Como usar el cluster sin morir en el intento

Get it started

El cluster (*brick*) consta de momento de tres nodos (*brick01, brick02 y brick03*). Lo que sigue describe la operativa básica para ejecutar tareas en estas maquinas de una manera ordenada

El directorio */nas* de detritus se monta como */home* en cada uno de los *bricks* por lo que todo lo que se modifique en *detritus:/nas/user* cambiara automaticamente en *brick0X:/home/user*

La presentacion inicial (hecha con pinpoint y exportada a pdf) se puede bajar de aqui: http://detritus.fundacioace.com/files/cluster.pdf

como hacer que los nodos no pidan password

We don't talk anymore

Esto es básico para el funcionamiento correcto del resto de las herramientas

Empezamos creando una clave RSA en nuestro directorio de detritus y moviendo la clave publica a un archivo que podamos identificar facilmente

```
$ ssh-keygen
$ mv $HOME/.ssh/id_rsa.pub $HOME/.ssh/detritus.pub
```

Ahora añadimos esta clave a la lista de claves autorizadas en el HOME de los nodos. Como hasta ahora no hay ninguna se puede hacer con *cp*. En caso de haber alguna se podría hacer con *cat*. Y no podemos olvidarnos de dar los permisos correctos!

En mi caso:

```
$ mkdir /nas/osotolongo/.ssh
$ chmod 700 /nas/osotolongo/.ssh
$ cp ~/.ssh/detritus.pub /nas/osotolongo/.ssh/authorized_keys
$ chmod 600 /nas/osotolongo/.ssh/authorized_keys
```

A partir de ahora, cuando se entra de *detritus* a cualquiera de los nodos no deberia ser necesario introducir ningun password.

como ejecutar la misma orden en varios nodos del cluster (pssh y mpssh)

Nobody Wants To Be Lonely

pssh

- primero crear un archivo con la lista de hosts:

```
[osotolongo@detritus ~]$ cat host.pssh
osotolongo@brick01
osotolongo@brick02
osotolongo@brick03
```

- despues ejecutar un comando en todas las maquinas

```
[osotolongo@detritus ~]$ pssh -h host.pssh 'ps ax | grep ssh'
[1] 13:11:46 [SUCCESS] osotolongo@brick03
[2] 13:11:46 [SUCCESS] osotolongo@brick02
[3] 13:11:46 [SUCCESS] osotolongo@brick01
```

mpssh

- la syntaxis cambia ligeramente, y la salida tambien

```
[osotolongo@detritus ~]$ mpssh -f host.pssh 'ps ax | grep ssh'
MPSSH - Mass Parallel Ssh Ver.1.3.3
(c)2005-2013 Nikolay Denev <ndenev@gmail.com>

  [*] read (3) hosts from the list
  [*] executing "ps ax | grep ssh" as user "osotolongo"
  [*] spawning 3 parallel ssh sessions

brick01 ->   1920 ?        Ss     0:00 /usr/sbin/sshd -D
brick01 ->  73087 ?        Ss     0:00 sshd: osotolongo [priv]
brick01 ->  73090 ?        S      0:00 sshd: osotolongo@notty
brick01 ->  73091 ?        Ss     0:00 bash -c ps ax | grep ssh
brick01 ->  73101 ?        S      0:00 grep ssh
brick02 ->   1887 ?        Ss     0:00 /usr/sbin/sshd -D
brick02 ->  21242 ?        Ss     0:00 sshd: osotolongo [priv]
brick02 ->  21245 ?        S      0:00 sshd: osotolongo@notty
brick02 ->  21246 ?        Ss     0:00 bash -c ps ax | grep ssh
brick02 ->  21256 ?        S      0:00 grep ssh
brick03 ->   1928 ?        Ss     0:00 /usr/sbin/sshd -D
brick03 ->  21270 ?        Ss     0:00 sshd: osotolongo [priv]
brick03 ->  21273 ?        S      0:00 sshd: osotolongo@notty
```

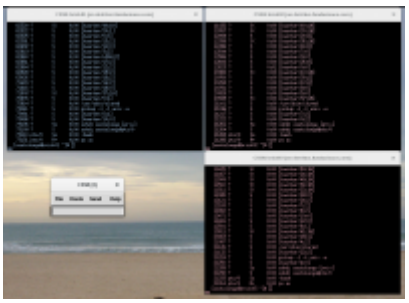
```
brick03 ->  21274 ?        Ss     0:00 bash -c ps ax | grep ssh
brick03 ->  21284 ?        S      0:00 grep ssh

  Done. 3 hosts processed.
```

o como hacerlo mas interactivo (cluster-ssh)

Dance again

Una imagen vale mas que mil palabras

```
$ cssh brick
```



Esta herramienta permite, cuando se escribe en la ventana comun, escribir lo mismo en todas las terminales abiertas o en una sola si se selecciona expresamente.

Slurm

This is what you came for

srun:

Este comando es capaz de lanzar un proceso en varios nodos del cluster al mismo tiempo

```
[osotolongo@detritus ~]$ srun -N3 -l /bin/hostname
0: brick01
2: brick03
1: brick02
```

sbatch:

Este comando lanza un script (tipicamente una lista de comandos *srun*) en el cluster. Es posible escoger el numero de procesos a lanzar e incluso los nodos en los cuales hacerlo

```
[osotolongo@detritus ~]$ cat test_script.sh
#!/bin/sh
#SBATCH --time=1
/bin/hostname
```

```
srun -l /bin/hostname
srun -l /bin/pwd
[osotolongo@detritus ~]$ sbatch -n6 -w "brick0[1-3]" -o test_stdout
test_script.sh
Submitted batch job 30
[osotolongo@detritus ~]$ cat /nas/osotolongo/test_stdout
brick01
1: brick01
2: brick02
4: brick03
3: brick02
5: brick03
0: brick01
0: /home/osotolongo
5: /home/osotolongo
3: /home/osotolongo
1: /home/osotolongo
4: /home/osotolongo
2: /home/osotolongo
```

squeue:

Este comando permite ver los trabajos que se han lanzado e informacion sobre ellos

```
[osotolongo@detritus ~]$ cat test_script.sh
#!/bin/sh
#SBATCH --time=1
/bin/hostname
srun -l /bin/hostname
sleep 20
srun -l /bin/pwd
[osotolongo@detritus ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
[osotolongo@detritus ~]$ sbatch -n6 -w "brick0[1-3]" -o test_stdout
test_script.sh
Submitted batch job 35
[osotolongo@detritus ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
                35     debug test_scr osotolon  R       0:01      3
brick[01-03]
```

scancel:

OK, y cuando la caguemos podemos usar *scancel* para parar los trabajos. El kill de toda la vida pero a traves de un workload manager

```
[osotolongo@detritus ~]$ cat test_script.sh
#!/bin/sh
/bin/hostname
```

```
srun -l /bin/hostname
sleep 200
srun -l /bin/pwd
[osotolongo@detritus ~]$ sbatch -n6 -w "brick0[1-3]" -o test_stdout
test_script.sh
Submitted batch job 36
[osotolongo@detritus ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
                36     debug test_scr osotolon  R       0:03      3
brick[01-03]
[osotolongo@detritus ~]$ scancel 36
[osotolongo@detritus ~]$ squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES
NODELIST(REASON)
```

https://slurm.schedmd.com/quickstart.html

mas: https://wiki.fysik.dtu.dk/niflheim/SLURM#configure-slurm

HPC::Runner::Slurm (AKA the hat trick)

Super duper love

Hay una forma sencilla de lanzar una lista de scripts utilizando unwrapper escrito en Perl : HPC::Runner::Slurm

Importante: los bricks necesitan los paths

Para que funcione esto hay que exportar en *detritus* los *paths* de las librerias Perl. Esto es, al principio de **todo** habria que añadir:

```
export PERL5LIB=$PERL5LIB:/usr/local/perl5/lib
export PATH=$PATH:/usr/local/perl5/bin
```

~~como me ha costao el jodio~~

O bien añadir las lineas al *.bashrc* o *.bash_profile* (el que se este usando)

Generalidades

Primeramente ha de hacerse un archivo con la lista de comandos a ejecutar.

```
job1
job2
job3
job4
```

```
# Lets tell slurmrunner.pl to execute jobs5-8 AFTER jobs1-4 have completed
wait
job5
job6
job7
job8
```

Ahora se envia los comandos a slurm para que los ejecute en los nodos

```
$ slurmrunner.pl --infile /path/to/fileofcommands --outdir slurmoutput --
jobname slurmjob
```

Ejemplo de paralelizacion basica

Supongamos que tenemos un script que ejecuta secuencialmente 22 ordenes, cada una por cada cromosoma,

```
for i in {1..22}
do
vcftools --gzvcf
/nas/marato/1KGP/ALL/chr$i.phase1_release_v3.20101123.snps_indels_svs.genoty
pes.refpanel.ALL.vcf.gz --snps SNPlist.txt --recode --out HapMap.vcf.chr$i
plink --vcf HapMap.vcf.chr$i.recode.vcf --make-bed --out HapMap.chr$i
rm HapMap.vcf.chr$i.recode.vcf
done
```

Para ejecutar estas tareas en el cluster habria que hacer algo como,

```
for i in {1..22}
do
echo "vcftools --gzvcf
/nas/marato/1KGP/ALL/chr$i.phase1_release_v3.20101123.snps_indels_svs.genoty
pes.refpanel.ALL.vcf.gz --snps /path/to/file/SNPlist.txt --recode --out
/path/to/file/HapMap.vcf.chr$i";
done > dale.in
echo "wait" >> dale.in
for i in {1..22}
do
echo "plink --vcf /path/to/file/HapMap.vcf.chr$i.recode.vcf --make-bed --out
/path/to/file/HapMap.chr$i" ;
done >> dale.in
echo "wait" >> dale.in
slurmrunner.pl --infile dale.in
for i in {1..22}
do
rm /path/to/file/HapMap.vcf.chr$i.recode.vcf
done
```

Ojo que,

- ~~el ultimo *wait* lo he puesto para esperar porque termine la ejecucion de todos los *jobs* pero no es imprescindible.~~
- los *rm* probablemente no sea correcto paralelizarlos pues al ser operaciones a disco deberian ejecutarse mas rapido secuencialmente
- hay que escribir los paths completos de todos los archivos. los *bricks* no tienen idea de donde esta nada!

Ejemplo completo de slurmrunner

(*pero minimo*)

Voy a empezar creando el problema. Digamos que en un directorio tengo unos cuantos archivos y quiero leer su contenido y crear unos archivos nuevos cuyo nombre dependa de ese contenido. Esto es muy basico pero puede ser un buen ejemplo.

Primero voy a crear los archivos de trabajo

```
$ mkdir /nas/osotolongo/cluster
$ cd /nas/osotolongo/cluster
$ for x in {1..20}; do echo "`shuf -i1-100 -n1`" > file"$x".txt; done
```

De forma que me quedan 20 archivos

```
$ $ ls -l
total 10
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file10.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file11.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file12.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file13.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file14.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file15.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file16.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file17.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file18.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file19.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file1.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file20.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file2.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file3.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file4.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file5.txt
-rw-rw---- 1 osotolongo osotolongo 2 Nov 17 18:32 file6.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file7.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file8.txt
-rw-rw---- 1 osotolongo osotolongo 3 Nov 17 18:32 file9.txt
```

que contienen cada uno un numero aleatorio entre 1 y 100.

```
$ cat file*
50
```

```
25
37
88
51
47
70
84
16
72
45
14
18
69
29
35
8
11
50
40
```

Ahora lo que quiero es leer el contenido de cada archivo y crear un archivo que se llame numberXX.txt, siendo XX el numero que he leido, y que contenga el nombre del archivo original.

(*esto es una chorrada pero es ilustrativo*)

La solucion sequencial es trivial

```
$ for x in file*; do y=$(cat ${x}); echo ${x} > number${y}.txt; done
```

Pero vamos a suponer que esta tarea consume tiempo y lo que queremos es que esto se haga en los distintos nodos. Asi que creamos un archivo con las distintas ordenes

```
$ for x in file*; do echo "y=\$(cat ${x}); echo "${x}" > number\${y}.txt";
done > dale.in
```

de manera que obtengamos estas ordenes,

```
$ cat dale.in
y=$(cat file10.txt); echo file10.txt > number${y}.txt
y=$(cat file11.txt); echo file11.txt > number${y}.txt
y=$(cat file12.txt); echo file12.txt > number${y}.txt
y=$(cat file13.txt); echo file13.txt > number${y}.txt
y=$(cat file14.txt); echo file14.txt > number${y}.txt
y=$(cat file15.txt); echo file15.txt > number${y}.txt
y=$(cat file16.txt); echo file16.txt > number${y}.txt
y=$(cat file17.txt); echo file17.txt > number${y}.txt
y=$(cat file18.txt); echo file18.txt > number${y}.txt
y=$(cat file19.txt); echo file19.txt > number${y}.txt
y=$(cat file1.txt); echo file1.txt > number${y}.txt
y=$(cat file20.txt); echo file20.txt > number${y}.txt
y=$(cat file2.txt); echo file2.txt > number${y}.txt
```

```
y=$(cat file3.txt); echo file3.txt > number${y}.txt
y=$(cat file4.txt); echo file4.txt > number${y}.txt
y=$(cat file5.txt); echo file5.txt > number${y}.txt
y=$(cat file6.txt); echo file6.txt > number${y}.txt
y=$(cat file7.txt); echo file7.txt > number${y}.txt
y=$(cat file8.txt); echo file8.txt > number${y}.txt
y=$(cat file9.txt); echo file9.txt > number${y}.txt
```

Esto es precisamente lo que queremos, la variable *$x* se ha interpretado pero la variable *$y* no, esta la queremos averiguar en los nodos. Asi, la operacion de ordenar los archivos es secuencial, pero la operacion de hacer algo con esos archivos (que supuestamente es la que consume tiempo) la paralelizamos.

pues vamos a probar,

Primero exporto el *enviroment*,

```
export PERL5LIB=$PERL5LIB:/usr/local/perl5/lib
export PATH=$PATH:/usr/local/perl5/bin
```

y ahora lanzo los procesos

```
$ slurmrunner.pl --infile dale.in --outdir test --jobname test
Submitting job /nas/osotolongo/cluster/test/001_test.sh
    With Slurm jobid 99
Submitting job /nas/osotolongo/cluster/test/002_test.sh
    With Slurm jobid 100
Submitting job /nas/osotolongo/cluster/test/003_test.sh
    With Slurm jobid 101
```

El sistema ha repartido todos los trabajos entre los nodos y los ha lanzado.

El slurmrunner ha creado el directorio *test* con todo el output. Y por supuesto ha resuelto el problema planteado:

```
$ ls
dale.in     file12.txt  file15.txt  file18.txt  file20.txt  file4.txt
file7.txt  number11.txt  number18.txt  number35.txt  number45.txt
number51.txt  number72.txt  number8.txt
file10.txt  file13.txt  file16.txt  file19.txt  file2.txt   file5.txt
file8.txt  number14.txt  number25.txt  number37.txt  number47.txt
number69.txt  number84.txt  test
file11.txt  file14.txt  file17.txt  file1.txt   file3.txt   file6.txt
file9.txt  number16.txt  number29.txt  number40.txt  number50.txt
number70.txt  number88.txt

$ cat number*
file7.txt
file20.txt
file18.txt
file2.txt
```

```
file11.txt
file4.txt
file5.txt
file12.txt
file9.txt
file1.txt
file15.txt
file8.txt
file14.txt
file3.txt
file16.txt
file19.txt
file17.txt
file13.txt
file6.txt
```

Enviar un email de aviso

Si los procesos que queremos enviar consumen mucho tiempo es bastante incomodo tener que estar comprobando todo el tiempo el estado en que se encuentran. Lo mas practico seria que cuando terminaran los jobs nos enviara un email. Hay una manera de configurar slurm para hacer esto cuando se lanza un *sbatch* pero no he encontrado una manera sencilla de hacerlo con *slurmrunner*. Asi que me he pensado un workaround.

El problema es que los jobs se envian a los nodos y la unica manera que ver que esta pasando es utilizando *squeue*. Asi que lo que he hecho es monitorizar que cuando todos los jobs terminen se envíe un email.

```
$ cat /nas/software/mail_after_queue.sh
#!/bin/sh
ARGS='$*'
if [ -z "$*" ]; then
  ARGS='-'
fi
msg=$(cat -- $ARGS)
u=$(echo $USER | cut -c1-8)
a=$(squeue | grep -v JOBID)
while [ ! -z "${a// }" ]; do a=$(squeue | grep $u); done
if [ ! -z "${msg// }" ]; then echo "${msg}." | mail -s "jobs done" $USER; fi
```

El problema es que si se envia mas de un trabajo todos los emails se enviaran cuando el ultimo trabajo haya terminado y no antes. Fix Me!

El script se usa añadiendolo tras la salida de *slurmrunner*:

```
$ slurmrunner.pl --infile dale.in --outdir test --jobname test |
/nas/software/mail_after_queue.sh &
```

Logs y troubleshooting

Todos los archivos intermedios y los logs estan (segun hemos indicado con *–outdir*) en el directorio *test*

```
test/001_test.in
test/001_test.sh
test/002_test.in
test/002_test.sh
test/003_test.in
test/003_test.sh
test/2016-11-17-001_test/2016-11-17-CMD_001.log
test/2016-11-17-001_test/2016-11-17-CMD_002.log
test/2016-11-17-001_test/2016-11-17-CMD_003.log
test/2016-11-17-001_test/2016-11-17-CMD_004.log
test/2016-11-17-001_test/2016-11-17-CMD_005.log
test/2016-11-17-001_test/2016-11-17-CMD_006.log
test/2016-11-17-001_test/2016-11-17-CMD_007.log
test/2016-11-17-001_test/2016-11-17-CMD_008.log
test/2016-11-17-001_test/MAIN_2016-11-17.log
test/2016-11-17-002_test/2016-11-17-CMD_001.log
test/2016-11-17-002_test/2016-11-17-CMD_002.log
test/2016-11-17-002_test/2016-11-17-CMD_003.log
test/2016-11-17-002_test/2016-11-17-CMD_004.log
test/2016-11-17-002_test/2016-11-17-CMD_005.log
test/2016-11-17-002_test/2016-11-17-CMD_006.log
test/2016-11-17-002_test/2016-11-17-CMD_007.log
test/2016-11-17-002_test/2016-11-17-CMD_008.log
test/2016-11-17-002_test/MAIN_2016-11-17.log
test/2016-11-17-003_test/2016-11-17-CMD_001.log
test/2016-11-17-003_test/2016-11-17-CMD_002.log
test/2016-11-17-003_test/2016-11-17-CMD_003.log
test/2016-11-17-003_test/2016-11-17-CMD_004.log
test/2016-11-17-003_test/MAIN_2016-11-17.log
test/2016-11-17-slurm_logs/001_test.log
test/2016-11-17-slurm_logs/002_test.log
test/2016-11-17-slurm_logs/003_test.log
test/2016-11-17-slurm_logs/2016-11-17
test/2016-11-17-slurm_logs/process_table.md
```

Como puede verse el wrapper ha separado las tareas en grupos que lanza en cada nodo y crea los lanzadores apropiados

```
$ cat test/001_test.in
y=$(cat file10.txt); echo file10.txt > number${y}.txt
y=$(cat file11.txt); echo file11.txt > number${y}.txt
y=$(cat file12.txt); echo file12.txt > number${y}.txt
y=$(cat file13.txt); echo file13.txt > number${y}.txt
y=$(cat file14.txt); echo file14.txt > number${y}.txt
y=$(cat file15.txt); echo file15.txt > number${y}.txt
```

```
y=$(cat file16.txt); echo file16.txt > number${y}.txt
y=$(cat file17.txt); echo file17.txt > number${y}.txt
```

```
$ cat test/001_test.sh
#!/bin/bash
#
#SBATCH --share
#SBATCH --get-user-env
#SBATCH --job-name=001_test
#SBATCH --output=/nas/osotolongo/cluster/test/2016-11-17-
slurm_logs/001_test.log
#SBATCH --cpus-per-task=4
#SBATCH --time=04:00:00
cd /nas/osotolongo/cluster
mcerunner.pl --procs 4 --infile /nas/osotolongo/cluster/test/001_test.in --
outdir /nas/osotolongo/cluster/test --logname 001_test --process_table
/nas/osotolongo/cluster/test/2016-11-17-slurm_logs/process_table.md --
metastr
'{"command_count":"1-8","job_batches":"1/3","batch":"1","batch_count":"1/3",
"commands":8,"total_batches":"3","total_processes":"20","jobname":"test"}'
```

Tambien hay una lista de los jobs enviados

```
$ cat test/2016-11-17-slurm_logs/2016-11-17
2016/11/17 18:50:07: DEBUG Submitted batch job 99
 2016/11/17 18:50:07: DEBUG Submitted batch job 100
 2016/11/17 18:50:07: DEBUG Submitted batch job 101
```

y de informacion sobre los procesos

```
$ cat test/2016-11-17-slurm_logs/process_table.md
### y=$(cat file10.txt); echo file10.txt > number${y}.txt
|122914|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 00 seconds|
### y=$(cat file11.txt); echo file11.txt > number${y}.txt
|122915|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 00 seconds|
### y=$(cat file12.txt); echo file12.txt > number${y}.txt
|122916|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 00 seconds|
### y=$(cat file13.txt); echo file13.txt > number${y}.txt
|122917|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file14.txt); echo file14.txt > number${y}.txt
|122926|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file15.txt); echo file15.txt > number${y}.txt
|122927|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file16.txt); echo file16.txt > number${y}.txt
|122930|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds
### y=$(cat file18.txt); echo file18.txt > number${y}.txt
|65217|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file20.txt); echo file20.txt > number${y}.txt
|65220|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file17.txt); echo file17.txt > number${y}.txt
|122931|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
```

```
### y=$(cat file2.txt); echo file2.txt > number${y}.txt
|65229|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file3.txt); echo file3.txt > number${y}.txt
|65230|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file5.txt); echo file5.txt > number${y}.txt
|65236|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file4.txt); echo file4.txt > number${y}.txt
|65233|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file9.txt); echo file9.txt > number${y}.txt
|122991|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file8.txt); echo file8.txt > number${y}.txt
|122990|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file6.txt); echo file6.txt > number${y}.txt
|122988|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
### y=$(cat file7.txt); echo file7.txt > number${y}.txt
|122989|0|0 years, 00 months, 0 days, 00 hours, 00 minutes, 01 seconds|
```

y un monton de logs con la salida (incluyendo *warnings* y *errors*) que deberia permitir ver con una inspeccion rapida si algo ha salido mal y que es

Docs

Mas info en,

- https://metacpan.org/pod/HPC::Runner::Slurm
- https://metacpan.org/pod/HPC::Runner::Scheduler
- https://metacpan.org/pod/distribution/HPC-Runner-Scheduler/lib/HPC/Runner/Usage.pod

Ejemplo de Begonia

```
[bego@detritus bego]$ cat script_GWAS.hapmap.slurm
#!/bin/bash

dirFICHEROSCOMUNES=/nas/marato/ARIC2/FicherosComunes
db_name=SHARE_MESA_c1_LVH_founders
name=MESA
dirwork=/nas/marato/MESA/QC/bego

### Step 1.5 PCA con HapMap (db + HapMap)
#Preparo lista de SNPs de DB
#Las selecciono por cromosoma de la referencia de HapMap de 1000G

for i in {1..22}
do
echo "vcftools --gzvcf
/nas/marato/1KGP/ALL/chr$i.phase1_release_v3.20101123.snps_indels_svs.genoty
pes.refpanel.ALL.vcf.gz --snps ${dirwork}/SNPlist.txt --recode --out
${dirwork}/HapMap.vcf.chr$i";
```

```
done > ${dirwork}/vcftools.script
echo "wait" >> ${dirwork}/vcftools.script

for i in {1..22}
do
echo "plink --vcf ${dirwork}/HapMap.vcf.chr$i.recode.vcf --make-bed --out
${dirwork}/HapMap.chr$i";
done >> ${dirwork}/vcftools.script
echo "wait" >> ${dirwork}/vcftools.script

for i in {1..22}
do
echo "rm ${dirwork}/HapMap.vcf.chr$i.recode.vcf";
done >> ${dirwork}/vcftools.script
echo "wait" >> ${dirwork}/vcftools.script

slurmrunner.pl --infile ${dirwork}/vcftools.script --outdir
${dirwork}/Slurm_Output_gwas1 -jobname gwas1_bh
```

Esto no funcionara nunca! *jajaja*

```
for i in {2..22}
do
echo "${dirwork}/HapMap.chr$i.bed ${dirwork}/HapMap.chr$i.bim
${dirwork}/HapMap.chr$i.fam >> ${dirwork}/ListaArchivos.txt";
done >> ${dirwork}/vcftools.script
echo "wait" >> ${dirwork}/vcftools.script
```

Pendiente: Slurm templates

```
#SBATCH --get-user-env
#SBATCH --mail-type=END,FAIL      # notifications for job done & fail
#SBATCH --mail-user=myemail@harvard.edu # send-to address
```