



CANCER  
RESEARCH  
UK

CAMBRIDGE  
INSTITUTE

# CRUK cluster practical sessions (SLURM)

Part I – processes & scripts



UNIVERSITY OF  
CAMBRIDGE

# login

Log in to the head node, clust1-headnode, using **ssh** and your usual user name & password.

```
SSH Secure Shell 3.2.9 (Build 283)
Copyright (c) 2000-2003 SSH Communications Security Corp - http://www.ssh.com/
```

```
This copy of SSH Secure Shell is a non-commercial version.
This version does not include PKI and PKCS #11 functionality.
```

```
Last login: Mon Sep 19 10:44:07 2016 from bp7r25j.cri.camres.org
[user@cluster ~]$
```

You're ready to start.

## navigate

Find out where you are using **pwd**.

Make a directory (**mkdir**) and move into it  
**(cd)**

```
[user@cluster ~]$ pwd  
/home/user  
[user@cluster ~]$ mkdir training  
[user@cluster ~]$ cd training/  
[user@cluster training]$
```

## processes

You can see your current processes using **ps**.

```
[user@cluster training]$ ps
  PID TTY      TIME CMD
14859 pts/22    00:00:00 bash
18511 pts/22    00:00:00 ps
```

You can see what else *this* computer is doing using **top**

```
[user@cluster training]$ top
```

# top output

**top** uses the whole screen. Type 'q' to get your screen back.

```
top - 16:26:38 up 58 days, 22:33, 36 users,  load average: 0.12, 0.14, 0.12
Tasks: 618 total, 1 running, 617 sleeping, 0 stopped, 0 zombie
Cpu(s): 0.1%us, 0.2%sy, 0.0%ni, 99.5%id, 0.2%wa, 0.0%hi, 0.0%si, 0.0%st
Mem: 16437908k total, 10473016k used, 5964892k free, 2611564k buffers
Swap: 16779852k total, 162896k used, 16616956k free, 2158536k cached

      PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+   COMMAND
      975 root      0 -20  22712  3832  2196 S    1  0.0 28:44.67  lim
     4686 root     15    0      0      0      0 S    0  0.0  3:11.36  nfsd
    19175 user     15    0 11048  1592   864 R    0  0.0  0:00.14  top
        1 root     15    0 10364   600   564 S    0  0.0  0:12.04  init
```

## The ‘sleep’ command

The **sleep** command doesn’t do much – but you can control how many seconds it does it for, and it doesn’t use much CPU or I/O

```
[user@cluster training]$ sleep 10  
[user@cluster training]$
```

## Stop and suspend

If we get bored, change our mind, or think something is wrong we can interrupt jobs.

To stop a job, type '^C' at the command line (that's [Ctrl]+[C] together).

```
[user@cluster training]$ sleep 100  
[user@cluster training]$
```

If you don't want to stop the job, you can suspend it. Type '^Z' (that's [Ctrl]+[Z]).  
Type 'fg' to bring the job back to the foreground.

```
[user@cluster training]$ sleep 100  
[1]+  Stopped                  sleep 100  
[user@cluster training]$ fg
```

# backgrounding

When we have suspended a job (which will never finish). To get it to carry on, we can put it in the ‘background’ using **bg**

```
[user@cluster training]$ sleep 100
[1]+  Stopped                  sleep 100
[user@cluster training]$ bg
[1]+ sleep 100 &
[user@cluster training]$ ps
  PID TTY      TIME CMD
14859 pts/22    00:00:00 bash
24799 pts/22    00:00:00 sleep
25377 pts/22    00:00:00 ps
```

You can put a job in the background deliberately using the ‘&’ character at the end of the command.

```
[user@cluster training]$ sleep 100 &
[1] 787
[user@cluster training]$ ps
  PID TTY      TIME CMD
  787 pts/22    00:00:00 sleep
  804 pts/22    00:00:00 ps
14859 pts/22    00:00:00 bash
```

## Killing processes

If you don't want to wait for it to finish, or think it is broken in some way, you can terminate it using the **kill** command.

Kill has a variety of gentle options to allow the process to exit gracefully. If these fail one – signal **-9**, or **-KILL** will normally remove the process.

```
[user@cluster training]$ sleep 100 &
[1] 787
[user@cluster training]$ ps
  PID TTY      TIME CMD
  787 pts/22    00:00:00 sleep
  804 pts/22    00:00:00 ps
14859 pts/22    00:00:00 bash
[user@cluster training]$ kill -KILL 787
[user@cluster training]$
[1]+  Killed                  sleep 100
[user@cluster training]$
```

## A simple example

Sleep is a good example, but it doesn't produce any output. We want to wrap it up with messages – in unix you use **echo** to do this.

The colon here allows us to put multiple commands on a single line.

```
[user@cluster training]$ echo start; sleep 1; echo finish
start
finish
[user@cluster training]$
```

# Creating a script

Cluster programming makes use of scripts, so we'll turn this list of commands into a script.

Use the nano text editor to enter the following script:

```
[user@cluster training]$ nano script.sh
```

You can run a script by executing **bash <scriptname>** or by making it directly executable with **chmod**. The '**./**' is important – the shell only looks for executables in certain places – the '**PATH**'.

```
[user@cluster training]$ chmod u+x script.sh
[user@cluster training]$ ./script.sh
start
finish
```

```
GNU nano 2.3.1          New Buffer

#!/usr/bin/bash
echo start
sleep 10
echo finish

^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U Uncu
```

## Running the script

Now we are ready to start running our script, or sending it as a cluster job.

```
[user@cluster training]$ ./script.sh > script.out &
[1] 7594
[user@cluster training]$ ps
  PID TTY      TIME CMD
 7594 pts/22    00:00:00 bash
 7595 pts/22    00:00:00 sleep
 7598 pts/22    00:00:00 ps
14859 pts/22    00:00:00 bash
[user@cluster training]$
[1]+  Done                      ./script.sh > script.out
```



CANCER  
RESEARCH  
UK

CAMBRIDGE  
INSTITUTE

# Cluster practical sessions

## Part II – cluster job submission



UNIVERSITY OF  
CAMBRIDGE

# Submitting a job

Now we know enough to run our script on the cluster.

Simply submit the job using **sbatch**.

- the output file is written to a **Lustre file system** directory
- Create directory with username if it doesn't exist – e.g. `mkdir /scratcha/stlab/garret01`
- **/home** is writeable from cluster nodes, but won't perform as well.

```
[user@cluster training]$ sbatch --time=10 --output=/scratcha/stlab/garret01/%N-%j.out script.sh
Submitted batch job 200875
```

- All read and write operations from within jobs running on nodes should use either `/scratchb` or `/scratcha` directories.

## SLURM time formats

Acceptable time formats include "minutes", "minutes:seconds", "hours:minutes:seconds", "days-hours", "days-hours:minutes" and "days-hours:minutes:seconds".

## Look at running jobs

While the job is running, you can see it with **squeue**.

```
[user@cluster training]$ squeue
   JOBID PARTITION     NAME      USER ST       TIME  NODES NODELIST (REASON)
200876  general  script.s    user  R      0:02      1 clust1-node-3
200867  general      bash  sawle01  R  4:01:05      1 clust1-node-2
175393  general MB99.6.v eldrid01  R 9-00:28:46      1 clust1-node-30
175330  general vardict_ eldrid01  R 9-01:40:03      1 clust1-node-1
```

Once it's finished, you can see the output.

```
[user@cluster training]$ squeue
   JOBID PARTITION     NAME      USER ST       TIME  NODES NODELIST (REASON)
200867  general      bash  sawle01  R  4:01:33      1 clust1-node-2
175393  general MB99.6.v eldrid01  R 9-00:29:14      1 clust1-node-30
175330  general vardict_ eldrid01  R 9-01:40:31      1 clust1-node-1
[user@cluster training]$ ls /scratcha/group/user/
clust1-node-3-200877.out
```

## What happened?

The output went into the file as expected:

```
[user@cluster training]$ cat /scratcha/group/user/clust1-node-3-200877.out
start
finish
```

Other information is stored, and available via  
sacct:

```
[user@cluster training]$ sacct -j 200877
  JobID   JobName Partition Account AllocCPUS      State ExitCode
-----  -----
 200877     script.sh    general   group        1  COMPLETED    0:0
 200877.batch      batch           group        1  COMPLETED    0:0
```

```
[user@cluster training]$ sacct -j 200877 --format JobID,MaxRSS,State,AllocCPUS
  JobID      MaxRSS      State AllocCPUS
-----  -----
 200877          2012K  COMPLETED      1
 200877.batch      2012K  COMPLETED      1
```

## An alternative way to submit

You can submit a job directly to SLURM with **srun**. This still requires resources – it's more commonly used as part of an existing job.

```
[user@cluster training]$ srun --time=1-12:30:59 /usr/bin/bash script.sh
start
finish
```

You can also generate an interactive session:

```
[user@cluster training]$ sintr
[user@clust1-node-3 training]$
```

**sintr** is a local implementation of a script written by Pär Andersson (National Supercomputer Centre, Sweden) which will sort out X Forwarding, launch screen on a node, and connect you to the session.

# Killing a job

Just as for processes, but using **scancel**

```
[user@cluster training]$ sbatch --output=/scratcha/group/user/%N-%j.out script.sh
Submitted batch job 200889
[user@cluster training]$ squeue
      JOBID PARTITION      NAME      USER ST       TIME  NODES NODELIST (REASON)
      200889  general  script.s    user   R      0:02      1 clust1-node-3
      200867  general      bash  sawle01   R  4:28:21      1 clust1-node-2
      175393  general MB99.6.v eldrid01   R 9-00:56:02      1 clust1-node-30
      175330  general vardict_ eldrid01   R 9-02:07:19      1 clust1-node-1

[user@cluster training]$ scancel 200889
[user@cluster training]$ squeue
      JOBID PARTITION      NAME      USER ST       TIME  NODES NODELIST (REASON)
      200867  general      bash  sawle01   R  4:28:21      1 clust1-node-2
      175393  general MB99.6.v eldrid01   R 9-00:56:02      1 clust1-node-30
      175330  general vardict_ eldrid01   R 9-02:07:19      1 clust1-node-1
```

**NOTE:** Do not use **skill** it is **NOT** a SLURM command!

## Killing isn't bad...

The scheduler manages the shutdown and still records details of the job.

```
[user@cluster training]$ sacct -j 200889
    JobID      JobName   Partition   Account AllocCPUS       State ExitCode
-----
200889        script.sh     general     group      1 CANCELLED+    0:0
200889.batch      batch           group      1 CANCELLED    0:15
```

# Basic parallelism

Now we're ready to use the cluster at full power!

One way to do this is with a job array. You can create one of these using the

**--array=1-N** syntax in **sbatch**

```
[user@cluster training]$ sbatch --time=10 --array=1-10 --output=/scratcha/group/user/%N-%j.out script.sh
Submitted batch job 200900
[user@cluster training]$ ls /scratcha/group/user
clust1-node-10-200908.out  clust1-node-12-200900.out  clust1-node-4-200902.out
clust1-node-10-200904.out  clust1-node-12-200906.out  clust1-node-11-200909.out
clust1-node-3-200901.out   clust1-node-5-200903.out   clust1-node-7-200905.out
clust1-node-9-200907.out
```

Or using the **srun** with the **-n** or **-N** parameters.

```
[user@cluster training]$ srun --time=10 -n hostname
clust1-node-9.cri.camres.org
...
clust1-node-13.cri.camres.org
[user@cluster training]$
[user@cluster training]$ srun --time=10 -N 3 hostname
clust1-node-19.cri.camres.org
clust1-node-25.cri.camres.org
clust1-node-8.cri.camres.org
[user@cluster training]$
```

**The final example maps specific names to  
their file position in a reference file list.  
The map is written to names-list.out**

```
#!/bin/bash

for i in Matthew Luca Mohammed Lochlan Leighton Keegan Dawid Magnie Zygmunt Zen
do
    srun --time=10 -e /<YOUR SCRATCH DIR/%j-names-list.err -o /<YOUR SCRATCH DIR/names-list.out grep -n $i
    /scratchb/training/refdata/names-list.txt 2>&1 &
done
```



CANCER  
RESEARCH  
UK

CAMBRIDGE  
INSTITUTE

Fin



UNIVERSITY OF  
CAMBRIDGE