CANCER RESEARCH UK | CAMBRIDGE INSTITUTE

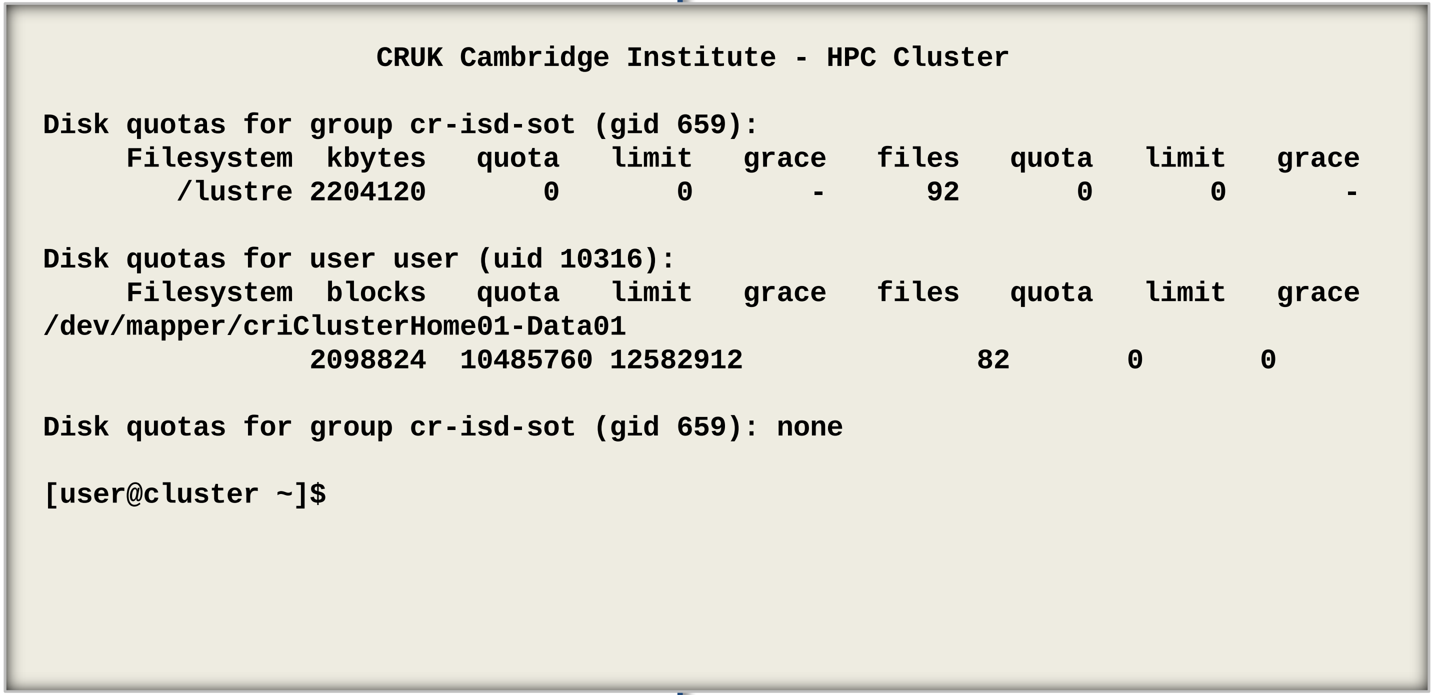
CRUK cluster practical sessions

Part I – processes & scripts

UNIVERSITY OF CAMBRIDGE

login

Log in to the head node, uk-cri-lcst01, using **ssh** and your usual user name & password.



```
                    CRUK Cambridge Institute - HPC Cluster

Disk quotas for group cr-isd-sot (gid 659):
     Filesystem  kbytes   quota   limit   grace   files   quota   limit   grace
        /lustre 2204120       0       0       -      92       0       0       -

Disk quotas for user user (uid 10316):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
/dev/mapper/criClusterHome01-Data01
                2098824  10485760 12582912             82       0       0

Disk quotas for group cr-isd-sot (gid 659): none

[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/
```

processes

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

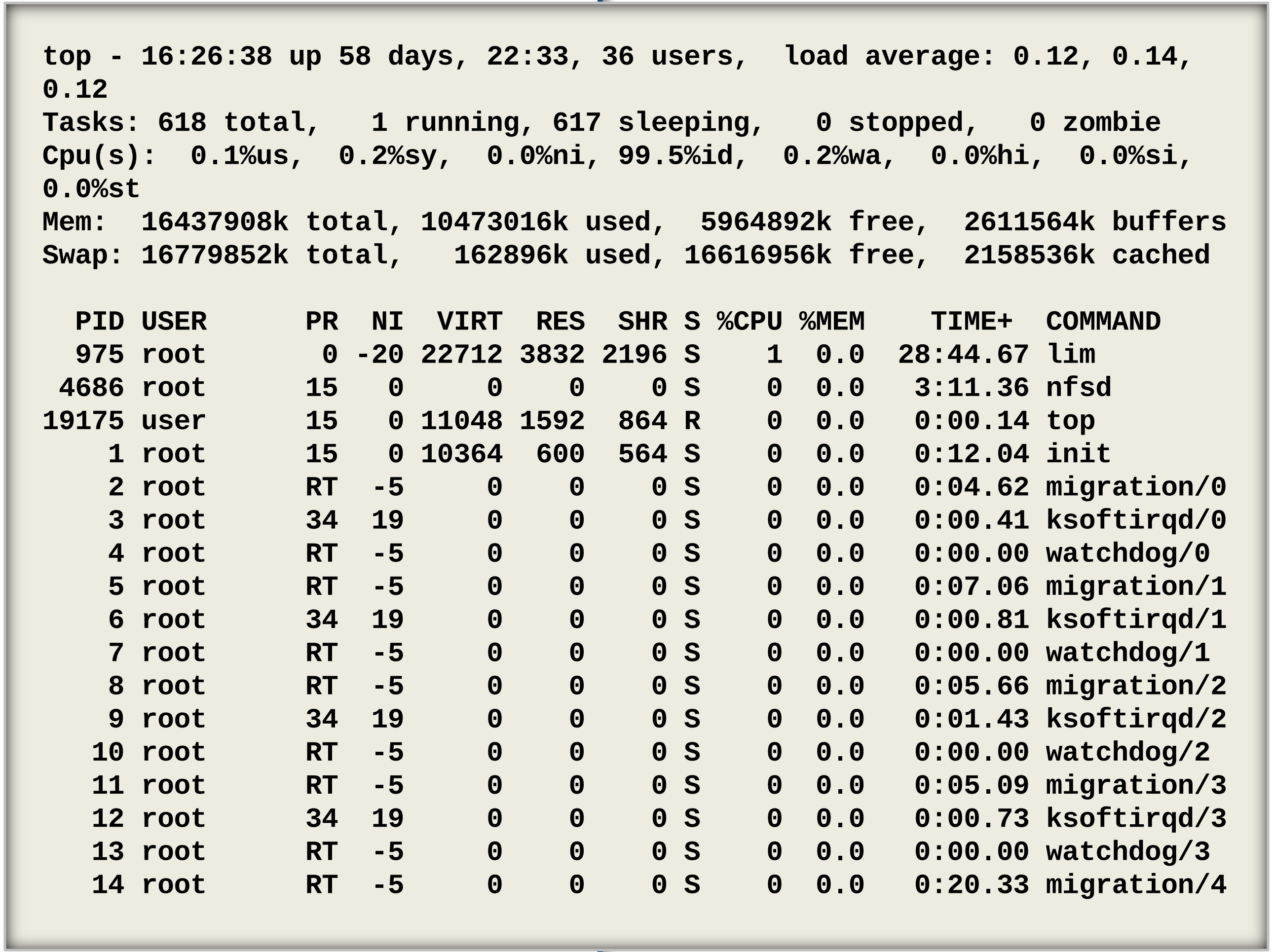
```
[user@cluster training]$ ps
  PID TTY          TIME CMD
14859 pts/22   00:00:00 bash
18511 pts/22   00:00:00 ps
```

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
    2 root      RT  -5     0    0    0 S    0  0.0   0:04.62 migration/0
    3 root      34  19     0    0    0 S    0  0.0   0:00.41 ksoftirqd/0
    4 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/0
    5 root      RT  -5     0    0    0 S    0  0.0   0:07.06 migration/1
    6 root      34  19     0    0    0 S    0  0.0   0:00.81 ksoftirqd/1
    7 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/1
    8 root      RT  -5     0    0    0 S    0  0.0   0:05.66 migration/2
    9 root      34  19     0    0    0 S    0  0.0   0:01.43 ksoftirqd/2
   10 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/2
   11 root      RT  -5     0    0    0 S    0  0.0   0:05.09 migration/3
   12 root      34  19     0    0    0 S    0  0.0   0:00.73 ksoftirqd/3
   13 root      RT  -5     0    0    0 S    0  0.0   0:00.00 watchdog/3
   14 root      RT  -5     0    0    0 S    0  0.0   0:20.33 migration/4
```

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]).

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

backgrounding

We now have a suspended 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
```

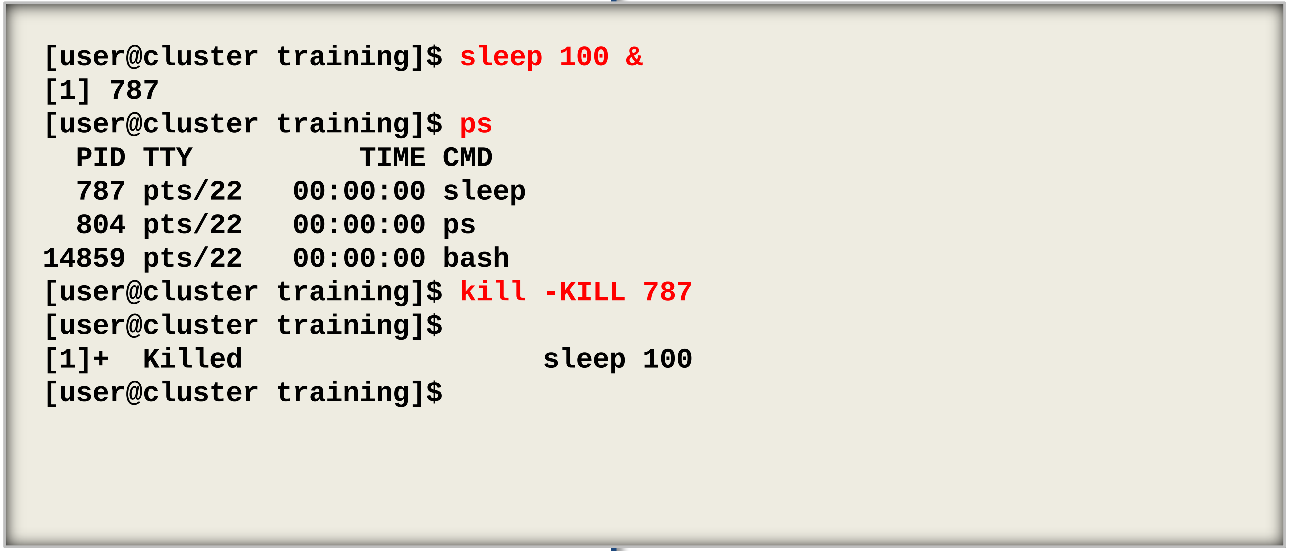
You can put a job in the background deliberately using the '&' character at the end of the command.

```
[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. You only need to know one – signal **-9**, better known by its name **-KILL**

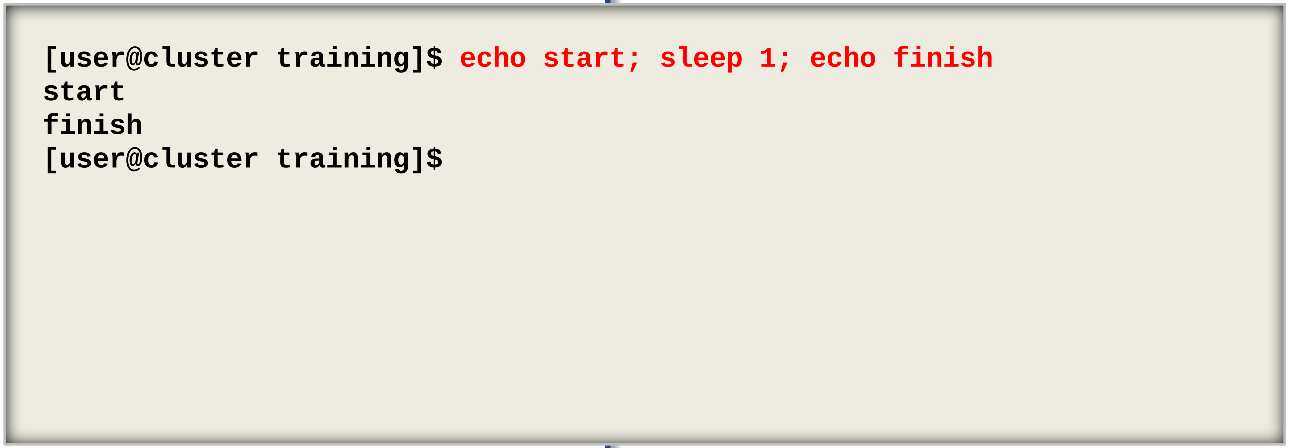


```
[user@cluster training]$ sleep 100 &
[1] 787
[user@cluster training]$ ps
  PID TTY          TIME CMD
  787 pts/22   00:00:00 sleep
  804 pts/22   00:00:00 ps
14859 pts/22   00:00:00 bash
[user@cluster training]$ kill -KILL 787
[user@cluster training]$
[1]+  Killed                  sleep 100
[user@cluster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

You can type directly into a file using **cat** if you know that the end of file character is a '^D'.

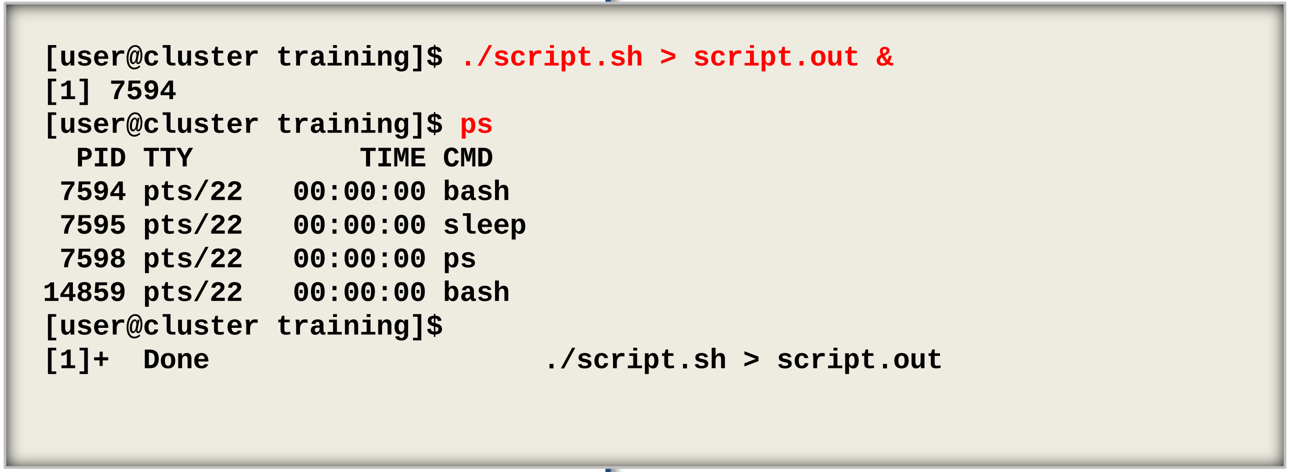
```
[user@cluster training]$ cat > script.sh
echo start
sleep 10
echo finish
[user@cluster training]$
```

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

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

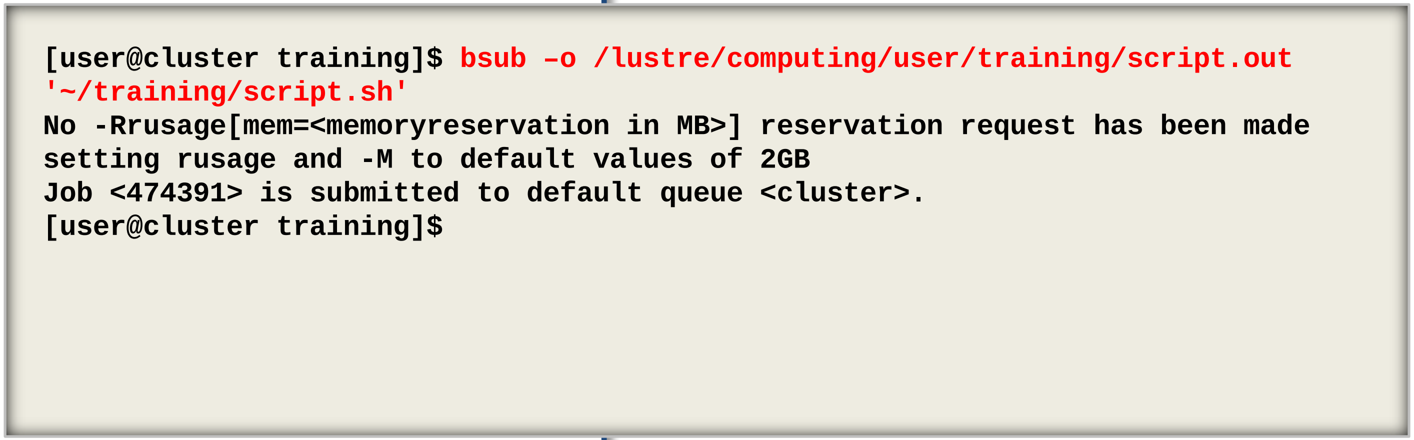
Submitting a job

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

Simply submit the job using **bsub**.

Notice

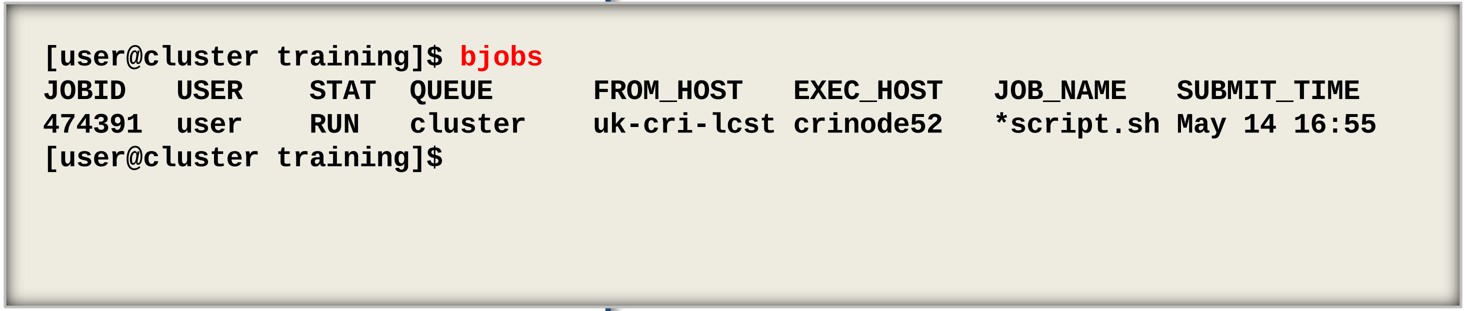
- the output file, in a **/lustre** directory - **/home** isn't writeable from cluster nodes.
- The script is fully located (and recall we made it executable). Each job gets a fresh shell, initially located in **/home**.



```
[user@cluster training]$ bsub -o /lustre/computing/user/training/script.out
'~/training/script.sh'
No -Rrusage[mem=<memoryreservation in MB>] reservation request has been made
setting rusage and -M to default values of 2GB
Job <474391> is submitted to default queue <cluster>.
[user@cluster training]$
```

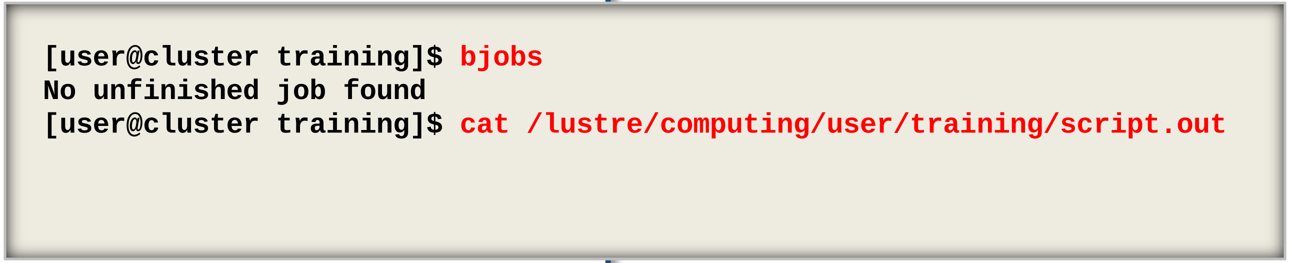
Look at running jobs

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

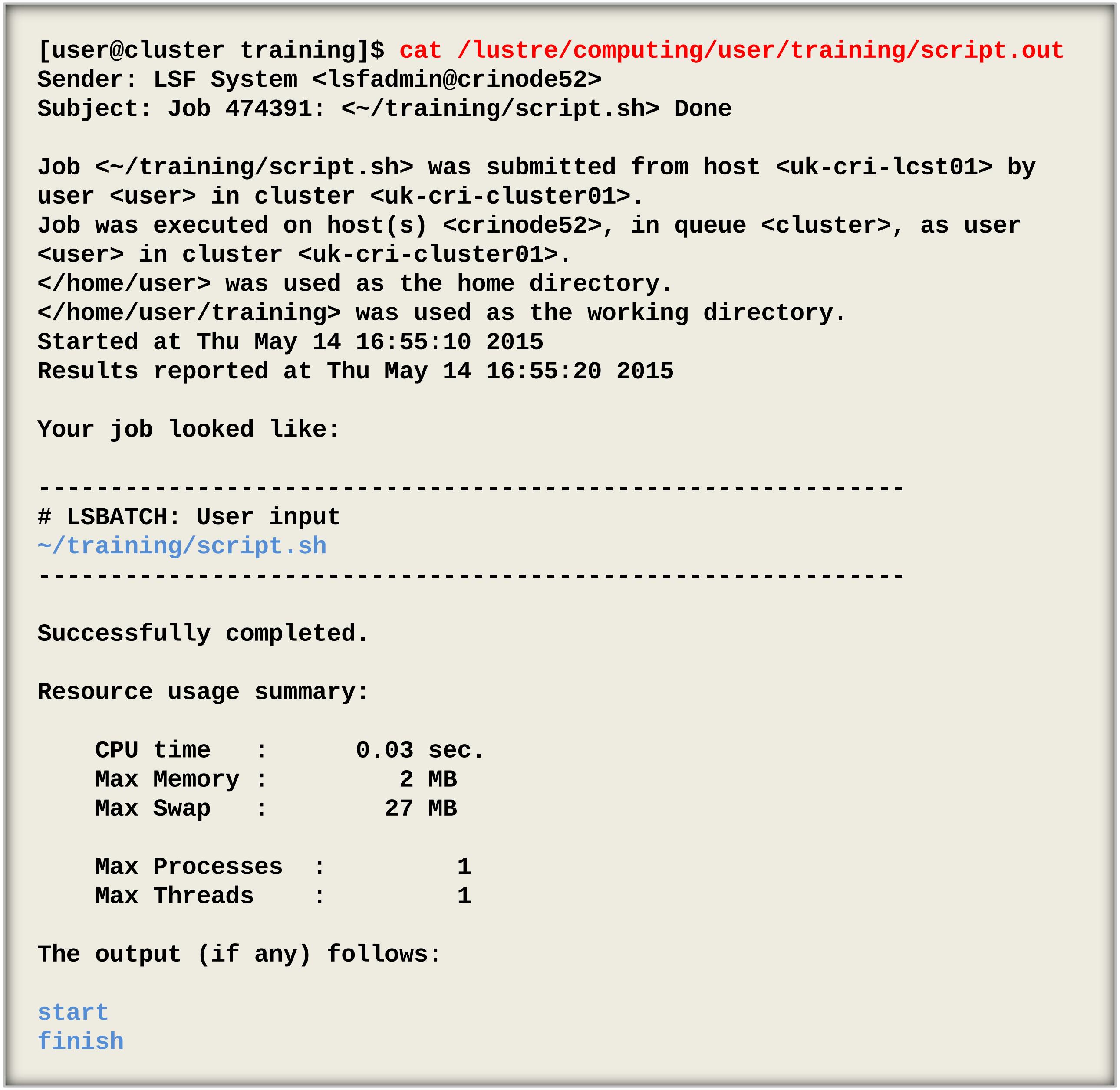


```
[user@cluster training]$ bjobs
JOBID   USER    STAT  QUEUE      FROM_HOST   EXEC_HOST   JOB_NAME   SUBMIT_TIME
474391  user    RUN   cluster    uk-cri-lcst crinode52   *script.sh May 14 16:55
[user@cluster training]$
```

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



```
[user@cluster training]$ bjobs
No unfinished job found
[user@cluster training]$ cat /lustre/computing/user/training/script.out
```



```
[user@cluster training]$ cat /lustre/computing/user/training/script.out
Sender: LSF System <lsfadmin@crinode52>
Subject: Job 474391: <~/training/script.sh> Done

Job <~/training/script.sh> was submitted from host <uk-cri-lcst01> by
user <user> in cluster <uk-cri-cluster01>.
Job was executed on host(s) <crinode52>, in queue <cluster>, as user
<user> in cluster <uk-cri-cluster01>.
</home/user> was used as the home directory.
</home/user/training> was used as the working directory.
Started at Thu May 14 16:55:10 2015
Results reported at Thu May 14 16:55:20 2015

Your job looked like:

------------------------------------------------------------
# LSBATCH: User input
~/training/script.sh
------------------------------------------------------------

Successfully completed.

Resource usage summary:

    CPU time   :      0.03 sec.
    Max Memory :         2 MB
    Max Swap   :        27 MB

    Max Processes  :         1
    Max Threads    :         1

The output (if any) follows:

start
finish
```

An alternative way to submit

Read the commands from **stdin** using ‘<‘

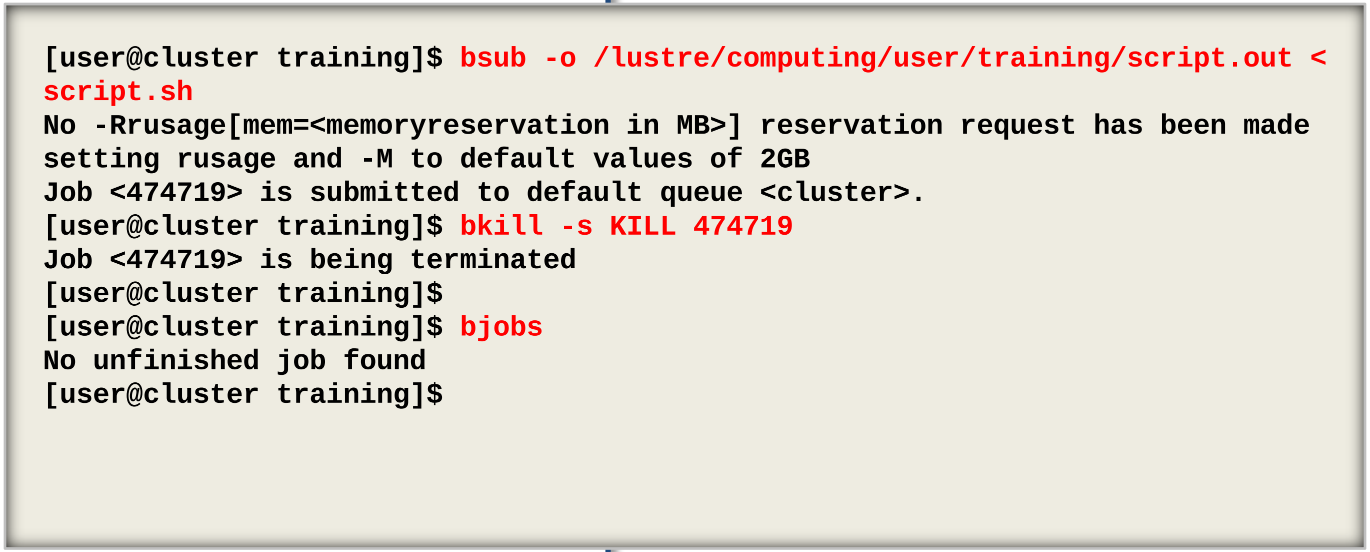
```
[user@cluster training]$ bsub -o /lustre/computing/user/training/script.out <
script.sh
```

How is this different?

```
[user@cluster training]$ cat /lustre/computing/user/training/script.out
Sender: LSF System <lsfadmin@crinode98>
Subject: Job 474618: <echo start;sleep 10;echo finish> Done
[…]
Your job looked like:

------------------------------------------------------------
# LSBATCH: User input
echo start
sleep 10
echo finish

------------------------------------------------------------
[…]
The output (if any) follows:

start
finish
```

Killing a job

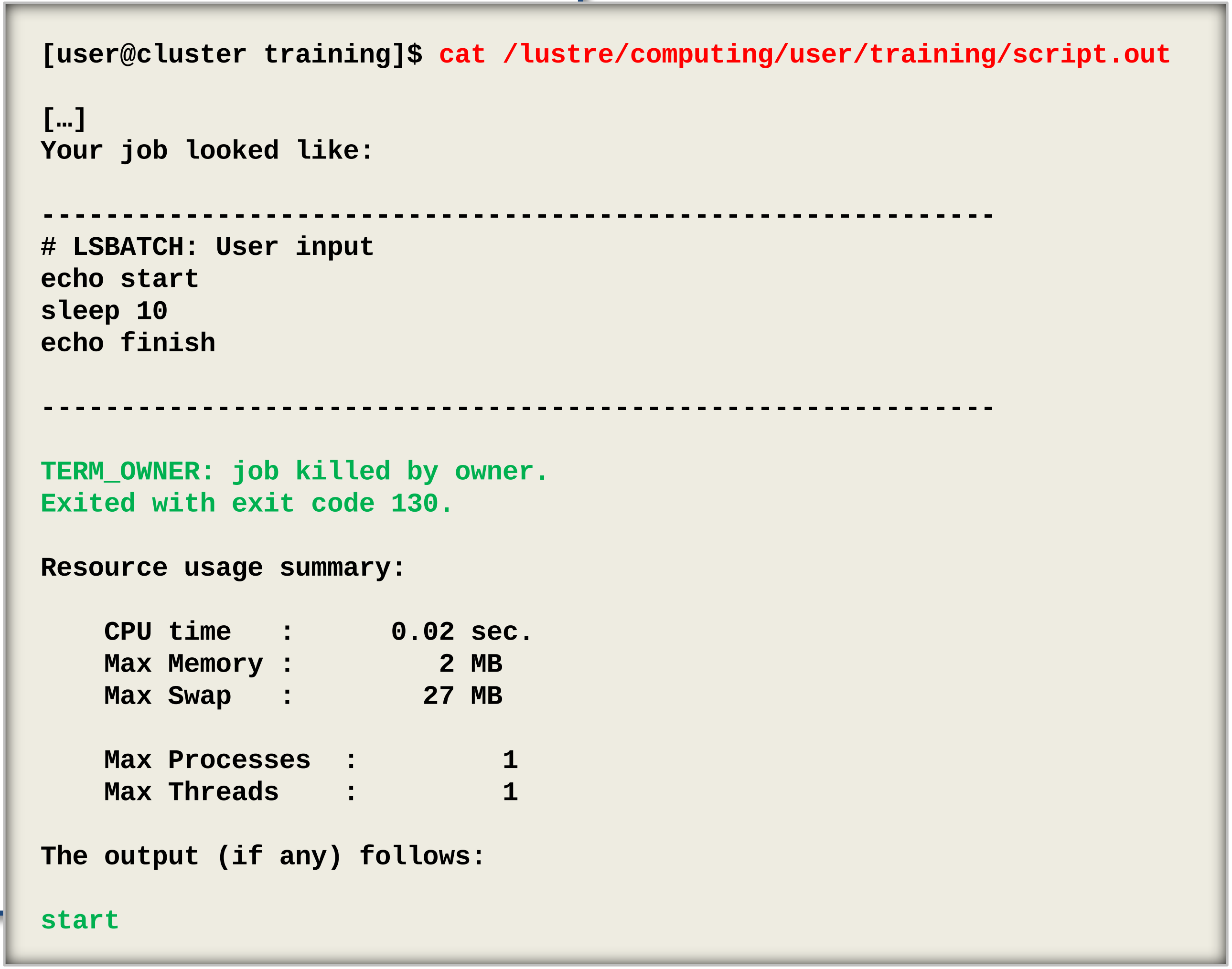
Just as for processes, but using **bkill**



```
[user@cluster training]$ bsub -o /lustre/computing/user/training/script.out <
script.sh
No -Rrusage[mem=<memoryreservation in MB>] reservation request has been made
setting rusage and -M to default values of 2GB
Job <474719> is submitted to default queue <cluster>.
[user@cluster training]$ bkill -s KILL 474719
Job <474719> is being terminated
[user@cluster training]$
[user@cluster training]$ bjobs
No unfinished job found
[user@cluster training]$
```

Killing isn’t bad…

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

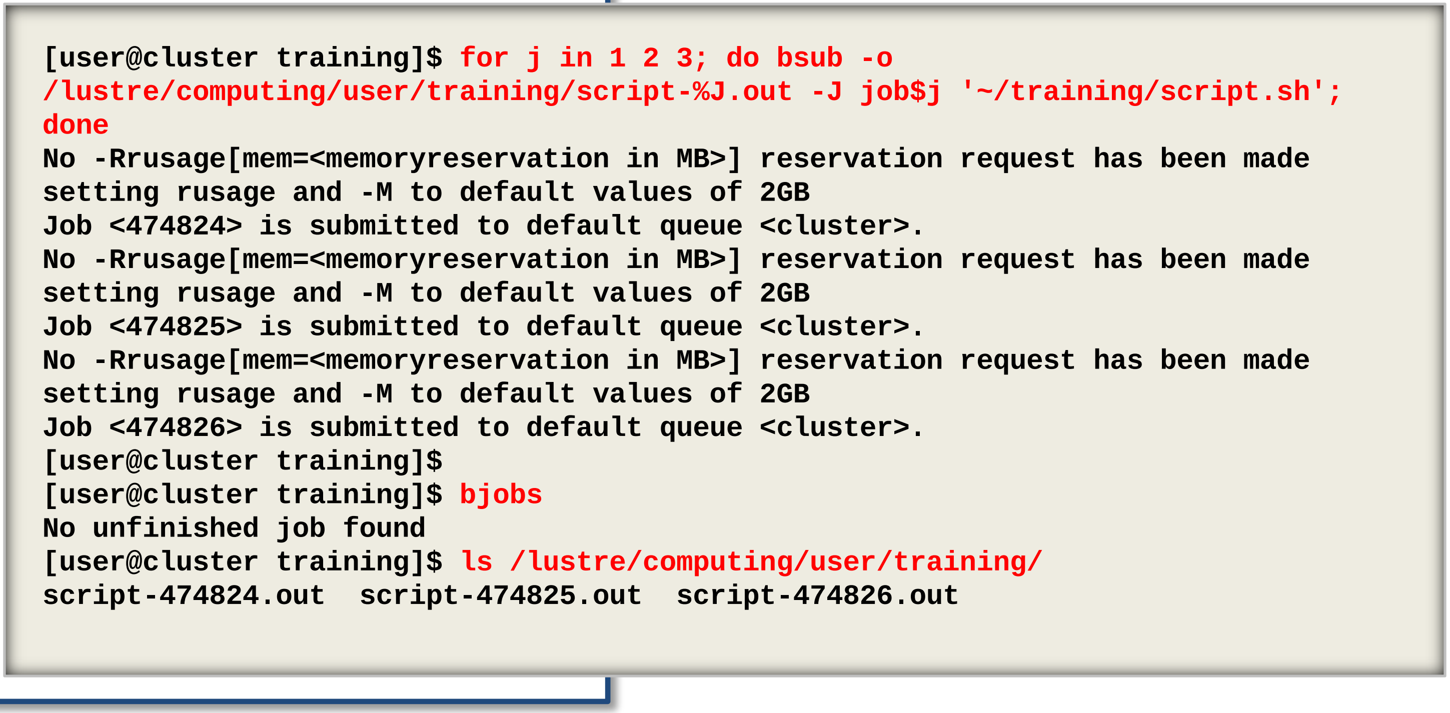


```
[user@cluster training]$ cat /lustre/computing/user/training/script.out

[…]
Your job looked like:

------------------------------------------------------------
# LSBATCH: User input
echo start
sleep 10
echo finish

------------------------------------------------------------

TERM_OWNER: job killed by owner.
Exited with exit code 130.

Resource usage summary:

    CPU time   :      0.02 sec.
    Max Memory :         2 MB
    Max Swap   :        27 MB

    Max Processes  :         1
    Max Threads    :         1

The output (if any) follows:

start
```

Basic parallelism

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

The simplest way to do this is simply running multiple jobs at once. Here we use the **bash 'for…; do …; done'** construct, the **bsub –J** option and the **%J** environment variable.



```
[user@cluster training]$ for j in 1 2 3; do bsub -o
/lustre/computing/user/training/script-%J.out -J job$j '~/training/script.sh';
done
No -Rrusage[mem=<memoryreservation in MB>] reservation request has been made
setting rusage and -M to default values of 2GB
Job <474824> is submitted to default queue <cluster>.
No -Rrusage[mem=<memoryreservation in MB>] reservation request has been made
setting rusage and -M to default values of 2GB
Job <474825> is submitted to default queue <cluster>.
No -Rrusage[mem=<memoryreservation in MB>] reservation request has been made
setting rusage and -M to default values of 2GB
Job <474826> is submitted to default queue <cluster>.
[user@cluster training]$
[user@cluster training]$ bjobs
No unfinished job found
[user@cluster training]$ ls /lustre/computing/user/training/
script-474824.out  script-474825.out  script-474826.out
```

Fin