



LHC COMPUTING GRID

LCG-2 USER SCENARIO

<i>Document identifier:</i>	CERN-LCG-GDEIS-498081
<i>EDMS id:</i>	498081
<i>Version:</i>	v1.0
<i>Date:</i>	September 23, 2004
<i>Section:</i>	LCG Experiment Integration and Support
<i>Document status:</i>	DRAFT
<i>Author(s):</i>	Patricia Méndez Lorenzo, Flavia Donno, Antonio Delgado Peris
<i>File:</i>	UserScenario2

Abstract: In this document we describe in a practical way, the steps a user has to follow in order to send and run jobs on LCG-2 Grid and to retrieve and process successfully the output.



CONTENTS

1. INTRODUCTION.....	3
2. THE USER IS JUST LOGGED INTO HIS UI; FIRST THINGS TO DO.	4
2.1. TRANSFERRING THE CERTIFICATE ON THE UI.....	4
2.2. GETTING A PROXY CERTIFICATE.....	5
3. MAKING THE USER’S SOFTWARE AVAILABLE ON THE GRID.....	7
4. REGISTERING PRELIMINARY DATA FILES ON THE GRID.....	8
5. CHECKING THE STATUS OF THE GRID	9
6. CREATING THE USER’S JOB.....	11
6.1. THE JDL LANGUAGE.....	11
7. SENDING THE JOB TO THE GRID.....	13
7.1. THE JOB IS INSIDE GRID.....	13
8. ACCESSING THE DATA	16
8.1. COPING AND REGISTERING A DATA FILE FROM A WN TO A SE	16
8.2. LAST PART OF THE SCENARIO: THE USER REGISTERS HIS OUTPUT FILE ON THE GRID.....	18
9. BEFORE LEAVING THE GRID	19



1. INTRODUCTION

This *User Scenario* is intended to provide HEP users with a guide which explains the steps to be followed in order to submit, run and execute jobs on the LCG-2 Grid. This guide answers the *How* and the *What*: How the user should proceed in order to make his job run on Grid and What happens to the job on the Grid once it has been submitted

We consider the case of a user who wants to compile and run his job on Grid. For this purpose he may need software (i.e., compilers, generators, application sources...) as well as concrete data (such as information about the experiment, results, input parameters and so on). The job output may come as histograms, ntuples, calculations, output files etc. The following sections describe the steps to be done in order to perform production and analysis on LCG-2 Grid.

In this manual, we assume that the user has already obtained his X.509 certificate to have access to the Grid, that he belongs to a Virtual Organization (VO), that he is registered with LCG and he has an account on a User Interface (UI). A description on the above steps can be found in section X.X of the LCG-2 User Guide [la referencia]. We therefore take as initial point the moment when the job is first created and submitted for execution.

The **User Scenario** presented in this manual will be the following: a user of the ALICE experiment wants to generate an ALICE simulated event on the Grid. He will need the following:

- Input parameters (number of hits in each sensitive region of the detector, position and momentum of the particles, vertex positions...) which are recorded in some ASCII files.
- HEP libraries, as supporting software for the task.
- I/O and processing software (such as PAW, ROOT...)
- Visualization programs (to display events).

After finishing his simulation, he will produce some files which can contain data or histograms. These are files of interest for the whole ALICE Collaboration, so he will register this output in order to be used in the future by other people.

The second part of the scenario foresees a second ALICE collaborator who will handle this data in a second job. This second user will access this data and will work with it.

This document will not give details concerning Grid concepts or definitions. All these details and concepts are explained in depth in the User Guide. We encourage the user to read it carefully in order to have a complete view of the grid functionality.



2. THE USER IS JUST LOGGED INTO HIS UI; FIRST THINGS TO DO.

Here we assume that the Grid user has just logged on a UI. This UI will be his work station, his input gate to the Grid. From this machine he will send his jobs, replicate files, have access to data, retrieve output, etc. This machine, however, is not a “*special-use-only Grid computer*”. If scripts or programs need to be checked before submitting them to the Grid (something that at the beginning can be useful), the UI can be used to compile and to run them as on a common Unix station.

Subsections 2.1 and 2.2 will give details about importing the user’s certificate on the UI and getting a proxy.

2.1. TRANSFERRING THE CERTIFICATE ON THE UI

First of all the user has to import his certificates on this UI. This is done by coping the user certificate and key files *usercert.pem* and *userkey.pem* into the `$HOME/.globus` directory (created previously if non existing).

Checking the Certificate

It is possible to check whether a certificate is correct or has any problem. This can be done using following commands:

- To print information about a certificate:

```
% grid-cert-info
```

If the certificate is properly installed in the `$HOME/.globus` directory of the user account on the UI, information regarding the certificate output related to his certification will appear on the screen. For example the subject of the certificate such as in the following example:

```
Subject: C=CH, O=CERN, OU=GRID, CN=Patricia Mendez Lorenzo 3183
```

This command accepts many options which can be seen using the option `-help`.

- To verify a user certificate, just type:

```
% openssl verify -CApath /etc/grid-security/certificates  
/.globus/usercert.pem
```

and if the certificate is valid, the output will be:

```
/home/pmendez/.globus/usercert.pem: OK
```

If the certificate of the CA is not found in `-CApath`, an error message will appear saying:

```
usercert.pem: /O=Grid/O=CERN/OU=cern.ch/CN=Patricia Mendez Lorenzo  
error 20 at 0 depth lookup: unable to get local issuer certificate
```



2.2. GETTING A PROXY CERTIFICATE

Before interacting with the Grid, the user will need a proxy certificate. This proxy is his credential which authenticates him/her in every secure interaction on Grid and has a temporary lifetime.

* In order to create a proxy:

```
% grid-proxy-init
```

– If the command is successful, the output is as follows:

```
Your identity: /O=Grid/O=CERN/OU=cern.ch/CN=Patricia Mendez Lorenzo
Enter Grid pass phase for this identity:(the user introduces his password)
Creating proxy .....Done
Your proxy is valid until: Tue Jun 24 23:48:44 2003
```

– In case of success, the proxy certificate is created in /tmp, unless the environment variable X509_USER_PROXY is defined (e.g. X509_USER_PROXY=\$HOME/.globus/proxy). In the last case the proxy with that file name will be created, if possible. By default, the proxy has a life time of 12 h. Nevertheless, the user can specify its duration with the option -valid H:M. (It is important to take into account, that long living proxies increase security risks). When a proxy expires, a new one has to be created to continue further works.

* To get information about the proxy:

```
% grid-proxy-info
```

– If the command is successful, it prints something like the following:

```
subject : /C=CH/O=CERN/OU=GRID/CN=Patricia Mendez Lorenzo 3183/CN=proxy
issuer  : /C=CH/O=CERN/OU=GRID/CN=Patricia Mendez Lorenzo 3183
type    : full
strength : 512 bits
path    : /tmp/x509up_u1654
timeleft : 9:12:02
```

The generated proxy is used by subsequent Grid commands/requests and it is “shipped” to the proper Grid services. But what happens if the user is running a long job and his proxy expires before the end of his job?. The job will be aborted. To avoid such problem, it is possible to renew the proxy certificate for a longer time through the proxy credential repository system.

Users are encouraged to look up the following web page to see what are the Proxy servers available for their VO:

<https://goc.grid-support.ac.uk/gridsite/db/index.php>

To access this page, the user will need a certificate which can be obtain writting to Matt Thorpe to the following email address:



m.thorpe@rl.ac.uk

- * In the case the user needs the proxy to be automatically renewed, he has to register his proxy with a proxy server, with the command:

```
% myproxy-init -s <host_name> -d -n
```

And the output generated is similar to that obtained with a common proxy:

```
Your identity: /O=Grid/O=CERN/OU=cern.ch/CN=Patricia Mendez Lorenzo
Enter Grid pass phase for this identity:(the user introduces his password)
Creating proxy .....Done
Your proxy is valid until: Thu Jul 17 18:57:04 2003
A proxy valid for 168 hours (7.0 days) for user
/O=Grid/O=CERN/OU=cern.ch/CN=Patricia Mendez Lorenzo now exists on
lxshare0207.cern.ch.
```

Where `<host_name>` is the host name of the machine where a Proxy Server runs. In the case that the hostname Proxy Server is wrong, or the service is unavailable, the proxy will still be created, but it will be a standard 12 h life proxy.

- * One can retrieve information about the long-time proxy, using a syntax similar to that of standard proxies:

```
% myproxy-info -s <host_name> -d
```

The corresponding output, is the same to that obtained with the standard proxy commands.



3. MAKING THE USER'S SOFTWARE AVAILABLE ON THE GRID

Each VO declares one (or more) of their members as software manager(s). The certificate of this person (people) will be mapped to local special grid accounts known as software grid manager (sgm) accounts per VO (for example, alicesgm for Alice, dteamsgm for dteam, etc). Each site has foreseen an area per VO which can be shared between all the WNs of the site. Therefore a VO user should find needed software already installed at a site.

LCG provides two tools for installing experiment software: *lcg-ManageSoftware* and *lcg-ManageVOTag*. While the first tool installs the software into the WNs, the second tool publishes a tag corresponding to the version of the installed software. Both tools can be used only by sgm people. While installing the software with the *lcg-ManageSoftware* tool, the tag publication is automatically done, since it includes *lcg-ManageVOTag*. The name of the tag is provided by the user as an argument of *lcg-ManageSoftware*. Otherwise if the user wants to install his software with his own tools, he can still use *lcg-ManageVOTag* to publish the corresponding software tag. Through *lcg-ManageVOTag*, the user can add, list and delete software tags. The use of the add and delete features is restricted to sgm people. However any user of any VO can know the tags already published using the list feature of this tool.



4. REGISTERING PRELIMINARY DATA FILES ON THE GRID

While the user is running his job, he may need some data files. These files can be temporary and used for the life of his job only or they can be registered and stored on the Grid to be used by other Grid users or by other subsequent jobs. These two cases are treated in detail in sections 7 and 8. Section 7 explains the job submission, while section 8 concentrates on the registration and storage of data files on Grid.



5. CHECKING THE STATUS OF THE GRID

- Before sending a job for execution on LCG-2 user might want to check the status of Grid resources accepting a certain VO.

In order to cover some user requests, LCG-2 has just included a new tool to see the status of the LCG2 resources. At this moment this tool called **lcg-infosites** is provided on UIs as an information provider for the user. For example, in the case the user is interested on the CPUs status of his VO, he can make the following:

```
% lcg-infosites --vo alice ce --is lxb2006
```

In this command the user has specified the VO, the option (ce) and the name of the BDII node he wants to query to. This last argument is not mandatory. In the case the user does not specify it, the default BDII will be queried.

The output of the command below is the following:

```
*****
These are the related data for alice: (in terms of CPUs)
*****
```

#CPU	Free	Total Jobs	Running	Waiting	ComputingElement
18	11	0	0	0	lxt03.jinr.ru:2119/jobmanager-pbs-long
40	40	0	0	0	ce.prd.hp.com:2119/jobmanager-lcgpbs-long
18	11	1	1	0	lxt03.jinr.ru:2119/jobmanager-pbs-short
59	0	8	0	8	cclcgceli01.in2p3.fr:2119/jobmanager-bqs-A
59	58	0	0	0	cclcgceli01.in2p3.fr:2119/jobmanager-bqs-G
97	0	98	97	1	cclcgceli01.in2p3.fr:2119/jobmanager-bqs-T
40	40	0	0	0	ce.prd.hp.com:2119/jobmanager-lcgpbs-alice
40	40	0	0	0	ce.prd.hp.com:2119/jobmanager-lcgpbs-short
670	175	0	0	0	gridkap01.fzk.de:2119/jobmanager-pbspro-long
956	186	22	22	0	lxn1184.cern.ch:2119/jobmanager-lcglsf-grid

[.....]

```
-----
The total values are:
```

```
-----
8195    4169    640    473    167
```

For additional features of this command, please refer to the User Guide.

- The user can know which Grid resources match with his jdl using the following command:



```
% edg-job-list-match --vo = alice myscript.jdl
```

The system will look for the best resources to run this job and it will give the user as output:

```
*****  
COMPUTING ELEMENT IDs LIST
```

The following CE(s) matching your job requirements have been found:

```
*CEId*
```

```
adc0029.cern.ch:2119/jobmanager-lcgpbs-infinite  
adc0029.cern.ch:2119/jobmanager-lcgpbs-long  
adc0029.cern.ch:2119/jobmanager-lcgpbs-short  
adc0037.cern.ch:2119/jobmanager-pbs-infinite  
adc0037.cern.ch:2119/jobmanager-pbs-long  
adc0037.cern.ch:2119/jobmanager-pbs-short
```

```
*****
```

6. CREATING THE USER'S JOB

The user is ready to submit a job but first he needs to describe the requirements for it. He will give such description in a file that is written with the so called Job Description Language (JDL). This file containing his job description will be shipped to Grid. The next subsection gives a complete example.

6.1. THE JDL LANGUAGE

We here explain the content of a JDL file based on a real Alice example presented in the User Scenario section of this manual:

```

Executable          =  ``/bin/sh``;
StdOutput           =  ``aliroot.out``;
StdError            =  ``aliroot.err``;
InputSandbox        =  { ``start_aliroot.sh``, ``rootrc``, ``grun.C``,
                        ``Config.C`` };
OutputSandbox       =  { ``aliroot.out``, ``aliroot.err``, ``galice.root`` };
RetryCount          =  7;
InputData           =  { ``\fn:ALICE-hits`` };
DataAccessProtocol =  { ``file``, ``gsifpt``, ``rfio`` };
Arguments           =  ``start_aliroot.sh 3.02.04 3.07.01``;
Environments        =  { ``ALICE_ROOT_DIR= $ALICE_ROOT_DIR``,
                        ``ROOT_ALICE = $ALICE_ROOT_DIR/root/`` };
Requirements        =  Member( ``ALICE-3.07.01``, other.
                            GlueHostApplicationSoftwareRunTimeEnvironment);
                    && Member( ``ROOT``, other.
                            GlueHostApplicationSoftwareRunTimeEnvironment);
Rank                =  1000*(other.GlueCEInfoTotalCPUs -
                            other.GlueCEStateWaitingJobs)
                    / other.GlueCEInfoTotalCPUs;

```

- Executable (mandatory) specifies to the name of the executable, whether it is a script or a binary file. If the executable is not in the current working directory, the corresponding absolute path must be written here.
- StdOutput (optional) specifies the name of the file where stdout should be redirected to. This file can be retrieve via the OutputSandbox described later on.
- StdError (optional) specifies the name of the file stderr should be redirected to.
- InputSandbox (optional) specifies the list of files that need to be packed and shipped with the JDL from the user's UI to the Grid to be processed together with the job, including the executable.
- OutputSandbox (optional) lists the files that the user wants to get back after job completion. He may want to retrieve the standard error/output and possibly the output files with the results of his. To be noticed that Input and Output Sandbox should never exceed 20-50MB size.



-
- `RetryCount` (optional) gives the number of retries for resubmission in case of Grid failures. If this value is 0, no retries will occur.
 - `InputData` (optional) gives the data files already registered with the Grid that the job must access during execution. User must specify explicitly an lfn or a GUID identifier.
 - `DataAccessControl` (optional) specifies which protocols the user application will use to access the files given in the `InputData` argument. This argument is mandatory when `InputData` is present.
 - `Arguments` (optional) is used to pass arguments to the executable.
 - `Environments` (optional) allow users to modify the environment of the job.
 - `Requirements` (optional) force the system to choose resources that satisfy the conditions imposed by the user. In this example the system to run the job at sites where the ALICE-3.07.01 software and the ROOT package are installed. Many conditions can be imposed. For details refer to the LCG-2 User Guide.
 - `Rank` (optional) is a user-definable measurement of the CE goodness. The job will be sent to the CE with the best rank, among all CEs satisfying all job requirements and having the maximum number of file replicas on a SE close to them.



7. SENDING THE JOB TO THE GRID

At this point we are ready to send the job for execution to the Grid. Actually we do not know where the job is going to run and if there are resources matching the user requirements. For complice jobs for which computing resources need to access data files served by data servers, this information can be quite important. Data management on Grid will be treated in detail in section 8.

7.1. THE JOB IS INSIDE GRID

Once we have checked the status of resources (CEs and SEs) in LCG, found the data files, etc, we are ready to submit the job for execution on the Grid.

- We submit the job with this command:

```
% edg-job-submit --vo=<vo_name> -o jobIDfile myjob.jdl
```

We specify the VO and the -o option to create in the current directory a file called jobIDfile that contains the jobID. We avoid in this way to work with the non-human readable jobID.

If the submission is successful, the user gets something like the following:

```
Connecting to host lxshare0219.cern.ch, port 7772
```

```
Logging to host lxshare0219.cern.ch, port 9002
```

```
*****
```

```
JOB SUBMIT OUTCOME
```

```
The job has been successfully submitted to the Network Server.
```

```
Use edg-job-status command to check job current status. Your job identifier (edg_jobId) :
```

```
- https://lxshare0219.cern.ch:9000/2A30F5hIrUGse0GBFpSSQQ
```

```
*****
```

This <https://lxshare0219.cern.ch:9000/2A30F5hIrUGse0GBFpSSQQ> is the user's JobID which is unique to each job.

- He can check the current status of his job while running on the grid with the command:

```
% edg-job-status --vo=<vo_name> -i jobIDfile
```

Now the jobIDfile is already created and can be specified as input for the command edg-job-status. The output will be something like this:

```
*****
```

```
BOOKKEEPING INFORMATION:
```

```
Printing status info for the Job : https://lxshare0219.cern.ch:9000/
```

```
2A30F5hIrUGse0GBFpSSQQ
```

```
Current Status:    Scheduled
```

```
Status Reason:    Job successfully submitted to Globus
```



```
Destination:      adc0029.cern.ch:2119/jobmanager-lcgpbs-short
reached on:      Wed Nov 11 18:01:31 2004
*****
```

When the job is over, the Current Status will be DONE. If everything was fine the Current Status is: DONE (Success) and Exit Code: 0, if some problem occurred the Exit Code is different than 0 and the user can check what happened retrieving the stdout and the stderr via the outputsandbox mechanism.

- At this point we are going to retrieve the output of the job, in our case the files: `std.out` and `std.err`:

```
% edg-job-get-output --dir <file_name> -i jobIDfile
```

The option `--dir` tells the system which is the path where to put the output. If nothing specified this command will retrieve the output sandbox in a subdirectory of `/tmp` named after the JobID.

The message is:

```
Retrieving files from host lxshare0219.cern.ch
*****

                                JOB GET OUTPUT OUTCOME

Output sandbox files for the job:
- https://lxshare0219.cern.ch:9000/2A30F5hIrUGse0GBFpSSQQ
have been successfully retrieved and stored in the directory:
/home/pmendez/2A30F5hIrUGse0GBFpSSQQ
*****
```

in our case the command entered is:

```
% edg-job-get-output --dir . -i jobIDfile
```

Now the user has in his HOME directory a subdirectory called: `2A30F5hIrUGse0GBFpSSQQ`. Here we can find the three output files. The file `alroot.out` contains the standard outputs of the job; `galice.root`, contains the specific output (in terms of histograms, data files...) while the `alroot.err` file is empty since we did not have any error during the job execution.

- The user can get the logging information of his job, with the following command:

```
% edg-job-logging-info https://lxshare0219.cern.ch:9000/
2A30F5hIrUGse0GBFpSSQQ
```

The command above gives details about the job from submission time until the output is retrieved.



Some Final Considerations

The aim of this section was to explain the Jdl language and the job management commands in LCG-2. The management of data files is a central issue of the Grid system and we will talk about in section xxx. At this point we will like to stress that in this section the user has already handled local files not published in Grid and he has managed to “move” them to Grid for processing.

- Using the `InputSandBox` argument, input files, stored on the UI, were moved to the CE where the job ran.
- A `jobIDfile` containing the jobID of the job has been created.
- At the end of the job, the user has retrieved the output files, including the results of his job from the CE to his UI.

In the next section we will concentrate on grid files that are already registered on the system or that the user wants to register for future jobs.



8. ACCESSING THE DATA

In section 6.1, in the JDL file for the job, we specified two attributes, `InputData` and `DataAccessProtocol`, to tell the system which data files the user needs to run his job and which kind of protocol his application would use to access them. We are therefore dealing with data management on Grid. Now we show how to find data files already registered in the system, replicate them where we run our job in case they are on another SE and to say explicitly to the system which protocol we use to access to them.

The jdl example is:

```
InputData          = { '\lfn:ALICE-hits' };
DataAccessProtocol = { 'file', 'gsiftp' };
```

With these two attributes, the user is telling the system: “I need the “ALICE-hits file”. This name is one of its logical file names (lfn), (To be noticed that the user is not specifying the location of the input file but he is just saying: “I need it”. The system will find it for him. and He wants to access it through the supported protocols: “rfio” and “gsiftp”.

gsiftp: the GSI version of FTP. The application will access the file via GSIFTP, for instance copying the file first on a locally accessible storage.

rfio: a CERN’s remote file access protocol, which allows the user to read and write files remotely, but only within a local area network.

NOTES:

- The protocol “file”, available with LCG-1 is not longer supported in LCG-2.
- rfio is supported at some LCG-2 sites. For more info refer to the LCG-2 User Guide.
- All “edg-rm” commands to manage the data supported by LCG-1 are also available in LCG-2. However the “lcg-utils” set of commands and APIs are also deployed in LCG-2. Although edg-rm tools are still available, the support will soon be dropped. In this manual we will therefore explain the data management in terms of lcg-utils commands. A complete set of edg-rm examples can be found in the User Scenario for LCG-1.

8.1. COPING AND REGISTERING A DATA FILE FROM A WN TO A SE

In this section we will illustrate how to replicate a data file from one SE where it is stored to the SE close to the CE where the user will run his job.

- Let’s suppose the lfn of the data file is known. In order to know all physical replicas associated to that lfn we can issue the command:

```
lcg-lr --vo alice lfn:ALICE-hits
```

In case that the file is properly registered in the VO Grid file catalogue, the following output is displayed:

```
sfn://se01.gridpp.shef.ac.uk/data/alice/generated/2004-08-23/
file1b05cf08-e99b-4 8d0-ae9a-8edce930a7fc
```



In this way the user will retrieve the file SURL (site URL) which contains the hostname of the SE where the file is stored, its directory and possible the GUID corresponding to this lfn.

- Now we know the physical location of the data file but we still do not know how to access it using the “gsiftp” or “rfio” protocols. This is done using the `getTurl` command, specifying the VO, the SURL of the file and the `DataAccessProtocol` as follows:

```
lcg-gt sfn://se01.gridpp.shef.ac.uk/data/alice/generated/2004-08-23/  
file1b05cf08-e99b-4 8d0-ae9a-8edce930a7fc gsiftp (or) rfio
```

The output of this command is (in case of “gsiftp” protocol) will be:

```
gsiftp://se01.gridpp.shef.ac.uk/data/alice/generated/2004-08-23/  
file1b05cf08-e99b-48d0-ae9a-8edce930a7fc
```

And this is the string that the user has to use in his job in order to access the data. For example, the user could use the command `globus-url-copy` to copy files between two Grid resources and from/to any non-Grid resource. Its functionality is quite similar to `lcg-cp` but in this case, this command does not register or unregister files. It means the files could be “ghosts” in the Grid taking a certain space but not being registered in the catalog. This is a very low level tool and use it only in case of real necessity. The `globus-url-copy` command arguments are URLs and the output of `lcg-gt` can be used now as argument:

```
% globus-url-copy gsiftp://se01.gridpp.shef.ac.uk/data/alice/generated/2004-08-23/  
file1b05cf08-e99b-48d0-ae9a-8edce930a7fc  
file://`pwd`/my_file
```

In what follows we show how to copy the file already registered in the Grid using the `gridftp` protocol to a local directory on the WN and how to access it locally.

```
#!/usr/bin/perl  
system "lcg-cp --vo alice gsiftp://se01.gridpp.shef.ac.uk/data/alice/  
generated/2004-08-23/ file1b05cf08-e99b-48d0-ae9a-8edce930a7fc  
file:$WORK_DIR/mydata";  
open file "$WORK_DIR/mydata";
```

The `lcg-cp` command:

```
lcg-cp --vo alice gsiftp://<host and directory which contains the file>  
<destination>
```

Then, the user can open the file inside his code, just directly with the path and once he has opened it, he can handle the data contained inside as he prefers.



8.2. LAST PART OF THE SCENARIO: THE USER REGISTERS HIS OUTPUT FILE ON THE GRID

The output of the user's job can be a big data file which may be used in the future by other people belonging to the same VO. Therefore it might be convenient to copy and register this data on Grid. The output of his job can be created on the WN and then copied and registered on the SE or can be created directly on the close SE and registered with the Grid for further use.

- This can be done within a job using the following:

```
lcg-cr --vo alice file://'/pwd'/galice.root  
-l lfn:alice-event -d cclcgseli02.in2p3.fr;
```

The `lcg-cr` command is used in this case. With this command we copy and register the file created on the WN to the SE at the same time. We specify the following information:

- `file://'/pwd'/galice.root` specifies the source directory of the file. It is the current directory of the WN where the job runs.
- `-l lfn:alice-event`. We give a lfn label for the file.
- `-d cclcgseli02.in2p3.fr`. We give the destination host where the file will be stored. The information about a close SE where a file can be stored can be retrieved with the `lcg-infosites` command explained before.

NOTE: `edg-rm` commands did not force the user to give a destination while copying and registering files. In case no destination was specified, a default SE was used. However the concept of default SE has disappeared in LCG-2. The user must therefore specify a destination for his output. LCG-2 however provides an environment variable (starting from release LCG-2.2.0) set by the system manager at each site which specifies a default SE per VO at that site. The user can therefore use the following:

```
lcg-cr --vo alice file://'/pwd'/galice.root  
-l lfn:alice-event -d $VO_ALICE_DEFAULT_SE;
```

to send the output file to the default SE defined at that site for his VO.

Now the user's file has been registered on the Grid and in the future, he or any other user will be able to access it.



9. BEFORE LEAVING THE GRID

* A user should destroy any existing proxy:

```
% grid-proxy-destroy
```

* In the case of a long-life proxy

```
% myproxy-destroy -s <host_name> -d
```