E-mails to a Young Computationalist

A guide to `make`ing and running executables on HPC machines at the University of Rochester by example of *FLASH*

M. B. Adams[1]
University of Rochester
Last Updated: October 12, 2018

[1]madams@pas.rochester.edu

Forward

As an undergraduate studying physics, I distinctly remember my peers and authority figures enforcing, or posturing, the philosophy and cultural expectation that true genius self-manifests, and that some are truly gifted for some pursuits – whether they realized it or not. Now that I am a bit older, I see how flawed and untrue this is. My hope is to squash this cultural attitude, and make the work I care about seem accessible.

An example, or misattribution of this philosophy can be made by misapplying a quote in Rainer Maria Rilke's *Letters to a Young Poet* (as I am about to do). Rilke said to the young cadet, Kappus, who had asked him for advice on his poetry, said "Nobody can advise you and help you, nobody. There is only one way. Go into yourself." This outlook, despite it's clear usefulness in the conveyance of emotion through art and poetry, is pervasive, and has infiltrated attitudes and philosophies in many academic disciplines (not completely blaming Rilke for my perceived "bad attitudes" and "mentoring methods" (or lack there of)) – especially physics and physics-adjacent fields. The reality is that folks are busy, and it is difficult to think back to where one starts to learn something. For instance, it's hard to think of how to teach someone who has never seen a derivative before, to take a derivative, when it is like breathing to someone who has been taking derivatives for forty years. I extend this analogy now to working with high production magneto/hydrodynamics (MHD) codes in my *E-mails to a Young Computationalist*.

I am writing this as a graduate student who works with the parallelized, modular MHD code, *FLASH*[2], for undergraduate students who have fallen under my limited-mentorship. The first set of e-mails are with a student named Shira who is working on eventually developing a simulation utilizing laser energy deposition to simulate jets. I suppose as time goes on, and more students may come my way, I'll append more helpful e-mails as time goes on. If you're reading this, hopefully you'll find it useful!

Marissa B. Adams (October 2018)

[2]See: http://flash.uchicago.edu/site/

Gmail Marissa Adams <adams.marbea@gmail.com>

Running FLASH on BlueHive + tips of the trade

7 messages

Marissa Adams <madams@pas.rochester.edu> Wed, Oct 3, 2018 at 9:08 PM
To: Shira

Hi Shira,

I could probably detail all of this in a PDF document now that I think about it. For now, this e-mail will do. This is how one can run FLASH on BlueHive 2.5 (BH2.5 or BH25). Honestly, the general procedure will be the same for any sort of code or supercomputer you bump into along the way in your career.

Note: commands I type into the terminal/stdout/err/in will be `fixed width and have green font with black background`. Directories, paths, and file names will be *italicized*. Text embedded commands or references to flags will be in plainly `fixed width`. I will try to highlight all concepts that are important or steps by numbering and large letters, such as...

1. Log into BH25

I prefer to use `ssh`. I think you may have your own thing to log in on, but if you use a UNIX system, you open up the terminal and type:

```
ssh [netid]@[machine address]
```

hit enter, and then it may ask you for your password. When you type your password, you won't see anything, including the dots that typically are used to represent passwords. For instance

```
madams@styx:~$ ssh madams15@bh25fen.circ.rochester.edu
You are attempting to connect to a computer system that is private
property.  Any and all login attempts are recorded.  Unauthorized
or improper use of this system may result in civil and criminal
penalties.
Password:
```

You can also add the flags `-X` (which enables X11 forwarding) or `-Y` (which is preferred, trusted X11 forwarding -- it is more secure), e.g.

```
ssh -Y madams15@bh25fen.circ.rochester.edu
```

Also, note that you can `ssh` into virtually any machine if you have an account or access. For instance, I could `ssh` into my work machine from my laptop using:

```
ssh -Y madams@styx.pas.rochester.edu
```

This is just to show you that each computer has an address, and effectively you can log into any computer via another one this way.

So CIRC had an attack on their computers last year where someone was trying to mine for slavcoin or something. They've made their system more secure, annoyingly so, but including an extra layer of authentification using VPN. You need to use the VPN whether you're working on or off campus.

Here is more on acquiring the VPN Duo Mobile: https://tech.rochester.edu/services/two-factor-authentication/

You download the app to your phone, and when it knows you're trying to log in to BH25, it'll send a push notification on your phone that you'll approve; then you can log into the CIRC machine.

So after accepting the VPN on my phone and entering the password, I get

```
Autopushing login request to phone...
Success. Logging you in...
Last login: Mon Sep 24 16:46:11 2018 from [an IP address I've deleted here]
This machine is private property of the University of Rochester.
Unauthorized access is strictly prohibited.
All activity may be monitored.
Filesystem    GB used  GB soft limit  GB hard limit  status
     /home       9.95          20.00          25.00  under soft limit
  /scratch      39.15         200.00        1000.00  under soft limit
```

So you see how above it tells you the file system directories and their quotas? If you ever want to see that again without logging in again, type `quota` anytime while logged in and it'll tell you how much space you've used. So your *home/* otherwise known as *~/* directory is where you'll want to store your code, and debugging, analysis tools you develop/scripts, things that are special effectively that you want to keep for a long time. The *scratch/* space is actually where you'll run your executable from your code and generate simulation output.

Alright, so you've successfully logged into BH25.

2. Add the *Makefile.h*

Attached you will find my Makefile.h for BH25.
Move your version of FLASH4.5 into your home directory, i.e. when you `ls` in your *home/*, make sure you see *FLASH4.5/.*
You will want to move the Makefile.h from your local machine to BH25. The way to do that, I prefer, is to use `rsync`.

2.1 Moving files between machines

If you want to pull from a machine,

```
madams@styx:~/Desktop$ rsync -Pav madams15@bh25fen.circ.rochester.edu:/home/madams15/
FLASH4.5/sites/bh25fen.circ.rochester.edu/Makefile.h .
```

If you want to push from a machine:

```
madams@styx:~/Desktop$ rsync -Pav Makefile.h madams15@bh25fen.circ.
rochester.edu:/home/madams15/FLASH4.5/sites/bh25fen.circ.rochester.edu/
```

When you get the *Makefile.h* onto BH2.5, make a directory in the following path:

```
cd FLASH4.5/sites/
mkdir bh25fen.circ.rochester.edu
```

Move the Makefile.h into that new directory

```
mv Makefile.h FLASH4.5/sites/bh25fen.circ.rochester.edu/
```

We don't really need to worry about the contents of the Makefile.h and what it does just yet.

3. Loading modules

So FLASH needs some assistance from other dependencies to run, such as a Fortran compiler, MPI for parallel processing (allows the supercomputer to be useful! spreads the work across different cores and then brings it back all together), a package that helps make the output from FLASH (called HDF5), and some other fancy packages. Play around with the following commands:

```
module list
module avail
```

When I do `module list`, I get:

```
[madams15@bluehive FLASH4.5]$ module list
Currently Loaded Modulefiles:
  1) slurm/16.05.9     3) intel/2017        5) hdf5/1.8.17/b4
  2) circ              4) impi/2017         6) hypre/2.11.1/b2
```

You may not have the same ones loaded. You'll probably only have the first two. I have mine loaded right when I log in via my *.bashrc* file. I would suggest editing your *.bashrc* file so you don't need to load the modules everytime you log into BH25.

3.1 Editing your *.bashrc*

Remember when I told you about *ls -ltr*? Get ready for this! Try *ls -A*. Oh snap! There are a bunch of files and directories with a period in front of their name? Weird right? I don't understand it either. But generally in every *home/* directory ever on a computer, there is some sort of *.bashrc* (LINUX) or *.bash_profile* (MAC). Open this file up in your *home/*:

```
emacs .bashrc -nw
```

Again you can use any other editor you like in the command line, but bear with me, I like `emacs`. Attached you will find my version of my *bashrc* on bh25 without the dot in the front (if you want to use it directly, just change the name via `mv bashrc .bashrc`). You should be able to open it the same way as above though. Edit it to your purposes, but I'll detail some things about it here:

```
# .bashrc
# Source global definitions
if [ -f /etc/bashrc ]; then
        . /etc/bashrc
fi
# User specific aliases and functions
alias scratch='cd /scratch/madams15/'
alias bh25='ssh -X madams15@bh25fen.circ.rochester.edu'
alias bhhome='cd /home/madams15/'
alias bgqhome='cd /gpfs/fs2/bgqscratch/madams15/'
alias modvisit='module load visit'
alias int='interactive -p debug -t 60'
```

```
module load impi/2017
module load hdf5/1.8.17/b4
module load hypre/2.11.1/b2
module load intel/2017
```

So all of this is necessary for you except for the `aliases`. You can create `aliases` for yourself as you see fit! It's a way of making your own command!
Once you finish editing the *.bashrc*, make sure once saved you type into the command line, `source .bashrc`. This does the same thing as logging in and out again.
Once you've finished doing that, type `module list` again and see if you have those modules in your *.bashrc* loaded! If not you can do `module load [name and version of module]` in the command line and that'll do it too.

IMPORTANT
You want to make sure that the same modules loaded are the same modules referred to in the *Makefile.h*, and in your job submission script, which I'll tell you about now.

4. Running your executable, `flash4`

Okay so you followed through on the steps in the user manual where you've set up your code with an *object/* directory, typed `make` in the *object/* directory, watched the code use the modules we loaded link all of the files, optimize it all, and dump all the information into an executable file called **`flash4`**. I will detail how to do this in another e-mail that I will perhaps send tomorrow, but we'll probably hit on this soon. Here is what you're going to want to do:

1. `mv` or `cp` **`flash4`** to your *scratch/* space
2. To be honest, before you even do this, make a workspace in your *scratch/* space. I like to designate things by date and time period for good workflow, but you could make a directory called: *FLASHProject/*, then in there make a new directory called *Sedov/*, i.e. */scratch/[netid]/FLASHProject/Sedov*
3. Okay once you `mkdir` that directory move your **`flash4`** into there!
4. In your *object/* directory search for a *flash.par* file, move that into your new *scratch/* directory too!
5. Attached you'll find a job submission script (*job.sh*). This makes the supercomputer run your job.

IMPORTANT

> Also if you haven't already, check out this link here: https://info.circ.rochester.edu/
> This website can be your BEST FRIEND when it comes to running on the CIRC machines. If you can't manage to get VisIt working

Okay, take a look at the link above before doing any of this honestly ^^^^^
When you want to run an executable to receive output, we refer to that as a job. To manage all the jobs for all of the users using BH25, CIRC has implemented this service called SLURM. Here are some basic SLURM commands (and you can read more about SLURM above)
`sbatch job.sh` : submits the job
`squeue -u [netid]`, e.g. `squeue -u madams15` : allows you to see what jobs of yours are running, still pending in the queue, or have failed.
you can also do `watch -n 1 squeue -u madams15` to see it update every second
`scancel [job ID]` : cancels the job, and you can find the job ID when you use `squeue` above
Here is more on SLURM commands: https://www.rc.fas.harvard.edu/resources/documentation/convenient-slurm-commands/

If your job is accepted and running you'll likely see output!! Which is super exciting. You can watch your log files using `tail -f flash4.log` (for instance). You can either use the **fastX** thing they mention in **info.CIRC**, or `rsync` your output to your local machine to use your own version of **VisIt**. We'll figure out what works for you to visualize.

Nifty commands/tips:

- https://www.tutorialspoint.com/unix_commands/grep.htm
- https://www.tutorialspoint.com/unix_commands/find.htm
- https://www.tutorialspoint.com/unix_commands/sed.htm
- This one person made a whole list of the UNIX commands they find most important: http://oliverelliott.org/article/computing/ref_unix/
- A beginners guide to `emacs` (sorry): http://www.jesshamrick.com/2012/09/10/absolute-beginners-guide-to-emacs/
- If you've heard of a command and want to learn more you can always do man [name of the command] to learn more, then press q to exit
- Google/stackexchange is your friend

On debugging/questions to ask yourself when things go wrong:

- Commenting out sections and print statements are your best friend
- Always check the boundary conditions
- Do you have all of the files needed to run?
- Are your modules loaded?
- Don't glaze over the error output, look at it closely and it'll tell you where you want to look next.
- DON'T GET DISCOURAGED
- ASK FOR HELP ONCE YOU START TO SPIN YOUR MENTAL WHEELS

Okay, so this was probably a lot of information. Don't get overwhelmed. I hope you can use this e-mail as a resource in the future. That is its main purpose! If I send you anymore like this, I'll do as a response to this e-mail as an ongoing series.

Good night!
Marissa

Marissa Adams
PhD Student
Website: http://www.pas.rochester.edu/~madams
University of Rochester
Department of Physics & Astronomy
371 Bausch & Lomb Hall

3 attachments

Makefile.h
5K

bashrc
1K

job.sh
1K

Marissa Adams <madams@pas.rochester.edu> Wed, Oct 10, 2018 at 12:11 PM
To: Shira

Hey Shira,

Hope you don't mind but on the following addition of e-mails; I am going to CC some folks who may also be interested in these tips (Hannah Hasson is a first-year graduate student in my group who also does some computational work, if you want, I am happy to introduce you two!; Pierre is my adviser, same offer goes to him; of course Hussein). Also with your permission, I'd like to eventually export these e-mails as HTML/PDF but covert them to PDF format and make them available to future students between groups and stuff. Let me know if that is okay with you.

I know you asked me this a few days ago, but in case this e-mail gets forwarded to anyone else, this is the procedure to get into the supercomputers off campus, or on the University network in general (say you wanna use the library resources and access journals online but then that paywall comes up when you're off campus -- using (1) below can alleviate that too!):

More on VPN-ing

1. Look into downloading the **VPN Cisco AnyConnect client**. (see this webpage for more details: http://tech.rochester.edu/services/remote-access-vpn/). You want to first connect on your laptop/computer using this application/software. It'll effectively treat your network as if you're on U of R's once you sign in. You need to do this before you SSH anywhere. You can probably do a push notification to your phone, input a series of numbers for a second authentification/enter your password. (**Note:** if you have to reboot your phone or get a new one, you need to set up the duo client I mentioned in the previous e-mail all over again *sigh*)
2. **You can now SSH once you're logged in via VPN.** See last e-mail.
3. **You can then VPN via push notification again using DUO.** Effectively once you get into the University network, you can go through the process I detailed in the last e-mail.

In this e-mail want to address some of the structure/systems of supercomputers and how that can be useful for submitting jobs, and then also some helpful workflow/data management for when you start accumulating more simulation output.

Obviously, supercomputers are super useful due to their processing power. Running a code on your own laptop may take many days when on a supercomputer it may take a half an hour or less. Why?

On supercomputers

Generally, supercomputers have a comparable hardware set up to your own computer. Computers generally have the same sort of structure, so many of these concepts can also apply to your own machine.

The machines at CIRC BlueHive 2.5 ([netID]@bh25fen.circ.rochester.edu - https://info.circ.rochester.edu/BlueHive/System_Overview.html) and Blue Gene Q or BlueStreak ([netID]@bluestreak.circ.rochester.edu - https://info.circ.rochester.edu/Blue_Gene_Q/System_Overview.html) are the two Rochester supercomputers managed by CIRC. Blue Gene is a standard project/SC set up by IBM that has high operating speeds at low power consumption, so you'll come across many Blue Gene machines if you use different supercomputers in your career. I am not sure if Blue Hive is also a Blue Gene machine, but I know that it has a bunch of Intel processors. What is a processor?

A **processor** or central processing unit (**CPU**) is a bunch of circuitry that somehow, through electrical computer engineering (i.e. magic), processes all of the basic commands and instructions that drive the computer (fetch, decode, execute, and write). There are a bunch of elements to a processor, or different types of processors, but I can't speak to them all. The two you'll come across are CPUs and **GPU**s. GPUs are graphics processing units. Back in the day when computers first became a thing a processor would take up a whole room, but now you can make a processor so small you can barely see it. Not all processors are the same. Cores/processors will share memory and bandwidth.

But... what is a **core**? A core is effectively a microprocessor. It is the smallest unit for execution on a computer. Cores are processors called functional processing units (**FPU**s) which do most of the number crunching. These things (cores versus processors) are all the same concept. It is really confusing, primarily because of branding and weird politics/competition between different companies. The word "core" was introduced by Intel and is sort of their brand word. Turns out that actually processors can be comprised of cores.

*A **chip** is comprised of typically four processors.*
*A single chip is typically thrown onto a **compute card**, which has a bunch of necessary hardware to make it apart of...*
*A **node**. A node is comprised of a bunch of these chip cards; a server. A node will have several processors sharing memory but may effectively be treated as a "computer".* Effectively your laptop is a node. If you connect your computer to a printer, then you have two nodes! If you connect your computer to some backup drive or another computer, then you have another node! If you think of those cool graphs you see CS people make that describe networks, the nodes are where the lines meet.
*Once you have a bunch of nodes, you can throw them onto a **node rack**.* You can then put a bunch of cabling to connect racks together, and then you get a **system** or **cluster**.

You're probably heard of (multi-)threading, or parallelization, MPI, openMP/I. All these weird terms. Effectively these things are types of software that allow the computer to split up the processing of the code in an efficient/optimized way so that you can crunch the numbers more quickly. It'll take the job you want the computer to process and break it up appropriately, and then stitch it back together when it is done. That's why we need to load these things before we `make` stuff and run the exe we've made.

> **FYI:** Found these slides from CIRC on some stuff you may want to check out (https://info.circ.rochester.edu/Training/Workshops.html). CIRC will have a Winter Bootcamp that I highly suggest you go to if you can manage. I've participated in them before and they are fun, helpful, and a good way to meet other computationalists/users of CIRC. I'd suggest signing up for the Linux, and Visualization classes they'll have when the time comes. Also if you ever have a problem with your Makefile, the modules, or anything that pertains directly to CIRC, you can e-mail them (they are very quick to respond: circ@rochester.edu)

How does this pertain to my job.sh?

So everything I've told you above is helpful to know something about for when you're running jobs, so you can tell the supercomputer how many resources you need to run your code. I will be referring to this page: https://info.circ.rochester.edu/BlueHive/Running_Jobs.html and the script I've given you in my pervious e-mail.

Folks have different reasons for running jobs. Those reasons need to be categorized so resources can be used appropriately among the community of users. These reasons are typically described by a **partition**. People also call partitions, queues. I typically use the *debug* queue, and *standard* queue. If I am going to visualize stuff ON BH2.5, I'll use an *interactive* job (this is effectively what FastX does, i.e. what we did when we signed in on the internet to BH2.5 w. the GUI interface to run *Visit*.)

The **wall time** is how long you want to submit your job for once it is accepted and running. Sometimes it'll fail, or finish before the wall time is over. So it is important to babysit your jobs sometimes, especially while debugging.

You can specify how many nodes you want to use, and how many cores/node.

An example job.sh

This should effectively be the same file I've sent you in the previous e-mail.

```
#!/bin/bash

#SBATCH -J flash4-test                  Job name
#SBATCH -p debug                        What partition you're using
#SBATCH --mem-per-cpu=50MB              The amount of memory you want per core
#SBATCH -N 1                            Number of nodes you want
#SBATCH --mem=24gb                      Memory per node (note MB < GB)
#SBATCH --ntasks-per-node=1             Using bash
#SBATCH -c 16                           Number of cores/task for a multithreaded job
#SBATCH -t 0:30:00                      Wall time
#SBATCH --no-requeue

##module load openmpi/2.1.1/b1          The modules we're using
module load impi/2017
module load hdf5/1.8.17/b4
module load hypre/2.11.1/b2
module load intel/2017

mpirun -np 16 ./flash4 > flash4.log     Command to run the code!
```

If you want to know more about the options to add to your bash scripts and how to run jobs in general, see this webpage: https://info.circ.rochester.edu/BlueHive/Running_Jobs.html

Job management

Now say you're running a bunch of jobs and you've got the hang of this. Based on our last meeting I'd say you're definitely ready to do that! I gave you a homework of running this executable you've made for a longer amount of time so we can understand the full grasp of the physics other than just making sure the code runs. You want to run this executable with different boundary conditions (reflective, outflow, and periodic), and for different explosion energies (0.1, 1, and 10 ergs) -- it is 1 in the flash.par. Now how do you organize all of this?

Well here is an old example of how I organized things and tracked my jobs with a different MHD code called AstroBEAR (https://astrobear.pas.rochester.edu/trac/blog/madams07282014). This blog post I wrote a super long time ago will give you an idea of how you may wanna structure your directories, and what things to keep track of as you're running stuff. I'd suggest writing all this stuff down in a nice organized page, or detailing it in some markup webpage or something.

Best,
Marissa

Marissa Adams
PhD Student
Website: http://www.pas.rochester.edu/~madams
University of Rochester
Department of Physics & Astronomy
371 Bausch & Lomb Hall

[Quoted text hidden]

Marissa Adams <madams@pas.rochester.edu> Thu, Oct 11, 2018 at 10:14 PM
To: Shira

Hi Shira,

While you're working on that suite of runs, you'll probably want to start visualizing the output. So in today's e-mail I want to address...

Visualization

The majority of the time, the output you will get from running a code like FLASH, will be in HDF5 format. HDF stands for Hierarchical Data Format, and I guess the 5 is for the version of them working on this type of package. You need to use a software that is capable of reading simulation in this output format. I know of a few:

1. **Visit:** (https://wci.llnl.gov/simulation/computer-codes/visit) a Lawrence Livermore National Laboratories (LLNL) software, which has python underneath. When I am searching for Visit related things, often I google for "Visit LLNL" since visit is a terrible name to name software.
2. **Paraview** (https://www.paraview.org/)
3. **h5py:** (https://www.h5py.org/) is a python package that allows you to access the output and do analysis via python. This requires some coding since it isn't really a software w. a GUI unlike Visit and Paraview. I would use this in tandem with `h5dump`, a command you can use in terminal, to see the structure of the output.
4. **yt:** (https://yt-project.org/) another python package that a lot of astrophysics people seem to be using nowadays. This uses h5py underneath, I think. It is a bit more user-friendly but I don't find their documentation that friendly.

There are probably more out there but this is what I am most familiar with or have used. I would also like to emphasize that visualization is **not** analysis. (1) and (2) are very useful for visualization, (3) is super useful to do analysis with, and (4) is a combination of the two, I'd say. Both analysis and visualization are very important. Typically you do visualization first to make sure things look okay. Visualization is very useful to get an idea of the topology and structure of flows, as that is primarily what we are dealing with: fluids, and fluid approaches to plasmas.

- Here is a link to the Visit User Manual: https://wci.llnl.gov/codes/visit/1.5/VisItUsersManual1.5.pdf
- Adam Frank's group implements really nice documentation on their group activities using wikipages. They have a section on Visit that may be useful but keep in mind for the expressions, and many of their tutorials are based on their code: https://astrobear.pas.rochester.edu/trac/wiki/VisIt
- There are also some FLASH specific plug-ins for Visit: http://flash.uchicago.edu/site/flashcode/user_support/visit/

So there are multiple Visit(s) (?) that you could use, or rather, different places you can load up visit:

1. Your laptop: you've already downloaded Visit. Do you need help installing it?
    - If you're going to be running Visit on your laptop, make sure that you also `rsync` over the files you want to visualize (see my first e-mail). I would suggest ``pulling'' the files from bluehive, i.e.
        - `rsync -Pav [netID]@bh25fen.circ.rochester.edu:/scratch/[netID]/FLASHProject/Sedov/* .`
        - Note the "`.`" above at the end, that means, "take all, i.e. "`*`", those files in that directory, */scratch/[netID]/FLASHProject/Sedov/*, recursively (in the `-Pav` flags), and put them HERE." The period means, "here" effectively.
    - I say ``pulling'' because, from your laptop, you're trying to pull those BH2.5 files to your laptop. If you're to ``push'' it'd be moving files from your laptop to BH2.5. This is easiest to do because it's difficult to know the address of your own personal machine, and honestly, I've never learned mine and I am not sure if it is the IP or something... just pull to your laptop lol
2. It may be easier to visualize on BH2.5, if you so choose, using FastX (https://info.circ.rochester.edu/BlueHive/FastX.html).
    - Follow the instructions on the above webpage to use the supercomputer with a GUI.
    - Open the terminal and type, `module load visit`
        - you may want to tab just in case there are different builds of visit; choose the latest build
    - Then after you've loaded visit, you can then type the command, `visit`, in the next line, and it should load up and be ready for use.
    - Note that you can do the same thing on your computer, just type `visit` in the terminal for it to start running (you won't need to module load it, however). It may be `./visit`, where ever that exe is stored.

Work Flow

Okay, so given that we've talked about making an executable (exe), running the exe, and then visualizing the output from the exe, we can effectively diagnose a standard workflow to do computation:

1. Coding your problem (making your own problem module, adding a unit, adding a new module, doing whatever type of code development, etc).
2. Making the executable to run your problem module
3. Running your executable
4. Visualizing the output from your executable

5. Doing analysis on your output from your executable

I would say that between all of these steps are layers of debugging. Learning these steps can take a long time. Typically the way to start learning is by beginning with (2), (3), and (4) with one of the provided problem modules (as we are doing now with Sedov). A sign that you're ready to move on is that you're capable of doing (2) --> (3) --> (4) within one day, or maybe the span of a few hours, or one hour. Once you get quick at the (2) --> (3) --> (4) process, then you can start doing (1). In order to do the debugging required in (1), you need to be able to do the (234) process very quickly to find your errors and resolve them. Then typically (5) comes around when you're preparing for publication. Doing analysis can also aid you in visualizing because 1D line-outs can be useful in making sure your result makes sense as some problems have very well known 1D profiles (such as Sedov). VisIt can do some of those things, but when you start trying to do more complicated stuff it gets icky.

Running FLASH on your own laptop

This can be super useful when you're doing (1) above because you can start to work very quickly to get your problem module up to scrub to then run on BH2.5 later for production runs (very high resolution, meant to have analysis done on them -- needs more computing power).

If you want to do this I highly recommend reading section 2.3 in the FLASH user manual. You'll want to download docker to set up a container so that you can run the code in there (has all the modules set up etc). If you want to do this, let me know and I can help you.

If you don't want to use docker, then you have to install comparable libraries/modules to the ones you load on BH2.5 which can be a pain, and take some time.

If you are eventually able to get FLASH set up on your local machine then you can run the executable right in your object directory using `./flash4` . Then the output accumulates in the directory, and you can have visit access it right there and quickly check the output as you debug. This is a good debugging strategy for when you're starting to test out your problem module.

Running FLASH on BH2.5

Okay, there is a general workflow here:

1. Put the FLASH4.5 directory in your home directory.
2. In FLASH4.5/ you can make some set up scripts that you can run to set up your new object directory.
    - To get some practice doing this, do the following:
    - `emacs Sedov.sh -nw`
    - In the file you're writing, now type,
    - `./setup -auto Sedov -objdir=Sedov`
    - Then do control-x s (to save) control-x c (to close) -- these are emacs commands, learn more here: https://www.cs.colostate.edu/helpdocs/emacs.html, note that instead of saying "control" they say C.
    - That is literally the same as what you do to in the command line! But now we have it in a script file that we can run. We do this because sometimes the commands we have to "set up" our object directory can get very long and tedious and we get sick of writing them all the time.
    - Once you've made that file, you can check to see if it is there by doing `ls`.
    - Make it executable by doing `chmod a+x Sedov.sh`
    - Then execute it, `./Sedov.sh`
    - Once you do that, it should successfully set up, and then once you ls you'll see a new directory, *Sedov/*.
        - You know the *object/* directory from before? You made a new one with a new name. This is useful because sometimes you're going to be building different problems at the same time.
    - Note that everything I am saying about the set up here also applies to set up on your own machine, not just BH2.5
3. Go into *Sedov/* and type `make`
    - If you're running on your own personal computer you can do `make -j` (it'll make more quickly) but don't do this in your home directory on BH2.5 because that is hosted on the "head node" of the machine, which is shared, and people will be salty that you're taking up more processors. However you could probably get away with it once or twice, but the CIRC folks will notice and probably e-mail you.
4. Once it is done making, remember to move/copy your new executable (*flash4*), and *flash.par* to your run directory in your scratch space and that there is a job submission script in there. As you start making your own problem module, you may have other files in your object directory that you need to move too (like EOS tables for instance)

Other notes:

- I mentioned symbolic links last time, you'll see these if you look in your object directory and see files pointing back to files of the same name in the problem module directory. These are symbolic links. You can create and make your own.

See the following website: https://www.howtogeek.com/297721/how-to-create-and-use-symbolic-links-aka-symlinks-on-a-mac/

I hope this is helpful!
Marissa

Marissa Adams
PhD Student
Website: http://www.pas.rochester.edu/~madams
University of Rochester
Department of Physics & Astronomy
371 Bausch & Lomb Hall

[Quoted text hidden]