FAQ & Scientific Software Use Cases

0.0 A Very Short Course to A Batch Scheduling

The first thing you need is user permission to the clusters. These are handled with IDM group management.

Welcome, and nice to see you got here. I take it you have seen what we provide? If not, please do have a look. Now, let me introduce you to a novel little utility called batch scheduling. Very old invention but batch schedulers are in use in most, if not all supercomputers over the world and you're bound to meet one eventually if you are planning to work with computational challenges. They come in many flavors, but fundamentals are the same. So, let's get started by logging into Kale:

```
ssh ukko2.cs.helsinki.fi
```

We could just start our little process in the host we just logged in, but should everyone do that, nobody would get very far. In fact, we'd struggle to get anywhere really. Let's try something else instead...

```
/usr/bin/srun hostname
```

What just happened? Command hostname was run in some host called ukko2-05. Yes, that was the compute node of a cluster.

Translation: launch (srun) and request (default resources) and execute the 'hostname' command in the cluster compute node.

Actually, this equal to doing this with ssh:

```
ssh username@yourhost.name.here hostname
```

Why would you then need a batch thing? Well, there is more to srun than meets the eye. Let's try this:

```
/usr/bin/srun -c8 --mem=2M -t1-0 hostname
```

What is the significance of this awkward looking spell? You can allocate resources to the task at hand (*In short, you can ask for specified amounts of memory, CPU's, and time*). Srun handles the process placement, so you actually do run the command with 8 cores and not with just one while seven others idle out. The hostname is just not the most scalable of processes so you do not get a significant performance gain. Try to do that with ssh.

Translation: launch (*srun*) and request (*8 cores (-c)*), 2MB of memory (-*mem*) for a 1 day (-*t*)) to execute the 'hostname' command in the cluster compute node, or finish when the command has been executed, or become terminated if you exceed what you asked for. 8 cores and a day to run hostname is a bit of overkill but let's not think about that now.

You could use pseudo-terminal instead of hostname and have shell opened on the login node:

```
/usr/bin/srun -c1 --mem=2M -t1-0 --pty bash
```

We're not done yet - and my program takes a day, not just a second or two, so what now? *I could start a screen and then execute that there, and log out...* Wait! Now I am drifting a bit. Hold on. Let's, make this a bit more convenient and see what we come up with (*pick an editor you like, I prefer vi, others do prefer other editors*):

```
vi job.script.sh
```

Then type the following:

```
#!/bin/bash
#SBATCH -o /wrk/users/myusernamehere/output.txt
#SBATCH -c8
#SBATCH --mem=2M
#SBATCH -t1-0

srun hostname
srun sleep 600
```

Well, that wasn't really more compact, wasn't it? Now, let's see if we do the following:

```
sbatch job.script.sh
```

Here's an interesting thing. You could now end your session and log out. Your job would go about doing its business without your further input. This allows you to send jobs into the system in batches of one or more. Hence we called it a batch job. You could ask it to send you an e-mail once it's done if you'd like.

Output is written to the output.txt file.

Translation: sbatch command sends the script into a queue waiting for execution, and then srun executes the command in the resources you have requested. This is called submitting a batch job (*every line with #SBATCH is a resource reservation, lines after that, well that is good old Unix*). -o option defines the output file, in this case, output.txt.

Finally, for the bigger picture...

Slurm does not replace Linux functionalities. In fact, Slurm has two parts. One part (sbatch script) creates the boundaries of the playground, second (srun) makes sure your toys are placed in the right places in the playground.

What one does inside the playground is entirely up to the boundaries set by Linux and creativity. All functions, environment variables, etc. are working as they would in any ordinary shell. The difference is that if you exceed the playground boundaries in any way, your right to play will be terminated.

The only limitation is, that Slurm allows ordinarily a process to run once, after which it considers the process successfully completed and then terminates the surrounding playground. This makes it non-trivial to play with daemons, or processes set to operate in a similar fashion.

Congratulations! You have just finished the short course for batch scheduling! Was that hard, was it? There is plenty more you can do...

0.1 How to install software

Well, let's make one thing straight. You may have already thought about it, or asked about it but you cannot gain root privileges to install readily packaged software to a cluster as you could on your laptop. However, you can install a lot of software under $WRKDIR or $PROJ directory. You can of course use your own codes, and vast majority of other software you do come up with does not require privileges to install. Also remember that in a tricky looking cases Conda, or Python virtual environments are your dearest friends.

To make it easier, and to maintain different versions, you can and should use Environment Modules. That way you can run your work independently from system libraries. Here is our simplified guide to get you going. Building modules manually is really easy.

Administrators install new modules system-wide when the software is needed by multiple users, and when they are related to development, and when they have time to do that. They cannot provide marginally used, or eccentric software packages and modules for every need.

1.0 PYTHON VIRTUALENV

Python virtualenv is the most common software package that is requested and extremely easy to create and customize to your specific needs. Here's the example to get you around the creation of Python virtualenv. Simple, efficient, and customizable. Remember that if you wish make sure that the pip caches end up in the right places, set --cache-dir -flag to the right location.

Note: We use the default module versions in the examples. You may wish to use some other than the default, or define the version you wish to use. Also, note that default module versions are bound to change in time.

> *1. module purge*
>
> *2. module load Python*
>
> *3. cd $PROJ*
>
> *4. python -m venv myVirtualEnv*
>
> *5. source myVirtualEnv/bin/activate*
>
> *6. pip install --cache-dir <cache directory> <package you need>*
>
> *7. pip install* <any additional packages you wish to install...>

When you are done for the day

> *deactivate*

... and when ready to start again

> *source myVirtualEnv/bin/activate*

Oh, yes. Last but not least. Your environment is indeed inherited when you start a batch job, or interactive session on the compute nodes. This includes Python virtualenv.

2.0 Tensorflow

Since TensorFlow is very common, here are instructions how you can create your own Tensorflow installation by using Python virtualenv above. Additionally, we'll go through how to use it in interactive mode and through the batch system, and also a sample program to get you off the ground. If you do not specify the versions of modules, you will be loading the defaults, which are the newest available. If in doubt, or not sure about dependencies, you can always have a clean slate by purging modules and then load a correct set. Note that if you need to relocate pip cache from the default directory of ~/ to someplace else, you can use the --cache-dir option.

You cannot leave out the cache -option because the default location at $HOME is restricted by quota. Leaving the cache -option out will cause error messages about missing files, etc. Your cache directory can be in $WRKDIR, or $PROJ as per your discretion, albeit we do recommend $WRKDIR.

Note: Use Python 3.6. It seems that Tensorflow does not work well with 3.7.

> *1. module purge*
>
> *2. module load Python cuDNN*
>
> *3. cd $PROJ*
>
> *4. python -m venv myTensorflow*
>
> *5. source myTensorflow/bin/activate*
>
> *6. pip --cache-dir </wrk/users/<username>> install tensorflow*
>
> *7. pip --cache-dir </wrk/users/<username>> install tensorflow-gpu*
>
> *You can then add additional libraries etc..*
>
> *8. pip --cache-dir </wrk/users/<username>> install keras*
>
> *9. pip --cache-dir </wrk/users/<username>> install* <any additional packages you wish to install...>

2.1 Running Example

Here's an example batch script for you to use with TensorFlow, assuming that you have followed the directions above to create the virtual environment and wish to evoke it in the batch script. Note that because the module environment variables are inherited when the batch job is submitted, and hence module purge to make sure that you load only the modules required.

```
#!/bin/bash
############## This section states the requirements the job requires:
#SBATCH --job-name=test
#SBATCH --workdir=$WRKDIR
#SBATCH -o result.txt
#SBATCH  -p gpu
#SBATCH -c 2
#SBATCH --gres=gpu:1
#SBATCH --mem-per-cpu=100
############## Here starts the actual UNIX commands and payload:
module purge
module load Python cuDNN
source myTensorflow/bin/activate
srun tensorflow-program
```

Interactive use example (*gives a default wall time. This can be altered by -t<time in format [DD]-[hh][mm][ss]>*):

```
srun -c 1 --ntasks-per-core=1 --gres=gpu:1 -pgpu --pty bash
```

And once your session on the node starts, you can do the regular unix -things:

```
module purge
module load Python cuDNN
source myTensorflow/bin/activate
srun tensorflow-program
```

2.2 TensorFlow Example

Here's a simple program script.py example you can use to determine that your environment works as expected.

```
Import `tensorflow`
import tensorflow as tf
# Initialize two constants
x1 = tf.constant([1,2,3,4])
x2 = tf.constant([5,6,7,8])
# Multiply
result = tf.multiply(x1, x2)
# Intialize the Session
sess = tf.Session()
# Print the result
print(sess.run(result))
# Close the session
sess.close()
```

2.3 Additional Tensorflow References

https://www.tensorflow.org/install/install_sources#tested_source_configurations.

https://www.datacamp.com/community/tutorials/tensorflow-tutorial#basics

3.0 Anaconda

Anaconda is not available as a module and we strongly recommend that you install Anaconda by yourself to your home directory. The reason for this is to give you better control over the environment and packages. You have the option to use either Python 2.7 or Python 3 as in the examples below.

3.1 Python2 version is now obsolete. - Python 2.7

First chance to your proper working directory, here we assume that you are using $PROJ:

```
cd $PROJ
```

Then, get the installation packages for Anaconda:

```
wget https://repo.continuum.io/archive/Anaconda2-4.4.0-Linux-x86_64.sh
```

Run the installer:

```
sh ./Anaconda2-4.4.0-Linux-x86_64.sh -p $PROJ/anaconda2
```

Following line in .bashrc will make Anaconda your default Python. To disable, use #.

```
echo 'export PATH="$PROJ/anaconda2/bin:$PATH"' >> ~/.bashrc
```

Remove installation packages:

```
rm Anaconda2-4.4.0-Linux-x86_64.sh
```

Source your bashrc:

```
source ~/.bashrc
```

3.1 Python 3

First, get the installation packages for Anaconda.

```
wget https://repo.continuum.io/archive/Anaconda3-4.4.0-Linux-x86_64.sh
```

Run the installer:

```
sh ./Anaconda3-4.4.0-Linux-x86_64.sh -p /proj/$USER/anaconda3
```

Following line in .bashrc will make Anaconda your default Python. To disable, use #.

```
echo 'export PATH="/proj/$USER/anaconda3/bin:$PATH"' >> ~/.bashrc
```

Remove installation packages:

```
rm Anaconda2-4.4.0-Linux-x86_64.sh
```

Source your bashrc:

```
source ~/.bashrc
```

3.2 Setting up the environment

You can now create a conda environment for your project and install all the necessary packages to the environment. Note that you can have multiple environments with conflicting requirements without the need to reinstall Anaconda. See Tensorflow Python virtual environment above as an example of a similar use case.

Go to the project directory:

```
cd $PROJ
```

Create project:

```
conda create -y -n myProject
```

Activate your project:

```
source activate myProject
```

Install some additional packages, just like in Python virtualenv:

```
conda install -y <packages>
```

Or use pip, as you see fit:

```
pip install <packages>
```

When you need to activate your project again, then just:

```
source activate myProject
```

It is just that easy. Enjoy. Once you are done, then just:

```
conda deactivate
```

Sharing your environment

1. Activate the environment to export:
2. conda activate myenv. Note. Replace myenv with the name of the environment.
3. Export your active environment to a new file: conda env export > environment. yml.
4. Email or copy the exported environment.yml file to the other person.

3.3 Additional References

Conda User Guide

Machine Learning Anaconda

Conda Extra Packages

4.0 Comsol

For Comsol users, here is a specific example of a batch script when running Comsol in batch mode:

```
#!/bin/bash
#
#SBATCH --workdir=$WRKDIR
#SBATCH --time=04:00:00
#SBATCH --mem-per-cpu=2048
#SBATCH --ntasks=8
#### SLURM 8 processor COMSOL test to run for 4 hours.
module purge
module load COMSOL
# Change the below path to an actual temporary directory!
# To create one on the fly on local node storage, use:
# export TMPDIR=$(mktemp -d)
export TMPDIR="/path/to/temp/dir"
comsol -clustersimple batch -inputfile ${SLURM_JOB_NAME} -outputfile ${SLURM_JOB_NAME//\.mph/}.out.mph -batchlog
${SLURM_JOB_NAME}.${SLURM_JOB_ID}.comsol_log -tmpdir $TMPDIR
```

You can run Comsol in the interactive session, but the graphical visualization toolkit is not an optimum method to use cluster resources, and there are some limitations (including the use of OpenGL). Once you start an interactive session, Comsol can be started in the interactive session as follows:

```
srun --ntasks=8 --time=04:00:00 --pty bash
```

After the session starts, you can start Comsol as in any other Unix system:

```
comsol -3drendr sw
```

Note that Comsol will require .comsol, which is created the first time the binary is executed. As a result, the execution of the Comsol binary may end up dumping the core if the .comsol is not present.

5.0 MiniSat

If you need a MiniSat solver the most convenient way is to compile it to your own software repository, for example, $PROJ/MySoftware.

First, get the sources:

```
wget http://minisat.se/downloads/minisat-2.2.0.tar.gz
```

Unpack the sources to the proper place. Then load necessary modules:

```
module load GCC zlib
```

Set the environment to match the location of sources:

```
export MROOT=$PROJ/MySoftware/sources/minisat
```

Choose which one to compile:

```
cd { core | simp }
```

Compile static binary:

```
make rs
```

After which you can copy the binary to your program folder. If you wish, then you can create your own module for the program and even share it with your colleagues.

6.0 CosmoMC

Below you can find a step to step instructions for installing CosmoMC on Ukko2 and Kale. Please note that for Ukko2 the installation location should be in $PROJ. First, load necessary modules - in this case, OpenMPI does include the necessary dependencies:

```
 module load OpenMPI/3.1.1-GCC-7.3.0-2.30
```

Get CosmoMC:

```
git clone https://github.com/cmbant/CosmoMC.git
```

Compile CosmoMC from sources:

```
cd $PROJ/cosmoMC
```

```
make
```

Get COM likelihood code: plc-2.0:

```
wget http://irsa.ipac.caltech.edu/data/Planck/release_2/software/COM_Likelihood_Code-v2.0.R2.00.tar.bz2
```

Open package:

```
tar xvfj COM_Likelihood_Code-v2.*.tar.bz2
```

...and compile plc-2.0:

```
cd plc-2.0
```

```
 ./waf configure --lapack_mkl=${MKLROOT} --install_all_deps --extra_lib m
```

And finally, Install:

```
 ./waf install
```

Don't forget: If you wish, you can create your own module for the program and even share it with your colleagues.

7.0 SUMO

http://sumo.dlr.de/wiki/Installing/Linux_Build

First, load necessary modules:

```
module load GDAL/2.2.3-foss-2018a-Python-3.6.4
module load X11/20180131-GCCcore-6.4.0
```

Then change to the proper directory:

```
cd $PROJ
```

Install 3rd party libraries, FOX Toolkit:

```
wget ftp://ftp.fox-toolkit.org/pub/fox-1.6.57.tar.gz
```

```
cd fox-1.6.57/
```

```
./configure --prefix $PROJ
```

```
make install
```

```
cd $PROJ
```

Then time to install SUMO

```
git clone --recursive https://github.com/eclipse/sumo
```

```
cd sumo
```

```
git fetch origin refs/replace/*:refs/replace/*
```

```
export SUMO_HOME="$PROJ/sumo"
```

```
make -f Makefile.cvs
```

```
./configure --prefix $PROJ --with-fox-config=$PROJ/bin/fox-config
```

```
make install
```

Then you are done.

If you encounter an error about version.h (*example: https://github.com/eclipse/sumo/issues/3967*), then you have to do:

```
python sumo/tools/build/version.py
```

And then it should compile fine.

Don't forget: If you wish, you can create your own module for the program and even share it with your colleagues.

8.0 General Notes

8.1 IBM ILOG CPLEX

A free license is available for academic use:

https://www.ibm.com/developerworks/community/blogs/jfp/entry/CPLEX_Is_Free_For_Students

8.2 Turbomole

If Turbomole is set to use a semi-direct method, it will require fast or very fast IO to provide a performance boost. Conversely, any slowness in I/O will dramatically affect the performance. ~~This can be overcome by using local fast drives when available, or the creation of RAMdisk for the node. In either case, the content needs to be loaded to the node by sbcast.~~ This can be provided with Lustre for high throughput and low latencies, by storing data under /wrk/users/$USER. Turbomole may benefit from exclusive node reservations, and MPI implementation & performance tuning. Hence, slurm special flag may resolve some of the performance-related issues. However, bear in mind that reserving entire nodes exclusively is not a good idea if only minor portion of node core count is used:

```
#SBATCH --exclusive
```

8.3 Matlab

8.3.1 Kilosort

GPU based Spike sorting program Kilosort provides quite good GPU performance and simple installation. To start, user needs to clone the git repo, and then compile within Matlab development tools. In our case, clone git repo, then load up necessary modules for compilation:

```
module load gcccuda/2019b MATLAB/2019a CUDA/10.0.130
```

You can then start up matlab in interactive session (*mostly to avoid loading login node too much*), and proceed with compilation:

```
matlab -nodisplay -nodesktop
```

```
>> addpath(genpath('/wrk/$USER/path/here/Kilosort'))
>> cd('/wrk/$USER/path/here/Kilosort/CUDA')
>> run mexGPUall
```

Matlab will place your newly compiled packages in:

```
/wrk/$USER/path/here/Kilosort/CUDA
```

Then you have to alter the startup file (*and perhaps have a specific copy of it elsewhere*) to point to the correct packages:

```
/wrk/$USER/path/here/Kilosort/main_kilosort3.m
```

8.3.2 Phy GUI

For visualization you can use Phy GUI on the Infiniband capable VDI machines which have Kappa $WRKDIR mounted directly on them. They provide an excellent throughput/latencies and pretty good capacity of storage. You can use the Conda installation instructions straight from the box.

9.0 Development Tools

9.1 PrgEnv: Python Parallel memory consumption

By default, Parallel uses the Python multiprocessing module to fork separate Python worker processes to execute tasks concurrently on separate CPUs. This is a reasonable default for generic Python programs but it induces some overhead as the input and output data need to be serialized in a queue for communication with the worker processes.

9.2 PrgEnv: MPI Implementations and performance

IntelMPI is a commercial MPI implementation developed and supported by Intel. IntelMPI may provide a significant performance boost in certain applications compared to OpenMPI. Based on the 3.1 standard, it does support multiple interconnects. The free runtime environment is available through Intel: https://software.intel.com/en-us/intel-mpi-library.

MVAPICH2 is another MPI implementation, now available under the BSD-like license. It may provide a significant performance boost with some applications compared to OpenMPI, and in some cases over Intel implementation.

9.3 PrgEnv: Intel Compilers

Intel Parallel Studio XE compilers are available for free for classroom teaching and student use. Please see intel for details. For others, we have two floating licenses.

9.4 Profiling

Intel vTune is available through the Parallel Studio XE license agreement and is the default profiling tool at this time.

If you wish to do I/O profiling, please see details for Lustre.

9.5 Jupyter

Please see details about University JupyterHub implementation from the User Guide (*not available yet*).

https://zonca.github.io/2015/04/jupyterhub-hpc.html

9.6 Data Scientists - Julia

The user should install Julia packages locally. They are not yet stable enough to be installed as a global module.

Julia in HPC environment and parallel use: http://www.stochasticlifestyle.com/multi-node-parallelism-in-julia-on-an-hpc/

Julia sources etc: https://github.com/JuliaLang

Julia vs. Python - optimization: https://www.ibm.com/developerworks/community/blogs/jfp/entry/Python_Meets_Julia_Micro_Performance?lang=en

9.7 Environment Modules

You can create your own modules.

9.8 AMD Specific issues

Currently, only Carrington is AMD-based, but this may change in time.

9.8.1 Python libgomp Error (*resource temporarily unavailable*)

You may encounter the following issues with Python and libgomb.

```
joblib import Parallel, delayed
```

Idiom:

```
Parallel(n_jobs=num_cores)(delayed(...))
```

Resulting error:

```
libgomp: Thread creation failed: Resource temporarily unavailable
```

In this case, you have to change the MKL_THREADING_LAYER variable either before launching the job (*inherited by the batch/interactive session*) or prior Python threading:

```
MKL_THREADING_LAYER=GNU
```

10.0 Additional Quirks For Advanced Users

We are adding here some additional ideas of how you can get the most out of the system. Things listed here are not supported, and it is your own responsibility to create and maintain your own creations. Most things here do require some level of Linux understanding and some more than others. These are not intended for people new to the system.

10.1 Slurm and bash functions

One can do a whole lot with bash functions. However, if you ever thought that it would be nice to be able to start an interactive session directly from your desktop without first logging into the system - for example when you have $WRKDIR mounted to your Linux desktop, you might consider this kind of approach on your .bashrc profile:

```
function ukko2-interactive() {
ntasks=${1:-8}
mem=${2:-32}
workdirectory=/wrk/users/$USER
ssh -YA $USER@ukko2.cs.helsinki.fi -t "salloc -pshort -t1:00:00 -c${ntasks} --mem=${mem}G srun --chdir=$workdirectory -c${ntasks} --mem=${mem}G --pty /bin
/bash"
}
```

If you wish to try it out, you can add the function to your .bashrc and then export it:

```
export -f ukko2-interactive
```

Then just call it and there it goes...

```
ukko2-interactive 1 1
salloc: Pending job allocation 4457837
salloc: job 4457837 queued and waiting for resources
salloc: job 4457837 has been allocated resources
salloc: Granted job allocation 4457837
bash-4.2$
```

You can, of course, modify this according to what you need, but as basic, it would allow you to adjust several tasks and memory, but time and partition would be predefined. With a little bit of additional work, you could create yourself a batch job submitter which could allow you to create a batch script and submit batch jobs directly from your desktop.

If you need to access the environment from outside the university network, you would need to include a jump host to the above:

```
ssh -J $USER@pangolin.it.helsinki.fi
```

10.2 Code optimization

You should have a look at NLOpt for your code optimization needs. The package is not globally available at this time but you can easily compile it to your $PROJ. We will consider making it publicly available at a later time.

You can find I/O optimization and profiling hints from Lustre User Guide.

10.3. Array example: Boosting up Data Transfer

Why not move your files parallel in a batch job?

--array specifies the number of batch processes you would like to have. This correlates with the number of lines of the arrayparams.txt where you define the transfers. The script assumes that you use rsync, but this can be modified as required.

```
#!/bin/bash
#SBACTH -c 1
#SBATCH -t 1-0
#SBATCH -p short
#SBATCH --array=1-528

n=$SLURM_ARRAY_TASK_ID                  # define n
line=`sed "${n}q;d" arrayparams.txt`    # get n:th line (1-indexed) of the file

echo $line
time rsync -rah $line
```

File arrayparams.txt would contain the directories to transfer...

```
/scratch/$USER/foo /wrk/$USER/foo
/scratch/$USER/bar /wrk/$USER/bar
/scratch/$USER/oof /wrk/$USER/oof
/scratch/$USER/rab /wrk/$USER/rab
....
```

Then just submit the batch job and it does the I/O ops in parallel while you wait...

P.S. You can use this Array template for many, many different purposes, and you can use this as the temple.

11.0 Containers

Singularity is available on ukko2 and Kale compute nodes ready to be used. Note that the login node does not have singularity installed because we do not wish containers to be run there. At this time we do not have our own examples of how to build containers, nor do we have ready containers yet. However, you can use for example Dockers to develop containers and then use Singularity to run them on the clusters.

When you use Singularity containers on the clusters, please do set cache directory environment variable and point to a proper place (eg. /wrk/users /$USER). Default points to your $HOME and building container there will fill quota quickly:

```
SINGULARITY_CACHEDIR
```

Note that /wrk is not automatically visible to the running container. You will need to specify two options, first clear environment, and then set /wrk for the running environment (**Important to note**: interactive session on compute node does see /wrk normally, while batch job does not):

```
srun singularity run --cleanenv -B /wrk:/wrk <actual container stuff here>
```

Here are pretty good building instructions for the time being.

12.0 Scientist's Full Use Case: Trinotate Annotation of the Transcriptome of An Organism

Work in progress... (Juhana)

This tutorial describes a full use case of a scientist. We attempt an automatic annotation of the transcriptome of a mammal genome using a pipeline called Trinotate (https://github.com/Trinotate/Trinotate.github.io/wiki). We describe everything from installing individual pieces of software needed by the Trinotate pipeline to transferring your data and visualizing our results.

1. Install Trinotate: https://github.com/Trinotate/Trinotate.github.io/wiki

13.0 HPC Garage FAQ

13.1 My Tmux does not work, or Socket does not work on Lustre

Tmux does not like lustre /tmp dir. If this is changed to another/tmp location, it works. One should avoid creating sockets to Lustre.

```
alias tmux="tmux -S /run/user/$UID/tmux-socket"
```

13.2 Can one use Zarr or HDF5 on Lustre?

Yes, you can, these are like filesystems on a file. This causes no issues at all. You may get some performance difference by adjusting the stripe count, lfs getstripe -c filename shows current striping. See Lustre user guide for examples of how to alter stripe settings.

13.3 I use Conda but Cannot Remember Commands

You are lucky that Cheat Sheet is available for download. Handy helper for most common, and some less common commands and options for Conda users.