

# Computer Set-Up Instructions

Welcome to the School! We're excited to have you!

Astronomy software is sometimes a little bit difficult to install and maintain. In order to make this as easy as possible on you, we have created a virtual machine for you to work with, which will have all of the software installed!

This document has some instructions to help you get started. Because you may all have different operating systems (e.g. Windows, Linux or MacOS), you might need different instructions for each of those, so **please make sure you follow the instructions for your operating system** at the steps below.

## What to do if you can't connect to your VM

Occasionally, the internet connection of the VM might break, which means you can't log into your VM via ssh. If this happens, you can fix it by following the step-by-step instructions below:

<https://docs.google.com/document/d/1qT3nwbVJgibs0oig7gYx35BqIFQgk1z9lUkWOflpTDE/edit?usp=sharing>

If this doesn't work, please find **Daniela**.

## Required Software

1. A browser. We have tested these instructions in the browsers below. If you have a different browser, please make sure to install one of those below, or we can't guarantee that everything will work as expected.
  - a. Firefox
  - b. Chrome
  - c. Safari
2. A command-line interface + ssh client: we will be often working through a command-line interface rather than click on things on the screen. In order to connect to the actual virtual computer you'll be using, you need to connect via a programme called ssh.
  - a. **Linux, MacOS:** you don't need to install anything. You can find a command-line interface under the name "Terminal" in Applications. Type `ssh` into the terminal after you open. It should be installed by default on both MacOS and Linux.
  - b. **Windows:** please install [Git for Windows](#) (instructions on website)
3. An XWindow server: this is a programme that can open windows (e.g. with plots) from your command line. We will be using a programme called X11
  - a. **Linux:** you should have X11 installed by default, so nothing to install
  - b. **MacOS:** The newest releases of MacOS might not have X11 installed by default (search for X11.app in Finder to find out). If not, please install [XQuartz](#) (instructions on that page)

- c. **Windows:** You won't have an X11 client installed. Please install [Xming](#) (instructions on page).

## Preparation

**Linux/MacOS:** You're ready to go!

**Windows:** in order to be able to open Windows from the command line interface, there are a couple of additional steps to follow.

1. Open GitBash, which you have installed with Git for Windows in Step 2 earlier.
2. Type the following commands

```
# press enter to submit each command
echo "export DISPLAY=localhost:0.0" >> $HOME/.bashrc
source $HOME/.bashrc
```

## Logging Into Your Virtual Machine

We will have a virtual machine set up for you to work on. A virtual machine is like a real computer, except it runs *within* another computer. In our case, we have set up a number of virtual computers on a large high-performance computing (HPC) environment. Setting these up for you means that you all get to work in the same operating system, with the same software installed, and this should hopefully minimize the time you need to spend getting things to run.

When you get to the school, the virtual machines will all be running. You will receive an IP address to your machine, which you'll be using for the duration of the school. All virtual machines are set up with the same user, called `ataschool`. The password to log in is also the same and will be written on the board. **Please don't share this with anyone outside the school!**

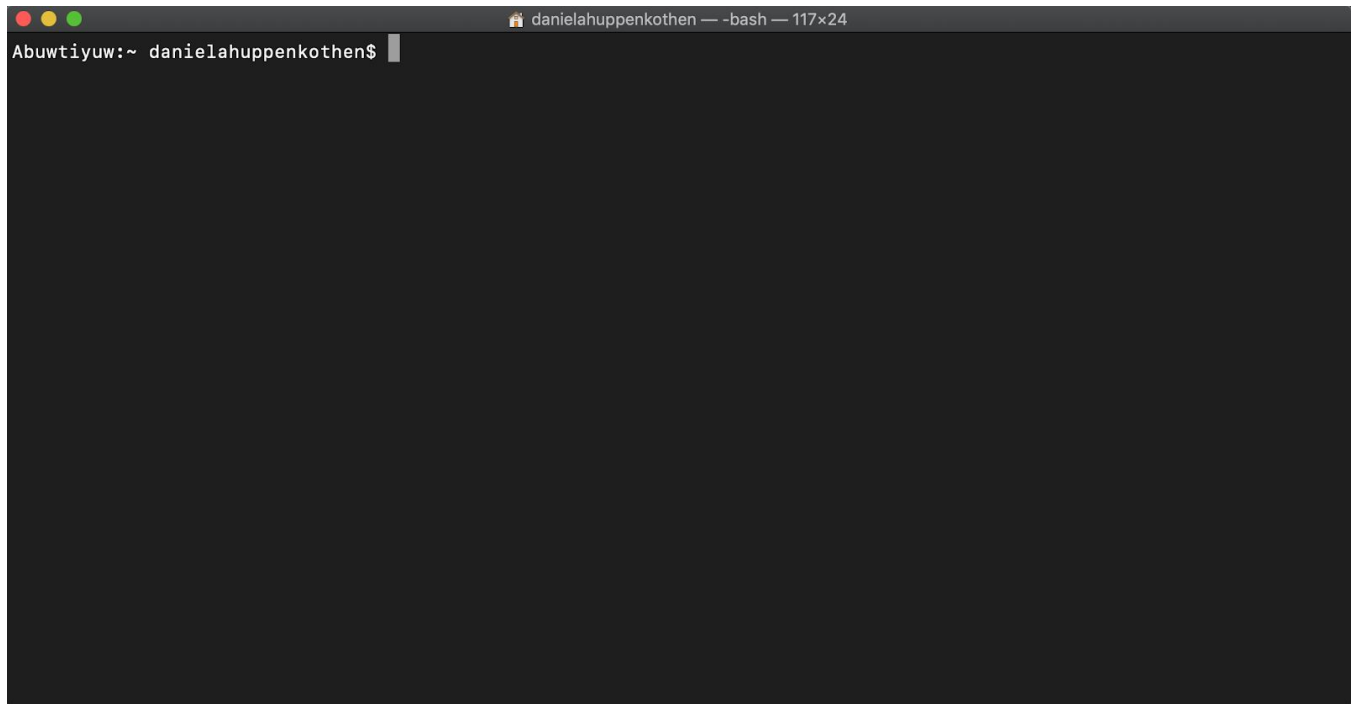
You can find out the IP address for your virtual machine using [this spreadsheet](#). If you're a **student**, please search for **your e-mail address**. If you're a member of the **LOC** or one of the **teachers**, **please claim one of the VMs marked "LOC" or "lecturer"**, respectively, and write your name in the field "Name or E-mail address". Please remember to remove your name from it when you leave the school or if you're no longer using the VM.

You may be used to interacting with a computer using your mouse and windows on the screen. However, the virtual machines we've set up for you are not physically in the same building with us! You'll be using your own computers (or those given to you) to connect to them, and instead of a mouse and windows, we are going to use something called the **command line** or **terminal**. This is essentially a prompt where you can type commands as text, which the computer will then execute.

On **Mac/Linux**, look for the Application called "Terminal" and open it.

On **Windows**, open GitBash.

You should see a window like this:



The blinking cursor is where you can type commands. If you're not familiar with the command line, please have a look at [this tutorial from Software Carpentry](#).

You can log into your virtual machine using a programme called ssh, using this command:

```
$ ssh ataschool@000.000.00.00
```

Important notes:

- Do not include the "\$" in front of the command when you copy it into your terminal. This is just a character used to denote that this command should go into the command line!
- Don't forget to replace the "000.000.00.00" with the IP address you were given to your virtual machine!!s

The command line will prompt you for the password, which you should see written on the board. Type the password when prompted. Congratulations! You should now be logged into your virtual machine! If this does not work, please ask for help!

**Exercise:** Take a look around! What folders are in your home directory? Can you navigate between them? Can you open a new text file in a text editor and edit it?

## Important Notes on Saving

Virtual machines are not like real computers in one very important respect: when we shut them down at the end of the school (or if we need to restart them during the school for technical reasons), everything you have changed within your home directory might **vanish**. This means that you should **never save important data or results within your home directory** (there's not a lot of space for you to do so anyway).

We've made a special storage disk for each of you. You can reach it by typing

```
$ cd /data
```

once you've logged into your virtual machine. Data and files stored on this disk **will be saved**. So whenever you generate code or data or plots you want to save, make sure to store them on that disk, rather than in your home directory!

## Setting up SSH Authentication

When you use ssh to log into your virtual machine, it asks you to type a password. Over the coming two weeks, you'll be logging into this machine a lot, so having to type that password every time can get annoying. Below are some instructions to set up ssh authentication, which will make it no longer necessary for you to type the password every time.

The ssh authentication uses two "passwords", called "keys": one that's on your computer, and private (i.e. don't share it with anyone!). And one that is public, which you can put on your virtual machine. Every time you use ssh, the VM will send the public key, which your computer will compare with your private key. If they match correctly, then you will be automatically logged into the VM without having to type your password.

The setup has three steps: (1) find (or generate) a public/private key pair on your computer, (2) activate the ssh-agent, (3) put your public key on the VM.

Check whether you have an active pair of SSH keys

In Windows, open Git Bash

In Linux or MacOS, open a Terminal.

First, we'll need to check whether you have a directory called .ssh. For this, in your home directory, type

```
$ cd .ssh
```

If this works, you have the directory you need! If not, you should see an error message

```
-bash: cd: .ssh: No such file or directory
```

If you see this message, create the directory with

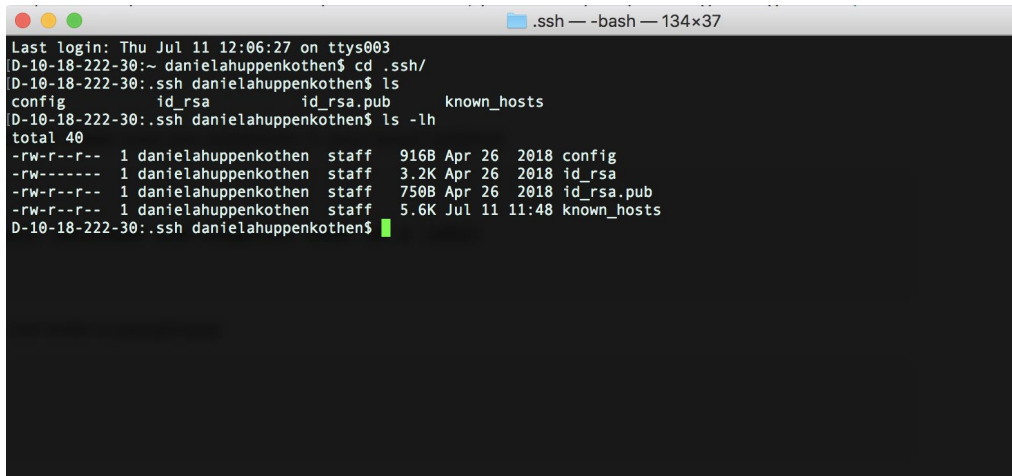
```
$ mkdir .ssh
```

Now let's see what's in this directory! If you've just created it, it'll be empty.

```
$ cd .ssh
```

```
$ ls -lh
```

The first command moves you into that directory, the second lists all the files that are present. Here's what this looks like on my computer:

A terminal window titled ".ssh -- -bash -- 134x37" showing the output of 'cd .ssh/' and 'ls -lh'. The output lists files: config, id\_rsa, id\_rsa.pub, and known\_hosts with their permissions, sizes, and timestamps.

```
Last login: Thu Jul 11 12:06:27 on ttys003
D-10-18-222-30:~ danielahuppenkothen$ cd .ssh/
D-10-18-222-30:~.ssh danielahuppenkothen$ ls
config      id_rsa      id_rsa.pub  known_hosts
D-10-18-222-30:~.ssh danielahuppenkothen$ ls -lh
total 40
-rw-r--r--  1 danielahuppenkothen  staff   916B Apr 26  2018 config
-rw-----  1 danielahuppenkothen  staff   3.2K Apr 26  2018 id_rsa
-rw-r--r--  1 danielahuppenkothen  staff   750B Apr 26  2018 id_rsa.pub
-rw-r--r--  1 danielahuppenkothen  staff   5.6K Jul 11 11:48 known_hosts
D-10-18-222-30:~.ssh danielahuppenkothen$
```

I have a public/private key pair for ssh, which are stored in the files

```
id_rsa (private key)
id_rsa.pub (public key)
```

On your system, they might also be called

```
id_dsa.pub
is_ecdsa.pub
id_ed25519.pub
```

If you have any of the above, skip the next step and go straight to “start ssh-agent” below. Otherwise read on!

## Generating the ssh key

If you don't have a public and private key, let's create one!

You can do this using a program called ssh-keygen:

```
$ ssh-keygen -t rsa -b 4096 -C "myemail@domain.com"
```

Make sure to substitute your own e-mail address in the command above! It might ask you to enter pass phrase. You can set a password here, or you can press Enter twice to continue without a password.

## Adding your key to the ssh-agent

We now need to add the key to the program that actually manages them in the background, called `ssh-agent`. First, let's start that program:

**On Linux, MacOS**, you can type the following

```
$ eval $(ssh-agent -s)
```

**On Windows** (in Git Bash), type the following command:

```
$ eval `ssh-agent`
```

In both cases, you need to add your ssh key to the agent:

```
$ ssh-add ~/.ssh/id_rsa
```

Note: if you have an existing private key with a different name (i.e. not "id\_rsa"), make sure to replace it with the private key that exists in your `.ssh` directory!

## Add your SSH key to the Virtual Machine

In the final step, we need to add the ssh key to the virtual machine, so it'll know to trust your computer.

You can do this with the following command:

```
$ cat ~/.ssh/id_rsa.pub | ssh ataschool@000.000.00.00 'cat >>
~/.ssh/authorized_keys'
```

Remember to replace the zeros in the IP address with the actual IP address of your VM! It will ask for your password this time, but hopefully, once you've completed these steps, you should be able to log into your VM without you having to type your password every time!

## Plotting (in Python)

Throughout the school, we will be running code (both for simulations and data) and will plot the results. Let's check that plotting works for everyone! We'll use the programming language Python to do that. When you make a plot on your own computer, it knows that there's a display attached to it and automatically forwards the relevant information to be displayed on your screen. However, your virtual machine does not have a screen! We need to tell the virtual machine that it has to send the relevant information to display a figure to the screen on your computer. Generally, we do this when we use ssh to connect, by adding a *command line option*:

```
$ ssh -Y ataschool@000.000.00.00
```

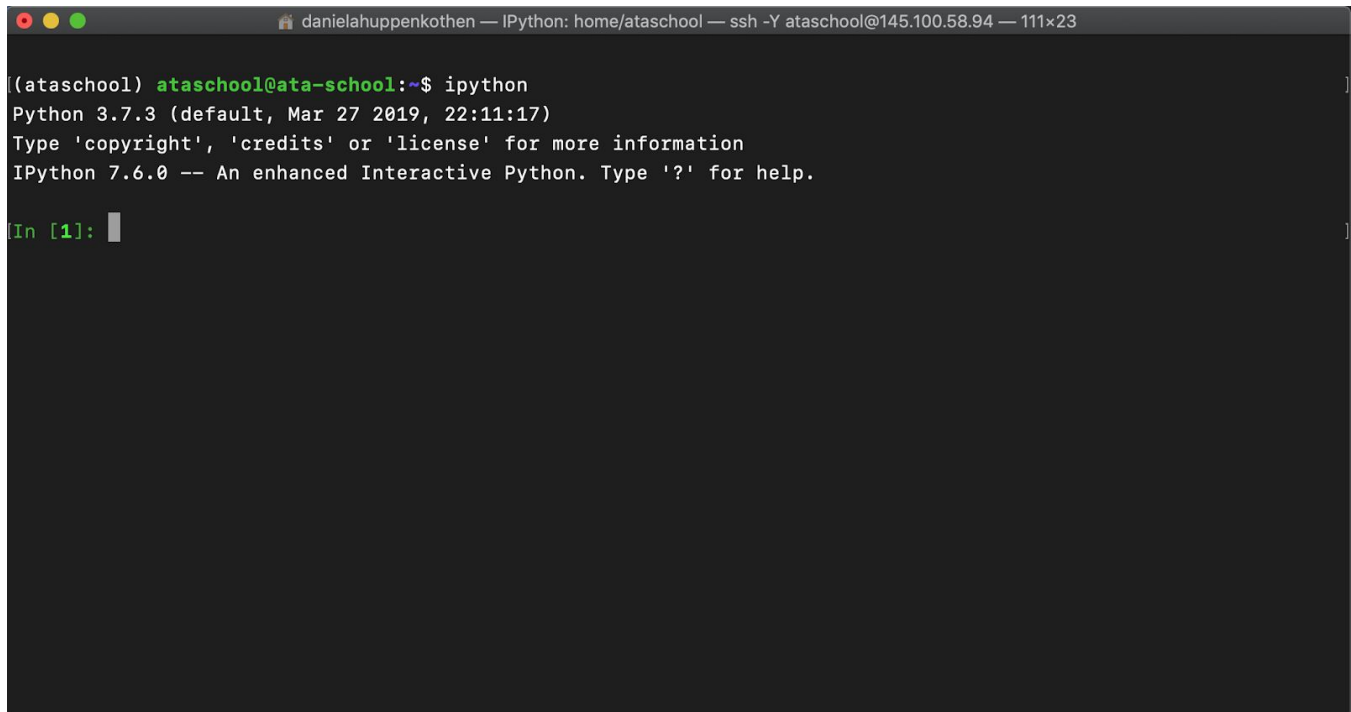
The “-Y” is called a flag, which tells the ssh programme to run with a special option. In this case, it lets ssh know that whenever it receives information that ought to be displayed on a screen that it should send it to your physical display.

**Exercise:** Close down your ssh connection and reconnect using the “-Y” flag as above.

Now we’re ready to start plotting. Let’s open an iPython terminal. This is basically a special command line that allows you to type in commands in Python:

```
$ ipython
```

Your terminal should look something like this:

A screenshot of a terminal window. The title bar at the top shows a home icon, the name 'danielahuppenkothen', and the command 'IPython: home/ataschool — ssh -Y ataschool@145.100.58.94 — 111x23'. The terminal content shows the command '(ataschool) ataschool@ata-school:~\$ ipython' being entered. The output shows 'Python 3.7.3 (default, Mar 27 2019, 22:11:17)', 'Type \'copyright\', \'credits\' or \'license\' for more information', and 'IPython 7.6.0 -- An enhanced Interactive Python. Type \'?\' for help.'. The prompt has changed to '[In [1]: ]' with a cursor.

As you can see, the command line interface has changed, and now says “In [1]: “ in front. This lets you know that iPython is running.

We’re now going to do three things: we’re going to import the libraries for numerical computing (numpy) and plotting (matplotlib), then we’re going to generate some test data points, and finally we’re going to plot them.

Here’s how you import libraries in python:

```
import numpy as np
import matplotlib.pyplot as plt
```

Now let’s generate some data points:

```
x = np.arange(100)
```

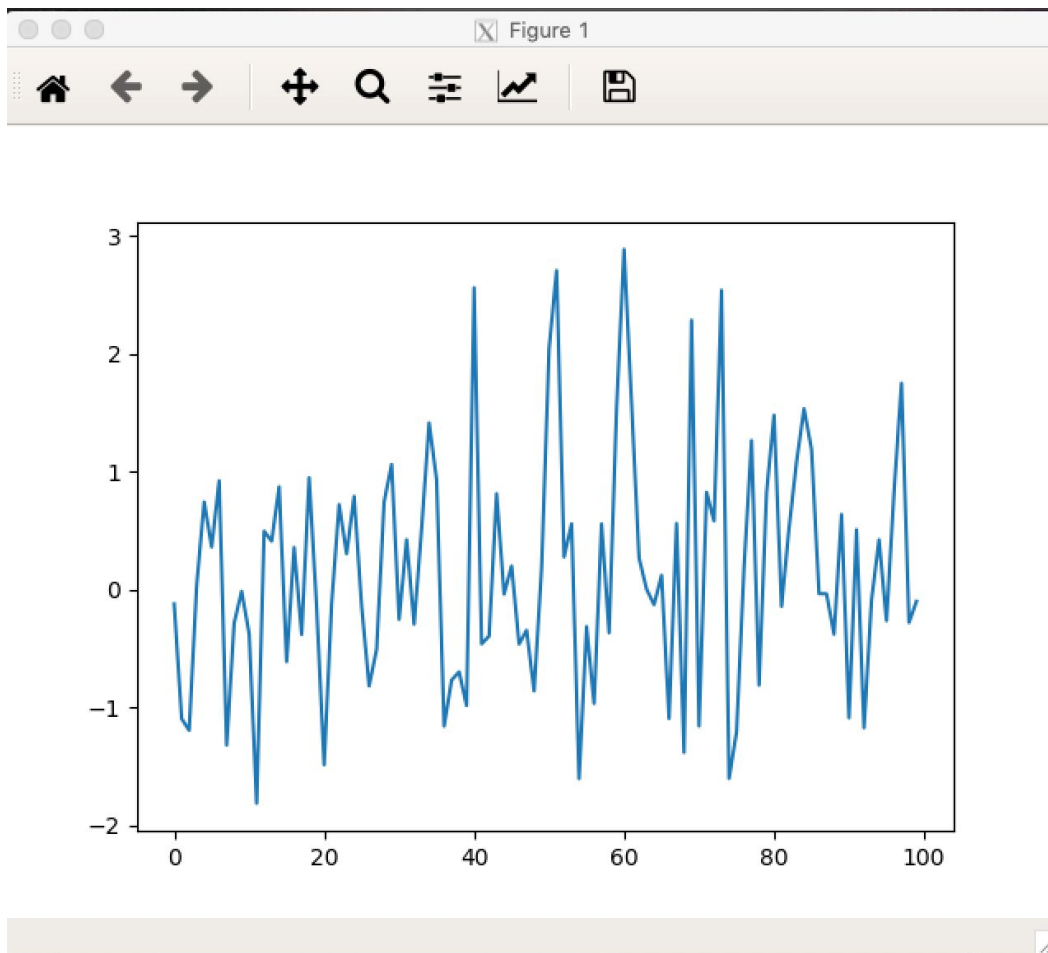
```
y = np.random.normal(size=100)
```

This generates a list of ordered integers going from 0 to 99, and stores them in an array “x”. It then generates one hundred random numbers from a normal distribution and stores them in an array “y”.

You’re now ready to plot these data points as a line:

```
plt.plot(x, y)
```

It might take a moment, but a new window should pop up displaying a wiggly line, something like this:



**If you do not see a plot, please ask for help!**

You can close the window again by clicking on the leftmost red button at the very top of the window (MacOS), or by typing

```
plt.close()
```

into your iPython terminal. To exit iPython, you can use the combination control-d, or you can type

```
exit
```



into the terminal.

## Running Jupyter Notebooks

Some of the school's exercises will use [Jupyter notebooks](#). These are nifty documents that are displayed in your browser, which allow you to combine text, images and code in the same document. Much like with plotting, we will be running these notebooks on the virtual machine, which won't know about the browser in your computer, so we'll need to tell it about your browser. Also like with plotting, we're going to use ssh to do this.

To run Jupyter notebooks, first start what's called a notebook server (this is the programme on your VM that will run the code).

On your VM, type the following command into your command line:

```
$ jupyter notebook --no-browser --port=8887 &
```

This will start a notebook server in the background.

Now open a new terminal **on your own computer (i.e. not in the VM)**, and type the following command

```
$ ssh -f -N -L localhost:8887:localhost:8887 ataschool@000.000.00.00
```

(nothing will happen after this step, don't be alarmed, keep reading!!)

Remember to replace the zeros with the IP address of your virtual machine. This last command sets up a connection between your computer and the virtual machine, and tells it to forward the important information to display Jupyter notebooks.

When you started the notebook server, it should have printed out some relevant information onto the screen, something like this:

```
advancingtheoastro — ataschool@ata-school: ~ — ssh ataschool@145.100.58.94
* Canonical Livepatch is available for installation.
- Reduce system reboots and improve kernel security. Activate at:
https://ubuntu.com/livepatch

227 packages can be updated.
0 updates are security updates.

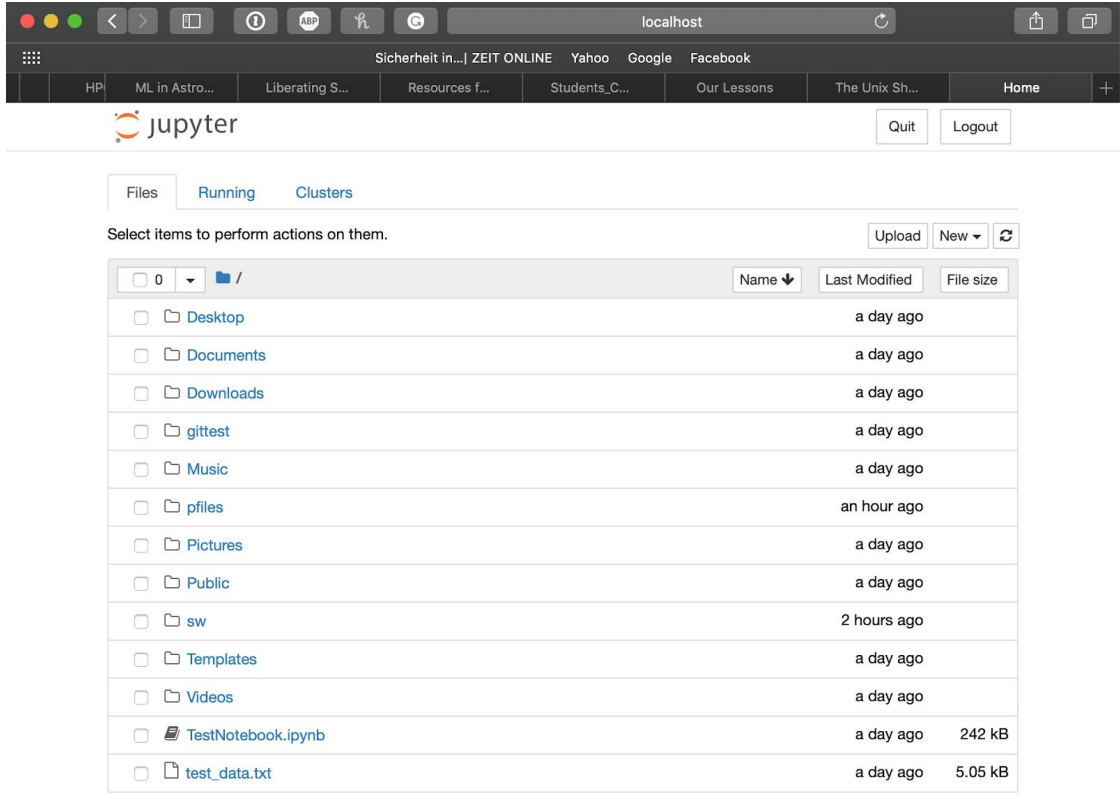
Your Hardware Enablement Stack (HWE) is supported until April 2023.
Last login: Sat Jul 6 23:40:01 2019 from 73.53.28.136
(ataschool) ataschool@ata-school:~$ jupyter notebook --no-browser --port=8887 &
[1] 3000
(ataschool) ataschool@ata-school:~$ [I 00:00:02.563 NotebookApp] Serving notebooks from local directory: /home/ataschool
[I 00:00:02.564 NotebookApp] The Jupyter Notebook is running at:
[I 00:00:02.564 NotebookApp] http://localhost:8887/?token=18ddfa0f352a115f6e523d8b29d54172bdcab97dff36ff9
[I 00:00:02.564 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 00:00:02.572 NotebookApp]

To access the notebook, open this file in a browser:
file:///home/ataschool/.local/share/jupyter/runtime/nbserver-3000-open.html
Or copy and paste one of these URLs:
http://localhost:8887/?token=18ddfa0f352a115f6e523d8b29d54172bdcab97dff36ff9
(ataschool) ataschool@ata-school:~$ [I 00:00:49.374 NotebookApp] 302 GET /?token=18ddfa0f352a115f6e523d8b29d54172bdcab97dff36ff9 (127.0.0.1) 2.36ms
[W 00:00:53.522 NotebookApp] 404 GET /apple-touch-icon-precomposed.png (127.0.0.1) 17.54ms referer=None
[W 00:00:53.752 NotebookApp] 404 GET /apple-touch-icon.png (127.0.0.1) 2.41ms referer=None
```

From your own terminal, copy the line **below** the one that starts with “The Jupyter Notebook is running at:” into a browser. Here, this line starts with:

```
http://localhost:8887/?token=18[...]
```

You should see something like this:



Click on the entry that says **TestNotebook.ipynb**. To execute the code, click on the first line, then press shift-enter on your keyboard. Do this to all lines until you see a plot. If you run into any trouble, please ask for help!

## Shutting down Jupyter Notebooks

After you finish the job, you need to shut down the Jupyter notebook and release the port (8887 in the previous example).

You can shut down the Jupyter notebook by Ctrl+C on your VM terminal. However, sometimes it would still be running after then. You can check it with the command below:

```
$ jupyter notebook list
```

If you can find the port you used in the list of “currently running servers.” ,

```
(ataschool) ataschool@ata-school:~$ jupyter notebook list
Currently running servers:
http://localhost:8001/?token=33ab7cf070674ff1758164b26200c88d4e191cd9b4753923 :: /home/ataschool
http://localhost:8887/?token=862381055ed9e54bf07fc4e2c712417547e720d54a02119e :: /home/ataschool
```

you can force to shut down the notebook by the command:

```
$ jupyter notebook stop 8887
```

**On your own computer**, you also need to release the port to avoid the overuse of ports. From your own terminal (not VM), you can type the command below to check the ssh port which is currently occupied:

```
$ ps -Afl | grep ssh
```

You can release the port by killing the ssh process in background by the command:

```
$ killall ssh
```

# Cartesius

## Getting onto Cartesius

First let's put our previously generated ssh key on cartesius. Navigate to

<https://portal.surfsara.nl/password>

Where you will be asked to enter your username and preliminary password (came to you via mail from SURFsara Helpdesk). You should generate a new password and accept the usage agreement.

Afterwards you can click on 'Manage ssh keys'. Upload the public part of the ssh key, it is probably called `id_rsh.pub`. In any case, it ends with `.pub`. You can simply copy-paste the contents. Make sure you have the private counterpart on the VM from which you want to connect. For example:

```
scp ~/.ssh/id_rsa ataschool@000.000.00.00:~/.ssh/.
```

Here `000.000.00.00` should be replaced with the IP of **your** VM, the same as you use for login to the VM itself.

(in general it is actually not recommended to copy private keys, so you could also generate a new pair directly on the VM and upload the `.pub` part from that without further copying)

Now you should be able to connect to the Cartesius machine:

```
ssh -X username@cartesius.surfsara.nl
```

Where username stands for the name given to you from SURFsara Helpdesk.

There is not much in your folder yet, to get started, proceed to the section **On Cartesius** below.

## On Cartesius

How to get started running the codes.

1. copy the entire contents of the folder `/scratch-shared/ata2019/` to your own home directory. It has all the codes you need:

```
cp -r /scratch-shared/ata2019/* $HOME
```

You only need to do this once. From now on, you'll be making changes to the code you have copied above.

2. You need to make a choice regarding compilers and parallel environment of the cluster. This is done with the module infrastructure. Simply type:

```
module load openmpi/gnu/mt/ilp64/2.0.4.3
```

```
module load openmpi/gnu/mt/ilp64/2.0.4.3
```

For GRTRANS you also need python, so load the module with:

```
module load python
```

3. Your simulations will be run on the 'scratch-directory' of the cluster. This is a high-performance parallel file-system which ensures all cores can write into the same file at the same time, neat!

So make a directory for your simulations on the scratch directory like so:

```
mkdir /scratch-shared/$USER
```

4. First, change to the code directory:

```
cd $HOME/codes/harmpi
```

5. Now, choose the problem. In this example, we will start with a two-dimensional accretion problem. To select it, change line 98 in `decs.h` to the following:

```
#define WHICHPROBLEM TORUS_PROBLEM
```

6. Change the resolution to match the following (`decs.h`, lines 165-167):

```
#elif WHICHPROBLEM == TORUS_PROBLEM
#define N1          (64)
#define N2          (64)
#define N3          (1)
```

This selects a resolution of 64x64x1 cells per MPI process. Later on, we will choose the number of MPI processes (in the submission script, see below), which will determine the total resolution.

7. Since we want to run simulations on a fast filesystem, we need to copy the code to `/scratch-shared/$USER`. It is a good idea to make a new sub-directory for every run, for example:

```
cd /scratch-shared/$USER/
mkdir torus2D
cd torus2D
```

We suggest that you copy the entire `harmpi` folder to the run directory, so you have all the source code next to your data:

```
cp -r ~/codes/harmpi .
```

8. To compile HARMPI, type:

```
cd harmpi
make clean && make
```

(if this fails, you probably don't have the compiler modules loaded, go back to 2 and try 8 again)

you should get loads of output on the terminal and upon typing:

```
ls
```

you see a new file called 'harm'. Change back to the run directory:

```
cd ..
```

9. To run a simulation, we need to tell the cluster some parameters: mostly how many cores and for how long to run the simulation (in 'Wall-clock-time'). This is done via a 'batch-system'. The one used on Cartesius is called 'slurm'. There is some documentation with the particulars concerning the Cartesius on the site of surfSara: <https://userinfo.surfsara.nl/systems/cartesius/usage/batch-usage> . You need to check it if you run into problems.

The bare essentials are these six lines:

```
#!/bin/bash
#SBATCH -t 1:00:00
#SBATCH -n 16
#SBATCH -J torus2D
##SBATCH --reservation=ata_school_XX #uncomment this line to use
reservation, see Sec. 10 below
module load openmpi/gnu/mt/ilp64/2.0.4.3
srun ./harmpi/harm 4 4 1
```

This tells Cartesius to do the following:

- a) Run the simulation for one hour (1:00:00)
- b) Use 16 cores to do so
- c) Call the job 'torus2D' so you can easily identify it in the queue (see below)
- d) use the reserved nodes on the 18th of July (see next section)
- e) run the executable 'harm' with the tiling 4 4 1 (see [harmpi tutorial](#), specific to harmpi)

In the folder next to the executable 'harm', make a new file called 'job.sh' with the above six lines.

10. You can get your job to start faster on Monday-Friday, 9:30-19:30. The cluster has reserved nodes for the school during these times. In your job-file, make sure to include the appropriate line

```
#SBATCH --reservation=ata_school_XX
```

In the #SBATCH part of the file. Outside from the times specified below (for example over the weekend or when you want to run on more than the reserved nodes), you should remove the reservation line and wait in the normal queue.

18th of July: 9:30 - 19:30: reservation name "ata\_school\_18": students should run jobs including "--reservation=ata\_school\_18"

19th of July: 9:30 - 19:30: reservation name "ata\_school\_19": students should run jobs including "--reservation=ata\_school\_19"

20th of July: 14:00 - 18:00: reservation name "ata\_school\_20": students should run jobs including "--reservation=ata\_school\_20"

22nd of July: 9:30 - 19:30: reservation name "ata\_school\_22": students should run jobs including "--reservation=ata\_school\_22"

...

26th of July: 9:30 - 19:30: reservation name "ata\_school\_26": students should run jobs including "--reservation=ata\_school\_26"

Here is an example for July 19th, 2019:

```
#!/bin/bash
#SBATCH -t 1:00:00
#SBATCH -n 16
#SBATCH -J torus2D
#SBATCH --reservation=ata_school_19
module load openmpi/gnu/mt/ilp64/2.0.4.3
srun ./harmpi/harm 4 4 1
```

11. Now we want to submit the job to the queueing system. The queueing system (slurm) makes sure everyone gets their turn to run on the cluster also when it is full at the moment. Type:

```
sbatch job.sh
```

and your job will be submitted to Cartesius.

12. To check the status of your run, type:

```
squeue -u $USER
```

The output could look something like this:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES
NODELIST (REASON)						
6438418	normal	torus2D	ccurs000	PD	0:00	1

(Priority)

Which means we are still waiting for free nodes. You see here also the job-name we chose above: 'torus2D'.

Once the simulation started running, it will look like this:

JOBID	PARTITION	NAME	USER	ST	TIME	NODES
NODELIST (REASON)						
6438428	normal	torus2D	ccurs000	R	0:42	1

tcn1321

You can cancel a job by typing

```
scancel JOBID
```

Where jobid in this case would be 6438428.

13. Output will be produced as the simulation progresses and for harmpi, new folders called 'dumps' and 'images' appear. Within them you find data to analyze further (see harmpi tutorial). Next to these, Cartesius will generate a file called 'slurm-YYYYYY.out' (and possibly one called 'slurm-YYYYYY.err' when errors occur). Here YYYYYY stands for the JOBID in the list above. This file is frequently updated with messages from thse code, so you can monitor its progress.

14. Now you just need to wait for the job to complete and download the data (for example via rsync, see above) to see what you got.

15. In the meantime, you can keep checking on the job in two ways:

15.1. You can look at the code output by

```
less slurm-JOBID.out
```

15.2 You can take a look at the files the code produced by typing:

```
ls dumps
```

15.3 Once the run reached time around  $t \sim 500$ , you can start looking at the data. To keep the cluster people happy, follow the steps below, to copy the simulation results to the VM.

16. What else you can do? Try out the HARMPI tutorial:

<https://github.com/atckekho/harmpi/blob/master/tutorial.md>

## Transferring data between VM and Cartesius

To copy data back to your VM, use rsync:

```
rsync -Pav username@cartesius.surfsara.nl:/scratch-shared/username/*  
/data/.
```

will sync all data under the simulation directory (you are supposed to make a folder for yourself `/scratch-shared/username`). As always, `username` should be replaced with your actual username.

To reduce the load on the network, please sync only the files which you are interested in. Its best to make sub-folders under `/data` corresponding to the ones on the `/scratch-shared` filesystem on Cartesius, so you don't loose the overview.