HPC Job Submission Made Easy

- Motivation: Automate the process of launching several jobs on the HPC using the PBS queueing system

- We spend so much time doing the same things over and over again:
  - Ssh'ing into the HPC
  - Taring - Untaring
  - Copying the .rst files and results back and forth
  - Editing the submission scripts
  - Rinse & Repeat

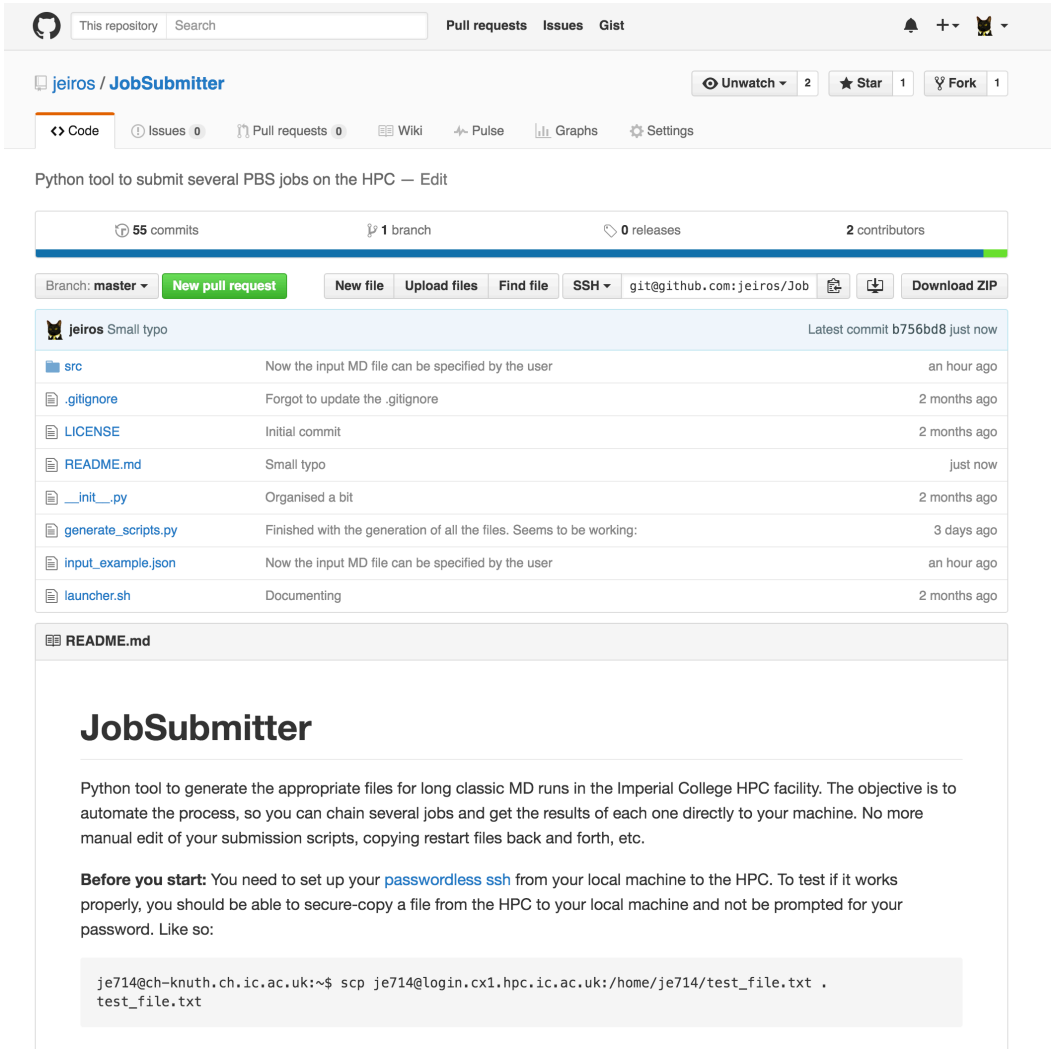
“I had a problem so I automated it. Now I have two problems.”

- JobSubmitter is a Python program that allows you to specify all of the settings in one JSON file.

- Get it from https://github.com/jeiros/JobSubmitter

- Download as a ZIP or easy as:

```
$ git clone https://github.com/jeiros/JobSubmitter
```



JSON file (http://www.json.org/)

- JavaScript Object Notation. It is a widespread data format – very easy to read, write and parse.
- An example input file is provided: *input_example.json*
- At the moment the only scheduler that can be used is ‘pbs’.
- Three sections:
  - Pbs settings: Used to create the headers of the .pbs files
  - Simulation details: Name of the system, different files to use, times, pre simulation commands, etc.
  - Local machine: Details of your local machine to automate the data copying process.

- Prerequisite: Passwordless SSH.
- This is done via an RSA key pair.
- If properly set up, you should be able to ssh into the HPC or do secure copy commands from it without being prompted for your password.

SSH login without password

Your aim

You want to use Linux and OpenSSH to automate your tasks. Therefore you need an **automatic** login from host A / user a to Host B / user b. You don't want to enter any passwords, because you want to call `ssh` from a within a shell script.

How to do it

First log in on A as user a and generate a pair of authentication keys. Do not enter a passphrase:

```
a@A:~> ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/a/.ssh/id_rsa):
Created directory '/home/a/.ssh'.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/a/.ssh/id_rsa.
Your public key has been saved in /home/a/.ssh/id_rsa.pub.
The key fingerprint is:
3e:4f:05:79:3a:9f:96:7c:3b:ad:e9:58:37:bc:37:e4 a@A
```

Now use `ssh` to create a directory `~/.ssh` as user b on B. (The directory may already exist, which is fine):

```
a@A:~> ssh b@B mkdir -p .ssh
b@B's password:
```

Finally append a's new public key to `b@B:.ssh/authorized_keys` and enter b's password one last time:

```
a@A:~> cat .ssh/id_rsa.pub | ssh b@B 'cat >> .ssh/authorized_keys'
b@B's password:
```

From now on you can log into B as b from A as a without password:

```
a@A:~> ssh b@B
```

A note from one of our readers: Depending on your version of SSH you might also have to do the following changes:

- Put the public key in `.ssh/authorized_keys2`
- Change the permissions of `.ssh` to `700`
- Change the permissions of `.ssh/authorized_keys2` to `640`

1. Generate all the submission files with the program
2. Copy them along to the HPC w/ the 'launcher.sh' scripts and any other necessary input files (prmtops, inpcrds / restarts, and MD inputs).
3. Launch all the .pbs scripts at one with `bash launcher.sh` in the HPC job directory
4. Collect the results in your local machine as .tgz files

```
#!/bin/bash
first=true
for file in *job*.pbs; do
    if [ "$first" = true ]; then
        first=false
        ID=$(qsub $file)
    else
        ID=$(qsub -W depend=afterok:$ID $file)
    fi
done
```

- Since everyone uses different protocols, I've added a section in the settings file for you to specify any commands you like prior to the production run.

- They have to be specified in the settings file like so:

```
"pre_simulation_cmd": [
      "command1",
      "command2",
      "command3"
],
```

- Note the lack of the comma at the end of command3. The JSON format is very strict with the syntax, so preserve the one in the example file.
- These can be run in the GPU (in the first .pbs file) or in the CPU (a bash script is written).