).

The file should contain the factors [(5, 2), (7, 1), (11, 1)], let's check if this is correct:

```
In [9]:(5**2)*7*11 == 1925
```

Remember, standard python syntax works at the Ganga prompt!

OK, so we've run a job and checked the output using Ganga's magic but for a real analysis you'll often want *direct* access to the file. So, where is factors-1925.dat? It's in the job's output directory. You can obtain the full path of this directory via

```
In [10]:j.outputdir
```

This is a *normal* directory that you own; thus, you have permission to access the files there from a process independent of Ganga. So, you could exit Ganga and examine factors-1925.dat using, e.g., cat on the Linux command line...or, you could do this from Ganga. You can access shell commands from the Ganga prompt using ! as follows:

```
In [11]:!ls ~/.globus
```

```
In [12]:!cat $j.outputdir/factors-1925.dat
```

Notice that you can use the $ character to access python variables when using the ! to access shell!

A few other basic *convenience* features which you can play around with involve scrolling through the history and using the TAB completion. Try using the arrow-UP to scroll through the history of the Ganga commands you've executed so far (works the same as when in a shell). You can use TAB completion on keywords,

variables, objects, etc. Try the following (where `TAB` and `arrow-UP` mean hit those keys, don't type it out):

```
In [13]:j.app<TAB>

In [13]:j.application

In [13]:j.application<arrow-UP>

In [13]:j.application = PrimeFactorizer(number=1925)
```

The `arrow-UP` key scrolls through the history of commands that match what's been typed so far. In this case it scrolls through all commands which start with `j.application` (which is only 1 command so far, but try it again latter on in the tutorial!). This behavior is similar to using `ESC-P` in `tcsh` or `CTRL-R` in `bash`.

Splitting a Ganga Job into Multiple Concurrent Jobs

Now that you've seen some of the basics of Ganga, let's try something a little more interesting - factorizing a very large integer. For this we'll need a `PrimeTableDataset` which contains all 15 tables of prime numbers. To speed things up, we will also split the job into 5 local `subjobs` which will run concurrently.

First, define a job as before but w/ a larger number and using all 15 prime number tables (feel free to use the `arrow-UP` and `TAB` keys to do this instead of typing it all out!):

```
In [1]: j = Job()
In [2]: j.application = PrimeFactorizer(number=118020903911855744138963610)
In [3]: j.inputdata = PrimeTableDataset()
In [4]: j.inputdata.table_id_lower = 1
In [5]: j.inputdata.table_id_upper = 15
```

Now add a splitter to divide up the task of finding all the prime factors (here we'll make 5 *subjobs*):

```
In [6]: j.splitter = PrimeFactorizerSplitter(numsubjobs=5)
```

For LHCb jobs, similar splitters are provided to split jobs up which run on multiple data files, etc.

We also want to add a merger to merge the output from each of the 5 subjobs:

```
In [7]: j.postprocessors = TextMerger(files=['factors-118020903911855744138963610.dat'])
```

When all 5 subjobs are complete, the merger will merge the contents of each of the 5 `factors-118020903911855744138963610.dat` files into a single file in the master job's output directory (we'll look at what this means below).

OK, now submit the job (actually, the 5 jobs) just like we did above:

```
In [8]: j.submit()
```

You can check the status off all 5 jobs by simpy doing:

```
In [9]:j.status
```

If any of the jobs is still running, the status of the master job will be listed as `running`. If all 5 jobs are `completed`, the master job's status will also be `completed`. Wait until all 5 jobs are done (should take less than a minute) before moving on (in the mean time you can play around with `help`, e.g. try `help(j.submit)`...remember, type `q` to quit).

Once the jobs are complete, let's look at the output of one of the subjobs (do exactly what we did above):

```
In [10]:j.subjobs[2].peek()
```

```
In [11]:j.subjobs[2].peek('factors-118020903911855744138963610.dat')
```

The file should contain the factor [(141650963, 1)]. Each of the `j.subjobs` is itself a `Job` (try printing it), so you can do anything you would do on an *independent* job on the subjobs.

Now examine the merged output of all the jobs:

```
In [12]:j.peek()
```

```
In [13]:j.peek('factors-118020903911855744138963610.dat')
```

The file should contain the factors [(2, 1), (3, 1), (5, 1), (7, 1), (15485867, 1)] [] [(141650963, 1)] [] [(256203221, 1)] (some of the prime number tables don't contain any factors of this particular number). You can check if they're right on the Ganga prompt like we did above. Notice that the *master* job doesn't have the `stdout` and `stderr` files since itself was never actually run. In fact, had we not added the merger to the job there would be no output in the master job's directory.

Running Ganga Jobs on the Grid

NOTE: This appears to be broken, as of 17 Dec 2012, and ganga complains that is_prepared is not set for PrimeFactorizer. Perhaps this error is why there is an attached document with a corrected PrimeFactorizer.py file, which is supposed to be an updated version for Ganga > 5.7 (current is 5.8)

For many LHCb jobs (which often involve processing large amounts of data), running concurrently isn't enough. When a large number of CPU's is required for a job, we need the grid! Specifically, we want to run on the LHC Computing Grid (LCG). For LHCb jobs, this involves the DIRAC workload manager.

As an example of running on the grid, we'll run the same set of jobs we ran above but using a different `backend`. We could retype all of the required info from the previous job definition or, better yet, we could use the `TAB` and `arrow-UP` functionality to re-enter the info. An easier way is to just copy the previous `Job` object, then change the `backend` so that the jobs run on the `Dirac`:

```
In [14]: j = j.copy()  # we could've also used Job(j), etc.
In [15]: j.backend = Dirac()
In [16]: j.outputfiles =  [SandboxFile('factors-118020903911855744138963610.dat')]
```

Notice that we had to add a SandboxFile to the `outputfiles`. This is to tell `ganga` that we want this file returned to us after the grid job is completed.

Now just submit the jobs the same way as before (since this is the 1st time we've done something that requires a grid proxy, you'll be asked for your grid password if you don't currently have a valid grid proxy on this machine):

```
In [17]: j.submit()
```

Congratulations! You've just submitted 5 jobs to the LCG grid via Dirac.

Let's check the status of the jobs:

```
In [17]: j.subjobs
```

Notice that the hostname of the computer which ran (or is running if the job hasn't finished yet) the job is displayed along with the current status. Hopefully your jobs will start soon, but it's possible (depending on where the job is running) that some of your jobs will stay in the `submitted` state for a while. If all the jobs are finished, go ahead and check the master's output. If some are still running, check some of the subjobs output and check that it matches what was output by the same subjob when run locally. Once any of the jobs is `running` or `completed`, you've run on *The Grid*!

Running a Later Version of Ganga

If you are running a version of Ganga that is at least v5.7.0, you may encounter problems when submitting the jobs to the Grid. To resolve these the application needs to be converted to a prepared application. This requires creating a local version of `GangaTutorial`, as described above.

Inside the `Lib` directory of the `GangaTutorial` folder, download the corrected version (you may need to specify the `-k` flag to ignore the unknown CERN certificate_.

```
curl https://twiki.cern.ch/twiki/pub/LHCb/GangaTutorial1/PrimeFactorizer.py.txt -o PrimeFactorize
```

This converts the PrimeFactorization app to a prepared app.

Recreate the job in Ganga (from scratch, not using `j.copy()`) and try submitting the job to the Grid (Dirac) again.

A Few Other Features

The Job Registry

All of the jobs you've ever run (and not deleted) are contained in the list `jobs`:

```
In [1]: jobs
```

If we wanted to rerun the first job, we could do the following:

```
In [2]: j = jobs(0).copy()
In [3]: j.submit()
```

The last job can always be accessed using the python list directly using `jobs[-1]`.

Job Templates

Often times when running LHCb jobs you will want to rerun a *type* of job (e.g. Monte Carlo production jobs). Rather than always copying a previous job, you could set up a template of it. To template the first job we ran, do

```
In [1]:t = JobTemplate(jobs(0))

In [2]:t.name = 'small-prime-factorizer'
```

You don't have to name it, but this will be useful later on to help you find the template you're looking for. The list of all your job templates is stored in the python list `templates` (the same way jobs are stored in `jobs`). Try printing it.

Now, create a new job from the template and run it:

```
In [3]:j = Job(t) # or j = Job(templates(0)),...

In [4]:j.submit()
```

Job templates are quite useful due to the fact that they're easy and fast to search through.

Removing Jobs

If you want to remove a job to save disk space or just because it's obsolete, simply do (try it):

```
In [1]: jobs


In [2]:jobs(0).remove()


In [3]: jobs
```

This removes the job workspace (i.e. the output directory and all output files) and all traces of the job in Ganga's registries....so be careful when doing this!

The GANGA Box

You can persist (store) any GANGA object in the GANGA box. E.g., you could create a bookkeeping query object:

```
In[1]: bkq = BKQuery()
In[2]: bkq.path = '/LHCb/Collision09/Beam450GeV-VeloOpen-MagDown/Real Data + RecoToDST-07/9000000
```

which can be used at any time to get an up-to-date list of LHCb data files obtained by this query by doing:

```
In[3]: data = bkq.getDataset()
```

This data could then be used as the input data for Ganga jobs. To store this object so that you don't need to recreate it every time you want to update the query, simply do:

```
In[4]: box.add(bkq,'example bk query')
```

You can then access this object at any time. E.g., try quitting and restarting GANGA and then do:

```
In[1]: data = box['example bk query'].getDataset()
In[2]:data[0]
```

Writing Your Own Functions

You can also write your own functions and load them into Ganga. Exit Ganga and create the file `~/.ganga.py`:

Ganga will automatically load this file, so restart it and try the following:

```
In [1]:foo()
```

As an exercise, try and write your own function that creates a job from your "small prime numbers" template, submits it and returns a reference to the `Job` object.

Etc...

There are many more features in Ganga which I don't have time to cover here. Remember to use the `help` function if you're unsure about something (try `help()` if you're unsure about everything!). There are also answers to common questions on the FAQ wiki page and the user's guides are also quite useful.

Good luck and happy grid-ing!

-- MikeWilliams - 09 Jan 2009

This topic: LHCb > GangaTutorial1
Topic revision: r14 - 2015-01-28 - NikitaKazeev

Copyright &© 2008-2020 by the contributing authors. All material on this collaboration platform is the property of the contributing authors.
Ideas, requests, problems regarding TWiki? Send feedback