

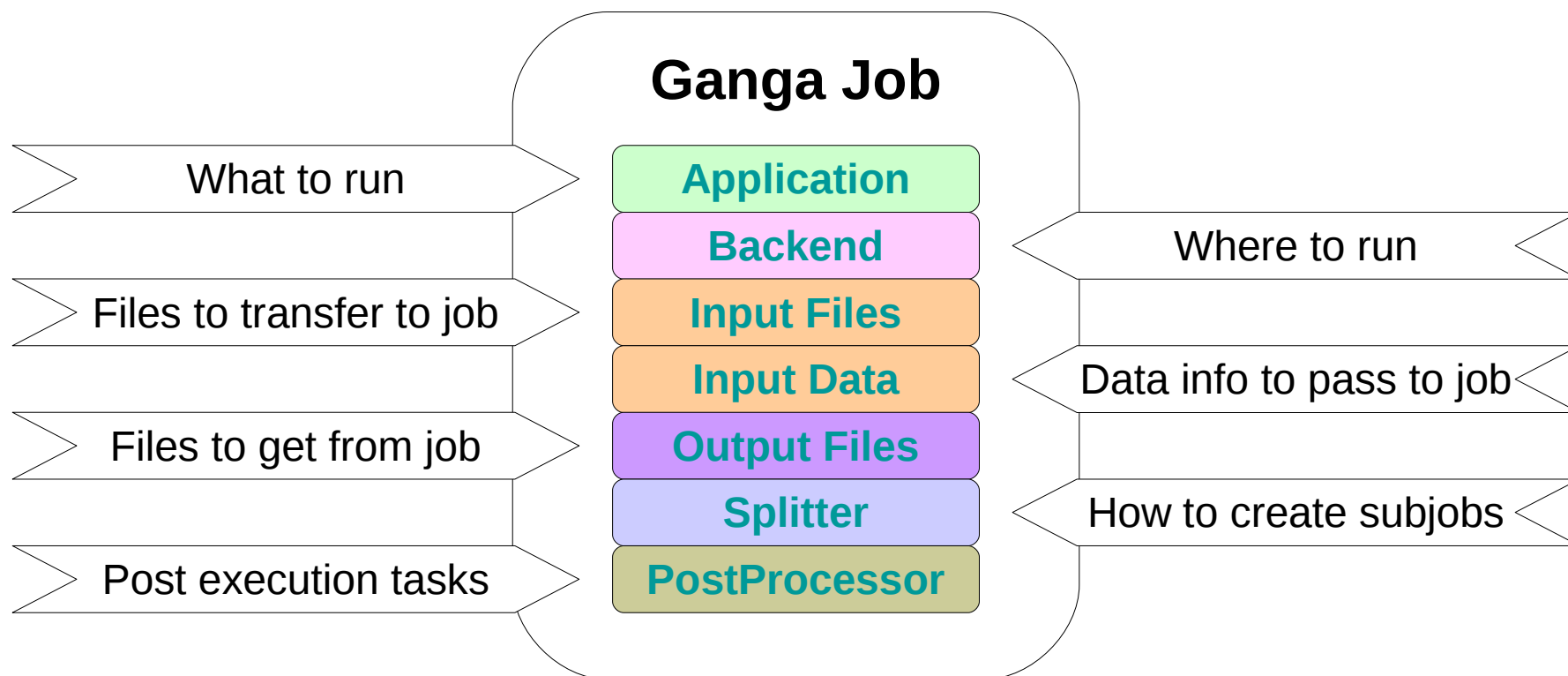


Recent Ganga Updates

GridPP Meeting, 30th April, 2015
Mark Slater, Birmingham University

For those who don't know, Ganga is a general job management tool used by many HEP experiments and beyond to simplify the submission and monitoring of both local and grid based tasks

It is built on the idea of *independent modules* that perform the various functions required by a typical job



There are several reasons why Ganga can help with job submission and management:

- It provides a common 'API' for many different backends
- Multiple ways of submission (command line, IPython, Service, etc)
- Hides much of the complexity for monitoring, etc.
- Easily customisable/expandable to suit the need and situation
- Written in plain python so will work with almost everything
- Active developers on hand to help
- Significantly lowers the barrier to entry for the grid
- Has many advanced features for performing complex tasks

Typical Job Submission

Ganga provides you with interfaces to the application your running (if available) and the backend you're running on. This allows you to completely configure the job and how it's run

```
In [2]:j = Job()
In [3]:j.application = Executable()
In [4]:j.application.exe = File("test.sh")
In [6]:j.inputfiles = [ LocalFile("calib.root") ]
In [7]:j.outputfiles = [ LocalFile("out.txt") ]
In [8]:j.backend = LCG()
In [9]:j.backend.requirements.cputime = 3600
In [10]:j.submit()
```

Start by creating a basic Job object

Set what you want the job to do (run the test.sh script)

What extra files are needed to run the job?

What output is this script going to produce?

Where do you want it to run (the LCG grid in this case) and configure this as you want

And finally, submit the job!

After submission, you can use Ganga to monitor and manage your jobs using Ganga's IPython interface by just typing 'ganga'

Ganga keeps track of all your jobs over all backends and gives you access to all the information using a local job repository

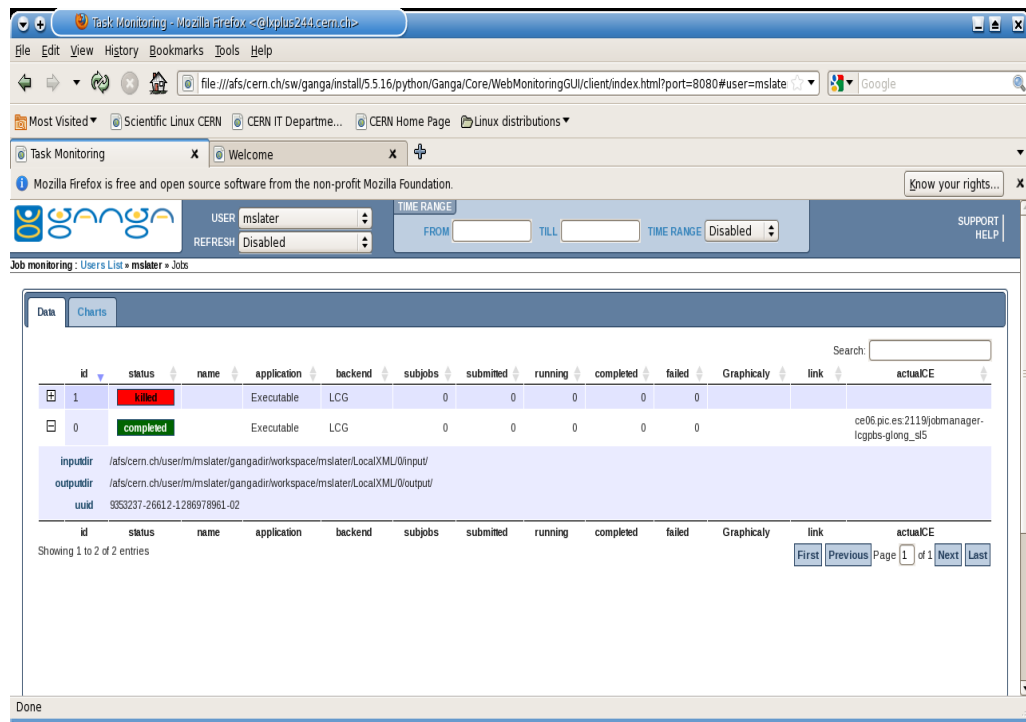
There is also a web gui available by starting Ganga with the --webgui option

```

Ganga.GPDev.Lib.Job : INFO job 7.5 status changed to "failed"
Ganga.GPDev.Lib.Job : INFO job 7 status changed to "failed"
Ganga.GPDev.Lib.Job : INFO job 6.4 status changed to "completing"
Ganga.GPDev.Lib.Job : INFO job 7.3 status changed to "failed"
Ganga.GPDev.Lib.Job : INFO job 7.4 status changed to "failed"
Ganga.GPDev.Lib.Job : INFO job 6.4 status changed to "failed"
Ganga.GPDev.Lib.Job : INFO job 6 status changed to "failed"
Ganga.GPDev.Lib.Job : INFO job 7.2 status changed to "completed"
Ganga.GPDev.Lib.Job : INFO job 7.6 status changed to "completed"
Ganga.GPDev.Lib.Job : INFO job 7.1 status changed to "completed"
Ganga.GPDev.Lib.Job : INFO job 7.0 status changed to "completed"
Ganga.GPDev.Lib.Job : INFO job 6.5 status changed to "completed"
Ganga.GPDev.Lib.Job : INFO job 7.8 status changed to "completed"
Ganga.GPDev.Lib.Job : INFO job 7.7 status changed to "completed"

In [1]: jobs
Out[1]:
Registry Slice: jobs (8 objects)
-----
fqid | status | name | subjobs | application | backend | backend.actualCE
-----
0 | completed | | | TagPrepare | Local | epdt107.ph.bham.ac.uk
1 | new | | | | | 
2 | failed | jpt_test | 15 | Athena | Panda | 
3 | failed | jpt_test | 15 | Athena | Panda | 
4 | failed | jpt_test | 15 | Athena | Panda | 
5 | new | | | | | 
6 | failed | exe_test | 9 | Athena | LCG | 
7 | failed | | | | | 

In [2]: jobs(2).subjobs
Out[2]:
Registry Slice: jobs(2).subjobs (15 objects)
-----
fqid | status | name | subjobs | application | backend | backend.actualCE
-----
2.0 | failed | jpt_test | | Athena | Panda | ANALY_LRZ
2.1 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.2 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.3 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.4 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.5 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.6 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.7 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.8 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.9 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.10 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.11 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.12 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.13 | completed | jpt_test | | Athena | Panda | ANALY_LRZ
2.14 | failed | jpt_test | | Athena | Panda | ANALY_QMUL
  
```



The screenshot shows the Ganga web GUI interface. At the top, there's a navigation bar with 'Task Monitoring' and 'Welcome'. Below that, there are search and filter options for 'USER' (mslater) and 'TIME RANGE'. The main content area displays a table of job monitoring data with columns for 'id', 'status', 'name', 'application', 'backend', 'subjobs', 'submitted', 'running', 'completed', 'failed', 'Graphically', 'link', and 'actualCE'. The table shows two entries: one with status 'killed' and another with status 'completed'. Below the table, there are navigation buttons like 'First', 'Previous', 'Page 1 of 1', 'Next', and 'Last'.

There are a number of plugins supplied with Ganga Core apart from the experimentally specific ones for e.g. LHCb and Atlas:

Backends

Local – The computer you're on
PBS – Torque/Slurm style batch system
SGE – Sun Grid Engine
Condor
LCG – Glite WMS Grid
CREAM – Direct CREAM CE
Dirac – The Dirac WMS
ARC – Direct ARC CE

Input/OutputFiles

LocalFile – A Local (direct access) File
DiracFile – File handled by DIRAC DMS
MassStorageFile – e.g. CASTOR
LCGSEFile – Glite/LFC Storage
GoogleFile – Google Drive
CERN Box File – The new Cloud service
WebDAV File – In development

Applications and Splitters ●

As each individual/VO's needs are going to be very dependent on the software they use and how it can be split, there are only a few generic ones available to run Executable or Root jobs and split any particular parameter of the application

The input/output file system is very powerful and can be used to include and send files to and from all sorts of systems:

- Input Files specified in the job will be transferred to the WN
- Output files are checked for by the job and sent back as requested
- Wildcards are supported for both file types
- Ganga will handle transfers from different systems in the background
- Very configurable depending on how/where you want the transfer
- Allows easy transfer of output data → input data
- Can be used standalone as an API to storage systems

You can also specify multiple different files for the same job, e.g.:

```
In [1]:j=Job()
In [2]:
In [2]:j.outputfiles = [ SandboxFile('BdKsMuMu.root'), DiracFile('Stripped.Dimuon.dst') ]
In [3]:j.outputfiles
Out[3]: [SandboxFile (
  namePattern = 'BdKsMuMu.root' ,
  compressed = False ,
  localDir = ''
), DiracFile (
  namePattern = 'Stripped.Dimuon.dst' ,
  guid = '' ,
  compressed = False ,
  localDir = None ,
  lfn = '' ,
  failureReason = '' ,
  locations = []
)]
In [4]:j.submit()
```

A LocalFile (NOT SandboxFile!)
has been declared that will be
copied back to the local machine

A DiracFile will copy the output to
to Dirac storage and can be
downloaded using the 'get' method
on job completion

Though there are many experimentally specific issues associated with handling input data, Ganga provides a generic File handling and splitter modules for the most basic use case

This GangaDataset object takes a set of GangaFiles and provides a text file to the job that lists the paths for these files. If required, it can also download them. The Splitter will create subjobs based on these input files

```
j = Job()
j.application.exe = "test.sh"
j.inputfiles = [ LocalFile("test.sh") ]

j.inputdata = GangaDataset()
j.inputdata.files = [ LocalFile("/home/mws/*.txt") ]

j.splitter = GangaDatasetSplitter()
j.splitter.files_per_subjob = 4

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

Specify any data files you want to run over. These can be any type of GangaFile

The Splitter will split the input data files across the appropriate number of subjobs

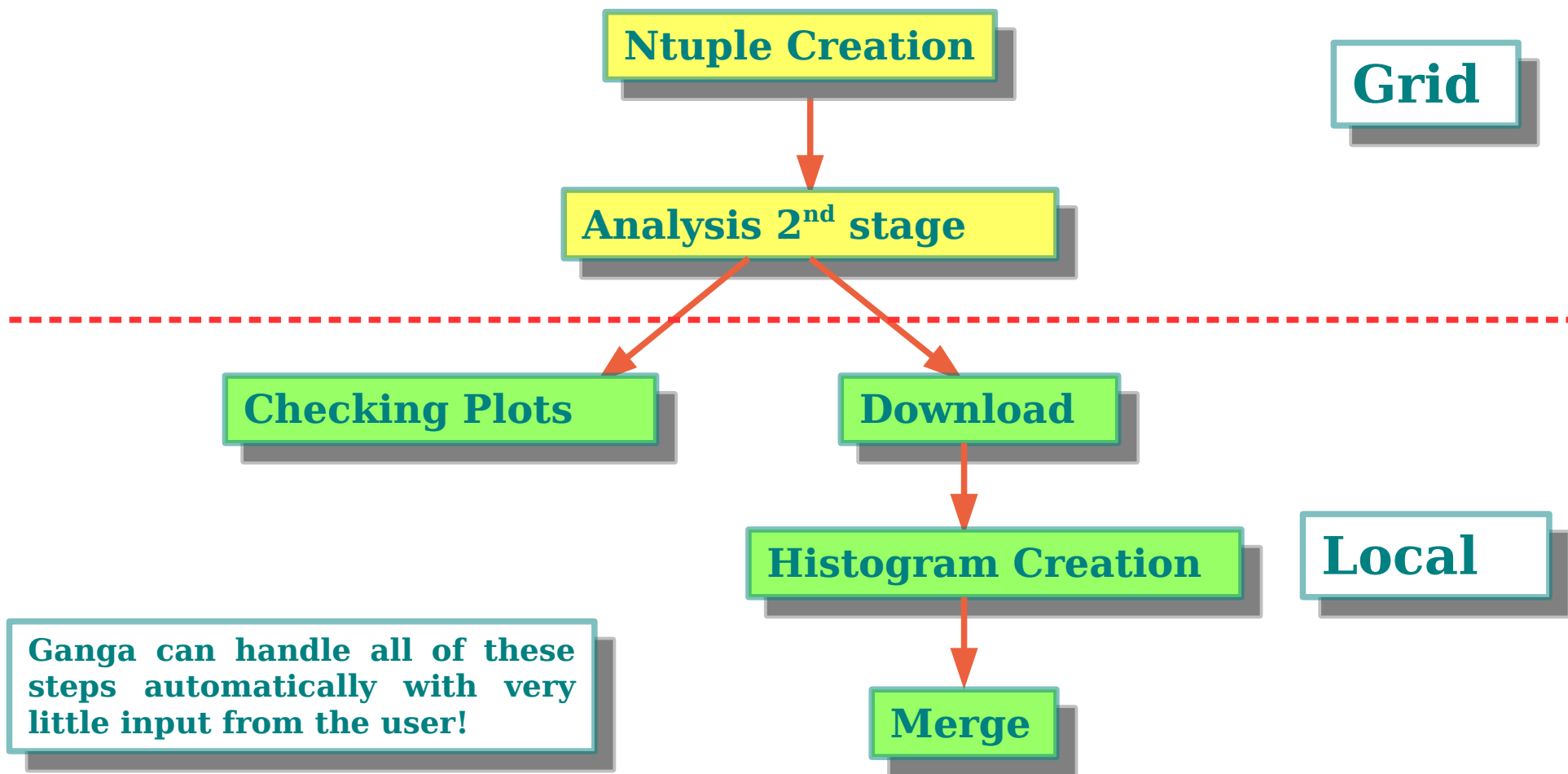
As analyses increase in size and complexity, users will encounter scaling problems:

- More failures on both grid and locally ●
- Keeping track of 100s-1000s of jobs both locally and on the grid ●
- Having multi-stage analyses across multiple systems ●

Originally developed for Atlas, Ganga provides a framework to handle these issues – Tasks – which will (while Ganga is running):

- Resubmit failed jobs if it seems sensible to do so ●
- Submit more jobs when others complete ●
- Monitor and track running jobs, storing output data in appropriate containers ●
- Automatically retrieve and merge data ●
- Chain types of jobs together across any backend ●

Typical Analysis Example



A number of jobs in Ganga can take some time, e.g. job submission, output download, etc. and quite often, these happen synchronously and hold up the user.

A similar problem existed in monitoring in that, even though jobs could in theory be monitored in parallel, this wasn't possible given the previous implementation

To get around this, a thread pool was created to handle system and user generated 'Ganga-aware' threads that:

Execute commands asynchronously ●

Allow tasks that can be carried out independently to be done so ●

Balance performance issues by limiting the no. of concurrent threads ●

Viewing the status of the queues:

```
In [4]: queues
Out[4]:
```

Ganga user threads:			Ganga monitoring threads:		
Name	Command	Timeout	Name	Command	Timeout
Worker_0	idle	N/A	Worker_0	idle	N/A
Worker_1	idle	N/A	Worker_1	idle	N/A
Worker_2	idle	N/A	Worker_2	idle	N/A
Worker_3	idle	N/A	Worker_3	idle	N/A
Worker_4	idle	N/A	Worker_4	idle	N/A

```
Ganga user queue:
-----
[]
Ganga monitoring queue:
-----
[]
```

Adding function calls to the queue:

```
In [3]: queues.add(j.remove)
```

```
In [4]: Ganga.GPIDev.Lib.Job : INFO removing job 8
```

```
In [5]: def f(x):
...:     print x
...:
```

```
In [6]: queues.add(f, args=(123,))
```

```
In [7]: 123
```

Ganga already provides a Template system for storing often used job descriptions. As an aid to the user, we have extended this to allow these templates to be named and included in a Ganga release

This allows documentation to simply say: Use template 'Basic Analysis' and change X and Y values

This also now includes everything that can be put in the Ganga Box (the storage area for Ganga Objects), e.g. “Use Backend template 'LCG setup 1' and it should work.”

These will start to be included in GangaCore to go along with the GridPP User Guides

Previously, it was only possible for a user to specify the type of Merger they wanted to be performed on a job after completion

We have now extended this to include 3 types of post processor (but with the option to extend to arbitrary objects):

Mergers ●
Checkers ●
Notifiers ●

Several types of these objects are already available, e.g. LHCbFileMerger, FileChecker, EmailNotifier

These are added to the new postprocessor job attribute that replaces the merger attribute. This is a list and so you can have as many as you want operate on one job

A Custom Merger is also provided so you can execute any arbitrary code at job completion

Simply provide a python file with a check function that returns a bool:

```
import ROOT
import os

def check(job):
    outputfile = os.path.join( job.outputdir, 'BdKsMuMu.root' )
    if not os.path.exists( outputfile ): return False
    if os.path.getsize( outputfile ) <= 100: return False

    tfile = ROOT.TFile( outputfile )
    tree = tfile.Get( 'Jpsi_Tuple/DecayTree' )
    return tree.GetEntries() >= int( job.events['input'] )
```

```
In [5]: cc = CustomChecker()
In [6]: cc.module = './checker.py'
In [7]: cc.checkSubjobs = True
In [8]: j.postprocessors.append( cc )
```

It is also relatively simple to create your own from scratch!

The GangaService was developed to fill more production roles with Ganga or when long term submissions, etc. need to be run

It behaves as you would expect a service to behave. It can:

Run Ganga as a daemon ●

Access a running Ganga instance using a simple API ●

A typical use of the GangaService is shown:

```
from GangaService.Lib.ServiceAPI.ServiceAPI import
GangaService

gs = GangaService()
gs.gangadir = "/home/mws/TaskTest/gangadir-server"
gs.prerun = "export GANGA_CONFIG_PATH='GangaAtlas/Atlas.ini'"
gs.gangacmd = "/home/mws/Ganga/install/6.0.21-
hotfix2/bin/ganga"

print gs.sendCmd("""
j = Job()
j.submit()
""")

gs.killServer()
```

Import the API

Setup how you want
the Ganga service to
run

Send the commands as
if you were in IPython.
The returned string is
stoud/err

After some discussions with CERN IT, we have finally got a CVMFS repository for Ganga

This will always be kept up-to-date with the latest release and over time will include helper scripts for setting up the initial Ganga configuration

From a clean SL6 shell, you can now do:

```
bash-4.1$ ls /cvmfs/ganga.cern.ch/Ganga/install/
6.0.44  LATEST

bash-4.1$ /cvmfs/ganga.cern.ch/Ganga/install/LATEST/bin/ganga

*** Welcome to Ganga ***
Version: Ganga-6-0-44
Documentation and support: http://cern.ch/ganga
Type help() or help('index') for online help.

Ganga.Utility.Config      : INFO      reading config file /home/mws/.gangarc

In [1]:
```

One of the main advantages with Ganga is that you can relatively easily develop plugins to handle the specifics of certain elements of your use case

Any plugin type can be added, however the most usual are:

Applications

These usually provide a set of parameters that map to the specific program to be run and perform some checks to make sure they are sensible

Splitters

These can be dependant on how you want to split up a master job into smaller subjobs, e.g. by input files or parameters and can therefore provide a simpler interface than the generic splitter provided

If these plugins are designed correctly, they will fit seamlessly into the Ganga framework and will be able to take advantage of all the other functionality

I plan to add significant material to the GridPP Wiki entry on how to develop different plugins and integrate them into Ganga

The current credential system is having a complete overhaul in the near future to provide:

- Multiple local and backend credentials handled (not just grid and AFS) ●
- Easy runtime management of required credentials ●
- Credential dependent submission/monitoring ●

This is in beta and should be available by the end of the year

```
# examine startup creds
for c in CredentialStore.getCredentials():
    print c.identity()
    print c.timeleft()

# create a new credential
c = VomsProxy( role = "..", proxy_path = ".." )
c.renew()

# submit job with a particular cred
j = Job()
j.credentials = [ AFSToken(), c ]
j.submit()

# Define default credentials in .gangarc:
DefaultCredentials = { {'AFSToken' : {'Backend':'LCG', 'Application':'Athena'}},
                       {'UID/Type' : {'Backend':'westgrid', 'Application':'Athena'} } }
```

A growing Wiki that covers the basics and some advanced topics can be found at:

<http://ganga.cern.ch>

A growing Wiki that covers the basics and some advanced topics can be found at:

https://www.gridpp.ac.uk/wiki/Guide_to_Ganga

You can always get help at:

[project-ganga<AT>cern.ch](mailto:project-ganga@cern.ch)

OR

[project-ganga-developers<AT>cern.ch](mailto:project-ganga-developers@cern.ch)

Or emailing me: mws@hep.ph.bham.ac.uk