The GridPP UserGuide

T Whyntie[1]

[1]Particle Physics Research Centre, School of Physics and Astronomy, Queen Mary University of London, Mile End Road, London, E1 4NS, United Kingdom

E-mail: t.whyntie@qmul.ac.uk

Abstract. The GridPP Collaboration is a community of particle physicists and computer scientists based in the United Kingdom and at CERN. This document is an offline version of the GridPP UserGuide, written as part of GridPP's New User Engagement Programme, that aims to help new users make use of GridPP resources through the DIRAC, Ganga and CernVM suite of technologies. The online version may be found at: http://www.gridpp.ac.uk/userguide

Contents

0. Introduction

Welcome to the *GridPP UserGuide*. The GridPP Collaboration [1, 2] is a community of particle physicists and computer scientists based in the United Kingdom and at CERN. It supports tens of thousands of CPU cores and petabytes of data storage across the UK which, amongst other things, played a crucial role in the discovery of the Higgs boson [3, 4] at CERN's Large Hadron Collider [5]. The aim of this document is to help new users - like you - join this community and access these resources to make a difference to the world beyond the realm of particle physics.

So, if you have a data-intensive problem that could be solved using large-scale distributed computing, read on!

0.1. Who is this guide for?

This guide is primarily aimed at people from user communities that have not previously engaged with grid (a.k.a. distributed computing) technology. You could be:

- a researcher from a UK institution with a problem that could be solved by the application of thousands of computers running software in parallel over large, structured data sets;
- a student from the UK who would benefit from being able to access computing and data storage resources that your own institution cannot provide (i.e. your school);
- a tech entrepreneur from a start-up or Small-to-Medium Enterprise (SME) who would like to test how your software or app scales to thousands of machines for zero cost and minimal risk.

The *GridPP UserGuide* isn't really aimed at:

- Members of scientific collaborations who already have a grid presence and infrastructure in place (e.g. CMS, ATLAS, SNO++, T2K);
- Users from outside of the UK - you should refer to your own country's National Grid Initiative (NGI) to find out about the best way of getting on the grid,

although it may serve as a useful reference for members of these communities.

Assumed knowledge

While every effort has been made to cover as many bases as possible, some computing knowledge is assumed. You can read more about what you might need to know in the prerequisites section.

0.2. What do I do next?

You should read *Before We Begin* (Section 1) to go over the *Prerequisites* (Section 1.1), the *conventions* (Section 1.2), and how to *get help* from the GridPP community (Section 1.6). If you do not have access to a **Grid User Interface** (UI) (or don't know what that means), you should then look at creating one following the instructions in Appendix A. Then it's simply a case of getting a **grid certificate**, joining a **Virtual Organisation** (VO), and getting on the Grid!

What does this all mean?

Don't worry, we'll explain all of those terms as we go along - mostly using little information boxes like this one. For now, though, you can start here.

1. Before We Begin

- Prerequisites: This section gives a brief run-down of what you'll need - and what you'll need to know - before we can get started on the grid with the *UserGuide*;
- Conventions in this guide: This section introduces some of the conventions used by the *UserGuide*. A guide to the guide, if you will;
- Getting help: One of the great things about the GridPP project is the community of experts who are there support grid users. This section looks at the various ways you can get help if you get stuck, run into problems, or need advice on a grid-related issue.

1.1. Prerequisites

While we want to make the grid accessible to everyone, there are some things you are going to need - and are going to need to know - in order to take full advantage of the resources on offer. Let's go through these now.

- **A valid email address**: This sounds kind of obvious - and who *doesn't* have an email address these days? - but you'll need a valid email address from which you can send and receive emails.

Which email address should I use?

If you can, use your institutional or organisational email account (such as that given to you by your school, university, or company) as this will make life a little easier when it comes to granting you access to grid resources.

- **A GitHub account**: GridPP is an Open Source project. Much of the software used by the GridPP Collaboration is hosted on our GitHub repository - including the *UserGuide* itself. It means we can track developments and issues in a public forum and so maximise collaboration opportunities. You can sign up for a free GitHub account on their website.

Online code repositories

GitHub isn't the only online code repository available. For example, BitBucket also offers a similar git-based versioning system. CERN, too, have their own git system. GitHub is used because they allow unlimited public repositories with an unlimited number of collaborators - which is what GridPP is all about. BitBucket, on the other hand, offer an unlimited number of private repositories for users, but limit the number who can collaborate. Please get in touch if you'd like to know more.

- **Experience with the command line**: The command line allows you to type instructions into your computer in order to get it to do things for you, rather than relying on clicking on icons, buttons, and other graphical elements of a software package. In his guide, Learn Enough Command Line to be Dangerous, Michael Hartl uses a nice analogy with magic. While it is *technically* possible to use the Grid without using the command line (using, for example, a web browser to access specific Grid systems), using the command line is infinitely easier and gives you much, much more flexibility. Hartl's tutorial, is well worth following if you've not used it before (or even if you have!).
- **A text editor**: we'll be writing scripts - series of commands to be executed one after the other - and for this you'll need a text editor of some description. Emacs, Vim, Vi - whatever you feel most comfortable with. Vim, for example, allows you to edit text from the command line.

- **Programming with Python**: Once we start getting fancy with the Grid, we're going to use the Python programming language (via an Application Programming Interface) to do a lot of the work for us. As such, some familiarity with Python will be handy. There are plenty of (free!) online tutorials available that can get you started. We'll provide plenty of examples too, so don't panic!
- **Contact with GridPP**: Before we can let you loose on GridPP's vast computing resources, it'd be nice to know who you are and what you're doing with them. In fact, this is a requirement of the UK Grid policy. You may already be in contact with a GridPP representative at your local institution - if not, feel free to drop us a line.
- **A Scientific Linux 6 command line with CVMFS access**: This will either be provided by your friendly GridPP contact (see above) or via a GridPP CernVM, a Virtual Machine made by CERN that you can run yourself. The CernVM-File System, a.k.a. CernVM-FS or CVMFS, gives you (and any grid node, for that matter) instant access to all sorts of software *without having to install anything*. So it's worth sorting out! You can find instructions for creating a GridPP CernVM in this appendix.

Got all of that? Good. Now let's look at the conventions used by the *GridPP UserGuide*.

1.2. Conventions in this guide

The conventions used in the *GridPP UserGuide* are, by and large, self-explanatory. Here we'll look at a few that might not be.

1.3. The command line

Following Hartl, we'll present command line examples using a Unix-style command line prompt (a dollar sign), as follows:

```
$ echo "Hello, MoEDAL!"
Hello, MoEDAL!
```

i.e. you type what follows the dollar sign, and hopefully see the same (dollar sign-less) output in your terminal.

Comparing the output

Computer systems are always going to vary from machine to machine, so you may not see exactly the same output from a given command. We've tried to eliminate this as much as possible by using the CernVM (see later) but more often than not a combination of common sense and Googling the output should confirm if you're on the right track.

Where possible we'll use bash environment variables to account for differences in working directories. However, if a particularly user-specific input is needed we'll use square brackets to denote parts of the command that require input specific to your circumstances. For example, when setting your working directory environment variable `WORKING_DIR`, we'd write this:

```
$ export WORKING_DIR=[Your working directory.]
$ echo $WORKING_DIR
[The value of $WORKING_DIR, hopefully your working directory.]
```

which would actually be completed using:

```
$ export WORKING_DIR=/home/alovelace/grid-stuff/
$ echo $WORKING_DIR
/home/alovelace/grid-stuff/
```

1.4. Code listings

Generally speaking, we have tried to avoid listing large swathes of code in the *UserGuide* itself - that's what GitHub is for. From time-to-time it may be useful to include a code snippet like the following:

```
#!/usr/bin/env python
print("* This works!")
```

Following Hartl, we will use vertical dots to represent code omitted for the sake of brevity:

```
#!/usr/bin/env python

class GridJob:
    .
    .
    .
    def submit(self, id):
        self.id = id
        .
        .
        .
```

These dots should not be copied into your code. Obvs.

1.4.1. Hints, warnings, and information

The *GridPP UserGuide*, like many instructional handbooks, uses little pop-out boxes to highlight important points throughout the text.

Hint boxes

This is a hint. Hint boxes are used for pointing out things that might be useful while carrying out the task being described (particularly where we have received user feedback on a given step!).

Warning boxes

This is a warning. These are used to flag up potential pitfalls or issues you may need to be aware of to avoid making mistakes or doing Something Bad.

Information boxes

This is a point of information. These boxes will generally present things that may not directly relate to the topic being discussed but are nonetheless interesting.

1.4.2. Checklists

Once you've waded through the waffle associated with a given section, you'll be presented with a **checklist** section that will give you a simple, bullet-pointed list of the things you should be able to do once you've read that section. You should go through these to make sure you have done them and, more importantly, understood them. If not, re-read the section. Alternatively,

you could plough on and try the tests - see below - to see if it makes more sense when you try to actually do something based on what you've just read.

1.5. Testing

All well-written, well-packaged code should come complete with **unit tests**; scripts or bits of code that can be run to test whether one's software is working as expected (especially during development as changes are made and new versions are produced). We can try to emulate this approach by trying to test the success of each of the steps taken while following the instructions presented in the *UserGuide*. At the end of each section you will therefore find a **Testing** page that will present a number of tasks or tests for you to complete to verify that you have followed the *UserGuide*. As a rule you should not proceed to the next section until you have passed all of these tests.

If you're struggling, there are plenty of ways to get help and support. We'll find out more about these in the next section.

1.6. Getting help

There are many ways of getting help and support if you run into problems while working through the *GridPP UserGuide*. If you don't happen to have a GridPP expert in the office down the corridor, you can try the methods described below.

1.6.1. Check the troubleshooting guide

We've added a short troubleshooting guide for problems that users have come across that we know are specific to particular systems, generally raised via the GitHub Issues page. It might be worth checking here first for anything obvious.

1.6.2. Googling the error

We can't possibly account for every error a user might encounter when working through the *UserGuide*, so on encountering a problem your first port of call should be sticking the error message into your Search Engine of Choice.

Errors on the Internet

This is actually a pretty good approach to software development in general. Thanks to vibrant, enthusastic communities like those at StackExchange many common computing gotchas have been documented and solved on the World Wide Web - so it's always worth checking!

1.6.3. Issue tracking via GitHub

The easiest way to report problems, make suggestions, or submit comments about the *UserGuide* is by raising an **issue** on the *GridPP UserGuide* GitHub repository. Simply log in to GitHub, visit the *UserGuide* issues page and click on the New issue button.

Submitting an issue to GitHub

Provide as much information as you can when raising an issue. You can also use the MarkDown format to create hyperlinks and add formatting to your issue.

We'll then have a public record of the issue which we can then aim to solve as soon as we can. It's also possible to link issues to the pull requests that fix them.

Watching GitHub repositories

Don't forget to Watch the repository too. You can do this by going to the repository, signing in with your GitHub account, and clicking on the Watch button at the top-right of the page. You'll then be kept up-to-date with issues and new versions as the UserGuide evolves over time.

1.6.4. Mailing lists

A great way to tap into the expertise represented by the GridPP Collaboration is to join one of the mailing lists in the table below. You'll need a valid email address, but if you've read the prerequisites you know that already.

Table 1: The GridPP support mailing lists.

List	Description	Subscribe
`GRIDPP-USERS`	A list for announcements and discussions aimed at UK Grid users.	JISCMail
`GRIDPP-SUPPORT`	A list for discussion and support aimed at UK Grid users.	JISCMail

Using the mailing lists

These mailing lists are public, so keep it nice people!

1.6.5. Contact us

Finally, if none of the other methods yield results, drop us a line using the details here:

https:///www.gridpp.ac.uk/contact/

2. First Steps: Hello World(s)!

It's something of a tradition to start any new computing activity with an exercise that produces the phrase, "Hello, World!" with whatever you're doing. And it's not a tradition we'll be breaking with now. Distributed computing, however, is all about doing things on a bigger scale - so we'll be saying "Hello" to many worlds all at the same time. We'll do this using the Ganga toolsuite. By the end of this chapter you'll have used Ganga to submit multiple jobs with a single command and check their output.

Jobs

A job is the term we use to describe a task, or set of tasks, we run on our local machine, our cluster's machines, or the grid's Worker Nodes (WNs). We'll come back to these concepts later in the UserGuide.

We won't be using the grid yet - they will run on your local machine - but thanks to Ganga and the other tools used by GridPP making the switch to grid running is pretty straightforward.

Ready? Let's say "Hello!"

2.1. Starting Ganga

Ganga is a Python-based toolkit used for submitting jobs and managing data on the grid. You can read more about it on its CERN page here. The code is all on GitHub, of course. Crucially, Ganga is available via CVMFS so *you don't even have to install it*. On your terminal with CVMFS access, Ganga can be started by simply typing

```
$ source /cvmfs/ganga.cern.ch/runGanga.sh
```

After various welcome messages have been presented, you should see the Ganga command prompt:

```
Ganga In [1]:
```

iPython

If you're familiar with iPython, this prompt style should look very familiar!

Numbered prompts

The number you see in the Ganga (iPython) prompt is the number of the command that you've executed in the terminal. This is great for interactive running, but rubbish for writing user guides. We'll replace this is with an `X` *in what follows.*

Quitting Ganga

To quit Ganga, press `Ctrl-d` *and then type* `y` *or press Enter.*

You will be using Ganga a lot. You may want to create an alias for the Ganga start command in your ~/.bashrc file.

All good so far? Great. Now you're ready to submit your first **job**.

2.2. Submitting a Hello, World! job

Ganga has an iPython-esque command line interface for real-time job management. Things like jobs are modelled using Python objects. So creating and submitting a job is as simple as this:

```
Ganga In [X]: j = Job()
Ganga In [X]: j.submit()
```

You should see an output that looks like something like this:

```
INFO     submitting job X
INFO     job X status changed to "submitting"
INFO     Preparing Executable application.
INFO     Created shared directory: [temporary directory name]
INFO     Preparing subjobs
INFO     submitting job X to Local backend
INFO     job X status changed to "submitted"
```

What's going on here? Well, the `Job()` object has a bunch of default settings that, on instantiation, create a Hello, World! job that submits to the "Local" backend - i.e. the machine you are running on.

Back-ends

The back-end is wherever you want your jobs to run - your local machine, your local computing cluster, or the grid (via GridPP DIRAC). The beauty of Ganga is that you have the same interface for whichever you are using - which makes switching between them a case of tweaking some configuration files.

In the time it has taken to read the above, you should see the following output (press return if not).

```
INFO     job X status changed to "running"
INFO     Job X Running PostProcessor hook
INFO     job X status changed to "completed"
INFO     removing: [temporary directory name]
```

Your job has finished. You can check this - and the status of any other jobs - using the `jobs` command, which produces a neat little summary table of all of your jobs:

```
Ganga In [X]: jobs
Ganga Out [X]:
Registry Slice: jobs (1 object)
```

OK, so your job has run and finished, but where is the famous phrase that signifies success? Well, Ganga manages your job output for you in a series of output directories. More on this later, but you can peek at the output with the following command from Ganga:

```
Ganga In [X]: j.peek('stdout', 'more')
Hello World
```

Reading the output

We've used the **`more`** *program to read the output, but you can specify your own if you like...*

Ta da! You've submitted, run, completed, and checked the output from, your first grid-like job. OK, so it didn't run on the grid this time, but thanks to Ganga the process isn't actually that different.

Local running for testing workflows

In fact the ability to run grid-like jobs locally is very useful for testing your workflow out before unleashing it on the grid...

Finally, to tidy up:

```
Ganga In [X]: j.remove()
INFO     removing job X
```

You can check with the `jobs` command that the job has gone from the list.

Congratulations - you've submitted your first job! But we can do better than that: with distributed computing, the idea is to break your problem into bits and tackle them with multiple jobs: *divide and conquer*. Let's see how easy this is to do with Ganga.

2.3. Submitting the Hello, World(s)! jobs

We're going to use a Python script to generate and submit multiple jobs to the Local backend with Ganga. First, use your favourite editor to create the following script (which we will call `hello_worlds.py`):

```
$ cat hello_worlds.py
worlds = ['Mercury', 'Venus', 'Mars', 'Earth', 'Jupiter', 'Saturn',
```

```
'Uranus', 'Neptune', 'Pluto']

for world in worlds:
    j = Job()
    j.name = "hello_%s" % (world.lower())
    j.application.args = ["Hello, %s!" % (world)]
    j.submit()
```

Two terminals

You may want to have two terminals running in your working directory – one to run Ganga in, and one to write scripts in. This will save having to quit Ganga each time you want to edit a script with a command-line editor (e.g. vim).

There are a few things to note here:

- We have given each job a `name` using the script. This will make life easier later on once the jobs have finished.
- The `executable` used is still the default (`echo`), but now we have supplied varying arguments for the different jobs.

Ganga can then run this script with the `execfile` command. The script uses a `for` loop to create the jobs and submit them:

```
Ganga In [X]: execfile('hello_worlds.py')
[... updates on the job submission ...]
```

All being well, all nine jobs will run and complete. Using `jobs` to find the job ID, you can look at the output as before:

```
Ganga In [X]: jobs(7).peek('stdout', 'more')
Hello, Neptune!
```

2.3.1. Job manipulation tips and tricks

Of course, now you're able to create and submit potentially huge numbers of jobs, you may want to think about how to keep on top of which jobs are which, how to remove jobs, etc. This is where Ganga really comes in to its own. For example:

- **Selecting jobs by name**: remember how we gave each job a name? Well, this allows us to select the jobs we want in one go:

```
Ganga In [X]: my_jobs = jobs.select(name='hello_*')

Ganga In [X]: my_jobs
Ganga Out [6]:
Registry Slice: jobs.select(minid='None', maxid='None', name="None") (9 objects)
```

You can now use the `my_jobs` object to do things to your jobs, such as `submit`, `copy`, `resubmit`, etc.

Tab complete

Use tab complete to see what's possible with the **`my_jobs`** *you've selected.*

- **Removing multiple jobs**: To tidy up all the jobs in one go, use the slice you've created with the `select` command:

```
Ganga In [X]: my_jobs.remove()
INFO     removing job X
[...]
INFO     removing job X+8
```

You can verify this has been successful with the `jobs` command.

So there we go - your first multiple job submission with Ganga. Obviously we are going to need to incorporate more complicated features to adapt your workflow for grid running - using your own executables and software libraries, uploading input data, extracting the output, etc. - but hopefully you can see how we might go about this using Ganga and, ultimately, the Grid.

Now take a look at the following checklist to make sure you've got everything from this chapter nailed. Then we'll look at a more complicated workflow in Section 3.

2.4. Checklist

- I can start Ganga from my command line;
- I can submit a simple "Hello, World!" job using the Ganga default `Job()`;
- I can give a job a `name`;
- I can write a script that creates and submits multiple jobs;
- I can select a group of jobs based on the name I assigned them on creation;
- I can remove multiple jobs with a single command operating on my selection of jobs.

2.5. Testing

- **Running Ganga from the command line**: if you have successfully run Ganga, you should now have the following in your `$HOME` directory:

```
$ cd $HOME
$ ls ~/.gangarc
/home/alovelace/.gangarc
$ ls ~/gangadir
repository  shared  thread_trace.html  workspace
```

- **Looking at the output from local Ganga jobs**: assuming you haven't removed them (you can always re-run them again if you have), you should be able to find the actual output files from your jobs in your `gangadir`. So for user `alovelace`'s job 0, the output can be found in:

  ```
  $ ls $HOME/gangadir/workspace/alovelace/LocalXML/0/output/
  __jobstatus__ stdout stderr __syslog__
  ```

  You should see something similar (and you can look at the output in `stdout` directly if you like!).

3. An Example Workflow: Local Running

Hello, World! jobs are all very well, but we're guessing your workflows are more complicated than a printing out a simple, if polite, greeting. We're now going to demonstrate the capabilities of Ganga and the **CernVM-File System** (CernVM-FS, or CVMFS) for running jobs with input data, remotely-managed software, and output data.

3.1. An aside: what is the CernVM-FS?

The CernVM File System [6] is "*a network file system based on HTTP and optimized to deliver software in a fast, scalable, and reliable way*". It was developed to solve the problem of installing and maintaining the software used by all of the different particle physics communities involved with work at CERN. Simply put, systems with the CernVM-FS installed have instant access to a given community's software repositories via the command line. This means it can be used:

- by community members working on university computing clusters outside of CERN;
- by Worker Nodes (WN) anywhere on the grid where the repository is supported;
- by CernVM Virtual Machines.

You've already used CVMFS to run Ganga itself. But it can be used to host your own software that will run anywhere on the Grid (or, indeed, anywhere with access to CVMFS).

The example workflow we'll use comes from the CERN@school research programme. We'll take some raw particle detector data in ASCII format, turn it into some pretty images of detected particles using the Python `matplotlib` software, and retrieve the frame information and images as output.

3.2. The workflow itself

The workflow here is pretty straightforward:

- **Input**: raw detector data from a Timepix hybrid silicon pixel detector. These ASCII text files represent the data (and detector settings) recorded by a Timepix detector during a background measurement reading. The dataset we use here is hosted on the web at FigShare. We will download a `zip` file containing the data and upload it with our job.
- **Processing**: the data is processed with a Python script in the CERN@school CVMFS repository called `process-frames.py`. This script in turn uses Python modules (both custom and standard) that are also hosted on and sourced from CVMFS.
- **Output**: `process-frames.py` produces a log file, a JSON containing information about the processed data frames, and a directory of images representing the particles detected in each frame.

We will create a workflow in Ganga that uploads the input data we've downloaded from FigShare, process it on our local machine, compress the images into a single `tar` archive, and retrieve the log file, JSON file, and `tar` archive as output.

3.3. Getting the input data

In your working directory, which we will associate with the environment variable `$WORKINGDIR`, download the dataset as follows:

```
$ export WORKINGDIR=$PWD
$ cd $WORKINGDIR
$ wget http://files.figshare.com/2600426/CERNatschool_backgroundrad_dataset.zip
$ unzip CERNatschool_backgroundrad_dataset.zip
$ rm CERNatschool_backgroundrad_dataset.zip
$ ls
B06-W0212  E09-W0092  README.md
```

Data and jobs

We could actually get our grid job to do this as part of the job. Or we could tell our job to use input data that is already hosted on a grid storage element. For simplicity, though, we will upload the data we have just downloaded to our working directory with our job.

We'll see how this is uploaded with the job below.

3.4. Writing the executable

With our *Hello, World!* job(s), we used the built-in executable `echo` to print a simple string. This workflow will use the shell script below, `run.sh`, that contains the commands we want our job to execute.

```
$ vim run.sh
$ chmod a+x run.sh
$ cat run.sh
#!/bin/bash
#
# Add the Python packages from the CERN@school CVMFS
# repository to the PYTHONPATH environment variable.
export PYTHONPATH=/cvmfs/cernatschool.egi.eu/lib/python2.6/site-packages/: \\
/cvmfs/cernatschool.egi.eu/lib64/python2.6/site-packages/:$PYTHONPATH
#
# Add CERN@school libraries to the LD_LIBRARY_PATH.
```

```
export LD_LIBRARY_PATH=/cvmfs/cernatschool.egi.eu/lib/: \\
/cvmfs/cernatschool.egi.eu/lib64/: \\
/cvmfs/cernatschool.egi.eu/lib64/atlas:$LD_LIBRARY_PATH
#
# Add CERN@school libraries to the PATH.
export PATH=/cvmfs/cernatschool.egi.eu/lib64/:/cvmfs/cernatschool.egi.eu/lib/: \\
/cvmfs/cernatschool.egi.eu/lib64/atlas:$PATH
#
# Unzip the uploaded input data.
unzip CERNatschool_backgroundrad_dataset.zip
#
# Run the CVMFS-hosted Python script on the data.
python /cvmfs/cernatschool.egi.eu/code/particle-rate-plotter/process-frames.py \\
$1 ./.
#
# Compress the images ready for retrieval.
tar -cvf output_images.tar PNG/
```

You should make `run.sh` in your `$WORKINGDIR` by copying and pasting the above into your favourite text editor.

Executable scripts

Don't forget to make the script executable with the **`chmod`** *command.*

3.5. Preparing the job

We'll use Ganga's `execfile` functionality to create, configure, and submit our job with a short Python script called `local_job.py`, listed with explanatory comments below:

```
$ vim local_job.py # Copy and paste away!
$ cat local_job.py
## The Ganga job.
j = Job()

# Name the job.
j.name = "CERN@school_local_01"

# Tell Ganga it's running an executable: run.sh
j.application = Executable()
j.application.exe = File('run.sh')
```

```
# run.sh takes one argument - the dataset directory.
j.application.args = ['B06-W0212/2014-04-02-150255/']

# Specifiy which local files to upload with the job.
j.inputfiles = [ LocalFile('CERNatschool_backgroundrad_dataset.zip') ]

# Specify which files should be downloaded as output from the job.
j.outputfiles = [ LocalFile('frames.json'), \\
LocalFile('log_process-frames.log'), LocalFile('output_images.tar') ]
j.submit()
```

You can then run this within Ganga using the `execfile` command as before:

```
Ganga In [X]: execfile('local_job.py')
[... job output messages ...]
```

Location location location

Make sure you are running Ganga from **`$WORKINGDIR`** *so that it can find* **`local_job.py`** .

You can monitor the status of the job as before with the `jobs` command. As the job is actually doing some work now, you may be able to see the job assume the `running` status.

3.6. Getting the job output

Once the job has finished, you can view the output of the text files as before with the `peek` command:

```
Ganga In [X]: j=jobs(X)
Ganga In [X]: j.peek('log_process-frames.log')
INFO:root: * Creating directory '././PNG'...
INFO:root:
INFO:root:* Found 60 datafiles.
```

To view the images you've created, you'll need to find where they have been downloaded to on your local machine. If the job ID was "1", you can do this with:

```
Ganga In [X]: j = jobs(1)
Ganga In [X]: j.outputdir
Ganga Out [X]: '/home/alovelace/gangadir/workspace/alovelace/LocalXML/1/output/'
```

```
$ cd $WORKINGDIR
$ cp /home/alovelace/gangadir/workspace/alovelace/LocalXML/1/output/output_images.tar
output_images.tar
$ tar -xvf output_images.tar
```

```
PNG/
PNG/E09-W0092_2014-04-02-143123.png
[...]
PNG/E09-W0092_2014-04-02-142120.png
```

You can view these images with the following command, which will bring up the Eye Of Gnome image viewer (assuming you have it installed on your local machine). Use the arrow keys to move through the frames of data.

```
$ eog PNG/ &
```

So there we have it - we've run a (rather simple) workflow on our local machine using Ganga. But here's the thing: to move this workflow (and, indeed, most workflows) to the grid, we only need to do three things:

1. Configure the job to use the GridPP DIRAC backend;
2. Put our input data on the grid and configure our job to use this;
3. Configure the job to write the output data to the grid.

2) and 3) take a bit more work, but are optional (it really depends on your input and output data as to what actually *needs* to go on the grid. But once you're setup for grid running, 1) only takes a single line in your job configuration script:

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

Ganga does the rest. Does that sound good? Let's get you set up on the grid then.

3.7. Checklist

- I can create, configure, and submit a local job that uses local-sourced data as input and software hosted on CVMFS using Ganga;
- I can view the output logs from my local job with the `peek` command;
- I can find and retrieve the output of my local job using the `outputdir` command in Ganga.

3.8. Testing

- **Successful running of the CERN@school example job**: Once you have run and retrieved the images from the example job, the first frame image should look something like that shown in Figure 1.

For reference, that's a beta particle in the bottom left corner, and five gammas in the rest of the frame! You can also find the source code on the CERN@school GitHub page.

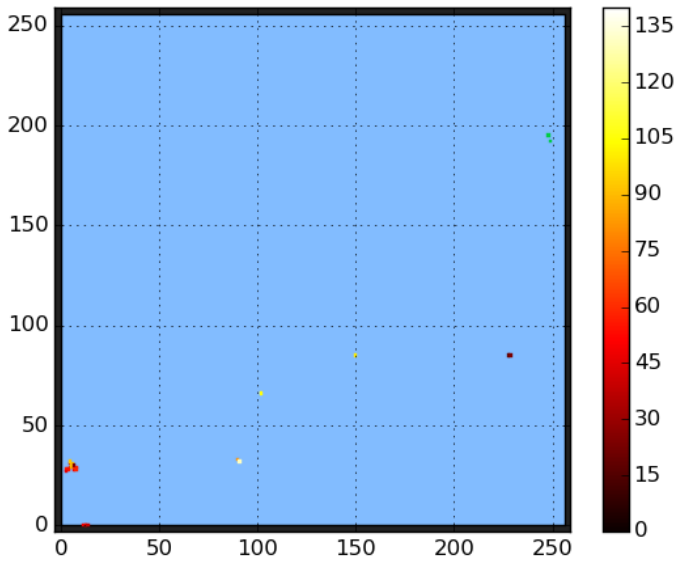

Figure 1: The example output from the CERN@school job.

4. Getting on the Grid

We have run local jobs so far. Now it's time to get on the grid - after which, we will be able to configure Ganga to submit our jobs to the GridPP DIRAC system and so access all of the computing and data resources GridPP has to offer. The following sections will cover:

- Getting a grid certificate (Section 4.1);
- Joining a Virtual Organisation (Section 4.2);
- Logging on to GridPP DIRAC.

We'll find out more about what each of these steps mean and entail as we go.

Thinking ahead

Some of these steps require interaction with a human being. For example, to get a grid certificate you have to visit your local Registration Authority in person with photographic ID. Please bear this in mind before you begin - it is not an automated process and may take a little time. It will be worth it though!

Let's start by getting you a Grid certificate, shall we?

4.1. Your Grid certificate

Your grid certificate is your passport to the grid. It will give you access to the vast array of computational resources that GridPP (and the wLCG) has to offer. As such, getting a grid certificate is an understandably non-trivial, multi-step process. For example, you will need to identify yourself to your local Registration Authority (RA) so that the grid knows who you are.

In concrete terms, a grid certificate is a .p12 file (i.e. a pkcs12 web browser certificate file) that you will later convert into a *user key* file and a *user certificate* file. Encoded according to the X.509 standard, these are used by the grid to confirm who are you are and so give you access to grid resources.

4.1.1. Requesting a Grid certificate

Grid certificates in the UK are managed by the UK e-Science Certificate Authority[†]. To start the process, you need to choose a web browser that you will have consistent access to. We recommend Firefox as this process has been tested and confirmed to work with Firefox on most Operating Systems. This is because you need to use the same system for both requesting your certificate and retrieving it when it is ready.

[†] See http://ngs.ac.uk/ukca

Temporary logins

Do not use a temporary login anywhere when requesting a grid certificate.

Using your browser of choice visit the CA portal[†] and select the *Request New User Certificate* option. This almost goes without saying, but make sure you supply a **valid email address** which you can access. You will also be asked to do things like supply a PIN and passwords that you will need later on, so **make sure you write everything down**!

Registration Authroties

You will need to select a Registration Authority (RA) as part of this process. If your institution does not have its own RA, select the nearest on the drop-down menu. You will need to visit the RA in person with some photographic identification, so don't pick one that is too far away! If no contact information is listed for a given RA, they will almost certainly be retrievable using a Search Engine of Your Choice or via their department's webpage. They will be delighted to hear from you!

Further instructions will then be emailed to you at the email address you have supplied during the registration process. Once that has happened you should get a further email from someone at the RA asking you to visit them in person to complete the validation process.

Who are you?

You may also be asked to supply a letter of recommendation (or, rather, an email from a suitable authority) explaining why you need to use the grid and with whom you will be working. If you are unsure about who to ask for this, please contact us[†] and we should be able to help you out.

[†] See https://www.gridpp.ac.uk/contact

4.1.2. Installing your Grid certificate in your web browser

Assuming all has gone to plan, you should receive a confirmation email with a link that will let you download your grid certificate file and install it in your browser. You will now be able to export and backup your grid certificate using your browser's certificate management functionality. This process will vary from browser to browser and from OS to OS, so consult the UK CA documentation if in doubt.

Congratulations - now you can be identified on the grid, you're ready to join a **Virtual Organisation**.

[†] See https://portal.ca.grid-support.ac.uk/caportal/

4.2. Joining a Virtual Organisation

Your Grid certificate identifies you to the grid as an individual user, but it's not enough on its own to allow you to use grid resources; you also need to join a **Virtual Organisation** (VO). These are essentially just user groups - typically one per experiment - and individual Resource Centres (RCs) can choose to support work by users of a particular VO. Most RCs support the four VOs associated with the Large Hadron Collider (LHC) experiments. The sign-up procedure varies from VO to VO. UK-based VOs typically require a manual approval step, while LHC VOs require an active CERN account. If you are already part of an experiment that is represented by a VO, they should provide you with any specific instructions you need to join.

If you're interesting in using the grid but are not (yet) working as part of a user community already represented by a VO, worry not. GridPP have created a catch-all VO - `gridpp` - and four Regional Virtual Organisations (RVOs) corresponding to the four Tier 2s that can be joined to test out what the grid has to offer. Once you have used these "incubator" VOs to see if the Grid meets your needs, you can then think about creating your own Virtual Organisation to represent your user community.

Is there a VO for you already?

Your user community may already have a VO associated with it. Check the GridPP wiki page of supported VOs to see if you can join that to speed things up.

4.2.1. Joining an incubator VO

Just browsing

Some users have reported that the VOMS registration described below fails using the Safari web browser. We have tried and tested the process using Mozilla Firefox.

Trusting the VOMS servers

Please ignore any "untrusted connection" warnings when accessing the VOMS server pages. GridPP is aware that the VOMS server uses unsigned certificates, but this situation is unlikely to be resolved any time soon.

4.2.2. Joining the GridPP VO

To join the `gridpp` VO, visit this page using a browser that has your grid certificate installed and follow the instructions.

4.2.3. Joining a Regional VO

Likewise, you can join one of the four regional VOs:

- `vo.londongrid.ac.uk`
- `vo.northgrid.ac.uk`
- `vo.scotgrid.ac.uk`
- `vo.southgrid.ac.uk`

Confirming VO your membership

Your VO membership request needs to be confirmed manually by one of the VO administrators, so please wait for the membership confirmation email to arrive before proceeding. You may wish to keep an eye on your junk folder(s) too.

Once you have joined a VO, congratulations - you are ready to start using the Grid!

4.3. Logging on with GridPP DIRAC

There are many ways of accessing and using Grid resources. Larger organisations - such as the four LHC experiments - have developed their own frameworks, architectures and mechanisms to enable their members to run jobs and access experimental data.

One such framework – **DIRAC** [7] – is used by LHCb [8], but also many other Grid projects, to manage grid jobs and storage. You can read more about DIRAC (Distributed Infrastructure with Remote Agent Control) on their website[†] or in [7], but for our purposes all you need to know for now is that DIRAC provides a way for you to access grid resources without worrying too much about what's going on behind the scenes.

4.3.1. The GridPP DIRAC instance

The Imperial College London GridPP Resource Centre (RC) hosts an instance of DIRAC on behalf of GridPP [9, 10]. The GridPP DIRAC instance is is capable of serving multiple VOs, providing grid job and data management capabilities for smaller, non-LHC user communities wishing to make use of GridPP resources. As a new user, you are automatically registered with the GridPP DIRAC instance and the Virtual Organisations you have joined. You can then test out various bits of grid functionality to determine if grid computing will meet your needs and the needs of your users.

[†] See http://diracgrid.org

You can interact with the Grid via the GridPP DIRAC web portal at:

https://dirac.gridpp.ac.uk

When you access the portal, you should see yourself listed as a **Visitor** in drop-down menu in the bottom-right corner of the browser. Once you have joined one or more DIRAC-supported Virtual Organisations, you will be able to select which VO you use DIRAC as using this drop-down menu.

The GridPP DIRAC mailing list

You should also join the GridPP DIRAC mailing list to keep informed of the latest developments and receive notices of any downtime. You can join the mailing list here.

So you've accessed the GridPP DIRAC web portal. Congratulations! However, as discussed, we'll be using GridPP DIRAC via the Ganga interface. In order to do that, you'll need to install your Grid certificate on your local machine - and the instructions for doing this are in the next section.

4.4. Preparing your Grid certificate

Ganga will assume that your grid certificate is in a certain location and in a certain format in order to use it. Your grid certificate therefore needs to be moved and prepared accordingly - which you can do by following the instructions below.

4.4.1. Moving your Grid certificate to your UI

The first thing to do is move your Grid certificate (the one you got after following the instructions in Section 4.1) to the `~/.globus/` directory in your home folder.

```
$ cd ~
$ pwd
[Your home directory.]
$ mkdir .globus
$ cp [The location of your certificate file.]/[Your certificate filename].p12 ./.globus/.
```

Certificates on CernVMs

If you are using a CernVM and have moved your personal Grid certificate file to it, you should change the password of the gridpp account so that no-one else can use it. This can be done in the standard UNIX way with the passwd command.

4.4.2. Converting your Grid certificate

In order to use your Grid certificate, you need to convert them into separate certificate and key files. Don't worry, this straightforward enough to do with the following commands:

```
$ cd ~/.globus
$ openssl pkcs12 -in [Your certificate filename.].p12 -clcerts -nokeys -out usercert.pem
Enter Import Password:
MAC verified OK
$ openssl pkcs12 -in [Your certificate filename.].p12 -nocerts -out userkey.pem
Enter Import Password:
MAC verified OK
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
```

You will then need to change the file permission settings on the two newly-generated files:

```
$ chmod 400 userkey.pem
$ chmod 600 usercert.pem
```

And that's it! You're now ready to use GridPP DIRAC with Ganga. It may have seemed like a lot of work, but hopefully the next section will demonstrate it will all have been worth it.

4.5. Checklist

4.5.1. Your Grid certificate

- I have requested a grid certificate from the UK Certificate Authority (UKCA).
- I know where my nearest Registration Authority (RA) is.
- I have visited my nearest RA and confirmed my identity.
- I have downloaded my grid certificate .p12 file and installed it in my browser.
- I have backed up my grid certificate .p12 file in a secure location.

4.5.2. Joining a Virtual Organisation – Checklist

- I have submitted a request to join a Virtual Organisation (VO).
- My request has been approved by a VO manager and I have received email confirmation of the approval.
- I have followed all of the instructions in the confirmation email.

4.5.3. First steps with GridPP DIRAC

- I have accessed the GridPP DIRAC web portal with my Grid certificate-enabled browser.
- I am recognised by the GridPP DIRAC web portal as a *Visitor*.

- I have joined the GridPP DIRAC mailing list.

4.6. Testing

4.6.1. Your Grid certificate

- **Viewing your certificate details**: visit this website with the broswer in which your certificate is installed. You should see your grid certificate details displayed.

4.6.2. Joining a Virtual Organisation

- **Joining the GridPP VO**: Click here. If your request has been approved and confirmed, you should be listed as a VO member.
- **Joining a Regional VO**: Click on `vo.londongrid.ac.uk`, `vo.northgrid.ac.uk`, `vo.scotgrid.ac.uk`, or `vo.southgrid.ac.uk` (depending on which VO you have attempted to join). If your request has been approved and confirmed, you should be listed as a VO member.

4.6.3. First steps with DIRAC

- **Accessing the GridPP DIRAC server**: access https://dirac.gridpp.ac.uk with your browser. If your grid certificate has been successfully installed in your browser, you should be asked to identify yourself with the certificate in question. You will then see the GridPP DIRAC server homepage. Check the bottom right-hand corner - if you can only see *Visitor* and **not** your username and DN, something has gone wrong and you are not (or rather, your certificate is not) not registered with the GridPP DIRAC server.
- **Joining the GridPP DIRAC mailing list**: once you have *subscribed* and have *been approved*, you should be able to view the subscribers list from the list homepage and confirm that you are indeed on it.

5. Moving Your Workflow to the Grid

Now that you have your Grid certificate, and it is installed in your browser and in your `~/.globus` directory, you're ready to try submitting a job to the Grid with DIRAC.

5.1. Activate DIRAC!

Thanks to CVMFS and the Ganga team, activating the GridPP DIRAC functionality is easy:

```
$ source /cvmfs/ganga.cern.ch/dirac_ui/bashrc
```

You should now be able to tab-complete `dirac-` to see all of the DIRAC commands that are available.

DIRAC on CVMFS

The inclusion of the pre-configured DIRAC UI in the Ganga CVMFS repository means that you no longer need to install your own DIRAC UI. Which makes life a lot easier...

5.1.1. Generate a Grid proxy

With DIRAC activated via CVMFS, you should now be able generate a DIRAC-specific proxy to be used as a member of a DIRAC-enabled VO. Your proxy is a file that identifies you on the Grid, letting the system know that you're good for using its resources. If, for example, you were a member of the `gridpp` catch-all VO you would use the following commands to generate a proxy as a `gridpp_user`:

```
$ dirac-proxy-init -g gridpp_user -M
Generating proxy...
Enter Certificate password:
Added VOMS attribute /gridpp
Uploading proxy for gridpp_user...
Proxy generated:
subject      : /C=UK/O=eScience/OU=QueenMaryLondon/L=Computing/CN=ada lovelace/CN=proxy
issuer       : /C=UK/O=eScience/OU=QueenMaryLondon/L=Computing/CN=ada lovelace
identity     : /C=UK/O=eScience/OU=QueenMaryLondon/L=Computing/CN=ada lovelace
timeleft     : 23:53:59
DIRAC group  : gridpp_user
path         : /tmp/x509up_u501
username     : ada.lovelace
properties   : NormalUser
VOMS         : True
VOMS fqan    : ['/gridpp']
```

If you get output that looks like the above, then it has all worked.

Proxy status

You can check the status of your DIRAC proxy at any time with the dirac-proxy-info command.

5.1.2. Configuring Ganga to use the DIRAC backend

There's just one more step required before you can enjoy Grid running with Ganga - you need to add the following to your `~/.gangarc` configuration file:

```
[Configuration]
RUNTIME_PATH = GangaDirac

[LCG]
GLITE_SETUP = /cvmfs/ganga.cern.ch/dirac_ui/bashrc

[DIRAC]
DiracEnvSource = /cvmfs/ganga.cern.ch/dirac_ui/bashrc

[defaults_DiracProxy]
group = <dirac user group>

[defaults_DiracFile]
defaultSE = <your SE of choice>
```

where `<dirac user group>` should be replaced by your default VO (e.g. `gridpp_user`) and `<your SE of choice>` should be replaced by a suitable Storage Element, e.g. `UKI-LT2-QMUL2-disk`.

Finding an SE

You can find a list of Storage Elements names by using the dirac-dms-show-se-status command from the command line.

You can then re-start Ganga; it will now be ready to connect to the DIRAC backend.

5.2. Run your job on the Grid

Now you're ready to run your job on the Grid. First, make a copy of the `local_job.py` script:

```
$ cd $WORKINGDIR
$ cp local_job.py dirac_job.py
$ vim dirac_job.py
```

All you need to do is add the line `j.backend=Dirac()` before submitting. That's it. That's all there is to it.

```
$ cat dirac_job.py
j = Job()
j.name = "CERN@school_dirac_01"
j.application = Executable()
j.application.exe = File('run.sh')
j.application.args = ['B06-W0212/2014-04-02-150255/']
j.inputfiles = [ LocalFile('CERNatschool_backgroundrad_dataset.zip') ]
j.outputfiles = [ LocalFile('frames.json'), LocalFile('log_process-frames.log'),
LocalFile('output_images.tar') ]
j.backend = Dirac()
j.submit()
```

(*Oh, you may want to change the job name too.*)

If all has gone to plan, not only will you now be able to monitor your job via your local Ganga instance (i.e. with the `jobs` command), you can see it on the GridPP DIRAC web portal. Select *Jobs* from the top-left menu below the URL bar, then *Job Monitor*.

Submission time

Job submission to the Grid is not an instant process - a bit annoying when you're submitting one or two test jobs (which is why local testing wit Ganga is great!), but not such an issue with thousands of jobs. You may wish to make a cup of tea, or do a bit of washing up.

Once the job shows up as green in the web portal, it's completed and you can retrieve the output exactly as before.

So there you go - your first *bona fida* grid job. Note that:

- Once your Grid stuff/DIRAC configuration was done, all it took to switch to Grid running was a single line of code.
- Thanks to CVMFS, you didn't have to do anything extra to deploy your software to the Grid;
- You didn't need to care about where the job actually ran - Ganga and DIRAC sorted that all for you.

There's only one more thing to look at now before we've covered all of the Grid-bases - getting data on and off the Grid. We'll look at that in the next section.

5.3. Checklist

- I can activate DIRAC by sourcing the DIRAC `bashrc` script from CVMFS;
- I can generate a Grid proxy using the `dirac-proxy-init` command;
- I can submit the CERN@school example workflow job to the Grid using Ganga;
- I can monitor my Grid job(s) using the GridPP DIRAC web portal.

5.4. Testing

- **Sourcing the DIRAC environment**: You can test the DIRAC environment variables have been set using the `echo` command. For example:

  ```
  $ echo $DIRAC
  /cvmfs/ganga.cern.ch/dirac_ui/
  ```

  If the DIRAC home directory on CVMFS is not listed, the environment variables have not been set correctly.

- **Generating a DIRAC proxy**: You can test if your proxy generation has been successful by using the `dirac-proxy-info` command.

- **Successful running of the CERN@school example job**: As with the locally-run example, once you have run and retrieved the images from the example job the first frame image should look something like that seen in Figure 1.

6. Putting Data on the Grid

We've now moved the local example workflow to the Grid. However, we've still only used data that's been present on our local system, and we've manually retrieved the output to our local system. To harness the full power of the Grid, we'll need to put data on it. We'll use tools provided with DIRAC to do this, namely:

- The DIRAC File Catalog Command Line Interface (DFC CLI);
- The DIRAC command line tools;
- Some first steps with the DFC's metadata functionality.

First, though, let's look at some basic concepts in grid-based data management.

6.1. Storage Elements, File Catalogs, and Replicas

The first thing to wrap one's head around with distributed computing is the notion that you don't really need to care about where your data is stored. You may well be used to this concept if you've dealt with cloud-based storage services such as Dropbox, Google Drive, or even Amazon S3 storage. Your files are on one or more servers *somewhere*, and all that you need to know are the file names and the directories that they're in to access them later.

It's the same with the grid. You upload your files to a grid **Storage Element** (SE) and label them with a **Logical File Name** (LFN) that gets registered in something called a **File Catalog**. If you make copies of a particular file - a **replica** - on one or more additional SEs, the locations of these replicas are recorded in the File Catalog too.

Storage Elements and replicas

With most cloud-based storage services, you won't even really care about the Storage Elements (or their non-grid equivalents, whatever the they happen to be called) and file replicas. However, when considering running grid jobs at a particular grid site, the location of your replicas can matter (you'll want to make sure your data is available at sites that will run jobs for your Virtual Organisation). We'll come back to all of these concepts - and provide concrete examples - later.

The GridPP DIRAC system provides a suite of tools to help you manage all of this. If you're familiar with UNIX-based file systems you should find it all pretty straightforward. We'll start with the DIRAC File Catalog Command Line Interface.

6.2. The DFC Command Line Interface

The DIRAC File Catalog (DFC) Command Line Interface (CLI), a.k.a. the **DFC CLI**, provides a way of interacting with DIRAC's File Catalog via - you guessed it - the command line. The DFC CLI lets you manually upload and download files to Storage Elements (SEs), browse the DFC associated with your Virtual Organisation (VO), create and remove directories in the DFC, and manage the replicas associated with each entry in the DFC.

Using the DFC CLI

The DFC CLI is great for small-scale tasks such as creating and tweaking test data sets, but ultimately we will want to use scripts to help coordinate large-scale upload operations and managing metadata (i.e. data about the data).

6.2.1. Getting started with the DFC CLI

Accessing the DFC CLI

The DFC CLI is accessed via a DIRAC command, so we'll need to source our DIRAC environment and generate a DIRAC proxy.

```
$ source /cvmfs/ganga.cern.ch/dirac_ui/bashrc
$ dirac-proxy-init -g gridpp_user -M
Generating proxy...
Enter Certificate password: # Enter your grid certificate password...
.
. [Proxy information-based output.]
.
```

Which VO?

If you wish to use a different VO, replace gridpp with the name of the VO in the commands in this section.

The DFC CLI is then started with the following DIRAC command:

```
$ dirac-dms-filecatalog-cli
Starting FileCatalog client

File Catalog Client $Revision: 1.17 $Date:

FC:/>
```

DIRAC command groupings

We'll come back to the DIRAC command line tools in the next section, but the **dirac-dms-** *at the start of the command refers to the DIRAC Data Management System tools. All DIRAC commands are grouped in this way which, combined with tab completion, can be very handy for finding the command you're looking for!*

The FC:/> at the command prompt tells you that you're in the DFC CLI. You can now explore the DFC using commands that are very similar to those used with a typical UNIX file system. Let's do this now.

6.2.2. Finding your user space in the DFC

Let's start by listing the root directories in the DFC, which will give us a list of the Virtual Organisations supported by GridPP DIRAC:

```
FC:/> ls
cernatschool.org
gridpp
vo.londongrid.ac.uk
vo.northgrid.ac.uk
vo.scotgrid.ac.uk
vo.southgrid.ac.uk
```

We're using GridPP DIRAC as a member of gridpp VO, so let's move into that directory.

```
FC:/> cd gridpp/user
```

If one hasn't been created for you already, you can create your own user space on the VO's File Catalog like so:

```
FC:/gridpp/user> cd a
FC:/gridpp/user/a> mkdir ada.lovelace
FC:/gridpp/user/a> chmod 755 ada.lovelace
FC:/gridpp/user/a> ls -la
drwxr-xr-x 0 ada.lovelace gridpp_user 0 2015-12-16 10:24:54 ada.lovelace
FC:/gridpp/user/a> exit
```

Your DIRAC username

If you don't know your DIRAC username (which should be used as your user directory), exit the DFC CLI and use the dirac-proxy-info command.

Listing files

Using the `-la` *option with the* `ls` *command works just as it does with the normal command line, allowing you to see file owners, groups (VOs), permissions, etc.*

File permissions

Don't forget to change the file permissions on your files so that other users can't modify them.

You've now got your own space on the GridPP DFC. Let's put some files in it.

6.2.3. Uploading files

Firstly, we'll need a file to upload. Any file will do, but to keep things simple let's create one in a temporary directory:

```
$ cd ~
$ mkdir tmp; cd tmp
$ vim TEST.md # Or whichever editor you use...
$ cat TEST.md
#Hello Grid!
This is a test **MarkDown file**.
```

Next we'll need to know which **Storage Elements** are available to us.

Storage Elements

Storage Elements "are physical sites where data are stored and accessed, for example, physical file systems, disk caches or hierarchical mass storage systems. Storage Elements manage storage and enforce authorization policies on who is allowed to create, delete and access physical files. They enforce local as well as Virtual Organization policies for the use of storage resources. They guarantee that physical names for data objects are valid and unique on the storage device(s), and they provide data access. A storage element is an interface for grid jobs and grid users to access underlying storage through the Storage Resource Management protocol (SRM), the Globus Grid FTP protocol, and possibly other interfaces as well."
Credit: Open Science Grid (2012)

We can list the available SEs with the following DIRAC command:

```
$ dirac-dms-show-se-status
SE                          ReadAccess WriteAccess RemoveAccess CheckAccess
=============================================================================
```

```
[... more disks ...]
UKI-LT2-QMUL2-disk            Active     Active      Unknown      Unknown
[... more disks ...]
UKI-NORTHGRID-LIV-HEP-disk    Active     Active      Unknown      Unknown
[... more disks ...]
```

While we don't need to know the details of where and how our data will be stored on an SE, we do need to know its name. We'll use the `UKI-LT2-QMUL2-disk` SE for now. We add the file to the DFC as follows using the add command, which takes the following arguments:

`add <LFN> <Local file name> <SE name>`

where:

- `<LFN>` is the **Logical File Name** (LFN) of the file in the DFC. This can either be relative to your current position in the DFC (which can be found with the `pwd` command in the DFC CLI), or made absolute by preceeding the name with a slash `/`;
- `<Local file name>` should be the name of the local file you want to upload. Again, this can be relative to wherever you were on your local system when you started the DFC CLI, or the absolute path to the file on your local system;
- `<SE name>` is the name of the SE as retrived from the `dirac-dms-show-se-status` command.

Let's add our file to the grid now.

```
$ dirac-dms-filecatalog-cli
Starting FileCatalog client

File Catalog Client $Revision: 1.17 $Date:

FC:/> cd /gridpp/user/a/ada.lovelace
FC:/gridpp/user/a/ada.lovelace> mkdir tmp
FC:/gridpp/user/a/ada.lovelace> cd tmp
FC:/gridpp/user/a/ada.lovelace> add TEST.md TEST.md UKI-LT2-QMUL2-disk

File /gridpp/user/a/ada.lovelace/tmp/TEST.md successfully uploaded...
FC:/gridpp/user/a/ada.lovelace/tmp>ls -la
-rwxrwxr-x 1 ada.lovelace gridpp_user 47 2015-12-16 11:47:28 TEST.md
```

And there we go! Your first file has been uploaded to a Storage Element on the grid. Have a biscuit. You've earned it.

6.2.4. Replicating files

Part of the joy of using the grid is being able to distribute computational tasks to different sites. However, if you want to look at the same data with a different task at different sites in an efficient manner, ideally you'd need copies of that data at those sites. This strategy also

makes sense from a backup/redundancy perspective. We can achieve this on the grid by using *replicas*.

Replicas

A replica is a copy of a given file that is located on a different Storage Element (SE). The file is identified by its Logical File Name (LFN) in the DIRAC File Catalog (DFC). Associated with each LFN entry is a list of SEs where replicas of the file can be found.

To list the locations of replicas for a given file catalog entry, we use the `replicas` command in the DFC CLI:

```
replicas <LFN>
```

so continuing with our example:

```
FC:/gridpp/user/a/ada.lovelace/tmp>replicas TEST.md
lfn: /gridpp/user/a/ada.lovelace/tmp/TEST.md
```

We replicate files with the `replicate` command:

```
replicate <LFN> <SE name>
```

Let's replicate our test file to the Liverpool disk and check that the replica list has been updated:

```
FC:/gridpp/user/a/ada.lovelace/tmp>replicate TEST.md UKI-NORTHGRID-LIV-HEP-disk
```

Replicas can be removed with the `rmreplica` command:

```
rmreplica <LFN> <SE name>
```

Let's remove the Liverpool disk replica:

```
FC:/gridpp/user/a/ada.lovelace/tmp>rmreplica TEST.md UKI-NORTHGRID-LIV-HEP-disk
lfn: /gridpp/user/a/ada.lovelace/tmp/TEST.md
Replica at UKI-NORTHGRID-LIV-HEP-disk moved to Trash Bin
```

Finally, we can remove a file completely using the (somewhat familiar) `rm` command:

```
rm <LFN>
```

Let's tidy up our test file:

```
FC:/gridpp/user/a/ada.lovelace/tmp>rm TEST.md
lfn: /gridpp/user/a/ada.lovelace/tmp/TEST.md
File /gridpp/user/a/ada.lovelace/tmp/TEST.md removed from the catalog
```

6.2.5. Downloading files

Finally, we can download files using the DFC CLI with the `get` command:

```
get <LFN> [<local directory>]
```

Note that the local directory argument is optional. Let's download a test file from the gridpp examples directory now:

```
FC:/> get /gridpp/userguide/WELCOME.md ./.
FC:/> exit
$ cat WELCOME.md
#Welcome to GridPP!

It looks like your download has worked. Congratulations!
$ rm WELCOME.md
```

As we said earlier, the DFC CLI is only useful for small-scale operations. On our way to scaling up, we can look at starting to automate our workflows using scripts. In the next section we'll look at how the DIRAC command line tools can help with this.

6.3. The DIRAC Command Line Tools

So you've mastered the DFC Command Line Interface. Great stuff. What you'll have probably noticed is that, while it's great for small-scale operations, it's not ideal for doing things with lots of files on any sort of scale. We will therefore want to take a look at the **DIRAC command line tools** for data management.

The command line tools

All of the DIRAC command line tools start with **`dirac-`** *. The data management tools start with* **`dirac-dms-`** *, as in Data Management System. Press the tab key after typing* `dirac-dms-` *to see all of the available commands.*

Why are these commands useful? Well, it means you can use *scripting* to automate large-scale tasks involving many files. There are many ways to *script* the DIRAC (or indeed any command line) commands. You've probably got your own preferred method that reflects your coding background. For the purposes of the UserGuide, we'll use simple bits of **Python** code (along with Python-based file management libraries) to generate some simple `bash` scripts that can then be run to perform the DIRAC operations we want to perform.

Scripting DIRAC commands

Of course, bash experts will be able to write scripts that perform all of the operations below purely in bash. This is left as an exercise for the reader - answers on a punch card please! (Also, we'll be using Python for the DIRAC Python API, so it's not a bad thing to use Python at this stage!)

6.3.1. Uploading files

The DIRAC file upload command takes the following form:

```
$ dirac-dms-add-file <LFN> <FILE> <SE>
```

where:

- <LFN> is the Logical File Name (LFN) of the entry for the file in the DIRAC File Catalog (DFC);
- <FILE> is the path to the file on your local machine, and;
- <SE> is the name of the destination Storage Element (SE).

Finding SE names

Remember, you can find the names of the available SEs with the dirac-dms-show-se-status command.

Suppose we have a number of files on our local machine in /home/gridpp/mydata/ that we want to upload to the grid. The following Python code will generate a bash script that will upload them to one of the Queen Mary Storage Elements:

```
$ cat make_upload_script.py
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import os, glob

data_path = '/home/gridpp/mydata'

lfn_dir = '/gridpp/user/a/ada.lovelace/mydata/'

se = 'UKI-LT2-QMUL2-disk'

s = "#!/bin/bash\n"

for my_file in sorted(glob.glob(data_path + "/*")):
    base_name  = os.path.basename(my_file)
    upload_lfn = os.path.join(lfn_dir, base_name)
    s += "dirac-dms-add-file %s %s %s\n" % (upload_lfn, my_file, se)

with open("upload_script.sh", "w") as sf:
    sf.write(s)
```

After you've generated a proxy and sourced the DIRAC environment, you can run the generated script as follows:

```
$ python make_upload_script.py
$ chmod a+x upload_script.sh
$ . upload_script.sh
```

The results of this will, of course, depend on the contents of `/home/gridpp/mydata/`, but all being well you should see the message:

```
Successfully uploaded file to UKI-LT2-QMUL2-disk
```

(or whichever SE you specified in your Python code) after each file has been uploaded.

Using Screen

If you're uploading a lot of files, you may wish to consider using something like the screen tool so that you can log off your terminal session and come back to it later.

And there we go! Multiple file uploads, all registered in the DIRAC File Catalog, using a DIRAC command line tool and a bit of (admitedly slightly clumsy) coding.

6.3.2. Replicating files

Now, as we did with the DFC CLI, we can also make replicas of files, list information about the replicas of a given file, and remove replicas with the following command line tools:

```
dirac-dms-replicate-lfn <LFN> <SE>
dirac-dms-lfn-replicas <LFN>
dirac-dms-remove-replicas <LFN> <SE>
```

Likewise, we can take the same approach with...

6.3.3. Downloading and removing files

```
dirac-dms-get-file <LFN>
dirac-dms-remove-files <LFN>
```

i.e. the DIRAC command line tools exist for these operations. However, getting information from the DFC about which files you would like to replicate, download, remove, etc. is non-trivial when taking the command line approach. This is especially true if you're writing scripts.

One approach is to use the **metadata** functionality the DIRAC File Catalog provides to find files of interest.

Metadata

Metadata *is data about the data. By assigning metadata to the files we upload to the DIRAC File Catalog, we can perform queries that will select only the files we are interested in. It also helps us to manage our data. We'll find out more about the DFC's metadata functionality later.*

The `dirac-dms-find-lfns` command finds LFNs based on the DFC path and metadata query supplied as options. For example, to find all files in the DFC that have been assigned to the experiment `UserGuide`, we can type:

```
dirac-dms-find-lfns Path=/ "experiment=UserGuide"
{'experiment': 'UserGuide'}
/gridpp/userguide/WELCOME.md
```

`experiment` here is the **metadata element** or **index**. This is a string assigned to the file's LFN that, in this case, has the value `UserGuide`. We can use the results of this to download the files we want.

```
$ dirac-dms-get-file /gridpp/userguide/WELCOME.md
{'Failed': {},
 'Successful': {'/gridpp/userguide/WELCOME.md': '/home/gridpp/tmp/WELCOME.md'}}
$ cat WELCOME.md
#Welcome to GridPP!

It looks like your download has worked. Congratulations!
```

Let's take a closer look at the DFC's metadata functionality using the DFC CLI.

6.4. First steps with the DIRAC metadata

6.4.1. Finding files using metadata

When you're uploading vast amounts of data, it's nice to be able to find it later. **Metadata** - data *about the data* - can help with this. DIRAC allows you to assign metadata such as strings, integers, and floating point numbers to files and directories (via their Logical File Names in the DIRAC File Catalog). You can then query the DFC to return a list of the files you want.

For example, once you have sourced your DIRAC environment, generated a proxy, and started the DFC CLI, you can find all files associated with the `UserGuide` experiment like so:

```
FC:/> find / experiment=UserGuide
Query: {'experiment': 'UserGuide'}
/gridpp/userguide/WELCOME.md
QueryTime 0.98 sec
```

We have assigned the value UserGuide to the file WELCOME.md for the experiment element or index. The find command in the DFC CLI performs the query for us.

```
FC:/> help find
 Find all files satisfying the given metadata information

        usage: find [-q] [-D] <path> <meta_name>=<meta_value> [<meta_name>=<meta_value>]

FC:/> exit
```

In our query above, <path> was / (i.e. search the entire catalog from the base directory), <meta_name> was experiment (i.e. a metadata string index indicating to which experiment the data belongs), and <meta_value> was UserGuide (OK, so the UserGuide isn't really an experiment - at least not in the scientific sense - but you get the idea!).

Getting help with the DFC CLI

You can get a list of all of the available commands in the DFC CLI by using the help command. To list the instructions for a given command (as above), type help [command].

There is only one file belonging to the UserGuide experiment in the DFC, and it's a pretty harmless MarkDown file. But you can hopefully see how, particularly when we start using multiple metadata indices with different *types*, DIRAC's metadata functionality is going to be pretty useful.

6.4.2. Assigning metadata to a file

We can also use the DFC CLI to *assign* metadata to our files. Let's create a file with our favourite text editor and upload it to the grid using the DFC CLI:

```
$ vim TODO.md
$ cat TODO.md
ToDo
====
* Email Charles re. engine
* Re-do punchcards
* Write to Dad
$ dirac-dms-filecatalog-cli
Starting FileCatalog client

File Catalog Client $Revision: 1.17 $Date:

FC:/> add /gridpp/user/a/ada.lovelace/TODO.md TODO.md UKI-LT2-QMUL2-disk
File /gridpp/user/a/ada.lovelace/TODO.md successfully uploaded...
```

We can now set the `owner` index for the LFN using the `meta set` command:

```
FC:/> meta set /gridpp/user/a/ada.lovelace/TODO.md owner ada.lovelace
/gridpp/user/a/ada.lovelace/TODO.md owner ada.lovelace
```

We can now find the file again using the `find` command:

```
FC:/> find / owner=ada.lovelace
Query: {'owner': 'ada.lovelace'}
/gridpp/user/a/ada.lovelace/TODO.md
QueryTime 0.01 sec
```

As we've said before, the DFC CLI is useful for small-scale operations on your data. Hopefully, though, you can start to appreciate the power of **metadata** when it comes to organising your data and performing analyses on it.

The most important thing for the moment, though, is that we can now put data on the Grid (i.e. on a Storage Element). This means we can use it in our Grid jobs without needing to upload with our job as an `inputfile`. We'll now complete making our example workflow fully Grid-enabled in the next section.

6.5. Checklist

- I know what a Grid Storage Element (SE) is;
- I know what the DIRAC File Catalog (DFC) is and what it is used for;
- I know what LFN stands for and what it means with respect to the DFC;
- I can access the DIRAC File Catalog Command Line Interface (DFC CLI);
- I can find the Grid Storage Elements (SEs) available for me to use;
- I can use tab-complete with `dirac-dms-` to find the available DIRAC command;
- I can list the contents of my Virtual Organisations's area in the DFC
- I can create a user area within this area and set the permissions accordingly;
- Using both the DFC CLI and the DFC command line tools, I can:
- upload and download files to and from an SE;
- replicate files to another SE;
- remove a replica from a specified SE;
- remove a file from the DFC.
- I know how to assign metadata to an LFN using the DFC CLI.

6.6. Testing

- **Accessing data on a GridPP Storage Element (SE)**: If everything is setup correctly, the following commands should result in the output below:

```
$ cd ~
$ mkdir tmp
```

```
$ cd tmp
$ dirac-dms-get-file LFN:/gridpp/userguide/WELCOME.md
{'Failed': {},
 'Successful': {'/gridpp/userguide/WELCOME.md': '/home/alovelace/tmp/WELCOME.md'}}
$ cat WELCOME.md
#Welcome to GridPP!

It looks like your download has worked. Congratulations!
```

You should have been able to follow everything else in the previous subsections too, of course - but much of the input and output will depend on your username, VO, etc. - the real test come when you start putting data in your own user area in the next section!

7. Using Grid Data in Your Workflow

The example workflow we've used so far isn't particularly sophisticated, but it does allow us to demonstrate the final key concept we'll look at here: incorporating Grid-based data in your workflows. Below we'll go through:

- Putting the initial input data onto the Grid;
- Using that data as the input to your workflow;
- Writing the output data from your workflow to the Grid;
- Retrieving the output from the Grid.

You'll need to have worked through Section 5 first – mainly because we actually only need to tweak a few lines to achieve what we want!

7.1. Putting our input data onto the Grid

The first thing to do is put the ZIP file containing the raw frame data into your user area on the DFC. You know how to do this (if not, have another look at Section 6) so now we're getting into the realm of user areas, VOs, etc. we're not going to give the explicit commands for this part. We'll also leave you to come up with a LFN for the file, and choose which SE to use (you might have a favourite by now).

- Download the ZIP file to your working directory;
- Upload it to your user area in your VO's area in the DFC;
- **Note down the LFN you assigned the file**. You'll need this below.

Structuring your LFNs

While the directories don't strictly exist with LFNs, it's useful to keep things organised with sensible structuring/naming conventions. Use the DFC CLI to create directories in your user area as required.

7.2. Getting data from the Grid

Thanks to Ganga, there's actually not much to using a Grid-hosted data file as input. All you need to do is add a `DiracFile` to the job's `inputputfile` list with the LFN as the input argument. So Ada's modified `dirac_job.py` script would have the line:

```
j.inputfiles = [ DiracFile('LFN:/.../CERNatschool_backgroundrad_dataset.zip') ]
```

Replacing the `...` in the LFN with the full path. The job will now retrieve the ZIP file from whichever Storage Element 1) has a replica of the file and/or 2) is closest to the site running the job to the working directory, just as it would with the `LocalFile`.

Replica loaction and management

In fact, GridPP DIRAC will work out where is best to send your job based upon where you have replicas of the file (i.e. which SEs you added/replicated it on). So once you're into optimisation territory, replica management is something to think about.

7.3. Writing data to the Grid

What about the output data? If you have an intermediary data layer (i.e. output that is used as input for another job/workflow) you may wish to write the output to the Grid. This is possible with a few tweaks, but there's a slight subtlety: GridPP DIRAC will assign LFNs for your job output based on the DIRAC job ID and an LFN base specified in your `.gangarc` file. This can be set with something like the following:

```
[DIRAC]
DiracLFNBase = /gridpp/user/a/ada.lovelace
```

Preparing to submit

Make sure you set this before starting Ganga and submitting your job(s).

Specifying which files get written to the Grid is then pretty similar to specifying the input files - switch the `LocalFile` to `DiracFile`:

```
j.outputfiles = [ DiracFile('output_images.tar') ]
```

With these changes made (and maybe a change of job name), you can now submit your job.

7.4. Retrieving the output from the Grid

You already know how to retrieve files from the Grid. The only extra detail you'll need to know is the DIRAC job ID. **This is different to the job ID in Ganga**. Both can be obtained with the following commands within Ganga:

```
Ganga In [X]: j.id
Ganga Out [X]: 1

Ganga In [X]: j.backend.id
Ganga Out [X]: 1234567
```

(i.e. the DIRAC ID will have many more digits.)

The DIRAC ID will determine the LFN the output files are assigned. So once the job has finished running, you should end up with something like this:

```
$ dirac-dms-filecatalog-cli
Starting FileCatalog client

File Catalog Client $Revision: 1.17 $Date:

FC:/> cd gridpp/user/a/ada.lovelace/
FC:/gridpp/user/a/ada.lovelace>ls
1234
FC:/> ls 1234
1234567
FC:/> ls 1234/1234567
frames.json
output_images.tar
```

So the full LFN for the image archive is:

```
LFN:/gridpp/user/a/ada.lovelace/1234/1234567/output_images.tar
```

This can be used to retrieve the file in the ways we have described already - or used as an `inputfile` to another job.

So there we go - we've completely Grid-ified our example workflow. You should now have all of the tools you need to start making your own workflows Grid-ready. Of course, there's a lot more that can be done and we'll mention some of the more advanced topics in the next section. But you should have plenty to get your teeth into for now!

7.5. Checklist

- I can successfully modify the Grid workflow example to:
- use an input `DiracFile` to with an lfn corresponding to data I have uploaded to the Grid;
- write the output data to my user area on the DFC.
- I can set the output LFN base in my Ganga configuration file;
- I can run the modified Grid workflow example and retrieve the output from the Grid.

7.6. Testing

- **Successful running of the CERN@school example job**: Does Figure 1 look familiar? It really should do by now.

8. What's Next?

8.1. Uploading your software to CVMFS

To start with, and if your software package/packages is/are small enough, you can probably get away with uploading your software as `LocalFiles` with your Grid job (perhaps using an archive file). *In fact, you may prefer this approach as http-based CVMFS repositories are world-readable.*

However, at some point you and/or your Virtual Organisation are going to want your own CVMFS repository. For small, UK-based VOs the best way to do this is on the RAL Tier-1 Stratum-0. Contact us to find out more about doing this - access to the repository is governed by your Grid certificate.

In short, the process of uploading your software amounts to:

- **Generating a proxy with DIRAC**: as this will be used to determine who you are and which repository you are accessing;
- **Logging in to the repository**: You can then log in to your CVMFS repository with:

```
$ gsissh -p 1975 cvmfs-upload01.gridpp.rl.ac.uk
Last login: [Date/time] from [hostname]

  _. _.   o| _ ._
 (_|(_||_|||(_)| |
      |

         Location: r89.harwell.europe hpd r89rack137
           Branch: cc34/cvmfs-uploader (sandbox)
        Archetype: ral-tier1
      Personality: cvmfs-uploader
 Operating System: sl640-x86_64
    Snapshot Date: 2016-10-26

[{vo-name}sgm@cvmfsXXXXX ~]$ cd cvmfs_repo
bin lib code README.md
```

- **Retrieve your software**: You can now place your software in the repository, arranging it however works best for you. If your code is hosted in an online Git-based repository, simply `git clone` it straight to an appropriate directory.

Looking at other CVMFS repositories

You can take inspiration from other VOs. For example, you can browse the ATLAS experiment's (CERN-hosted) CVFMS repository with: code$ ls /cvmfs/atlas.cern.ch/ *and so on.*

Upload times

Once put into the repository, it can be several hours before it becomes available on the worker nodes - it's not instant. Make sure your software is well-tested before putting it up - CVMFS is not appropriate for software under development!

However, once it *is* on there, it's available everywhere. Which is nice.

8.2. Advanced DIRAC functionality

DIRAC has a great deal of functionality of its own, particularly when you start looking at the Python API. However, Ganga provides a nice wrapper for much of this so you shouldn't need to touch it. You can find out more on the DIRAC homepage, and check out the source code on their GitHub page.

8.3. Advanced Ganga functionality

Likewise, there is a lot more to Ganga than we have covered here. Ganga has its own documentation page:

`http://ganga.readthedocs.io`

which features things like:

- Configuration;
- Job manipulation;
- Splitters;
- Post-processors;
- Queues;
- Etc.

The GitHub repository is also worth watching for the latest updates and developments, as well as raising any bugs or problems you may have in the Issues section.

8.4. And finally...

Moving your workflow to the Grid won't necessarily be straightforward, but at GridPP we're here to help – if you've got a problem, just ask! To keep up to date with the UserGuide, watch the *UserGuide* GitHub repository to be notified of Issues and Pull Requests relating to additions or improvements to the *UserGuide*.

Many thanks for reading so far, and happy Gridding!

9. Troubleshooting

This section contains brief notes on specific problems users have encountered when working on specific systems, generally raised via the GitHub Issues page.

9.1. Ganga

Ganga isn't working.

If you're having problems with Ganga, the best thing to do is to raise an issue with the Ganga dev team through the GitHub repository. At the time of writing, we've been using Ganga version 6.3.1 - but as Ganga is under active development this may change so you may want to watch the repository too.

Ganga throws errors relating to not being able lock files.

If the file system your cluster is based on can't handle (or hasn't been set up to allow) file locks, which Ganga uses. If your home directory is on such a file system (e.g. NFS), files in your `~/gangadir` won't be lockable and Ganga won't work. Assuming your cluster can't be reconfigured, the simplest thing to do is change the location of your `gangadir` to somewhere that can handle file locks, such as a scratch directory on the cluster. To do this, set the `gangadir` option in `~/.gangarc` to something like:

```
gangadir = /scratch/alovelace/gangadir
```

and restart Ganga in the usual way.

9.2. Miscellaneous problems

My problem isn't listed here and search engines aren't helping either.

Raise an issue on the GitHub issues page and we'll see if we can help!

Good luck!

A. Creating a GridPP CernVM

If you don't have an account on a grid-ready cluster, don't worry - this section will show you how to create a **Virtual Machine** (VM) that will, essentially, act as a grid User Interface (UI) within whatever operating system you happen to be using at the moment. We will do this using the CernVM service [11], and create a guest CernVM on your host system. There are a number of reasons to do this:

1. The CernVM can act as a pre-built Grid User Interface (UI) that will give you all the tools you need (e.g. a command line terminal, text editors, etc.) to get the most out of the Grid;
2. Your CernVM will also give you instant access to the CernVM-File System (CVMFS). A lot of the software we will use to manage our grid jobs and data is provided using this, with the huge bonus of *not needing any installation by you*.
3. Most of the Grid tools we will use are compiled to run on Scientific Linux 6; With a CernVM you'll be able to use them out of the box;
4. The standard Grid Worker Nodes you'll be using to run your software on use SL6 machines. If your code runs on your CernVM, it will run on the Grid;
5. What's more, if everyone uses the CernVM as their Grid UI, we at GridPP will only have to support one operating system (i.e. the SL6 supplied with the CernVM). If we're singing from the same (virtual) hymn sheet, we'll be able to recreate your problems and help you solve them more easily.

All you need to provide is the RAM and the hard disk space on your local host machine.

SL6 cluster access?

If you already have access to a user account on an SL6 terminal, for example on a university computing cluster, you can probably skip this section.

A.1. An overview of the process

To create and configure a CernVM that will meet our needs, you will need to:

1. Install some virtualisation software on your host machine;
2. Download the appropriate CernVM Virtual Machine baseline image from the CernVM service;
3. Create a new guest CernVM using the CernVM image;
4. Configure the guest CernVM so that it has access to the host hard disk;
5. Configure the CernVM for Grid use by applying the GridPP *contextualisation*.

Let's look at each of these stages in a bit more detail.

A.2. The virtualisation software

You can find a list of compatible virtualisation software solutions on the CernVM image download page here. It doesn't really matter which you use but we've had success with this version of VirtualBox (Windows 7).

Setting up the virtualisation software

This is the one bit that's a bit tricky for us to support - how you do this will depend on your local machine and its setup. Remember, search engines and StackOverflow are your friends here.

A.3. Creating your CernVM

Depending on the virtualisation solution you picked (and got working) above, download the baseline CernVM image file from here.

Then use your virtualisation software to create a new VM from the downloaded image. You should be able to find instructions for how to do this from your virtualisation software provider. Use as much RAM as you can spare (up to about half of your host machine's total RAM) and use a virtual hard disk of about 30 GB.

Once you have created your CernVM, but *before you start it*, you will need setup the *contextualisation* for your CernVM.

A.4. Contextualising your CernVM

The CernVM service offers the ability to contextualise a CernVM with pre-defined settings, environment variables, etc. that are put in place when the CernVM is first booted. This is known as *pairing* the VM. You can create your own contextuallisations, but it is also possible for individuals to create public contexts (e.g. for LHC experiments, open data initiatives, etc.) that anyone can use.

We have created such a context for GridPP. You can use it to get going with the Grid. Importantly, you do not need a CERN account to do this, so it is possible for anyone with a grid certificate to use it!

To pair your local CernVM with the GridPP context:

1. **Log in to the CernVM service**: You can do this here. If you don't have a CERN account, you can register here.
2. **Visit the CernVM Marketplace**: This can be found here.

3. **Select the GridPP CernVM context**: Select *Experimental* from the panel on the right, and then click on the `gridpp-cernvm` context.
4. **Boot up your local CernVM**: Once the boot process has finished, you should be presented with a login prompt. Don't enter anything yet.
5. **Pair with the GridPP context**: Returning to the GridPP CernVM Marketplace, click on the *Pair* button on the panel on the right. This will generate a six-figure PIN.
6. **Apply the context to your VM**: Returning to your booted-up CernVM, enter the PIN (preceeded by a hash, as instructed at the login prompt). The CernVM webpage will now update indicating the VM has been successfully paired. Your VM will then be restarted and contextualised.
7. **Log in to your GridPP CernVM**: Once the context has been applied to your CernVM, you will be presented with a graphical login screen (as opposed to the text login prompt). Use the username `gridpp` and password `gridpp` to log in to your CernVM.

And that's it! You now have a shiny new GridPP CernVM from which you'll be able to access the Grid.

A.5. Two final things

A.5.1. Configure git

We'll be using `git` (and GitHub) to access various pieces of code and scripts. The CernVM comes with `git` installed but you'll need to configure it with your username and email address. This can be done with the following commands:

```
$ git config --global user.name "Ada Lovelace"
$ git config --global user.email alovelace@qmul.ac.uk
```

A.5.2. Local hard disk access

Before we move on, it's important to note that you will need to be able to access the hard disk of your host machine from that of your guest CernVM. This is so that you can move any files that you need across to the CernVM - most importantly, your Grid certificate file.

With VirtualBox, for example, this is achieved using the Shared Folders functionality.

Before proceeding, you should make sure that you can access the parts of your local hard disk that contain any software or data that you want to copy across.

A.6. Checklist

- I have installed some virtualisation software on my local machine that is compatible with the CernVM images listed at:

http://cernvm.cern.ch/portal/downloads.

- I have downloaded the corresponding CernVM image to my local machine from the CernVM image download page;
- I have created a new guest VM on my local machine using the CernVM image;
- I have registered with the CernVM service (or I am a CERN computing account holder);
- I have generated a six-digit PIN to pair my CernVM with the GridPP context via the CernVM Marketplace;
- I have entered the six-digit PIN (preceeded by the hash symbol) into the login prompt of my newly-booted CernVM;
- I have logged in to my new GridPP CernVM using the `gridpp` user account;
- I have accessed folders from my local machine's hard disk from my GridPP CernVM.

A.7. Testing

- **Downloading the CernVM image**: Your `Downloads` folder (or whichever location you chose to download the CernVM image file) contains the image file.

```
$ cd ~/Downloads
$ ls -l | grep cernvm
cernvm-3.5.1.iso
```

- **Creating the CernVM**: When you start up your CernVM from your virtualisation software, you are eventually presented with a login screen.

```
Welcome to CERN Virtual Machine, version 3.5.1.4
  based on Scientific Linux release 6.6 (Carbon)
  Kernel 3.18.20-18.cernvm.x86_64 on an x86_64

IP Address of this VM: [IP address]
In order to apply cernvm-online context, use #<PIN> as user name.

localhost login: _
```

- **Registering with the CernVM Service**: You can login via this page. When you access this page you are redirected to the CernVM Dashboard.

- **Pairing your CernVM with the GridPP CernVM context**: After selecting the GridPP CernVM context in the CernVM Marketplace and clicking on the *Pair* button on the right-hand panel, you are presented with a six-digit number. After entering the PIN into your CernVM's login screen (preceeded by the hash symbol), your CernVM reboots to display a graphical CernVM login screen. Meanwhile, the CernVM Pairing web page has updated to display the message, > **Setup instance - Completed** > > Your vm is now paired with CernVM Online! Upon logging in to your new CernVM with the username `gridpp` and the password `gridpp`, you are presented with the CernVM desktop.

- **Accessing your local machine's hard disk from you CernVM**: You can access (from the command line or otherwise) folders on your local machine's hard disk on the GridPP CernVM. So (for example) if you're using VirtualBox with the *Shared Folders* functionality, after adding the folders you want access to via the VM's *Settings* and rebooting the VM you'll do something like this:

```
$ sudo usermod -a -G vboxsf gridpp
[sudo] password for gridpp: # Enter 'gridpp' here.
```

before loggin out and logging in again. You'll then be able to access the folders you specified.

```
$ cd /media/sf_alovelace/ # Shared Folder "alovelace"
$ ls
punch-card-01.dat dad-poem-001.txt my-poem-005.txt
```

References

[1] The GridPP Collaboration. GridPP: Development of the UK Computing Grid for Particle Physics. *J. Phys. G*, 32:N1–N20, 2006.

[2] D. Britton et al. GridPP: the UK grid for particle physics. *Phil. Trans. R. Soc. A*, 367:2447–2457, 2009.

[3] CMS Collaboration. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Phys. Lett. B*, 716:30, 2012.

[4] ATLAS Collaboration. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Phys. Lett. B*, 716:1, 2012.

[5] L. Evans and P. Bryant. LHC Machine. *JINST*, 3:S08001, 2008.

[6] J. Blomer et al. The Evolution of Global Scale Filesystems for Scientific Software Distribution. *Computing in Science and Engineering*, 17:61–71, 2015.

[7] Tsaregorodtsev, A. and others. DIRAC pilot framework and the DIRAC Workload Management System. *J. Phys.: Conf. Ser.*, 219:062049, 2010.

[8] The LHCb Collaboration. The LHCb Detector at the LHC. *JINST*, 3:S08005, 2008.

[9] D. Bauer et al. The GridPP DIRAC project: Implementation of a multi-VO DIRAC service. *J. Phys.: Conf. Ser.*, 664:062009, 2015.

[10] D. Bauer et al. The GridPP DIRAC project - DIRAC for non-LHC communities. *J. Phys.: Conf. Ser.*, 664:062036, 2015.

[11] G. Ganis et al. Status and Roadmap of CernVM. *J. Phys.: Conf. Ser.*, 664:022018, 2015.

Acknowledgements

The author would like to thank GridPP Collaboration members Steve Jones (Liverpool) and Steve Lloyd (QMUL), as well as the students from the Institute for Research in Schools, for providing invaluable feedback on the initial drafts of the *GridPP UserGuide*. Andrew McNab (Uni. Manchester) has provided invaluable technical support for the operation and maintenance of the GridPP website on which the online version of this guide is hosted[†]. Much inspiration for the structure and philosophy of the *GridPP UserGuide* has been drawn from Michael Hartl's *Ruby on Rails Tutorial* - an excellent read (even if you're not particularly fussed about Ruby on Rails). We are also grateful to GitHub for our Organizational Repository – thanks!

This work was supported by the UK Science and Technology Facilities Council (STFC) via the GridPP Collaboration [1, 2] and grant ST/N00101X/1 as part of work with the CERN@school research programme.

About the author

Tom Whyntie is Public Engagement Fellow at the School of Physics & Astronomy, Queen Mary University of London. He is supported by the UK's Science and Technology Facility Council (STFC) through their Public Engagement Fellowship programme and through the GridPP Project. Having used the Worldwide LHC Computing Grid (WLCG) extensively during his time on the Compact Muon Solenoid (CMS) experiment at CERN's Large Hadron Collider, Tom works as part of the GridPP Community to engage as many people as possible with the Grid through GridPP's New User Engagement Programme. He is also consultant scientist for the CERN@school programme, Scientific Officer for the Institute for Research in Schools, and Software Coordinator for the MoEDAL Collaboration.

[†] See http://www.gridpp.ac.uk/userguide

Version History

Table 2: Version history.

Version	Description	DOI	Author
1.0	Initial version.	10.5281/zenodo.222702	TW