HPC Cluster TU Delft

The official documentation

https://login.daic.tudelft.nl/

How to connect to the cluster

1. Open terminal (cmd on Windows).
2. Type ssh TUDelft_ID@linux-bastion.tudelft.nl to connect to the 'linux-bastion' proxy.
    a. TUDelft_ID: **zli10**
    b. Password: **11235813Lzy~**
3. Connect to one of the login servers using ssh <login_server>. The available servers are:
    a. Login.hpc
    b. login1.hpc
    c. login2.hpc
    d. login3.hpc

An example of a regular connection would be:

1. Open terminal:
    a. For MacOS: Press *Command+Space*, then type *terminal*.
    b. For Windows: Click **Start**, **All Programs**, **Accessories**, **Command Prompt**.
2. Type: *ssh TuDelft_ID@linux-bastion.tudelft.nl*.
3. Type your password.
4. Type: ***ssh login2.hpc***
    a. Other running nodes are as above
5. You're connected!

How to run/cancel a job

Load module

```
module use /opt/insy/modulefiles
module load miniconda/3.8
```

Activate environment

```
conda activate /tudelft.net/staff-umbrella/zlitransfer/env-train
# conda deactivate
```

Submit job

```
cd /tudelft.net/staff-umbrella/zlitransfer/finetune/
sbatch train.sbatch

# console output will be loaded in /out/log.out
#  You might change the name of the output file so that different jobs have
different output file

# Check status
squeue -u zli10
```

To cancel a job

```
scancel [JOBID]
```

All steps

```
module use /opt/insy/modulefiles
module load miniconda/3.8
conda activate /tudelft.net/staff-umbrella/zlitransfer/env-train
cd /tudelft.net/staff-umbrella/zlitransfer/
```

File structure

/finetune, /util - scripts

```
/hf           - huggingface downloading path
/out          - console output from the running script
/datasets     - training datasets
/env-train    - conda environment path
```

Faster authentication (Linux)

If you log into the cluster often it can get quite inconvenient typing in your password or doing several operations repeatedly.

You can generate an RSA key-pair to authenticate yourself automatically by typing:

Generate your personal RSA key-pair
ssh-keygen -t rsa -b [key-length:between-2048-and-4096]
N.B. do NOT change the proposed path, and do NOT use a password to protect you private key (unless you know what you're doing)

And then

Allow the remote host to authenticate you using your public key
cat ~/.ssh/id_rsa.pub | ssh [username]@[hostname] "cat >> ~/.ssh/authorized_keys"

This let's you login using simply ssh username@linux-bastion.tudelft.nl

This can be further shortened using your ~/.ssh/config file, if you add a line like

Host delft
  HostName linux-bastion.tudelft.nl
  User bhalpern

Which will enable you to login with simply typing ssh delft.

Convenient navigation

If you are logged into the cluster, you can make a .bashrc file which you can modify to set up some things as you desire.

You can add i.e a line which automatically navigates to your working directory on the group share, i.e

cd /tudelft.net/staff-bulk/ewi/insy/SpeechLab/bhalpern

If you install libraries using pip (see below), you might need to download statically built binaries. The .bashrc file is an ideal place to set your environment variables, i.e:

export PATH="/tudelft.net/staff-bulk/ewi/insy/SpeechLab/bhalpern/ffmpeg/:$PATH"

It might happen that source bashrc is not loaded upon login, in that case, make sure that you have a .profile file containing ~/.bashrc and a ~/.bash_profile file containing source ~/.profile.

How to upload/download files (FileZilla)

Graphical user interface (GUI)

1. Download FileZilla (https://filezilla-project.org/download.php?type=client).
2. Open the program and fill in the next values:
    a. Host: sftp://linux-bastion.tudelft.nl
    b. Username: <TUDelft_ID>
    c. Password: <TUDelft_Password>
    d. Port: leave it empty.

Terminal commands

It is also possible to use Terminal in Mac OSX/Linux to perform file copies with the *scp* utility.

The default pattern is scp [source] [dest], some examples:

1. If I wanted to copy a file cheese.txt into my folder in the cluster I would write: scp bhalpern@linux-bastion.tudelft.nl:/tudelft.net/staff-umbrella/zlitransfer
2. A whole folder could be copied with the scp -r /home/user/my_favourite_directory bhalpern@linux-bastion.tudelft.nl:/tudelft.net/staff-umbrella/zlitransfer
3. Similarly you can swap the the order to create backups of your experiments to your local computer as: scp -r bhalpern@linux-bastion.tudelft.nl:/tudelft.net/staff-umbrella/zlitransfer my_sota_experiments /home/all_experiments

The scp utility is documented in detail at https://man7.org/linux/man-pages/man1/scp.1.html, otherwise please refer to scp --help.

Where can I upload or store my files?

/tudelft.net/staff-umbrella/zlitransfer

Transfer files to the cluster

```
scp -r heterogeneous_graph/finetune/train.py zli10@linux-bastion.tudelft.nl:/tudelft.net/staff-umbrella/zlitransfer
```

You have two places for that. The first one is the folder to which you connect when using your FTP service (‘home/nfs/<TUDelft_ID>’). The space is limited, but no-one can access that folder but you. It’s a good place for your codes. The second one is in ‘/tudelft.net/staff-bulk/ewi/insy/MMC/’. There you can / a folder for your data, but keep in mind that everyone in the department will have access to it. It’s the place for results, datasets, and anything that occupies a medium-high space on disk.

How does the cluster work?

The cluster makes use of ‘.sbatch’ files to work. You can think of these files as orders in which you specify your requirements, such as number of CPUs or GPUs or the amount of time that the specific job will be using them. Depending on your past use, the amount of resources you’re asking for and the time that you say you will be using them, your job gets a priority. The lower, the earlier your job starts.

How do I create a ‘.sbatch’ file?

‘.sbatch’ files can be created in a regular text editor, adding .sbatch as the extension instead of .txt
On Windows, I recommend Notepad++. The standard Notepad ends lines with a DOS code instead of a UNIX one. As a result, SLURM cannot read it. If you have this problem, just open the .sbatch file with Notepad++ and go to Edit -> EOL Conversion -> UNIX/OSX format.

A basic ‘.sbatch’ file will have the following structure:

```
#!/bin/sh
#SBATCH --partition=general
#SBATCH --qos=short
#SBATCH --time=01:00:00 (one hour)
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=2
#SBATCH --gres=gpu
#SBATCH --mem=40960
#SBATCH --mail-type=END
#SBATCH --chdir=/tudelft.net/staff-umbrella/zlitransfer
```

#!/bin/sh

```
#SBATCH --partition=general
#SBATCH --qos=short
#SBATCH --time=04:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=1G
#SBATCH --mail-type=ALL
#SBATCH --chdir=/tudelft.net/staff-bulk/ewi/insy/MMC/<YourFolder>

srun -u --output="<LogFile>.out" python3.4 <PythonFile>.py <Parameters>
```

In such a file, you usually want to modify the following parameters:

- **--qos**
  It stands for Quality of Service. If the time specified is not larger than 4 hours, use 'short'. Your priority will be higher. If you plan your job to run longer than 4 hours, but less than a week, use 'long'. If your job is going to be running for more than a week, use 'infinite'.
- **--time**
  The time you code is going to be running. The format is {Days}:{Hours}:{Minutes}:{Seconds}.
- **--cpus-per-task**
  The number of CPUs that you are going to use. They are allocated in multiples of two, considering all the tasks. For example, if you have 4 tasks, each of them using 3 cpus, you will have 12 cpus. However, if you have 3 tasks, you will use 10 (9+1).
- **--mem**
  The amount of RAM memory that you need to use. You can use 'M' for Megabytes and 'G' for Gigabytes.
- **--mail-type**
  You receive an email when your job starts to run, and ends.
- **--chdir**
  The base directory for your code. Unless you declare an absolute path, everything will be referred to this.

Basic commands

- **cd**
  It is used to move from one folder to the other when connected to the cluster. To go back, use *cd ..* .
- **ls**
  It prints the files and folders existing in the directory in which you are.
- **sbatch <file>.sbatch**
  It sends your request to the cluster, putting it in the queue. You will receive a message telling you that, and specifying the ID of your job.
- **scancel <job_id>**

It stops the specified job. If it is still in the queue, it gets removed from it. You can also define ranges; for example, *scancel <job_id0>-<job_idN>*. It will only delete the jobs you've sent. You can cancel all your jobs using *scancel -u ${USER}* or *scancel -u <TUDelft_ID>*.

- **squeue**
  It prints the jobs currently running or waiting to start. If you are just interested in your jobs, type *squeue -u ${USER}* or *squeue -u <TUDelft_User>*.
- **seff <job_id>**
  It gives you information about a certain job. It is only reliable after the job has finished.
- **sacct [-u username, -S starttime (Month/Day/Year)]**
  It gives you information about all the jobs a certain user sent from a specific point in time.

Error codes and their meanings

https://hpc-discourse.usc.edu/t/exit-codes-and-their-meanings/414

[Python] How much RAM memory do I need for a job?

There is a library in Python called *Guppy* that allows you to measure (roughly) the amount of memory that a code needs to run. It could be a good first estimation to run your jobs more efficiently on the Cluster.

```
from guppy import hpy

hp = hpy()

# All your code

print(hp.heap())
```

After running the code, you will see on your terminal something similar to:

```
Partition of a set of 360196 objects. Total size = 1071748535 bytes.
 Index  Count   %     Size   % Cumulative  % Kind (class / dict of class)
     0    115   0 951122428  89 951122428  89 numpy.ndarray
     1      1   0 73156768   7 1024279196  96 pandas.core.frame.DataFrame
     2  99565  28 15203337   1 1039482533  97 str
     3 100924  28  8604480   1 1048087013  98 tuple
     4  45117  13  3670809   0 1051757822  98 bytes
     5  22942   6  3325280   0 1055083102  98 types.CodeType
     6  20913   6  3011472   0 1058094574  99 function
     7   2302   1  2181296   0 1060275870  99 type
     8   4186   1  1972304   0 1062248174  99 dict (no owner)
     9   1000   0  1626000   0 1063874174  99 dict of module
<708 more rows. Type e.g. '_.more' to view.>
```

As a reminder:

- $10^3\ bytes\ =\ 1\ KB$
- $10^6\ bytes\ =\ 1\ MB$
- $10^9\ bytes\ =\ 1\ GB$

DISCLAIMER: This package works for python2. However, there is a version for python3 that works in exactly the same way, but it has to be installed through 'pip3 install guppy3'.

[Python] Managing your Python dependencies in the cluster

There are a myriad of ways your Python project can manage its dependencies. It is extremely dependent on the project which might be the best to proceed with. As a general rule of thumb, using Miniconda should be the preferred route for installation.
If you are checking out a git repository, look for the following:

- Environment.yml file (suitable for conda)
- Requirements.txt file (suitable for pip)
- README.md containing what libraries to install (can be installed through any route)
- Dockerfile (suitable for Docker-based installations)

In summary, the order of preference for the cluster is: (1) conda install (2) pip install (3) pip install -user

Conda

You can install Miniconda by visiting this link:
https://docs.conda.io/en/latest/miniconda.html#linux-installers

To avoid downloading and reuploading it is recommended to use a *wget* for downloading the executable. The exact place for installation can either be your home directory or your folder in the group drive.

You can install any version of Python needed for your project, independent of the Conda version that you are using.

If the repository you are using has a conda file, the installation of the required software can be as easy as applying:

conda env create -f environment.yml

However, it's well worth looking at the environment.yml for the following reasons:

- Change your conda environment name, which might conflict with env names that you have
- Checking if you have pip libraries.
- If you installed the conda on your home, you can change the prefix part in the environment.yml to install the env itself in the group share. This can be useful if you run out of the space on your home drive (8 GBs).

If you don't have any pip libraries the installation should happen without a problem.

If you have pip libraries, and your environment is on the group share, you can run into an error **Operation not permitted.** This is because pip tries to compile a dependency of one of your packages and asks for 777 rights, which you don't have in the group share.

There are two workarounds for this:

- If you have enough space in your home, install the conda environment there (by changing prefix)
- If you don't have enough space, you need to manually install all pip libraries using the command **pip install --user package_name**. There are many downsides to this approach, (1) it might turn out that you still don't have enough space (2) packages installed this way will be global to all your conda/pip environments, possibly ruining isolation/reproduction quality of your work.

Some workarounds if you don't have enough space in your /home/nfs/username:

- Delete files that you don't need
- Conda clean --all (clears conda cache)
- rm -r /home/nfs/username/.cache (clears all cache, including pip cache)

Pip

If you don't like conda for some reason, you can use pip. As far as I know, there are no advantages to pip apart from the fact that (1) some libraries are only available through PyPi (2) there are some Linux distros like ArchLinux, where conda is not supported. Note that if you have pip-only libraries, it's still a good idea to use conda.

There are many disadvantages, however:

(1) pip can only install Python dependencies and most scientific Python libraries need dependencies like gcc, cuda, cudnn.

This can be circumvented by using the modules in the cluster, i.e for loading CUDA, one would normally:

Module use /opt/insy/modulefiles
Module load cuda/10.0

This has to be repeated every time you log into the cluster and need this dependency, and in your job scripts too.

(2) pip has an inferior dependency solver compared to conda. Pip's dependency solver only looks at one library at a time, and tries to solve the dependencies for only that library. This can create weird issues, i.e if tensorflow requires a higher numpy version and you install another package which requires a lower Python version, then you could have an installation without any errors, until you find out that your code can't run because a numpy namespace is missing.

In theory, the pip installations are also simple, namely you just need to type:
Pip install -r requirements.txt

But you will likely run into many errors during the installation, which will need to be solved as you go:

- For isolation of the libraries, it is still recommended to use venv if you are using pip.
- Make sure that you install your venv on your home to avoid compilation problems.
- Make sure to check which CUDA, gcc version you need to and load it from the insy modulefiles.

Docker

Docker doesn't work in the cluster, but there is an alternative called Singularity. If you use it, please provide a guide here for other's benefit.

Sending multiple jobs simultaneously to the cluster: job-arrays.

In order to avoid creating a number of different '.sbatch' files in which you only change a few parameters, there is the possibility of using *job arrays*. It is a way to define variables that depend on the *TASK_ID* of your jobs, so you only have to send one job whose parameters will vary. Besides, you can define the number of jobs that you'd like to have running in parallel at the same time. Thus, you don't have to wait until everything finishes if you need to run an urgent simulation while something else is running. Or you can simply create multiple files for multiple datasets (or simulations) and have them all running at the same pace. You can see an example right below.

```
#!/bin/sh
#SBATCH --partition=general
#SBATCH --qos=short
```

```
#SBATCH --time=04:00:00
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=1
#SBATCH --mem=1G
#SBATCH --array=0-<NumberOfJobs-1>%<NumberOfJobsRunning>
#SBATCH --mail-type=ALL
#SBATCH --workdir=/tudelft.net/staff-bulk/ewi/insy/MMC/<YourFolder>

case $SLURM_ARRAY_TASK_ID in
   0) ARGS="--<var1> <val1_var1> --<var2> <val1_var2> …" ;;
   1) ARGS="--<var1> <val2_var1> --<var2> <val2_var2> …" ;;
   ...
esac

case $SLURM_ARRAY_TASK_ID in
   0) OUTPUT="Folder1/Folder2/File0.out" ;;
   1) OUTPUT="Folder1/Folder2/File1.out" ;;
   ...
esac


srun -u --output=${OUTPUT} python3.4 <PythonFile>.py ${ARGS}
```

DISCLAIMER: If you copy the code above, please make sure that the quotation marks are detected as such. If you see your jobs failing when sending them as a job-array and otherwise working, change them. If you use an app such as Notepad++, you should see a change of colour in the name of the file/path that's in quotation marks.

It doesn't change much from the basic structure, just a new variable '--array' that allows us to define how many jobs we want to run when we send the job and how many jobs we allow the cluster to run in parallel. That means that you could define more jobs that you actually want to run. For large sets of simulations, this code can be generated through python.
Keep in mind that both the memory and the number of CPUs affect each individual task, not the whole group of jobs.