Abel tutorial for beginners

Katerina Michalickova

November 5th , 2014

The Research Computing Service Group

http://www.uio.no/hpc

Contents

1. Resources

This tutorial is intended as a start for people who would like to use Abel. Links below point to a complete user guide and information about the system.

Abel page: http://www.uio.no/english/services/it/research/hpc/abel/

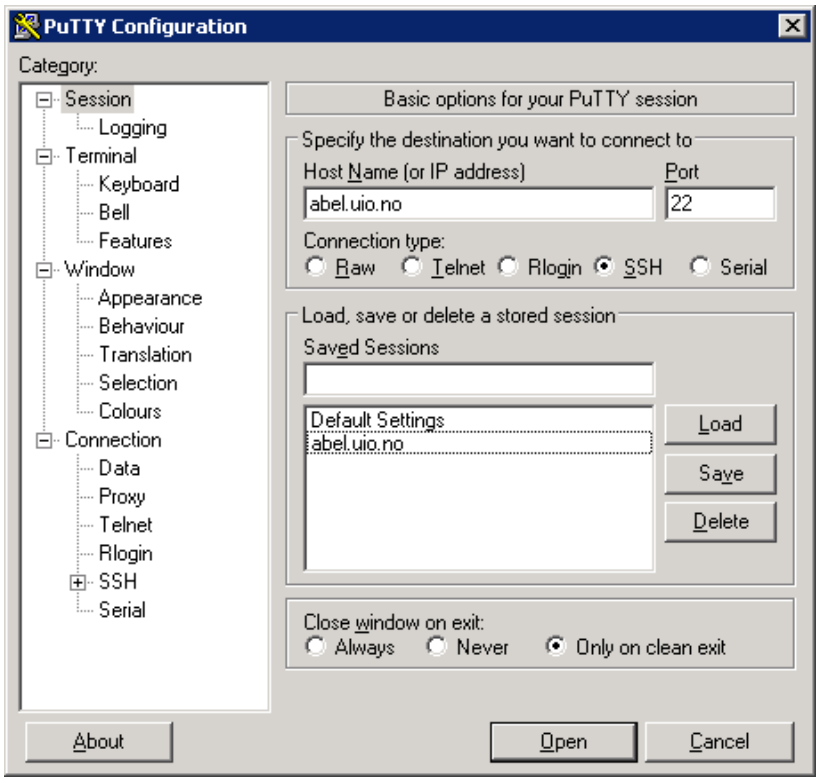
Abel user guide: http://www.uio.no/english/services/it/research/hpc/abel/help/user-guide/

2. Log into Abel

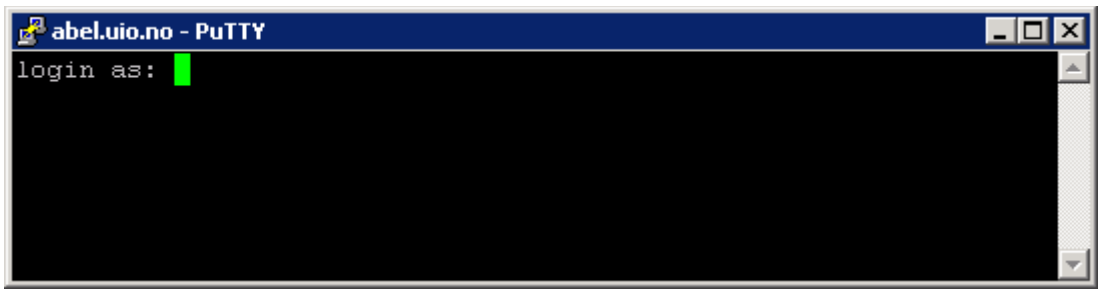
2.1 From a Windows machine

Download putty, a windows ssh client, from http://www.putty.org/

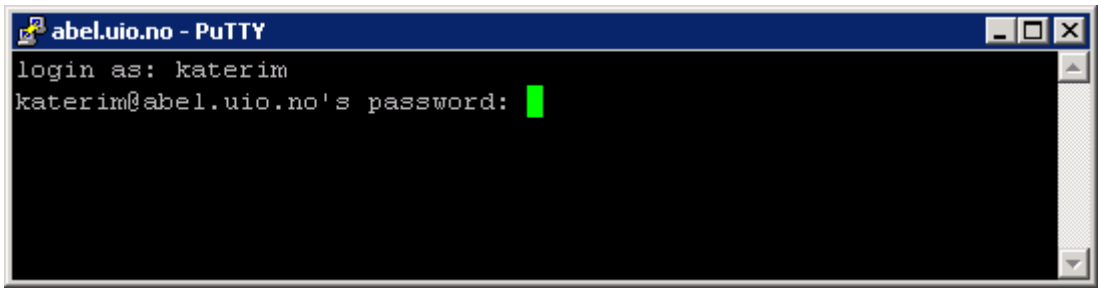
Start putty and type "abel.uio.no" into the host mane dialog box:



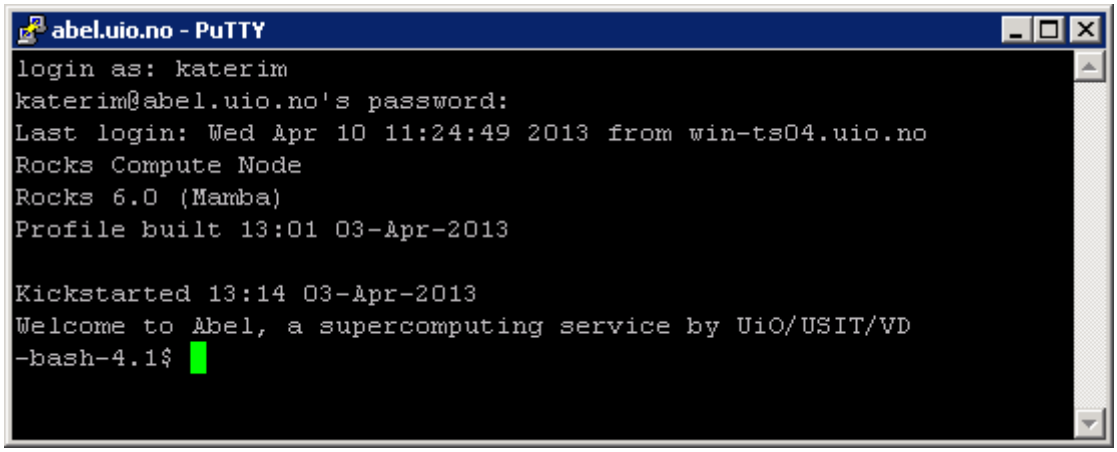
Press Open button:



Type in your UiO user name (and press Enter):

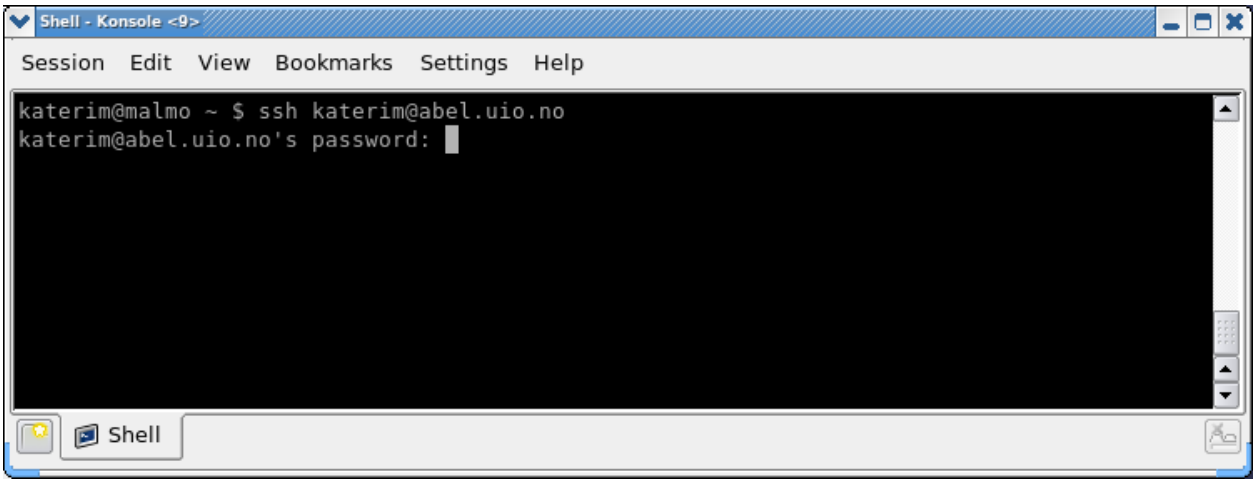


Type in your UiO password (and press Enter):

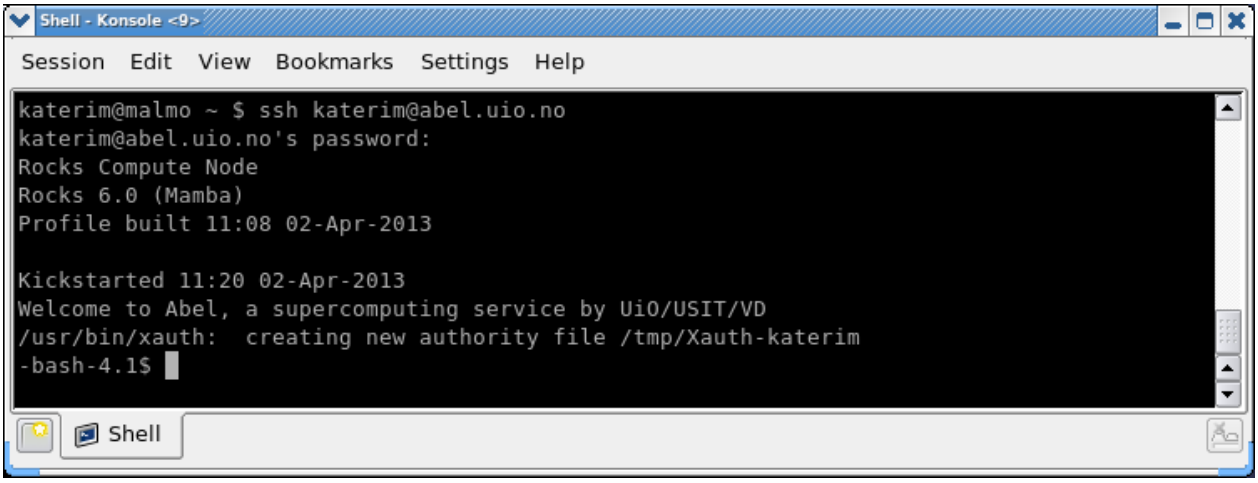


2.2 From a Linux machine

Open a terminal and type "ssh your_UiO_user_name@abel.uio.no":



Type in your UiO password:



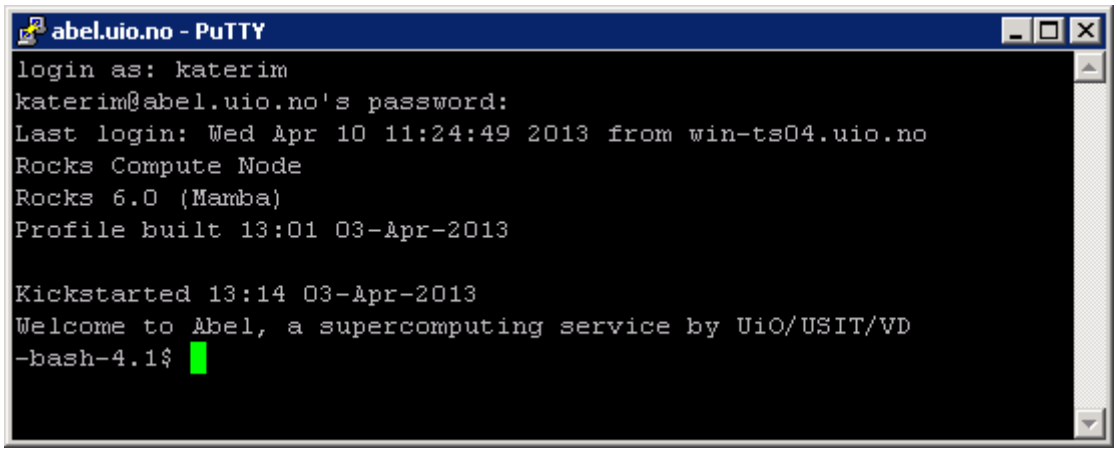
2.3 From a Mac

Open terminal app and type type "ssh your_UiO_user_name@abel.uio.no". Proceed in the same way as on the linux system.

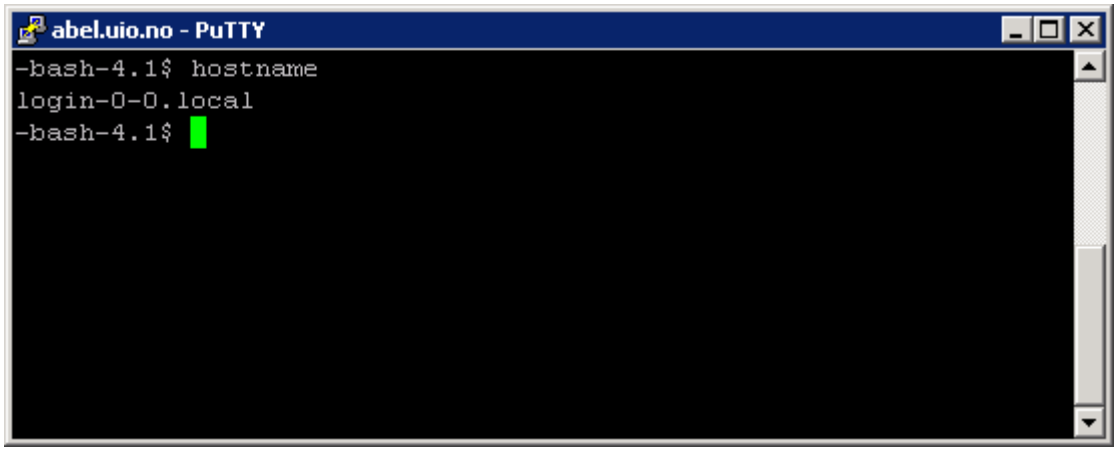
3. Home area on Abel

3.1 Home directory

You are logged in and it looks like this:



Which machine are you on? Type "hostname":



You are on one of the two so-called login nodes of the Abel computer cluster. Abel consists of 600+ computers and only two are available for login. If you are interested, you can read about Abel technical specifications here:
http://www.uio.no/english/services/it/research/hpc/abel/more/

When you login into Abel, you always find yourself in your "home directory". It is a place in the directory tree where you keep your files. Regardless what machine you are on, this directory is available to you. Type "pwd" (print working directory) to see the full path:

4. Unix command line environment

4.1 Tutorials

Users need some knowledge of the command line environment and shell scripting to efficiently work on Abel. Below is a link to the RCS command line tutorial for those who missed it.

http://www.uio.no/english/services/it/research/hpc/courses/linux-command-line/unix_tutorial_nov2014.pdf

For more examples, I recommend to review online tutorials, find one that suits you and read through:

http://www.ee.surrey.ac.uk/Teaching/Unix/

http://people.ischool.berkeley.edu/~kevin/unix-tutorial/toc.html

http://www2.ocean.washington.edu/unix.tutorial.html

http://www.youtube.com/watch?v=v4YpWACs6Ts

4.2 Shell scripting exercise

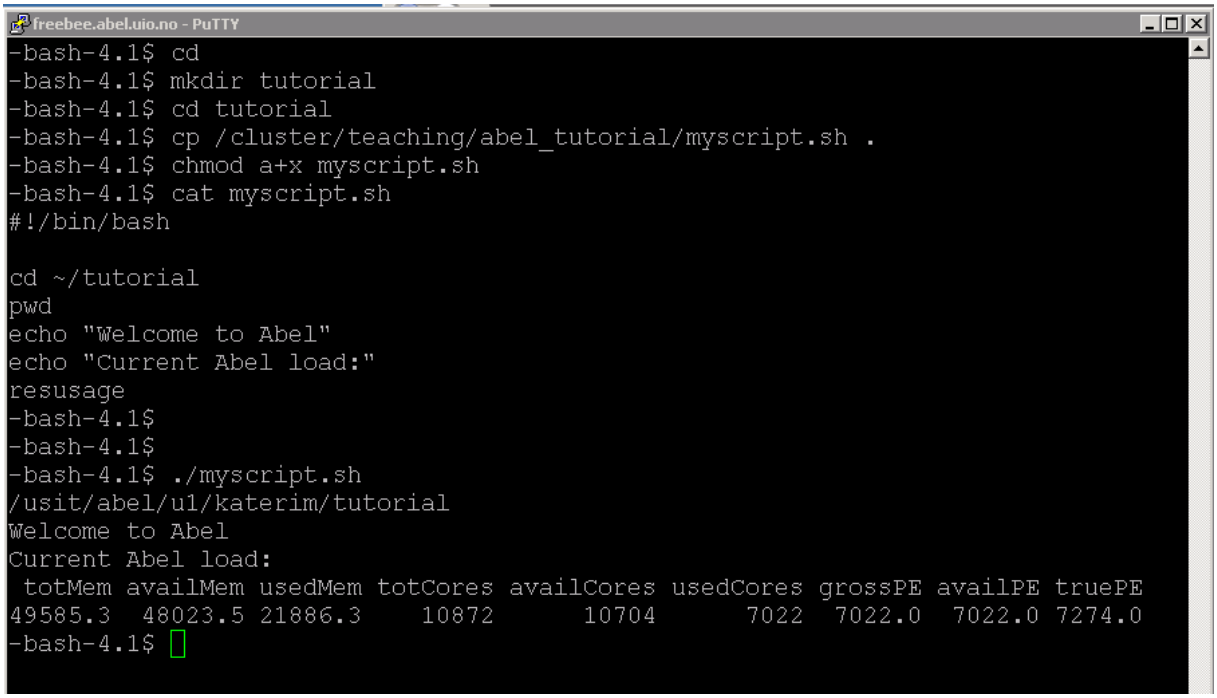
In order to run jobs on Abel, you will have to write job scripts. Job scripts are essentially shell scripts and this section contains a brief overview.

Shell script is a series of Unix commands written in a text file. Each command is on a separate line. Upon execution, all commands are executed sequentially. Shell scripts can be used for programming as they support flow control statements and variables.

Make sure that you are in your home directory. You can type "cd ~" (or "cd" for short). Make a directory named "tutorial" ("mkdir tutorial") to keep all tutorial materials together. Change working directory to "tutorial" ("cd tutorial").

Copy an example script called myscript.sh from the common area to your tutorial directory ("cp /cluster/teaching/abel_tutorial/myscript.sh ."). Change permissions so the script can be executed; the command "chmod a+x myscript.txt" makes the file executable for everybody.

Examine the script. Note the first line of the script “#! /bin/bash” (#! is called a “hashbang”) tells the program loader to use an interpreter /bin/bash. Run the script by typing “./myscript.sh” and examine the output of the script. The script produced a current directory path (output of “pwd”), welcoming message and statistics of Abel load (output of “resusage”).

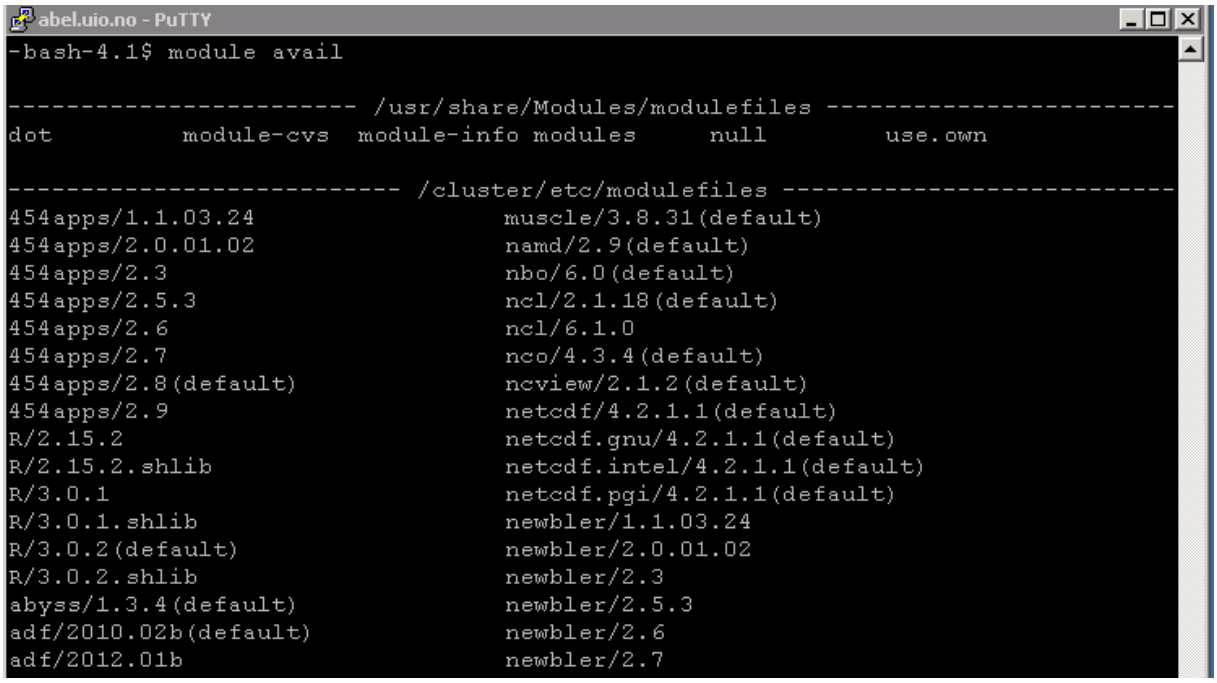


Read more about shell scripting - http://www.uio.no/english/services/it/research/hpc/courses/shell-scripting/20140325_intro_shell_scripting.pdf

5. Prepare and run your first job

5.1 Software and modules

Type "module avail" to see all software available on Abel. If you find a program that you want to use, type "module load module_name". This sets up the $PATH variable (and more) so the software is accessible to you. If you do not find the software that you are looking for, you can always place the executable in your home directory and use it from there.



5.2 Job script

To submit a job (i.e. run a software), users have to communicate with the job manager on Abel. Job manager is a software that schedules jobs and oversees their execution on cluster compute nodes. On Abel, we use Simple Linux Utility for Resource Management – SLURM https://computing.llnl.gov/linux/slurm/.

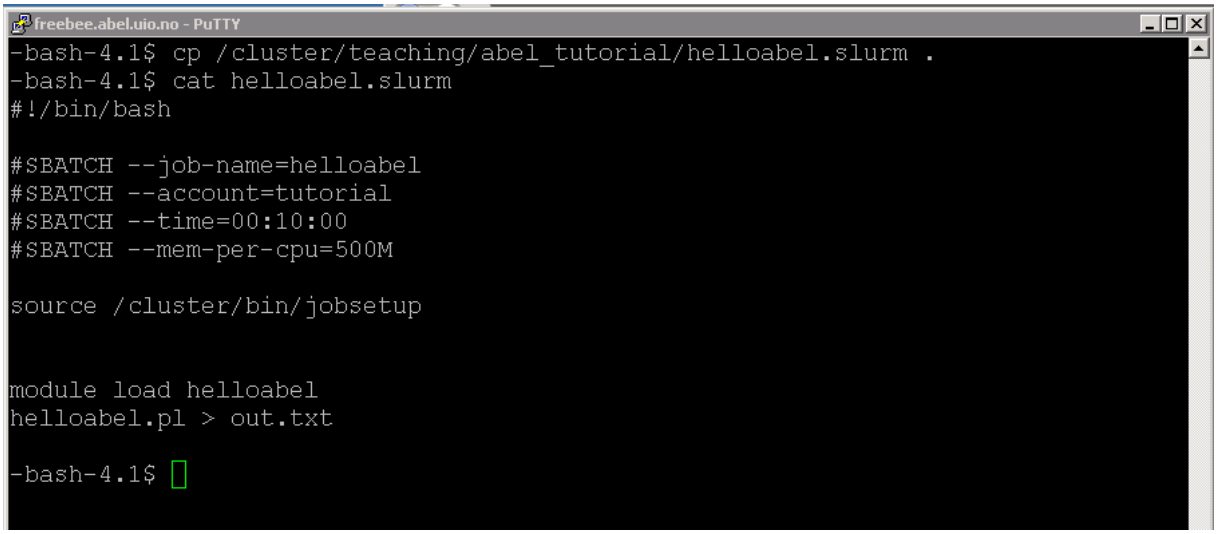
Users communicate with the job manager using job scripts. Job scripts are shell scripts that contain the command that user wants to execute plus several job parameters. Job parameters inform the job manager on Abel about resources needed by the job. Without this information, the job cannot be scheduled.

Abel documentation http://www.uio.no/english/services/it/research/hpc/abel/help/user-guide/ contains a detailed description of job parameters and examples of job scripts.

Note of caution: Abel users are not allowed to use command line on login nodes to run jobs, this might render the login nodes irresponsive to other user login requests.

If you need interactive login, use qlogin command (http://www.uio.no/english/services/it/research/hpc/abel/help/user-guide/interactive-logins.html). Qlogin reserves time on one of the cluster nodes, once the allocation is received, users can work directly on the command line of the node.

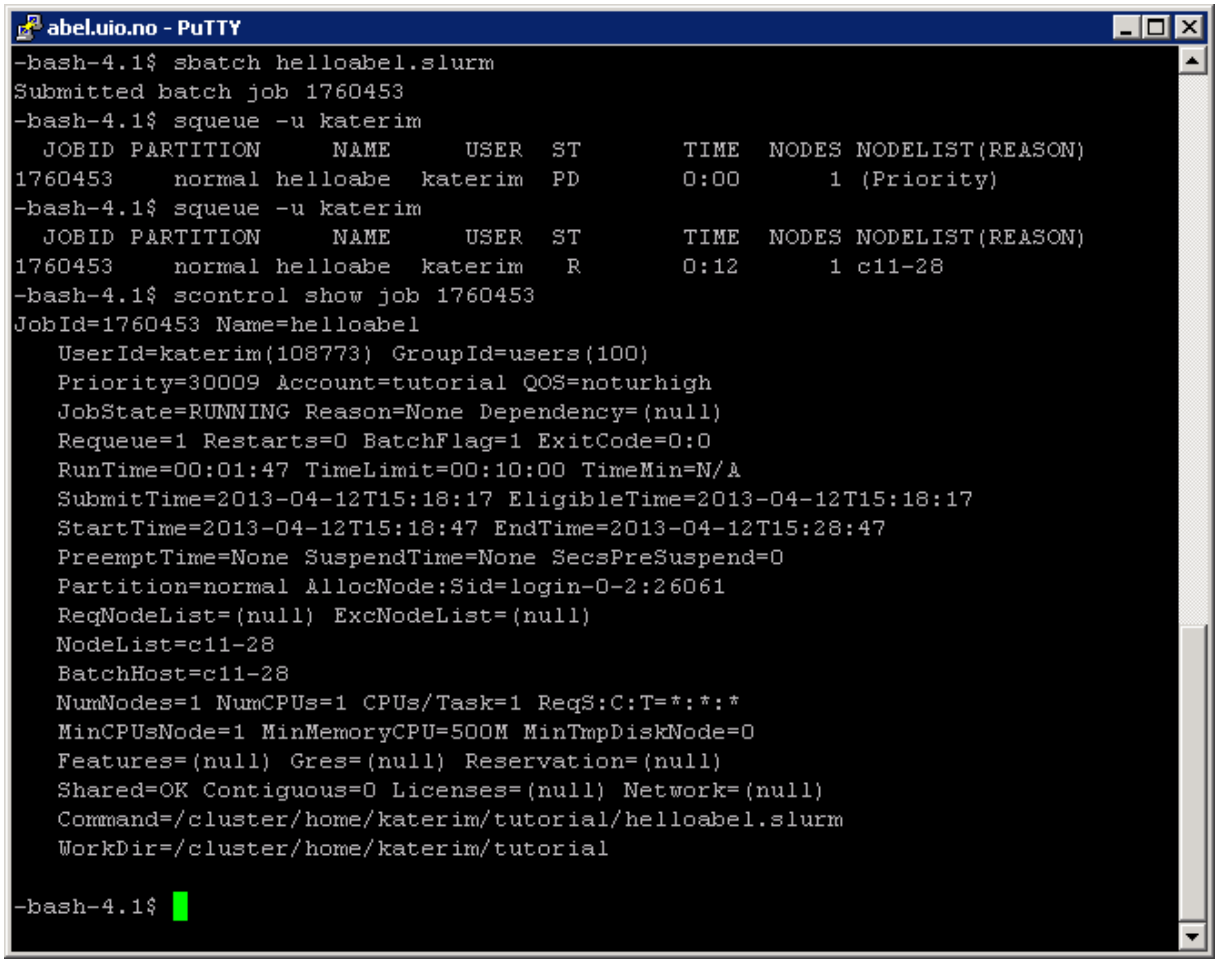
Copy a simple job script to your tutorial directory ("cp /cluster/teaching/abel_tutorial/helloabel.slurm .") and look at the contents. This is an example of a minimal job script, i.e. script that contains minimum information for the SLURM controller to schedule the job.



- The script contains #SBATCH keywords that communicate to the job manager various settings for the job. The minimal set of these involve an account, time and memory.
  - The parameter **"--account**" is an administrative group of Abel users that a user has a right to use. Each account has a number of resources (cpus and cpu hours) assigned to it. Type "projects" on the command line to see which account(s) you have access to.
  - The parameter **"--time**" is the real time you expect your program to run.
  - The parameter **"--mem-per-cpu**" is the RAM requirement for your job. The job manager has to know this so it can schedule jobs in the most efficient manner. If you exceed time or memory specified, your job is cancelled.
  - The parameter "**--job-name**" is not compulsory but it is useful if you want to follow your job easily. We recommend to use it.
- The "**source /cluster/bin/jobsetup**" command sets up environment variables for your job. This line must always be present.
- The "**module load helloabel**" command gives you access to the helloabel executable.
- The "**./helloabel.pl > out.txt**" command is the core of the script, it is the program that is executed on Abel. In this case, the output is captured into the file called out.txt.

5.3 Submit your job and follow the progress

To execute the helloabel.pl script on Abel, submit the job using "sbatch helloabel.slurm". The command submits the job into the job queue and returns a job identifier (jobid). Jobs can be followed as they are waiting in the queue and as they are executed. Type "squeue" and the whole job queue is displayed. To limit the output to your jobs only, type "squeue –u user_name". Typing "scontrol show job jobid" produces detailed output including job parameters and the job script path. If in need to cancel a job, type "scancel jobid".



- In the example above, the job id is 1760453.
- There are two instances of the squeue command output. In the first case, the status (ST) of the job is PD or pending as the job is waiting for available resource. In the second case, the job has been running on the node c11-28 for the last 12 sec.
- The example above also shows the output of the scontrol command that contains detailed information about the job. Note the last two rows that show the path to the working directory and to the job script. The RunTime and TimeLimit items (in the 5th row) tell you how much total time you have and how long the job has been running. If you are running out, let us know, we can extend the time.

5.4 Job output

List the contents of your tutorial directory. The file **out.txt** contains the output of our program. The file **slurm-jobid.out** contains messages from the queuing system. It also contains time, place and other parameters of cluster execution. If there is a problem, error messages are logged in this file.

Exercise: Learn how to run a script that is in your tutorial directory.

- Copy a perl script called hellolocal.pl from /cluster/teaching/abel_tutorial to your tutorial directory.

- Prepare a job script to run hellolocal.pl. First, copy ~/tutorial/helloabel.slurm into ~/tutorial/hellolocal.slurm. Second, make changes to hellolocal.slurm so it executes "hellolocal.pl" from your directory (hint use ./ to tell the script that the program is in the current directory).

- Finally, submit the job and check the output files.

Note: Executing jobs in you home area does not require using module load command, instead you have to provide path to the executable.

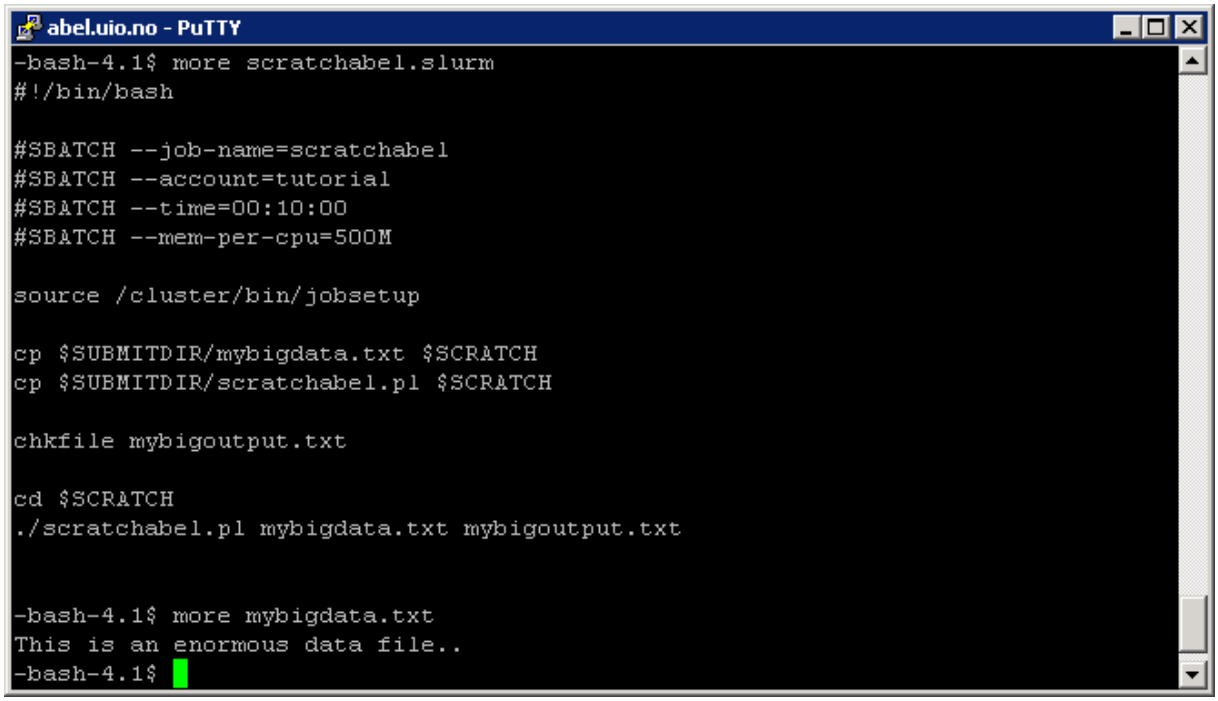
6. Prepare a job using scratch area

If your job reads and writes lots of data (and especially if this happens often and in small chunks), we recommend to stage the job execution onto a faster file system that is available on Abel. This partition is more suitable for I/O demanding jobs than your home (and slower) directory. On Abel, this partition is called /work and every job receives a temporal (scratch) space on this partition (specifically in /work/jobs/jobid.d subdirectory). This directory is removed when the job is finished. It cannot be used for data storage.

The path to the scratch directory is contained in the SCRATCH environment variable that is instantiated when you submit you job. Every job creates several environment variables, e.g. SUBMITDIR that contain path to the directory containing the job script.

6.1 Job script

Copy the script "scratchabel.slurm" from the common area. In addition copy the files scratchabel.pl and mybigdata.txt. s

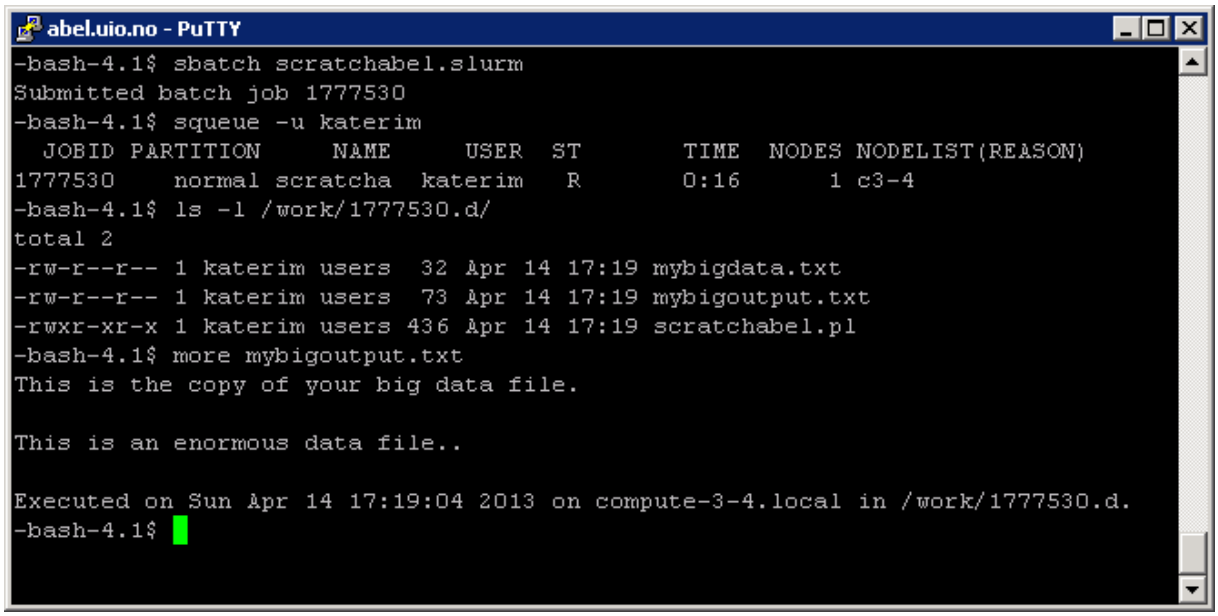


- The script takes advantage of environment variables SUBMITDIR (directory that you are submitting from) and SCRATCH (temporal directory assigned to your job). They are accessible to your job script and your executable during job execution.
- To use SCRATCH, you have to copy your executable and your input data files (mybigdata.txt and scratchabel.pl) there. This is accomplished using "cp $SUBMITDIR/mybigdata.txt $SCRATCH" and "cp $SUBMITDIR/scratchabel.pl $SCRATCH" lines.
- Register output file(s) that you want copied back from the scratch area using the chkfile command. This is the recommended way of doing so since this command is

always executed at the end of your job regardless if the job finished successfully or not. Alternatively, you could simply copy the result file at the end of your script (e.g. "cp $SCRATCH/mybigoutput.txt $SUBMITDIR"). This is fine if the job runs successfully. In case of a failed run, the last copy statement is not executed as opposed to chkfile that always runs.

- Before the job is executed, you have to change working directory to scratch area using "cd $SCRATCH".

Submit the script using "sbatch scratchabel.slurm", note job id and check the /work/jobid.d subdir where you can see your script and data files. When done, check the output:



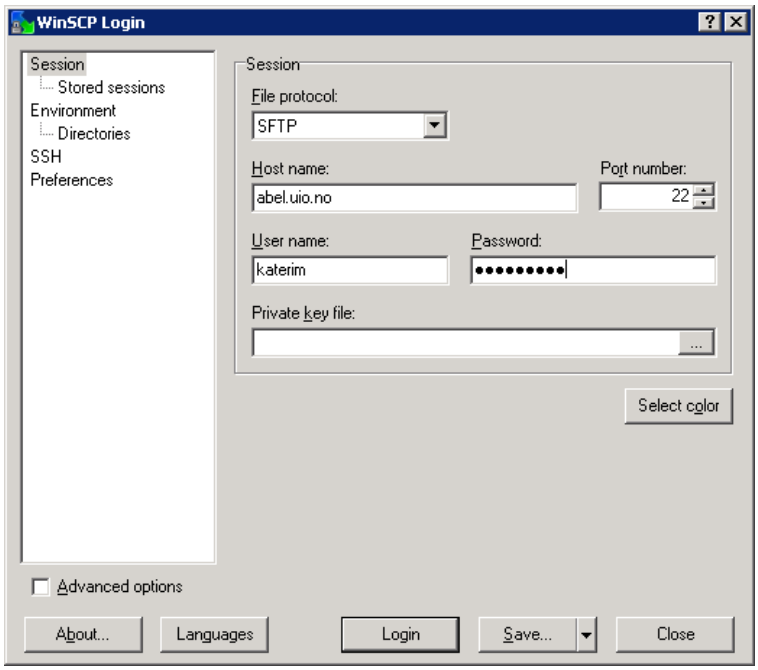
The script scratchabel.pl is only a demonstration of a principle, it simply echoes back your input data and adds information about time, node and current directory.

6.2 Upload you own file

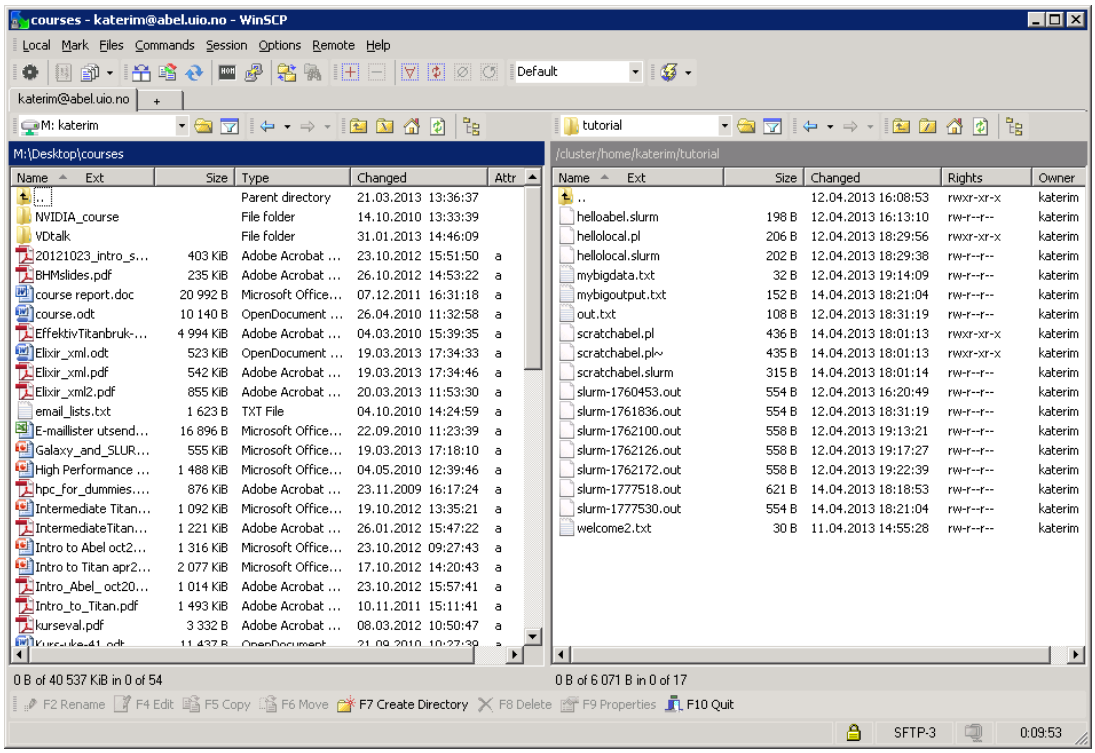
This section explains how to upload data from your local computer to Abel.

6.2.1 On Windows

Download Winscp from http://winscp.net/eng/download.php. Install and run. Type abel.uio.no into the "Host name" dialog box. Below, specify user name and password:



Press “Login” and manipulate your files using “drag and drop”:



6.2.2 On Linux

Open a terminal and use scp (secure copy) command. Type “scp myfile.txt your_user_name@abel.uio.no:~/tutorial” to copy a file from a current directory to abel tutorial directory. For large file the rsync command is preffered since, in case of interruption, it is able to resume where it left off; “rsync –z myfile.txt your_user_name@abel.uio.no:~/tutorial” uploads your file to the tutorial directory on Abel.

6.2.3 On the Mac

Open a terminal and use scp the same way as on Linux.

7. Parallel jobs

The strength of cluster computing lies in parallel processing i.e. executing many instances of the same executable at the same time. This is often needed when you have many input datasets or when you run simulations with different input parameters. It might often be possible to split a large input file into chunks and parallelize you job.

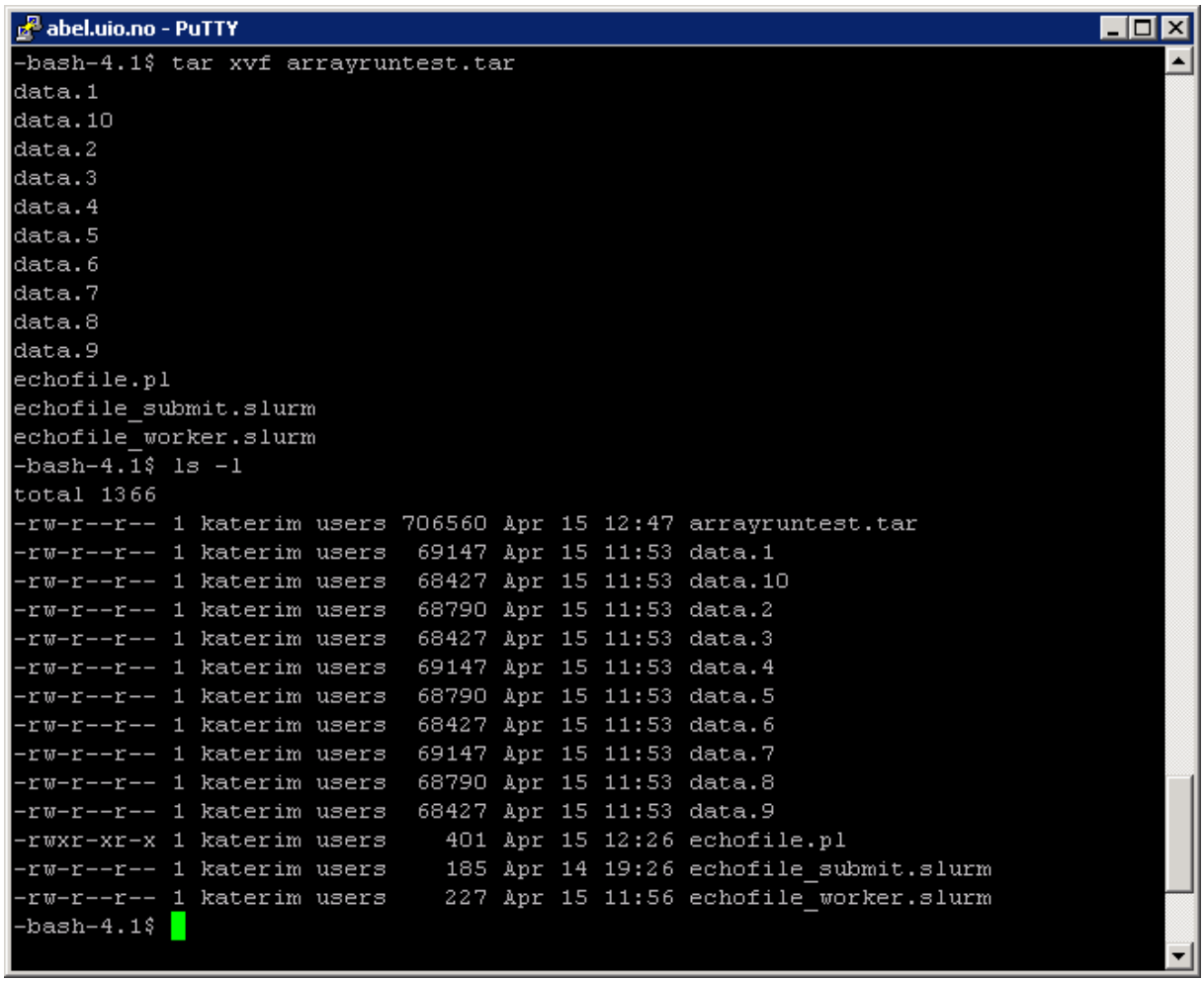
Abel offers a utility called "arrayrun" that can start a number of parallel jobs using the same job script.

Since the parallel execution is driven by the same job script from the same directory, the main consideration during setup is to manage input and output files in such a way that the correct files are read and the output does not get overwritten. The recommended way is to use the TASK_ID environment variable in file names. This variable has unique value for each run; if you run your program 10 times using arrayrun command, the TASK_ID ranges from 1 to 10.

7.1 Arrayrun

In your tutorial directory, make a subdirectory called arrayruntest and cd into it ("cd ~/tutorial; mkdir arrayruntest; cd arrayruntest"). Copy the file arrrayruntest.tar from /cluster/teaching/abel_tutorial to your arrayruntest directory. Unpack all files by "tar xvf arrayrun.tar".

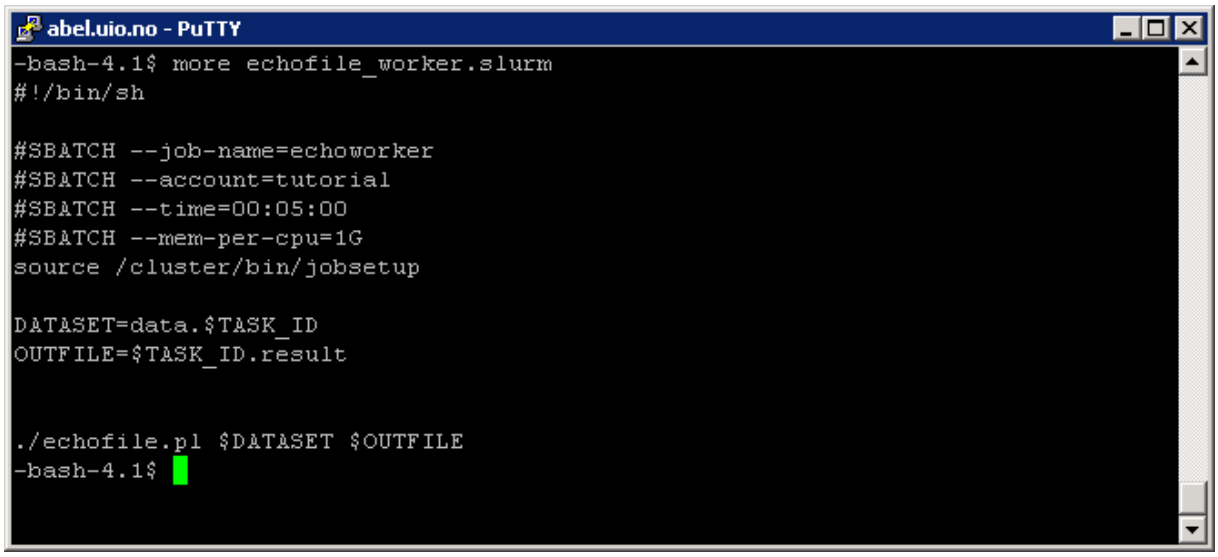
The archive (arrayruntest.tar) unpacks into datafiles, two job scripts and one executable. The perl executable, echofile.pl, is a very simple placeholder for your own program, it copies an input file to output and adds some info about time and place. There are ten data files called data.* and two job scripts – echofile_submit.slurm and echofile_worker.slurm.



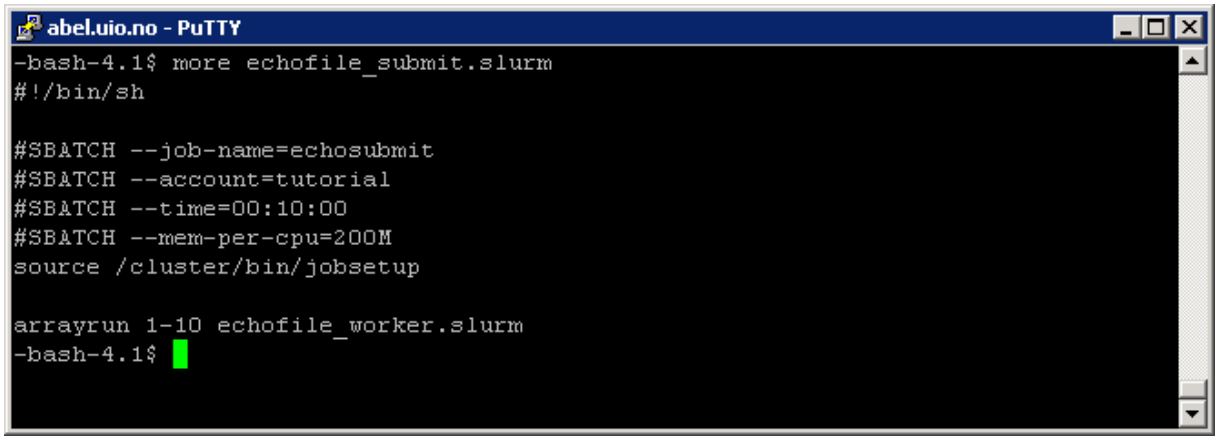
7.3. Arrayrun job scripts

Every arrayrun needs one “worker” script and one “submit” script.

The worker script calls your program and it is similar to the scripts we have already used in this tutorial. The modifications consist of the file management using the TASK_ID variable. Two extra variables DATASET for input and OUTFILE for output were added. Their names include the value of TASK_ID. In this example in run 1, the input is called “data.1” and output “result.1” and in the last, 10th run “data.10” and “result.10” respectively.



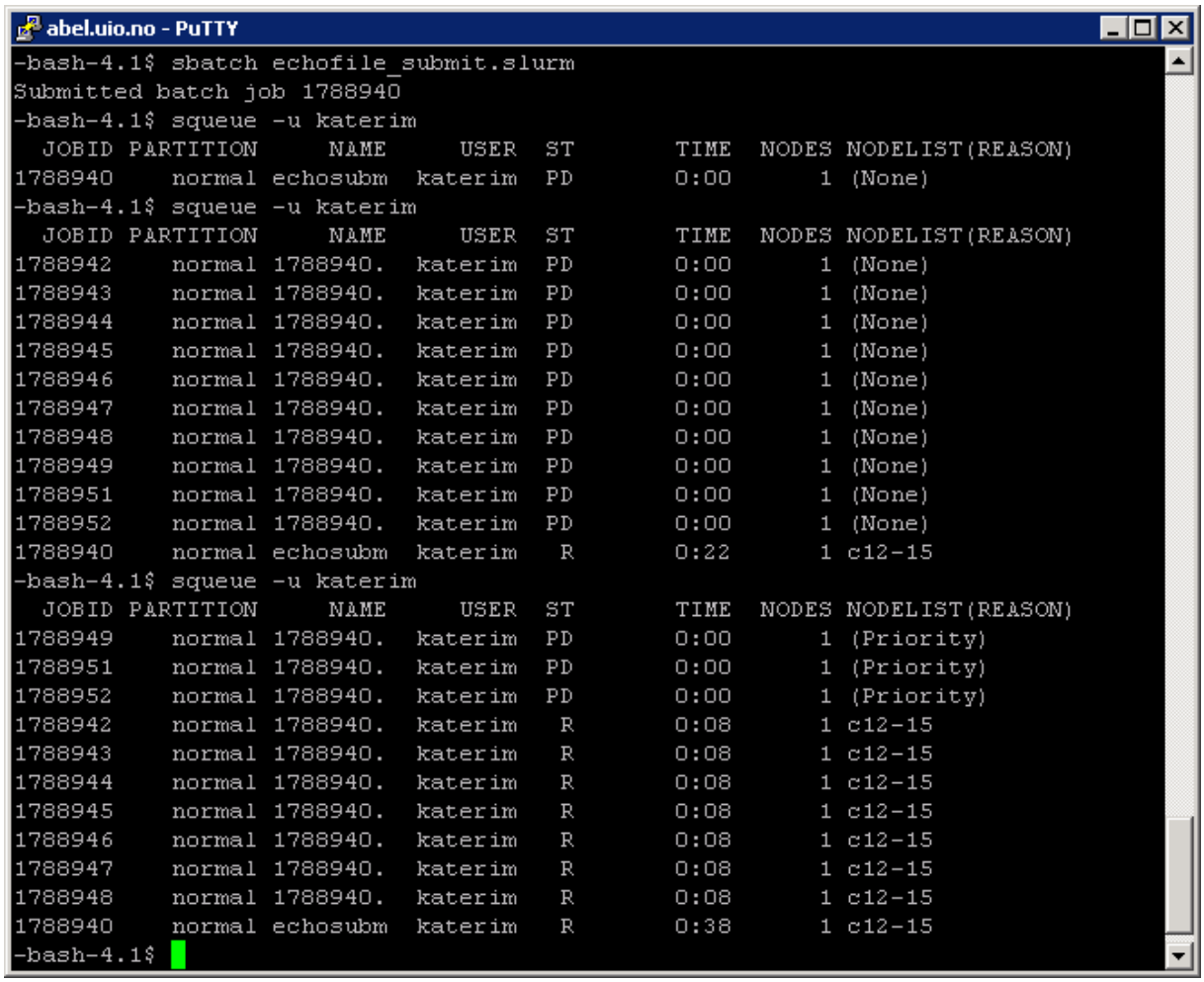
The submit script executes the arrayrun command. The arrayrun will submit all the jobs and then check on their status. It will be finished when all jobs are done. The submit script contains the same SBATCH settings as an ordinary job script. Note that the memory consumption for this job is low but the time setting is high. The time for the submit job should always be much longer than the time for individual jobs because the jobs are not started all at once but are released in batches. The command arrayrun needs parameters specifying the number of runs and the name of the worker script:



Submit the arrayrun by "sbatch echofile_submit.slurm" and then type "watch squeue –u your_user_name". You can see the individual jobs being released by the submit script.

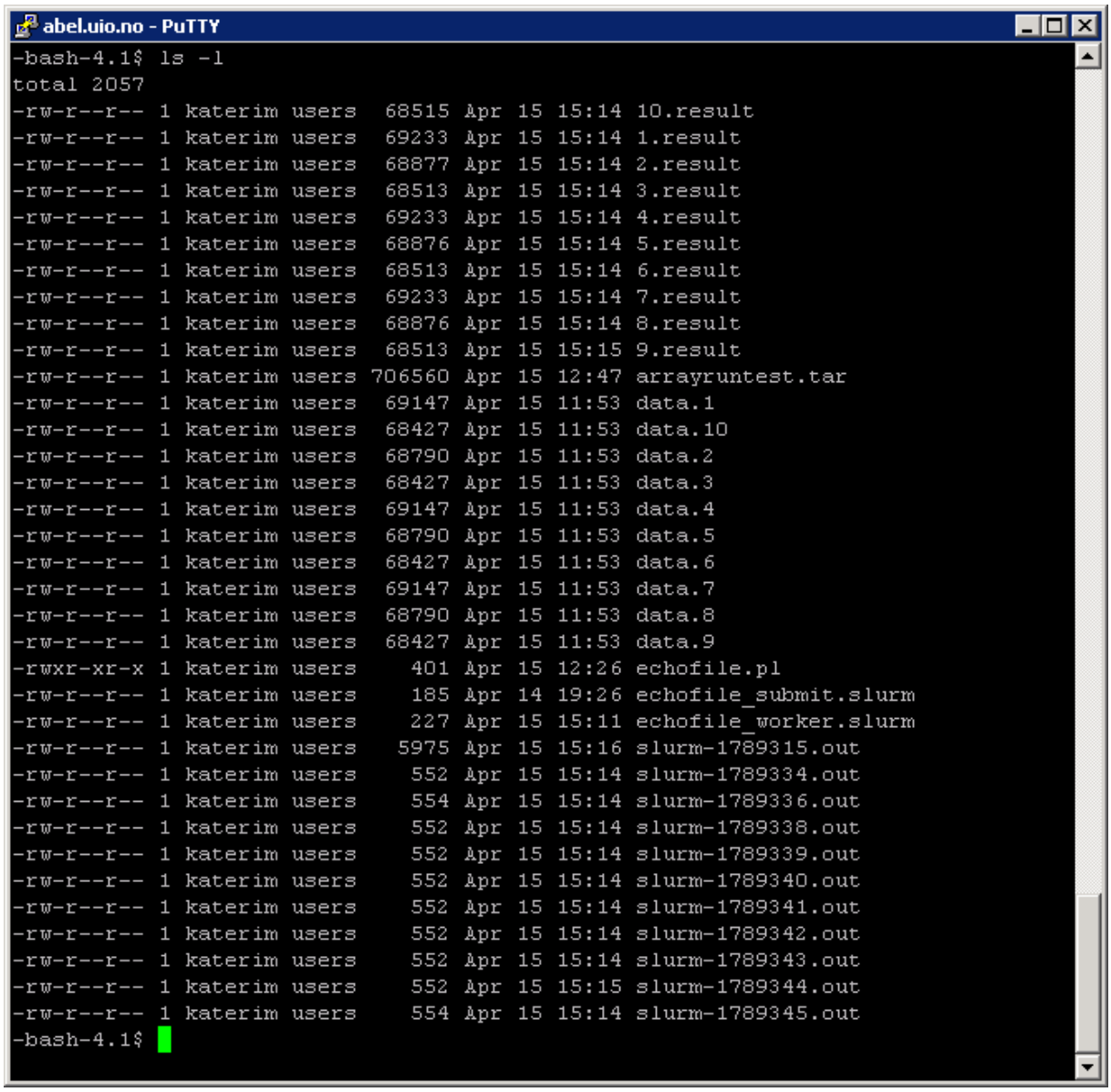
Since arrayrun can release many jobs, it is necessary to know how to cancel them if needed. To cancel arrayrun and all worker processes, use "scancel jobid", jobid being the id of the submit script.

In the following figure, you can see output of the squeue command. In the first instance of, the submit script is waiting to run. In the second instance, the worker scripts have been released and are waiting to run. Finally, in the third instance, most of the worker scripts started to run:



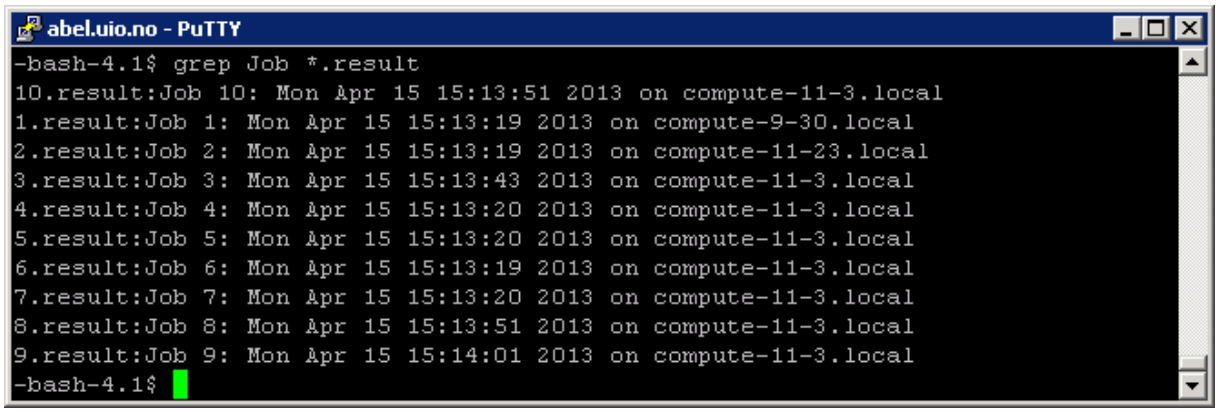
7.3 Arrayrun results

When you see no more jobs in the queue you can examine the results. Your arrayruntest directory should look like this:



You can see the result files (1-10.result) and also slurm log files. There is one log file for each run as well as for the submit script (11 all together). The biggest slurm log is always the one produced by the submit script.

Finally, we examine the content of the result files. They contain log of time and place of execution. Type "grep Job *.result" to see this information from each file at once.



Exercise: Try to run larger arrayrun:

- edit the submit script to add more runs (pls do not go over 50, our queue is limited).
- get more input data files, use a tar file called bigdata.tar in /cluster/teaching/abel_tutorial that contains more data
- change the input file name in your worker script
- increase the time limit in your submit script to one hour (--time=01:00:00)

Watch the queue, you see the jobs being released in batches. When your all is finished you can examine the time information in the output files. Can you see the batches of jobs being released by arrayrun?

Tip: To examine the results, use “grep Job *.result | sort –n “. This will give you an overview about the arrayrun execution.