



# **Supercomputing environment**

**TMA4280—Introduction to Supercomputing**

NTNU, IMF

February 21. 2018

# Supercomputing environment



- Supercomputers use UNIX-type operating systems.
- Predominantly Linux.
- Using a shell interpreter is the only way to interact with the system.

Documentation and tutorials are usually offered on the System Administration group's website:  
<https://www.hpc.ntnu.no/display/hpc/User+Guide>

# Login



- Non-graphical interaction with the supercomputer.
- Two kinds of nodes: login/interactive nodes and compute nodes.
- Login is handled through *Secure SHell* (SSH).
- On Linux/UNIX/macOS: pre-installed OpenSSH.
- On Windows: third-party client PuTTY.

# Login



Three ingredients:

- Username: NTNU login name.
- Host: `training.hpc.ntnu.no`.
- Credential: a password or an authentication key.

```
ssh username@training.hpc.ntnu.no
```

/ \$ \$ \$ \$ \$ \$	/ \$ \$	/ \$ \$	/ \$ \$ \$ \$ \$	/ \$ \$
\$ \$ _ _ _ _ /	_ _ /	/ \$ \$ /	_ \$ \$ _ /	\$ \$
\$ \$	/ \$ \$ \$ \$ \$	/ \$ \$	/ \$ \$ \$ \$ \$ \$	/ \$ \$
\$ \$ \$ \$	/ \$ \$ _ \$ \$	/ \$ \$	/ \$ \$ _ \$ \$	/ \$ \$
\$ \$ _ /	\$ \$ \ \$ \$	\$ \$	\$ \$	/ \$ \$
\$ \$	\$ \$	\$ \$	\$ \$	/ \$ \$ /
\$ \$ \$ \$ \$ \$	\$ \$ \$ \$ \$ /	\$ \$	\$ \$ \$ \$ \$ \$	/ \$ \$ /
_ _ _ _ _ /	\$ \$ _ _ /	_ _ \ _ _ _ _ _ /	_ _ _ _ _ /	\ _ _ _ _ _ /
\$ \$				
\$ \$				
/				

NB !

NB !

```
to list available modules software:
```

Starting 6th of february. 2018:

Registration: Send an e-mail to: [adm@hpc.ntnu.no](mailto:adm@hpc.ntnu.no)

To get **help** and support, please send email to: [help@hpc.ntnu.no](mailto:help@hpc.ntnu.no)

Or check our web page: <http://www.hpc.ntnu.no/>

Be Nice!

## File transfer



- File transfers is performed using *Secure Copy* (`scp`).
- On Linux/UNIX/macOS: pre-installed OpenSSH.
- On Windows: third-party client *WinSCP*.

Advice: for source code and result files in text format use a revision control system like GIT.

# Authentication with SSH key



- Avoid typing your password, use key authentication.
- Type only return if you want an empty passphrase.
- Generate an SSH key on the **local**:

```
ssh-keygen
```

- Copy the content of **public** key `id_rsa.pub` to the **remote** host file  
scp

```
scp ~/.ssh/id_rsa.pub username@training.hpc.ntnu.no:~/.ssh/authorized_keys
```

Tutorial:

[https://debian-administration.org/article/530/SSH\\_with\\_authentication\\_key\\_instead\\_of\\_password](https://debian-administration.org/article/530/SSH_with_authentication_key_instead_of_password)

## Editing files



For such small project, only a good text editor is required:

- Emacs (use locally if you can)
- Vim (handy for using remotely, a bit of a learning curve)
- Nano (simpler than Vim)
- Gedit (nice graphical editor)
- Kate (same, not installed on Lille)
- Notepad++ (good for Windows users, not installed on Lille)
- ...

In practice, non-graphical editors are preferred since working on a login node requires using the terminal: most people use Vim, Emacs, or Nano.



## Graphical display (X11 forwarding)



If you want to run graphical programs on Lille you have to tunnel the display through ssh. This is called *X forwarding*.

— In Linux, it's quite easy:

```
ssh -X username@training.hpc.ntnu.no
```

Or in your ~/.ssh/config:

```
ForwardX11 Yes
```

— In OSX, you have to start `X11.app`, then do the same.

— In Windows, you can use *X-Win32*, which is available on progdist.

This is usually not required and puts unnecessarily load on the login nodes.

# Modules

As people sharing a supercomputer have different needs, the tools cannot be all installed in the default system directories. Software is offered through a *modules* system. They will not be available to you until you load the module in question.

- List all available modules:

```
module spider
```

- List available modules:

```
module avail
```

- Load a module:

```
module load gcc
```

- Load a module with a specific version:

```
module load gcc/6.3.0
```

```
module load openmpi/2.0.1
```

- List loaded modules:

```
module list
```

# Modules



Some relevant modules for this course:

- gcc/6.3.0: GCC compilers (gcc, g++ and gfortran).
- openmpi/2.0.1: OpenMPI implementation of MPI (*Message Passing Toolkit*).
- openblas/0.2.19: BLAS library.

Note that if you use CMake to build your programs, you may need to pass the compiler you want to use:

```
mkdir build
```

```
cd build
```

```
CXX=g++ CC=gcc FC=gfortran cmake ..
```

# Modules



```
[aurelila@lille-login2 ~]$ module avail
```

```
----- /share/apps/modules/all/Core -----  
EasyBuild/3.3.0      Go/1.8.1      foss/2017a      (D)  
FLUENT/18.0          Java/1.8.0_92  icc/2017.1.132-GCC-6.3.0-2.27  
FLUENT/18.2          MATLAB/2016b   ifort/2017.1.132-GCC-6.3.0-2.27  
GCC/4.9.3-2.25       MATLAB/2017a   intel/2017a  
GCC/5.4.0-2.26       foss/2016a  
GCC/6.3.0-2.27       foss/2016b  
(D)
```

```
----- /share/apps/modulefiles/Core -----  
easybuild/2.9.0      gcc/6.2.0      matlab/R2016b  
gcc/4.9.4            gcc/6.3.0 (D)   python/2.7.3
```

Where:

D: Default Module

Use "module spider" to find all possible modules.

Use "module keyword key1 key2 ..." to search for all possible modules matching any of the "keys".

# Modules



Modules have dependencies: for example openmpi cannot be loaded unless a compiler has been loaded already:

```
[aurelila@lille-login2 ~]$ module load openmpi
Lmod has detected the following error:  These module(s) exist but cannot be
loaded as requested: "openmpi"
```

```
Try: "module spider openmpi" to see how to load the module(s).
```

Load gcc first:

```
[aurelila@lille-login2 ~]$ module load gcc openmpi
```

You can add this line in your shell profile or write a script to do it.

## Batch scheduler/Queuing system

To schedule jobs run by users, a queueing system is installed on supercomputers:

- each job submitted is appended to the queue with a given priority,
- then launched when reaching the top of the queue (*workq* on Lille),
- status (success/failure) is reported accordingly,
- computational time  $n$  core.hour is charged to the project (maximum resource is 20 processes on 2 nodes).

A simple job:

```
echo "sleep 30;echo hello world"|qsub -q training \  
-W group_list=itea_lille -tma4280 \  
-lselect=2:ncpus=20:mpiprocs=20
```

If the queue you need to use has another name, substitute the *training* argument.

# Batch scheduler/Queuing system



The status of the training queue can be inspected:

```
[aurelila@lille-login2 ~]$ qstat -Q training
```

Queue	Max	Tot	Ena	Str	Que	Run	Hld	Wat	Trn	Ext	Type
training	0	0	yes	yes	0	0	0	0	0	0	Exec

with training the name of requested queue for example.

The status of jobs for a given username can be displayed:

```
[aurelila@lille-login2 ~]$ qstat -u username
```

# Running jobs on Lille

After compiling your program, you have to write a *job script*. Example (the *pi* program):

```
#!/bin/bash

#PBS -N pi
#PBS -A itea_lille -tma4280
#PBS -W group_list=itea_lille -tma4280
#PBS -l walltime=00:01:00
#PBS -l select=2:ncpus=20:mpiprocs=16

cd $PBS_O_WORKDIR
module load openmpi
mpiexec ./pi 1000000
```



# Running jobs on Lille



```
#PBS -N pi
```

My job is called “pi”.

```
#PBS -A itea_lille-tma4280
```

The time spent executing this job should be charged to itea\_lille-tma4280.

```
#PBS -l walltime=00:01:00
```

The walltime limit for this job is one minute.

```
#PBS -l select=2:ncpus=20:mpiprocs=16
```

I want two units of 20 CPUs each (two nodes, that is) and I want 16 processes on each of them (32 in total). On Lille, `ncpus` should *a/ways* be equal to 20.

## Running jobs on Lille



```
cd $PBS_O_WORKDIR
```

Ensure that we are in the correct directory. This should always be in your job script.

```
module load openmpi
```

Make sure the `openmpi` module is loaded so that the `mpiexec` command is available to run MPI programs.

```
mpiexec ./pi 1000000
```

Run the program.

# Running jobs on Lille



Submit a job using qsub:

```
qsub job.sh  
5723717.service2
```

qsub will reply with a job ID number. You can ask for the status of your job with

```
qstat -f 5723717.service2
```

or see a list of all jobs running and queued

```
qstat
```

## Running jobs on Lille



When the program has completed, the accumulated output will be written to files in the same folder you launched it from.

```
ls
job.sh  pi  pi.c  pi.e5723717  pi.o5723717
```

The e-file contains stderr (empty?) and the o-file contains output from stdout (the most interesting one).

```
cat pi.o5723717
Agent pid 21651
pi=3.141593e+00, error=8.437695e-14, duration=2.177000e-03
Start Epilogue v3.0.1 Wed Jan 27 14:18:27 CET 2016
clean up
End Epilogue v3.0.1 Wed Jan 27 14:18:28 CET 2016
```

## Other PBS options



```
#PBS -o stdout  
#PBS -e stderr
```

I want my output files to have more sensible names.

```
#PBS -m a b e
```

I want an e-mail notification when the job starts (b), ends (e) or if it aborts (a).

```
#PBS -M some@where.com
```

...and this is where that e-mail should be sent to.

```
#PBS -l ....:ompthreads=16
```

for 16 OpenMP threads per process.

See <https://www.hpc.ntnu.no/display/hpc/PBS+Professional>

## More information



The NTNU HPC Wiki has a very good user guide.

<https://www.hpc.ntnu.no/display/hpc/User+Guide>