DIRAC Service Tutorial

This is a brief introduction to the DIRAC system based on examples of commands usage. For more detailed tutorials please visit the web pages of the project:

- https://github.com/DIRACGrid/DIRAC/wiki
- https://github.com/DIRACGrid/COMDIRAC/wiki

1. Getting started

Set the DIRAC client environment

All the DIRAC commands are available once the environment is set up ::

```
$ source /opt/dirac/bashrc
```

This is usually done in the login script of a user account

Getting help

All the DIRAC commands have *-h* argument to provide help information, for example ::

```
$ dirac-info -h

Report info about local DIRAC installation
Usage:
  dirac-info [option|cfgfile] ... Site

General options:
  -o  --option <value>         : Option=value to add
  -s  --section <value>        : Set base section for relative parsed options
  -c  --cert <value>           : Use server certificate to connect to Core Services
  -d  --debug                  : Set debug mode (-ddd is extra debug)
  -   --autoreload             : Automatically restart if there's any change in the modu
  -   --license                : Show DIRAC's LICENSE
  -h  --help                   : Shows this help
```

In case of problems, executing commands with *-d* or even *-ddd* flag will provide additional output.

Get information

Get information about the client and service that the client will work with

```
$ dirac-info
```

Get information about user credentials

```
$ dirac-proxy-info
```

Client configuration

The DIRAC client configuration is stored in $HOME/.dirac/dcommands.conf file. The configuration can be visualized and edited with the *dconfig* command ::

```
$ dconfig
[global]
default_profile = training_user

[training_user]
group_name = training_user
home_dir = /
default_se = DIRAC-USER
```

Changing some options::

```
$ dconfig training_user.home_dir=/training.egi.eu/user/u/user40
```

Starting DIRAC session

The DIRAC client session is started with *dinit* command ::

```
$ dinit training_user
Enter Certificate password:
subject      : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev/CN=proxy
issuer       : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev
identity     : /O=GRID-FR/C=FR/O=CNRS/OU=CPPM/CN=Andrei Tsaregorodtsev
timeleft     : 23:59:59
DIRAC group  : training_user
rfc          : False
path         : /tmp/x509up_u1042
username     : atsareg
properties   : NormalUser
```

Users with PUSP pregenerated proxies (Per User SubProxies), please do ::

```
$ dinit -p
subject      : /C=IT/O=INFN/OU=Robot/L=Catania/CN=Robot: EGI Training Service - Diego Or
issuer       : /C=IT/O=INFN/OU=Robot/L=Catania/CN=Robot: EGI Training Service - Diego Or
identity     : /C=IT/O=INFN/OU=Robot/L=Catania/CN=Robot: EGI Training Service - Diego Or
subproxyUser : user40
timeleft     : 20:41:31
DIRAC group  : training_user
rfc          : True
path         : /tmp/x509up_u1040
username     : user40
properties   : NormalUser
VOMS         : True
VOMS fqan    : ['/training.egi.eu']
```

2. First Job

To submit you first job you can use *dsub* command by doing just ::

```
$ dsub echo.sh Hello World
```

The job status can be now examined with *dstat* command ::

```
$ dstat
JobID    Owner   JobName  OwnerGroup     JobGroup  Site  Status   MinorStatus
=============================================================================
9022022  user40  echo.sh  training_user  NoGroup   ANY   Waiting  Pilot Agent Submissior
```

This shows all the jobs currently being executed and not reached yet their final state. To see the jobs *Done* ::

```
$ dstat -S Done
JobID    Owner   JobName     OwnerGroup     JobGroup  Site                 Status  Minc
=============================================================================
9022022  user40  echo.sh     training_user  NoGroup   CLOUD.bifi-unizar.es  Done    Exec
```

Now it is time to get the output with the *doutput* command ::

```
$ doutput 9022022
$ ls -l 9022022
total 4
-rw-r--r-- 1 user40 dirac 12 Nov  8 11:41 std.out
$ cat 9022022/std.out
Hello World
```

3. Understanding job description

JDL description

The job description is done with a JDL language, example ::

```
$ cat echo.jdl
[
JobName = "Test_Hello";
Executable = "echo.sh";
Arguments = "Hello world !";
StdOuput = "std.out";
StdError = "std.err";
InputSandbox = {"echo.sh"};
OutputSandbox = {"std.out","std.err"};
]
```

Edit the *Arguments* field and submit a job using its JDL description ::

```
$ dsub -J echo.jdl
9022409
```

Exercise

- Modify the JDL parameters in the "echo.jdl" file in your local directory
- Submit jobs and verify that your modifications are taken into account

4. Getting data on the Grid

DIRAC File System

DIRAC is presenting the grid/cloud storages as a single file system accessible with Unix-like commands ::

```
dpwd, dls, dcd, dmkdir, dchmod, dchgrp, dchown, dfind, drm
```

Some examples ::

```
user40@stoor16:~$ dpwd
/training.egi.eu/user/u/user40
user40@stoor16:~$ dls -l
/training.egi.eu/user/u/user40:
-rwxrwxr-x 2 user40 training_user 12 2015-11-07 23:33:43 std.out
-rwxrwxr-x 2 user40 training_user 30 2015-11-07 23:19:06 test.sh
user40@stoor16:~$ dmkdir newdir
user40@stoor16:~$ dls -l
/training.egi.eu/user/u/user40:
drwxrwxr-x 0 user40 training_user  0 2015-11-08 12:14:05 newdir
-rwxrwxr-x 2 user40 training_user 12 2015-11-07 23:33:43 std.out
-rwxrwxr-x 2 user40 training_user 30 2015-11-07 23:19:06 test.sh
user40@stoor16:~$ dchmod 755 test.sh
user40@stoor16:~$ dls -l
/training.egi.eu/user/u/user40:
drwxrwxr-x 0 user40 training_user  0 2015-11-08 12:14:05 newdir
-rwxrwxr-x 2 user40 training_user 12 2015-11-07 23:33:43 std.out
-rwxr-xr-x 2 user40 training_user 30 2015-11-08 12:14:36 test.sh
user40@stoor16:~$ dcd newdir
user40@stoor16:~$ dpwd
/training.egi.eu/user/u/user40/newdir
```

Uploading data to the Grid

Get local file to the Grid with the *dput* command ::

```
$ dput test.sh test.sh
$ dls -l
/training.egi.eu/user/u/user40/newdir:
-rwxrwxr-x 1 user40 training_user 22 2015-11-08 12:21:09 test.sh
```

To see physical replicas of the file use *dreplicas* ::

```
$ dreplicas test.sh
/training.egi.eu/user/u/user40/newdir/test.sh:
   CYFRONET-USER dips://dirac-dms.egi.eu:9148/DataManagement/StorageElement/training.egi
```

If not specified explicitely, the default Storage Element is used. To choose another Storage Element do ::

```
$ dput echo.sh echo.sh -D TRAINING-USER
$ dreplicas echo.sh
/training.egi.eu/user/u/user40/newdir/echo.sh:
   TRAINING-USER dips://dirac-dms.egi.eu:9149/DataManagement/TrainingStorageElement/trai
```

Downloading data from the Grid

Get a remote file locally with *dget* command ::

```
$ dget echo.sh
$ ls -l
total 4
-rw-r--r-- 1 user40 dirac 22 Nov  8 13:32 echo.sh
```

Replicating data

Make another physical copy of the data with *drepl* command ::

```
$ dreplicas test.sh
/training.egi.eu/user/u/user40/newdir/test.sh:
 CYFRONET-USER dips://dirac-dms.egi.eu:9148/DataManagement/StorageElement/training.egi.e
$ drepl test.sh -D TRAINING-USER
$ dreplicas test.sh
/training.egi.eu/user/u/user40/newdir/test.sh:
 CYFRONET-USER dips://dirac-dms.egi.eu:9148/DataManagement/StorageElement/training.egi.e
 TRAINING-USER dips://dirac-dms.egi.eu:9149/DataManagement/TrainingStorageElement/traini
```

Removing data

Remove files with *drm* command, for example, the following will remove all the replicas of a file from the Grid ::

```
$ drm /training.egi.eu/user/u/user40/newdir/echo.sh
```

To remove just one copy from a given Storage Element do the following ::

```
$ drm /training.egi.eu/user/u/user40/newdir/test.sh -D TRAINING-USER
```

Exercise

- Repeat examples described above
- Create your own local file and upload it to the Grid
- Replicate it to another Storage Element
- Download the file from the grid
- Remove one replica of the file
- Remove the file completely from the Grid.

5. Jobs with output data

Specifying output data

Jobs produce data which should be made available after the job finishes. To instruct job to upload output data use *OutputData* JDL parameter. You can also specify optional *OutputSE* and *OutputPath* parameters. For example ::

```
[
JobName = "Mandel_tutorial";
Executable = "mandelbrot";
Arguments = "-X -0.464898 -Y -.564798 -W 600 -H 400 -M 1500 -P 0.0000092 mandel.bmp";
StdOuput = "std.out";
StdError = "std.err";
OutputSandbox = {"std.out","std.err","mandel.bmp"};
InputSandbox = {"mandelbrot"};
CPUTime = 1000;
OutputSE = "TRAINING-USER";
OutputPath = "/special_path";
OutputData = { "mandel.bmp" };
]
```

By default, the output data are stored in the user home directory in the DIRAC File Catalog, a subdirectory per job is created, for example ::

```
/training.egi.eu/user/u/user40/9022/9022770/mandel.bmp
```

OutputPath replaces the *job* subdirectory. With the example above the output file wil be ::

```
/training.egi.eu/user/u/user40/special_path/mandel.bmp
```

Finally, the *OutputData* field can be specified with a full Logical File Name (LFN). This will become an exact path in the catalog, for example ::

```
[
...
OutputData = { "LFN:/training.egi.eu/user/u/user40/tutorial/mandel.bmp" };
]
```

Note that you should be sure that you have write access to the specified location.

Downloading output data

The *doutput* command can be used to also download output data produced by the jobs, for example ::

```
$ doutput --Data 9022409
```

This command will download output data instead of the sandbox of the specified job. See the help information of the command for more options.

Exercise

- Submit several *mandelbrot* jobs with the output data going to the desired location
- Download the output data to your local disk

Hint: the *mandelbrot* program is available in the grid storage at this path ::

```
/training.egi.eu/user/u/user40/mandelbrot
```

Use "mandelbrot.jdl" example job description from your home directory

6. Jobs with input data

Jobs can process input data which will downloaded by DIRAC to the local disk of a running job. The input data is specified by *InputData* JDL parameter. You should provide LFN names of one or more files, for example ::

```
[
JobName = "Input_Test";
Executable = "/bin/cat";
Arguments = "echo.sh";
StdOuput = "std.out";
StdError = "std.err";
InputData = {"/training.egi.eu/user/u/user40/newdir/echo.sh"};
OutputSandbox = {"std.out","std.err"};
]
```

You can provide input data as part of the *InputSandbox* rather than *InputData*. This can be useful if you want to bypass the mechanism of scheduling your jobs only to the sites which are declared close to storage elements where your data reside. For example ::

```
[
...
InputSandbox = {"LFN:/training.egi.eu/user/u/user40/newdir/echo.sh"};
...
]
```

Exercise

- Submit jobs with input data and make sure that the data is processed by the job

7. Multiple job submission

Specifying parametric jobs

You can submit multiple jobs in one go by describing a sequence of parameters in the job JDL. For example ::

```
[
JobName = "Test_param_%n";
JobGroup = "Param_Test_1";
Executable = "echo.sh";
Arguments = "This job is for %s, the job ID is %j";
Parameters = {"Andrei","Alexandre","Victor","Pierre"};
StdOuput = "std.out";
StdError = "std.err";
InputSandbox = {"echo.sh"};
OutputSandbox = {"std.out","std.err"};
]
```

This description will generate 4 jobs, one for each value of *Parameters* field. The placeholders will be substituted

- *%n* -> parameter consecutive number
- *%s* -> parameter value

- *%j* -> job ID

The numerical parameters can be specified also with a formula ::

```
P(0) = ParameterStart
P(i) = P(i-1)*ParameterFactor + ParameterStep
```

In the JDL ::

```
[
...
Parameters = 10;
ParameterStart = 0;
ParameterStep = 0.02;
ParameterFactor = 1;
...
]
```

Job groups

All the jobs in the same bulk submission operation will belong to the same Job Group specified in the JDL description. This parameter can be used in various commands. Monitor job progress by group ::

```
$ dstat -g Param_Test_1 -a
JobID    Owner   JobName       OwnerGroup     JobGroup      Site                 Status
=============================================================================================
9023573  user40  Test_param_0  training_user  Param_Test_1  ANY                  Waitin
9023574  user40  Test_param_1  training_user  Param_Test_1  ANY                  Waitin
9023575  user40  Test_param_2  training_user  Param_Test_1  CLOUD.cesnet.cz      Done
9023576  user40  Test_param_3  training_user  Param_Test_1  CLOUD.bifi-unizar.es Done
9023577  user40  Test_param_4  training_user  Param_Test_1  CLOUD.ceta-ciemat.es Comple
9023578  user40  Test_param_5  training_user  Param_Test_1  ANY                  Waitin
9023579  user40  Test_param_6  training_user  Param_Test_1  CLOUD.ukim.mk        Runnin
9023580  user40  Test_param_7  training_user  Param_Test_1  CLOUD.cesnet.cz      Done
9023581  user40  Test_param_8  training_user  Param_Test_1  ANY                  Waitin
9023582  user40  Test_param_9  training_user  Param_Test_1  CLOUD.ceta-ciemat.es Done
```

Get outputs for all the jobs in a group ::

```
$ doutput -g Param_Test_1
```

Exercise

- Modify the *parametric.jdl* example from you home directory
- Submit parametric jobs
- Get the job status by their group
- Get the output of the jobs by their group

8. Putting this all together

Final exercise

Now we have all the elements for the final exercise.

- Create a sequence of images zooming into the Mandelbrot Universe using parametric jobs
- The images will be uploaded to the Storage Element as output data
- Once all the images are generated, download them local in one go
- Glue all the images together in a single movie

For the exercise you can start with the *mandelbrot_quest.jdl* in your home directory. Choose your own starting point to create a unique movie ! Try to do the exercise without reading further first.

Hints for the exercise

You can use the following sequence of commands to accomplish the exercise ::

```
$ dsub -J mandelbrot_quest.jdl
$ dstat -g Mandel_Quest -a
...
$ doutput -g Mandel_Quest -n -D mandel_quest --Data
$ cd mandel_quest
$ convert *.bmp mandel_movie.gif
```

This will result in *mandel_movie.gif* file. You can visualize it with a Web browser.