The epib-genstat computer cluster

Dr. L.C. Karssen

February 1, 2016

Contents

1 Introduction

Welcome to the epib-genstat computer cluster. This cluster is maintained by the Genetic Epidemiology group of the Erasmus Medical Center in Rotterdam, The Netherlands. This page intends to help the users of the cluster with their daily tasks. A PDF version of this document can be downloaded here.

The epib-genstat cluster can be accessed using the server name `epib-genstat.erasmusmc.nl` and presently consists of three servers (so the term is actually a bit of an overstatement):

- `epib-genstat.erasmusmc.nl`, the "central hub" (also called `epib-genstat4.erasmusmc.nl`) and
- `genstat-node01`, `genstat-node02`, two compute nodes,

all running the Linux operating system. If you have never used Linux before take a look here for exercises and slides from the course "Linux for scientists" given in October 2013.

We hope the information on this page will help you in your work on the epib-genstat cluster. Keep an eye on this page as it is updated regularly. If you think of any subject that should be added to this page, feel free to contact Najaf Amin.

In several section below you will find examples of commands to type. If the first character is a `$` this indicates the command prompt of the Linux shell. You don't have to type this character.

1.1 Other sources of information

Apart from the information on this page the following resource might be of help:

- The GenEpi Wiki: http://epib-genstat.erasmusmc.nl/genepiwiki
- The "Linux for Scientists" course given at Erasmus MC

2 Connecting to the cluster

The epib-genstat cluster can be accessed from both inside the Erasmus MC network as well as from the outside world. All connections are secure (i.e. all data you send and type is encrypted). Be sure to keep your password safe!

After five failed attempts to log in, e.g. because you mistyped your password, your computer will be blocked for 10 minutes (except if you connect from within the `erasmusmc.nl` domain). This security measure makes it more difficult for people trying to hack the system.

2.1 Obtaining an account

An account can be obtained from Najaf Amin, Room Na-2716.

Remember that you are the person responsible for any use or abuse of your account. Therefore, behave responsibly (e.g. do not use the servers to hack into other systems) and **don't share accounts/passwords** (prospective users can contact me for their own account). Non-compliance with these rules will lead to your account being suspended (or worse).

When asking for an account please provide the following details:

- First name
- Last name
- City
- Department/group
- e-mail address
- employee type (MSc student/PhD student/postdoc/guest/other)
- supervisor (in case of a student account)
- account duration (the time span for which you would like to have an account, e.g. 4 years for a PhD student, 1/2 year for an MSc student; this can always be extended)
- A short description of the research the account will be used for as well as the names of the cohorts you would like to use.

2.2 Disk quota

In order to make sure that no single user can use up all disk space (accidentally or not), every user has a certain disk quota allotted for his/her files. You can check your quota status with the `quota` command:

```
lennart@epib-genstat4:~$ quota -s
Disk quotas for user lennart (uid 1305):
  Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
    /dev/md0    206G    250G    255G            4800       0       0
```

The columns of interest are columns 2, 3, 4 and 5 (you can forget about the last four columns). The `blocks` column shows your current usage (206GB in this case), the `quota` column shows your maximum. You will be notified by e-mail if you exceed it. After exceeding this maximum disk space you will have 7 days of 'grace' (the number of days left is noted in the `grace` column), in which you can still use some more space (up to the value in the `limit` column, 255GB in this example). Once the grace period has expired you will not be able to create any more files. This can even prevent you from logging in!

Therefore, make sure that you remove some files in your `/home` directory as soon as you notice you are exceeding your quota, e.g. by throwing away unnecessary files (any unused or oversize `.Rdata` files lying around?), or making backups of files on another computer or a DVD). If, after cleaning up, you still think you need more disk space, you can go to the system administrator and try to convince him to increase your quota.

2.3 Connecting to or from another Linux system or an Apple computer

Connecting from another Linux system (or any other Unix-like, e.g. Apple's OS X) can be done with the SSH (Secure SHell) program:

```
$ ssh your_username@epib-genstat.erasmusmc.nl
```

If you also want windows (e.g. R's plot screen) to appear on your screen than the `-X` option must be added:

```
$ ssh -X your_username@epib-genstat.erasmusmc.nl
```

The first time you connect to the server you will be asked to accept the server's signature. Simply do so. If such a message occurs at a later time be careful. Unless there has been any major changes to the server (of which you will be notified by e-mail) this likely indicates an attack on the server. Please contact Najaf Amin (room Na-2716) if this happens.

2.3.1 Transferring files (Linux)

Files can be copied to and from the server with the `scp` command. A copy action is structured like this:

```
$ scp source destination
```

where either `source` or `destination` consist of a `username@server` part followed by a colon and the file(s) to copy from or to, respectively. For example, the command

```
$ scp local_file your_username@epib-genstat:a_directory/on/the/server/
```

copies a local file to the server and puts it in the specified directory. Copying a file from the server to your own computer works similarly:

```
$ scp your_username@epib-genstat.erasmusmc.nl:~/path/to/the/file ~/a/local/dir/
```

When copying directories including the files they contain the `-r` option must be used:

```
$ scp -r some_dir/ your_username@epib-genstat.erasmusmc.nl:the_target_directory/
```

For transfers of many and/or large files the `rsync` command is better because if something goes wrong and the transfer is aborted (as mentioned in section The batch queue system tasks taking more than 10 minutes of processor time will be killed unless the batch queue system is used), rerunning the `scp` command will start from scratch again. `rsync` is much smarter about these cases and will transfer only (parts of) files that have not been sent yet. An `rsync` command looks just like an `scp` command:

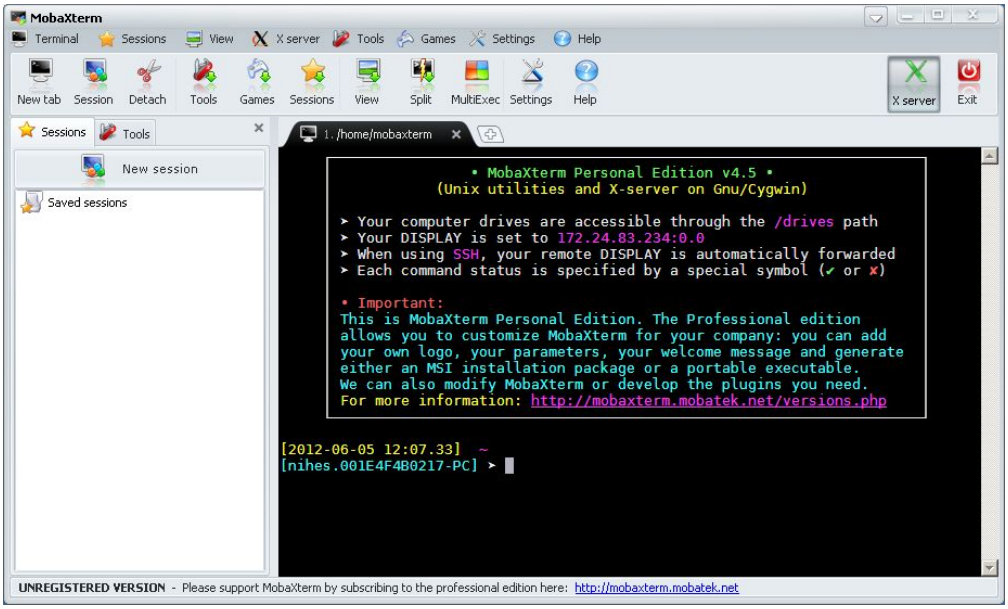
```
$ rsync -azP source destination
```

The `-a` option preserves file properties like dates, owner (if possible) etc. and sets the recursive option so that directories are transferred as well. The `-z` option turns on compression of the data (which can increase transfer speeds for some files) and the `-P` option adds progress information and keeps partially transferred files. As with `scp` `source` or `destination` can be a remote location specified as `username@servername:path/to/files`.

2.4 Connecting from a Windows computer

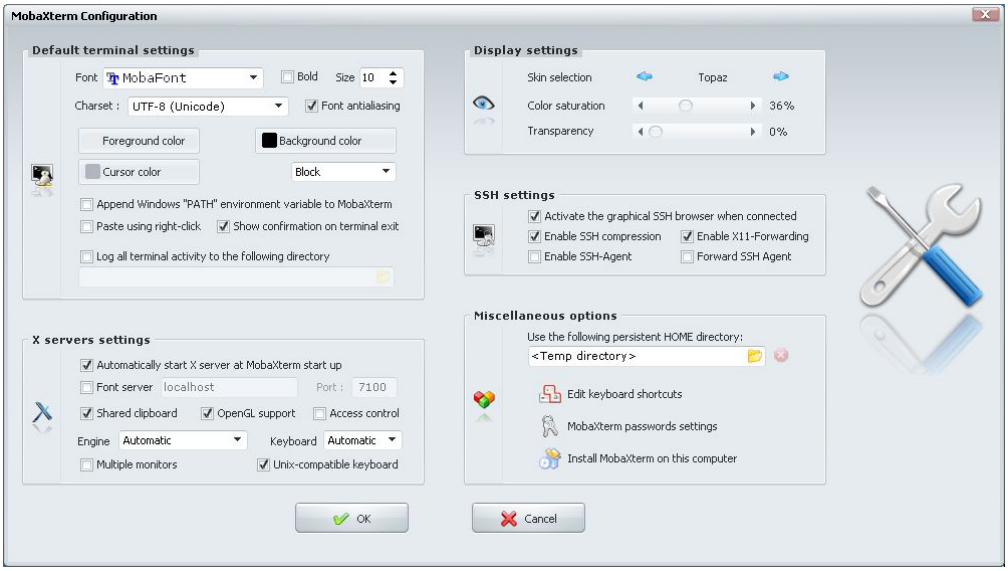
To connect to the cluster from a computer running MS Windows the MobaXterm program can be used. Download the MobaXterm program here and unzip it to a new folder. Because the program creates some extra files it is better not to save it on your Desktop, but in a separate folder instead (e.g. a subdirectory of your *My Documents* folder). Of course you can create a shortcut to the program on your Desktop.

Start the MobaXterm program and you will see the following window



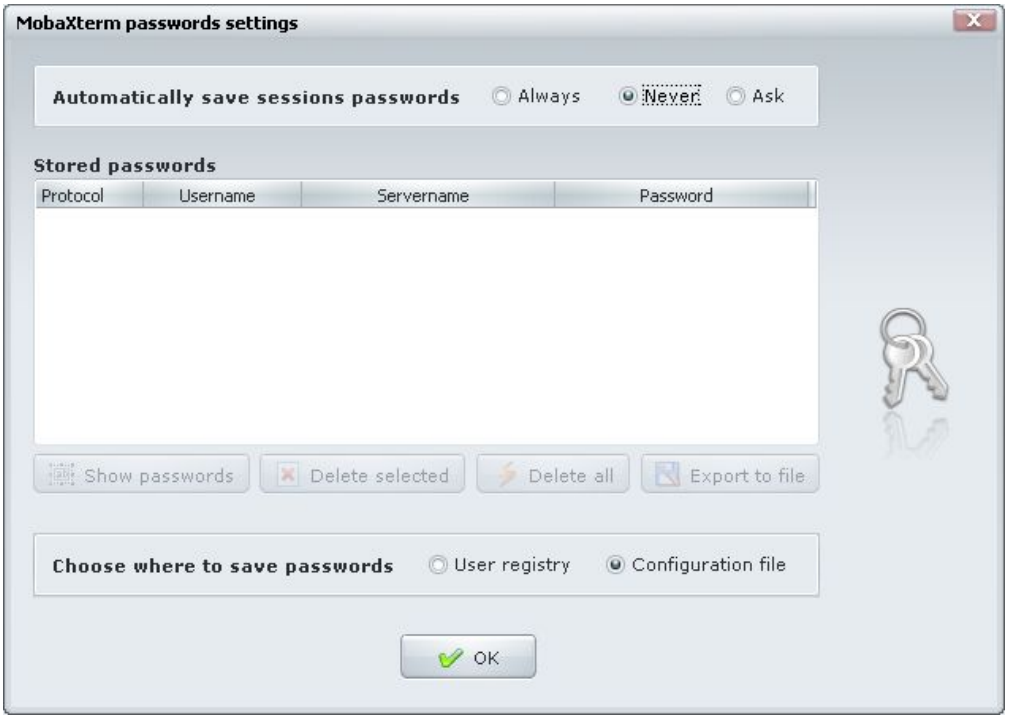
2.4.1 Configuring MobaXterm

If this is the first time you run MobaXterm take some time to change a few settings. Click on the Settings button, which will show the following window:



In the top left corner of this window change the *Charset* to *UTF-8 (Unicode)* (this settings determines with how certain characters will be displayed on the screen).

Next, go to the lower left corner and click on *MobaXterm passwords settings*, which will pop up the following window

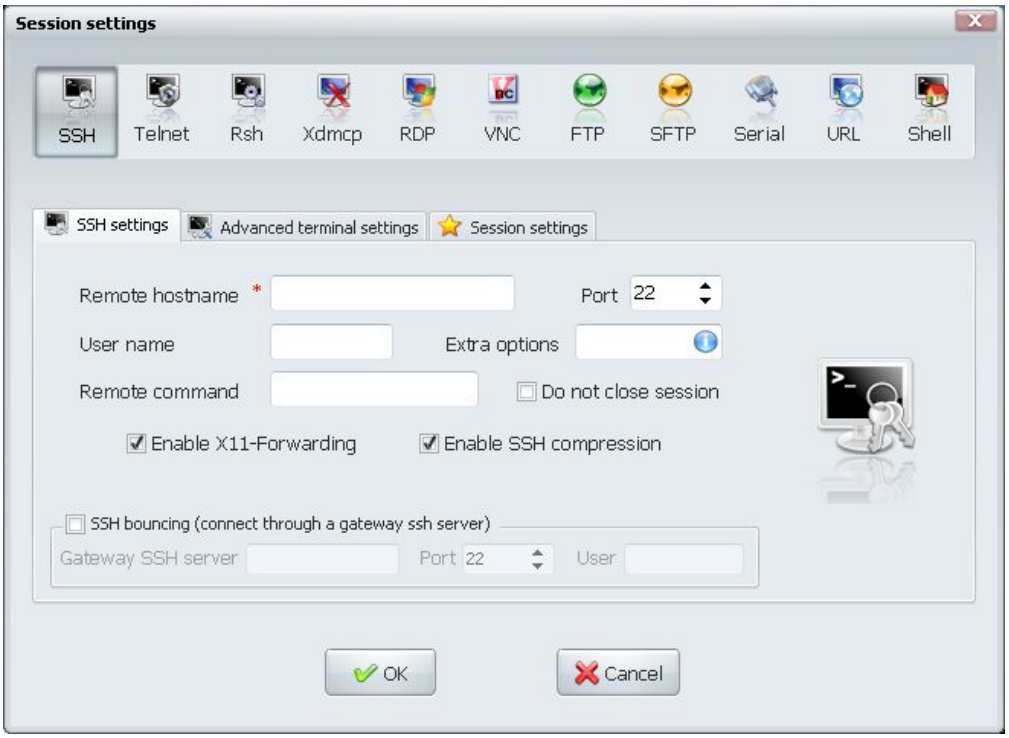


At the top of the window set the option *Automatically save sessions passwords* to *Never*. This way you won't connect to a server accidentally or have someone else (ab-) use your account.

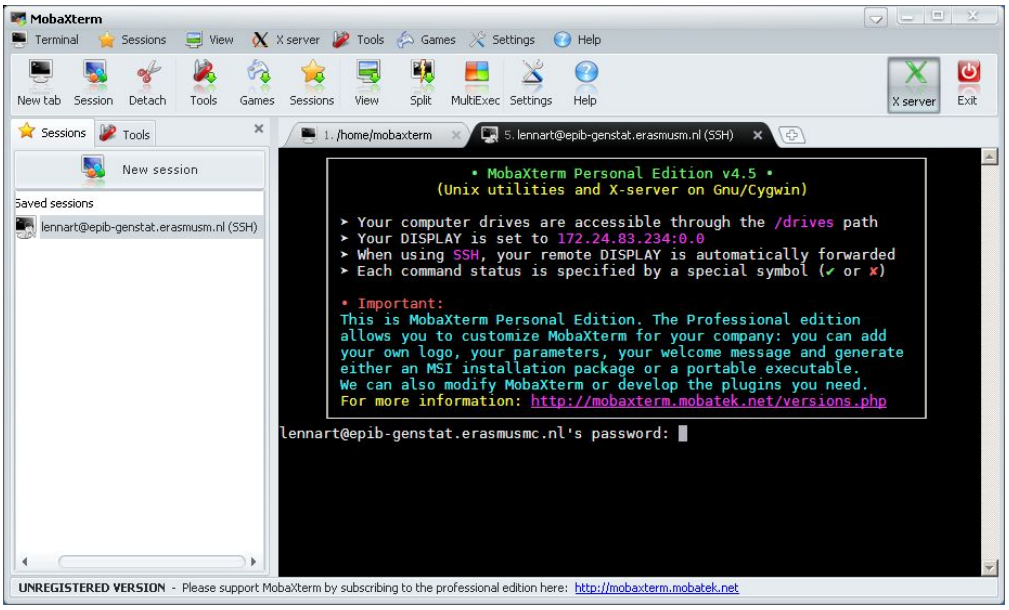
2.4.2 Access to the command line

To access a server or cluster click on the *session* button in the main MobaXterm window. This will pop up a window where you can select the type of connection.

Select *SSH* in the top row. You will be asked for the server name (*Remote hostname*, which is `epib-genstat.erasmusmc.nl` in our case), as well as your user name:

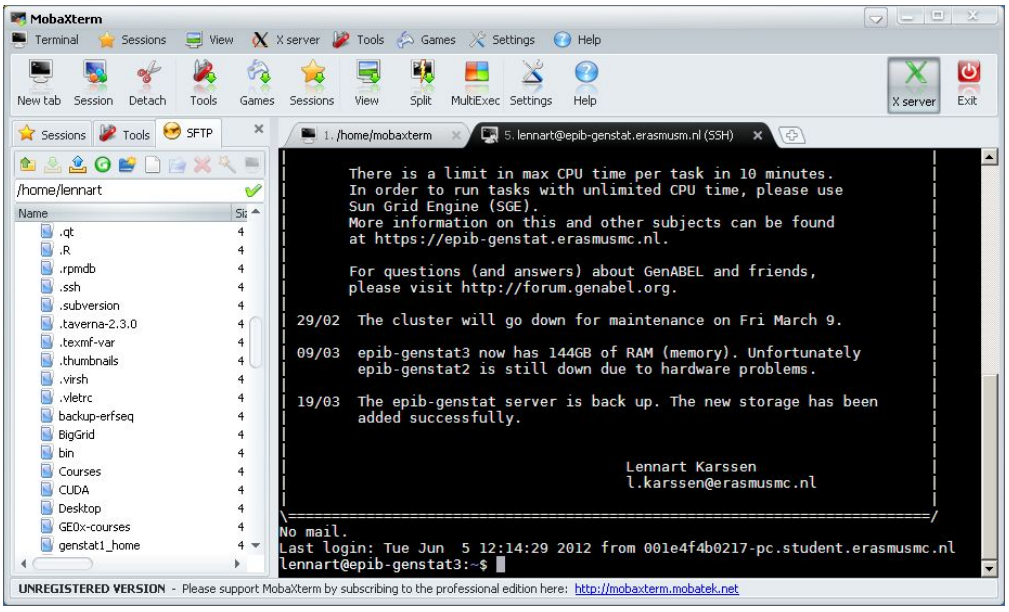


After clicking *OK* you will be asked for your password.



The first time you connect to the server you will be asked to accept the server's signature. Simply do so. If such a message occurs at a later time be careful. If there have not been any major changes to the server (of which you will be notified by e-mail) this likely indicates an attack on the server. Please contact Najaf Amin (room Na-2716) if this happens.

Finally you will end up on the command line, where the server is happily waiting to obey your commands:



On the left hand side of the MobaXterm window you will find a tab called *Sessions* which lists all servers that you have connected to. Double click on an entry to open a (or another) connection to a given server. The *SFTP* tab lists the directories as they exist on the server. This tab can also be used to copy files to an from the cluster (see Transferring files (Windows)).

Notice that the main (black) window can contain multiple tabs. Each new connection to a server that you start will be in its own tab, creating a clean and efficient interface. The first tab (named `/home/mobaxterm/`) is actually a Linux-like command line interface to your own PC. Most people won't use it and you can close that tab if you want.

The Putty program is an alternative option for connecting to the cluster. It is similar to MobaXterm, but less sophisticated. For example it doesn't forward windows from the cluster to your PC as discussed in the following section. Also transferring files to and from the cluster is not included. For that you would need a program like WinSCP (cf. section Transferring files (Windows)).

2.4.3 Forwarding windows from the server to your PC

By default MobaXterm forwards any window you create on the cluster (like plot windows in R) to your PC, so contrary to Putty (another program often used to connect to Linux servers) you don't have to do anything special.

To test if everything works enter the following command on the `epib-genstat4:~$` command prompt (don't copy the `$` sign, it is only there to indicate the shell prompt):

```
$ xclock &
```

A window showing a clock should pop up on your screen. If this works, start R and run the following code to get your graphical R window:

```
x <- 1:10
y <- x^2
plot(x, y, "b")
```

This should give you a nice plot of half a parabola:
./Rplots.pdf

2.4.4 Some useful programs for viewing different types of files

In the previous section you have seen that applications that produce windows can be run on the server as well. In the table below you will find a list of viewers for different file types. You can simply run them on the server by typing the command from the table followed by the name of the file you would like to view. This saves you the time of transferring the file up and down. Most of the time you will see some error or warning messages when you run these viewers. You can safely ignore them.

file type	viewer command
images like `.jpg`, `.gif`	`eog yourfile.png`
`.pdf`	`dbus-launch evince yourfile.pdf`
`.ps`	`dbus-launch evince yourfile.ps`

Note that for large files (> 100MB) or slow internet connections this is not ideal because every time you scroll or click this information has to be sent to the server and back, which will take a lot of time. For such large files it is better to download them to your PC and view them from there, as will be described in the next section.

2.4.5 Transferring files (Windows)

After connecting to the cluster you will see a list of files and directories on the left side of MobaXterm's window in the *SFTP* tab. You can drag and drop files to and from this window to transfer them to and from the cluster.

Alternatively, transferring files to and from the cluster can be done with the WinSCP program. Its screen is split up in two halves, the left one showing your files and directories on your local computer, and the right one showing the files and directories on the cluster. On ErasmusMC computers, the WinSCP programme can be installed through the "advertised programs" system.

2.5 Sharing data with other genstat users

The easiest way to share data with a colleague who also has an account on genstat is the following:

- Copy the file (or directory) that you want to share to the `/tmp/` directory. Note that this directory is used for temporary stuff (hence its name) and there is no guarantee that files stored there will not be deleted (in fact, `/tmp/` is cleaned up regularly).
- The other user can then copy the file (or directory) to his or her own home directory for further usage.
- The original owner of the file shouldn't forget to remove the file from `/tmp` once it has been copied.

By default (although this can be altered on an individual basis) the access permissions on files you create may be too restrictive and as a result, the other user will get a "permission denied" error. The permissions on a file can be checked with the `ls -l` command:

```
$ ls -l /tmp/myFile
-rw-r----- 1 lennart genepi 28 2010-11-30 18:43 /tmp/myFile
```

The permissions are listed in the first column of 10 characters, `-rw-r-----` in this case. Each of these positions indicate a permission. The first one is not actually a permission, it is labelled `d` for directories. Then follow three groups of three positions each. These three possible positions can be `rwx` for read, write and executable, respectively. If a certain permission is absent this is indicated by a `-`. The first group of three permissions is for the user who owns the file. The second group of three permissions pertains to the primary group of which the owner is a member. The last group of three permissions is for the other users (those that are not the owner and not a member of the primary group of the owner).

In the example above, the user `lennart` is the owner of the file and the group is called `genepi`, see the corresponding columns in the `ls` output. The permissions for the user `lennart` are `rw-`, which means that this user can read and write the file. Members of the `genepi` group can also read the file, but cannot write to it (that includes moving or deleting the file), because the permissions are `r--`. Finally, the permissions for other users (and groups) are `---`, so they cannot read from or write to the file.

To fix the access permissions, make the file readable to all users with the `chmod` command and notice how the output of the subsequent `ls` command has changed:

```
$ chmod a+r /tmp/myFile
$ ls -l /tmp/myFile
-rw-r--r-- 1 lennart genepi 28 2010-11-30 18:43 /tmp/myFile
```

To set the access permissions on all files (and subdirectories) of a directory `myDir` to readable by all users us the `-R` option (note that this is a capital `R` even though for other commands like `cp` and `rm` the option for recursive behaviour is `-r`). Also the executable permission must be set (if a directory is "executable" it means that you can `cd` into it):

```
$ chmod -R a+r /tmp/myDir
$ chmod a+rx /tmp/myDir
```

2.6 Sharing data with external people

The easiest with to share files with people who don't have an account on the genstat cluster is to put the data on your Personal web space, keep in mind that anyone with a web browser can access these files!

Information on how to copy files to other Linux servers (via `scp` or `rsync`) can be found in section Transferring files (Linux).

2.7 Personal web space

All users on the genstat cluster can easily create a personal web site. It basically consists of the following steps:

- Creating a directory `public_html` in your home directory

- Copying the files that make up the web site to that directory
- Setting the correct permissions on the directory and the files (they must be world-readable)
- Checking http://epib-genstat.erasmusmc.nl/~your_username to see if you can access the page/files.

The following sections describe the steps in more detail. Note that the most important part is getting the access permissions on the files and the directories right. **Be sure to read the Caveats section at the end**.

2.7.1 Setting up the directory

The directory used for your personal web space is called `public_html`. Everything you put in this directory will be accessible from the web (as long as you have set the correct permissions). If the directory doesn't exist, create it:

```
$ cd
$ mkdir public_html
```

Check the access permissions:

```
$ ls -ld public_html
drwxr-xr-x 2 lennart genepi 4096 2010-12-06 19:05 public_html/
```

The important part here are the permissions for 'others', the last three positions in `rwxr-xr-x` (i.e. the last `r-x`). This means that 'others' (i.e. all people who are not yourself or part of your main group) have read (`r`) and access (`x`) permissions. If the 'other' permissions are not `r-x` set them now:

```
$ chmod o=rx public_html
```

and check with `ls -ld` as before.

2.7.2 Copying files to the directory

Now that the directory is set up correctly you can copy files to it. By default, if you visit http://epib-genstat.erasmusmc.nl/~your_username/ the web server software looks for a file with the name `index.html`. To test if everything works create an `index.html` file with the following contents:

```
<head>
<title>My own test page</title>
</head>
<body>
<p>Hello there, this is a test page</p>
</body>
```

2.7.3 Setting the correct permissions on files and subdirectories

The files in the `public-html` directory must be readable by the web server. Therefore all files must have the `read` permission set for the 'others' group, and all directories should have both the `read` and the `execute` permission set. To set this right for all files and directories at once run the following commands:

```
$ find ~/public_html -type f -print0 | xargs -0 chmod o=r
$ find ~/public_html -type d -print0 | xargs -0 chmod o=rx
```

2.7.4 Caveats

Setting up a publicly accessible web site comes with responsibilities:

- Everything you put in the `public_html` directory and the has read permissions for *others* will be accessible (and will also be indexed by search engines (Google)).
- You are responsible for the content you put up there. Make sure that is conforms to the Erasmus MC guidelines.

3 The batch queue system

The cluster uses the Sun Grid Engine (SGE) v6.2u5 job queue system in order to spread the load on the systems in the cluster. Background jobs (i.e. programs not using the SGE queues) will be killed after 10 minutes. So it is usually of little use to start an R session or run ProbABEL directly from the command line.

The idea behind this queue system is that the computational load can easily be shared across multiple servers and to make sure two jobs don't interfere with each other or fight for resources (e.g. disk access or memory). As you can see in the server specifications section, the cluster has many processors (CPUs) at its disposal. Therefore, always try to split your tasks into small chunks. For example, instead of running a genome-wide association analysis from chromosome 1 tot 22 on one processor you should submit one job for each chromosome. These 22 jobs will then be run in parallel (assuming the cluster is not too busy), so you won't have to wait very long for your results.

After submitting a job to SGE it will be processed in a so-called queue. Each queue has a certain number of slots. If there are more jobs than slots in a queue the excess jobs will wait for a slot to become available (cf. section Following progress to find out the number of slots per queue or to see whether your job is already running).

Each queue has specific properties and SGE uses these properties to decide which queue to send your job to. The first property that is checked is which group you are a member of. One set of queues is for members of the `genepi` group, the other set of queues is for all other people (use the `groups` or the `id` command to find out which

groups you are in). See also section Long vs. short vs. low priority queue to find out about the differences between the short and long queues.

3.1 Submitting jobs to the SGE queues

Suppose you want to run a certain R script `myscript.R`. Normally you would either start R and then

```
> source("myscript.R")
```

or from the Linux command line you would run:

```
$ R --vanilla -q -f myscript.R
```

3.1.1 Quick and dirty

The quickest way to submit such a task to an SGE queue is the following:

```
$ qsub -cwd -b y R --vanilla -q -f myscript.R
```

Here `qsub` is the command to submit a job to the queue system. It is followed by zero or more options that relate to `qsub` (two in this case, `-cwd` and `-b y`) and finally the actual command you want to run (together with the options that belong to that program). The first `qsub` option, `-cwd`, stands for "use the current working directory". It tells SGE to look into the present directory for the files you specify (`myscript.R` in this case) and to write its output files there as well. The option `-b y` tells SGE that the command you want to execute is not a script but a binary program (R in this case).

Each job that is sent to the queues receives a unique ID, the job ID. This is useful for distinguishing between several jobs you might have waiting in the queue, but is also necessary when you want to delete a job from the queue (cf. section Deleting jobs from a queue).

By default SGE will create two files for each job in the queue. One that contains the output that would normally appear on the screen and one that contains the errors that would normally be sent to the screen. These files will have a name that starts with the name of the command you sent to the queue followed by a period, the letter `o` or `e` for output and error, respectively, and finally the job ID. For the R command that we submitted earlier the files would be called

```
R.o2823
R.e2823
```

(where of course the number at the end is the job ID, which will be different in your case).

3.1.2 Using a submission script

The preferred way to send a job to the queue system is by using a submission script. Using a script has several advantages:

- you don't have to remember all the command line options for the `qsub` command, you simply copy your submission script from the previous time you used it (or this website) and only change the program that you want to run.
- All the standard Linux shell scripting tricks are at your disposal.

  A simple example of a submission script for the example of the R script `myscript.R` used earlier would be:

```
#!/bin/bash
# This is a sample submission script. Lines starting with # are
# comments. The first line (with #!) should be in every script.

# Let's set some variables for SGE. Lines starting with #$ are
# interpreted by SGE as if they were options to the qsub command
# (dn't remove the # from the lines starting with #$).
#$ -S /bin/bash
#$ -cwd

# This is the command we would like to run in the queue
R --vanilla -q -f myscript.R
```

Save this submission script to the same directory as where `myscript.R` is located and call it for example `job.sh`. Make sure the script is executable by running

```
$ chmod a+rx job.sh
```

Now it can be submitted to the queues like this:

```
$ qsub job.sh
```

ORG-LIST-END-MARKER

3.1.3 Refinements to the submission script

The script presented in the previous section is simple but sufficient for basic tasks. Here we present some additions to the script that can make life with SGE easier. A script file with all the suggested options can be downloaded here. The only things that need to be changed are the e-mail address and the last line where you fill in the command(s) you want to run.

- Start/stop e-mails
  Since most jobs will take more than 10 minutes to complete (otherwise you could have run them without using the queue system, right!) it would be nice to get an

e-mail when the job is finished so that you don't have to run `qstat` all the time (cf. section Following progress). To get an e-mail when a job begins and when it ends simply add the following two lines after the `#$ -cwd` line in the aforementioned simple script:

```
#$ -M your_address@erasmusmc.nl
#$ -m be
```

- Output to a single file
  As discussed at the end of the section Quick and dirty the output and error messages of a job are recorded in separate files. When running a large amount of jobs this can lead to a proliferation of these files. By adding

  ```
  #$ -j y
  ```

  to your submission script the output and error files will be joined into one file of the form `script.o2345`.

3.2 Long vs. short vs. low priority queue

By default jobs end up in the long queues (i.e. `genepi-long` or `non-genepu-long`, depending on your group membership). Jobs that you know take less than 2 hours (CPU time) can be run in the short queues (`genepi-short` and `non-genepi-short`). To submit a job in the short queue add `-l short` to your qsub command, for example:

```
$ qsub -cwd -l short myscript.sh
```

Note that the two-hour limit is automatically enforced. If your jobs takes more than two hours CPU time it will automatically be killed.

Jobs in the short queue will run with a higher priority than those in other queues. If the total number of jobs is too high, jobs in the long queues will be suspended. until the jobs in the short queues have finished. This way someone with a short task will not have to wait until all long tasks have finished.

There is also a low priority queue. In this queue the jobs are only execute when there are no other jobs waiting to be executed. Jobs that are low priority will be temporary paused (suspended) when other jobs from the long and short queu want to run. This is a perfect solution when you have a great number of jobs, but do not want bug other users. To submit a job in the low priority queue add `-l lowprio` to your qsub command, for example:

```
$ qsub -cwd -l lowprio lowscript.sh
```

3.3 Following progress

The `qstat` command allows you to see whether your job is accepted by one of the queues, which jobs you have submitted so far, how many jobs are waiting in the queues, etc. Simply running

```
$ qstat
```

will show your own running and waiting jobs. Running the command

```
$ qstat -f
```

will give an overview of all queues, even the ones in which you don't have any jobs running. To show all jobs of all users in all queues use

```
$ qstat -f -u \*
```

This gives you an idea how busy the cluster is. A sample output on a quiet day looks like this:

```
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
genepi-long@epib-genstat4.eras BP    0/3/11         1.31     lx26-amd64
    996 0.50207 EGL        user1       r     01/17/2012 11:36:49     1
   1223 0.06115 conscienti user2       r     01/23/2012 14:31:03     1
   1224 0.06050 conscienti user2       r     01/23/2012 14:43:48     1
---------------------------------------------------------------------------------
genepi-short@epib-genstat4.era BP    0/0/11         1.31     lx26-amd64
---------------------------------------------------------------------------------
int@epib-genstat4.erasmusmc.nl IP    0/1/5          1.31     lx26-amd64
   1226 0.50000 QRLOGIN    user3       r     01/24/2012 09:02:02     1
---------------------------------------------------------------------------------
non-genepi-long@epib-genstat4. BP    0/0/11         1.31     lx26-amd64
---------------------------------------------------------------------------------
non-genepi-short@epib-genstat4 BP    0/0/5          1.31     lx26-amd64
```

It shows the user `user2` has two jobs in the queue called `genepi-long` with job IDs 1223 and 1224. Both jobs are running some script whose name starts with `conscienti`. The queues called `genepi-short`, `non-genepi-long` and `non-genepi-short` are empty as is the queue called `int`, which is used for interactive jobs only (see section Interactive jobs). The `resv/used/tot.` column show the number of reserved and available slots in each queue as well as the total number of available slots. So in this example `all.q` has no reserved slots and two slots are in use out of a total of seven. If there are more jobs than slots in a queue the excess number of jobs will have to wait until slots become available again.

On a busier day you can come across the following output of `qstat -f -u \*`:

```
queuename                      qtype resv/used/tot. load_avg arch          states
---------------------------------------------------------------------------------
genepi-long@epib-genstat4.eras BP    0/11/11        12.86    lx26-amd64
    996 0.50000 EGL        user1        r     01/17/2012 11:36:49     1
   1297 0.00895 pipeline_t user2        r     01/27/2012 09:48:55     1
   1326 0.00501 user3_cc_a user3        r     01/27/2012 11:43:19     1 1
   1326 0.00501 user3_cc_a user3        r     01/27/2012 11:43:19     1 2
   1326 0.00501 user3_cc_a user3        r     01/27/2012 11:43:19     1 3
   1326 0.00501 user3_cc_a user3        r     01/27/2012 11:43:19     1 4
   1326 0.00501 user3_cc_a user3        S     01/27/2012 11:43:19     1 5
   1326 0.00501 user3_cc_a user3        S     01/27/2012 11:43:19     1 6
   1326 0.00501 user3_cc_a user3        S     01/27/2012 11:43:19     1 7
   1326 0.00501 user3_cc_a user3        S     01/27/2012 11:48:04     1 8
   1326 0.00501 user3_cc_a user3        S     01/27/2012 11:48:04     1 9
---------------------------------------------------------------------------------
genepi-short@epib-genstat4.era BP    0/5/11         12.86    lx26-amd64
   1339 0.00007 R          user6        r     01/27/2012 14:08:07     1
   1340 0.00004 R          user6        r     01/27/2012 14:08:52     1
   1341 0.00003 R          user6        r     01/27/2012 14:09:07     1
   1342 0.00003 R          user6        r     01/27/2012 14:09:07     1
   1343 0.00003 R          user6        r     01/27/2012 14:09:07     1
---------------------------------------------------------------------------------
int@epib-genstat4.erasmusmc.nl IP    0/3/5          12.86    lx26-amd64
   1273 0.04823 QRLOGIN    user6        r     01/26/2012 14:46:02     1
   1298 0.00750 QRLOGIN    user5        r     01/27/2012 10:31:09     1
   1337 0.50015 R          user4        r     01/27/2012 14:04:52     1
---------------------------------------------------------------------------------
non-genepi-long@epib-genstat4. BP    0/0/11         12.86    lx26-amd64
---------------------------------------------------------------------------------
non-genepi-short@epib-genstat4 BP    0/0/5          12.86    lx26-amd64

############################################################################
 - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS
############################################################################
   1327 0.25435 GWAS       user4        qw    01/27/2012 12:02:24     1
   1330 0.17089 GWAS       user4        qw    01/27/2012 12:06:08     1
   1331 0.12920 GWAS       user4        qw    01/27/2012 12:06:43     1
   1326 0.00501 linda_cc_a user3        qw    01/27/2012 11:43:14     1 10-22:1
```

Here you see that 11 jobs are in the `genepi-long` queue, 6 of which are running (indicated by `r` in the fifth column), but 5 are suspended (indicated by an `S`). These last 5 jobs are suspended because `user6` has submitted 5 jobs to the `genepi-short` queue,

which takes precedence. At the bottom you see a list of pending jobs. Those jobs have been submitted, but can not start yet because there are no slots available. That's why they have `qw` in the fifth column. As soon as one of the running jobs finishes the one from the top of the pending list will take its place (job ID 1327 in this case).

3.4 Deleting jobs from a queue

At some point you will find that you want to delete a job from the queue. This may happen because you submitted five R jobs, for example, and the first one finished early because you made a typing mistake in the R code. Since the other jobs use the same R code they will finish with an error as well so you decide to remove them from the queue. For this you use the `qdel` command followed by the job ID. Use `qstat` (cf. section Following progress) to find the job ID of your jobs. Running

```
$ qdel 2844
```

would kill the second job in the list shown in Following progress section. Of course a user can only delete her/his own jobs.

3.5 Getting info on a finished job

To get information on a job that has finished, use the `qacct` command in combination with the job ID:

```
$ qacct -j 8765
==============================================================
qname        genepi-short
hostname     epib-genstat4.erasmusmc.nl
group        genepi
owner        some_user
project      NONE
department   genepi
jobname      probabel.pl
jobnumber    8765
taskid       undefined
account      sge
priority     8
qsub_time    Mon Mar  7 22:56:01 2011
start_time   Mon Mar  7 22:56:15 2011
end_time     Tue Mar  8 03:59:44 2011
granted_pe   NONE
slots        1
failed       0
exit_status  0
ru_wallclock 18209
ru_utime     17362.282
ru_stime     816.388
ru_maxrss    0
ru_ixrss     0
```

```
ru_ismrss     0
ru_idrss      0
ru_isrss      0
ru_minflt     328576642
ru_majflt     2
ru_nswap      0
ru_inblock    0
ru_oublock    0
ru_msgsnd     0
ru_msgrcv     0
ru_nsignals   0
ru_nvcsw      12597
ru_nivcsw     443538
cpu           18178.669
mem           21867.259
io            170.594
iow           0.000
maxvmem       2.037G
arid          undefined
```

The most interesting elements of the output are `start_time`, `end_time`, for the time at which the job started and when it finished, respectively. Note that the submit time has a separate entry. The value of `cpu` shows you the number of seconds of CPU time the job used. The value of `ru_wallclock` shows the total time (in seconds) that the job took (i.e. CPU time, but also time used for reading/writing files etc.). The values `qname` and `hostname` tell you in which queue and on which server the job was run.

In order to find out how much time your jobs have spent in the queue over the last 15 days run (inserting your own username, of course):

```
$ qacct -o lennart -d 15
OWNER        WALLCLOCK          UTIME          STIME            CPU ↩
                MEMORY                     IO                   IOW
================================================================================

lennart          96279      10798.155       1903.326      12701.481 ↩
              3529.165            4165.959                  0.000
```

The `WALLCLOCK` time is the total time in seconds that your jobs have spent running in the queue (in the last 15 days). The `CPU` column shows the time (in seconds) the job was actively using a CPU, i.e. not waiting for other things like reading or writing a file (which is listed in the IO column).

3.6 Interactive jobs

Although it is not the preferred way to run programs, sometimes it may not be possible or efficient to write a complete script to submit to the queue. For example, you'd like to start an R session and enter the commands on the R command line because you are not sure whether a certain construction works on the cluster.

In such a case the interactive queue (`int.q`) can be used. Starting a session in the interactive queue is done like this:

```
$ qrsh -pty y <command>
```

where `<command>` is the program you'd like to run (`R` or `solar` are likely examples). You will then be asked for your password and an interactive session is started on either one of the servers in the cluster (if there are slots available in the interactive queue of course).

Note that jobs running in an interactive queue will be killed after 48 hours of CPU usage and they will run with a very low priority.

If, for some reason you can't access your interactive queue session anymore, you can kill the old one using the `qdel` command.

3.7 Array jobs, or: repeating the same thing many times

One of the most common things in our field is to run the same analysis 22 times (once for each chromosome). Whether it is genotyping sequencing data or running a GWAS, the same steps have to be repeated for each chromosome. Remembering the things you learned from the Linux Course and/or the section on Using a job submission script your instinct should be to write a script that does what you want for a single item (file, chromosome, individual, whatever is applicable in your case).

You could be forgiven for thinking that the solution of adding a for-loop to your script is the solution. Of course, that will work but it will simply run one analysis after another and not make use of the multiple slots/CPUs that are available on the cluster. Alternatively, you could write a for loop that runs the `qsub` commands for you. That would already be much better. Assuming that enough slots are available all your jobs will be run simultaneously.

However, writing a for-loop to do the `qsub` commands for you isn't necessary. SGE has an option for that. Assuming that you can enumerate the items you are looping over (which happens to be the case with 22 chromosomes), you can set up what is called an "array job".

In short this is how it's done:

1. Create a job script that does the analysis for one item (e.g. chromosome)
2. In the script the variable `$SGE_TASK_ID` will be replaced with the sequence number of each item
3. Submit you job with the `-t start-stop` option (where `start` and `stop` are numbers you define)

3.7.1 An example

Let's say we need to run `probabel.pl` to do a GWAS for each of the 22 chromosomes. To run a GWAS for chromosome 1 your command line would look similar to this:

```
$ probabel.pl 1 1 linear ERF4_1KG_phaseIv3 --additive phenotypefile -o mypheno.chrom1
```

where the two 1s indicate the start and stop chromosome. Also notice the `-o` option where we specify start of the name of the output file. Because we want each of the chromosomes to have a separate output file (otherwise they may overwrite each other), I added the `chrom1` part.

In order to create an array job write a job submission script (call it `probabel_arrayjob.sh`, for example):

```
#!/bin/bash
# This is a quick submission script for an SGE array job

# Some SGE options
#$ -S /bin/bash
#$ -cwd
#$ -j y

# Create a variable $chr that contains the task ID
chr=${SGE_TASK_ID}

# Do a ProbABEL run for this chromosome
probabel.pl ${chr} ${chr} linear ERF4_1KG_phaseIv3 --additive ↩
    phenotypefile -o mypheno.chrom${chr}
```

Now we can submit the job to the queue for all chromosomes:

```
$ qsub -t 1-22 probabel_arrayjob.sh
```

When running `qstat` you will see the job show up like this:

```
---------------------------------------------------------------------------------
non-genepi-long@genstat-node02 BP    0/15/15        16.64    lx26-amd64
  10922 0.39428 probabel-a username     r     02/04/2013 23:50:34     1 2
############################################################################
- PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS - PENDING JOBS
############################################################################
  10922 0.39426 probabel-a username     qw    02/04/2013 15:44:04     1 3-22
```

The number in the last columns shows that chromosome 2 is being run on `genstat-node02` and that jobs 3-22 are still waiting in the queue (the subjob for chromosome one has probably finished already in this example).

3.7.2 Some refinements

The example script above is a bit rudimentary. Running it directly from the command line (i.e. not in the queue) in order to test it doesn't work because the variable `$SGE_TASK_ID` is not defined in that case. Here is an improved script that allows you to specify a number on the command line if you don't use the SGE system:

```
#!/bin/bash
# This is a quick submission script for an SGE array job
# If it is not run as an array job it will take the first command line
# argument as number.

# Some SGE options (you can add more of them, e.g. for sending e-mails
# when a job starts and/or finishes)
#$ -S /bin/bash
#$ -cwd
#$ -j y

# Create a variable $chr that contains the task ID. If $SGE_TASK_ID
# doesn't exist, use the first command line argument
if [ -n "${SGE_TASK_ID}" ]; then
    chr=${SGE_TASK_ID}
else
    if [ ${#} -ge 1 ]; then
        chr=$1
    else
       echo "No command line argument given and '$SGE_TASK_ID' doesn't ↩
          exist" 1>&2
        exit 1
    fi
fi

# Do a ProbABEL run for this chromosome
probabel.pl ${chr} ${chr} linear ERF4_1KG_phaseIv3 --additive ↩
   phenotypefile -o mypheno.chrom${chr}
```

Another refinement you could make is adding the line

```
#$ -t 1-22
```

to the script, so you don't even have to type that anymore when `qsub`-ing your job.

By default the `-t` option has a start an a stop value. If, for some reason your job number needs to increase by another value than one (say steps of two to only do even or odd numbers) you can add the step size to the `-t` option. Using

```
-t 100-200:5
```

makes `$SGE_TASK_ID` start at 100 and run to 200 with steps of 5. The step size can be accessed in the job script using the `$SGE_TASK_STEPSIZE` variable. The start value is available in `$SGE_TASK_FIRST`, the stop value is in `$SGE_TASK_LAST`.

3.7.3 If your variable is not a number

If the variable you want to loop over is not a number the solution is to create a file with all values on a separate line. In the job script you can then use `sed` to print only the relevant line and put it in a variable:

```
my_variable=‘sed -n "${SGE_TASK_ID}p" file_with_variables.txt‘
```

Now, if for example you have an analysis script that takes the name of a phenotype as input on the command line and you have a file listing each phenotype on a separate line you can use this as follows:

```
#!/bin/bash
# This is a submission script for an SGE array job that doesn't have a
# simple number to run through.
# If it is not run as an array job it will take the first command line
# argument as number.

# Some SGE options (you can add more of them, e.g. for sending e-mails
# when a job starts and/or finishes)
#$ -S /bin/bash
#$ -cwd
#$ -j y

# Create a variable $line that contains the task ID. If $SGE_TASK_ID
# doesn't exist, use the first command line argument
if [ -n ${SGE_TASK_ID} ]; then
    line=${SGE_TASK_ID}
else
    if [ ${#} -ge 1 ]; then
        line=$1
    else
        echo "No command line argument given and '$SGE_TASK_ID' doesn't exist" 1>&2
        exit 1
fi

# Extract the name of the phenotype from the phenotype list
phenotype=‘sed -n "${line}p" phenotypelist.txt‘

# Run your analysis script for this phenotype
my_analysis_script.sh $phenotype
```

Now, like before you can submit the job to the queue:

```
$ qsub -t 1-10 array_submission_script.sh
```

Of course you have to make sure that the stop number is not larger than the number of lines in your phenotype file.

3.7.4 Limiting the number of array subjobs that can be run simultaneously

Assuming you have an array job that needs to run from 1-100, but you don't want to run more than two subjobs at the same time (e.g. because they use too much memory if 10 of them would be running at once) you can add the `-tc` option to the `qsub` command line:

```
$ qsub -t 1-100 -tc 2
```

This will run at most 2 concurrent subjobs at the same time, even if more slots are available.

3.8 Running multi-processor jobs

Some programs can make use of the fact that computers have multiple processors (CPUs) or multicore CPUs. They can split their workload across several CPUs, thereby reducing the time to get your results.

So far, we have assumed that jobs that are run in the queue only use one CPU. If you run a job that requires for example four CPUs and you submit your job in the way described earlier (Submitting jobs to the SGE queues), the SGE system will think your job only uses one processor and will allow other jobs to use the remaining processors. If the server has a maximum of 11 CPUs, it will allow 11 single-CPU jobs. So if your job actually uses 3 CPUs, but you only reserve 1 slot (using `qsub` in the regular way) the SGE software will thing that 10 slots are still available instead of 8. Consequently, more SGE will oversubscribe the server and jobs will get in each others way, resulting in a slowdown of a all jobs in the queues. To fix this an option needs to be added to your `qsub` command, as will be shown below.

In short, submitting a multi-CPU job requires you to take the following steps:

1. find out how you tell your program to use multiple CPUs (for example by setting a command line option)
2. check how busy the queue is
3. based on that information decide on the number of CPUs you want to let the program use
4. submit the job

The following sections will give a bit more detail on each of these steps, based on an example of running the Unified Genotyper from the GATK.

3.8.1 Find out if (and how) a program supports using multiple CPUs

Usually information about multi-CPU capabilities of a piece of software can be found in the manual. Search for keywords like *multicore*, *SMP*, *parallel computing* and *shared memory* (terms like *MPI* and *Hadoop* refer to a different kind of multi-processor computing).

In the case of the GATK you can find information on this page. It turns out you simply need to add the `-nct` option followed by the number of processors you want to use to your GATK command line, for example

```
java -jar /path/to/GenomeAnalysisTK.jar \
          -l INFO \
          -R humref.fasta \
          -T UnifiedGenotyper \
          -I your.bam \
          -o my.vcf \
          --output_mode EMIT_ALL_SITES \
          -nct 4
```

(for using 4 processors).

3.8.2 Check how busy the queue is

Before submitting a multi-CPU job to the queue it is a good idea to see how many slots are available (with the command `qstat -f -u \*`). In principle, you could use create a job that requests all 11 CPUs of the server, but not only would that be inconsiderate to your colleagues, it would also mean that your job needs to wait until all currently running jobs have finished.

3.8.3 Decide on the number of CPUs to request from the queue

Now that you have an idea of the present load on the cluster you have to decide on how many CPUs you will claim for your job. In general I would suggest you use as most 5 or 6 CPUs per task.

3.8.4 Submit the job to one of the queues

To start a multi-CPU job in the queue you need to add (yet another) option to the list of `qsub` options. The option is `-pe smp 5`, assuming you request 5 CPUs. Note that this number must match the number of CPUs you tell your program to use.

For the GATK example the complete command line for a 4 CPU job, assuming you are not using a submission script, would be

```
qsub -cwd -b y -pe smp 4 java -jar /path/to/GenomeAnalysisTK.jar \
          -l INFO \
          -R humref.fasta \
          -T UnifiedGenotyper \
          -I your.bam \
          -o my.vcf \
          --output_mode EMIT_ALL_SITES \
          -nct 4
```

4 Server specifications

The servers in the epib-genstat cluster have the following specifications:

machine	CPU cores	CPU specs	RAM	disk space
`epib-genstat4`	16	2x 8-core Intel E5-2640 v3 @ 2.60GHz	256 GB	~ 135 TB
`genstat-node01`	16	2x 8-core Intel E5-2670 @ 2.60GHz	128 GB	-
`genstat-node02`	16	2x 8-core Intel E5-2670 @ 2.60GHz	128 GB	-

All machines are connected are connected using a 40 Gb/s Infiniband network connection. The two compute nodes hardly have any storage. All files a shared over the network by `epib-genstat4`.

The two nodes each have two nVidia Tesla GPUs connected to them for those who want to use graphics processors for their calculations.

5 Other computing facilities at our disposal

For massive analyses, e.g. computations that can be split into parallel jobs (e.g. analysis of many traits on several populations), but also analyses the have to run for a long time, the Lisa computer cluster at SURFsara in Amsterdam can be used.

Another option is Calligo, the High Performance Computing Cloud system (also at SARA). This is more an option for machines that either don't have to run 24/7 and/or for groups that want to have a Linux or Windows server but don't want to have to deal with the hardware. Note that in the cloud you have to install, configure and manage everything yourself (i.e installing the OS and software, keeping them up to date, etc.).

Please contact Najaf Amin for more information about these options.