
MareNostrum User's Guide



Barcelona Supercomputing Center

Copyright © 2012 BSC-CNS

Table of Contents

1. Introduction	1
2. System Overview	2
3. Online Documentation	2
3.1. Resources on the BSC web page	2
3.2. Man pages	2
4. Connecting to MareNostrum	3
4.1. Login nodes	4
4.2. Transferring files	4
4.3. Graphical applications	5
5. File Systems	6
5.1. Root Filesystem	6
5.2. GPFS Filesystem	6
5.3. Local Hard Drive	7
5.4. HSM	7
6. Running Jobs	9
6.1. Classes	9
6.2. Submitting jobs	9
7. Software Environment	13
7.1. C Compilers	13
7.2. FORTRAN Compilers	14
7.3. Optimization	16
7.4. Debuggers	17
7.5. Software in MareNostrum	18
7.6. Numerical Libraries	18
7.7. Modules Environment	19
8. Acknowledgment in publications	21
9. Getting help	21
10. FAQ's	22
A. SSH	24

1. Introduction

This user's guide for the MareNostrum supercomputer is intended to provide the minimum amount of information needed by a new user of this system. As such, it assumes that the user is familiar with many of the standard aspects of supercomputing as the Unix operating system.

We hope you can find most of the information you need to use our computing resources: from applications and libraries to technical documentation about MareNostrum; how to include references in publications and so on. Please read carefully this document and if any doubt arises don't hesitate to contact our support group at <support@bsc.es>

2. System Overview

MareNostrum comprises 2560 JS21 compute nodes (blades) and 42 p615 servers. Every blade has two processors at 2.3 GHz running Linux operating system with 8 GB of memory RAM and 36 GB local disk storage. All the servers provide a total of 280 TB of disk storage accessible from every blade through GPFS (Global Parallel File System).

For additional technical documentation about MareNostrum you can follow this link:

<http://www.bsc.es/marenostrum-support-services/marenostrum-system-architecture/documentation>

The networks that interconnect the MareNostrum are:

- Myrinet Network: High bandwidth network used by parallel applications communications.
- Gigabit Network: Ethernet network used by the blades to mount remotely their root file system from the servers and the network over which GPFS works.

3. Online Documentation

Since system upgrades and enhancements occur on an ongoing basis, it is almost impossible to keep hardcopy documentation up to date. We provide as much information as possible online through the following facilities:

- BSC web page: <http://www.bsc.es>
- UNIX man pages

3.1. Resources on the BSC web page

Most of the basic information to use the MareNostrum system is provided in this document that you are reading at this moment. This document is also available on the BSC web page. You can also find on this document information about compilers, libraries, applications, operating systems, and all other software installed on the machine.

The description of the different hardware components of MareNostrum supercomputer, its function and a technical description of each can be found on the resources web page:

<http://www.bsc.es/marenostrum-support-services/marenostrum-system-architecture>

3.2. Man pages

Information about most commands and software installed is available via the standard UNIX man command. For example, to read about command-name just type on a shell inside MareNostrum:

```
usertest@login1:~> man command-name
```

which displays information about that command to the standard output.

If you don't know the exact name of the command you want but you know the subject matter, you can use the -k flag. For example:

```
usertest@login1:~> man -k compiler
```

This will print out a list of all commands whose man-page definition includes the word 'compiler'. Then you could execute the exact man command line to know about the exact command you were looking for.

Just to know more about the man command itself, you could also type:

```
usertest@login1:~> man man
```

4. Connecting to MareNostrum

Once you have a login and its associated password you can get into MareNostrum system, connecting to one of the next login nodes:

From *mn1.bsc.es* to *mn4.bsc.es*

You must use Secure Shell (ssh) tools to login into or transfer file into MareNostrum. We do not accept incoming connections from protocols as telnet, ftp, rlogin, rcp, or rsh commands. Once you are logged into MareNostrum you cannot make outgoing connections for security reasons.

To get more information about the secure shell version supported and how to get ssh for your system (including windows systems) see Appendix A.

Here you have an example of logging into MareNostrum from a UNIX environment:

```
localsystem$ ssh -l usertest mn1.bsc.es
usertest's password:
+-----+
|                                     |
|               Welcome to MareNostrum               |
|                                     |
| - All home directories are in GPFS and quotas are enabled |
| - Applications are located at /gpfs/apps              |
| - All users are encouraged to change their passwords    |
| - For further information read MareNostrum User Guide:  |
|   http://www.bsc.es/support/MareNostrum-ug.pdf         |
|                                     |
| Please contact support@bsc.es for questions           |
|                                     |
+-----+

usertest@login1:~>
```

As can be seen, you will be prompted for your password before access is granted. If you are on a Windows system, you need to download and install a Secure Shell client to perform the connection to the machine (See appendix A for more information).

Most of these applications are graphical and you will have to fill some information in some of the fields offered, in the field 'Host name' or 'Remote Host name' you will need to introduce: *mn1.bsc.es*. Then, enter your username and password as requested by the Secure shell client. After this procedure you may be logged into MareNostrum.

The first time that you connect to the MareNostrum system secure shell needs to interchange some initial information to establish the communication. This information consists of the acceptance of the RSA key of the remote host, you must answer 'yes' or 'no' to confirm the acceptance of this key.

Please change your initial password after you login the first time into the machine. Also use a strong password (8 characters, do not use a word or phrase from a dictionary and do not use a word that can be obviously tied to your person). Finally, please make a habit of changing your password on a regular basis.

If you cannot get access to the system after following this procedure, first consult Appendix A for an extended information about Secure Shell, or you can contact us, (see section 9 to know how to contact with us).

4.1. Login nodes

Once inside the machine you will be presented with a UNIX shell prompt and you'll normally be in your home (\$HOME) directory. If you are new to UNIX, you'll have to learn the basics before you could do anything useful.

The machine in which you will be logged in will be one of the 12 login nodes of MareNostrum (login1 .. login12). These machines act as front ends, and are used typically for editing, compiling, preparation/submission of batch executions and as a gateway for copying data inside or outside MareNostrum.

It is not permitted the execution of cpu-bound programs on this nodes, if some compilation needs much more cputime than the permitted, this needs to be done through the batch queue system.

It is not possible to connect directly to the compute blades from the login nodes, all resource allocation is done by the batch queue system.

4.2. Transferring files

There are three ways to transfer files to MareNostrum. From Unix systems, you can use secure copy (scp) or secure ftp (sftp), both tools have the same syntax as the old and insecure tools such as rcp (remote copy) and ftp. The third method users can use to transfer data is FTPS.

In order to increase the bandwidth of massive data movements among the machines located at the different RES sites, the use of FTPS (FTP + SSL) is proposed. FTPS protocol provides encrypted authentication, but the data transfer is performed without ciphering, allowing a better network usage by reducing the amount of cpu needed.

- *Setting the FTPS client:*

We propose GFTP as FTPS client, it can be found at /gpfs/apps/FTPS of the different RES Machines.

Users can download it from: <http://gftp.seul.org/> to deploy gftp at their local machines. Gftp has to be compiled with ssl support. (hence, openssl-devel package is required to compile).

Compilation steps should be as follows:

```
./configure --prefix=[your_prefix] --with-ssl
make
make install
```

Depending on the configuration text and graphic versions will be created (gftp-text, gftp). In this guide, we'll focus only in the text-based tool.

Once installed, peer certificate checking has to be disabled by setting the "verify_ssl_peer key" to 0 at ~/.gftp/gftp.rc (verify_ssl_peer=0)

- *Moving data with gftp:*

Sites allowing FTPS transfers will provide one dedicated node where remote users must connect to. Data transfers will always be in pull mode. At MareNostrum (Barcelona) this node is mn1-fts.bsc.es; at other sites, local support teams will provide this node when available.

The gftp session is started as follows.

```
~> gftp-text ftps://[username]@mn1-fts.bsc.es
```

And authenticated with the password you use to access that machine.

Once authenticated, you'll be in an ftp-like environment with access to the different gpfs filesystems (projects,scratch,home,apps), you can navigate through them with ls and cd, and download / upload with get / mget / put / mput commands. Remove operations are restricted in this environment for security reasons.

EXAMPLE: steps needed to transfer /gpfs/projects/bsc99 to the local site:

```
~> gftp-text ftps://[username]@mn1-ftp.bsc.e
gftp> lcd [your path]
gftp> cd /gpfs/projects/bsc99
gftp> mget *
gftp> quit
```

As it have been said before no connections are allowed from inside MareNostrum to the outside world, so all scp and sftp commands have to be executed from your local machines and not inside MareNostrum.

Here there are some examples of each of this tools transferring files to MareNostrum:

```
localsystem$ scp localfile usertest@mn1.bsc.es
usertest's password:
```

```
localsystem$ sftp usertest@mn1.bsc.es
usertest's password:
sftp> put localfile
```

These are the ways to retrieve files from MareNostrum to your local machine:

```
localsystem$ scp usertest@mn1.bsc.es:remotefile localdir
usertest's password:
```

```
localsystem$ sftp usertest@mn1.bsc.es
usertest's password:
sftp> get remotefile
```

On a Windows system, most of the secure shell clients comes with a tool to make secure copies or secure ftp's. There are several tools that accomplishes the requirements, please refer to the Appendix A, where you will find the most common ones and examples of use.

4.3. Graphical applications

You could execute graphical applications from the login nodes, to do that the only way is tunneling all the graphical traffic through the Secure shell connection established.

You will need to have an Xserver running on your local machine to be able to show the graphical information. Most of the UNIX flavors have an X server installed by default. In a Windows environment, you will probably need to download and install some type of X server emulator. (see appendix A)

The second step in order to be able to execute graphical applications is to enable in your secure shell connection the forwarding of the graphical information through the secure channel created. This is normally done adding the '-X' flag to your normal ssh command used to connect to Marenostrum.

Here you have an example:

```
localsystem$ ssh -X -l usertest mn1.bsc.es
```

For Windows systems, you will have to enable the 'X11 forwarding', that option normally resides on the 'Tunneling' or 'Connection' menu of the client configuration window. (See appendix A for further details).

5. File Systems

IMPORTANT: It is your responsibility as a user of the MareNostrum system to backup all your critical data. **We only guarantee a daily backup of user data under /gpfs/home and /gpfs/projects.**

Each user has several areas of disk space for storing files. These areas may have size or time limits, please read carefully all this section to know about the policy of usage of each of these filesystems.

There are 3 different types of storage available inside a node:

- *Root filesystem:* Is the filesystem where the operating system resides
- *GPFS filesystems:* GPFS is a distributed networked filesystem which can be accessed from all the nodes
- *Local hard drive:* Every blade has an internal hard drive

Let's see them in detail.

5.1. Root Filesystem

The root file system, where the operating system is stored doesn't reside in the blade, this is a NFS filesystem mounted from one of the servers. To know in more detail about the structure of the MareNostrum system please consult our website on the Resources link.

As this is a remote filesystem only data from the operating system has to reside in this filesystem. It is NOT permitted the use of /tmp for temporary user data. The local hard drive can be used for this purpose as you could read in section 5.3.

Furthermore, the environment variable \$TMPDIR is already configured to force the normal applications to use the local hard drive to store their temporary files.

5.2. GPFS Filesystem

The IBM General Parallel File System (GPFS) is a high-performance shared-disk file system that can provide fast, reliable data access from all blades of the cluster to a global filesystem. GPFS allows parallel applications simultaneous access to a set of files (even a single file) from any blade that has the GPFS file system mounted while providing a high level of control over all file system operations. These filesystems are the recommended to use with most jobs, because GPFS provides high-performance I/O by "striping" blocks of data from individual files across multiple disks on multiple storage devices and reading/writing these blocks in parallel. In addition, GPFS can read or write large blocks of data in a single I/O operation, thereby minimizing overhead.

An incremental backup will be performed dailys only in the /gpfs/home and /gpfs/projects (not in /gpfs/scratch).

These are the GPFS filesystems available in MareNostrum from all blades:

/gpfs/home: This filesystem has the home directories of all the users, when you log into MareNostrum you start in your home directory by default. Every user will have their own home directory to store the executables, own developed sources and their personal data. Quotas are in effect that limit the amount of data that can be saved here, a default quota will be enforced to all users.

The quota and the usage of space can be consulted via the quota command:

```
usertest@login1:~> quota -v
```

If you need more disk space in this filesystem or in any other of the GPFS filesystems, the responsible of your project has to make a request for this extra space needed, specifying the requested space and the reasons why it is needed. The request can be sent by email or any other way of contact to the user support team as it is explained in section 9 of this document.

/gpfs/projects: In addition to the home directory, there is a directory in */gpfs/projects* for each group of users of Marenostrum. For instance, the group bsc01 will have a */gpfs/projects/bsc01* directory ready to use. This space is intended to store data that needs to be shared between the users of the same group or project. A quota per group will be enforced depending on the space assigned by Access Comitee.

You can query the status of the space used in */gpfs/projects* via the *quota* command, where you will have to specify the group that you belong to:

```
usertest@login1:~> quota -v -g <GROUP>
```

All the users of the same project will share their common */gpfs/projects* space and it is responsibility of each project manager to determine and coordinate the better use of this space, and how it is distributed or shared between their users. If a project needs more disk space in this filesystem or in any other of the GPFS filesystems, the project manager has to make a request for this extra space needed, specifying the space needed and the reasons why it is needed. The request can be sent by email or any other way of contact to the user support team as it is explained in section 9.

/gpfs/scratch: Each MareNostrum user will have a directory over */gpfs/scratch*, you must use this space to store temporary files of your jobs during its execution. By default, files may reside for up to 7 days without modification in this filesystem, any older file might be removed. A quota per group will be enforced depending on the space assigned.

/gpfs/apps: Over this filesystem will reside the applications and libraries that have already been installed on Marenostrum. Take a look at the directories or to section 7 of this document to know the applications available for general use. Before installing any application that is needed by your project, first check if this application is already installed on the system. If some application that you need is not on the system, you will have to ask our user support team to install it. Check section 9 how to contact with us. If it is a general application with no restrictions in his use, this will be installed over a public directory, that is over */gpfs/apps* so all users on MareNostrum could make use of it. If the application needs some type of license and his use must be restricted, a private directory over */gpfs/apps* will be created, so only the required users of MareNostrum could make use of this application. All applications on */gpfs/apps* will be installed, controlled and supervised by the user support team. This doesn't mean that the users could not help in this task, both can work together to get the best result. The user support can provide his wide experience in compiling and optimizing applications in the MareNostrum platform and the users can provide his knowledge of the application to be installed. All that general applications that have been modified in some way from its normal behavior by the project users' for their own study, and may not be suitable for general use, must be installed over */gpfs/projects* or */gpfs/home* depending on the usage scope of the application, but not over */gpfs/apps*.

5.3. Local Hard Drive

Every blade has a local hard drive that can be used as a local scratch space to store temporary files during executions of one of your jobs. This space is mounted over */scratch* directory. The amount of space within the */scratch* filesystem varies from node to node (depending on the total amount of disk space available). All data stored in these local hard drives at the compute blades will not be available from the login nodes. Local hard drive data is not automatically removed, so each job should have to remove its data when finishes.

5.4. HSM

HSM stands for Hierarchical Storage Management, and provides more than 3 PB of total space. It will be mounted from login5 to login8 at */HSM/<group>/*. Under this filesystem all the unused data

is automatically moved to tapes and if some space is needed, the system will remove the unused data because it has been previously saved.

The blocksize for this filesystem is 1 MB, so we strongly recommend to save large files in order to optimize the use of that filesystem. If you need to save a large number of small files, they should be saved in tar format all together.

To check the quotas and the space used in this filesystem you may use `hquota`. It must be executed from one of the logins that have the HSM mounted (logins 5 to 8), so before running it you should jump to one of them:

```
usertest@login4:~> ssh login8
usertest@login8:~> hquota
HSM Quota for group <usertest>

              In use              Soft limit              Hard limit
              -----              -
Number of files:              3              100000              100100
Space on tape:              2.20 GB              95.37 GB              104.90 GB
```

There are some special tools to simplify the transference between filesystems as well. They create a batch job to execute the original command on queues to avoid the cputime limitation in interactive sessions, and they can be run from any node in the cluster:

- *htar*: submits a tar command to queues.

Example 1. Taring data from /gpfs/ to /HSM

```
> htar -cvf /HSM/usertest/outputs.tar \
/gpfs/home/usertest/usertest/OUTPUTS
```

- *hcp*: submits a cp command to queues. Remember to delete the data in the source filesystem once copied to HSM to avoid duplicated data.

Example 2. Copying data from /gpfs to /HSM

```
> hcp -r /gpfs/home/usertest/usertest/OUTPUTS \
/HSM/usertest/
```

Example 3. Copying data from /HSM to /gpfs

```
> hcp -r /HSM/usertest/OUTPUTS \
/gpfs/home/usertest/usertest/
```

- *hmv*: submits a mv command to queues.

Example 4. Moving data from /gpfs to /HSM

```
> hmv /gpfs/home/usertest/usertest/OUTPUTS \
/HSM/usertest/
```

Example 5. Moving data from /HSM to /gpfs

```
> hmv /HSM/usertest/OUTPUTS \
/gpfs/home/usertest/usertest/
```

To check the state of these commands, you can use `mnq`, explained in the next section.

6. Running Jobs

Slurm+MOAB is the utility used at MareNostrum for batch processing support, so all jobs must be run through it. This document provides information for getting started with job execution at MareNostrum.

In order to keep the login nodes in a proper load, a 10 minutes limitation in the cpu time is set for processes running interactively in these nodes. Any execution taking more than this limit should be carried out through the queue system.

6.1. Classes

The user's limits are assigned automatically to each particular user (depending on the resources assigned to the group he or she belongs) and there is no reason to explicitly set the `#@class` directive. Anyway you are allowed to use the special class: "debug" in order to perform some fast short tests. To use the "debug" class you need to include the mentioned directive in your job scripts

Table 1. Classes

<i>Class</i>	<i>Max CPUs</i>	<i>Wall clock time limit</i>
debug	64	10 min
interactive	6	1 h

The specific limits assigned to each user depends on the priority granted to the group. RES users granted with "high priority hours" will have access to a maximum of 1024 CPUs and a maximum `wall_clock_limit` of 72 hours. For RES users with "low priority hours" the limits are 1024 CPUs and 24 hours. BSC users can submit jobs up to 48 hours of `wall_clock_limit`. If you need to increase these limits please contact the support group.

- *debug*: This class is reserved for testing the applications before submitting them to the 'production' queues. Only one job per user is allowed to run simultaneously in this queue, and the execution time will be limited to 10 minutes. Only a limited number of jobs may be running at the same time in this queue.
- *interactive*: This class is reserved for executions which need to be interactive, i.e. a graphical application, and demanding more time than the interactive cpu time limit in the nodes. There is a wall clock limit of 1h for this queue, and it must be used from logins 2 to 4. The best way to use it is through the "mninteractive" command, explained with more detail in the next section.

6.2. Submitting jobs

A job is the execution unit for the SLURM. A job is defined by a text file containing a set of directives describing the job, and the commands to execute.

6.2.1. SLURM wrapper commands

These are the basic directives to submit jobs:

- `mnsbmit <job_script>`

submits a "job script" to the queue system (see below for job script directives).

- `mninteractive <command>`

Submits an interactive job, executing the command or applications specified as argument. The user must be logged into login2 to 4, with the X11 forwarding active. The restrictions of the interactive queue are detailed in the previous section.

- `mnq`

shows all the jobs submitted.

- `mncancel <job_id>`

removes his/her job from the queue system, canceling the execution of the job if it was already running.

- `checkjob <job_id>`

obtains detailed information about a specific job, including the assigned nodes and the possible reasons preventing the job from running.

- `mnstart <job_id>`

shows information about the estimated time for the specified job to be executed.

- `mnhold -j <job_id>`

Sets a block to the specified job. To release a job, the same command must be run with `-r` option.

6.2.2. Job directives

A job must contain a series of directives to inform the batch system about the characteristics of the job. These directives appear as comments in the job script, with the following syntax:

```
# @ directive = value
```

Additionally, the job script may contain a set of commands to execute. If not, an external script must be provided with the 'executable' directive. Here you may find the most common directives:

```
# @ class = class_name
```

The queue where the job is to be submitted. Let this field empty unless you need to use "interactive" or "debug" queues.

```
# @ wall_clock_limit = HH:MM:SS
```

The limit of wall clock time. This is a *mandatory* field and you must set it to a value greater than the real execution time for your application and smaller than the time limits granted to the user. Notice that your job will be killed after the elapsed period

```
# @ initialdir = pathname
```

The working directory of your job (i.e. where the job will run). If not specified, it is the current working directory at the time the job was submitted.

```
# @ error = file
```

The name of the file to collect the stderr output of the job.

```
# @ output = file
```

The name of the file to collect the standard output (stdout) of the job.

```
# @ total_tasks = number
```

The number of processes to start.

```
# @ cpus_per_task = number
```

The number of cpus allocated for each task. This is useful for hybrid MPI+OpenMP applications, where each process will spawn a number of threads. The number of cpus per task must be between 1 and 4, since each node has 4 cpus (one for each thread).

```
# @ tasks_per_node = number
```

The number of tasks allocated in each node. When an application uses more than 1.7 GB of memory per process, it is not possible to have 4 processes in the same node and its 8GB of memory. It can be combined with the `cpus_per_task` to allocate the nodes exclusively, i.e. to allocate 2, processes per node, set both directives to 2. The number of tasks per node must be between 1 and 4.

```
# @ nodeset = clos
```

For executions requiring 1024 or less cpus, it is possible to ask for nodes sharing the same clos (switch Myrinet). This improves the communication performance of MPI applications when experiencing problems with delays in the communications, as the message routing will be faster.

However, this restriction makes more difficult the start of the job, and thus, waiting times in queues may be greatly increased. For this reason, this is only recommended to applications which would benefit from the fact of being contained in the same switch; ie: communication intensive MPI jobs. If you have any doubt or you are not sure whether your application can take advantage of this feature, please contact with support@bsc.es.

```
# @ mpi2 = 1
```

Every job running an MPI2 program must specify this directive in order to let it work properly.

```
# @ tracing = 1
```

Every job using tracing or profiling tools must set this option.

```
# @ x11 = 1
```

You can submit jobs applying for x-forwarding in the nodes. Those jobs can only be submitted via login(5-8). To achieve this, you must meet the following requirements:

1. You must login to any login of MareNostrum from outside with ssh option "-X".

```
ssh -X user@mn1.bsc.es
```

2. Once inside login(1-4), you must switch to login(5-8) using ssh "-Y" option.

```
ssh -Y login5
```

3. Then, you have to submit a job with this directive on. To be able to use it, the system requires at least a full exclusive node. There are two ways of asking for a full node, depending on the kind of job you are submitting:

```
# @ total_taks      = 4
# @ tasks_per_node  = 4
```

OR

```
# @ total_tasks     = 1
# @ cpus_per_task   = 4
```

Note that in spite of requesting 4 tasks or 4 threads, the application might use only 1 cpu.

Example of X11 script:

```
#!/bin/bash
# @ job_name           = test_xclock
# @ initialdir         = .
# @ output             = xclock_%j.out
# @ error              = xclock_%j.err
# @ total_tasks        = 4
# @ tasks_per_node     = 4
# @ x11                = 1
# @ wall_clock_limit   = 00:30:00

/usr/X11R6/bin/xclock
```

There are also a few SLURM environment variables you can use in your scripts:

Table 2. SLURM environment variables

<i>Variable</i>	<i>Meaning</i>
SLURM_JOBID	Specifies the job ID of the executing job
SLURM_NPROCS	Specifies the total number of processes in the job
SLURM_NNODES	Is the actual number of nodes assigned to run your job
SLURM_PROCID	Specifies the MPI rank (or relative process ID) for the current process. The range is from 0-(SLURM_NPROCS-1)
SLURM_NODEID	Specifies relative node ID of the current job. The range is from 0-(SLURM_NNODES-1)
SLURM_LOCALID	Specifies the node-local task ID for the process within a job
SLURM_NODENAME	Specifies the list of nodes on which the job is actually running

6.2.3. Examples

Example for a sequential job :

```
#!/bin/bash
# @ job_name           = test_serial
# @ initialdir         = .
# @ output             = serial_%j.out
# @ error              = serial_%j.err
# @ total_tasks        = 1
# @ wall_clock_limit   = 00:02:00
./serial_binary
```

Examples for a parallel job :

```
#!/bin/bash
# @ job_name           = test_parallel
# @ initialdir         = .
# @ output             = mpi_%j.out
# @ error              = mpi_%j.err
# @ total_tasks        = 56
# @ wall_clock_limit   = 00:02:00
srun ./parallel_binary
```

The job would be submitted using:

```
usertest@login1:~/Slurm/TEST> mns submit ptest.cmd
```

7. Software Environment

7.1. C Compilers

In MareNostrum you can find this C/C++ compilers :

`xlC / xlc -> IBM XL C/C++ Enterprise Edition version 10.1`

```
man xlc
man xlc
```

`gcc /g++ -> GNU Compilers for C/C++, Version 4.1.2`

```
man gcc
man g++
```

All invocations of the C or C++ compilers follow these suffix conventions for input files:

`.C, .cc, .cpp, or .cxx -> C++ source file.`

`.c -> C source file`

`.i -> preprocessed C source file`

`.so -> shared object file`

`.o -> object file for ld command`

`.s -> assembler source file`

By default, the preprocessor is run on both C and C++ source files.

These are the default sizes of the standard C/C++ datatypes on MareNostrum

Table 3. Data types

<i>Type</i>	<i>Length (bytes)</i>
bool (c++ only)	1
char	1
wchar_t	2
short	2
int	4
long	4 / 8 (64 bit mode -q64)
float	4
double	8
long double	8

7.1.1. Distributed Memory Parallelism

Invoking the script "mpicc" or "mpiCC", enables the program for running across several nodes of the SP. Of course, you are responsible for using a library such as MPI to arrange communication and coordination in such a program. Any of the mpi compilers sets the include path and library paths to pick up the MPI library.

```
% mpicc a.c -o a.exe
% mpiCC a.C -o a.exe
```

7.1.2. Shared Memory Parallelism

The IBM compilers with the extension `_r` invokes the thread safe version of compiler, for example `xlC` and `xlC_r`. It should be used when any kind of multi-threaded code, like OpenMP, is being built.

The IBM C and C++ compilers support a variety of shared-memory parallelism. **OpenMP** directives are fully supported by the IBM C and C++ compilers when one of the invocations with `_r` suffix is used.

`xlC_r -qsmp=omp` for C, and `xlC_r -qsmp=omp` for C++.

```
usertest@login1:~> xlC_r -qsmp=omp -o exename filename.c
usertest@login1:~> xlC_r -qsmp=omp -o exename filename.C
```

For mixed codes with MPI + OPENMP you need to execute the `mpicc` / `mpiCC` scripts with the Thread safe version of `xlC`/`xlC_r`.

```
usertest@login1:~> mpicc -qsmp=omp -o test.exe test_mpi-OMP.c
usertest@login1:~> mpiCC -qsmp=omp -o test.exe test_mpi-OMP.C
```

7.1.3. Automatic Parallelization

The IBM C compiler will attempt to automatically parallelize simple loop constructs. Use the option `"-qsmp"` with one of the `_r` invocations:

```
% xlC_r -qsmp a.c
```

7.1.4. 64 bit addressing

Both the IBM C and C++ compilers can support 64 bit addressing through the `-q64` option. Using this option causes all pointers to be 64 bits in length and increases the length of long datatypes from 32 to 64 bits. It does not change the default size of any other datatype. The default mode for both compilers is 32 bit addressing. The following points should be kept in mind if `-q64` is used:

If you have some object files that were compiled in 32-bit mode and others compiled in 64-bit mode the objects will not bind. You must recompile to ensure that all objects are in the same mode.

Your link options must reflect the type of objects you are linking. If you compiled 64-bit objects, you must also link these objects with the `-q64` option.

7.1.5. Optimization

The level optimization that we recommend for MareNostrum (PowerPC 970-MP) is :

```
-O3 -qstrict -qtune=ppc970 -qarch=ppc970 -qcache=auto
```

7.2. FORTRAN Compilers

In MareNostrum you can find this compilers :

`xlF` / `xlF90` / `xlF95` -> IBM XL FORTRAN Enterprise Edition for PowerPC Version 12.1

```
man xlF
man xlF90
man xlF95
```

`gfortran` -> GNU Compilers for FORTRAN, Version 4.1.2

```
man gfortran
```

By default, the compilers expect all FORTRAN source files to have the extension ".f", and all FORTRAN source files that require preprocessing to have the extension ".F". To change this behavior, you can use the "-qsuffix" option:

```
% xlf90_r -qsuffix=f=f90:cpp=F90 file1.f90 file2.F90
```

This command will compile files ending in ".f90", and preprocess and compile files ending in ".F90". By default, the compilers with FORTRAN 77 defaults require fixed form FORTRAN source. Compilers with FORTRAN 90 defaults require free form FORTRAN source. To change this behavior, use either the -qfree=f90, or -qfixed=72 option:

```
% xlf90_r -qfixed=72 filename.f
% xlf_r -qfree=f90 filename2.f
```

The file filename.f is compiled under the assumption that it is written in 72 column (using a number other than 72 is allowed) FORTRAN 77 source form, and the file filename2.f is compiled assuming it is written using the FORTRAN 90 free form.

To provide data to the preprocessor, such as defining a macros etc., use the following syntax:

```
% xlf_r -WF,-DSINGLE a.F
```

This command will preprocess the files a.F, passing the option "-DSINGLE" to the cpp command.

The default sizes of some of the datatypes can be changed with the "-qrealsize" and "-qintsize" options:

```
% xlf90_r -qrealsize=8 -qintsize=8 a.f
```

The sizes of pointers are changed via the "-q64" option, by default the option is -q32 (pointers of 32 bits)

7.2.1. Distributed Memory Parallelism

The scripts mpif77 and mpif90 allow to use the MPI calls to get parallelism

```
% mpif77 a.f -o a.exe
% mpif90 -qsuffix=f90 -qfree=f90 a.f90 -o a.exe
```

7.2.2. Shared Memory Parallelism

```
xlf_r / xlf90_r / xlf95_r
```

This invokes the thread safe version of xlf. It should be used when any kind of multi-threaded code, like OpenMP, is being built.

OpenMP directives are fully supported by the IBM C and C++ compilers when one of the invocations with _r suffix is used.

```
% xlf_r -qsmp=omp
```

```
usertest@login1:~> xlf_r -qsmp=omp -o exename filename.f
```

7.2.3. Automatic Parallelization

The IBM Fortran compiler will attempt to automatically parallelize simple loop constructs. Use the option "-qsmp" with one of the _r invocations:

```
% xlf_r -qsmp a.c
```

7.2.4. 64 bit Addressing

The IBM FORTRAN compilers can support 64 bit addressing through the `-q64` option. Using this option causes all pointers to be 64 bits in length and increases the length of long datatypes from 32 to 64 bits. It does not change the default size of any other datatype. The default mode for both compilers is 32 bit addressing. The following points should be kept in mind if `-q64` is used:

- If you have some object files that were compiled in 32-bit mode and others compiled in 64-bit mode the objects will not bind. You must recompile to ensure that all objects are in the same mode.
- Your link options must reflect the type of objects you are linking. If you compiled 64-bit objects, you must also link these objects with the `-q64` option.

7.3. Optimization

The level optimization that we recommend for MareNostrum (PowerPC 970-MP) is :

```
-O3 -qstrict -qtune=ppc970 -qarch=ppc970 -qcache=auto
```

Table 4. GNU Compilers (gcc-v4.1.2 /gfortran) Summary

Optimizations	<ul style="list-style-type: none"> • <code>-O0</code> (Do not optimize. Default) • <code>-O</code> (Optimizes generated code) • <code>-O2</code> (Optimize even more) • <code>-O3</code> (Aggressive optimization)
32 / 64 bits	<ul style="list-style-type: none"> • <code>-m32</code> (default) • <code>-m64</code>
Optimization for the PPC970-FX processor	<ul style="list-style-type: none"> • <code>-mcpu=970</code> • <code>-mtune=970</code>
VMX support	<ul style="list-style-type: none"> • <code>-maltivec</code> • <code>-mabi=altivec</code> (need to include <code>thealtivec.h</code> file)
OMP and threads	<ul style="list-style-type: none"> • OpenMP NOT SUPPORTED • <code>-lpthreads</code>
Debug flags	<ul style="list-style-type: none"> • <code>-g</code> (produce debugging information) • <code>-pg</code> (generates extra code to write profile information suitable for gprof)
MPI	<p>Compile with <code>mpicc/mpicc</code> or <code>mpif77/mpif90</code> and export the variable environment <code>MP_CC</code>, <code>MP_CXX</code>, <code>MP_FC</code>, <code>MP_F90</code></p> <pre>export MP_CC=gcc export MP_FC=gfortran mpicc test_mpi.c -o test mpif77 test_mpi.f -o test</pre> <p>*the default object mode is 64 bits in mpi wrappers (<code>mpicc/mpif77/mpif90</code>)</p>

Table 5. IBM Compilers for PPC64 (IBM XLC-v10.1 / XLF / XLF90 v12.1) Summary

Optimizations	<ul style="list-style-type: none"> • -qnooptimize (Do not optimize) • -O (Optimizes generated code. Default) • -O2 (Same as -O) -O3 (Performs some memory and compile-time intensive optimizations) • -O4 (aggressive optimizations) -O5 (high aggressive optimization)
32 / 64 bits	<ul style="list-style-type: none"> • -q32 (default) • -q64
Optimization for the PPC970-FX processor	<ul style="list-style-type: none"> • -qarch=ppc970 • -qtune=ppc970
VMX support	-qaltivec
OMP and threads	-qsmp=omp
Debug flags	<ul style="list-style-type: none"> • -g (produce debugging information) • -pg (generates extra code to write profile information suitable for gprof)
MPI	<p>Compile with mpicc/mpicc or mpif77/mpif90 and export the variable environment MP_CC, MP_CXX, MP_FC or MP_F90.</p> <pre>export MP_CC=xlc export MP_FC=xlf mpicc test_mpi.c -o test mpif77 test_mpi.f -o test</pre> <p>*the default object mode is 64 bits in mpi wrappers (mpicc/mpif77/mpif90)</p>

7.4. Debuggers

Parallel Debuggers:

- **Allinea's DDT:** For more info about DDT please visit:

<http://www.allinea.com/index.php?page=48>

Current version: 4.2.1

Binary: /gpfs/apps/DDT/2.4.1/bin/ddt

Usage: In order to run DDT you must take into account that in MareNostrum there are no specific nodes/queue to do debugging, so any job to debug must be sent to queues as any other usual task. In the case of DDT you must configure the software to use the SLURM batch system. This is done by starting DDT from the command line and then click on "*Run and Debug a Program*". A new window will appear letting you to introduce, among other options, the binary name to debug as well as the number of processors. Here you will find an entry called "*Options...*" followed by a button "*Change...*". Click this button and select *slurm* in the drop down menu "*MPI Implementation*". The last thing to do to complete DDT's setup in MN is to click on "*Job*"

Submission" in the left panel of the same window as before. Here are three important things: "*Submit command*" which must be set to *mnsu*submit; "*Cancel command*" which must be set to *mnc*cancel; and the last thing is to select the "*Submission template file*". This file is very important because this will be the file used to submit the job being debugged to the batch system. Here select: */gpfs/apps/DDT/2.4.1/templates/run.qtf*. This template must fit must of your needs. Anyhow, you can always copy this file to your home directory, modify it as needed and change the latter selection to point to your new template file. And this is it, now click on *Submit* to start debugging. This setting should be automatically saved so you won't need to reconfigure DDT next time you use it.

- **RogueWave's TOTALVIEW:**

Current version: 8.7.0-7

Binary: */gpfs/apps/TOTALVIEW/totalview*

Usage: Again, as said before, there is needed a template submission file in order to submit the job to the batch system.

You can use the following script and modify it as needed:

```
#!/bin/bash
# @ job_name           = TVD_MN
# @ initialdir         = .
# @ output             = mpi_%j.out
# @ error              = mpi_%j.err
# @ total_tasks        = <PUT HERE THE TASKS TO USE>
# @ wall_clock_limit   = <PUT HERE THE WALL CLOCK TIME>
# @ x11 = 1
# @ tracing = 1

/gpfs/apps/TOTALVIEW/totalview -mpi SLURM \
-np <NUMBER OF TASKS> -starter_args \
<other srun args> program_name -a <program args>
```

IMPORTANT: You must submit the job from one of the login nodes: 5 to 8; otherwise you will get an X11 error. In addition remember to log into MareNostrum (mn1,mn2,mn3 or mn4) using the "-X" ssh option and then jump to the other login with the "-Y" option. See Section 6.2.2 for more information on the x11 directive.

Serial Debuggers:

- */usr/bin/gdb*
- */usr/bin/gdb64*
- */usr/bin/gdb32*
- */gpfs/apps/DDD*

7.5. Software in MareNostrum

All software installed at MareNostrum can be found at */gpfs/apps/*. At the following link you will find a list of all installed software and a brief description:

<http://www.bsc.es/marenostrom-support-services/available-software>

7.6. Numerical Libraries

All listed libraries can be found at */gpfs/apps/* except ESSL (*/usr/lib*) and MASS (*/opt/ibmcomp/xlmass/4.3/*)

Table 6. Summary of numerical libraries available in MareNostrum

ATLAS	Versions 3.5.1 and 3.6.0 with ALTIVEC
BLACS	
BLAS	Version 1.0.0
BOOST	Versions 1.34.1 and 1.36.1
ESSL/PESSL	Versions ESSL 4.1 and PESSL 3.2
CFITSIO	Version March 2006
FFTW	Versions 2.1.5(default), 3.0.1 and 3.1.1
GLPK	Version 4.8
GNU-GSL	Versions 1.6 and 1.9
GOTO	Optimized for PPC970
HDF4	Version 4.1r5
HDF5	Versions 1.4.4 and 1.6.5
IOAPI	Version 2.2
LAPACK	Version 3.0
LP_SOLVE	Version 5.1
MASS	Version 4.1
NCARG	Version 4.4.1
NETCDF	Versions 3.6.0, 3.6.1 and 3.6.2

7.7. Modules Environment

The Environment Modules package provides for the dynamic modification of a user's environment via modulefiles. Each modulefile contains the information needed to configure the shell for an application or a compilation. Modules can be loaded and unloaded dynamically and atomically, in a clean fashion. All popular shells are supported, including bash, ksh, zsh, sh, csh, tcsh, as well as some scripting languages such as perl.

<http://modules.sourceforge.net/>

The first step in order to get the first environment loaded is to load BSC module.

```
module load bsc
```

Then, some default modules will be loaded and a set of available modules will be added.

This coherent set of software packages are divided into five categories:

- Environment: modulefiles dedicated to prepare the environment, for example, get all necessary variables to use mpi2 to compile and even run programs
- Tools: useful tools which can be used at any time (php, perl, ...)
- Applications: High Performance Computers programs (CPMD, NAMD, ...)
- Libraries: Those are typically loaded at a compilation time, they load into the environment the correct compiler and linker flags (FFTW, LAPACK, ...)
- Compilers: You can play with different compilers and versions in this package (c, c++, fortran, java, ...)

7.7.1. Modules tool usage

This section will explain a Quick Guide for the Modules environment usage at MareNostrum.

If you run "module help", you will be able to see all module commands. More information in module(1) manpage.

```
Modules Release 3.1.6 (Copyright GNU GPL v2 1991):
Available Commands and Usage:
+ add|loadmodulefile [modulefile ...]
+ rm|unloadmodulefile [modulefile ...]
+ switch|swapmodulefile1 modulefile2
+ display|showmodulefile [modulefile ...]
+ avail[modulefile [modulefile ...]]
+ use [-a|--append]dir [dir ...]
+ unusedir [dir ...]
+ update
+ purge
+ list
+ clear
+ help[modulefile [modulefile ...]]
+ whatis[modulefile [modulefile ...]]
+ apropos|keywordstring
+ initaddmodulefile [modulefile ...]
+ initprependmodulefile [modulefile ...]
+ initrmmodulefile [modulefile ...]
+ initswitchmodulefile1 modulefile2
+ initlist
+ initclear
```

The most important commands are: list, avail, load, unload, switch and purge

- **module list** shows all the modules you have loaded:

```
% module list
Currently Loaded Modulefiles:
  1) c++/10.1                3) fortran/12.1
  2) c/10.1                  4) mode/64
```

- **module avail** shows all the modules that user is able to load:

```
% module avail
----- environment -----
compilerwrappers/no          mpi2/no
compilerwrappers/yes(default) mpi2/yes(default)
mode/32                      oldcompilers/yes(default)
mode/64(default)
----- compilers -----
c++/10.1(default)            c/8.0
java/1.4(default)           c++/8.0
fortran/10.1                 java/1.5
c/10.1(default)              fortran/12.1(default)
----- libraries -----
blas/4(default)              fftw/3
mass/4(default)              blassmp/4(default)
fftw/3.0.1                   netcdf/3(default)
essl/4(default)              fftw/3.1.1
scalapack/1.7(default)       esslsmp/4(default)
hdf5/1.6(default)           fftw/2.1.5(default)   lapack/3.0(default)
```

```
----- tools -----
emacs/21(default)      omniorb/4.0(default)
tcl/8.4(default)       globus/4.0(default)
openssh/3(default)     tk/8.4(default)
gmake/3(default)       perl/5.8(default)
totalview/8(default)   hpm/2.5(default)
python/2.4(default)    nedit/5(default)
python/2.5
----- applications -----
cpmd/3.11(default)      cpmd2cube/jan05
namd/2.6(default)       cpmd2cube/apr06(default)
lammps/22Jan08(default)
```

- **module load** let user load the necessary environment variables for the selected modulefile (PATH, MANPATH, LD_LIBRARY_PATH...)

```
% module load fftw
load fftw/2.1.5 (CFLAGS,FFLAGS,LDFLAGS)
```

- **module unload** removes all environment changes made by module load command:

```
% module unload fftw
remove fftw/2.1.5 (CFLAGS,FFLAGS,LDFLAGS)
```

- **module switch** acts as module unload and module load command at same time:

```
% module load fftw
load fftw/2.1.5 (CFLAGS,FFLAGS,LDFLAGS)
% module switch fftw fftw/3.2.2
switch1 fftw/2.1.5 (CFLAGS,FFLAGS,LDFLAGS)
switch2 fftw/3.2.2 (CFLAGS,FFLAGS,LDFLAGS)
switch3 fftw/2.1.5 (CFLAGS,FFLAGS,LDFLAGS)
ModuleCmd_Switch.c(243):VERB:4: done
% module list
Currently Loaded Modulefiles:
  1) /fftw/3.2.2
```

8. Acknowledgment in publications

If you want to include any BSC acknowledgment in publications you can use some of the following:

["The simulations have been done in the supercomputer MareNostrum at Barcelona Supercomputing Center - Centro Nacional de Supercomputación (The Spanish National Supercomputing Center)"]

["The author thankfully acknowledges the computer resources, technical expertise and assistance provided by the Barcelona Supercomputing Center - Centro Nacional de Supercomputación."]

9. Getting help

In addition to our online information tools described in section 3, BSC provides to users excellent consulting assistance. User support consultants are available during normal business hours, Monday to Friday, 09 a.m. to 18 p.m. (CEST time).

User questions and support are handled at:

support@bsc.es If you need assistance, please supply us with the nature of the problem, the date and time that the problem occurred, and the location of any other relevant information, such as output

files. Please contact BSC if you have any questions or comments regarding policies or procedures. Our address is:

Barcelona Supercomputing Center – Centro Nacional de Supercomputación

C/ Jordi Girona, 31, Edificio Capilla

08034 Barcelona

10. FAQ's

10.1. *How can I get some help*

The best way to get help is to email us at: <support@bsc.es>

10.2. *How do I know the position of my first job in queue?*

You can use the command:

```
mnstart <job_ID>
```

shows information about the estimated time for the specified job to be executed.

10.3. *How can I see the status of my jobs in queue?*

The wrappers commands `mnq` and `checkjob` provide you information about your jobs in queues:

```
mnq
```

shows all the jobs submitted by your user.

```
checkjob <job_ID>
```

obtains detailed information about a specific job.

10.4. *What is the default object mode compilation in MareNostrum?*

It depends on the compiler used. If you use the XL compilers or the GNU compilers the default object mode is 32 but with the mpi wrappers as `mpicc` the default object mode is 64. You can also use the options `-m64` or `-m32` with GNU compilers and `-q64` or `-q32` with XL compilers explicitly.

10.5. *What version of MPI is currently available at MareNostrum?*

Currently the default MPI version is `mpich-mx`, implementing MPI-1 standard. Additionally, `mpich2-mx`, which implements MPI-2 is also available at `/gpfs/apps/MPICH2`. In order to use it, some environment variables must be set before compiling:

```
export PATH=/gpfs/apps/MPICH2/mx/default/bin/:$PATH
export MP_IMPL=anl2
```

Also, remember to add the following directive in your job scripts:

```
# @ mpi2 = 1
```

10.6. *Which compilers are available at MareNostrum?*

You can find the XL compilers from IBM: `xlC`, `xlC_r`, `xlF`, `xlF90`, `xlF95` (along with the thread safe versions `xlC_r`, `xlC_r_r`, `xlF_r`, `xlF90_r`, `xlF95_r`) and the GNU compilers `gcc`, `g77`, `g+`. OpenMP is fully supported by IBM compilers when `_r` suffix is used. There also exist

wrappers useful when compiling MPI applications: mpicc, mpiCC, mpif77, mpif90. By default, this wrappers will use the IBM compilers but you can modify this behavior setting some environment variables properly:

```
export MP_CC=gcc
export MP_CXX=g++
export OBJECT_MODE=64
```

The Versions available for these compilers are: xlc v. 10.1; xlf v. 12.1; gcc v. 4.1.2; gfortran v 4.1.2

10.7. *What options are recommended to compile in MareNostrum?*

The recommended options to compile in *MareNostrum* for IBM compilers are:

```
-O3 -qstrict -qtune=ppc970 -qarch=ppc970 -qcache=auto
```

10.8. *Should I be careful with the memory consumption of my jobs?*

Yes, you should. Each one of the MareNostrum Blades has 8 Gb of RAM. shared by two (dual core) processors. Up to 90 % of this memory can be consumed for user jobs, while, at least, 10 % has to be available for the Operating System and daemons. According to that, you must limit the memory consumption of your job to 1.8 Gb per Process (which is 7.2 Gb per Blade when there is one task per processor).

10.9. *Where should I install programs common to all the members of my project group?*

You should install programs accessible to all your group members in the filesystem `/gpfs/projects/` (All the groups will have a directory such as `/gpfs/projects/<group_id>`).

10.10. *Where should I store temporary data?*

You can use the local hard disk of the blade (`/scratch`) to store temporary data for your jobs. However, you must make sure all this data is deleted when the job finishes.

10.11. *Is there any way to make my jobs wait less in queue before running?*

You must tune the directive `#@ wall_clock_limit` to the expected job duration. This value will be used by the batch system to decide when to schedule your job, so, shorter values are likely to reduce waiting time; However, notice that when a job exceeds its `wall_clock_limit` will be canceled, so, it is recommended to work with an small security margin.

10.12. *How can I improve the transfer rate from my computer to MareNostrum?*

There exists a patch for SCP that can improve transfer rates in high bandwidth, high latency connections. This patch has only to be applied in the client side. For more information about this patch please visit <http://www.psc.edu/networking/projects/hpn-ssh/> [<http://www.psc.edu/networking/projects/hpn-ssh/>]. Before applying this patch PLEASE contact the support team at `<support@bsc.es>`

10.13. *How can I know how much time I have consumed?*

There will be a web page with your activity time consumption upgraded weekly. You will receive information from the support team about how to access this page.

10.14. *Should I be careful with the Input / Output over parallel filesystem (GPFS)?*

Parallel Filesystem can be a bottleneck when different processes of one job are writing to GPFS along the execution. In this kind of jobs, one possible way to improve the job performance is to copy the needed data for each job to the local scratch at the beginning and

copy back to gpfs at the end, (with this scheme, most of I/O will be performed locally). This scheme is also recommended for massive sets of sequential jobs.

10.15. How can I use GM Myrinet protocol with my MPI applications?

Myrinet GM protocol is no longer available at MareNostrum. MX should work for you.

A. SSH

SSH is a program that enables secure logins over an insecure network. It encrypts all the data passing both ways, so that if it is intercepted it cannot be read. It also replaces the old an insecure tools like telnet, rlogin, rcp, ftp, etc. SSH is a client-server software. Both machines must have ssh installed for it to work.

We have already installed a ssh server in our machines. You must have installed an ssh client in your local machine. SSH is available without charge for almost all versions of Unix. BSC recommend the use of OpenSSH client that can be download from <http://www.openssh.org>, but any client compatible with SSH version 2 can be used.

In windows systems BSC recommend the use of putty. It is a free SSH client that you can download from <http://www.putty.nl/>. But you can also, any client compatible with SSH version 2 can be used.

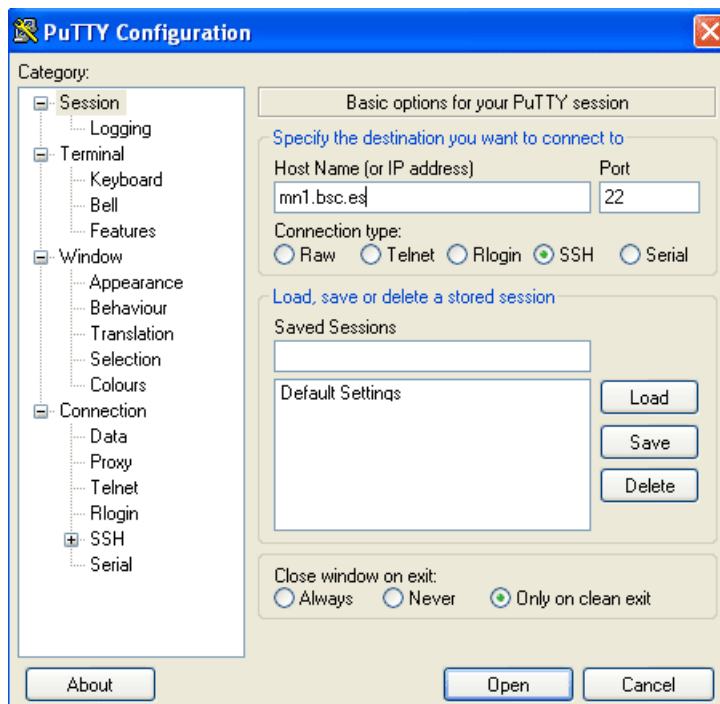
In the next lines we will describe how to install, configure and use a ssh client under Windows systems.

Once the client has been installed, it is needed to fix some settings. At least it is necessary to

- Select SSH as a default protocol
- Select port 22
- Specify the remote machine and username

For example with putty client:

Figure A.1. Putty client



This is the first windows that you will see at putty startup. Once finished, press the “Open” button. If it is your first connection to the machine, you will get a Warning telling you that the host key from the server is unknown, and will ask you if you are agree to cache the new host key, press Yes.

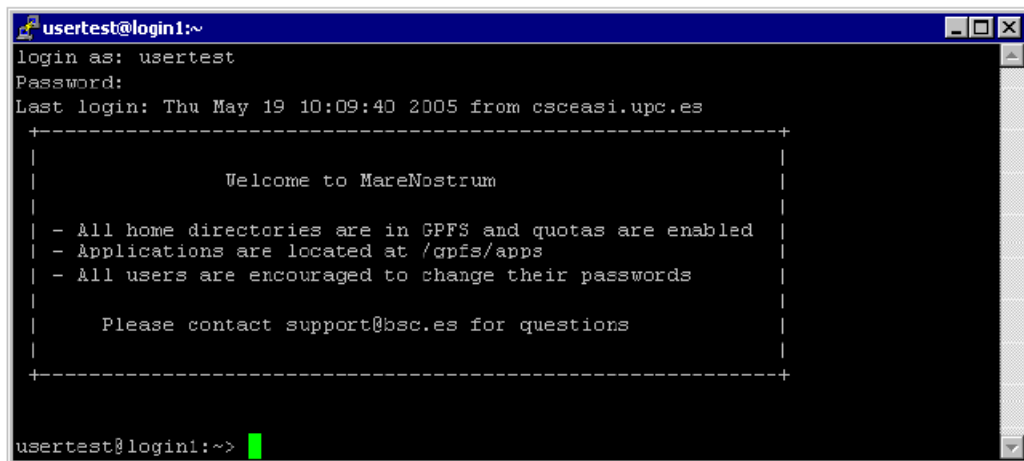
Figure A.2. Putty certificate security alert



IMPORTANT: If you see this warning another time and you haven't modified or reinstalled the ssh client , please, don't log in , and contact with BSC support.

Finally, a new window will appear asking for your login and passwd:

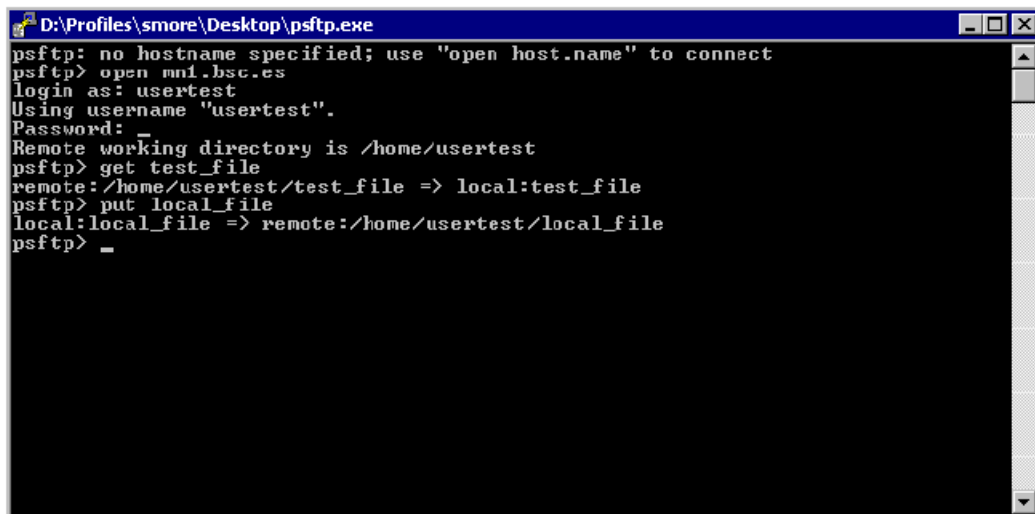
Figure A.3. MareNostrum login



To transfer files to or from MareNostrum you need a secure ftp (sftp) o secure copy (scp) client. There are several different clients, but as previously mentioned, BSC recommend the use of putty clients for transferring files: *psftp* and *pscp*. You can find it at the same web page as putty (<http://www.putty.nl/> [<http://www.putty.nl/>]).

Some other possible tools for users requiring graphical file transfers could be:

- WinSCP: Freeware Sftp and Scp client for Windows (<http://www.winscp.net>)
- SSH: Not free. (<http://www.ssh.org>)

Figure A.4. psftp screenshot

```
D:\Profiles\smore\Desktop\psftp.exe
psftp: no hostname specified; use "open host.name" to connect
psftp> open mn1.bsc.es
login as: usertest
Using username "usertest".
Password: _
Remote working directory is /home/usertest
psftp> get test_file
remote:/home/usertest/test_file => local:test_file
psftp> put local_file
local:local_file => remote:/home/usertest/local_file
psftp> _
```

Example of using psftp:

As you can see, first it asks for the machine name (mn1.bsc.es), and then for the username and passwd. Once you are connected, it's like a Unix command line.

With command *help* you will obtain a list of all possible commands. But the most useful are:

- get file_name : To transfer from MareNostrum to your local machine.
- put file_name : To transfer a file from your local machine to marenostrum.
- cd directory : To change remote working directory.
- dir : To list contents of a remote directory.
- lcd directory : To change local working directory.
- !dir : To list contents of a local directory.

Example of using pscp:

First of all you need a command windows were you will execute the pscp.exe

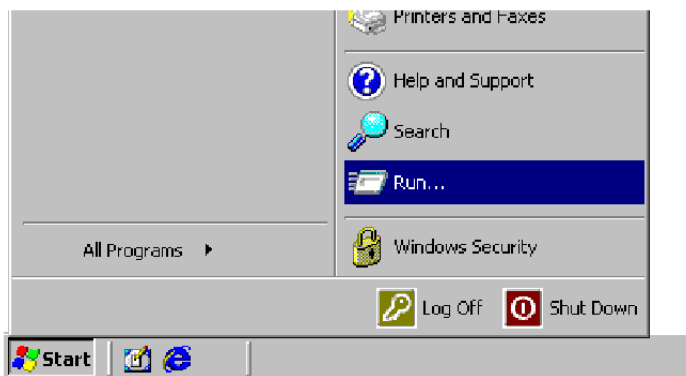
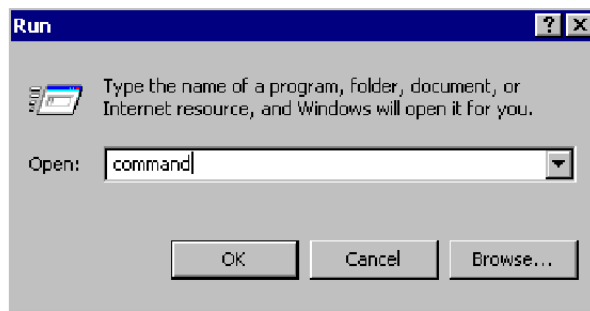
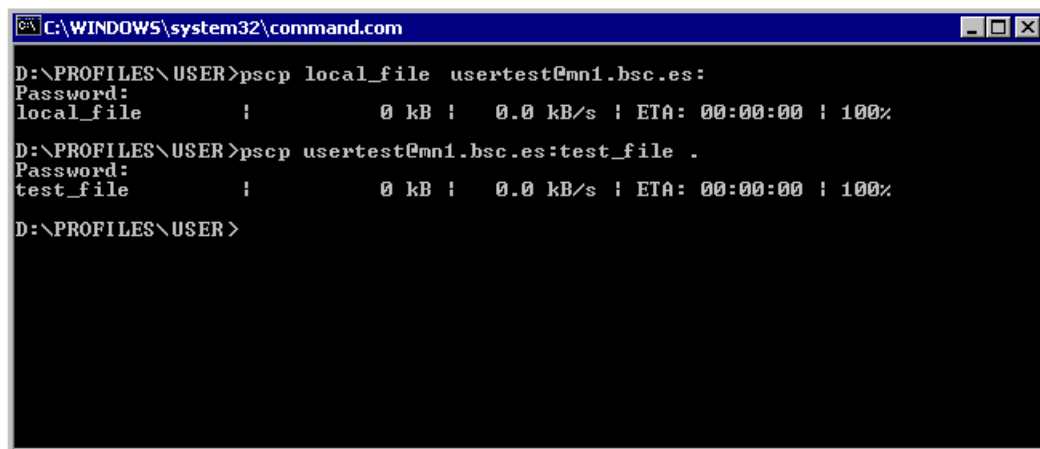
Figure A.5. step 1

Figure A.6. step 2**Figure A.7. step 3**

In step 3 you can see two examples: copying files from your local machine to MareNostrum, and from MareNostrum to your local machine. The syntax is the same that cp command except that for remote files you need to specify the remote machine:

Copy a file from MareNostrum:

```
pscp.exe YourUserName@mn1.bsc.es:remote_file local_file
```

Copy a file to MareNostrum:

```
pscp.exe local_file YourUserName@mn1.bsc.es:remote_file
```

In order to start remote X applications you need and X-Server running in your local machine. Here is a list of most common X-servers for windows:

- Cygwin/X: <http://x.cygwin.com>
- X-Win32 : <http://www.starnet.com>
- WinaXe : <http://labf.com>
- XconnectPro : <http://www.labtam-inc.com>
- Exceed : <http://www.hummingbird.com>

The only Open Source X-server listed here is Cygwin/X, you need to pay for the others.

Once the X-Server is running run putty with X11 forwarding enabled:

Figure A.8. Putty X11 configuration