

## 0 Preliminaries

During this course, you are working on portals prepared by GridKa. The login information are included in the welcome package which you have received at the beginning of the GridKa School. Please make sure that you have a stable network connection and that you can login to the server assigned to you. If you are running Windows on your laptop, you need a SSH client. If not already installed, load Putty from <http://www.chiark.greenend.org.uk/~sgtatham/putty/>. Under Linux or MacOS, you can simply open a shell.

- use the credentials from your welcome package to login to the specified portal (use the options `-p 24 -X` to use port 24 and to enable X-forwarding)
- wired network access is available in the first rows
- download the material of the course and unzip it:

```
> wget http://tinyurl.com/gks2009/course.tar.gz
> tar xfvz course.tar.gz
> cd course
```
- create a proxy to submit your jobs and check its content

```
> voms-proxy-init -voms dech -valid 24:00
> voms-proxy-info -all
```
- execute `export LS_COLORS=""` if you don't like the colors of the shell

## 1 Greeting the World!

This task introduces the first and most important commands of the gLite middleware. Following tradition, your first grid job will simply print "Hello World!". The commands will reappear in each subsequent task, so do all steps carefully and don't hesitate to ask if you encounter any problems.

- change to the subdirectory `course/01_hello_world`
- have a look at `hello_world.jdl`
- submit your first job by executing

```
> glite-wms-job-submit -a hello_world.jdl
```
- remember the job identifier (e.g. copy it to the clipboard)
- check the status of your job with `glite-wms-job-status` one or two times a minute to see the different states of a grid job
- get the output of your job with `glite-wms-job-output` (don't forget to specify an output directory with `--dir!`)

## 2 Get Information!

This task extends the previous one. The job is now a shell script that prints some information about the environment on the worker node where the job is actually executed. Additionally, you learn how to pass arguments to an executable. For many jobs, you will make use of both environment variables on the worker node and arguments.

- change to the subdirectory `course/02_get_infos`
- have a look at `job.sh` and execute it locally
- note the new statements in `get_info.jdl`
- submit the job to the grid as before and compare its output to the local output
- note the numerous extra environment variables on the worker node

## 3 Use Requirements!

In this task you become acquainted with requirements and site matching. You will need this very often, e.g. to specify the CPU time your job typically needs or to run a job that needs a certain architecture or software environment.

- change to the subdirectory `course/03_requirements`
- write a `jdl`-file without any requirements
- invoke `glite-wms-job-list-match` to see how many sites really match
- it is not necessary to submit jobs during this task
- add one requirement after the other and check how many sites still match
- if your job does not match check the syntax of your requirements, misspelled words prevent your job from matching without any further notice
- if there is still plenty of time, you can do an extra task:
  - test for the tag `VO-dech-fortune` in your requirements
  - define `$VO_DECH_SW_DIR/fortune.sh` as executable
  - submit the job and look at the output

## Example Statement

In the following example statement, the four different types of statement are shown.

```
Requirements = !RegExp("rwth", other.GlueCEUniqueID)
    && other.GlueHostNetworkAdapterOutboundIP
    && other.GlueCEPolicyMaxCPUTime >= 60
    && other.GlueCEPolicyMaxWallClockTime >= 60
    && other.GlueHostMainMemoryRAMSize >= 512
    && Member("VO-dech-gks09",
        other.GlueHostApplicationSoftwareRunTimeEnvironment);
```

## Glue Tags

The following list explains the most important glue tags. A complete list can be found in appendix G of the gLite manual.

**GlueCEUniqueID:** name of the Computing Element, including the queue (you can get a list of this information easily with `lcg-infosites --vo dech ce`)

**GlueCEPolicyMaxCPUTime:** required CPU time (affects esp. the assignment to a queue)

**GlueCEPolicyMaxWallClockTime:** required total running time (also affects the assignment to a queue)

**GlueHostOperatingSystemName:** name of the operating system on the workernode

**GlueHostOperatingSystemRelease:** release of the operating system on the workernode

**GlueHostArchitecturePlatformType:** processor architecture on the worker node

**GlueHostApplicationSoftwareRunTimeEnvironment:** list of tags set by the software managers of the VO, has to be accessed with regular expressions

**GlueCEStateEstimatedResponseTime:** estimated time between submission to the site and start of the job on a worker node (for ranking)

**GlueCEStateFreeCPUs:** number of free cpus on a site (useful for ranking)

**GlueHostNetworkAdapterOutboundIP:** boolean variable that specifies if the worker node has an outbound IP, i.e. a way to the internet

## 4 Manage your Data!

This task introduces data management on the grid. You will spread the words of one of Shakespeare's most famous plays. Hopefully, it will not result in a tragedy but in deeper insight how data is stored on the grid.

- find the LFC server for your VO and set the environment variable `LFC_HOST`

```
> lcg-infosites --vo dech lfc
rb.scai.fraunhofer.de
> export LFC_HOST=rb.scai.fraunhofer.de
```

- create a new directory under `/grid/dech/gks/` (replace `YOURNAME` with your name)

```
> lfc-mkdir /grid/dech/gks/YOURNAME
```

- do an `lfc-ls` on `/grid/dech/gks/`
- get a list of available storage elements with `lcg-infosites --vo dech se`
- upload `romeojuliet.txt` in `04_data_management` to a storage element and assign an LFN (`$PWD` points to the current directory)

```
> lcg-cr -d scaise-2.scai.fraunhofer.de \
-l lfn:/grid/dech/gks/YOURNAME/romeojuliet.txt \
file://$PWD/romeojuliet.txt
```

- check where the file has been stored using the LFN and the guid (don't miss the prefixes)

```
> lcg-lr lfn:/grid/dech/gks/YOURNAME/romeojuliet.txt
> lcg-lr guid:bdc2crazy-3hex-4num-ber7-acf4ab04ca7c
```

- replicate the file with `lcg-rep` to another storage element and repeat the last step
- download the file using `lcg-cp`, the option `-v` gives more information about the download process
- add a comment to a file and check it with `lfc-ls --comment`
- remove the file from one storage element
- remove the file from all storage elements (`lcg-del -a`)
- `lcg-lr` should now give a `No such file or directory` message

## 5 Let's animate!

The last task of this course combines all previously learned commands. You will generate a video animation using parameterized jobs on the grid. Although this task is very tough, it gives you an impression of the “real life” and, in the end, it gives you an inspiring video.

The animation consists of 100 frames that can be easily split into 10 subprocesses. The rendering is done by POV-Ray, an open-source raytracer. The source code for the scenes, the script to call the renderer as well as the x86-binaries of POV-Ray are given. You find all files in `course/05_render_on_the_grid` and its sub-directories. After your grid jobs have succeeded, `make_movie.sh` helps you to build an animation out of the single frames.

- upload `povlinux-3.6.tgz` and replicate it to several other storage elements (use `/grid/dech/gks/YOURNAME` as in the previous task); do not change the name of the archive
- open `job.sh` to add the two missing statements: one to set `LFC_HOST` and one to copy `povlinux-3.6.tgz` to the worker node
- extend `render_on_the_grid.jdl`:
  - add the statement for a parameterized job with a number going from 0 to 9 and pass this number as an argument to the executable (i.e. `job.sh`)
  - store `StdOut` and `StdErr` in files
  - the output sandbox should include the previous output and the output of `job.sh`, `ergebnis_PARAM_.tar.gz` (`_PARAM_` is equal to the argument you pass to `job.sh`)
  - you can use the tag `V0-dech-gks09` in your requirements
  - ensure that your jobs run only on CEs “close to” SEs where you have replicated `povlinux-3.6.tgz`
  - the input sandbox should include `job.sh` and the two files from the directory `source/<your favorite scene>/` that describe the scene you want to render
- make sure that some sites match your requirements before sending the job
- submit your jobs; in case of any errors try to debug your job using the output
- call `make_movie.sh` after all jobs have succeeded and enjoy the video