Torch Quick Start Guide
Last updated: September 21, 2020

Contents

1 Introduction **2**

2 Quick Start **2**
2.1 Prerequisites . . . 2
2.2 Environment setup (Josh thesis Appendix A.1) . . . 3
2.3 Install Torch . . . 4
2.4 Build FLASH 4.5 (Josh thesis Appendix A.2 + FLASH docs) . . . 4
2.5 Configure AMUSE (Josh thesis Appendix A.3 + AMUSE docs) . . . 4
2.6 Run a Torch simulation (Josh thesis Appendix B) . . . 5
2.7 Look at the output . . . 6
2.8 Next steps . . . 7

3 Caveats and Known Issues **8**

4 Help! **8**
4.1 General checks . . . 8
4.2 FLASH setup/compile problems . . . 9
4.3 AMUSE configure/make problems . . . 9
4.4 Runtime and MPI problems . . . 9
4.5 Navigating the source code . . . 10

A Torch component references **10**

B Notes on build process for specific clusters **10**
B.1 Cartesius - General . . . 10
B.2 Cartesius - Refactor . . . 11

1 Introduction

Torch is a star formation simulation code with magnetohydrodynamics (FLASH), radiative transfer (Fervent), self-gravity & sinks, star particles, feedback (stellar winds, SNe), stellar evolution tracks (SeBa), and N-body dynamics (AMUSE). Torch features and add-ons are listed in Figure 1, and references for the major components are in Appendix A.

You can find a high-level overview of the code in:

- Wall+ 2019, arXiv:1901.01132
- Wall 2019, Ph.D thesis

Torch comprises (1) interface code that plugs into both FLASH and AMUSE, and (2) a good number of add-on code units for FLASH. To use Torch, you install interface code into both FLASH and AMUSE, compile FLASH alone, and then compile an AMUSE "worker" binary that enables FLASH to be called from AMUSE python scripts.

Flash (AMUSE interface)		**ph4 / Multiples / SeBa**
MHD (Fryxell+2000)	Atomic cooling (Hill+2012)	N-body (McMillian)
Rad Trans (Bacynzski+2015)	Mol cool (Neufield+1996)	Binary formation (McMillan+)
Winds (Markova+2008,Vink+2000, Kudritzski+2000)	Dust<->Gas (Hollenbach+1989)	Binary dynamics (McMillan+)
SN (Simpson+2016)	Background FUV (Weingartner+2001)	Binary accretion (PZ+1996)
Star formation (Sormani+2017)	Cosmic rays (Galli+2015)	Stellar flux (Lanz+2003)
Ionization fraction solver	Local gas extinction (Banerjee+2006)	SE (PZ+1996)
Ionization heating (Bacynzski+2015)	Radiation momentum	Stellar mass loss rate (other than OB winds) (PZ+1996)
FUV local stellar heating (Weingartner+2001)	EUV on dust (Draine 2011)	

Figure 1. Adapted from Wall+ 2018, MODEST conference proceedings

2 Quick Start

Torch has many moving parts. You'll likely have to adjust the procedure below for your personal setup. In any case, don't despair! Torch has been successfully built and run on a variety of systems.

We assume that you're using (1) the bash shell, and (2) conda for Python package management. But, the setup procedure should work for any shell and Python package manager with some adjustments.

2.1 Prerequisites

- FLASH 4.5 (support for 4.6.2 is coming soon): `http://flash.uchicago.edu/site/flashcode`
- AMUSE: `https://github.com/amusecode/amuse`
- Torch: `https://bitbucket.org/torch-sf/torch`
- Conda / Python 2.7
- Python 3.X (for Torch `refactor` only)
- MPI

- HDF5
- git, wget

To download FLASH, you'll need to register, which takes a few days.

To download AMUSE, do:

```
cd your/code/directory
git clone https://github.com/amusecode/amuse.git
```

To download Torch:

```
cd your/code/directory
git clone https://bitbucket.org/torch-sf/torch.git
```

MPI and HDF5 may be pre-installed if you are using a cluster. If not, you will have to install these packages from source. Be sure to use the same compiler set that you will use for the rest of this Torch build.

Torch is known to work with OpenMPI 2.x, 3.x, and 4.x; HDF5 1.8.x and 1.10.x.

`git` and `wget`, if not already provided by your system, can be installed in your custom conda environment (see next section) with `conda install git wget`.

2.2 Environment setup (Josh thesis Appendix A.1)

Start a new shell session. You may want to check your `.bashrc` for any old MPI or HDF5 configuration that could conflict with Torch setup – module load commands, environment variable exports, et cetera – and comment them out.

Load MPI and HDF5 into your environment. If you're using shared cluster builds, do something like:

```
module load OpenMPI/x.y.z/blah # for clusters with lmod
module load HDF5/x.y.z/blah
```

Else, if you built MPI or HDF5 from scratch, set your environment manually. You can check what environment variables might be needed by calling `module show` on existing HDF5 or MPI builds on your cluster. An example for one cluster is below:

```
export MPI_HOME=/path/to/openmpi-x.y.z
export MPI_RUN=/path/to/openmpi-x.y.z/bin/mpirun
export PATH=$MPI_HOME/bin:$PATH
export MANPATH=$MPI_HOME/share/man:$MANPATH
export LD_RUN_PATH=$MPI_HOME/lib:$LD_RUN_PATH
export LD_LIBRARY_PATH=$MPI_HOME/lib:$LD_LIBRARY_PATH
export CPATH=$MPI_HOME/include:$CPATH

export HDF5_HOME=/path/to/hdf5-x.y.z
export HDF5DIR=/path/to/hdf5-x.y.z/lib
export HDF5INCLUDE=/path/to/hdf5-x.y.z/include
export HDF5LIB="hdf5_fortran -lhdf5"
export PATH=${HDF5_HOME}/bin:$PATH
export LD_LIBRARY_PATH=${HDF5_HOME}/lib:$LD_LIBRARY_PATH
export LD_RUN_PATH=${HDF5_HOME}/lib:$LD_RUN_PATH
export LIBRARY_PATH=${HDF5_HOME}/lib:$LIBRARY_PATH
export C_INCLUDE_PATH=${HDF5_HOME}/include:$C_INCLUDE_PATH
export CPLUS_INCLUDE_PATH=${HDF5_HOME}/include:$CPLUS_INCLUDE_PATH
```

Setup a clean, isolated Python environment for AMUSE:

```
mkdir {your/conda_env/directory}
conda create --prefix {your/conda_env/directory}/{env_name} python=2.7
source activate {your/conda_env/directory}/{env_name} # give absolute path so conda can find your env
```

Install some Python packages required by AMUSE:

```
conda install ipython matplotlib numpy scipy docutils gsl gmp h5py mpfr nose
conda clean --all
pip install --no-cache-dir mpi4py
```

2.3 Install Torch

Declare the following environment variables:

```
export AMUSE_DIR=/path/to/amuse
export FLASH_DIR=/path/to/FLASH4.5
export TORCH_DIR=/path/to/torch
```

Then, cd to the `torch` repo and do:

```
./install.sh
```

This will copy interface files from `torch` to both the AMUSE and FLASH repos.

2.4 Build FLASH 4.5 (Josh thesis Appendix A.2 + FLASH docs)

Let's `cd` to the FLASH4.5 directory and setup a turbulent sphere collapse problem.

First, create a `Makefile.h` in `FLASH4.5/sites/mysitedir`. We recommend that you fork an example file from `sites/Prototypes`. You may have done this already while traversing the FLASH manual's quick start.

If you used the example `Makefile.h` for the Netherlands supercomputer Cartesius in Josh's thesis, you may need to (1) remove `-march=core-avx2` flags throughout, and/or (2) remove `-lhdf5_fortran` from the `LIB_HDF5` variable declaration.

Then, issue the setup command (update "mysitedir"):

```
./setup Cube -a -3d +amuseUSM +amuseMG +rayPE +HandCnew +amuseSinksAndStars +amuseWind +enerinj \
   +supportPPMupwind +pm4dev -maxblocks=100 +cube16 \
   --site={mysitedir}
```

Compile FLASH. The `flash4` binary is not needed for the AMUSE/FLASH bridge, but compiling FLASH alone helps to catch errors and save time in the AMUSE compile step.

```
cd object
make -j
```

2.5 Configure AMUSE (Josh thesis Appendix A.3 + AMUSE docs)

Now, `cd` to the AMUSE repository. You can follow the AMUSE docs up to the `make` step, but take a look at these notes for extra information.

If you used a local conda environment following this guide, you'll probably need to tell `configure` where GMP, MPFR, and GSL reside. Often, you'll have to tell `configure` where HDF5 and FFTW live, too.

Here is an example `configure` call for the Netherlands supercomputer Cartesius (early 2019):

```
./configure \
   --with-fftw=/hpc/eb/RedHatEnterpriseServer7/FFTW/3.3.8-gompi-2018b \
   --with-hdf5=/hpc/sw/hdf5-1.8.12-intel-seq \
   --with-gmp={your/conda_env/directory}/{env_name} \
   --with-mpfr={your/conda_env/directory}/{env_name} \
   --with-gsl-prefix={your/conda_env/directory}/{env_name}
```

Library path errors are rather common at this step. FFTW and NetCDF library errors are OK, but HDF5 library and/or header errors are not OK. The STDOUT from a successful configure build should include (valid for AMUSE v12.0.0, but changed by AMUSE commit 9a3b134e)

```
checking hdf5.h usability... yes
checking hdf5.h presence... yes
checking for hdf5.h... yes
checking for H5_init_library in -lhdf5... yes
```

See Section 4.3 for more troubleshooting suggestions.

You may need to manually edit the files:

```
config.mk
src/amuse/community/flash/Makefile
```

to select the correct compilers and libraries.

Now, you can make the AMUSE `flash_worker`, which is basically an AMUSE-hooked replacement for the normal `flash4` executable. The standard `make` takes 10–20 minutes and builds all the codes included in AMUSE. To request only the codes needed for Torch, do:

```
make framework
make flash.code
make kepler.code
make ph4.code
make seba.code
make smalln.code
```

A successful make will create `flash_worker` in `src/amuse/community/flash/`.

2.6 Run a Torch simulation (Josh thesis Appendix B)

You are now ready to run a Torch simulation! We will simulate a $3 \times 10^3 M_\odot$, 5 pc radius gas sphere in ± 7 pc cubic domain with outflow boundary conditions. In about a free-fall time (≈ 2 Myr), the cloud will collapse under its own gravity, create a sink particle, and begin forming stars.

Choose a directory for your simulation and `cd` there. Then do:

```
export PYTHONPATH=$PYTHONPATH:$TORCH_DIR
export PYTHONPATH=$PYTHONPATH:$AMUSE_DIR/test
export PYTHONPATH=$PYTHONPATH:$AMUSE_DIR/src
ln -s $TORCH_DIR/cool.dat
ln -s $TORCH_DIR/cube128

cp $TORCH_DIR/bridge_multiples.py       bridge_multiples.py
cp $TORCH_DIR/flash.par.turbsph_standard flash.par
mkdir data
```

The file `bridge_multiples.py` is the heart of the simulation. It performs the split time-evolution with both FLASH and the n-body integrator ph4, handles stellar evolution, and lots more.

The simulation should take 1–10 minutes to reach $t_{\rm max} = 2$ Myr $= 6.31 \times 10^{13}$ s. It will use 7 threads: 2 for FLASH, 1 for n-body integration (ph4), 2 for n-body binary interactions (multiples), 1 for stellar evolution, and 1 for AMUSE itself. At around 1 Myr, a sink particle will form and begin creating stars. One or a few $\geq 7 M_\odot$ stars may begin blowing a wind, so the time step may drop; you may wish to end the simulation early.

To run the simulation interactively, execute the command:

```
mpiexec -n 1 python bridge_multiples.py
```

On a compute cluster with Slurm, copy `run.sh` from the `torch` repository to your simulation directory. Edit the `SBATCH` options as needed. Then do:

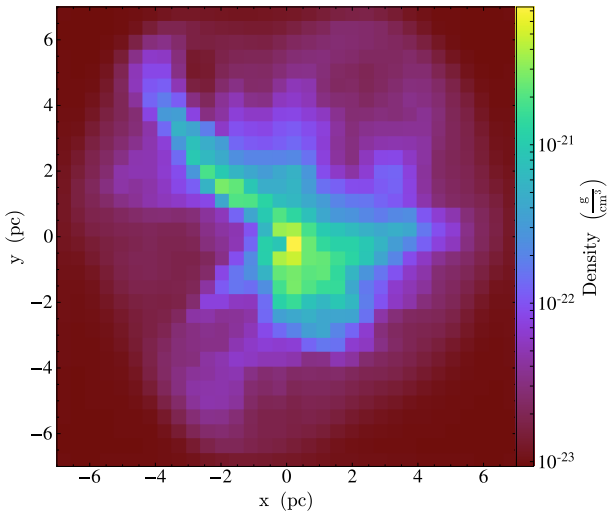

Figure 2. Density in x-y plane.

```
sbatch run.sh
```

If you increase the number of tasks requested, `bridge_multiples.py` will automatically assign more threads (workers) to FLASH.

Log files will be dumped in your current working directory, and simulation outputs will be written to the `data/` directory. If you re-start a simulation, Torch (FLASH) will overwrite existing data, but append to (some) existing logs. You may want to rename or delete your old logs to help keep track of your runs.

2.7 Look at the output

Let's have a quick look at this gestating star cluster. The example plots here were created with the `yt` package, version 3.4.1. To learn more about `yt`, visit http://yt-project.org/. To install `yt`:

```
conda install yt
conda clean --all
```

We can first look at density-slice plots in the three coordinate planes (x-y, x-z, y-z). At the command line, call:

```
yt plot data/turbsph_forced_hdf5_plt_cnt_0000
```

This will create a new directory `frames/` with three plots similar to Figure 2.

How about the particles? Let's see how they're distributed, and inspect some of their other properties. In a Python session:

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

ds = yt.load('data/turbsph_hdf5_plt_cnt_0018')

ad = ds.all_data()
ppx = ad['all', 'particle_posx'].to('pc').value
ppy = ad['all', 'particle_posy'].to('pc').value
ppm = ad['all', 'particle_mass'].to('Msun').value
windy = ad['all', 'particle_dmdt'].value
windy[windy > 0] = 1
```

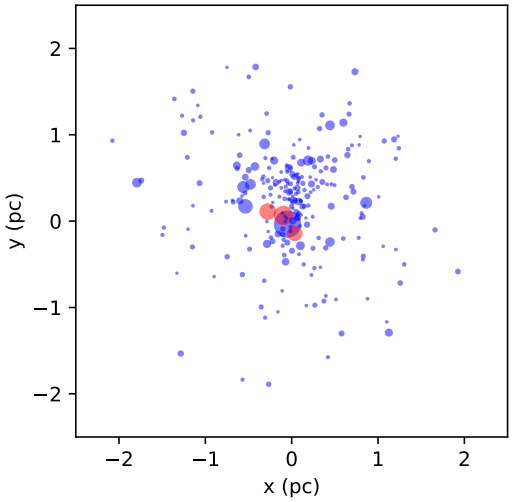

Figure 3. Star particles. Size is proportional to mass; red particles have $\mathrm{d}m/\mathrm{d}t > 0$ (wind-blowing stars), blue particles have $\mathrm{d}m/\mathrm{d}t = 0$ (quiescent).

```
plt.scatter(ppx, ppy, s=ppm*6, c=windy, cmap='bwr', alpha=0.5)
plt.gca().set_aspect('equal')
plt.xlim(-2.5, +2.5)
plt.ylim(-2.5, +2.5)
plt.xlabel('x (pc)')
plt.ylabel('y (pc)')
plt.show()
```

In this particular simulation, three stars are massive enough to blow a wind (Figure 3).

2.8 Next steps

To tweak the simulation parameters, you will need to edit `flash.par` and `bridge_multiples.py`. These two files specify most of the configuration options for the simulation.

Torch comes packaged with a few simulations in `FLASH4.5/source/Simulation/SimulationMain/`

```
Cube
EnergyInjection
StratBox
```

which provide initial conditions for (1) a turbulent gas sphere initialized from a 128^3 array, (2) a uniform medium for testing a stellar wind or supernova, and (3) a planar gas layer in a periodic domain, driven by random thermal supernova explosions.

It's often useful to reduce the number of Torch features for study or debugging. Below are setup calls showing some reduced feature sets:

Full Torch code with AMUSE/FLASH coupling.

```
./setup Cube -a -3d +amuseUSM +amuseMG +rayPE +HandCnew +amuseSinksAndStars \
   +amuseWind +enerinj +supportPPMupwind +pm4dev -maxblocks=100 +cube16
```

FLASH-only simulation, no AMUSE coupling. Sinks will work, but no stars will form.

```
./setup Cube -a -3d +usm +gravMgrid +rayPE +HandCnew +amuseSinksAndStars \
   +amuseWind +enerinj +supportPPMupwind +pm4dev -maxblocks=100 +cube16
```

FLASH-only simulation, no ray-tracing or sinks/stars. Only self-gravity and heating/cooling.

```
./setup Cube -a -3d +usm +gravMgrid +HandCnew \
   +supportPPMupwind +pm4dev -maxblocks=100 +cube16
```

FLASH-only simulation, only MHD.

```
./setup Cube -a -3d +usm +supportPPMupwind +pm4dev -maxblocks=100 +cube16
```

Note that `+rayPE` generally requires `+amuseSinksAndStars`. These setup flags are aliases for various FLASH units; the flags are defined in `FLASH4.5/bin/setup_shortcuts.txt`.

Don't forget that after a fresh setup + compile of FLASH, you will need to recompile the AMUSE worker too.

3 Caveats and Known Issues

Unphysical hot/empty zones are a common problem in finite volume codes, and are readily generated by under-resolved supernovae and stellar winds in Torch. To combat such zones, a few tactics include:

- Lower CFL
- Restart with more diffusive hydro solver parameters, temporarily
- Manually smooth out extremely sharp gradients
- Enable or increase artificial shock viscosity.
- Use the time-step limiter for positive-definite cell values (already enabled in the example Torch flash.par files). This is controlled by the FLASH runtime parameter `dr_usePosdefComputeDt`, among others.

See http://flash.uchicago.edu/pipermail/flash-users/2019-May/002908.html for a nice explanation by Sasha Tchekhovskoy.

In the FLASH unsplit solver, the logic for the hybrid Riemann solver is slightly altered; see the `torch` file: `src/flash/source/phys`

Torch does not support the use of BHTree with AMUSE. Code exists to hook FLASH/BHTree and AMUSE together, but it needs a small update and some testing to work.

Several add-on ParticleInitialization, ParticleMapping, etc. units are provided with Torch. These were created to trace supernova and superbubble ejecta in stratified box simulations run by Ibanez-Mejia et al. (2017). However, these have not been tested with the Torch sink/star framework and probably will not work correctly as is.

The module `source/physics/sourceTerms/GridInject` is in development and not recommended for use yet.

4 Help!

4.1 General checks

- Are all packages built using the same compiler set?
- Check your environment variables. Is PATH pointing somewhere it shouldn't? Are two different installs/versions of a module visible in PATH?
- Anaconda or Conda may provide its own HDF5. If you are using this HDF5, great, but if you are using a system HDF5 install, not so great. Check your PATH and make sure that the desired system HDF5 comes before the Anaconda/Conda HDF5.
- Check that FLASH, AMUSE, and mpi4py are using the same HDF5 and MPI libraries.
- If you had to swap MPI modules and reinstall `mpi4py` at any point, `pip install` needs the `--no-cache-dir` flag or else `mpi4py`'s compiled object library will not be reinstalled. You can check that `mpi4py` is linked to the correct MPI library by doing:

  ```
  ldd {your/conda_env/directory}/{env_name}/lib/python2.7/site-packages/mpi4py/MPI.so
  ```

- Here are some commands that may help diagnose library and path problems:

```
module list
module show some/module/name
which python
which mpiexec
python -c "from mpi4py import MPI; print MPI"
ldd flash4
ldd flash_worker
```

4.2 FLASH setup/compile problems

- **make fails immediately with something like** `/usr/local/mpich2//bin/mpif90: Command not found.`

  Check that you are using the correct `Makefile.h`, with `MPI_PATH` and `HDF5_PATH` appropriate for your system.

- **make is slow.**

  Try `make -j` to use multiple processes.

- **make succeeds, but flash4 or flash_worker fails at runtime with error like** `error while loading shared libraries: libhdf5.so.8: cannot open shared object file.`

  Take a look at http://flash.uchicago.edu/pipermail/flash-users/2013-July/001322.html.

- **FLASH runs out of memory.**

  Reduce MAXBLOCKS or allocate more memory per core. The memory cost can be estimated as:

  ```
  (NUNK_VARS + NFACEVAR + NFLUXVAR) * (NXB + 2*NGUARD) * (NYB + 2*NGUARD) * (NZB + 2*NGUARD)
  * MAXBLOCKS * (8 bytes)
  ```

  where the variables are defined in `FLASH4.5/object/Flash.h`. For example, with `NUNK_VARS=46`, `NFACEVAR=2`, `NFLUXVAR=12`, `NXB=NYB=NZB=16`, `NGUARD=6`, and `MAXBLOCKS=100`, the memory usage is around 1 GB.

4.3 AMUSE configure/make problems

- **How do I troubleshoot X configure error?**

  A few places to look include: STDOUT from `configure`, `config.log`, and the source code of `configure` itself.

  Also, `./configure --help` may give an idea of tunable parameters.

- **FLASH alone compiles, but the AMUSE `flash_worker` build fails with library errors.**

  Try comparing, piece-by-piece, the command-line invocations of `mpif90` for FLASH alone versus `flash_worker` to make sure that `-L` and `-l` flags are sensible.

  You can manually edit some compiler flags in `config.mk`, located in the top-level AMUSE directory.

4.4 Runtime and MPI problems

- **AMUSE fails spawn processes over > 1 cluster node, but works if all processes are on the same node. MPI may hang, or return errors of form (for Intel MPI).**

  ```
  HYDT_dmx_register_fd ({...}demux.c:101): registering duplicate fd 0
  HYDT_bscd_slurm_launch_procs ({...}/slurm_launch.c:258): demux returned error registering fd
  ...
  main (../../ui/mpich/mpiexec.c:1118): process manager error waiting for completion
  ```

  One known solution: use OpenMPI, and do not overspecify Slurm sbatch or srun options, give only `-n`. A general solution is not known. This was seen with a relatively old AMUSE version, and may be resolved in newer versions.

For OpenMPI specifically:

- **How do I get information about the OpenMPI build on my cluster?**

  The command `ompi_info` reports how a given OpenMPI installation was built, e.g., version, compiler, configure flags, etc.

- **How do I set MCA parameters?**

  https://www.open-mpi.org/faq/?category=tuning#setting-mca-params

- **How do I fix runtime warnings about infiniband ports?**

  ```
  By default, for Open MPI 4.0 and later, infiniband ports on a device
  are not used by default.  The intent is to use UCX for these devices.
  You can override this policy by setting the btl_openib_allow_ib MCA parameter
  to true.
  ...
  WARNING: There was an error initializing an OpenFabrics device.
    Local host:   t035
    Local device: mlx5_0
  ```

  Follow the suggestion and set `btl_openib_allow_ib=1`. The OpenMPI FAQ explains how to set MCA parameters.

4.5 Navigating the source code

- **How do I decipher FLASH setup calls and runtime parameters?**

  Look at `bin/setup_shortcuts.txt` and `object/setup_params`.

- **A runtime parameter X has little or no documentation in `object/setup_params`. What do I do?**

  The file `object/setup_params` will tell you which unit declared the mysterious parameter. Once you figure out what it does, consider adding some documentation (in the FLASH unit's `Config` file) and submitting a pull request to the Torch repository!

- **How do I quickly track down a FLASH error message?**

  In the object directory,

  ```
  grep -I "my error message" *
  ```

  The flag `-I` instructs grep to ignore binary files, greatly speeding up the search.

A Torch component references

This list includes most major components, but is not comprehensive.

- FLASH 4.5: user's guide, Fryxell+ 2000
- PARAMESH (in vanilla FLASH): archived v4.1 manual
- AMUSE: documentation, book
- Fervent (radiative transfer): Baczynski+ 2015, Baczynski 2015, Ph.D thesis
- Multigrid (self-gravity): Ricker+ 2008
- BHTree (self-gravity): Wünsch+ 2018
- Sink particles: Federrath+ 2010
- Multiples (N-body): AMUSE book (see Sec. 4.5)
- Stratified box setup: Joung & Mac Low 2006, Ibáñez-Mejía+ 2016

B Notes on build process for specific clusters

B.1 Cartesius

Cartesius is the Dutch national supercomputer managed by the cooperative educational and scientific association SURFsara. Many projects have utilized Cartesius and its up-to-date hard/software are well suited for the many projects involving Torch.

To run Torch on Cartesius, you will need to first request an account on the computer. Logging on to Cartesius is straight forward: `ssh [username]@doornode.surfsara.nl` which will then allow you to select `cartesius` after asking for your password. At this stage, you will be able to use Cartesius in its full capacity but you will not be able to `scp` over your `FLASH` repository until your IP address is whitelisted. This can be accomplished by providing your IP to the Surfsara help center. You will be able to use `git` to bring over the `torch` repository to install everything.

Before we install anything however, you must make sure to use the appropriate software packages. Cartesius operates by use of user-chosen modules to be loaded prior to any process you wish to run. Either by editing your `.bashrc` or creating your own bash script to be run at login, you will need to load the updated module environment, as well as the individual modules required by torch.

```
module load 2019
module load foss/2018b
module load Python/2.7.15-foss-2018b
module load HDF5/1.10.2-foss-2018b
module load h5py/2.8.0-foss-2018b-Python-2.7.15
module load GSL/2.5-iccifort-2018.3.222-GCC-7.3.0-2.30

export MODLOC=/sw/arch/RedHatEnterpriseServer7/EB_production/2019/software
export MPIHOME=$MODLOC/OpenMPI/3.1.1-GCC-7.3.0-2.30

export OMPI_MCA_mpi_warn_on_fork=0
```

At this point, following the installation steps for `Torch` should result in a fully functioning software suite!

Preparing a run. Make sure to edit the `flash.par` file you are using such that the `output_directory` is pointed to your desired directory and make sure your module list script has been executed (an easy check is typing `which mpiexec` in the command line).

Starting a run. Cartesius uses the `slurm` job managing system and so is fairly straightforward to use. Here I have provided my run.sh script that I deliver to Cartesius with `sbatch run.sh`. The script contains parameters read by the slurm system as well as the command you want executed.

```
#!/bin/sh
#SBATCH --job-name=[informative-job-name]
#SBATCH -n [number of procs]
#SBATCH --time=[day]-[hour]:[min]:[sec]
#SBATCH --mail-user=[your-email@email.com]
#SBATCH --mail-type=[email condition]
mpiexec --mca orte_base_help_aggregate 0 -n 1 python bridge_multiples.py
```

It may take some time before your run makes it through the waitlist. Once it does, there will be some information output in your `simulation` directory where you ran the script. The most useful file is the one named `slurm-#####.out` which is where your screen output is redirected to. I check this periodically to see what simulation time my run has achieved, how many stars I've made, if it is hung up on a process, etc.

B.2 Cartesius - Refactor

The refactored version of Torch (on git branch `refactor`) uses the latest AMUSE commit on branch `master` which requires Python 3.X as well as FLASH version 4.6.2. The setup for the refactored version of Torch on Cartesius is largely the same as the `master` branch as described in this document but with a dew caveats.

Firstly, make sure you download and unzip FLASH4.6.2. This can be done from the same FLASH distribution webpage from which FLASH4.5 or earlier versions can be accessed.

Second, make sure to have the AMUSE `master` branch checked out and is at the most recent commit.

In order to satisfy AMUSE's Python 3.X requirements, we will need to establish a module environment that's a bit different from the one described in Appendix B.1:

```
module load 2019
module load foss/2018b

module load Python/3.6.6-fosscuda-2018b
module load HDF5/1.10.2-foss-2018b
module load h5py/2.10.0-fosscuda-2018b-Python-3.6.6
module load GSL/2.5-iccifort-2018.3.222-GCC-7.3.0-2.30
module load pytest/4.4.0-fosscuda-2018b-Python-3.6.6

export MODLOC=/sw/arch/RedHatEnterpriseServer7/EB_production/2019/software
export MPIHOME=$MODLOC/OpenMPI/3.1.1-GCC-7.3.0-2.30
export OMPI_MCA_mpi_warn_on_fork=0
```

In addition to this slightly altered environment, the Cartesius Python 3.6.6 module does not currently have the `matplotlib` and `docutils` libraries installed. This is okay, we can just install them ourselves in our user environment using `pip install --user matplotlib docutils` after loading the Python/3.6.6-fosscuda-2018b module.

Finally, if running this module environment setup reports any error (such as "XYZ module cannot load due to conflict") try running the same environment setup file again and there should be no issue.