# Table Of Contents

1 LiDO3 - first contact 5
   1.1 Scope .............................................. 5
   1.2 Non-scope ........................................ 5

2 Prerequisites 6
   2.1 How do I get / extend a user account? .................. 6
      2.1.1 Application .................................. 6
      2.1.2 Usage ........................................ 9
      2.1.3 Publications ................................... 9
   2.2 SSH Key ............................................ 10
      2.2.1 Create SSH Key on Unix ......................... 10
      2.2.2 Changing your public key ......................... 11
      2.2.3 Create SSH Key on Windows .................... 11

3 Working with LiDO3 14
   3.1 Basic workflow ..................................... 14
   3.2 Connect ........................................... 16
      3.2.1 Unix ......................................... 16
      3.2.2 Windows ....................................... 16
   3.3 Environment ....................................... 20
      3.3.1 Filesystems .................................. 20
         3.3.1.1 /home and /work file systems ............... 20
         3.3.1.2 Read-only /home directory on compute nodes 22
         3.3.1.3 /scratch file system ........................ 23
      3.3.2 Working with the Linux shell .................... 24
      3.3.3 Working with files ............................. 24
      3.3.4 Working with files on your local Windows workstation 24
   3.4 Resource management ................................ 25
      3.4.1 Partition ....................................... 27
      3.4.2 Working with partitions ......................... 28
3.4.2.1 `srun` - interactive execution and jobsteps .............................. 30
3.4.2.2 `sbatch` - Submit a job script ............................................. 32
3.4.2.3 `salloc` - Allocate nodes ..................................................... 34
3.4.2.4 `scontrol`, `squeue` - Query Job status .................................. 35
3.4.2.5 `scancel` - Cancel a queued job ........................................... 36

3.4.3 Constraints on node-features ................................................. 37
3.4.4 Generic Resource (GRES) - request a GPU .................................. 40

3.4.5 Memory management ............................................................... 43
3.4.5.1 Memory limit per node ......................................................... 43
3.4.5.2 Memory limit per cpu .......................................................... 44

3.4.6 Utilize complete nodes ............................................................ 46
3.4.7 Slurm statements ................................................................. 46

3.5 Slurm for Torque/PBS users ....................................................... 48
3.5.1 Common user commands ......................................................... 48
3.5.2 User command for job control .................................................. 49
3.5.3 List of job states ................................................................. 51
3.5.4 Format options for slurm commands ......................................... 51
3.5.5 Job variables ................................................................. 52

3.6 Modules ......................................................................................... 52
3.6.1 Loaded modules ................................................................. 53
3.6.2 Available modules ............................................................. 53
3.6.3 Load a module ............................................................... 53
3.6.4 Unload a module ............................................................. 53
3.6.5 Modules in job scripts .......................................................... 54

3.7 Compiler modules ........................................................................ 54

3.8 Examples ....................................................................................... 56
3.8.1 Basic slurm script example ...................................................... 56
3.8.2 Example using multiple GPU nodes ........................................... 56
3.8.3 Common software example: R .................................................. 57
3.8.4 Common software example: Matlab .......................................... 60
3.8.5 Third-party node usage example ................................................. 61
3.8.6 Have a job automatically clean up when risking to exceed the configured walltime ............................................. 62
3.8.7 Example for job steps ............................................................. 64

3.9 System overview ........................................................................... 66

3.10 Dictionary ...................................................................................... 67
3.10.1 Walltime ................................................................................. 67
3.10.2 Backfilling ............................................................................... 67

3.11 Get support .................................................................................... 69

3.12 Appendix ......................................................................................... 70
3.12.1 Symbolic links for non-writable home directory . . . . . . . . . . 70
3.12.2 Picture credits . . . . . . . . . . . . . . . . . . . . . . . . . . . 71
Chapter 1

LiDO3 - first contact

1.1 Scope

This document intends to guide you through the first steps on LiDO3, TU Dortmund’s high performance cluster (HPC): to get access to the system and a job running.

We renounce the explicit mention of the female form and hope that this omission allows fluent reading of the instructions.

1.2 Non-scope

Programming, especially parallel programming and the usage of libraries like MPI¹ is not subject of this document. Neither is it a guide for structuring workload to scale on a HPC environment.

¹https://en.wikipedia.org/wiki/Message_Passing_Interface
Chapter 2

Prerequisites

2.1 How do I get / extend a user account?

2.1.1 Application

Applications must be submitted by permanent employees of the Technische Universität Dortmund or the institution of the applicant.

In most cases, employees of the Technische Universität Dortmund can use the service portal¹ to submit an application online.

In this application form, it is mandatory to provide information about association (e.g. faculty and chair), the intended purpose of LiDO usage, termination date of LiDO usage, name and email address of your approver (your supervising professor) and your public SSH key which you are supposed to have generated before submitting the form. For generating your public and private key pair see page 10.

The LiDO application form is not visible for students nor for external PhD students. If a student is supposed to use the LiDO cluster, his supervisor must submit the form and is encouraged to use the text box labelled “Additional information” to add the remark that the account application is actually for a student, supplying additionally the student’s real name, login name, email address and phone number.

¹https://service.tu-dortmund.de/group/intra/lido3neuantrag
The application form to manage an existing LiDO3 account (apply for extension of account validity, change SSH Public Key etc.) is not yet available.

Please resort to the existing application form “Account application LiDO3” if you intend to extend your LiDO3 account. Add the words “extension of existing LiDO3 account” in the text box labelled “Additional information”. This helps to avoid confusion.

Figure 2.1: Use “Neuantrag” to submit the application.
Figure 2.2: Insert your generated public key into the “SSH Public Key” field to submit your public key.

If the project is funded by the Fachhochschule Dortmund or UA Ruhr within the framework of a cooperation with the Technische Universität Dortmund, you have to apply directly at the Service Desk with the following filled out form\(^2\). Please note that this application can only be used by professors for their own projects, doctoral or post-doctoral research.

\(^2\)https://www.itmc.tu-dortmund.de/cms/Medienpool/pdfs/Hochleistungsrechnen/hauptantrag-lido-externe.pdf
If you have questions regarding the account or are an applicant from a university outside Dortmund, please contact

Maria Pefferkuch
Zentrale und Serviceeinrichtungen- IT und Medien Centrum (ITMC)
Otto-Hahn-Str. 12, room E.036
Phone: +49 231 / 755 - 2367

Users from the UA Ruhr, please contact the Service Desk by telephone to make an appointment.

2.1.2 Usage

After you have received an email stating that your account has been created, log in with your unimail address and the private key of the key pair you provided us in the application form using a SSH\(^3\)-client that connects you via the SSH protocol to one of the gateway servers:

- gw01.lido.tu-dortmund.de
- gw02.lido.tu-dortmund.de

If you used a passphrase to protect your private SSH key, the SSH client (or an authentication agent like pageant) will prompt you for that passphrase. If you are prompted, however, for a password something is either wrong in your setup or the private SSH key does not match the public SSH key stored on LiDO3.

2.1.3 Publications

Please drop us a short e-mail with a citation reference for publications for which LiDO3 has been used. We need this information in our reports to DFG (German Research Foundation) that funded the LiDO3 acquisition.

\(^3\)http://en.wikipedia.org/wiki/Secure_Shell
2.2 SSH Key

SSH keys are used to identify yourself to a computer using public-key cryptography instead of a password. On one hand, this is done for security reasons – a SSH key is much harder to crack than a password, if at all – and on the other hand for user comfort.

The internet is full of tutorials that show how to create and use a SSH key, so we will just refer to one example for Linux users and one for Windows/PuTTY users.

💡 The use of SSH-Keys is mandatory. You cannot log into LiDO3 with a username and password. In case you are prompted for a password other than your SSH key passphrase when you try to log in to either one of the gateway servers, something went wrong with your SSH keys: either the public key entered in the application form got scrambled or the private key does not match the public key (any more).

2.2.1 Create SSH Key on Unix

Open a shell and enter

```
$ ssh-keygen -t rsa -b 4096 -C "your.email@tu-dortmund.de"
```

If you already have other SSH-keys, you can change the filename here, otherwise just use the default.

```
Generating public/private rsa key pair.
Enter file in which to save the key (/home/<username>/.ssh/id_rsa):
```

When prompted, enter a secure passphrase to protect your – private – SSH key.

```
Enter passphrase (empty for no passphrase):
[Type a passphrase]
Enter same passphrase again:
```

---

4https://en.wikipedia.org/wiki/Public-key_cryptography
7If someone gains access to your computer, they also gain access to every system that uses your SSH-key-pair. To add an extra layer of security, you should add a passphrase to your SSH key.
Copy and paste only the public key (typically marked by the .pub file extension) into the user application form (see page 8) after the successful creation.

2.2.2 Changing your public key

Unlike on other Unix systems your SSH key will not be visible in ~/.ssh/authorized_keys on LiDO3. Thus any changes to your key must be advertised in the service portal.

The provision of the form for managing an existing LiDO3 account (extensions, SSH public key exchange, etc.) is delayed indefinitely. In the meantime, please misemploy the form "Account application LiDO3" if you wish to extend your existing LiDO3 account and add the keywords "Account renewal instead of new application" in the field "Additional information" to avoid misunderstandings.

2.2.3 Create SSH Key on Windows

Use puttygen from the PuTTY Software Suite. Click on Generate. To build the key pair, it is important to move your mouse in random directions.

https://service.tu-dortmund.de/group/intra/lido
https://www.chiark.greenend.org.uk/~sgtatham/putty/
Figure 2.3: Random movements of the mousepointer are used in order to create the key pair.

After that you have to enter a password which is later used to protect your private key. Save your private and your public key on the Windows machine. PuTTY uses its own file format (suffix .ppk) which can not be used on Linux directly. Therefore copy and paste only the public key into the user application form (see figure 2.4 on page 8).
Figure 2.4: Save the private and the public key. Copy and paste the public key (marked in yellow) to the user application form.

Unlike on other Unix systems your SSH key will not be visible in ~/.ssh/authorized_keys on LiDO3. Thus any changes to your key must be advertised in the service portal\[10\].

The provision of the form for managing an existing LiDO3 account (extensions, SSH public key exchange, etc.) is delayed indefinitely. In the meantime, please misemploy the form "Account application LiDO3" if you wish to extend your existing LiDO3 account and add the keywords "Account renewal instead of new application" in the field "Additional information" to avoid misunderstandings.

Unlike on other Unix systems your SSH key will not be visible in ~/.ssh/authorized_keys on LiDO3. Thus any changes to your key must be advertised in the service portal\[11\].

The provision of the form for managing an existing LiDO3 account (extensions, SSH public key exchange, etc.) is delayed indefinitely. In the meantime, please misemploy the form "Account application LiDO3" if you wish to change your SSH public key and add the keywords "SSH public key update instead of new application" in the field "Additional information" to avoid misunderstandings.

\[10\]https://service.tu-dortmund.de/group/intra/lido3verwaltung
\[11\]https://service.tu-dortmund.de/group/intra/lido
Chapter 3

Working with LiDO3

3.1 Basic workflow

The basic workflow is

- Connect to one of the gateway servers via SSH\(^1\).
- Create a job.
- Enqueue the job into the job queue.
- One or more nodes calculate the result.
- Receive the result on a gateway server.

\(^1\)http://en.wikipedia.org/wiki/Secure_Shell
As long as your operating systems has an up-to-date version of a SSH\(^2\)-client, you can connect to one of the gateway servers:

- gw01.lido.tu-dortmund.de
- gw02.lido.tu-dortmund.de

Both gateways have the same software stack and allow access to all jobs and files, it does not matter which one you choose. If one gateway is down due to maintenance or failure, there is still a second one.

\(^2\)http://en.wikipedia.org/wiki/Secure_Shell
3.2 Connect

3.2.1 Unix

On any Unix-style operating system you should be able to connect from a terminal via

```
ssh -i <private ssh-key> <account_name>@<gateway_name>
```

replacing `<private ssh-key>` with the path/filename of your private SSH Key (see page 10), `<account_name>` with your LiDO-account-name and `<gateway_name>` with one of the pre-mentioned names of the gateway servers.

If you connect to one of the gateway servers for the first time, you will be asked if the key fingerprint from this server is correct. This is done for security reasons to make sure that this really is one of the servers you want to connect to.

```
$ ssh-keygen -lf <(ssh-keyscan gw01.lido.tu-dortmund.de)
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw01.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
256 SHA256:SxL75DVFyNKVbSMk81M/fPty5qcPtWaSM9iHHe90ETU gw01.lido.tu-dortmund.de
   (ECDSA)
2048 SHA256:rG0Cmye6DibyWvajHcma6vnwsVfYATy1JM/O200Ns gw01.lido.tu-dortmund.de
   (RSA)
256 SHA256:1UQLD0VY/pTVpsSPwuUwvHA8jm/tNiGJ+GbaHF9sBpo gw01.lido.tu-dortmund.de
   (ED25519)

$ ssh-keygen -lf <(ssh-keyscan gw02.lido.tu-dortmund.de)
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
# gw02.lido.tu-dortmund.de:22 SSH-2.0-OpenSSH_7.4
256 SHA256:rG0Cmye6DibyWvajHcma6vnwsVfYATy1JM/O200Ns gw02.lido.tu-dortmund.de
   (RSA)
256 SHA256:1UQLD0VY/pTVpsSPwuUwvHA8jm/tNiGJ+GbaHF9sBpo gw02.lido.tu-dortmund.de
   (ED25519)
```

3.2.2 Windows

Older versions of Microsoft Windows come with no built-in SSH\(^3\)-client-software, so you have to download and install a third-party tool. Having Windows 10, you can install a Linux subsystem\(^4\) and use that to start a connection.


PuTTY\(^5\) is a well known and sufficient ssh-client and it’s free. MobaXterm\(^6\) is a fork based on PuTTY with X11 server, tabbed SSH client and network tools.

Since Windows users may not be used to connect to other computers via SSH\(^7\), we will describe it more detail here — assuming you use PuTTY as client software. Of course you can use other SSH client software if that suits you better.

Replace <gateway_name> at Connection→SSH with one of the pre-mentioned names of the gateway servers, enter the path to your SSH Key (see page 10) at Connection→SSH→Auth and click on Open.

![PuTTY Configuration](image)

**Figure 3.2:** Enter the gateway name, path so SSH-Key and click on Open.

---

\(^5\)http://www.chiark.greenend.org.uk/~sgtatham/putty/  
\(^6\)http://mobaxterm.mobatek.net/  
\(^7\)http://en.wikipedia.org/wiki/Secure_Shell
If you connect to one of the gateway servers for the first time, you will be asked if the key fingerprint from this server is correct. This is done for security reasons to make sure that this really is one of the servers you want to connect to.

Accept the key with a click on Yes.

Replace <account_name> with your LiDO-account-name and press the [Enter] key.
Figure 3.4: Enter your account name and press [Enter].

Enter the password for your private key and press the [Enter] key. Now you are logged in, welcome to the world of high performance computing.
You end the session with the command `exit`.

### 3.3 Environment

#### 3.3.1 Filesystems

##### 3.3.1.1 /home and /work file systems

On LiDO3 there are two file systems available on both gateway servers and all compute nodes:

- `/home`
- `/work`

On both of them user quotas are enabled. Available disk space quota and current quota usage is automatically shown on login.

We would like to point your attention to the different properties of the two file systems `/home` and `/work` available on LiDO3:
/home has a quota of 32 GiB for user data, but its content is backed up on tape such that in case of a file system problem the /home file system and its data can be restored. On login, the current quota usage is displayed. It can be manually queried by running

```
   df -h $HOME
```

/home is provided by two redundant NFS servers and is hence a network file system, but not a parallel file system.

/home is **read-only**, i.e. **write-protected** on the compute nodes! If the software you execute on the compute nodes needs to write to the home directory, you have two options:

- **Redefine** HOME before invoking the command. Bash users can prepend the actual command with `HOME=/work/${USER}`.
- **Create symbolic links** in your home directory to an alternate writable location. See on page 70 for some examples of already existing software.

/work has different characteristics: it has a default quota of 1 TiB for user data, but the files are **not** saved externally - due to financial limitations (human resources, backup capacity and intra-university network bandwidth). It is provided by several redundant file servers, uses the parallel file system BeeGFS and has a total size of 1.28 PiB. /work can be read from and written to on both gateways and all compute nodes. The **link** in your home directory called “nobackup” leads to the /work/${USER} directory.

The quota can be manually queried by running

```
   beegfs-ctl --getquota --uid $USER
```

In case of a severe file system problem the data might get **LOST** completely.
This is no mere theoretical risk, on its predecessor cluster LiDOnG it has happened multiple times. Please keep this in mind and backup important files in /work yourself at regular intervals. If it is technically possible when an emergency situation arises, we will grant a two days window to make backups. Don’t firmly rely on this chance, though, and keep in mind that when storing terabytes of data on LiDOnG your network bandwidth might not suffice to transfer all your data from LiDOnG within two days.

```bash
cd /home/<user>/nobackup/<my-app>
sbatch myjob.sh
```

Since it is in the nature of a high performance cluster that many nodes, cores and processes access data simultaneously on those file systems, the cluster uses a parallel distributed file system named BeeGFS\(^8\).

![Icon]

While being a specialist for parallel access patterns, there is also a caveat: working with many small files and accessing the directory structures (in doing any equivalent of `ls`) stresses the parallel file system. **Do not do that!**

### 3.3.1.2 Read-only `/home` directory on compute nodes

**X11** To be able to use X Window System software on compute nodes, the X11 magic cookie needs to be written to/updated in a file named `.Xauthority`. Typically, this file is stored in a user’s home directory. To work around the fact that the `/home` directory can not be written to on the compute nodes, a workaround has been set up system-wide, the file `/work/${USER}/.Xauthority` is used instead.

**GnuPG** If you plan to use software that uses `gpg` to verify the signature of files, please note that `gpg` tries to create temporary files in `~/.gnupg` while doing so. In order to have `gpg` successfully verify signatures on compute nodes, you need to move the directory `~/.gnupg` to e.g. `/work/${USER}/.gnupg` and set a symbolic link to this new location in your home directory instead:

```bash
test -d ~/.gnupg || mkdir ~/.gnupg
mv ~/.gnupg /work/${USER}/.gnupg
ln -s /work/${USER}/.gnupg ~/
```

\(^8\)https://en.wikipedia.org/wiki/BeeGFS
3.3.1.3 /scratch file system

If you need to do heavy I/O or parallel processing of data in files, consider using the /scratch file system. /scratch is a local file system on each node that can’t be accessed from other machines.

![Diagram of file system]

Figure 3.6: /home and /work can be accessed from any node, /scratch is only a local file system.

The workflow would look something like this:

- Job starts
  - Copy data from /work to /scratch
- Job runs
  - Process data on /scratch
- Job ends
  - Copy data from /scratch to /work

💡 It is a good approach to create a directory in /scratch consisting of your username and job ID as in /scratch/<username>_<job_id>.
3.3.2 Working with the Linux shell

If you have never worked with the Linux Shell Bash before, you can find more than one tutorial in the internet.

3.3.3 Working with files

Working with a Linux Shell and with LiDO means working with textfiles. Here is a list of installed text editors:

- vi
- Emacs
- Gedit

Choose the one that suits your needs.

3.3.4 Working with files on your local Windows workstation

Some of the editors might seem rather strange for Windows users and if desired, one can create and edit the text files locally on the Windows workstation and copy them via SSH to one of the gateway server or vice versa.

There are also some GUI clients for transferring the files back and forth from your Windows machine, e.g. FileZilla and WinSCP. For both programs, the respective websites explain how to set up SSH public key authentication.

Just keep in mind that the newline character is handled differently on Linux and Windows. You want to use a feature like ASCII mode = newline conversion in your SSH client software - if available.

---

9 https://en.wikipedia.org/wiki/Bash_%28Unix_shell%29
11 https://en.wikipedia.org/wiki/Vi
12 https://en.wikipedia.org/wiki/Emacs
13 https://en.wikipedia.org/wiki/Gedit
14 https://en.wikipedia.org/wiki/Graphical_user_interface
15 https://filezilla-project.org/
16 http://winscp.net/
17 https://wiki.filezilla-project.org/Howto
18 https://winscp.net/eng/docs/guide_public_key
19 https://en.wikipedia.org/wiki/Newline
3.4 Resource management

LiDO3 uses the Slurm Workload Manager\textsuperscript{20} to control batch jobs and cluster resources. Slurm takes care of running the users’ jobs on allocated nodes and keeps track of the users’ processes. Processes that are started directly on individual nodes – circumventing the queuing system – are immediately killed without further notification.

Slurm comes with a built-in scheduling system with the purpose of finding and allocating the necessary resources for a user’s job and organizes the usage between different users and jobs taking scheduling policies, dynamic priorities, reservations, and fairshare capabilities into account.

The entities managed by Slurm include:

- **nodes**: are the compute resource in Slurm.
- **partitions**: group nodes into logical – possibly overlapping – sets.
- **jobs**: allocate resources – inside a partition – to a user for a specified amount of time.
- **job steps**: are sets of – possibly parallel – tasks within a job (see page 64).
- **tasks**: The actual running code.

\textsuperscript{20}\url{https://slurm.schedmd.com/}
“The partitions can be considered job queues, each of which has an assortment of constraints such as job size limit, job time limit, users permitted to use it, etc. Priority-ordered jobs are allocated nodes within a partition until the resources (nodes, processors, memory, etc.) within that partition are exhausted. Once a job is assigned a set of nodes, the user is able to initiate parallel work in the form of job steps in any configuration within the allocation. For instance, a single job step may be started that utilizes all nodes allocated to the job, or several job steps may independently use a portion of the allocation.”

— Quoted from Slurm Quick Start User Guide
3.4.1 Partition

There are different partitions available on the LiDO3 cluster.

Table 3.1: Standard partitions

<table>
<thead>
<tr>
<th>Queue</th>
<th>max. walltime</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>short</td>
<td>02:00:00:00</td>
<td>—</td>
</tr>
<tr>
<td>med</td>
<td>08:00:00:00</td>
<td>—</td>
</tr>
<tr>
<td>long</td>
<td>2-00:00:00:00</td>
<td>—</td>
</tr>
<tr>
<td>ultralong</td>
<td>28-00:00:00:00</td>
<td>no GPU or &quot;non-blocking&quot; nodes</td>
</tr>
<tr>
<td>testpart</td>
<td>02:00:00:00</td>
<td>use when instructed by LiDO3 administrators</td>
</tr>
</tbody>
</table>

Table 3.2: Partitions with faculty hardware

<table>
<thead>
<tr>
<th>Queue</th>
<th>max. walltime</th>
<th>remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>ext_phy_prio</td>
<td>28-00:00:00:00</td>
<td>Xeon Phi “KNL”</td>
</tr>
<tr>
<td>ext_phy_norm</td>
<td>02:00:00:00</td>
<td>Xeon Phi “KNL”</td>
</tr>
<tr>
<td>ext_iom_prio</td>
<td>28-00:00:00:00</td>
<td>members group _iom only</td>
</tr>
<tr>
<td>ext_iom_norm</td>
<td>02:00:00:00</td>
<td>—</td>
</tr>
<tr>
<td>ext_trr188</td>
<td>28-00:00:00:00</td>
<td>members group trr188 only</td>
</tr>
<tr>
<td>ext_vwl_prio</td>
<td>28-00:00:00:00</td>
<td>members group _vwl only</td>
</tr>
<tr>
<td>ext_vwl_norm</td>
<td>02:00:00:00</td>
<td>—</td>
</tr>
</tbody>
</table>

The command `sinfo` provides an overview over the partitions.
3.4.2 Working with partitions

Working with jobs is done by using *Slurm commands* that describe the resource characteristics of the job, e.g. number of nodes, processor cores needed and *Walltime*. This can be done interactively from the shell or in a *job script*.

To start a job in *Slurm*, it must be put into a *Partition*. This is done with one of these three commands:

**srun**  
*“Run a parallel job on cluster managed by Slurm. If necessary, srun will first create a resource allocation in which to run the parallel job.”*  
— Quoted from the *srun* manpage.

**srun** is typically used to start *jobsteps* inside a shell script that was launched with *sbatch*. This way the code for preparing the job and clean-up afterwards can run even if a job is terminated.

**sbatch**  
*“sbatch submits a batch script to Slurm. The batch script may be given to sbatch through a file name on the command line, or if no file name is specified, sbatch will read in a script from standard input. The batch script may contain options preceded with “#SBATCH” before any executable commands in the script.”*
sbatch exits immediately after the script is successfully transferred to the Slurm controller and assigned a Slurm job ID. The batch script is not necessarily granted resources immediately, it may sit in the queue of pending jobs for some time before its required resources become available. By default both standard output and standard error are directed to a file of the name "slurm-%j.out", where the "%j" is replaced with the job allocation number. The file will be generated on the first node of the job allocation. Other than the batch script itself, Slurm does no movement of user files.

When the job allocation is finally granted for the batch script, Slurm runs a single copy of the batch script on the first node in the set of allocated nodes.

— Quoted from the sbatch manpage.

salloc “salloc - Obtain a Slurm job allocation (a set of nodes), execute a command, and then release the allocation when the command is finished.”

— Quoted from the salloc manpage.

Partitions with long configured walltimes are popular from the users view but on the other hand they are somehow an unloved child from the cluster administrators perspective.

- When you as a user put a job inside a partition with a long configured walltime, chances are high that you have to wait quite a long time before your job gets even started. Statistics teaches us that the average waiting time is half of the maximum configured walltime per partition.

- The same goes for maintenance windows. We have to drain those partitions (i.e. starting of jobs is prohibited, submissions of jobs is still possible) very early to make sure that not too many jobs are still running when we shut down the cluster. All jobs still running need to be canceled when the maintenance starts. Closing those partitions early can have a negative impact on the utilization of the cluster: with long running jobs ending one by one and no new long running jobs being allowed to start, compute nodes may become idle if not enough requests are made for partitions with shorter maximum walltimes that are still open.

- In case of an emergency shutdown of the cluster all currently running jobs will get canceled. This, obviously, translates to data loss for all those jobs. In a worst case scenario all calculated data from long running jobs gets lost...
maybe just a few minutes before its planned end of runtime.

This is no mere theoretical risk, unscheduled emergency downtimes have happened before.

So, please consider to use checkpointing in your jobs and in your code in a way that enables you to restart a canceled job and resume the work after the last checkpoint.

And while you are at it, think about breaking your long running job up in to smaller parts that can run one after another in a partition with a shorter maximum walltime. Best aim for under two hours, so your job(s) will fit in the short partition.

LiDO4 will – like most HPC clusters – probably not provide partitions with a walltime greater than 24 hours.

### 3.4.2.1 `srun` - interactive execution and jobsteps

S
Slurm
lurm offers the possibility to execute jobs interactively. Execution of `srun` with the command line option `--pty bash` results in Slurm reserving the requested node – by using `salloc` under the hood (see page 34) – and starts `bash` on that node with a login prompt due to the `--pty` option and waits for its execution. Since no partition was given, the default short ist used. The user can then start his program from that interactive shell.

**Example session:**

```bash
[<username>@gw01 ~]$ srun --pty bash
[<username>@cstd01-214 ~]$ echo $SLURM_TASK_PID
163545
[<username>@cstd01-214 ~]$ exit
[<username>@gw01 ~]$ exit
```

As soon as the walltime is exceeded, the shell is automatically terminated!

Other options to `srun` include number of nodes, Walltime etc., see also section Slurm statements.

**Example session with 4 nodes and 3 tasks per node:**

```bash
[<username>@gw01 ~]$ srun --nodes=4 --ntasks-per-node=3 --pty bash
[<username>@cstd01-214 ~]$ echo $SLURM_TASK_PID
166178
[<username>@cstd01-214 ~]$ exit
exit
[<username>@gw01 ~]$ exit
```
If the \texttt{--pty} option is omitted, no login prompt will be given and any input will get run \texttt{12 = (\texttt{--nodes=4} \times \texttt{--ntasks-per-node=3}) times.}

Example session with multiple times:


\begin{verbatim}
[<username>@gw01 ~]$ srun --nodes=4 --ntasks-per-node=3 bash
  # there is no prompt, so enter blindly:
  echo $SLURM_TASK_PID
  121395
  104316
  167463
  121396
  104317
  121397
  104318
  167464
  105575
  167465
  105576
  exit
[<username>@gw01 ~]$ \end{verbatim}

The following shell script \texttt{demoscript.sh} is used to start a job non-interactive:

\begin{verbatim}
#!/bin/bash -l
  echo "START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID $SLURM_TASK_PID) on $SLURMD_NODENAME"
sleep 60
  echo "STOP on $SLURMD_NODENAME"
\end{verbatim}

Example session:


\begin{verbatim}
[<username>@gw01 ~]$ srun --nodes=2 --tasks-per-node=4 demoscript.sh
  START SLURM_JOB_ID 10894 (SLURM_TASK_PID 173171) on cstd01-214
  START SLURM_JOB_ID 10894 (SLURM_TASK_PID 126888) on cstd01-215
  START SLURM_JOB_ID 10894 (SLURM_TASK_PID 173173) on cstd01-214
  START SLURM_JOB_ID 10894 (SLURM_TASK_PID 126889) on cstd01-215
  START SLURM_JOB_ID 10894 (SLURM_TASK_PID 173174) on cstd01-214
  START SLURM_JOB_ID 10894 (SLURM_TASK_PID 126891) on cstd01-215
  START SLURM_JOB_ID 10894 (SLURM_TASK_PID 173172) on cstd01-214
  START SLURM_JOB_ID 10894 (SLURM_TASK_PID 126890) on cstd01-215
  STOP on cstd01-214
  STOP on cstd01-214
  STOP on cstd01-215
  STOP on cstd01-215
  STOP on cstd01-214
  STOP on cstd01-215
  STOP on cstd01-214
  STOP on cstd01-215
  STOP on cstd01-215
[<username>@gw01 ~]$ \end{verbatim}
Note that the execution with `srun` blocks your session. Only after `demoscript.sh` is run
\[8 = (-\text{nodes} = 2) \times (-\text{ntasks per node} = 4)\] times, you return to your login prompt.

If you close your SSH session, all jobs started by `srun` – directly from your shell – will be terminated!

### 3.4.2.2 `sbatch` - Submit a job script

If don’t want to submit your jobs details by hand and stay in front of the terminal everytime, you can wrap the needed information into a job script for later execution. A job script is basically a shell script that contains Slurm statements in the header section. The rest of the script is code that should be executed AKA the job itself.

```bash
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=3
#SBATCH --time=0:30
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/job.out.txt
...some code...
```

A script can be submitted to the batch system with the command `sbatch`, followed by `<SCRIPT_NAME>`. By using `salloc` under the hood (see page 34) the requested nodes are reserved and used for job execution.

`sbatch my_submit_script.sh`

**Example of a job script:**

```bash
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=4
#SBATCH --ntasks-per-node=3
#SBATCH --time=0:30
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/job.out.txt
echo "sbatch: START SLURM JOB ID $SLURM_JOB_ID (SLURM_TASK_PID $SLURM_TASK_PID) on $SLURM_NODENAME"
echo "sbatch: SLURM_JOB_NODELIST $SLURM_JOB_NODELIST"
echo "sbatch: SLURM_JOB_ACCOUNT $SLURM_JOB_ACCOUNT"
srun /home/<username>/workerscript.sh &
```
wait
echo "sbatch: STOP"

The job script spawns 12 job steps, each calling `workerscript.sh`:

```bash
#!/bin/bash -l
echo "worker ($SLURMD_NODENAME): start"
executing SLURM_JOB_ID $SLURM_JOB_ID 
(SLURM_TASK_PID $SLURM_TASK_PID) 
on $SLURMD_NODENAME*
sleep 10
echo "worker ($SLURMD_NODENAME): stop"
```

Executing the job script:

```
[<username>@gw01 ~]$ sbatch my_submit_script.sh
Submitted batch job 11283

# waiting 10 seconds

[<username>@gw01 ~]$ cat /work/<username>/demo.out.txt
sbatch: START SLURM_JOB_ID 37170 (SLURM_TASK_PID 68044) on cstd01-205
sbatch: SLURM_JOB_NODELIST cstd01-[205-208]
sbatch: SLURM_JOB_ACCOUNT itmc
worker (cstd01-206): start
worker (cstd01-208): start
worker (cstd01-205): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 68077) on cstd01-205
worker (cstd01-207): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 66025) on cstd01-206
worker (cstd01-208): start
worker (cstd01-205): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 68078) on cstd01-205
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 72998) on cstd01-207
worker (cstd01-206): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 82054) on cstd01-208
worker (cstd01-205): start
worker (cstd01-207): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 66026) on cstd01-206
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 82053) on cstd01-208
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 68079) on cstd01-205
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 72999) on cstd01-207
worker (cstd01-206): start
worker (cstd01-208): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 82055) on cstd01-208
worker (cstd01-207): start
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 66027) on cstd01-206
executing SLURM_JOB_ID 37170 (SLURM_TASK_PID 73000) on cstd01-207
worker (cstd01-208): stop
worker (cstd01-205): stop
worker (cstd01-207): stop
worker (cstd01-208): stop
worker (cstd01-206): stop
worker (cstd01-206): stop
```

wait
echo "sbatch: STOP"
Due to race conditions, the order is not predictable.

If you need to use third party software in your job script that is available via the module system, see section Modules in job scripts on page 54.

If the path to the output file does not exist or cannot be written to (e.g. points outside of /work), the SLURM job will seemingly fail silently (unless mail notification is enabled). One can query the SLURM database explicitly for such failed jobs with sacct --starttime=HH:MM --state=FAILED.

### 3.4.2.3 `salloc` - Allocate nodes

Resources for a job can be allocated in real time with the command `salloc`. Those allocated resources are typically used to spawn a shell and – interactively – execute `srun` commands to launch parallel tasks.

Whereas `srun` uses `salloc` under the hood to acquire the needed resources, using `salloc` as a discrete command enables you to initiate different job steps inside an allocated set of nodes.

To allocate 10 nodes using the `--exclusive` option so no other users will be running jobs on the allocated nodes at the same time, enter

```
[<username>@gw01 ~]$ salloc --nodes=10 --exclusive
salloc: Granted job allocation 14008
salloc: Waiting for resource configuration
salloc: Nodes cstd01-[001-010] are ready for job
```

Now we will start 3 job steps on those 10 allocated nodes:

1. **Using 2 nodes** `--nodes=2` starting with the first node `--relative=0` of the allocated range.
2. **Using 4 nodes** `--nodes=4` starting with the third node `--relative=2` of the allocated range.
3. using 2 nodes (\texttt{--nodes=4}) starting with the seventh node (\texttt{--relative=6}) of the allocated range.

```
[<username>@gw01 ~]$ srun --nodes=2 --relative=0 --jobid=14008 /usr/bin/sleep 300s
[<username>@gw01 ~]$ srun --nodes=4 --relative=2 --jobid=14008 /usr/bin/sleep 300s
[<username>@gw01 ~]$ srun --nodes=4 --relative=6 --jobid=14008 /usr/bin/sleep 300s
```

Since no \texttt{--time} option was used with \texttt{salloc}, the allocation will last as long as the timelimit of the partition. Further job steps can be initiated during that timespan.

Allocations can also be used to start a session with the X Window System.

```
[<username>@gw01 ~]$ salloc --nodes=1 --exclusive --constraint=cstd01
salloc: Granted job allocation 14037
salloc: Waiting for resource configuration
salloc: Nodes cstd01-003 are ready for job
[<username>@gw01 ~]$ ssh -X cstd01-003
Warning: Permanently added 'cstd01-10.10.3.3' (ECDSA) to the list of known hosts.
[<username>@cstd01-003 ~]$ start-my-x-programm
[<username>@cgpu01-003 ~]$ exit
logout
Connection to cgpu01-003 closed.
[<username>@gw01 ~]$ scancel
[<username>@gw01 ~]$ salloc: Job allocation 14037 has been revoked.
```

3.4.2.4 \texttt{scontrol, squeue} - Query Job status

The status of each \texttt{Slurm} job can be queried with \texttt{scontrol show job <job_id>} and \texttt{squeue}.

```
[<username>@gw01 ~]$ scontrol show job 11283
JobId=11283 JobName=demoscript
UserId=<username>(<uid>) GroupId=<username>(<gid>) MCS_label=N/A
Priority=21149 Nice=0 Account=itmc QoS=normal
JobState=RUNNING Reason=None Dependency=(null)
RunTime=00:00:47 TimeLimit=00:02:00 TimeMin=N/A
StartTime=2017-08-11T14:20:13 EndTime=2017-08-11T14:22:13 Deadline=N/A
PreemptTime=None SuspendTime=None SecsPreSuspend=0
Partition=short AllocNode:Sid=gw01:60481
ReqNodeList=(null) ExcNodeList=(null)
NodeList=cstd01-[001-004]
BatchHost=cstd01-001
NumNodes=4 NumCPUs=12 NumTasks=12 CPUs/Task=1 ReqB:S:C:T=0:0:*:*
TRES=cpu=12,mem=1200M,node=4
Socks/Node=* NtasksPerN:B:S:C=3:0:*:* CoreSpec=* MinCPUsNode=3 MinMemoryCPU=100M MinTmpDiskNode=0
Features=(null) DelayBoot=00:00:00
Gres=(null) Reservation=(null)
```
OverSubscribe=OK Contiguous=0 Licenses=(null) Network=(null)
Command=/home/<username>/my_submit_script.sh
WorkDir=/home/<username>
StdErr=/work/<username>/demo.out.txt
StdIn=/dev/null
StdOut=/work/<username>/demo.out.txt
Power=

and `squeue` respectively.

### Example session to get all own jobs:

```bash
[<username>@gw01 ~]$ squeue -u $USER

JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
14004 short demoscri <username> R 0:03 2 cgpu01-[002-003]
13977 short bash <username> R 9:36 1 cgpu01-001
13978 med glidein <username> R 8:08 1 cstd01-021
```

### Example session to get information for a specific job:

```bash
[<username>@gw01 ~]$ squeue --jobs=14005

JOBID PARTITION NAME USER ST TIME NODES NODELIST(REASON)
14005 short demoscri <username> R 0:04 2 cgpu01-[002-003]
```

### Example session to get information for a specific job including job steps:

```bash
[<username>@gw01 ~]$ squeue --job=14008 --steps

STEPID NAME PARTITION USER TIME NODELIST
14008.0 sleep short <username> 0:35 cstd01-[001-002]
14008.1 sleep short <username> 0:23 cstd01-[003-006]
14008.2 sleep short <username> 0:13 cstd01-[007-010]
14008.Extern extern short <username> 2:09 cstd01-[001-010]
```

### 3.4.2.5 `scancel` - Cancel a queued job

A `Slurm` job can be removed from the job queue via `scancel <job_id>`.

```bash
[<username>@gw01 ~]$ sbatch my_submit_script.sh
Submitted batch job 11284
[<username>@gw01 ~]$ scancel 11284
[<username>@gw01 ~]$ scontrol show job 11284

UserID=<username> (<uid>) GroupId=<username> (<gid>) MCS_label=N/A
Priority=21158 Nice=0 Account=itmc QOS=normal
```
3.4.3 Constraints on node-features

The LiDO3-Team has assigned so-called features to the different nodes in the LiDO3-cluster. Those features can specifically requested with the `--constraint` parameter of the `srun`, `sbatch` and `salloc` commands.

Table 3.3: List of features.

<table>
<thead>
<tr>
<th>Nodelist</th>
<th>Features</th>
<th>CPU Type</th>
<th>max. cores</th>
<th>max. memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>cgpu01-[001-020]</td>
<td>all public</td>
<td>2 × Intel® Xeon E5 2640v4</td>
<td>20</td>
<td>64 GB</td>
</tr>
<tr>
<td></td>
<td>cgpu01</td>
<td>2.4 GHz, L3 cache 25 MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>xeon_e52640v4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>gpu</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tesla_k40</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ib_1to3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>xeon_e52690v4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>gpu</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tesla_p100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ib_1to3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_mem_per_node=64300</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cgpu02-[001-002]</td>
<td>all public</td>
<td>2 × Intel® Xeon E5 2690v4</td>
<td>28</td>
<td>256 GB</td>
</tr>
<tr>
<td></td>
<td>cgpu02</td>
<td>2.4 GHz, L3 cache 25 MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>xeon_e52690v4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>p100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>gpu</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>tesla_p100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ib_1to3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_mem_per_node=257800</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cknl01-[001-003]</td>
<td>all private</td>
<td>1 × Intel® Xeon Phi “KNL”7210</td>
<td>64</td>
<td>192 GB</td>
</tr>
<tr>
<td></td>
<td>cknl01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>xeon_phi7210</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ib_1to3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_mem_per_node=209400</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cquad01-[001-028]</td>
<td>all public</td>
<td>4 × Intel® Xeon E5 4640v4</td>
<td>48</td>
<td>256 GB</td>
</tr>
<tr>
<td></td>
<td>cquad01</td>
<td>2.1 GHz, L3 cache 30 MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>xeon_e54640v4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ib_1to3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_mem_per_node=257800</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>cquad02-[001-002]</td>
<td>all public</td>
<td>4 × Intel® Xeon E5 4640v4</td>
<td>48</td>
<td>1024 GB</td>
</tr>
<tr>
<td></td>
<td>cquad02</td>
<td>2.1 GHz, L3 cache 30 MB</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>xeon_e54640v4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ib_1to3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>max_mem_per_node=1031900</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

continued on next page...
... continued from previous page

<table>
<thead>
<tr>
<th>Nodelist</th>
<th>Features</th>
<th>CPU Type</th>
<th>max.</th>
<th>max.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>GPU Type</td>
<td>cores</td>
<td>memory</td>
</tr>
<tr>
<td>cquad03-[001-002]</td>
<td>all public cquad03 xeon_gold_6230 ib_1to3</td>
<td>4 × Intel®Xeon Gold 6230 2.1 GHz, L3 cache 28 MB</td>
<td>80</td>
<td>512 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MaxMemPerNode=498952</td>
<td></td>
</tr>
<tr>
<td>cstd01-[001-244]</td>
<td>all public cstd01 xeon_e52640v4 ib_1to3</td>
<td>2 × Intel®Xeon E5 2640v4 2.4 GHz, L3 cache 25 MB</td>
<td>20</td>
<td>64 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MaxMemPerNode=64300</td>
<td></td>
</tr>
<tr>
<td>cstd02-[001-072]</td>
<td>all public cstd02 xeon_e52640v4 ib_1to1 nonblocking_comm</td>
<td>2 × Intel®Xeon E5 2640v4 2.4 GHz, L3 cache 25 MB</td>
<td>20</td>
<td>64 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MaxMemPerNode=64300</td>
<td></td>
</tr>
<tr>
<td>cstd03[001-004]</td>
<td>all public cstd03 xeon_e52690v4 ib_1to3</td>
<td>2 × Intel®Xeon E5 2690v4 2.4 GHz, L3 cache 35 MB</td>
<td>28</td>
<td>256 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MaxMemPerNode=257800</td>
<td></td>
</tr>
<tr>
<td>cstd04[001-004]</td>
<td>all public cstd04 xeon_gold_6134 ib_1to3</td>
<td>2 × Intel®Gold 6134 CPU 2.4 GHz, L3 cache 24 MB</td>
<td>16</td>
<td>192 GB</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MaxMemPerNode=190000</td>
<td></td>
</tr>
</tbody>
</table>

Example session for `srun`

```bash
[<username>@gw01 ~]$ srun --constraint=cstd01 --pty bash
[<username>@cstd01-019 ~]$ echo $SLURM_TASK_PID
166178
[<username>@cstd01-019 ~]$ exit
exit
```

Example session for `sbatch`
As you can see with `man sbatch`, nodes can have **features** assigned to them by the Slurm administrator. Users can specify which of these **features** are required by their job using the constraint option. Only nodes having features matching the job constraints will be used to satisfy the request. Multiple constraints may be specified with **AND**, **OR**, matching **OR**, resource counts, etc. (some operators are not supported on all system types). Supported constraint options include:

**Single Name** Only nodes which have the specified feature will be used. For example, ```bash --constraint="ib_1to1"``` 

**Node Count** A request can specify the number of nodes needed with some feature by appending an asterisk and count after the feature name. For example, ```bash --nodes=16 --constraint=cstd01*4``` indicates that the job requires 16 nodes and that at least four of those nodes must have the feature "cstd01."
AND If only nodes with all of specified features will be used. The ampersand is used for an AND operator. For example, --constraint="xeon_e52640v4&gpu"

OR If only nodes with at least one of specified features will be used. The vertical bar is used for an OR operator. For example, --constraint="xeon_e52640v4|e54640v4"

Matching OR If only one of a set of possible options should be used for all allocated nodes, then use the OR operator and enclose the options within square brackets. For example: "--constraint=[rack1|rack2|rack3|rack4]" might be used to specify that all nodes must be allocated on a single rack of the cluster, but any of those four racks can be used.

Multiple Counts Specific counts of multiple resources may be specified by using the AND operator and enclosing the options within square brackets. For example: "--constraint=[rack1*2&rack2*4]" might be used to specify that two nodes must be allocated from nodes with the feature of "rack1" and four nodes must be allocated from nodes with the feature "rack2".

3.4.4 Generic Resource (GRES) - request a GPU

Reserving a GPU node by using constraints (see page 37) is only one half of the story. Other users may be already using the GPU when your job starts on one of those nodes and they seem too valuable to use them just for CPU-bound tasks.

GPUs are defined as a Generic Resource (short GRES) in Slurm and can be requested with the --gres=gpu:tesla[:count] option which is supported by the salloc, sbatch and srun commands. Where count specifies how many resources are required and has a default value of 1.

- For the 20 nodes with 2 GPU NVIDIA® K40 GPUs each, count has a valid maximum of 2.
- For the 2 nodes with 1 GPU NVIDIA® P100 GPU each, count has a valid maximum of 1.

```bash
# reserve 1 non-exclusive node and both GPUs on it
[<username>@gw01 ~]$ salloc --nodes=1 --gres=gpu:tesla:2
salloc: Granted job allocation 14037
salloc: Waiting for resource configuration
salloc: Nodes cgpu01-003 are ready for job
[<username>@gw01 ~]$ ssh -X cgpu01-003
[<username>@cgpu01-003 ~]$ module load nvidia/cuda/8.0
[<username>@cgpu01-003 ~]$ nvvp -data $WORK -configuration $WORK
```
Figure 3.9: NVIDIA Visual Profiler on a Windows client.

For each job step the environment variable `CUDA_VISIBLE_DEVICES` is set to determine which GPUs are available for its use on each node.

Example script that is executed on GPU nodes:

```bash
#!/bin/bash -l
echo "worker ($SLURMD_NODENAME): start"
echo "executing SLURM_JOB_ID $SLURM_JOB_ID \n(SLURM_TASK_PID $SLURM_TASK_PID, \nCUDA_VISIBLE_DEVICES $CUDA_VISIBLE_DEVICES) \non $SLURMD_NODENAME"
sleep 10
echo "worker ($SLURMD_NODENAME): stop"
```
Example batch script that is used to run workerscript.sh on each GPU node:

```bash
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=4
#SBATCH --exclusive
#SBATCH --gres=gpu:tesla:2
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt

for RELATIVENODE in 0 1 2 3
do
  srun --nodes=1 --relative=${RELATIVENODE} --gres=gpu:tesla:$(($RELATIVENODE%2+1)) --jobid=$SLURM_JOB_ID /home/<username>/workerscript.sh &
done
wait

```

Finally the execution and output:

```
[<username>@gw01 ~]$ sbatch my_submit_script.sh
Submitted batch job 37141
[<username>@gw01 ~]$ cat /work/<username>/demo.out.txt
sbatch: START SLURM_JOB_ID 37171 (SLURM_TASK_PID 31707, CUDA_VISIBLE_DEVICES 0,1) on cgpu01-001
sbatch: SLURM_JOB_NODELIST cgpu01-[001-004]
worker (cgpu01-004): start executing SLURM_JOB_ID 37171 (SLURM_TASK_PID 8348, CUDA_VISIBLE_DEVICES 0,1) on cgpu01-004
worker (cgpu01-002): start executing SLURM_JOB_ID 37171 (SLURM_TASK_PID 13088, CUDA_VISIBLE_DEVICES 0,1) on cgpu01-002
worker (cgpu01-003): start executing SLURM_JOB_ID 37171 (SLURM_TASK_PID 8950, CUDA_VISIBLE_DEVICES 0) on cgpu01-003
worker (cgpu01-001): start executing SLURM_JOB_ID 37171 (SLURM_TASK_PID 31755, CUDA_VISIBLE_DEVICES 0) on cgpu01-001
worker (cgpu01-002): stop
worker (cgpu01-004): stop
worker (cgpu01-001): stop
worker (cgpu01-003): stop
sbatch: STOP
```
Due to race conditions the order is not predictable. Although the option `--gres=gpu:tesla:2` is used, the number of GPUs must be explicitly required. The script alternated between `--gres=gpu:tesla:1` and `--gres=gpu:tesla:2` for every `srun`-call to show that effect.

### 3.4.5 Memory management

#### 3.4.5.1 Memory limit per node

**Slurm** enforces a memory limit on each job. The default is set to `DefMemPerCPU = 512` per node (512MB per core). If a job uses more than that, it is terminated with the *job Exceeded job memory limit* error message. You can set a larger limit by using the `--mem <memory>` option, where `<memory>` is the limit in MB — different units can be specified by using the suffix `[K|M|G|T]`. The maximum upper limit (MaxMemPerNode) can be seen in table 3.3 on page 37.

The larger the data processed by your job, the larger this limit needs to be. The lower you set this limit, the easier it is for the *Slurm scheduler* to find a place for your job to run in the partition.

Example session:

```bash
[<username>@gw01 ~]$ cat my_submit_script.sh
#!/bin/bash -l
#SBATCH --partition=short
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=3
#SBATCH --time=2:00
#SBATCH --mem=500M
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt
#SBATCH --constraint=cstd01
srun echo "START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID $SLURM_TASK_PID) on /uni21AA $SLURMD_NODENAME"
  srun echo "STOP on $SLURMD_NODENAME"
[<username>@gw01 ~]$ sbatch my_submit_script.sh
Submitted batch job 16572
```

If you are not sure what a good setting would be, you can try to determine an appropriate value by starting your job with a short runtime and a relatively large memory limit and then use the `sacct` command to monitor how much your job is actually using or has used.

Example session:
The output is in KB, so divide by 1024 to get a rough idea of what setting to use with \textit{--mem} (since you're defining a hard upper limit, round up that number a little bit). You can tell \texttt{ sacct} to look further back in time by adding a start time with \texttt{--starttime YYYY-MM-DD} if your job ran too far in the past.

The \textit{--mem} options sets the \textit{maximum memory} used on any one node running your job parallel spanning multiple nodes; to get an even distribution of tasks per node, you can use run using the \textit{--ntasks-per-node} option, otherwise the same job could have very different values when run at different times.

A memory size specification of zero is treated as a special case and grants the job access to all of the memory on each node. If multiple nodes with different memory layout are allocated for your job in the LiDO3 cluster, the node with the smallest memory size in the allocation defines the memory limit for each node of the allocation — the same limit will apply to every node.

The \textit{--mem} option and the the \textit{--mem-per-cpu} option are \textit{mutually exclusive}!

### 3.4.5.2 Memory limit per cpu

\textit{Slurm} enforces a memory limit on each job. The default is set to DefMemPerCPU = 512 per node (512MB per core). If a job uses more than that, it is terminated with \textit{job Exceeded job memory limit} error message. You can set a larger limit by using the
The `--mem-per-cpu <memory>` option, where `<memory>` is the limit in MB — different units can be specified by using the suffix [K|M|G|T]. The maximum upper limit (MaxMemPerNode) can be seen in table 3.3 on page 37.

The larger the data processed by your job, the larger this limit needs to be. The lower you set this limit, the easier it is for the Slurm scheduler to find a place for your job to run in the partition.

The number of cores times the memory per core must not exceed the maximum upper limit (MaxMemPerNode).

\[ \text{cpus} \times \text{mem} < \text{MaxMemPerNode} \]

```bash
[<username>@gw01 ~]$ cat my_submit_script.sh
#!/bin/bash
#SBATCH --partition=short
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=3
#SBATCH --time=2:00
#SBATCH --mem-per-cpu=500M
#SBATCH --job-name=demoscript
#SBATCH --output=/work/<username>/demo.out.txt
#SBATCH --constraint=cstd01
srun echo "START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID $SLURM_TASK_PID) on $SLURMD_NODENAME"
for i in {0..2}; do
  srun sleep 30
  srun echo "$SLURM_JOB_ID $SLURM_TASK_PID $SLURMD_NODENAME"
  srun sleep 30
done
srun echo "STOP on $SLURMD_NODENAME"
[<username>@gw01 ~]$ sbatch my_submit_script.sh
Submitted batch job 16571

[<username>@gw01 ~]$ sacct --format MaxRSS --job=16571
MaxRSS
--------
284K
88K
92K
[<username>@gw01 ~]$```

The `--mem-per-cpu` option and the `--mem` option are **mutually exclusive!**
3.4.6 Utilize complete nodes

If a user submits a job, it is very well possible that other jobs will run on the same nodes. To make a reservation for a complete node, use a \texttt{--exclusive} statement.

\begin{verbatim}
# Example reservation for 1 node:
[{username}@gw01 ~]$ salloc --nodes=1 --exclusive
salloc: Granted job allocation 140042
salloc: Waiting for resource configuration
salloc: Node cstd01-017 is ready for job
\end{verbatim}

3.4.7 Slurm statements

Here is a non-exhaustive list of frequently used \textit{Slurm} statements that can be used inside of a \textit{job script} generated with help of \texttt{man sbatch}.

- \texttt{#SBATCH --job-name=<jobname>}
  Specify a name for the job allocation. The specified name will appear along with the job ID number when querying running jobs on the system. The default is the name of the batch script, or just "sbatch" if the script is read on sbatch’s standard input.

- \texttt{#SBATCH --nodes=<minnodes[-maxnodes]>}
  Request that a minimum of \texttt{minnodes} nodes be allocated to this job. A maximum node count may also be specified with \texttt{maxnodes}. If only one number is specified, this is used as both the minimum and maximum node count. The partition’s node limits supersede those of the job. If a job’s node limits are outside of the range permitted for its associated partition, the job will be left in a PENDING state. This permits possible execution at a later time, when the partition limit is changed. If a job node limit exceeds the number of nodes configured in the partition, the job will be rejected. Note that the environment variable \texttt{SLURM_NNODES} will be set to the count of nodes actually allocated to the job. If \texttt{-N} is not specified, the default behavior is to allocate enough nodes to satisfy the requirements of the \texttt{-n} and \texttt{-c} options. The job will be allocated as many nodes as possible within the range specified and without delaying the initiation of the job. The node count specification may include a numeric value followed by a suffix of "k" (multiplies numeric value by 1,024) or "m" (multiplies numeric value by 1,048,576).

- \texttt{#SBATCH --partition=<partition_names>}

Request a specific partition for the resource allocation. If not specified, the default behavior is to allow the slurm controller to select the default partition as designated by the system administrator. If the job can use more than one partition, specify their names in a comma separate list and the one offering earliest initiation will be used with no regard given to the partition name ordering (although higher priority partitions will be considered first). When the job is initiated, the name of the partition used will be placed first in the job record partition string.

- \#SBATCH --time=<time>

Set a limit on the total run time of the job allocation. If the requested time limit exceeds the partition’s time limit, the job will be left in a PENDING state (possibly indefinitely). The default time limit is the partition’s default time limit. When the time limit is reached, each task in each job step is sent SIGTERM followed by SIGKILL. The interval between signals is specified by the Slurm configuration parameter KillWait. The OverTimeLimit configuration parameter may permit the job to run longer than scheduled. Time resolution is one minute and second values are rounded up to the next minute. A time limit of zero requests that no time limit be imposed. Acceptable time formats include "minutes", "minutes:seconds", "hours:minutes:seconds", "days-hours", "days-hours:minutes" and "days-hours:minutes:seconds".

- \#SBATCH --output=<filename pattern>

Instruct Slurm to connect the batch script’s standard output directly to the file name specified in the "filename pattern". By default both standard output and standard error are directed to the same file. For job arrays, the default file name is "slurm-%A_%a.out", "%A" is replaced by the job ID and "%a" with the array index. For other jobs, the default file name is "slurm-%j.out", where the "%j" is replaced by the job ID.

- \#SBATCH --error=<filename pattern>

Instruct Slurm to connect the batch script’s standard error directly to the file name specified in the "filename pattern". By default both standard output and standard error are directed to the same file. For job arrays, the default file name is "slurm-%A_%a.out", "%A" is replaced by the job ID and "%a" with the array index. For other jobs, the default file name is "slurm-%j.out", where the "%j" is replaced by the job ID.

- \#SBATCH --mail-type=<type>

LiDO3 | First Contact page 47 of 71
Notify user by email when certain event types occur. Valid type values are NONE, BEGIN, END, FAIL, REQUEUE, ALL (equivalent to BEGIN, END, FAIL, REQUEUE and STAGE_OUT), STAGE_OUT (burst buffer stage out and teardown completed), TIME_LIMIT, TIME_LIMIT_90 (reached 90 percent of time limit), TIME_LIMIT_80 (reached 80 percent of time limit), TIME_LIMIT_50 (reached 50 percent of time limit) and ARRAY_TASKS (send emails for each array task). Multiple type values may be specified in a comma separated list. The user to be notified is indicated with --mail-user. Unless the ARRAY_TASKS option is specified, mail notifications on job BEGIN, END and FAIL apply to a job array as a whole rather than generating individual email messages for each task in the job array. Omit for no email notification.

- #SBATCH --mail-user=<user>

User’s email-address to receive email notification of state changes as defined by --mail-type. The default value is the submitting user. In contrast to the depiction in the man-page the value for --mail-user must be set if email notification is wanted for a submitting user (AKA slurm account\textsuperscript{21}) that is not the login user.

- #SBATCH --export=<environment variables | ALL | NONE>

Identify which environment variables are propagated to the batch job. Multiple environment variable names should be comma separated. Environment variable names may be specified to propagate the current value of those variables (e.g. "--export=EDITOR") or specific values for the variables may be exported (e.g. "--export=EDITOR=/bin/vi") in addition to the environment variables that would otherwise be set. This option is particularly important for jobs that are submitted on one cluster and execute on a different cluster (e.g. with different paths). By default all environment variables are propagated. If the argument is NONE or specific environment variable names, then the --get-user-env option will implicitly be set to load other environment variables based upon the user’s configuration on the cluster which executes the job.

3.5 Slurm for Torque/PBS users

A Torque queue is a Slurm partition.

3.5.1 Common user commands

\textsuperscript{21}Usually the login user has the same name as the slurm account. Some faculties use a different slurm account to submit jobs so that they can share the job management and the results.
Table 3.4: Comparison of some common commands in SLURM and in Torque (PBS) and Maui.

<table>
<thead>
<tr>
<th>Task</th>
<th>Torque/PBS</th>
<th>Slurm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Submit a job</td>
<td>qsub &lt;script file&gt;</td>
<td>sbatch &lt;script file&gt;</td>
</tr>
<tr>
<td>Delete a job</td>
<td>qdel &lt;job_id&gt;</td>
<td>scancel &lt;job_id&gt;</td>
</tr>
<tr>
<td>Job status</td>
<td>qstat [job_id]</td>
<td>squeue [job_id]</td>
</tr>
<tr>
<td>Job status by user</td>
<td>qstat -u &lt;user name&gt;</td>
<td>squeue -u &lt;user name&gt;</td>
</tr>
<tr>
<td>Expected job start time</td>
<td>qstat -q</td>
<td>squeue -start</td>
</tr>
<tr>
<td>Job status</td>
<td>qstat -Q -f</td>
<td>scontrol show partition</td>
</tr>
<tr>
<td>Job details</td>
<td>qstat -f &lt;job_id&gt;</td>
<td>scontrol show job &lt;job_id&gt;</td>
</tr>
<tr>
<td>Job status</td>
<td>squeue -Q -f</td>
<td>scontrol show partition</td>
</tr>
<tr>
<td>Node details</td>
<td>pbsnode &lt;node name&gt;</td>
<td>scontrol show node &lt;node name&gt;</td>
</tr>
<tr>
<td>Node list</td>
<td>pbsnode -l</td>
<td>sinfo -N</td>
</tr>
<tr>
<td>QoS details</td>
<td>mdiag -q # in Maui</td>
<td>sacctmgr show qos</td>
</tr>
<tr>
<td>Cluster status</td>
<td>qstat -a</td>
<td>sinfo</td>
</tr>
<tr>
<td>GUI</td>
<td>pbsmon</td>
<td>sview</td>
</tr>
</tbody>
</table>

3.5.2 User command for job control

Table 3.5: Job control.

<table>
<thead>
<tr>
<th>Action</th>
<th>Slurm</th>
<th>Torque/PBS</th>
<th>Maui</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job information</td>
<td>squeue &lt;job_id&gt;</td>
<td>qstat &lt;job_id&gt;</td>
<td>checkjob</td>
</tr>
<tr>
<td></td>
<td>sccontrol show job &lt;job_id&gt;</td>
<td>qstat -f &lt;job_id&gt;</td>
<td></td>
</tr>
<tr>
<td>Job information (all)</td>
<td>squeue -al</td>
<td>qstat -f</td>
<td></td>
</tr>
<tr>
<td></td>
<td>sccontrol show job</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Queue information</td>
<td>squeue</td>
<td>qstat</td>
<td>showq</td>
</tr>
<tr>
<td>Delete a job</td>
<td>scancel &lt;job_id&gt;</td>
<td>qdel</td>
<td></td>
</tr>
<tr>
<td>Clean up leftover job</td>
<td></td>
<td>momctl -c &lt;job_id&gt;</td>
<td></td>
</tr>
<tr>
<td>Submit a job</td>
<td>srun &lt;jobfile&gt;</td>
<td>qsub &lt;jobfile&gt;</td>
<td>msub</td>
</tr>
<tr>
<td></td>
<td>sbatch &lt;jobfile&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>alloc &lt;jobfile&gt;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interactive job</td>
<td>srun -N &lt;minnodes)[-maxnodes] \n-p &lt;partition&gt; sh</td>
<td>qsub -I</td>
<td></td>
</tr>
<tr>
<td></td>
<td>srun -test-only -p &lt;partition&gt; \n-n 1 -t &lt;time limit&gt; sh</td>
<td>showbf</td>
<td></td>
</tr>
<tr>
<td>Free processors</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Expected start time</td>
<td>squeue -start -j &lt;job_id&gt;</td>
<td>showstart</td>
<td></td>
</tr>
<tr>
<td>Blocked jobs</td>
<td>squeue -start</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Queues/partitions</td>
<td>scontrol show partition</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>qstat -Qf</td>
<td></td>
</tr>
</tbody>
</table>
...continued from previous page

<table>
<thead>
<tr>
<th>Action</th>
<th>Slurm</th>
<th>Torque/PBS</th>
<th>Maui</th>
</tr>
</thead>
</table>
| Queue  | sinfo -h  
sinfo -o "%P %l %c %D "  
scontrol update JobId=<job_id>  
StartTime=now | qstat -q | |
| Start job | scontrol update JobId=<job_id>  
StartTime=now | qrun | runjob |
| Hold job | scontrol update JobId=<job_id>  
StartTime=now+30days | qhold <job_id> | sethold |
| Release hold job | scontrol update JobId=<job_id>  
StartTime=now | qrls <job_id> | releasehold |
| Pending job | scontrol requeue <job_id>  
sview | xpbs | |
| Graphical Frontend | scontrol update JobId=<job_id>  
-nice=-10000 | setspri 10000 \  
<job_id> | |
| set priority | scontrol requeue <job_id> | mjbgctl -R \  
<job_id> | |
| preempt job | scontrol suspend <job_id> | mjbgctl -s \  
<job_id> | |
| suspend job | scontrol resume <job_id> | mjbgctl -r \  
<job_id> | |
| resume job | | | |

---

22See also section *Format options for slurm commands* on page 51.
### 3.5.3 List of job states

Table 3.6: Job state.

<table>
<thead>
<tr>
<th>Short</th>
<th>Long</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA</td>
<td>CANCELLED</td>
<td>Job was explicitly cancelled by the user or system administrator. The job may or may not have been initiated.</td>
</tr>
<tr>
<td>CD</td>
<td>COMPLETED</td>
<td>Job has terminated all processes on all nodes.</td>
</tr>
<tr>
<td>CF</td>
<td>CONFIGURING</td>
<td>Job has been allocated resources, but are waiting for them to become ready for use (e.g. booting).</td>
</tr>
<tr>
<td>CG</td>
<td>COMPLETING</td>
<td>Job is in the process of completing. Some processes on some nodes may still be active.</td>
</tr>
<tr>
<td>F</td>
<td>FAILED</td>
<td>Job terminated with non-zero exit code or other failure condition.</td>
</tr>
<tr>
<td>NF</td>
<td>NODE_FAIL</td>
<td>Job terminated due to failure of one or more allocated nodes.</td>
</tr>
<tr>
<td>PD</td>
<td>PENDING</td>
<td>Job is awaiting resource allocation.</td>
</tr>
<tr>
<td>PR</td>
<td>PREEMPTED</td>
<td>Job terminated due to preemption.</td>
</tr>
<tr>
<td>R</td>
<td>RUNNING</td>
<td>Job currently has an allocation.</td>
</tr>
<tr>
<td>S</td>
<td>SUSPENDED</td>
<td>Job has an allocation, but execution has been suspended.</td>
</tr>
<tr>
<td>TO</td>
<td>TIMEOUT</td>
<td>Job terminated upon reaching its time limit.</td>
</tr>
</tbody>
</table>

### 3.5.4 Format options for slurm commands

The available field specifications include:

Table 3.7: Field specifications.

<table>
<thead>
<tr>
<th>Field</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>%a</td>
<td>State/availability of a partition</td>
</tr>
<tr>
<td>%A</td>
<td>Number of nodes by state in the format “allocated/idle”. Do not use this with a node state option (“%t” or “%T”) or the different node states will be placed on separate lines.</td>
</tr>
<tr>
<td>%c</td>
<td>Number of CPUs per node</td>
</tr>
<tr>
<td>%d</td>
<td>Size of temporary disk space per node in megabytes</td>
</tr>
<tr>
<td>%D</td>
<td>Number of nodes</td>
</tr>
<tr>
<td>%f</td>
<td>Features associated with the nodes</td>
</tr>
<tr>
<td>%F</td>
<td>Number of nodes by state in the format “allocated/idle/other/total”. Do not use this with a node state option (“%t” or “%T”) or the different node states will be placed on separate lines.</td>
</tr>
<tr>
<td>%g</td>
<td>Groups which may use the nodes</td>
</tr>
<tr>
<td>%h</td>
<td>Jobs may share nodes, “yes”, “no”, or “force”</td>
</tr>
<tr>
<td>%l</td>
<td>Maximum time for any job in the format “days-hours:minutes:seconds”</td>
</tr>
<tr>
<td>%m</td>
<td>Size of memory per node in megabytes</td>
</tr>
<tr>
<td>%N</td>
<td>List of node names</td>
</tr>
</tbody>
</table>

continued on next page
### 3.5.5 Job variables

The available field specifications include:

<table>
<thead>
<tr>
<th>Field</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>%P</td>
<td>Partition name</td>
</tr>
<tr>
<td>%r</td>
<td>Only user root may initiate jobs, “yes” or “no”</td>
</tr>
<tr>
<td>%R</td>
<td>The reason a node is unavailable (down, drained, or draining states)</td>
</tr>
<tr>
<td>%s</td>
<td>Maximum job size in nodes</td>
</tr>
<tr>
<td>%t</td>
<td>State of nodes, compact form</td>
</tr>
<tr>
<td>%T</td>
<td>State of nodes, extended form</td>
</tr>
<tr>
<td>%w</td>
<td>Scheduling weight of the nodes</td>
</tr>
<tr>
<td>.&lt;&lt;&gt;</td>
<td>right justification of the field</td>
</tr>
<tr>
<td>&lt;&lt;&gt;</td>
<td>size of field</td>
</tr>
</tbody>
</table>

#### Table 3.8: Job variables.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Torque/PBS</th>
<th>Slurm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job ID</td>
<td>PBS_JOBID</td>
<td>SLURM_JOB_ID / SLURM_JOBID</td>
</tr>
<tr>
<td>Job name</td>
<td>PBS_JOBNAME</td>
<td>SLURM_JOB_NAME</td>
</tr>
<tr>
<td>Node list</td>
<td>▲ PBS_NODELIST</td>
<td>SLURM_JOB_NODELIST / SLURM_NODELIST</td>
</tr>
<tr>
<td>Submit directory</td>
<td>PBS_O_WORKDIR</td>
<td>SLURM_SUBMIT_DIR</td>
</tr>
<tr>
<td>Submit host</td>
<td>PBS_O_HOST</td>
<td>SLURM_SUBMIT_HOST</td>
</tr>
<tr>
<td>Job array index</td>
<td>PBS_PBS_ARRAY_INDEX</td>
<td>SLURM_ARRAY_TASK_ID</td>
</tr>
<tr>
<td>User</td>
<td>PBS_USER</td>
<td>SLURM_JOB_USER</td>
</tr>
</tbody>
</table>

### 3.6 Modules

The software and tools needed for development and job execution are organized as modules. Modules dynamically modify the users environment and make it possible to

- get a clean environment with no software visible at all,
- install concurrent versions of the same software and
- use software that usually excludes each other.

Working with those modules is done with the `module` command.

²³ [http://linux.die.net/man/1/module](http://linux.die.net/man/1/module)
3.6.1 Loaded modules

The command `module list` shows the modules that are currently loaded in your environment:

```
$ module list
No Modulefiles Currently Loaded.
```

3.6.2 Available modules

To list the modules that can be potentially loaded, enter the command `module avail`.

```
$ module avail
--- /usr/share/Modules/modulefiles ---
dot module-git module-info modules null use.own

---- /cluster/sfw/modulefiles ---
abaqus/2016 gcc/6.4.0 openblas/0.2.19
clang/4.0.1 gcc/7.1.0 openmpi/mpi_thread_multiple/cuda/2.1.1
(...)
```

3.6.3 Load a module

To load a module into your environment, enter the command `module add`, followed by the `<MODULE_NAME>`:

```
$ module add clang
$ module list
Currently Loaded Modulefiles:
  1) clang/4.0.1
```

3.6.4 Unload a module

To unload a specific module, use the command `module rm`, followed by the `<MODULE_NAME>`:

```
$ module rm clang
$ module list
No Modulefiles Currently Loaded.
```

To unload all modules, use `module purge`.

Further documentation of the module concept is available at the [HLRN](https://www.hlrn.de/home/view/System2/ModulesUsage).
Important: in order to make the activated modules available on the compute nodes (during execution time) as well, the command module add must be included in the user’s shell init files (e.g. `.bash_profile` or `job script`).

### 3.6.5 Modules in job scripts

If you run a job that depends on modules, please ensure that these modules are included in the user’s shell init files (e.g. `.bash_profile`), so that the job has a proper environment set up when being executed on the compute nodes! Alternatively, the following lines are to be included in the Slurm job script before starting the application:

```bash
# Clean module environment
module purge
# Load modules needed
module load [compiler modules][MPI modules]
```

### 3.7 Compiler modules

Compilers and libraries are selected and activated via `module` commands (see section `Modules`).

<table>
<thead>
<tr>
<th>Compiler</th>
<th>Module</th>
<th>Commands</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU Compiler Collection</td>
<td>module add gcc</td>
<td>gcc, g++, gfortran</td>
</tr>
<tr>
<td>Intel Studio XE</td>
<td>module add intel</td>
<td>icc, icpc, ifort</td>
</tr>
<tr>
<td>Portland PGI compiler</td>
<td>module add pgi</td>
<td>pgcc, pgCC, pgf77, pgf95</td>
</tr>
<tr>
<td>Oracle Solaris Studio</td>
<td>module add orangecstudio</td>
<td>suncc, sunCC, sunf77, sunf95</td>
</tr>
<tr>
<td>Clang compiler</td>
<td>module add clang</td>
<td>clang, clang++</td>
</tr>
</tbody>
</table>

If you want to compile a parallel program using MPI you can use the corresponding compiler wrappers from the Open MPI modules.

The naming scheme for the openmpi modules is as follows:

```
openmpi//THREADINGSUPPORT/CUDASUPPORT/OPENMPIVERSION
```

with

- THREADINGSUPPORT: whether build with thread multiple support:\(^{25}\)
  
  ```
  mpi_thread_multiple/no_mpi_thread_multiple
  ```

---

- CUDASUPPORT: whether to enable the build-in support for data transfers between the GPUs and the network controller without explicit memory transfer statements.

- OPENMPIVERSION: the actual Open MPI version, e.g. 4.0.1

For a complete overview of all modules available please see:

```bash
module avail openmpi
```
3.8 Examples

3.8.1 Basic slurm script example

The following script asks for usage of 1 compute node with 20 cores for 10 minutes. See ‘man sbatch’ for details.

```bash
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --nodes=1 --cpus-per-task=20 --constraint=cstd01
#SBATCH --partition=short
#SBATCH --mem=60000
#SBATCH --mail-user=test.user@tu-dortmund.de
#SBATCH --mail-type=ALL

cd /work/user/workdir
module purge
module load pgi/17.5
export OMP_NUM_THREADS=20

echo "sbatch: START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID $SLURM_TASK_PID)
      on $SLURMD_NODENAME"

echo "sbatch: SLURM_JOB_NODELIST $SLURM_JOB_NODELIST"

echo "sbatch: SLURM_JOB_ACCOUNT $SLURM_JOB_ACCOUNT"

srun ./myapp
```

3.8.2 Example using multiple GPU nodes

The following script asks for usage of 2 compute node with 20 cores each and 2 GPUs per node for 10 minutes. See ‘man sbatch’ for details.

```bash
#!/bin/bash
#SBATCH --time=00:10:00
#SBATCH --nodes=2 --cpus-per-task=20 --constraint=cgpu01 --gres=gpu:2
#SBATCH --partition=short
#SBATCH --mem=60000
#SBATCH --mail-user=test.user@tu-dortmund.de
#SBATCH --mail-type=ALL

cd /work/user/workdir
module purge
module load pgi/17.5
export OMP_NUM_THREADS=20

```

```bash
```
module purge
d module load cuda
echo "sbatch: START SLURM_JOB_ID $SLURM_JOB_ID (SLURM_TASK_PID $SLURM_TASK_PID)
  on $SLURMD_NODENAME"
echo "sbatch: SLURM_JOB_NODELIST $SLURM_JOB_NODELIST"
echo "sbatch: SLURM_JOB_ACCOUNT $SLURM_JOB_ACCOUNT"
nvidia-smi -a

3.8.3 Common software example: R

A user can build additional R modules and store them in his home directory. That is the preferred way over having to install them systemwide to avoid conflict situations where user A needs a set of R modules in a certain version and user B needing them in older or newer versions. Installing them to a user’s home directory allows the user to quickly check whether up- or downgrading R modules resolves issues he is having with them.

Given that the home directory is writable only on both gateway servers, a user should not try to build R modules on any of the compute nodes (unless he redefines HOME to point to some writable location inside a SLURM job). The following convenience script facilitates downloading optional R packages:

```
#!/bin/bash
extra=""
if test -z "$@"; then
  extra="-O index.html"
fi
file=$( grep "">$1_ index.html | awk -F'>' '{ print $7 }' | sed 's|</a||' )
wget http://cran.r-project.org/src/contrib/$file $extra
```

Copy this content to a new file named `load_r_module`, set the executable bit for the script

```
chmod 755 load_r_mod
```

When invoked without arguments, the script downloads the index of the directory [https://cran.r-project.org/src/contrib/](https://cran.r-project.org/src/contrib/) and stores it as index.html in the local directory:

```
./load_r_mod
```
When invoked with an argument, the script queries the cache file index.html in the local directory for that given argument string and tries to download the tarball if a R module is found that matches this string. Example:

```
./load_r_mod digest
```

will download the most recent version of the R module digest, at the time of writing digest_0.6.23.tar.gz.

Subsequently, the user can compile and install this R module as follows:

```
module purge
module load R/<version of your liking>
platform=$( R --quiet -e 'R.version' | grep ^platform | awk '{ print $2; }' )
version=$( R --quiet -e 'R.version' | grep ^version.string | sed -e 's/^. */uni21AAversion [0-9.]*$/\1/' )
export INSTALL_HOME=/home/${USER}/R/${platform}-library/${version};
mkdir -p $INSTALL_HOME
for package in <list of desired R modules>; do test -d $INSTALL_HOME/$package | uni21AA || (./load_r_mod $package && R CMD INSTALL --library=$INSTALL_HOME --configure-args=--with-mpi=${OMPI_HOME} $package_*.tar.gz || ( echo "ERROR"; read junk )); done
```

Make sure to replace the strings <version of your liking> and <list of desired modules> in the instructions above appropriately. In case the R module(s) you want to install have unfulfilled dependencies, the R install command will fail, reporting the name of the missing dependency:

```
for package in spdep ; do test -d $INSTALL_HOME/$package | uni21AA || (./load_r_mod $package && R CMD INSTALL --library=$INSTALL_HOME --configure-args=--with-mpi=${OMPI_HOME} $package_*.tar.gz || ( echo "ERROR. Press any key to continue"; read junk )); done
```

ERROR: dependencies 'sp', 'spData', 'sf', 'deldir', 'LearnBayes', 'coda', 'expm', 'gmodels' are not available for package 'spdep'

* removing '/home/msvebuij/R/x86_64-pc-linux-gnu-library/3.6.1/rgdal'
ERROR. Press any key to continue
In this particular example, where one wanted to install the R module `spdep`, we needed to first compile and install `sp` and then a lot of others which had dependencies of their own. Prepend iteratively all the missing R modules to the list of desired modules in ascending order of dependency and start over until the compilations succeeds. This approach easily requires a dozen or so iterations, depending on the particular R module a user wants to use. For this example, the complete instruction would look like:

```bash
for package in sp raster spData e1071 classInt DBI units sf deldir LearnBayes coda expm gmodels spdep ; do test -d ${INSTALL_HOME}/${package} || (./load_r_mod ${package} && R CMD INSTALL --library=${INSTALL_HOME} --configure-args=--with-mpi=${OMPI_HOME} ${package}*.tar.gz || ( echo "ERROR"; read junk )); done
```

It may turn out, however, that some system software needs to installed as well upon which an R module relies. In that case, please inform the LiDO team what system software is required and, better yet, additionally what R module you are trying to compile.

The following script, when invoked via:

```bash
sbatch StartRBatchJobViaSLURM.sh
```

asks for 1 compute node with 1 cores for 90 minutes and 15 seconds:

```bash
#!/bin/bash -l
#SBATCH --job-name=Ranalysis
#SBATCH --output=/work/myusername/tmp/slurm_job
#SBATCH --error=/work/myusername/tmp/slurm_job
# run at most for 0 days, 1 hour, 30 minutes and 15 seconds
#SBATCH --time=0-01:30:15
#SBATCH --partition=short
# ask for a single compute core on one compute node
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=1
# memory requirement per CPU in megabytes
#SBATCH --mem-per-cpu=1536
# send mail when jobs starts, end, fails, gets requeued etc.
#SBATCH --mail-user=my.name@tu-dortmund.de
#SBATCH --mail-type=ALL
cd /work/myusername/tmp
module purge
module load R/3.6.1-gcc73-base openmpi/mpi_thread_multiple/no_cuda/4.0.1
# Run the R analysis, use an external script for the R instructions
Rscript my_script.R
```
3.8.4 Common software example: Matlab

The following script, when invoked via

```
sbatch StartMatlabBatchJobViaSLURM.sh
```

asks for 1 compute node with 10 cores for 90 minutes:

```
#!/bin/bash -l
#SBATCH --job-name=MatlabSimulation
#SBATCH --output=/work/myusername/tmp/slurm_job
#SBATCH --error=/work/myusername/tmp/slurm_job
#SBATCH --time=0-01:30:15
#SBATCH --partition=short
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=10
#SBATCH --mem-per-cpu=1536
#SBATCH --mail-user=my.name@tu-dortmund.de
#SBATCH --mail-type=ALL

cd /work/myusername/tmp
module purge
module load matlab/r2019b

# Run the Matlab simulation, stored in /work/myusername/tmp/matlab_main.m
srun matlab -nodisplay -nosplash -r 'matlab_main; quit;'
```

Listing 3.1: Contents of file ‘StartMatlabBatchJobViaSLURM.sh’

Once SLURM grants these resources, Matlab’s command line interface get invoked on the assigned compute node, spawns as many Matlab worker processes as cores requested in the SLURM job script in order to calculate in parallel an estimate for the value of π:

```
pc = parcluster('local')

% use a fixed number of Matlab worker processes, e.g. 5
% parpool(pc, 5)

% automatically choose as many Matlab worker processes as cores requested in
% SLURM job file
parpool(pc, str2num(getenv('SLURM_CPUS_ON_NODE')))

% run Matlab simulation that uses commands like 'parfor', 'parfeval',
% 'parfevalOnAll', 'spmd' or 'distributed' to have Matlab automatically
% distribute the workload on multiple worker processes
```
ranging on the helper script

Listing 3.2: Contents of file 'matlab_main.m'

```
% Calculate the value of Pi using a Monte Carlo simulation
itmax=1e9;
n=0;
tic;
parfor i = 1:itmax
    x=rand;
y=rand;
    if (x^2 + y^2 < 1.0)
        n=n+1;
    end
end
time = toc;
pi = 4.0 * n / itmax;
fprintf("Calculating pi = %.10f took %s seconds\n", pi, time);
```

Listing 3.3: Contents of file 'EstimatePi.m'

3.8.5 Third-party node usage example

In this case, the partition is related to the nodes itself and no additional constraint is needed to identify the nodes to be used.

```
#!/bin/bash -l
#SBATCH --time=00:10:00
#SBATCH --nodes=1 --cpus-per-task=20
#SBATCH --partition=ext_vwl_prio
#SBATCH --mem=250000
#SBATCH --mail-user=test.user@tu-dortmund.de
#SBATCH --mail-type=ALL

cd /work/user/workdir
module purge
module load pgi/17.5
export OMP_NUM_THREADS=20
srun ./myapp
```
### 3.8.6 Have a job automatically clean up when risking to exceed the configured walltime

By default, *Slurm* (up to version 20.02) sends the signal `SIGQUIT` and the signal `SIGTERM` to a job that exceeded its requested walltime at the same time (*Slurm* configuration file parameter *WaitTime*). If one needs more time between the two signals, for instance, to move result files from the `/scratch` file system to the parallel file system and to clean up any remaining temporary files, the user needs to set up three things in the *Slurm* job script:

1. A `SBATCH` instruction when to send what kind of signal. This can be done by including the following lines (only the last one is the actual workhorse) in the header section of the *Slurm* job script:

   ```bash
   #SBATCH --signal=B:SIGQUIT@120
   ```

   which sends approximately 2 minutes before exceeding the wall time the signal `SIGQUIT`.

2. A shell `trap` trying to catch the signal and defining an action to undertake upon receiving it. Example:

   ```bash
   trap -- 'echo "$\text{Got SIGQUIT at }$(date)$. Starting cleanup"; test -d /uni21AA/scratch/${USER}/${SLURM_JOB_ID} && rm -rf /uni21AA/scratch/${USER}/${SLURM_JOB_ID}';
   ```

   This oneliner is hard to read such that some users may prefer the alternative of a custom shell function defining the actions near job end:

   ```bash
   cleanup_before_exiting() { 
     echo -n 'Got SIGQUIT at $(date),';
     echo 'roughly 2 minutes before exceeding the';
     echo 'walltime. Starting clean up.';
     test -d /scratch/${USER}/${SLURM_JOB_ID} && 
     rm -rf /scratch/${USER}/${SLURM_JOB_ID} 
     exit 0;
   }
   trap -- 'cleanup_before_exiting' SIGQUIT
   ```
3. Finally, it is absolutely mandatory to send any of the long-running processes your SLURM job will execute immediately to the background by adding a trailing ampersand to that process’ command and to subsequently add a ‘wait’ shell command that causes the SLURM job file to wait for the completion of the long-running process before continuing. Example:

```
# Start the actual worker process (a simple 'sleep' command, for instance).
# Note: It is absolutely mandatory to immediately send the job to the background with the trailing ampersand and then use the 'wait' shell command to wait for the completion of the worker process. Otherwise the SLURM signal is not caught by this SLURM job script and the configured action to run shortly before exceeding the requested walltime will not run!

sleep 600 &
wait
```

A complete Slurm job file example is given below:

```
#!/bin/sh

#SBATCH --time=00:04:00
#SBATCH --nodes=1 --ntasks-per-node=1 --cpus-per-task=1
#SBATCH --partition=short
#SBATCH --mail-type=NONE

# When a job is within 120 seconds of its end time, send it the signal SIGQUIT.
# Note 1: due to the resolution of event handling by SLURM, the signal may be sent up to 60 seconds earlier than specified.
# Note 2: only *one* signal can be defined. Later SLURM signal definition override earlier definitions.
#SBATCH --signal=B:SIGQUIT@120
useTrapVariant=2

if test \$\(useTrapVariant\) = 1; then
  ###################################
  # Example 1: Simple signal handling with a one-liner: print a message, then start cleaning up in /scratch before job exceeds requested walltime.
  trap -- echo "Got SIGQUIT at \$(date). Starting cleanup."
  test -d /scratch/\$\{USER\}/$\{SLURM_JOB_ID\} & &
  rm -rf /scratch/\$\{USER\}/$\{SLURM_JOB_ID\} ' SIGQUIT;
else
  ###################################
  # Example 2: Same purpose, but more readable
  # with a user function; print a message, then start cleaning up in /scratch before job exceeds requested walltime.
```

A complete Slurm job file example is given below:
cleanup_before_exiting() {
    echo -n 'Got SIGQUIT at $(date),,'
    echo -n 'roughly 2 minutes before exceeding the,'
    echo 'walltime. Starting clean up.';
    test -d /scratch/${USER}/${SLURM_JOB_ID} && \
    rm -rf /scratch/${USER}/${SLURM_JOB_ID}
    exit 0;
}

trap -- 'cleanup_before_exiting' SIGQUIT

# Start the actual worker process (a simple 'sleep'
# in this example).
# Note: It is absolutely mandatory to immediately
# send the job to the background with the
# trailing ampersand and then use the 'wait'
# shell command to wait for the completion of
# the worker process. Otherwise the SLURM
# signal is *not* caught by this SLURM job
# script and the configured action to run
# shortly before exceeding the requested
# walltime will *not* run!

sleep 600 &
wait

3.8.7 Example for job steps

A job consists of

- one or more steps,
- each step executing one or more tasks,
- each task using one or more CPU.

Typically jobs are created with the sbatch command, containing steps that are created with the srun command.

Tasks are requested (at the job level or the step level) with --ntasks and CPUs\textsuperscript{26} are requested per task with --cpus-per-task.

Note that jobs submitted with sbatch have one implicit step — the Bash script itself.

\texttt{
#!/bin/bash -l
#SBATCH --nodes 7
#SBATCH --tasks-per-node 6
#SBATCH --cpus-per-task 1
# The job requests 42 CPUs, on 7 nodes, every task will use 1 cpu.

# STEP 01:
# request 7 nodes,
# sub-allocate 7 tasks (one per node) to create a directory in /scratch.
}

\textsuperscript{26}CPU cores to be more precise.
# Must run on every node but only one task per node needed.
```
srun --nodes 7 --tasks 7 mkdir -p /scratch/${USER}_${SLURM_JOBID}
```

# STEP 02:
# No explicit allocation, hence use all 64 tasks to run an MPI program
# on some data to produce some output.
```
srun mpi_process.mpi <input.dat > output.txt &
```

# STEP 03:
# sub-allocate of 24 tasks for a not well scaling program.
```
srun --ntasks 24 --nodes 4 --exclusive reduce_mpi_data < output.txt > result.txt &
```

# STEP 04:
# sub-allocate a single node.
# The gzip cannot run on separate nodes to compress output.txt.
# Thanks to the ampersand `&` this step runs at the same time as the
# previous step
```
OMP_NUM_THREAD=10; srun --ntasks 10 --nodes 1 --exclusive gzip output.txt &
```

# wait for the steps to finish
```
wait
```
3.9 System overview

- name: LiDO3
- architecture: Distributed Memory
- vendor: Megware
- installation: 2017

Figure 3.10: Schematic representation of the LiDO3 architecture.
3.10 Dictionary

3.10.1 Walltime

*Walltime, or Wall-Clock Time* is the passage of time from the moment a job is assigned one or multiple compute nodes and started until it ends, seen from the human perspective. In other words, if the job is started but some necessary resource is missing or becomes unavailable while the job is still running (e.g., filesystem, network, results from a previous computation as input data), walltime increases. In this case, whether or not CPU time increases depends on whether the processes started by the job perform a busy-wait or put the CPU to sleep while waiting for the necessary resource to become available again. So, if a requested CPU waits for seven hours for resources and intermittently uses the CPU for one hour, walltime is 8 hours, CPU time is 1 hour. When using multiple cores, the CPU time is accordingly scaled - walltime is not, obviously.

![Diagram of walltime](image)

Figure 3.11: A job waiting more than utilizing the CPU uses eight hours walltime total.

3.10.2 Backfilling

*Backfilling* is a mechanism that allows starting a job with lower priority before a job with higher priority in the queue without delaying the job with the higher priority. By doing this *Backfilling* helps to maximize cluster utilization and throughput.

Let *Job A* be a job that just has started. *Job B* needs the nodes that are currently used by *Job A* and some extra nodes. Thus it can only start after *Job A* has been finished.
Figure 3.12: *Job B* is waiting for nodes used by *Job A*.

*Job C* is smaller than *Job A* - it will use less *Walltime*. And it does not depend on nodes that are used by *Job A*. This means that *Job C* can be started before *Job B* without delaying *Job B*.

Figure 3.13: *Job C* is started before *Job B* because it will be finished before *Job B* can start.
Filling those gaps in the execution plan is called *Backfilling*.

### 3.11 Get support

For support and further assistance, please write an email to
the LiDO team mailing list
( lido-team.itmc@lists.tu-dortmund.de ).

---

27 [mailto:lido-team.itmc@lists.tu-dortmund.de?subject=LiDO3: %20support%20needed](mailto:lido-team.itmc@lists.tu-dortmund.de?subject=LiDO3: %20support%20needed)
3.12 Appendix

3.12.1 Symbolic links for non-writable home directory

Here is an example of some software that needs to write in the home directory during runtime. $(NEWUSER)$ contains the name of the user that is affected.

```bash
# Software like 'matplotlib' (standalone or inside ParaView) tries to write a
# lock file to $HOME/.cache/matplotlib/tex.cache/.matplotlib_lock-*. Without
# this symbolic link, matplotlib would fail when run on compute nodes.
$ ssh gw01
$ mkdir /work/${NEWUSER}/.allinea
$ ln -s /work/${NEWUSER}/.allinea /home/${NEWUSER}/.allinea
$ mkdir /work/${NEWUSER}/.ansys
$ ln -s /work/${NEWUSER}/.ansys /home/${NEWUSER}/.ansys
$ mkdir /work/${NEWUSER}/.cache
$ ln -s /work/${NEWUSER}/.cache /home/${NEWUSER}/.cache
$ mkdir -p /work/${NEWUSER}/.ccache
$ ln -s /work/${NEWUSER}/.ccache /home/${NEWUSER}/.ccache
$ mkdir /work/${NEWUSER}/.cmake/packages
$ ln -s /work/${NEWUSER}/.cmake/packages /home/${NEWUSER}/.cmake
$ mkdir /work/${NEWUSER}/.cfx
$ ln -s /work/${NEWUSER}/.cfx /home/${NEWUSER}/.cfx
$ mkdir /work/${NEWUSER}/.felix
$ ln -s /work/${NEWUSER}/.felix /home/${NEWUSER}/.felix
$ mkdir /work/${NEWUSER}/felix-cache
$ ln -s /work/${NEWUSER}/felix-cache /home/${NEWUSER}/felix-cache
$ mkdir /work/${NEWUSER}/.java
$ ln -s /work/${NEWUSER}/.java /home/${NEWUSER}/.java
$ mkdir /work/${NEWUSER}/.lesshst
$ ln -s /work/${NEWUSER}/.lesshst /home/${NEWUSER}/.lesshst
$ mkdir /work/${NEWUSER}/.matlab
$ ln -s /work/${NEWUSER}/.matlab /home/${NEWUSER}/.matlab
$ mkdir /work/${NEWUSER}/.oracle_jre_usage
$ ln -s /work/${NEWUSER}/.oracle_jre_usage /home/${NEWUSER}/.oracle_jre_usage
$ mkdir -p /work/${NEWUSER}/.ssh
$ ln -s /work/${NEWUSER}/.ssh /home/${NEWUSER}/.ssh
$ mkdir /work/${NEWUSER}/.subversion
$ ln -s /work/${NEWUSER}/.subversion /home/${NEWUSER}/.subversion
#does not work $ touch /work/${NEWUSER}/.viminfo
#does not work $ ln -s /work/${NEWUSER}/.viminfo /home/${NEWUSER}/.viminfo
$ touch /work/${NEWUSER}/.Xauthority
$ ln -s /work/${NEWUSER}/.Xauthority /home/${NEWUSER}/.Xauthority
```
3.12.2 Picture credits

- Windows Logo - [Wiki Commons](http://commons.wikimedia.org/wiki/Category:Microsoft_Windows_logos)
- Apple Logo - [Wiki Commons](http://commons.wikimedia.org/wiki/File:Apple_logo_black.svg?uselang=de)
- Tux Logo - [Wiki Commons](http://commons.wikimedia.org/wiki/Tux#/media/File:Tux.svg)
- Computer shape - [Openclipart](https://openclipart.org/detail/17391/computer)
- Server shape [Openclipart](https://openclipart.org/detail/171414/router)
- Light bulb - [Openclipart](https://openclipart.org/detail/211389/lightbulb)
- Warning triangle - [Openclipart](https://openclipart.org/detail/14428/h0us3s-Signs-Hazard-Warning-9-by-h0us3s)
- Clock - [Openclipart](https://openclipart.org/detail/217065/3-oclock)
- TU Dortmund ITMC - [itmc.tu-dortmund.de](https://www.itmc.tu-dortmund.de/cms/de/home/anfahrt/anfahrt-hauptgebaeude/index.html)
- Remaining screenshots and figures - created by the LiDO Team

---

28 http://commons.wikimedia.org/wiki/Category:Microsoft_Windows_logos
30 http://commons.wikimedia.org/wiki/Tux#/media/File:Tux.svg
31 https://openclipart.org/detail/17391/computer
32 https://openclipart.org/detail/171414/router
33 https://openclipart.org/detail/211389/lightbulb
34 https://openclipart.org/detail/14428/h0us3s-Signs-Hazard-Warning-9-by-h0us3s
35 https://openclipart.org/detail/217065/3-oclock
36 https://www.itmc.tu-dortmund.de/cms/de/home/anfahrt/anfahrt-hauptgebaeude/index.html