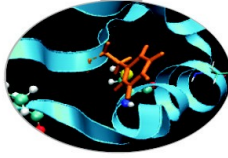


# Introduzione ambiente unix: moduli, job scripts, PBS

A. Grottesi (CINECA)



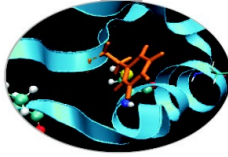
# Today you will learn...



- Basic commands for UNIX environment @ CINECA
- How to submit your job to the PBS queueing system on Eurora
- Tutorial #1:
  - Example: launch a small script to the PBS queueing system



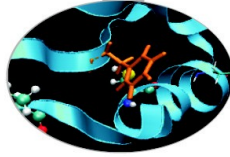
# How to become a CINECA user



- Please fill out the form on:  
<https://userdb.hpc.cineca.it/user/register>
- You'll receive userdb credentials: Then
  - Click on “HPC Access” and follow the on-screen instructions
  - You'll be asked to upload an image of a valid ID document
  - Ask your PI or send an email to [superc@cineca.it](mailto:superc@cineca.it) to be included on an active project.
- When everything is done an automatic procedure sends you (via 2 separate emails) the username/password to access HPC systems



# How to log in



- Establish a ssh connection

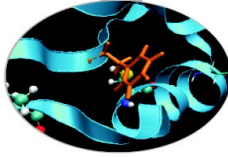
**ssh <username>@login.eurora.cineca.it**

- Remarks:
  - **ssh** available on all linux distros
  - **Putty** (free) or **Tectia** ssh on Windows
  - *secure shell plugin* for **Google Chrome!**
  - important messages can be found in the *message of the day*

Check the **user guide!** <http://www.hpc.cineca.it/content/documentation>



# Storage and Filesystem



## **\$HOME:**

- Permanent, backed-up, and local.
- Quota = 5GB.
- For source code or important input files.

## **\$CINECA\_SCRATCH:**

- Large, parallel filesystem (GPFS).
- Temporary (files older than 30 days automatically deleted), no backup.
- No quota. A cleaning procedure for files older than 30 days

## **\$WORK :**

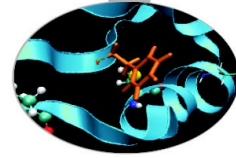
- Permanent, backed-up, project specific, 1 Tb quota by default.

## **More info:**

<http://www.hpc.cineca.it/content/data-storage-and-filesystems-0>



# Accounting: saldo



```
[mcestari@node342] (~)
```

```
$ saldo -b
```

```
-----
```

account	start	end	total (local h)	localCluster Consumed(local h)	totConsumed (local h)	totConsumed %
try11_test	20110301	20111201	10000	0	2	0.0
cin_staff	20110323	20200323	200000000	64581	6689593	3.3
<b>ArpaP_prod</b>	<b>20130130</b>	<b>20131101</b>	<b>1500000</b>	<b>0</b>	<b>0</b>	<b>0.0</b>

```
-----
```

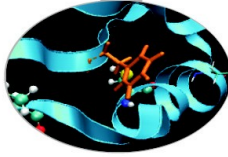
Accounting philosophy is based on the resources requested for the time of the batch job:

$$\text{cost} = \text{no. of cores requested} \times \text{job duration}$$

In the CINECA system it is possible to have more than 1 budget (“account”) from which you can use time. The accounts available to your UNIX username can be found from the `saldo` command.



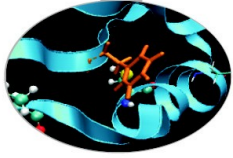
# module, my best friend



- all the optional software on the system is made available through the **"module" system**
  - provides a way to rationalize software and its env variables
- modules are divided in 3 *profiles*
  - **profile/base** (stable and tested modules)
  - **profile/engineering** (contains specific software for engineering simulations)
  - **profile/advanced** (software not yet tested or not well optimized)
- each profile is divided in 4 categories
  - **compilers** (Intel, GNU, Portland)
  - **libraries** (e.g. LAPACK, BLAS, FFTW, ...)
  - **tools** (e.g. Scalasca, GNU make, VNC, ...)
  - **applications** (software for chemistry, physics, ... )



# Modules

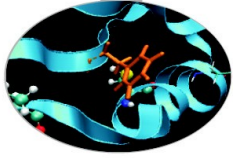


- CINECA's work environment is organized in modules, a set of installed libs, tools and applications available for all users.
- “loading” a module means that a series of (useful) shell environment variables will be set
- E.g. after a module is loaded, an environment variable of the form “<MODULENAME>\_HOME” is set





# Module commands



> **module available** (or just “> module av”)

Shows the full list of the modules available in the profile you're into, divided by: environment, libraries, compilers, tools, applications

> **module (un)load** <module\_name>

(Un)loads a specific module

> **module show** <module\_name>

Shows the environment variables set by a specific module

> **module help** <module\_name>

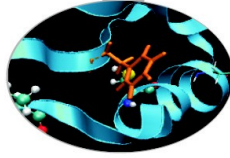
Gets all informations about how to use a specific module

> **module purge**

Gets rid of all the loaded modules



# Launching jobs

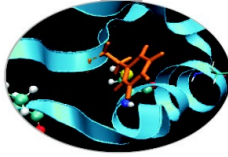


- Now that we have our executable, it's time to learn how to prepare a job for its execution
- Eurora, Pico and Galileo have the **PBS** scheduler.
- The job script scheme is:

```
#!/bin/bash  
#PBS keywords  
variables environment  
execution line
```



# Environment setup and execution line



The execution line starts with mpirun: Given: `./myexe arg_1 arg_2`

`mpirun -n 16 ./myexe arg_1 arg_2`

`-n` is the number of **cores** you want to use

`arg_1 arg_2` are the normal arguments of myexe

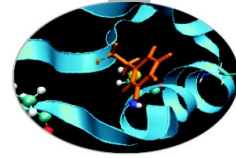
In order to use mpirun, **openmpi** (or **intelmpi**) has to be loaded. Also, if you linked dynamically, you have to remember to load every library module you need (automatically sets the LD\_LIBRARY\_PATH variable).

The environment setting usually starts with “**cd \$PBS\_O\_WORKDIR**”. That’s because by default you are launching on your home space the executable may not be found.

`$PBS_O_WORKDIR` points to the directory from where you’re submitting the job .



# PBS keywords



```
#PBS -N jobname
#PBS -o job.out
#PBS -e job.err
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=ngpus=2
#PBS -l walltime=1:00:00
#PBS -q <queue>
#PBS -A <my_account>
```

```
# name of the job
# output file
# error file
# resources
# hh:mm:ss
# chosen queue
# name of the account
```

**select** = number of chunk requested

**ncpus** = number of cpus per chunk requested

**mpiprocs** = number of mpi tasks per node/chunk

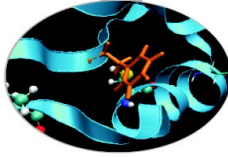
**mem** = RAM memory per chunk

**ngpus** = number of CUDA devices per node

**nmics** = numer of Intel-Phi devices per node



# PBS job script template



```
#!/bin/bash
#PBS -l walltime=2:00:00
#PBS -l select=1:ncpus=16:mpiprocs=16:ngpus=2:mem=14GB
#PBS -o job.out
#PBS -e job.err
#PBS -q parallel
#PBS -A <account_no>
#PBS -m mail_events ==> specify email notification
(a=aborted,b=begin,e=end,n=no_mail)

#PBS -M user@email.com

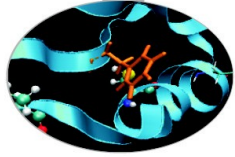
cd $PBS_O_WORKDIR

module load autoloader intelmpi/openmpi
module load somelibrary

mpirun ./myprogram < myinput
```



# PBS commands



## qsub

```
qsub <job_script>
```

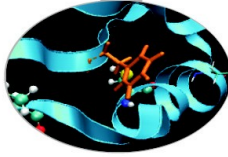
Your job will be submitted to the PBS scheduler and executed when there will be nodes available (according to your priority and the queue you requested)

## qstat

```
qstat -a
```

Shows the list of all your scheduled jobs, along with their status (idle, running, closing, ...) Also, shows you the job id required for other qstat options

# PBS commands



## qstat

```
qstat -f <job_id>
```

Provides a long list of informations for the job requested.

In particular, if your job isn't running yet, you'll be notified about its estimated start time or, if you made an error on the job script, you will learn that the job won't ever start

## qdel

```
qdel <job_id>
```

Removes the job from the scheduled jobs by killing it

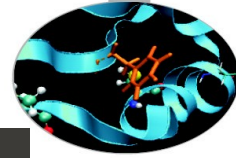
## qalter

```
qalter -l <resources> <job_id>
```

Alter one or more attributes of one or more PBS batch jobs.



# PBS commands: qstat



```

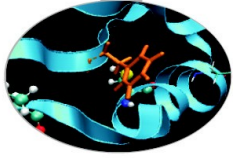
File Edit View Search Terminal Help

node129:

Job ID          Username Queue      Jobname      SessID NDS  TSK  Req'd  Req'd  Est
              Memory   Time   S        Time
-----
451912.node129  amessina parallel  abq_parall   --     4   32   4gb   00:10  H   --
1587463.node129 aemerson  parallel  qsub.job     --    12  192  120gb 00:30  H   --
1587543.node129 aemerson  parallel  qsub.job     --     9  144   90gb 00:30  H   --
1596264.node129 ggrazios  parallel  a7_nor_wat  113134  1   16   14gb 04:00  R   --
1596269.node129 dborello  parallel  sub.sh       26304  16  64   16gb 04:00  R   --
1596279.node129 adimasci  parallel  Xnavis95_c  83686  1   1    2gb 04:00  R   --
1596277.node129 adimasci  parallel  Xnavis95_c  87830  1   1    2gb 04:00  R   --
1596278.node129 adimasci  parallel  Xnavis95_c  101923 1   1    2gb 04:00  R   --
1596287.node129 gagate00  parallel  esegui_8SP  19442  1  16   14gb 04:00  R   --
1596304.node129 mrizzini  parallel  ipcOptimal  90425  1  12    9gb 04:00  R   --
1596305.node129 mrizzini  parallel  ipcOptimal  72089  1  12    9gb 04:00  R   --
1596306.node129 mrizzini  parallel  ipcOptimal  71042  1  12    9gb 04:00  R   --
1596307.node129 mrizzini  parallel  ipcOptimal   8235  1  12    9gb 04:00  R   --
1596308.node129 mrizzini  parallel  ipcOptimal  102900 1  12    9gb 04:00  R   --
1596309.node129 mrizzini  parallel  ipcOptimal  42979  1  12    9gb 04:00  R   --
1596310.node129 mrizzini  parallel  ipcOptimal  92927  1  12    9gb 04:00  R   --
1596311.node129 mrizzini  parallel  ipcOptimal  90698  1  12    9gb 04:00  R   --
1596290.node129 gagate00  parallel  esegui_4SP  78531  1  16   14gb 04:00  R   --
1596291.node129 gagate00  parallel  esegui_4SP  37027  1  16   14gb 04:00  R   --
1596292.node129 gagate00  parallel  esegui_2SP  78795  1  16   14gb 04:00  R   --
1596312.node129 mrizzini  parallel  ipcOptimal  105767 1  12    9gb 04:00  R   --
1596313.node129 mrizzini  parallel  ipcOptimal   87469 1  12    9gb 04:00  R   --
[agrottes@node129 ~]$
  
```



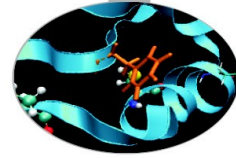




# Scripts for running MD codes on Eurora



# Gromacs 5.0.4, pure MPI on Eurora



```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=16:mpiprocs=16:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd12015
```

```
cd $PBS_O_WORKDIR
```

 ==> change to current dir

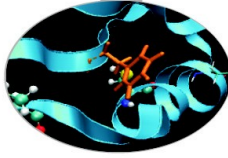
```
module load profile/advanced
module load autoloader gromacs/5.0.4
```

```
export OMP_NUM_THREADS=1
```

 ==> set nr. Of OpenMP threads to 1 per node

```
mdrun=$(which mdrun_mpi)
cmd="$mdrun -s topol.tpr -v -maxh 1.0 -nb cpu"
mpirun -np 16 $cmd
```





```
#!/bin/bash
#PBS -N gmx
#PBS -l select=1:ncpus=2:mpiprocs=2:ngpus=2:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd12015
```

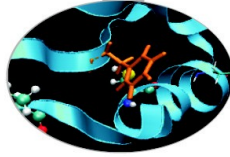
```
cd $PBS_O_WORKDIR           ==> change to current dir
```

```
module load profile/advanced
module load autoload gromacs/5.0.4
```

```
export OMP_NUM_THREADS=1    ==> set nr. Of OpenMP threads to 2 per node
#                             ==> set total mpi tasks = 2 and bind to two GPUs
```

```
mdrun=$(which mdrun_mpi_cuda)
cmd="$mdrun -s topol.tpr -v -maxh 1.0 -gpu_id 01 "
mpirun -np 2 $cmd
```





```
#!/bin/bash
#PBS -N MPI-OpenMP-GPU
#PBS -l select=1:ncpus=16:mpiprocs=2:ngpus=2:mem=14GB
#PBS -q parallel
#PBS -l walltime=1:00:00
#PBS -A train_cmd12015

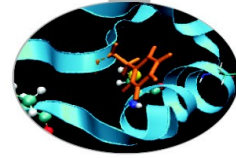
# go to submission dir
cd $PBS_O_WORKDIR

# load gromacs 5.0.4
module load profile/advanced
module load cuda/6.5.14
module load autoload gromacs/5.0.4

# we have asked for 1 nodes = 2 GPUs
# => set total mpi tasks = 2 (2 per node) and set omp tasks to fill up each node
export OMP_NUM_THREADS=8

mdrun=$(which mdrun_mpi_cuda)
cmd="$mdrun -s topol.tpr -deffnm MPI_OpenMP-GPU -v -maxh 1.0 -gpu_id 01"
mpirun -np 2 $cmd
```

# Gromacs 5.0.4 Intel Phi (symmetric mode)



```
#!/bin/bash
#PBS -N MPI-OpenMP-GPU
#PBS -l select=1:ncpus=16:mpiprocs=8:nmics=1:mem=14GB
#PBS -q <queue>
#PBS -l walltime=1:00:00
#PBS -A train_cmd12015
```

```
cd $PBS_O_WORKDIR
```

```
module load intel
module load intelmpi
module load gromacs
module load mkl
```

```
string=$HOSTNAME
MICNAME=${string-mic0}
echo -e "$HOSTNAME:4\n$MICNAME:4" > machinefile
```

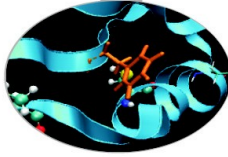
```
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/lib/mic:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/mkl/lib/mic:${LD_LIBRARY_PATH}
export LD_LIBRARY_PATH=/cineca/prod/compilers/intel/cs-xe-2015/binary/tbb/lib/mic:${LD_LIBRARY_PATH}
```

```
export I_MPI_MIC=1
export I_MPI_PIN_MODE=mpd
export MIC_ENV_PREFIX=MIC
export MIC_KMP_AFFINITY=verbose,compact,0 # KMP_AFFINITY for MIC threads
export IVB_KMP_AFFINITY=verbose,compact,1 # KMP_AFFINITY for Host threads
export MIC_OMP_NUM_THREADS=30 # number of OMP threads on MIC
export IVB_OMP_NUM_THREADS=4 # number of OMP threads on Host
```

```
exe="-s topol-10k.tpr -deffnm test -maxh 2.0 -v"
mpirun -n 16 -genvall -machinefile machinefile ./symmetric.sh
```

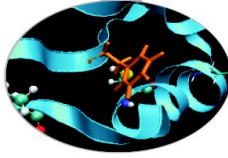


## Native execution of codes on Intel Phi



MIC-native programs need to be executed inside the MIC card itself.  
In order to log into a MIC card you have to:

- login to a MIC node with a PBS interactive session requesting at least 1 mic (nmics=1);
- use the "get\_dev\_list" script (available by loading the "superc" module on Galileo) in order to get the name of the specific MIC card assigned to you.
- get\_dev\_list will produce in output an hostfile named <job\_id>\_dev\_hostfile containing the lists of the assigned cards;
- connect through ssh into the MIC card (in the example node254-mic0)



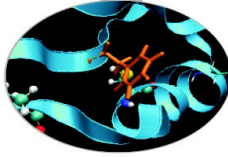
## Example of interactive session

```
> qsub -A <account_name> -I -l select=1:ncpus=1:nmics=1
qsub: waiting for job 10876.io01 to start
qsub: job 10876.io01 ready
...

cd $PBS_O_WORKDIR
module load superc
get_dev_list
cat ${PBS_JOBID}_dev_hostfile
node254-mic0
...

ssh node254-mic0  (*)
...
```

## Native execution of codes on Intel Phi



In order to SSH access the mic card you have to create the public key of the Galileo username in your \$HOME from login node

<http://www.hpc.cineca.it/content/how-connect-public-key>

1. ssh-keygen
2. ls /home/<NAME>/.ssh
3. Copy the id\_dsa.pub file into the \$HOME area of the remote system you want to connect to.
4. cat id\_dsa.pub >> \$HOME/.ssh/authorized\_keys