Introduction to HPC@UoP

Vincent Drach

School of Computing, Electronics and Mathematics
vincent.drach@plymouth.ac.uk

Task automatisation and data manipulation
- writing scripts -

UNIVERSITY OF
PLYMOUTH

Today's goals

- Develop a set of tools to automatise a typical workflow.
- Demonstrate how efficient and powerful is shell scripting
- Help to obtain reproducible results.

- See also Craig McNeile Data Science Tutorial using Python on GitHub :

https://github.com/cmcneile/HPC-tutorial/blob/main/HPC_python_tutorial.ipynb

- See also online documentation:

https://ecm-research.plymouth.ac.uk/hpc/foseres-uop/index.html

- Special thanks to Craig McNeile, Davide Vadacchino, Laurence Bowes and Maxwell Gisborne

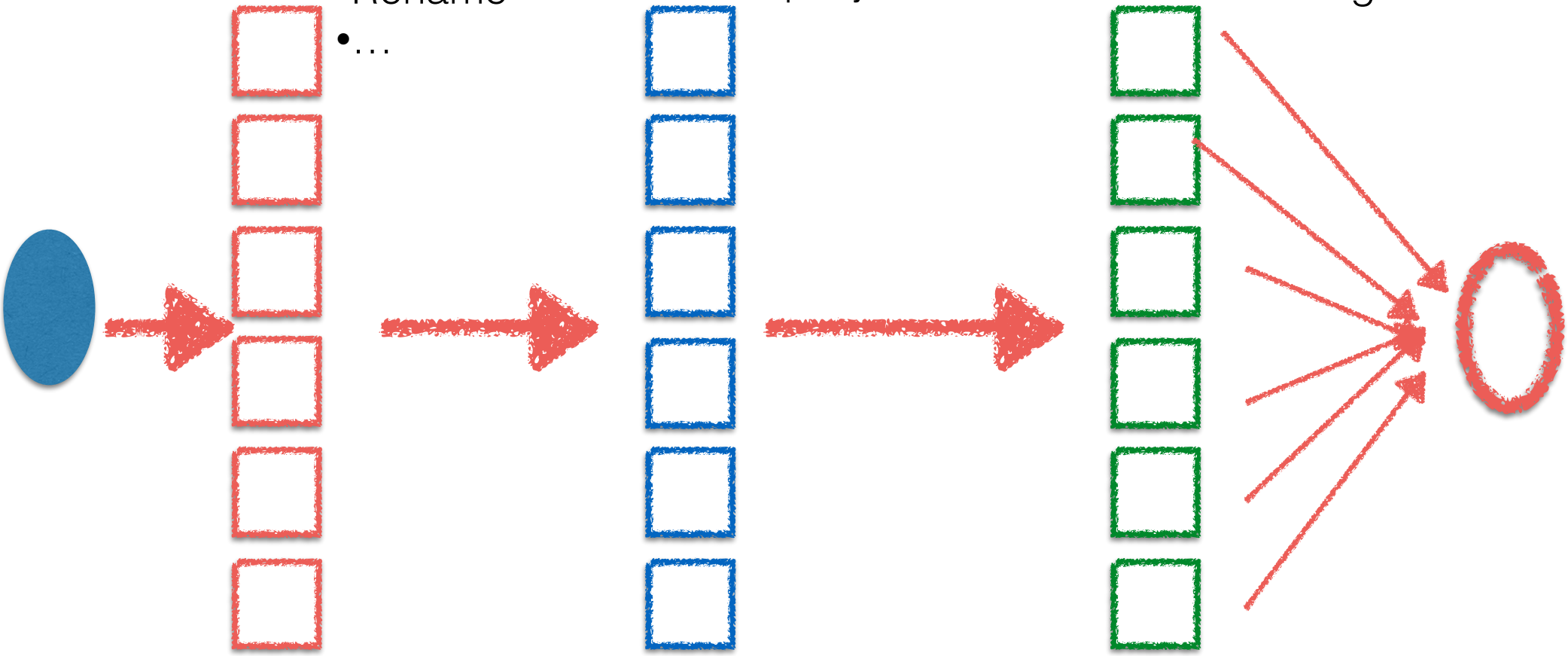
A typical workflow

Cloning
remote
repository

Preprocessing
- Remove corrupted
- Rename
- …

Processing
(Computing nodes)
- Smoothing
- Multiple job submission

Postprocessing
- Check crashed run
- Finding timing vs size
- Collect log files



git, tar

md5sum, cp, for

sed, parallel

grep,find, > and >>, awk?

Getting your data

- The original data you want to use on the cluster might be located on your own machine or in some repository (maybe hosted on GitHub).

- If the data are on your own machine: copy them using *scp* or *rsync*
- If the data are hosted on GitHub:
  - Clone the repository :
    git clone https://github.com/vdrach/training_automatisation.git

git is a free and open source distributed version control system designed to handle everything from small to very large projects with speed and efficiency.

Create a file on the Windows machine using notepad++, and add the git clone command to it.

Inspect the repository.

- Explore the files that have been downloaded.

- The original raw data are in ./data.
- They are stored in a tar (or more precisely tar.gz) file.

tar file: a collection of files wrapped up in one single file for easy storage.
tar files are often compressed after being created, giving it the .tar.gz file extension.

You can open the file by doing:

```
tar -xzf tarfile
```

- -v: Enables verbose mode, showing the progress of the command
- -x: Extract
- -z: Uses gzip, omit this if you just have a .tar
- -f: specifies file input, rather than STDIN

- Store the command to decompress/extract/unzip the tar file in your notepad++ file.

Inspect the repository.

- Explore the files that have been downloaded.
- The Python program that will use is in the root of the repository.

We are not going to discuss the code itself.

Load the following python modules:

```
module purge
module load python/3.9.12-nxrltcs  py-numpy/1.19.5-4wflrs2  py-packaging/21.3-hyqp3am py-matplotlib/3.5.3-k3xfyzm
```

Then execute the code by typing:

```
python simulation_ex.py input N M Niter outfile
```

Exercise: run the python code on one of the image in the data set.

- Input: path to the image that you want to process
- N: number of pixel in x-direction
- M: number of pixels in y-direction
- Niter: Number of smoothing iterations
- outfile: name of the file in which you want to save the output

Already to much to remember?

- Create a script on the Windows machine called *get_and_unzip.sh* :

- #!/bin/bash (called a shebang) tells the terminal in which version of the shell language should be used to run the script.

- Copy the file on foseres, transform it into an executable script by typing:

```
chmod u+x get_and_unzip.sh
```

- Remove the directory that you previously unzipped:

```
rm -r ./data
```

- Execute your first script by typing:

```
./get_and_unzip.sh
```

Congratulations ! You just wrote your first shell (bash) script.

Exercise

- Create a script on the Windows machine called load_modules.sh to load the python module that you will be using all the time.

- You can now clone the repository, unzip the data, and load your favourite modules by executing 2 scripts that you can decide how to name.

Exercise

- Create a script on the Windows machine called *load_modules.sh* to load the python module that you will be using all the time.

- You can now clone the repository, unzip the data, and load your favourite modules by executing 2 scripts that you can decide how to name.

Inspect the repository.

- Explore the files that have been downloaded.

- The original raw data are in ./data contains images and a file called checksum

Display the content of the checksum file.

It contains a list of all the files in the directory, and for each of them, a long character chain called a md5 sum.

You just need to know 2 things:

1. The command *md5sum* filename, returns the md5 sum of a file.
2. The *md5sum* **of any file** is designed so that two very similar files have very different *md5sum*

Calculating md5 sum is a way to check the integrity of a file, or to compare files.

A few useful commands:

- For loop

```
for i in 1 2 5
do
echo $i
done
```

```
for i in `seq 1 10`
do
echo $i
done
```

```
for f in *.jpg
do
echo $f
done
```

- Capture the output of a command in a variable

```
myvar=`pwd`
echo ${myvar}
```

- grep

```
grep np simulation_ex.py
```

- Count lines

```
ls *.jpg | wc -l
```

- Tests

```
x=0
if (( ${x} == 0 )); then
 echo ${x}
fi
```

Awk/cut

```
cat ex_awk

awk ‘{print $1}’ ex_awk
awk ‘{print $2}’ ex_awk
awk ‘{print “hello there”,$3}’ ex_awk

cat ex_awk2
awk –F','  '{print $1}'  ex_awk2
```

- Similar functionality can be achieved with “cut”

```
cut –d’ ‘ –f1
cut –d’ ‘ –f2
```

Exercise

- Some of the images are corrupted - its *md5sum* does not match the one recorded in the *checksum* file

- Write a script *cleanup.sh* that automatically find the files that are corrupted and remove them.

Use: for, grep, wc, if and awk.

Preprocess

- For convenience, we want to rename all the images, and add an additional information in the filename.
- We want to preserve all the raw data, so will perform copies in a new directory
- We want to rename as follows:

  img_hpc.jpg -> 1-200_500.jpg
  img_hpc_UoP -> 2-500_500.jpg
  ….

  where the two numbers separated by _ are the number of pixel in each direction.

  You are provided a tool to get them

  ```
  python get_N_M.py file.jpg
  ```

  returns eg 1024_429

-

Exercise

- Write a script that preprocess the folder ./data and create a new folder call ./data_preprocessed
- We want to keep in a logfile a dictionary between the old and new name, so that we can *reproduce* all the steps of our workflow later (and debug).

Processing:

- The goal is to process all the images with the Python program that has been provided.

Strategy ?

1. We now want to create a batch (not bash!!) script that run our python executable on a single image on a computing node
2. Once it is working, we will find how to create one batch script for each image and submit them automatically (use *sed*).
3. Then we will realise that it is a waste of time… In each job only one image is processed while there are 31 cores idling… We will use the *parallel* command to process 32 images per batch script

Sed (stream editor)

sed is a stream editor. A stream editor is used to perform basic text transformations on an input stream

Example:

```
i=12
sed “s/jobname/jobname${i}/g” job_template
```

will replace all the occurence of job name in job_template by jobname12 and display the result in the terminal (**the file job_template is not modified).**

GNU parallel

GNU *parallel* is a shell tool for executing jobs in parallel using one or more computers.

Example:

```
module load parallel
```

```
cat commands.sh
parallel < commands.sh
```

See the documentation online for other use or the page on the website (https://ecm-research.plymouth.ac.uk/hpc/foseres-uop/index.html) developed by D. Vadacchino.

Postprocessing

- Write a command that concatenate all the logfile into a single log
- Write a command to check that all the runs are complete.
- Copy all the images you have produced back on Windows (with one command)
- Find what is the simulation that took most time
- Store the list of the image that have been produced in a file (try to use *find*)