

# Cluster Documentation

Leibniz Universität IT Services

Scientific Computing Group

Sat, 23 Aug 2025

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/start](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/start)

Last update: **2021/04/04 09:57**



# I am a new user - Quick Start

---

**Please note:** If you have absolutely no time at all, read at least this page. Investing some time here will save you much time later.

First, the basics:

- In order to access and use the cluster, you need an account. Accounts are organized in “projects”. Quite frequently, your institute will already have a project, and if that's the case, they can easily create an account for you. Ask for the key word “BIAS”.
- Creating an account should automatically subscribe you to the Cluster-News mailing list, where important announcements (for example maintenance periods), are announced. Did you receive a confirmation mail?
- Did you already change your password? You can change your password using the command `passwd`.
- Just requesting lots of resources and starting sequential program at the cluster without any parallelization will NOT make it run faster — it will just make everybody wait (including you). Read the docs.
- Consider the cluster system as a tool to facilitate your research. Mastering any tool takes time. Consider attending one of our introductory talks and also read this Cluster Handbook. It will save your time as well.

## About the cluster system

---

In order to meet the University's demand for computing resources with a lot of CPUs and memory, LUIS as part of the services provided also runs a Linux cluster system that is well-equipped to run parallel jobs. Scientists of the Leibniz University and students either working for an LUH institute or attending a lecture that makes use of scientific computing methods can use the cluster free of charge.

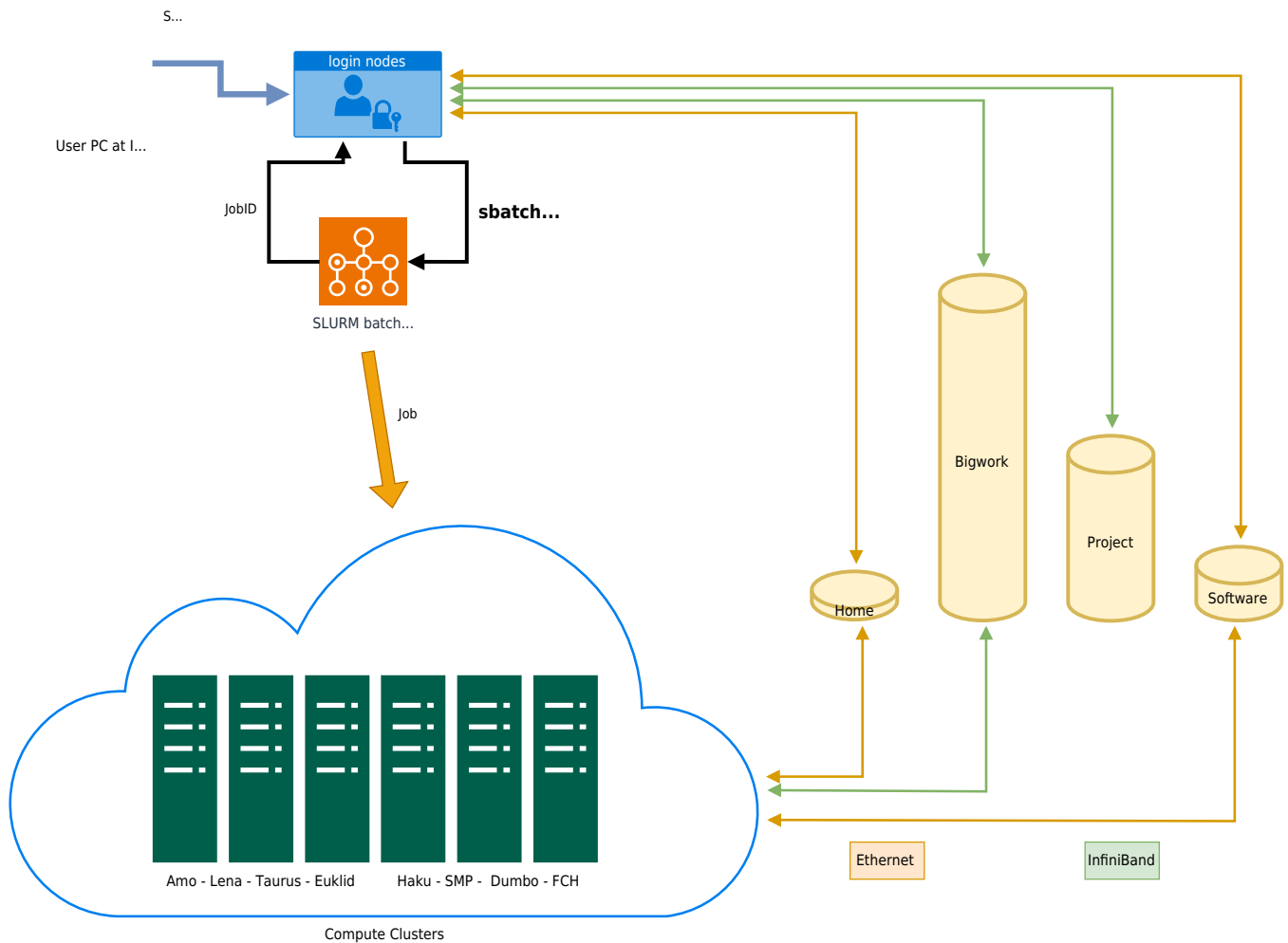


Fig. 1: Sketch of the main user-relevant components of the cluster system

Resources of the cluster system are largely a DFG major instrumentation. Therefore rules<sup>1)</sup> for DFG major instrumentation apply when using the cluster system. Project leaders of your EDV-Project bear responsibility to comply with the DFG rules.

## Getting access when your institute does not yet have a project

In case no project yet exists at your institute: a project is the frame in which you and others will manage accounts and get permission to use the cluster. To apply for a project and check the formal prerequisites and conditions, check <https://www.luis.uni-hannover.de/en/services/computing/scientific-computing> (the application form itself is called ORG.BEN.4). Basically, the person responsible for the project is responsible for the accounts they create.

Due to the long-term character of a project, the manager will usually be someone holding a permanent position at an institute. Someone doing a bachelor's or master's thesis ("temporary work" in this consideration) may get an account within such a project. Students can only get an account while working at an institute.

Once the project has been approved, the project manager can log in to the [BIAS](#) website and create accounts (usernames). Usernames should reflect the real name of the user, and the email addresses used should point to the real users and also need to be allocated within the .uni-hannover.de domain.

Note that user accounting on BIAS is not part of the service *Scientific Computing* and thus not part of the cluster system.

## What the cluster system may be used for

Parts of the cluster system are DFG major instrumentation, thus rules for DFG major instrumentation apply when using the cluster system. Furthermore software licenses are valid for research and teaching only. Accordingly the cluster system must only be used for research and teaching activities.

## Accessing the computing power of a cluster

The cluster system contains the compute resources listed on this page: [Computing Hardware](#). To access these nodes, you will need to generate so-called batch jobs, either by submitting a text file containing the job description or by configuring a job using the OpenOnDemand web portal we provide.

**IMPORTANT:** If you just log in to the cluster and run your programs directly on the login nodes ([login.cluster.uni-hannover.de](http://login.cluster.uni-hannover.de)), you will only use a small fraction of the power available. You'll also experience and generate all kinds of problems for yourself and others, depending on what you and other users do on the same login node.

So it is mandatory to submit batch job to the SLURM ressource manager (see the corresponding chapter of this documentation) or use the OpenOnDemand web portal <https://login.cluster.uni-hannover.de> to create these jobs. Using the login nodes to compute stuff is *absolutely not* the way you should do your work. To protect other users and keep the login nodes free for interactive command-line use, anything that tries to use more than 1800 cpu seconds on a login node will get killed automatically. The power of the cluster lies in the computing capabilities *behind* the login nodes, so please learn how to use them. It is, of course, okay to check out small things on a login node. But you should never try to run a real computation there.

Some institutes have a certain need for immediate job execution and priority. If certain conditions are met, they can ask to integrate their own hardware into cluster system in a service called [Forschungscluster-Housing](#) (FCH). Hardware in this service is reserved for the respective institute, usually during work days between eight o'clock in the morning and eight o'clock in the evening. At night-time and on weekends, all cluster users have access to these resources, which means that jobs that ask for less than 12 hours of wall time have a high chance of running on an FCH node overnight. Jobs with less than 60 hours can run on such a node during the weekend. So if you get directed to a machine that does not fit the name scheme of our main clusters, during off-hours, that is most likely an FCH node. For information about placing your institute's hardware into FCH, please get in touch with us.

<sup>1)</sup>

[www.dfg.de](http://www.dfg.de)

From:  
<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:  
[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/100\\_about\\_the\\_cluster\\_system](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/100_about_the_cluster_system)

Last update: **2025/01/27 09:29**



# How to connect to the cluster

---

You can only connect to the cluster from within the LUH network. If you want to connect from the “outside” (e.g. from Home), you'll have to establish a VPN connection to the university's network first. See [VPN Service](#) for details.

**Please note:** Working from outside the LUH may feel much slower than from your office, depending on the quality (bandwidth, latency, symmetry of up- and downstream) of your internet connection. If graphical applications feel sluggish, it is usually due to the connection and/or the network components in between, not a technical problem with the cluster system. You should also keep in mind the necessary bandwidth for a graphical connection on a high-resolution screen. Quite frequently, you can drastically improve performance by experimenting with the compression settings of X2Go, reducing window sizes, or the use of Open OnDemand as described below in this guide. If you have a good internet connection, you should be able to work quite comfortably from Home, except perhaps if you are in countries that are distant to Germany.

We support the following methods for connecting to the compute cluster:

- [SSH](#) - secure shell client
- [X2Go](#) - remote desktop sessions
- [Web browser](#) - Open OnDemand web portal

The following addresses should be used to connect to the cluster system:

- **[login.cluster.uni-hannover.de](https://login.cluster.uni-hannover.de)** in order to develop, prepare and submit jobs, editing text files.
- **[transfer.cluster.uni-hannover.de](https://transfer.cluster.uni-hannover.de)** whenever you want to transfer data to/from the cluster.

**❗IMPORTANT❗:** The login machines are *not* to be used for computations. In order to keep these nodes accessible for everyone, processes that use too much cpu time will get killed automatically after 30 minutes of elapsed cpu-time (that translates to only a few minutes if you try to start a parallel job there. Don't do that). We will also kill anything that uses more “elaborate” mechanisms trying to circumvent the limits, and we will disable accounts that we determine to be acting in bad faith. Please use interactive jobs for tasks like pre- or post-processing and even some larger compilations in order to avoid the frustrating experience of sudden shut down of your application. You can, of course, also use our Open OnDemand web portal which is reachable under <https://login.cluster.uni-hannover.de> if you get overwhelmed with all the options of the batch system at first.

**Please note:** File transfers on the login nodes will also get aborted if the execution time limit is reached. On the transfer node, execution time is unlimited, but you can not submit batch jobs from there to avoid clogging the node. We recommend to use ssh/scp or rsync to transfer files to/from the cluster, in particular for large files. The rsync command is particularly useful if your connection is unstable, because it can continue transfers that have been aborted.

**If you already know what you are doing: current hashes of the host key fingerprints of login and transfer**

## nodes

Connecting via ssh for the first time, you will probably get asked about the authenticity of the host key/hash presented to you by the login node you are trying to connect. This is done in order to ensure that you really connect to our servers and not a machine that someone could in theory have inserted as a so-called “man-in-the-middle-attack” in your connection or with another of various methods. So we suggest to check what the dialogue presents to you to ensure you do not enter your password at the wrong site. You will probably see only one of the following lines, since there are various algorithms used to compute the fingerprints and we do not know which one your tool will use - check for the algorithm used at the end of the line, e.g. ECDSA, ED25519.

The correct host key fingerprints are

```
MD5: bf:aa:71:bd:d1:19:d4:4f:b5:60:e7:cb:c4:26:85:a7 (ECDSA)
SHA256: cYIZZQC96J5bhannZnH2cGQLIBPLQVy29HNc6/vnyFg (ECDSA)
SHA1: fb:a1:47:ad:97:99:b3:66:86:06:8d:9d:56:46:61:3a:44:9c:a6:cb (ECDSA)

MD5: 3b:0a:17:0f:53:85:40:87:c2:be:ed:65:fb:40:59:8a (ED25519)
SHA256: sG8ZYabQctyGjxPD7X8K2IBxJIE5xHHZ9mQqjLVcjxo (ED25519)
SHA1: 51:f7:b9:0a:e3:84:8e:d8:2c:a4:e7:7d:42:14:d3:8b:62:2e:a6:5d (ED25519)

MD5: bd:b2:04:f9:4e:36:c1:b6:36:8e:d9:03:cc:5d:75:c4 (RSA)
SHA256: zkVF8Xyxmm7b0EKTN0lvRKe+nG1oHZDe0tY3Un60grg (RSA)
SHA1: ed:e3:ed:e6:bf:4f:29:9a:2c:72:92:c1:b4:ff:a6:b9:81:f6:6b:45 (RSA)

MD5: e3:c7:80:67:68:5b:d9:78:df:50:d7:e1:c5:ae:bf:e7 (DSA)
SHA256: DuWre1exwDsyTzD4yIMy60c2CFBCmx5o+l0LmWjMvUc (DSA)
SHA1: 85:95:f5:c9:9b:2b:29:82:5f:13:70:b9:2b:43:44:84:22:8f:87:40 (DSA)
```

If at least one of the fingerprints matches the one you get asked about, you can be reasonably sure that you are trying to connect to the cluster. In case you are using the command-line ssh client, it will remember an affirmative answer by adding a line with that fingerprint to your ~/.ssh/known\_hosts file. So you'll probably get asked only once.

## SSH Connection to the cluster

The most basic option to connect to the cluster system is via SSH. For this, an ssh client is required, which comes with most Linux distributions by default and also should be readily available under Mac OS (start a terminal, applications::utilities).

Under Windows, you'll need to install an [SSH Client](#) like e.g. [PuTTY](#) or much more feature-rich and user friendly [MobaXTerm](#). You could also install a Unix-like environment like [Cygwin](#) to benefit from some Unix-like functionality under Windows, but be aware that for many use cases, there may be easier ways to get you started (see below).

When you've found a terminal, the following command will establish a connection to the cluster system.

```
ssh username@login.cluster.uni-hannover.de
```

Replace username with your cluster user name.

**Please note:** The settings dialogue of a tool like PuTTY itself is *not* a terminal, it's only a settings dialogue. So PuTTY only needs to know the hostname you want to connect to, it executes the ssh command automatically when connecting. It will first ask for your username, except if you entered username@ in front of the host name, which would also be ok. It is sufficient to enter "login.cluster.uni-hannover.de" into the "Host name" field, make sure to NOT add "ssh" in front of it.

If you want to use graphical programs on the cluster system, add the option -X, which enables X11 forwarding (see below). Depending on your system, you may need to use -Y instead of -X.

```
ssh -X username@login.cluster.uni-hannover.de
```

**Please note:** Again, tools like e.g. PuTTY have their own setup. To get X11 forwarding with these tools, use the corresponding button. In the PuTTY configuration settings, this would be Connection::SSH::X11::Enable X11 forwarding.

## Remote Desktop Sessions via X2Go

One way to enable the usage of graphical programs on the cluster is to run an X-Server on your local machine, which means nothing more than running a piece of software that other software (called X-Clients in this context, but really just the graphic programs that you start) can make graphic requests to. Conveniently, an X-Server is usually the foundation for graphic environments under Linux so it is usually already installed here, and graphical connections tunneled through ssh (via ssh -X or ssh -Y, see above) should work right out of the box, but Mac OS users will typically need to install an external package such as [XQuartz](#). Under Windows, MobaXterm comes with an X-Server, while for PuTTY there is [VcXsrv](#).

For improved performance, however, it may still be useful to install and configure the X2Go client, as connections made via X2Go are compressed and optimized. The standard "X" protocol is not designed for use over wide-area networks, and newer tools can improve performance quite a bit. The X2Go client is part of most Linux distributions' package repositories.

On Windows, X2Go-client version [4.1.2.0](#) has been tested to work with the cluster.

X2Go can be obtained [here](#). **During installation, please make sure to check the "full installation" box to install all fonts. Otherwise, you may experience strange error messages later on.**

Please configure the X2Go client as described in [X2Go client configuration](#).

### X2Go client configuration

**Please note:** We recommend using X2Go in broker mode to avoid having multiple desktop sessions. See section about [X2Go Broker](#).



After starting the X2Go client, either using a desktop short-cut or using the start menu, a configuration dialogue is displayed. In this dialogue you should specify a session name and make the following four entries.

1. **Host:** login.cluster.uni-hannover.de
2. **Login:** Your user name
3. **SSH-Port:** 22
4. **Session type:** XFCE

The completed configuration dialogue is depicted in [figure 1](#). Entries in the red boxes have to be set accordingly. Afterwards leave the configuration assistant by clicking the **OK** button.

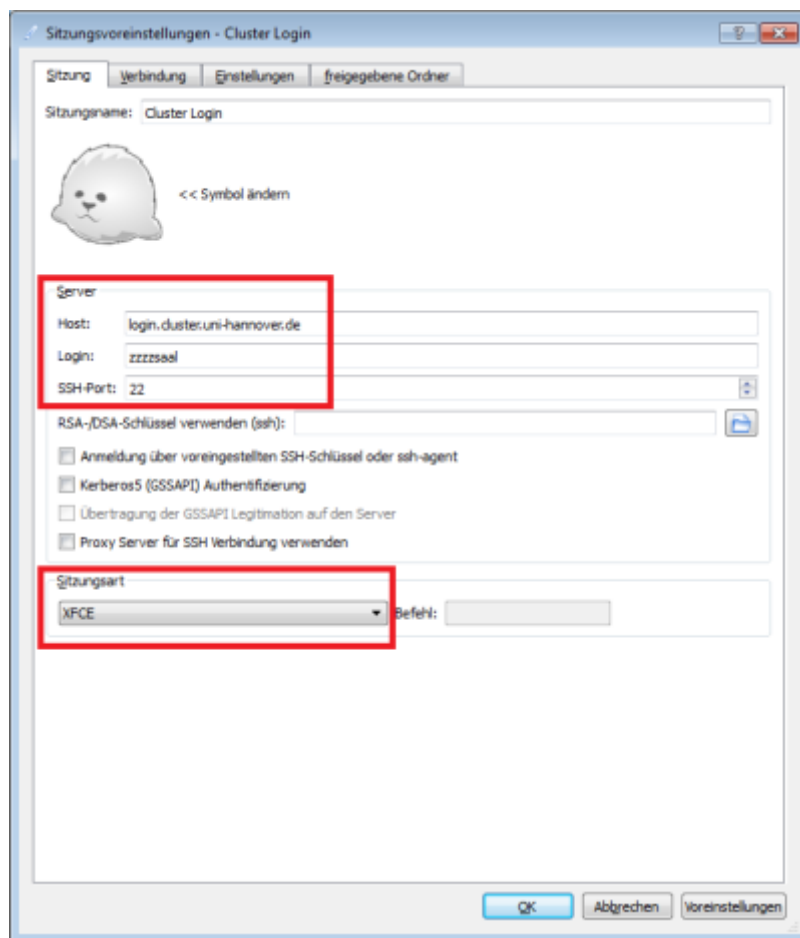


Fig. 1: X2Go configuration dialogue, entries in red boxes have to be set

On the right side of the main window the newly created session name is displayed, see [figure 2](#). You can start this session by clicking on the session name (in the upper right corner in [figure 2](#)) or by entering the session name in the dialogue box named **session**.

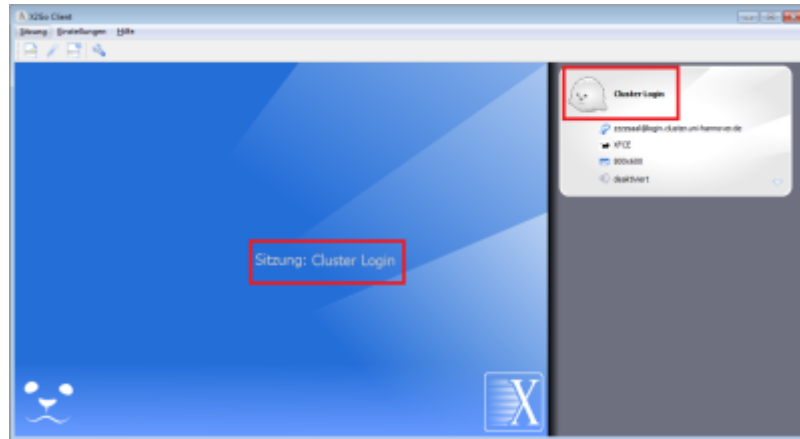


Fig. 2: Start a new session by clicking or entering the session name

The first time a connection is established, the login nodes' host-key is unknown. A notification will pop up and you need to accept the host-key (see figure [figure 3](#)) by pressing **yes**.

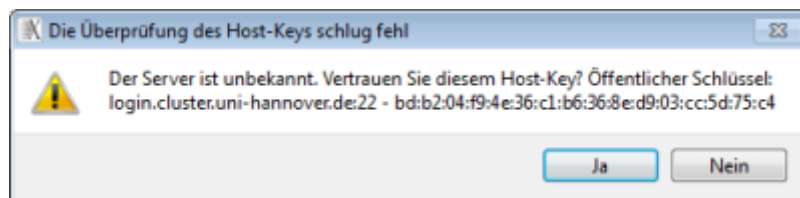


Fig. 3: Host key verification dialogue

After a connection was successfully established an XFCE Desktop is displayed as depicted in figure [figure 4](#).

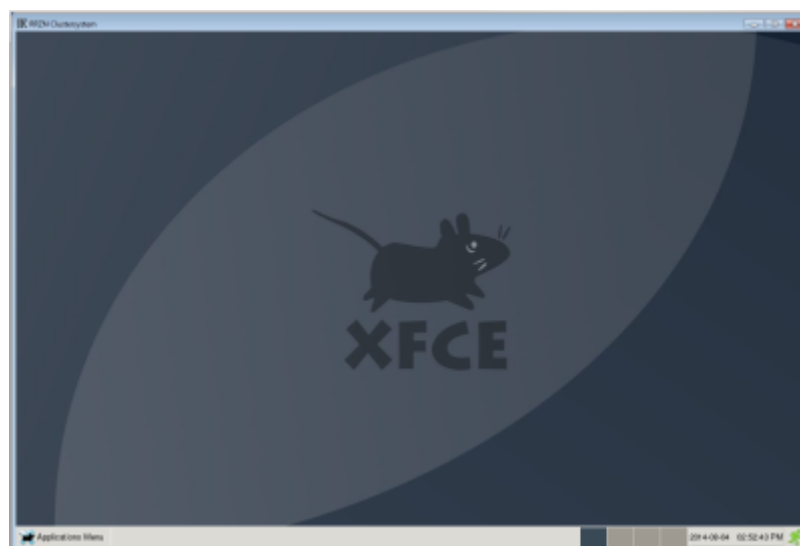


Fig. 4: XFCE desktop with applications menu in the lower left corner

The *Applications Menu* in the bottom left corner can be used to start a console window and then load modules or submit jobs into the queue. You can open editors e.g. to write or edit batch scripts. Particularly interactive jobs which open graphical program windows can be run. To end your session either go to the *Applications Menu* or press the little green icon in the bottom right corner of your desktop.

When connecting to the cluster using `login.cluster.uni-hannover.de` you will be assigned to one of the login nodes used for the cluster to balance the connection load. Therefore, it may happen

that by next connection you will be taken to a different login node rather than to a node you already have a running X2Go session. In order to avoid maintaining multiple simultaneously running sessions, which easily may lead to confusion, we recommend using X2Go client in broker mode, see the next section.

**Note:** Suspended X2Go sessions will be terminated after four weeks without prior notice.

**Note:** We had an issue that made it impossible to enter special characters using AltGr in a terminal within an X2Go session. We hope that this is now solved.

## X2Go Broker

If you would like to reconnect to a graphical session, use X2Go broker. For example you could start a session at the university and reconnect to it at home. In order to do this, you have to establish a connection to the cluster via X2Go broker. After installing X2Go client, proceed as described below.

### X2Go broker on Linux

Use the following command to establish a connection with X2Go broker (the following command should be on one line. Replace <username> with your cluster username).

```
x2goclient --broker-url=ssh://<username>@x2gobroker.cluster.uni-hannover.de/usr/bin/x2gobroker --broker-autologin
```

### X2Go broker on Windows

Either edit the existing shortcut to X2Go or create a new one (chose "Eigenschaften" in Picture [figure 5](#)).

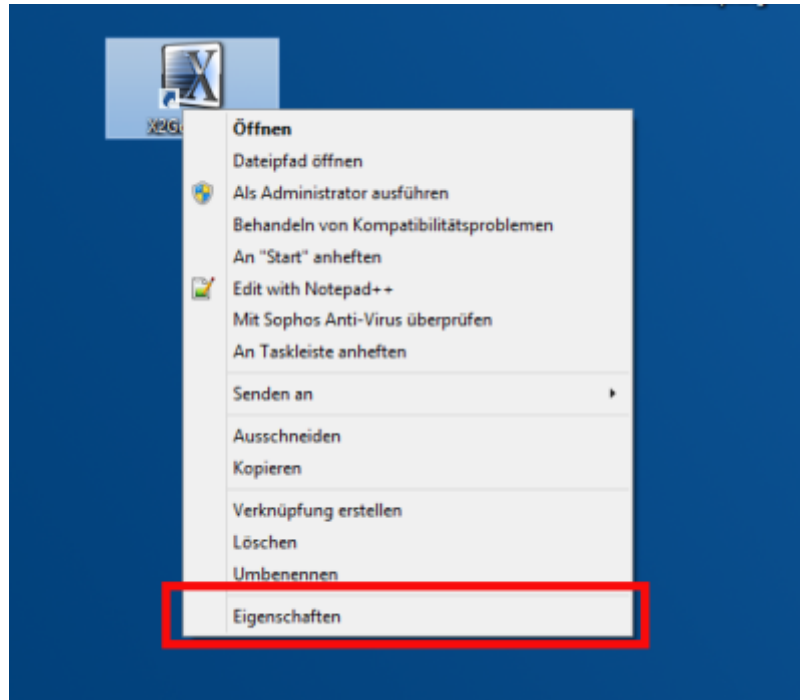


Fig. 5: Edit X2Go shortcut

Extend the command given as "Ziel", see Picture [figure 6](#), with the following parameters (the following command should be on one line. Replace <username> with your cluster username).

```
"[...]x2goclient.exe" --broker-url=ssh://<username>@x2gobroker.cluster.uni-hannover.de/usr/bin/x2gobroker
```

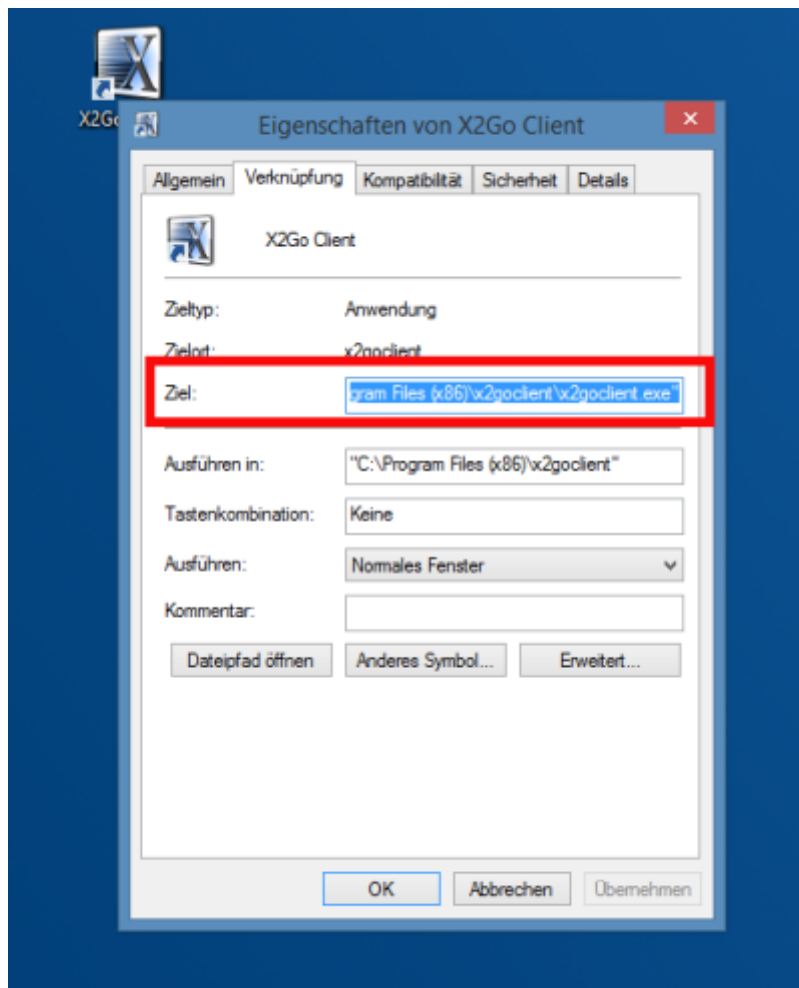


Fig. 6: X2Go broker command

After providing your password, a session is listed in the X2Go window, see picture . Choose this session. You will get a desktop on the cluster system. You can reconnect to this session and continue working graphically.

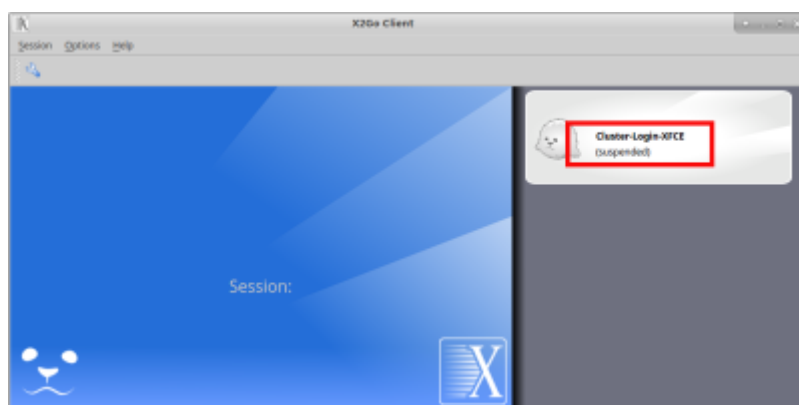


Fig. 7: X2Go broker session

**Please note:** Suspended X2Go sessions will be terminated after four weeks without prior notice.

## Cluster Web Portal

The web interface powered by the software package Open OnDemand allows you to access the LUIS

cluster resources using a web browser without installing additional software on your personal workstation. The portal is currently confirmed to work with newer versions of Google Chrome (70+), Firefox (85+) and Microsoft Edge (90+). Any device connected to the university network and compatible with these browsers can be used. Safari is partially supported.

From within the Open OnDemand environment, you can:


- Create, submit, cancel and monitor batch jobs
- Open a terminal connection to cluster login servers and compute nodes
- Browse, edit, download and upload files in/to your HOME, BIGWORK and PROJECT directories
- Run noVNC Remote Desktop sessions on compute nodes for heavy GUI applications
- Run other preconfigured interactive applications like Jupyter/JupyterLab, MATLAB, COMSOL, etc

The Open OnDemand [website](#) provides additional information about the current and future directions of the project.

## How to connect to the web portal

**Please note:** To access the portal, make sure you are connected to the University network, e.g. via the [LUIS VPN Service](#).

Log on to the cluster web portal using your normal cluster login credentials by opening a new page in your web browser pointing to <https://login.cluster.uni-hannover.de> (see figure [figure 8](#)).



**Leibniz Universität**  
IT Services

### Cluster Web Portal

Provide your cluster username and password

Username

Password

Login

**Caution:** Never enter your username and password on a web page unless the page is directly served by [weblogin.cluster.uni-hannover.de](https://weblogin.cluster.uni-hannover.de) server

[About LUIS Computing Cluster](#)

[Registration for LUIS Computing Cluster](#)

Fig. 8: Cluster web portal login page

Once you have been connected to the portal, you will be presented with the main dashboard page, see figure [figure 9](#). There, you will find several menus to enable access to the different Applications for File Browsing, Job Management and Interactive Computing. Tutorial videos in the section “Getting started with OnDemand” in the dashboard page (click on “LUIS OnDemand Dashboard” on the top left of the web page) explain further details about using the web portal.

**Web Access to LUIS Research Computing Cluster**

### Getting started with OnDemand

From this web portal you can

- Create, submit, cancel and monitor SLURM batch jobs
- Open a terminal connection to cluster front-end login servers and compute nodes
- Browse, edit, download and upload files in your HOME, BIGWORK and PROJECT directories
- Run noVNC Remote Desktop sessions on compute nodes for heavy GUI applications
- Run other preconfigured interactive applications like Jupyter/JupyterLab, MATLAB, COMSOL, etc

The site is currently confirmed to work with newer versions of Google Chrome (70+), Firefox (85+) and Microsoft Edge (90+). Any device connected to the university network and compatible with these browsers can be used. Safari is partially supported.

Tutorial videos below give an overview of OnDemand's capabilities which are functional at the LUIS cluster web portal.

**File Management and Transfer** [show tutorial](#)  
The menu **Files** provides a web-based File Explorer to upload, download and manage files in your HOME, BIGWORK and PROJECT directories. **Note:** the maximum size of each uploaded file must not exceed 1 GB

**Job Management and Monitoring** [show tutorial](#)  
In the menu **Jobs** OnDemand offers two Apps: **Active Jobs** provides you with the status of your current jobs, while **Job Composer** allows you to edit and submit jobs via your web browser

**Terminal Connection to Login Servers** [show tutorial](#)  
You can get shell access to one of the cluster login nodes by choosing the menu **Clusters**

### News

- 08-10-2021** - Mathematica as interactive application
- 06-10-2021** - Spyder Python IDE as interactive application
- 06-10-2021** - Manage files also in BIGWORK and PROJECT from the **Files** menu

Fig. 9: Cluster web portal dashboard page

## Services available via Open OnDemand

- [Remote visualization for interactive 3D work](#)
- [Jupyter in the cluster](#)

From:  
<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:  
[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/150\\_connecting\\_to\\_cluster](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/150_connecting_to_cluster)

Last update: **2025/05/22 12:16**



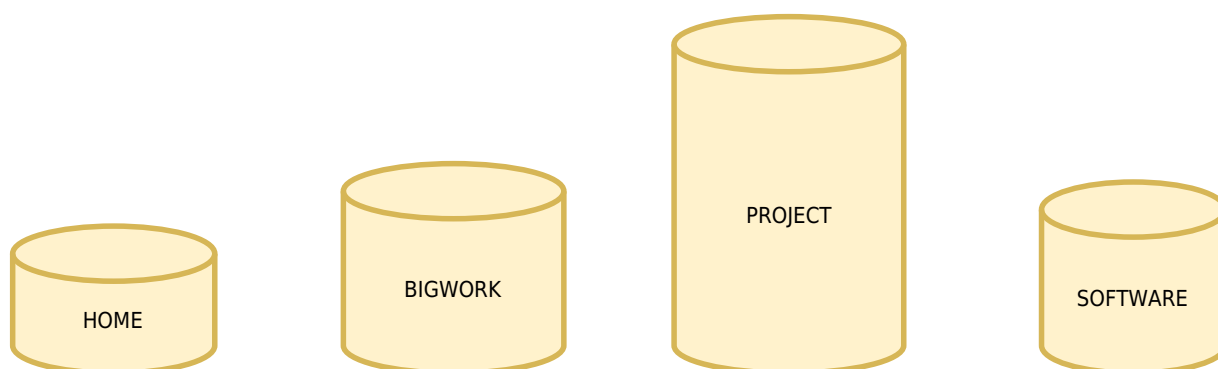
# File systems in the cluster

As you may have guessed, there are several file systems serving the cluster.

Each of these file systems has its own characteristics, making it the choice for a different workload.

Under Unix/Linux, file systems are simply *mounted* as “subdirectories” somewhere under the root of the file system (or “the root of the directory tree”), which is symbolized by the / (slash character). So, in case you are coming from a Windows environment, the first thing you'll notice is that there is no such thing as drive letters (C:, D:, etc.). Logically, everything appears to be in the same file system. You use a different file system by simply changing into its directory. File systems may be mounted almost anywhere in the file system, not just at the top. Use your favorite search engine to learn more about how unix organizes files in case you are not yet familiar. You'll need it.

For ease of use and consistency of access, we provide some automatically set variables called HOME, BIGWORK, SOFTWARE, PROJECT and TMPDIR that either point to your own directories in the file systems that are mounted under /home, /bigwork, /software and /project, or to the scratch space local to each compute node (/scratch). You can access the content of a variable by putting a '\$' sign in front of its name, like in the commands `echo $HOME` or `cd $BIGWORK`. To make it clear that those are variables, we'll refer to them including the dollar sign.



/home/USERNAME	/bigwork/USERNAME	/project/GROUPNAME	/software/GROUPNAME
10 GByte soft quota	100 GByte soft quota	10 TByte soft quota	50 GByte soft quota
12 GByte hard quota	1 TByte hard quota	12 TByte hard quota	100 GByte hard quota
backup	no backup	no backup	no backup
10 Gigabit Ethernet	Infiniband	Infiniband	10 Gigabit Ethernet

Figure 1: The most important file systems with specifications

## How to use the file systems

The following sections describe the characteristics and best utilization of each of the cluster's



filesystems.

## **\$BIGWORK - where you work**

Your main working directory. It provides a comparatively high amount of storage space. Due to the fact that large amounts of data are quickly changing in this storage, it is impossible to provide a backup. All computations should be done here, except for jobs requiring \$TMPDIR for a reason (see section \$TMPDIR). \$BIGWORK is connected via the InfiniBand network interface and thus inherently much faster than \$HOME. It is also being provided by a whole group of servers to increase performance. Bandwidth and scalability for multi-core jobs usually is very good, but performance for many small files is mediocre at best (for this use case, see \$TMPDIR). Both the amount of data and the number of files that you can store are limited (see below, [Quota and grace time](#)).

\$BIGWORK is mounted on all nodes in the cluster (all nodes see the same \$BIGWORK file system).

## **\$PROJECT - data sets and results of a group, but no access in compute jobs**

Your (and your colleagues') project directory. Each project gets assigned a project directory to which all members of a project have read and write access at `/project/<your-cluster-groupname>` (the environment-variable \$PROJECT points to this location). In order to store individual user accounts' data in this directory in such a manner that everyone can keep track of what belongs to whom, we suggest each account create their own sub directory named `$PROJECT/$USER` and set suitable access rights (like `mkdir -m 0700 $PROJECT/$USER`). How you best organize your data within your group is left up to you, however. The group's quota for the project directory is usually set to 10 TB.

\$PROJECT is ONLY mounted on the login and the transfer nodes, but NOT on the compute nodes, since the hardware serving it is quite modest. So you CAN NOT USE this file system IN YOUR JOBS (use \$BIGWORK instead).

Copying between \$BIGWORK and \$PROJECT is thus only possible on the transfer or the login nodes, where a fast connection between both file systems is available. It is intended to separately store files that are important for the work of an institute on the cluster, for preparation, curation and long time retention of both input data, results and setups. It is configured in such a way that all data are physically written in two copies. Due to the amount of data, however, no additional backup is provided.

We regard data in \$PROJECT as belonging to the institute or the project, respectively. If you want to make sure that the project management at your institute can still access your data after you have left the LUH, put a copy of your project results here, and we can easily give access to the person in charge of the project.

**Please note:** Backing up your data regularly from \$BIGWORK to \$PROJECT storage and/or to your institute's servers is considered essential, since \$BIGWORK is designed as scratch file system. \$PROJECT should also NOT be considered being a real backup as well, since it is also disk based. Disks may fail, and computing centers may also suddenly burn to the ground when you least need it.

## **\$HOME - configuration files and setups. "Don't do big work in your home", though (pun intended)**

Your home directory is the directory in which you are normally placed directly after a login from the command line. You should only place small and particularly important files here that would be especially time-consuming to create anew, like templates for job setups, important scripts, configuration files for software etc. DO NOT PUT DATA HERE that you use in compute jobs and DO NOT USE THIS DIRECTORY TO WRITE RESULTS. \$HOME is provided by only one server which intentionally is connected via relatively slow gigabit ethernet to the remainder of the system (to protect the server from becoming overloaded too quickly). In comparison to \$BIGWORK, this file system is very slow and particularly unsuitable to handle data intensive tasks. If you succeed overloading \$HOME, all users will notice.

If you overstep your quota (see following section) in this directory, – which will happen quickly if you do not heed the above advice and let your compute jobs run and put data here – you will make yourself unable to log in graphically with tools like X2Go. You will then need to delete data before you can continue to work. In case you run into that situation, do not make matters worse by simply deleting “anything”, because there are important files here that e.g. give you access to parts of the system. If you just indiscriminately delete files here, you'll definitely embarrass yourself and need to ask for administrative assistance next, which takes *much* longer than if you had instead taken some time to check which files were created by a job of yours. On the other hand, if we note that you did not read this documentation at all, we will surely send you a link to it.

So you should only place important and small files in your \$HOME that would be particularly difficult and laborious to re-create (like configs etc.), but meticulously avoid accessing data here from within compute jobs. If you disregard that rule, your jobs will quite probably not only take much longer to complete, but at the same time also impede other users' work, due to the technical properties described above. You should point all directories that possibly have been set automatically by the software you install/use – quite often e.g. temporary directories – to \$BIGWORK. Do NOT use symbolic links between \$BIGWORK and \$HOME in a compute job to creatively cheat around the circumstances, since these links need to get looked up as well. Use the environment variable \$BIGWORK for convenient access in a computation, and avoid putting anything in your \$HOME that can go somewhere else. Do NOT place pip, (ana)conda, CRAN/R or similar installations here. Use your \$SOFTWARE directory and, if possible, share with your colleagues. Also have a look at the [exercise](#).

Due to the limits in the amount of data and implicitly only slowly changing data, we have a daily backup of data in \$HOME.

**Memory Aid:** “Never WORK at HOME”. It simply is not built for that, and others will feel the impact of what you are doing. Do not abuse \$HOME as a backup directory for your compute results. That's a really bad idea.

\$HOME is mounted on all nodes in the cluster (all nodes see the same \$HOME file system).

## **\$TMPDIR - local to each node, different disk types**

Within jobs, the variable \$TMPDIR points to local storage available directly on each node. Whenever local storage is needed, such as when you are handling thousands of files that would make you hit your \$BIGWORK inode limit, \$TMPDIR should be used. The second reason may be that your jobs due to heavy parallelization are now I/O-bound and thus need a node that is specifically equipped with

local fast SSDs. Only a few nodes are equipped with these SSDs yet, and they usually belong to an institute that bought them within our FCH service, so that is a limited option for those who do not have access to a specific FCH partition.

In general, do not simply assume `$TMPDIR` to be faster than `$BIGWORK`. `$TMPDIR` can be used for temporary files for applications that *imperatively* require a dedicated temporary directory. There will probably be a performance improvement if you have many very small files and use `$TMPDIR`. You may get a big performance boost if you can use a node that is equipped with a local SSD scratch disk, see below, but most of the nodes in 2025 are still equipped with traditional HDDs. Lustre (`BIGWORK`) is not good at handling large numbers of small files, which is the reason that the number of files is strictly limited on `BIGWORK`. The only option for these jobs is to pack datasets containing millions of files into archive files, e.g. using the `tar` command and then “locally” extract them on the fast SSD scratch disk before the job starts. An example how to do this would be the pair `tar -cf <archive_file.tar> <directory_to_archive>` and `cd $TMPDIR ; tar -xf $BIGWORK/archive_file.tar`. In case of easily compressible data (for example text files, but usually NOT jpg files), you may try using mild compression to possibly increase bandwidth from disk and optimize the size of your archive, but if you are using millions of small jpegs, it's better not to do that — those are already relatively well compressed, and you'll usually not gain much in terms of file size, but it may cost you additional time to decompress. You may also want to have a look at how striping works on Lustre. See below for an explanation.

**`$TMPDIR` is strictly local to each node.** This means that **if you write data to the local `$TMPDIR` on one node, it will NOT be visible on any other node in the cluster.** If your job occupies two or more nodes, the tasks on each node will see a different local directory. You should have an idea of the size of `$TMPDIR` first, which you can easily check *before* submitting your jobs using the command `scontrol show nodes <nodename_here>`, or (of course) within your job using `df -h $TMPDIR`. The parameter “`TmpDisk`” shows you the configured size of the scratch/`$TMPDIR` partition in MiB.

On the Dumbo partition, we have an intentionally large `$TMPDIR` partition (16 TB) made up of classic (mechanical) HDD drives.

Most of the other nodes only have about 100 GB of local storage.

Some FCH partitions (nodes we run for various institutes) have expressly been equipped with large scratch SSDs by their owners due to their particular work load (many small files that can not reasonably be stored in Lustre). Even if you do not belong to one of these institutes and thus have no access to their daily 12-h-reservation, you may also submit your jobs to these nodes and run them at night (up to 12 hour-jobs) and during weekends (max. 60 hours), when the nodes are not reserved and thus participate in the general queue. Be aware, however, that in case the owners really put a high number of long-running jobs on their partition (which they are, of course, perfectly entitled to), and if other users have similar demands, your jobs may spend some time in the queue.

**Please note:** As soon as a job finishes, all data stored under `$TMPDIR` will be deleted automatically.

## **`$SOFTWARE` - install software for your group here**

This directory enables a project to install their own software in a way so the whole group can use it. You should also use this directory (or a personal subdirectory you create here) for installations done via `pip`, [conda](#), R/CRAN and the like.

`$SOFTWARE` points to a cluster-wide shared directory that is placed in `/software/<your-`

`cluster-groupname>`. If you want to narrow down the list of usernames in your project that should have access (e.g. read) to your software installation directory, we recommend creating a subdirectory that only your account can access using the command:

```
mkdir -m 0700 $SOFTWARE/mysoft
```

Thereafter, you may grant additional usernames the access you desire using an access control list (ACL):

```
setfacl --mask --modify user:<username>:rx,default:user:<username>:rx  
$SOFTWARE/mysoft
```

Check the ACL you set using

```
getfacl $SOFTWARE/mysoft
```

See `man acl`, `man setfacl` and `man getfacl` for more information about ACLs. Subdirectories created in `$SOFTWARE` can only be removed by the user who created them (the “sticky bit” is set for this directory) to prevent colleagues with no read/execute access to subdirectories from accidentally destroying your work.

In case you are the project manager of a project with us and your institute has special requirements (like, only a few persons should be able to install/maintain software in this directory), talk to us and we'll change ownership to the account of the project manager's account so you can completely decide for yourself whom to grant which permissions. In this case, you should take extra care that file permissions are set properly to make sure only members of your group can write into or have access to the directory so the executables you'll find here stay your own. Whomever you grant access may change files here.

The group's quota for the software directory is set to 50 GB. There's also no backup at this time. It would be an exceptionally bad idea to abuse `$SOFTWARE` to store job results. Not only because it's a shared and fairly limited resource in terms of storage, but also since this directory is only being served via Ethernet by just one server and thus again not suitable for massively parallel workloads (similar to `HOME`, but not quite as disastrous when overloaded). But it's a way to give you some storage for installations that you might possibly share in your work group, and it somewhat mitigates the problem that `$HOME` must be so strictly limited. It also keeps the many small files that some software installations create away from Lustre (`$BIGWORK`) to keep the load created by many metadata operations as small as possible there.

## Quota and grace time

On the shared storage systems (i.e. `$HOME`, `$BIGWORK` and `$PROJECT`), only a fraction of the whole disk space is made available to you or your account, respectively. This amount is designated as *quota*. The purpose is to ensure everybody gets a share of the resources, and also to limit the effects of accidents that otherwise could easily fill up the whole system. In case your current quota on `$BIGWORK` or `$PROJECT` would severely impede your work, let us know what you would need, stating account name, amount needed and the time frame (i.e. for how many months you would need an increased quota).

There is a *soft quota* and a *hard quota*. For each of these two quota types, two independent parameters are set, namely *block quota* (amount of gigabytes you can store) and *inode quota* (number of files you can create). So there are four values you need to consider.

The *hard quota* is the absolute upper bound which you can not exceed until an administrator raises the limit. The *soft quota*, on the other hand, may be exceeded for some time, the so-called *grace time*.

Exceeding any of the two parameters (*block* or *inodes*) of your *soft quota* starts the respective *grace timer*.

During the *grace time*, you are allowed to exceed your *soft quota* up to the limits of your respective *hard quota*. After the grace timer has run out, you will not be able to store any new data, unless you reduce disk space usage below the *soft quota*. *Since we get asked from time to time: no, the system will NOT delete random files of yours after the grace period has expired. What is on disk stays on disk. You will just not be able to write new data as long as you are over the soft limit when the grace period is over.*

As soon as your disk consumption falls below the *soft quota* again, the *grace time* counter for that file system and that parameter is reset. On \$HOME and \$BIGWORK, quotas are valid for each individual user, whereas on \$PROJECT, quotas are accounted for the whole group (the project you are a member of, usually a four-letter code followed by a five-digit number; use the `id` command to check).

For \$BIGWORK, \$PROJECT and \$SOFTWARE, the relevant (user/group) *block* and *inode* quota grace times are 30 days. On \$HOME, grace time is 10 days.

The *quota* mechanism protects users and system against possible errors of others, limits the maximum disk space available to an individual user, and keeps the system performance as high as possible by avoiding unnecessary clutter. In general, we ask you to delete files which are no longer needed to keep the file systems fast. Low disk space consumption is especially helpful on \$BIGWORK in order to optimise system performance. You can query your disk space usage and *quota* with the command `checkquota` – see also [exercise](#). In case the standard system quota values are too low for you, talk to us and tell us which account needs how many GB for how long (and why) – we try to accommodate requests within reason. The quota values are set such that usually 90 % of our users should not feel much of limits, except for the usual need to periodically clean up and backup results.

**Please note:** If your quota is exhausted on \$HOME, you will not be able to login graphically using X2Go any more. Connecting using `ssh` (without `-X`) will still be possible.

## Lustre file system and stripe count

**Please note:** All statements made in this section also apply to \$PROJECT storage.

On the technical level, \$BIGWORK is comprised of multiple components which make up the storage system. Generally speaking, using \$BIGWORK without changing any default values should usually work well. However, it may be useful under certain circumstances to change the so called *stripe count*, in particular if you need to handle files larger than at least 1 GB, or if you access different parts of the same file in highly parallel applications running on several nodes. Balancing I/O, investing some time in understanding Lustre and testing different setups may result in a higher performance and a better-balanced use of the overall system, which in turn is beneficial for all users.

Data on Lustre-based file systems such as \$BIGWORK is saved on OSTs, *Object Storage Targets*. The

*location* of the data (which OST(s)?) is registered with a directory server (“MDS”) and stored on a MDT, the *Meta Data Target*. Each OST and the MDT as well looks like one large hard drive (a “block device”) to the system. In reality, each Target consists of several hard drives bundled together in the background by a storage controller to ensure e.g. some fault tolerance and higher performance, but this is not visible to the computers using the file system. The first takeaway message here is that Lustre stores files on several things that look like separate hard drives, which may be taken into consideration when you need more performance, but together the whole thing just looks like another file system in a subdirectory (\$BIGWORK in this case).

By default, files are written to a single OST each, regardless of their size. This corresponds to a *stripe count* of one. The *stripe count* determines how many OSTs will be used to store data. Figuratively speaking, it determines in how many stripes a file is being split (like a Zebra, but possibly with more than just two colors...). Splitting data over multiple OSTs can increase access speeds, since the read and write speeds of several OSTs and thus a higher number of hard drives are used in parallel. At the same time, one should only distribute large files in this way, because access times can also increase if you have too many small requests to the file system. Depending on your personal use case, you may need to experiment to find the best setting. When doing this, consider that the current workload of the whole system may influence your results.

**Please note:** If you are working with files larger than 1 GB, and for which access times e.g. from within parallel computations could significantly contribute to the total duration of a compute job, please consider setting *stripe count* manually according to [section](#).

*Stripe count* is set as an integer value representing the number of OSTs to use, with -1 indicating all available OSTs. It is advised to create a directory below \$BIGWORK and set a *stipe count* of -1 for it. This directory can then be used e.g. to store all files that are larger than 100 MB. For files significantly smaller than 100 MB, the default stripe count of one is both better and sufficient.

**Please note:** In order to alter the of existing files, these need to be copied, see [section](#). Simply moving files with mv is not sufficient in this case.

## Exercises

In this section we consider several examples on working with the cluster storage systems.

### Using file systems

```
# where are you? lost? print working directory!
pwd

# change directory to your bigwork/project/home directory
cd $BIGWORK
cd $PROJECT
cd $HOME

# display your home, bigwork & project quota
checkquota
```

```
# make personal directory in your group's project storage
# set permissions (-m) so only your account can access
# the files in it (0700)
mkdir -m 0700 $PROJECT/$USER

# copy the directory mydir from bigwork to project
cp -r $BIGWORK/mydir $PROJECT/$USER
```

## Setting stripe count

```
# get overall bigwork usage, note different fill levels
lfs df -h

# get current stripe settings for your bigwork
lfs getstripe $BIGWORK

# change directory to your bigwork
cd $BIGWORK

# create a directory for large files (anything over 100 MB)
mkdir LargeFiles

# get current stripe settings for that directory
lfs getstripe LargeFiles

# set stripe count to -1 (all available OSTs)
lfs setstripe -c -1 LargeFiles

# check current stripe settings for LargeFiles directory
lfs getstripe LargeFiles

# create a directory for small files
mkdir SmallFiles

# check stripe information for SmallFiles directory
lfs getstripe SmallFiles
```

Use newly created LargeFiles directory to store large files

## Altering stripe count

Sometimes you might not know beforehand, how large files created by your simulations will turn out. In this case you can set *stripe size* after a file has been created in two ways. Let us create a 100 MB file first.

```
# enter the directory for small files
cd SmallFiles
```



```
# create a 100 MB file
dd if=/dev/zero of=100mb.file bs=10M count=10

# check filesize by listing directory contents
ls -lh

# check stripe information on 100mb.file
lfs getstripe 100mb.file

# move the file into the large files directory
mv 100mb.file ../LargeFiles/

# check if stripe information of 100mb.file changed
lfs getstripe ../LargeFiles/100mb.file

# remove the file
rm ../LargeFiles/100mb.file
```

In order to change stripe, the file has to be copied (cp). Simply moving (mv) the file will not affect stripe count.

First method:

```
# from within the small files directory
cd $BIGWORK/SmallFiles

# create a 100 MB file
dd if=/dev/zero of=100mb.file bs=10M count=10

# copy file into the LargeFiles directory
cp 100mb.file ../LargeFiles/

# check stripe in the new location
lfs getstripe ../LargeFiles/100mb.file
```

Second method:

```
# create empty file with appropriate stripe count
lfs setstripe -c -1 empty.file

# check stripe information of empty file
lfs getstripe empty.file

# copy file "in place"
cp 100mb.file empty.file

# check that empty.file now has a size of 100 MB
ls -lh

# remove the original 100mb.file and work with empty.file
```



```
rm 100mb.file
```

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/200\\_storage\\_systems](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/200_storage_systems)

Last update: **2025/04/08 16:14**



# Handling large datasets within the cluster

While working on the cluster, you may need to perform operations such as *copy*, *delete*, *sync*, *find*, etc. on a very large number of files or on files that are very large in size. While we do recommend to first have a look whether you can reduce the number of files you need to handle e.g. by packing them (because both the file system Lustre that is driving BIGWORK and the controllers of the disk drives have limits how many files they can handle at once. That means that if you need more IOPs, the overall performance you get may deteriorate significantly), we also realize that this frequently is more a long-term project. For this reason, we provide some parallel tools provided by the MPI-based package [mpiFileUtils](#). The standard Unix commands like `cp`, `rm` or `find` are often comparatively slow, as they are implemented as single process applications.

As a typical example, consider copying directories containing a large number of files from your `$BIGWORK` to the local `$TMPDIR` storage on the compute nodes you allocated for further processing by your job, or/and transferring computation results back to your `$BIGWORK`.

Another example would be a quick freeing up of space in your `$BIGWORK` by first copying files to your `$PROJECT` storage and then deleting them from `$BIGWORK`.

Also, you could utilize the command `dsync`, if you use your `$PROJECT` storage for backing up directories on your `$BIGWORK`.

Below we will look at some of the `mpiFileUtils` tools in these and other practical examples.

In order to speed up the recursive transfer of the **contents** of the directory `$BIGWORK/source` to `$TMPDIR/dest` on a compute node, put the lines below in your job script or enter them at the command prompt of your interactive job - we assume that you've requested 8 cores for your batch job:

```
module load GCC/8.3.0 OpenMPI/3.1.4 mpifileutils/0.11
mpirun -np 8 dcp $BIGWORK/source/ $TMPDIR/dest
```

Please note a trailing slash (/) on the source path, which means “**copy the contents of the source directory**”. If the `$TMPDIR/dest` directory does not exist before copying, it will be created. The command `mpirun` launches `dcp` (distributed copy) in parallel with 8 MPI processes.

The directory `$BIGWORK/dir` and its contents can be removed quickly using the `drm` (distributed remove) command:

```
mpirun -np 8 drm $BIGWORK/dir
```

**Note:** here and below we assume that the module `mpifileutils/0.11` is already loaded.

The command `drm` supports the option `--match` allowing to delete files selectively. See `man drm` for more information.

The next useful command is `dfind` - a parallel version of the unix command `find`. In this example we find all files on `$BIGWORK` larger than 1GB and write them to a file:

```
mpirun -np 8 dfind -v --output files_1GB.txt --size +1GB $BIGWORK
```

To learn more about other dfind options, type `man dfind`.

If you want to synchronize the directory `$BIGWORK/source` to `$PROJECT/dest` such that the directory `$PROJECT/dest` has content, ownership, timestamps and permissions of `$BIGWORK/source`, execute:

```
mpirun -np 8 dsync -D $BIGWORK/source $PROJECT/dest
```

Note that for this example the `dsync` command has to be launched on the login node where both `$BIGWORK` and `$PROJECT` are available.

The last `mpiFileUtils` tools we consider in this section are `dtar` and `dbz2`. The following creates a compressed archive `mydir.tar.dbz2` of the directory `mydir`:

```
mpirun -np 8 dtar -c -f mydir.tar mydir
mpirun -np 8 dbz2 -z mydir.tar
```

**Please note:** If the directory to be archived is located on your `$HOME`, the archive file itself should be placed on `$BIGWORK`.

**Please note:** Transferring a large number of files from the cluster to an external server or vice versa as a single (compressed) tar archive is much more efficient than copying files individually.

Some other useful commands are:

- `dstripe` - restripe(lustre) files in paths
- `dwalk` - list, sort, summarize files
- `ddup` - find duplicate files

A complete list of `mpiFileUtils` utilities and their description can be found at <http://mpifileutils.readthedocs.io/>.

You might also consider alternative tools like [GNU parallel](#), [pigz](#) or [pcp](#). They are all available as modules on the cluster.

From:  
<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:  
[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/201\\_handling\\_large\\_datasets](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/201_handling_large_datasets)

Last update: **2022/01/08 10:39**



# File transfer to/from the cluster

---

There is a special node dedicated to data transfer with the cluster system. Whenever you transfer data with the cluster system, please use this node:

```
transfer.cluster.uni-hannover.de
```

**Please note:** Use the dedicated transfer node for file transfers, because processes that use more than 30 minutes of cpu time on the login machines will be aborted by the system. Since ssh/scp use encryption, you may eventually use enough cpu time to get the transfer killed on the login nodes.

Files can be transferred to and from the cluster system in a number of ways. What has been said in the [section](#) about the availability of the tools on different operating systems also holds true here: the command line tools like scp, sftp or rsync are usually already available on Linux and Mac OS computers. For Windows, you may need to install additional software such as [MobaXterm](#) which offers a command line in addition to graphical interface for file transfer. You'll usually want to initiate the connection from your own workstation, since you need a running ssh server for most of the commands, and a Windows workstation usually does not have (should not have without good reason) that.

---

The following command, e.g., copies a whole directory mydata from the cluster to a local directory on your workstation:

```
[myworkstation]$ scp -r <username>@transfer.cluster.uni-hannover.de:/path/to/mydata .
```

Replace <username> above with your cluster user name.

---

An very efficient, but also very powerful and more demanding tool is rsync which can synchronize directories across sites and also complete interrupted file transfers. If you want to sync the contents of directory mydir (source) on your workstation with directory mydir (target) on your BIGWORK, run:

```
[myworkstation]$ rsync -av --delete /path/to/mydir/  
<username>@transfer.cluster.uni-hannover.de:/bigwork/<username>/mydir/
```

Watch out for the slashes '/' at the end of both source and target path, they are important. If in doubt, use --dry-run to check what you are about to do, and possibly create an extra subdirectory for sync'ing until you get familiar with the particular syntax. Whether you put a slash '/' at the end of source or target or not is interpreted as either "sync the *contents* of this directory" vs. "sync the directory itself (including everything in it)". So /path/to/mydir/ and /path/to/mydir have completely different meanings. Ask man rsync to get the complete picture.

The --delete option removes files in the target that are missing in the source directory, so the target is in sync with the source. Take care that you are in the correct subdirectory, or you may delete

local files that you still wanted. If in doubt, test without this option first.

Alternatively, you can use [FileZilla](#) if you want to use a graphical tool. FileZilla supports Linux, Mac OS, as well as Windows platforms. Information on how to configure FileZilla for use with the cluster system can be found in the next section.

## File transfer using FileZilla

The FileZilla client may be used to exchange files between your workstation and the cluster system in sftp mode. In the following section we provide instructions on how to install and configure FileZilla client version 3.14.1\_win64 on Windows, but the software is available for other platforms as well. FileZilla can be obtained from the following [URL](#). After downloading, you can install the FileZilla client to a directory of your choice.

After installing, open the *Site Manager* and create a new server which you can then connect to. The following options have to be set (cf. the red boxes in figure [figure 1](#)).

- **Host:** transfer.cluster.uni-hannover.de
- **Protocol:** SFTP - SSH File Transfer Protocol
- **Logon Type:** Ask for password
- **User:** Your user name

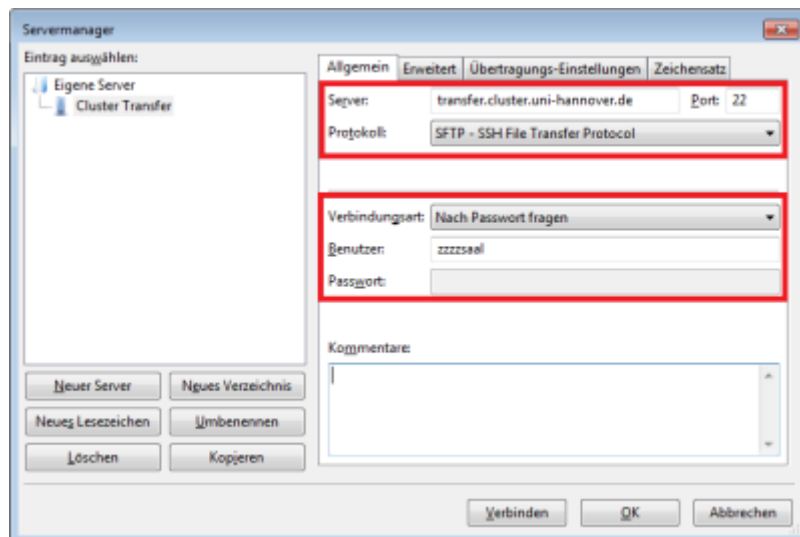


Fig. 1: Site Manager: General

Furthermore, it is possible to open the remote connection directly to \$BIGWORK. Without further configuration, the remote directory will be set to \$HOME. In order to configure this option, go to the *Advanced* tab and set *Default remote directory* accordingly, see figure [figure 2](#).

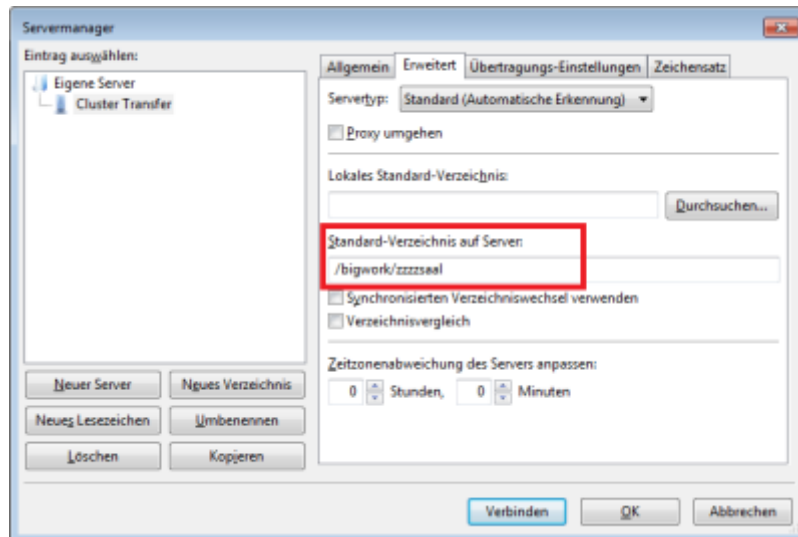


Fig. 2: Site Manager: Advanced

The first time a connection to the transfer node is made, you will need to confirm the authenticity of the node's host-key - cf. figure [figure 3](#).

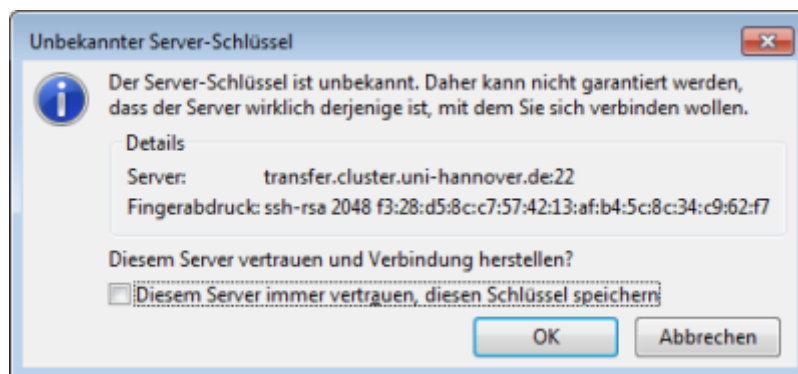


Fig. 3: Host-key verification on first connection attempt

After a connection is successfully established – cf. figure [figure 4](#) – you can exchange data with the cluster system.

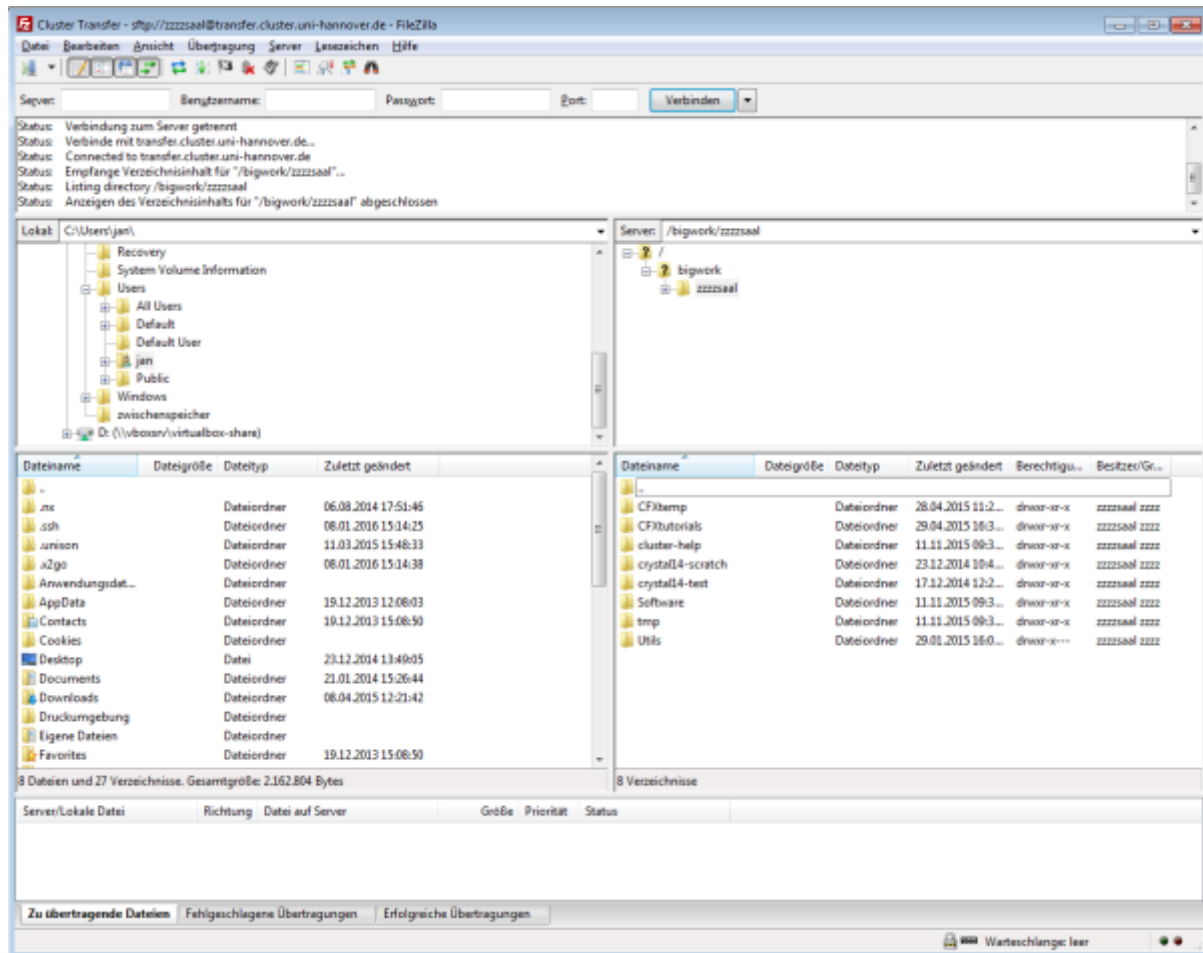


Fig. 4: Connection to transfer-node, established with FileZilla

If your own workstation runs Windows, you should now have a look into **Settings::Transfers::FTP File Types** and check that the transfer type selected will be correct for the files you will want to transfer. See the section ["Converting jobscripts written under Windows"](#)

In case you see different file sizes or if you experience corrupt files after the transfer, you may/will probably need to switch to "binary transfer mode". FTP is an old protocol that once took care that different encodings of ASCII text files were translated properly, and you'll quite probably don't want that for binary data files.

## File transfer using web browser

To transfer a limited number of relatively small files, you can use the cluster's OOD [web portal](#) - after logging into the portal, go to the File menu. Currently, the size of each uploaded file must not exceed 1 GB. You can also directly edit your files through this interface, eliminating the need to transfer files to your computer for editing, see figure [figure 5](#).

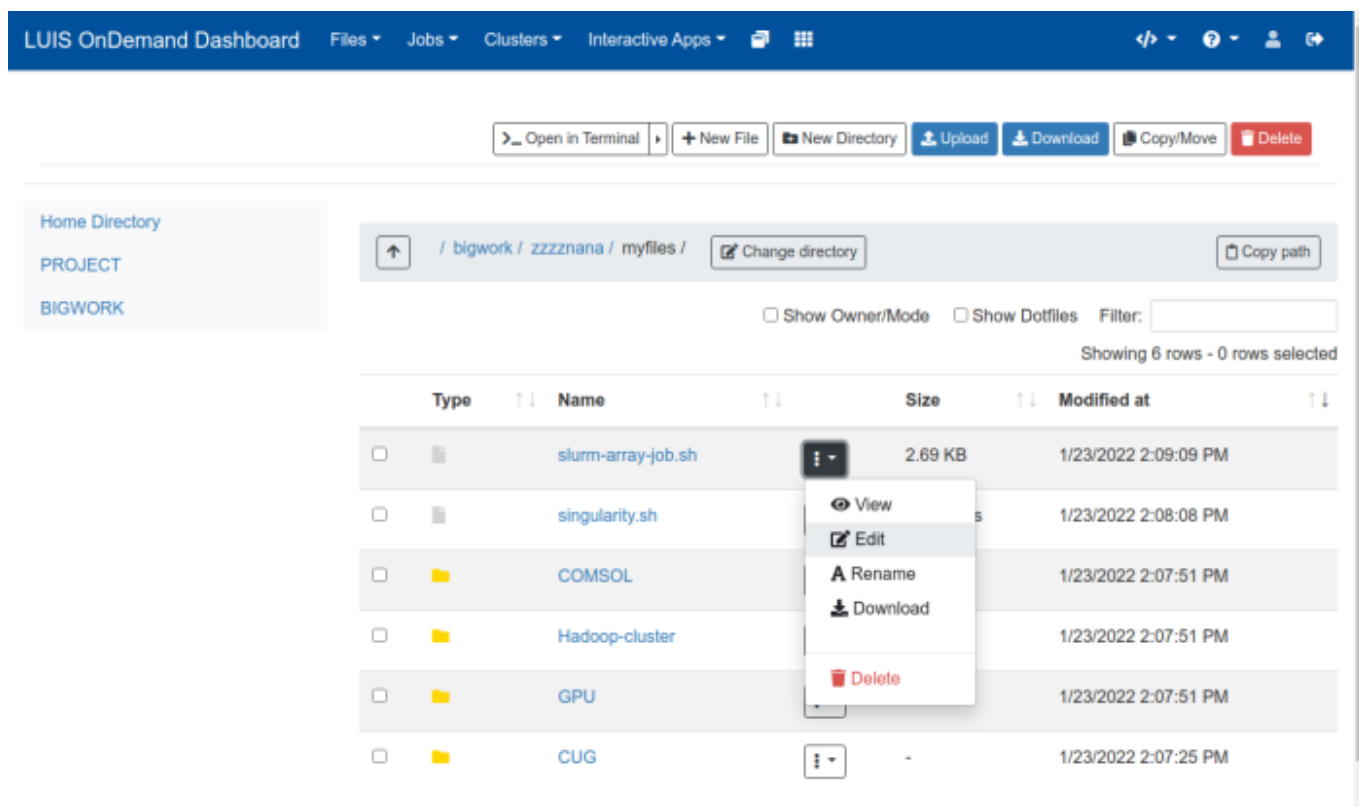


Fig. 5: Cluster web portal, file manager

## To ensure your files are really identical

There's a variety of tools with which you can create a checksum for your files to check that not only the size of the file is correct, but to also make it highly unlikely that it has been somehow corrupted during transfer. A good tool could e.g. be sha256sum, which you could use both on the source and the target end to check whether you get the same checksum for

```
sha256sum <file-to-check>
```

If the results are identical, your files probably are too.

## In case you experience strange aborts transferring data to Windows machines

... check whether your target storage is formatted with a file system like FAT32 that does not support files larger than 4 Gigabytes. Open a file explorer (LeftWindows+E), select the disk, right click, properties → common. In case you are using an unsuitable file system, you'll need to reformat to a file system that supports large files (e.g. exFAT or NTFS).

## Converting jobscripts written under Windows

Windows and Unix/Linux use a different byte/character to mark the end of a line in a text file. You usually won't see these characters, of course, since they are converted into a line change, but when



you transfer a file to a different operating system without taking care about this, you may get into trouble.

In Windows, the end of a line is marked by a combination of CR/LF ("carriage return" and "line feed"). In Linux, it's just LF.

So when you transfer text files - and scripts usually are text files - without taking care of that difference, you'll probably see "^M" characters at the end of the line (when you created a file on Windows, transferred it to the cluster and edit it there) or (when the file was created on the cluster and you transferred it back to a Windows machine) just one very long line.

To automatically take care about this, tools like e.g. FileZilla usually provide two transfer modes, ASCII and binary, and for text files (like job scripts), ASCII should be used. There's also frequently an "Auto" mode that selects the (hopefully) correct mode by looking at the file extension, like .bat, .job, .py, .txt and so on.

The problem is sometimes hidden, since Editors like WordPad may recognize the "wrong" format and display the file correctly, and not all editors show "^M", since this is a so-called control character. But you should always be able to check the type of your files with the "file" command (on Linux).

Example:

Creating a jobscript under windows and copying it onto the cluster system may create the following error message when submitting that jobscript with sbatch:

```
zzzz0001@login02:~$ sbatch file-from-windows.txt
sbatch: script is written in DOS/Windows text format
```

Check this file with the file command:

```
zzzz0001@login02:~$ file file-from-windows.txt
file-from-windows.txt: ASCII text, with CRLF line terminators
```

Convert the file to Unix format:

```
zzzz0001@login02:~$ dos2unix file-from-windows.txt
dos2unix: converting file file-from-windows.txt to UNIX format ...
```

Check the file again with the file command to see if conversion was successful:

```
zzzz0001@login02:~$ file file-from-windows.txt
file-from-windows.txt: ASCII text
```

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/250\\_ransferring\\_data](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/250_ransferring_data)

Last update: **2024/04/26 17:22**



# Transferring files from/to cloud storage

This document walks through some of the basics of using the command line program [Rclone](#) allowing to synchronize files between the cluster (compute and login nodes) and external cloud storage systems. Rclone has a rich set of features and can be used with many different cloud systems including Dropbox, ownCloud, Google Drive, etc. However, here we will consider [Seafile](#) as it is a cloud platform [the cloud service at the LUIS](#) is based on.

Rclone which provides cloud equivalents to the unix commands like `ls`, `cp`, `mkdir`, `rm`, `rsync`, etc. is installed on the login and transfer nodes.

**Please note:** If you plan to migrate a large amount of data, it is recommended to use the transfer node of the cluster, as it is connected to the university's 10Gbs network and has no limitation set on the CPU-time utilized by user processes.

**Please note:** The cloud storage systems provided by the LUIS can be accessed directly from the computing nodes.

## Configuring cloud endpoint for Seafile

Note that the Rclone configuration steps in this subsection need to be completed only once for each cloud storage endpoint.

Since Rclone does not currently support Seafile authentication over SSO, we will assume that you have configured your LUIS "Cloud-Seafile" storage for access via WebDAV. If you have not done so already, follow the instructions in the section "Zugriff über WebDAV" on [the service documentation page](#).

However, if you want to access your LUIS "Projekt-Seafile" at <https://seafile.projekt.uni-hannover.de>, you would only need to provide your LUH-ID credentials.

Each cloud storage provider you want to connect to from the cluster has to be first configured as the rclone *remote* (cloud endpoints that you connect to) using the command:

```
[username@transfer] $ rclone config
```

which will guide you through an interactive setup process. If you have not configured any *remotes* yet, type `n` at the prompt line to create a new one:

```
2021/09/27 14:19:13 NOTICE: Config file
"/home/username/.config/rclone/rclone.conf" not found - using defaults
No remotes found - make a new one
n) New remote
s) Set configuration password
q) Quit config
n/s/q> n
```

As you may notice, rclone stores its configuration in the file  
`$HOME/.config/rclone/rclone.conf`

Next, it asks for an endpoint name, which can be whatever you like, but as you will need to type it in every rclone command, you might want to keep it short and memorable:

```
name> cloud-luis
```

The next parameter is the cloud provider you want to connect to. Since we access “Cloud-Seafile” via WebDAV, enter here 38 or type in webdav (at the time of this writing, WebDAV is 38 in the listing):

```
Type of storage to configure.
Enter a string value. Press Enter for the default ("").
Choose a number from below, or type in your own value
....
37 / Uptobox
   \ "uptobox"
38 / Webdav
   \ "webdav"
39 / Yandex Disk
   \ "yandex"
....
Storage> 38
```

**NOTE:** if you are configuring your “Projekt-Seafile”, enter 43 or seafile above and follow respective instructions.

A list of all possible storage providers can be found [here](#) or by running `rclone help backends`.

In the next two steps enter <https://seafile.cloud.uni-hannover.de/dav> as the URL of LUIS “Cloud-Seafile” and other for the option vendor:

```
URL of http host to connect to
Enter a string value. Press Enter for the default ("").
url> https://seafile.cloud.uni-hannover.de/dav

Name of the Webdav site/service/software you are using
Enter a string value. Press Enter for the default ("").
Choose a number from below, or type in your own value
....
5 / Other site/service or software
   \ "other"
vendor> other
```

Next you will be prompted to enter your WebDAV username and password:

```
User name.
Enter a string value. Press Enter for the default ("").
user> username@uni-hannover.de
```

```
Password.  
y) Yes type in my own password  
g) Generate random password  
n) No leave this optional password blank (default)  
y/g/n> y  
  
Enter the password:  
password:  
Confirm the password:  
password:
```

Leave the next 3 parameters blank(default).

You will finally get a summary, where you should type y if everything is OK and then q to finish the configuration:

```
-----  
[cloud-luis]  
type = webdav  
url = https://seafile.cloud.uni-hannover.de/dav  
vendor = other  
user = username@uni-hannover.de  
pass = *** ENCRYPTED ***  
-----  
y) Yes this is OK (default)  
e) Edit this remote  
d) Delete this remote  
y/e/d> y  
  
Current remotes:  
  
Name                Type  
====                =====  
cloud-luis          webdav  
  
e) Edit existing remote  
n) New remote  
d) Delete remote  
r) Rename remote  
c) Copy remote  
s) Set configuration password  
q) Quit config  
e/n/d/r/c/s/q>  
e/n/d/r/c/s/q> q
```

## Rclone usage examples

**Note:** You can use TAB key for completing Rclone commands and their options.

The command shows all Rclone *remotes* you have configured:

```
username@transfer$ rclone listremotes
cloud-luis:
project-luis:
mycloud:
```

## Navigating objects in the cloud storage

The following displays top level directories (or buckets) on the cloud which has been configured as the rclone *remote* mycloud:

```
username@transfer$ rclone lsd mycloud:
```

Note the colon at the end of the name of *remote*.

To list all files in the path *mydir* on the cloud:

```
username@transfer$ rclone ls mycloud:mydir
```

Note that by default `ls` recursively lists contents of directories. Use the option `--max-depth N` to stop the recursion at the level *N*.

If you want to get more information about files like their size, modification time and path, run:

```
username@transfer$ rclone lsl mycloud:
```

The `tree` subcommand lists recursively the *remote* contents in a tree format. The option `-C` colorizes the output:

```
username@transfer$ rclone tree -C mycloud:
/
├── dir1
│   ├── dir1_2
│   │   └── file2
│   ├── myfile1
│   └── myfile5
└── dir3
    ├── dir3_1
    │   └── myfile3
    └── file4
```

To selectively display files and directories, with the commands `ls` and `lsl`, you can apply global options such as `--exclude`/`--include`, `--filter`, `--min-size`, etc. More information can be found [here](#).

To create a new directory *mydir* on the *remote* mycloud, type:

```
username@transfer$ rclone mkdir mycloud:dir1/mydir
```

**Note** that in case of Seafile cloud storage, you can not remove/create top level directories(also called Libraries) using Rclone.

## Copying & synchronizing data

To copy a file called `myfile.txt` from your local directory to a subdirectory `dir1` on the cloud:

```
username@transfer$ rclone copy myfile.txt mycloud:dir1
```

The following will transfer the contents of your `$BIGWORK/mydir` directory to the subdirectory `dir1/mydir` on the cloud storage:

```
username@transfer$ rclone copy --progress $BIGWORK/mydir mycloud:dir1/mydir
```

If the destination directory (`mycloud:dir1/mydir` in the example above) does not exist, Rclone will create it. Files that already exist at the destination are skipped. If you want to skip files that are newer at the destination use the global flag `--update`, which ensures that the latest version of the files is available in the cloud.

**Note:** To speed up copying a directory containing a large number of small files, the directory should be transferred as a compressed tarball archive file (see the [section](#) on working with large datasets). This can also help you to overcome the limitation imposed by some cloud storage providers(e.g. Google Drive) on the number of simultaneously transferred files.

As long as the network and storage systems(remote/local) can handle it, you may improve the overall transfer rates by increasing the values for these two Rclone global options:

```
--transfers=N (Number of file transfers to be run in parallel. default N=4)  
--drive-chunk-size=SIZE (Transfer chunk size in kilobytes. Must a power of 2  
>= 256k. default SIZE=8192)
```

The parameter `--drive-chunk-size` might be useful only when transferring large files.

If you would like your destination storage (remote or local) to have exactly the same content as the source, use the `sync` subcommand instead. Below is an example to sync the contents of your cloud directory `mycloud:dir1/mydir` (the source) to the `$BIGWORK/mydir` (the destination) directory at the cluster:

```
username@transfer$ rclone sync mycloud:dir1/mydir $BIGWORK/mydir
```

Contrary to the `copy` subcommand, if files are removed from the source, synchronizing the source and destination will delete files from the destination as well. Copying will never delete files in the destination.

**WARNING:** as the command can cause data loss at the destination, it is recommended to always test it first with the `--dry-run` flag to see exactly what would be copied and deleted.

Additional flags can be used similarly to the `copy` subcommand.

## Deleting objects from the cloud storage

The command removes the directory `mydir` and all its contents from the `dir1` at the *remote* `mycloud`:

```
username@transfer$ rclone purge mycloud:dir1/mydir
```

If you need to selectively delete files from the cloud use the `delete` subcommand instead. For example, the following will remove all files larger than 500MB from the `dir1/mydir` directory:

```
username@transfer$ rclone delete --min-size 500M mycloud:dir1/mydir
```

To remove files older than 60 days:

```
username@transfer$ rclone delete --min-age 60d mycloud:dir1
```

To see other Rclone global flags, execute `rclone help flags`. More information on how to filter objects is available [here](#).

You may first want to check what would be deleted with the `--dry-run` flag or use the `--interactive` option.

Note that the `delete` removes only files keeping the directory structure in the *remote* unchanged. Adding the option `--rmdirs` will remove empty sub-directories along with files.

## Creating a cron job

If you wish to periodically run rclone, you can achieve this with a cron job. To create a cron job, modify your current crontab using the `crontab -e` command. Once in the crontab editor you can input your rclone commands. Below is an example cron job executing the `rclone sync` every 20 minutes:

```
*/20 * * * * /bin/rclone sync <myremote>:<mydata> </local/path/to/mydata>
```

After you exit from the editor, the modified crontab will be installed automatically. To list all your current cron jobs invoke `crontab -l`.

**Note:** the cron job will only be executed on the machine you created it on. Therefore, the recommended way to work with cron jobs is to manage them on a fixed machine. A good candidate would be the transfer node.

## Further Reading

- Rclone [command line reference](#)
- Rclone [supported cloud storage systems](#)
- Rclone [configuration for various cloud storage](#)

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/251\\_cloud\\_storage\\_rclone](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/251_cloud_storage_rclone)

Last update: **2021/12/09 15:31**





# Transferring data into the archive

---

**Please note:** The archive is operated as part of the service [Archivierung](#) and thus not part of the cluster system.

The archive can be used to store results and simulation data permanently. Each account has to be registered for archive use, before using it. This can be done on the [BIAS-website](#) after logging in with your user name and password. After clicking on the link entitled *Ihren Benutzernamen für die Nutzung des Archivsystems zulassen* it takes roughly an hour, before the archive can be used.

## Quota

Archival storage in the archive system of Leibniz Universität Hannover is controlled by a quota mechanism. There is a quota on the amount of files as well as storage space. Please see the website of the archive service for further details at <http://www.luis.uni-hannover.de/archivierung.html>.

## Transferring data into the archive

In order to transfer data into the archive of Leibniz Universität Hannover, it is recommended to use the cluster's dedicated transfer node, see [section](#).

## Login with lftp

The archive can be reached at `archiv.luis.uni-hannover.de` using the `lftp` command.

```
username@transfer:~$ lftp <username>@archiv.luis.uni-hannover.de
```

After entering your cluster user name's password the `lftp` prompt appears.

```
lftp <username>@archiv.luis.uni-hannover.de:~>
```

Now you can use the `ls` command to list your directory contents at the archive. At the same time this is to test an established connection to the archive.

```
lftp <username>@archiv.luis.uni-hannover.de:~> ls
```

At your first login to the archive system with your account the directory is empty. The `ls` command will not return any listing. You can terminate the connection with `exit`.

```
lftp <username>@archiv.luis.uni-hannover.de:~> exit
<username>@transfer:~$
```

Aliases for exit are quit and bye.

## Copying files into the archive

On the cluster system's transfer node change to the directory where the data to be copied are located.

```
username@transfer:~$ cd $BIGWORK/my_data_dir
username@transfer:/bigwork/username/my_data_dir$
```

After logging in using lftp the put command is used.

```
username@transfer:/bigwork/username/my_data_dir$ lftp
<username>@archiv.luis.uni-hannover.de:~>
lftp <username>@archiv.luis.uni-hannover.de:~> put myfile.tar.gz
```

The file `myfile.tar.gz` is located inside the directory we previously changed to in this example. After using `put` to transfer the file it is also available on the archive. The TAB key works for completing file and directory names in lftp as well.

Saving multiple small files in the archive is not desired, because at least one copy of the data are kept on magnetic tape. Therefore a constant stream of data is desirable which can be achieved by some large files. It is recommended to use `tar` or `zip` to combine small files into one bigger file. This can also optimize your quota.

In order to transfer multiple (large) files at once, the `mput` command can be used. This is short for *multiple put*. The `mput` command understands the wildcard `*` as it is used in bash.

```
lftp <username>@archiv.luis.uni-hannover.de:~> mput mydata*.tar.gz
```

## Fetching files from the archive

In order to get fetch files from the archive, the `get` command can be used.

```
lftp <username>@archiv.luis.uni-hannover.de:~> get myfile.tar.gz
```

This command puts the file at the location the `lftp` command was issued from which transferred the file into the archive. For fetching more than one file the `mget` command can be used (*multiple get*). Fetching the file may take some time until transfer starts. This time is needed by the storage robot to find the respective magnetic tape and wind the tape to the position the file is located at.

## Some useful commands

Listing the current directories' contents can be achieved by the command `!ls`. An exclamation mark executes the command on the machine lftp was started on. On the contrary listing the current *local*

directory can be done with `lpwd` at the `lftp` prompt.

It is possible to create directories in the archive using the `mkdir` command.

```
lftp <username>@archiv.luis.uni-hannover.de:~> mkdir myDir
```

Changing directories works in the usual way using `cd`.

```
lftp <username>@archiv.luis.uni-hannover.de:~> cd myDir
```

And back up one directory.

```
lftp <username>@archiv.luis.uni-hannover.de:~> cd ..
```

A local directory can be changed using the `lcd` command, short for *local cd*.

```
lftp <username>@archiv.luis.uni-hannover.de:~> lcd  
/bigwork/<username>/datadir
```

## Further reading

- man page `lftp`, man `lftp`. Navigate using the arrow keys and exit with 'q'
- Service [Archivierung](#)

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/252\\_transferring\\_files\\_into\\_the\\_archive](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/252_transferring_files_into_the_archive)

Last update: **2021/10/22 06:52**



# Running PBS/Torque jobs

Read about the new [SLURM system here](#).



This part of the documentation **is only kept for reference purposes for the time being. It describes the old software combination PBS/Torque/Maui (“portable batch system”) that has been used in the cluster for many years, but now has been phased out and replaced by the more modern system SLURM.** We will shut down for good all PBS/Torque/Maui servers by the end of 2022.

PBS/Torque/Maui uses [TORQUE](#) as a resource manager and [MAUI](#) as a scheduler. PBS is kind of a synonym to Torque. The text is written as a kind of loose tutorial, so you can work your way from a very easy start to more complicated cases in the lower sections.

As with any scheduling system, job scripts are the basic blocks to access the computing resources of the cluster system. They are submitted to the system by the `qsub` command. Generally `qsub` has the following form.

```
qsub <options> <name of job script>
```

The manual page for `qsub` can be accessed via

```
man qsub
```

You can quit reading the manual page by pressing the 'q' key.

## Interactive jobs

The simplest mode to begin is by starting an interactive job. This can be done by issuing the `qsub` command with `-I` option on any login node.

```
nhabcdef@login02:~$ qsub -I
ACHTUNG / WARNING:
'mem' parameter not present; the default value will be used (1800mb)
'walltime' parameter not present; the default value will be used (24:00:00)
'nodes' parameter not present; the default value will be used
(nodes=1:ppn=1)
qsub: waiting for job 1001152.batch.css.lan to start
qsub: job 1001152.batch.css.lan ready

nhabcdef@bmwz-n001:~$
```

In this example, a user by the name `nhabcdef` issues a `qsub` command from the node `login02`. The

batch-system warns about any default parameters it set to complete the request and starts a job with ID 1001152.batch.css.lan. Using the short jobID 1001152 is more common. Immediately after the job becomes ready, the user nhabcdefg is placed into a shell / command prompt on the machine bmwz-n001 that has been allocated for the job. This can be seen by looking at the command prompt, which now shows @bmwz-n001 to indicate the new host computer. So you are now on node number 80 in the Lena cluster partition, and you can now use one cpu core (ppn=1) on this one node (nodes=1) for the next 24 hours (walltime), as long as you do not try to use more than 1800 MB of memory (your job gets killed if you try to use more. This may seem brutal at first, but it's done to protect the other jobs running on the same machine. And since all of your jobs are well-behaved, it mainly will protect your jobs, of course). If you don't need the reservation any more, please be so kind to type "exit" to finish the job and yield the resources to the next waiting user. You will get back to the prompt saying "@login".

This simplest form of an interactive job uses default values for all resource specifications. In practice, resource specifications should carefully be adapted to fit one's needs, you should consider what the optimal configuration for your job would be. This can be done by supplying the qsub command with additional options. A listing of possible options can be found in [section](#). The following example illustrates how user nhabcdef requests specific resources starting from login node login02 inside an interactive job. The user requests one cpu-core on one machine and 2 GB of memory for one hour. Additionally, the -X option is used to switch on X window forwarding, so applications with a graphical user interface can be used.

```
nhabcdef@login02:~$ qsub -I -X -l nodes=1:ppn=1 -l walltime=01:00:00 -l
mem=2GB
qsub: waiting for job 1001154.batch.css.lan to start
qsub: job 1001154.batch.css.lan ready

nhabcdef@bmwz-n001:~$
```

After job 1001154 has started, the machine bmwz-n001 is ready for use. An extended example of how to utilise interactive jobs is given in [section 6.10](#).

## Batch jobs

To prepare batch jobs that can run unsupervised, you'll usually start with an interactive job. Within that interactive job, all commands that are later going to make up a batch script can be entered in sequence, thus testing that everything works as you planned. Once everything works, put all the commands line by line into a batch script, add the appropriate resource requests and possibly additional instructions for the batch system, then submit it. In case you were given a batch script by other people, first take some time to test all the commands in an interactive job. This way, you'll familiarize yourself with what the individual commands do.

In order to request the same resources as the interactive job from [above](#) within a batch job, the following instructions should be added to the start of your batch file:

```
#!/bin/bash -login

# resource specification
#PBS -l nodes=1:ppn=1
```

```
#PBS -l walltime=00:05:00
#PBS -l mem=2GB

# commands to execute
date
```

Generally speaking, jobscripts can be divided into a header containing resource specifications and batch system instructions, then the body with all the commands to be executed. Lines beginning with `#` are usually comments with two very important exceptions: The very first line in a batch script has a special syntax (“shebang line”) that advises the system what interpreter to use for the script. For the batch system, you should normally stick with `/bin/bash`. Lines beginning with `#PBS` are recognized by the batch system as resource specifications and instructions where to place results, how to name a job for easier identification etc. In this case, the script requests one cpu-core on one machine and 2 GB of memory for five minutes. The body contains just one single “date” command, which is appropriate, since it's just a sample script. The date command simply returns the current date and time.

Save this file as `batch-job-example.sh` and submit it to the batch system by issuing `qsub batch-job-example.sh`:

```
nhabcdef@login02:~$ qsub batch-job-example.sh
1001187.batch.css.lan
```

After submitting the jobscript, a JobID is returned by the batch system, in this case it is 1001187. After the job has finished, two files can be found in the directory the jobscript was submitted from:

```
zzzzabcdef@login02:~$ ls -lh
total 12K
-rw-r--r-- 1 nhabcdef nhab 137 19. Apr 12:54 batch-job-example.sh
-rw----- 1 nhabcdef nhab  0 19. Apr 12:59 batch-job-example.sh.e1001187
-rw----- 1 nhabcdef nhab 30 19. Apr 12:59 batch-job-example.sh.o1001187
```

The first file has the extension `.e1001187`, which holds all error messages which occurred during job execution. In this case, this file is empty. The second file has the extension `.o1001187` and contains all messages which would have been displayed on the terminal and have been redirected here. By displaying this file's contents, this can be verified:

```
nhabcdef@login02:~$ cat batch-job-example.sh.o1001187
Tue Apr 19 12:59:18 CEST 2016
```

The file contains the output of the date command.

**Please note:** Jobscripts written under windows need converting, see [section](#)

## PBS options

In this section, we list selected PBS options that allow controlling your jobs. These options are valid for both interactive and batch jobs.

<b>-N name</b>	name the job
<b>-j oe</b>	join standard output and error streams
<b>-l nodes=n:ppn=p</b>	request $n$ nodes and $p$ cpu cores per node
<b>-l walltime=time</b>	requested wall clock time in format hh:mm:ss
<b>-l mem=value</b>	requests RAM according to <i>value</i> , possible suffixes are kb, mb, gb
<b>-M email address</b>	list of users to whom mail is sent about the job
<b>-m abe</b>	send mail on (one or multiple selections): <i>a</i> - job abort, <i>b</i> - begin of execution <i>e</i> - job end
<b>-V</b>	all environment variables are exported to the job
<b>-q queue</b>	destination queue of the job, usually all. See <a href="#">section</a>
<b>-W x=PARTITION:name</b>	partition to be used, e.g. to make the job run on a specific cpu architecture. See <a href="#">section</a>
<b>-I</b>	job is to be run interactively

More options can be found in the man page, which can be opened with the command `man qsub`. Use “q” to exit a man page.

## PBS environment variables

“When a batch job is started, a number of variables are introduced into the job’s environment that can be used by the batch script in making decisions, creating output files, and so forth. These variables are listed in the following table”<sup>2)</sup> :

<b>PBS_O_WORKDIR</b>	Job’s submission directory
<b>PBS_NODEFILE</b>	File containing line delimited list on nodes allocated to the job
<b>PBS_QUEUE</b>	Job queue
<b>PBS_JOBNAME</b>	User specified jobname
<b>PBS_JOBID</b>	Unique JobID

## PBS commands

<b>qsub script</b>	Submit PBS job
<b>showq</b>	Show status of PBS jobs
<b>qdel jobid</b>	Delete Job <i>jobid</i>

All of the above commands have detailed manual pages, which can be viewed with the following command:

```
man <command>
```

## pbsnodes

On the login-nodes, the `pbsnodes` command can be used to obtain information about resources. Example:

```
pbsnodes smp-n031
```

At first, the output may seem a little bit overwhelming. It shows, among other things, the following parameter:

```
physmem=1058644176kb
```

This output can be converted into gb. 1024kb equals 1mb, 1024mb equals 1gb etc. ... This way you know that the maximum amount of memory you can request on one machine in queue helena is 1009 gb. You should request (4-8 GB) less than that to leave some room for the operating system to allocate some buffers, which may make your job run more efficient.

Also note the other properties of this machine:

```
properties = helena,nehalem,ib,luis,smp,c32
```

That means that if you request a property like "smp", you may land on that node.

If you want to run your jobs anywhere BUT on the smp partition, use the mpp feature. Example:

```
#PBS -l nodes=1:ppn=16:mpp
```

In case you want to run only on nodes providing a specific number of cores, use the c-features (c12, c16, c24, c32, c40, ...). Example:

```
PBS -l nodes=1:ppn=8:smp:c24
```

To find out about all the features you can request, use

```
pbsnodes | egrep "([a-Z].*| .*properties)"
```

or

```
pbsnodes <nodename>, e.g. pbsnodes taurus-n001
```

To find out which/how many nodes carry a specific feature, use

```
pbsnodes :c12  
pbsnodes :mpp
```

## PBS errors

If you get a seemingly strange Error, first try to subtract 128 from it - thus, error 137 becomes error 9, which corresponds to SIGKILL, which in turn may be the result of PBS or Maui intentionally killing your job, for example because it tried to use more memory than you requested, or because it overstepped its wall time.

You can find out more about Unix signals and their meanings here:

[https://en.wikipedia.org/wiki/Signal\\_\(IPC\)#POSIX\\_signals](https://en.wikipedia.org/wiki/Signal_(IPC)#POSIX_signals)

A good page about PBS errors is here:



[https://www.nas.nasa.gov/hecc/support/kb/PBS-exit-codes\\_185.html](https://www.nas.nasa.gov/hecc/support/kb/PBS-exit-codes_185.html)

## Queues & partitions

The cluster system provides the resources listed in table [Computing Hardware](#). You will see that not much is left under control of PBS.

Only queue “all” is still available under PBS/Torque:

<b>all</b>	This is the default queue and does not have to be requested explicitly. PBS will route a job to matching nodes.
------------	---

## Resource request limits and scheduling fairness

The batch system implements some limits to help keep the system available for all users even in busy times. To prevent one account from completely flooding the system or blocking nodes for really long times, both the maximum number of simultaneously running jobs and the maximum walltime per job are limited. Furthermore, the total number of cpu cores that one account can use at the same time is limited to give all users a chance to get a reasonable share of the system in reasonable time. So it does neither help to flood the system with lots of 1-core-jobs nor to submit extremely large jobs. A single user name can only use a certain part of the system at one time, even when the system would be completely empty otherwise. The scheduler will scale up and down to a certain extent what you can run, depending on how busy the system currently is. In case you are really in urgent need, contact us and we will consider whether we can give you some priority at the cost of others, but you will understand that we try to keep things fair for all.

- **Walltime** Maximum walltime is 200 hours per job
- **Jobs** The maximum number of running jobs per user is 64 (on both PBS and SLURM, so 128 in total)
- **CPUs** The overall maximum number of CPUs (ppn) all running jobs can use is 768 per user (PBS) and 800 (SLURM)

## Exercise: interactive job

```
# start an interactive job, what happens?
qsub -I

# exit this interactive job
exit

# specify all resource parameters, so no defaults get used
qsub -I -X -l nodes=1:ppn=1 -l walltime=01:00:00 -l mem=2GB

# load module for octave
module load octave/3.8.1

# start octave
```

```
octave
# inside octave the following commands create a plot
octave:1> x = 0:10;
octave:2> y = x.^2;
octave:3> h = figure(1);
octave:4> plot(x,y);
octave:5> print('-f1','-dpng','myplot')
octave:6> exit

# display newly created image
display myplot.png
```

Interactive jobs are useful for debugging - always use interactive first

## Excercise: batch job

Create a file named MyBatchPlot.m

[MyBatchPlot.m](#)

```
x = 0:10;
y = x.^2;
h = figure(1);
plot(x,y);
print('-f1','-dpng','MyBatchPlot');
```

Create a file named MyFirstBatchJob.sh

[MyFirstBatchJob.sh](#)

```
#!/bin/bash -login
#PBS -l nodes=1:ppn=1
#PBS -l walltime=01:00:00
#PBS -l mem=2GB

# load octave module
module load octave/3.8.1

# start octave
octave MyBatchPlot.m
```

Submit the job script

```
qsub MyFirstBatchJob.sh
```

Check files MyFirstBatchJob.sh.o[jobidnumber] and MyFirstBatchJob.sh.e[jobidnumber]

2)

Source: <http://docs.adaptivecomputing.com/torque/4-2-10>

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/300\\_running\\_torque\\_jobs](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/300_running_torque_jobs)

Last update: **2022/12/13 15:39**



# SLURM usage guide

## If you are completely new to scientific computing, read this:

Please keep in mind that simply throwing a serial program without provisions for parallelization at the cluster will not make it run faster. In fact, it will often run SLOWER, since cpu cores on machines specialized in scientific computing usually run at lower clock frequencies than workstation cpus.



There are many commercial and open source software packages that are already very well parallelized, **but you'll definitely need to know how to use the parallel capabilities of your software or programming language**. Requesting lots of nodes, cpu cores, memory and time for a program that will use only one single cpu will only keep you and all other users waiting.

So **start small and simple**. There's no use waiting for a 10-node-job just to find out it immediately crashes, so test with a 1-node-1-task-job that requests only 10 minutes first, then moderately make it bigger and check that e.g. your parallel program delivers reasonable results. When you are sure you understand what you do and when your test case works reliably, when you see a decrease in run time when you add resources — then go for it.

## Why use a cluster at all?

The reason you want to use the cluster is probably the computing resources it provides. With about several hundred people using the compute cluster for their research every year, there has to be an instance organizing and allocating these resources. This instance is called the batch system ("scheduler", "resource manager", in the LUH-cluster: "SLURM"). The batch system sorts the resource requests ("batch jobs") it gets from the users according to resource availability and priority rules. When the priority of a job is sufficiently high to start, it gets scheduled on the requested resources (usually some compute node(s)) and starts. Requests to the batch system are usually made by submitting a text file containing (bash-) instructions about what to do (the "batch job"), or by requesting an "interactive" batch job from the command line that puts you directly into a command shell on a compute node (or a set of compute nodes) to work with, or, as a third option, by the Open OnDemand portal ("OOD") that is available on <https://login.cluster.uni-hannover.de> which translates the settings you enter via the web browser gui into a text file to again submit as a batch job to the batch system.

# Parallelization Basics

Batch jobs can roughly be divided into several categories:

- serial (also: single-threaded, single-task) batch jobs; they run just like normal programs/scripts. Usually, no changes/adaptions are needed. The time needed to complete such a job almost entirely depends on the speed of a single cpu core, and no scaling is achieved. As mentioned above, starting a serial program on a larger machine will NOT make it run faster, and since the compute servers in the cluster usually combine lots of cpu cores on each cpu socket, they actually may run even slower than cores on a smaller workstation that has fewer cores, due to the total heat generated by each chip that needs to be dissipated. So think about the size and characteristics of your workload. In the “real world”, smaller loads over short distances are better transported using a fast car, while large ones that travel around the world need a big container ship. Similar considerations are valid for workloads on computers, in particular, if the software is not parallelized.
- parallel jobs; these in turn can be distinguished by the kind of problems they treat and how they attempt to achieve scaling (“reaching results quicker”). Problem are either “trivially parallel computations” or not:
  - trivially parallel problems can be solved by simply starting as many tasks as needed / as possible, and each task will run happily minding its own business, until it achieves a partial result in the end, which is then combined with all the other results of the other tasks to get the result of the whole job. Luckily, many problems can be computed in this way.
  - non-trivial problems are those that usually need to exchange data *during* computation, for example when finishing a simulation time step to update the (so-called) “ghost”-borders of all the simulation subdomains the complete simulation region has been decomposed into.
- The other main distinction of parallel jobs is *how* they do it. The two main variants here are:
  - OpenMP jobs (shared-memory-processing, SMP, single-node, multi-threaded, multi-processing, can typically scale to the largest compute nodes available) usually run on ONE node only, using multiple threads or processes, but sharing memory. So the software needs some logic that specifies which parts of the program (e.g. loops) should run in parallel, but synchronization between threads is automatic. These programs typically are linked to an OpenMP library during compilation, so the node itself usually needs to have some libraries installed, but parallelization is relatively easy and low-effort. The software must ensure that only one process updates a specific memory location at the same time, but it can rely on having the same memory contents. Beware, though, that while many application programs are parallelized this way, and nowadays compute nodes may contain many cpu cores, scaling still may be quite limited depending on the specific kind of problem and how much effort has been spent to parallelize the regions of the software that should do parallel computations. So, while some software packages achieve very good scaling up to over 100 cpu cores on big servers, others already hit their limits when using more than 4 or 8 cpus. There's many hardware-factors limiting performance, too, like the number of cache levels, CPU cache sizes, memory bandwidth, NUMA architecture etc, and of course I/O.
  - MPI (message-passing-interface, possible multi-node, multi-processing, can scale up to millions of cpus); software parallelized using MPI is usually able to achieve the highest scaling, but the cost is that the software must explicitly specify which parts of the program run in parallel and how data / results are updated, which task does what, and

how the simulation is kept in sync. So the scaling here also depends on the genius of the person writing the software and their knowledge about specific hardware features. Each MPI-task (called a “rank”) is highly independent of the others, and it needs to explicitly communicate its results to the other tasks whenever there's a need for that, since each task uses their own memory that only they can access.

**Hint:** to avoid an easy understanding of complicated things, someone thought it would be a good idea to name one of the several MPI libraries available “OpenMPI”. Do not fall for this trap. OpenMPI is one specific implementation of the MPI programming interface (there are many others called IntelMPI, MVAPICH, IBMMPI, ...) that uses message-passing between independent tasks to achieve a high degree of parallelization. OpenMP, on the other hand, is a general term for a completely different programming interface that is using compiler-directives and which has NOT much to do with MPI. So OpenMP is NOT MPI, while OpenMPI is MPI, but NOT OpenMP.

## The SLURM Workload Manager

The software that decides which job to run when and where in the cluster is called SLURM. SLURM (**S**imple **L**inux **U**tility for **R**esource **M**anagement) is a free open-source batch scheduler and resource manager that allows users to run their jobs on the LUIS compute cluster. It is a modern, extensible batch system that is installed on many clusters of various sizes around the world. This chapter describes the basic tasks necessary for submitting, running and monitoring jobs under the SLURM Workload Manager on the LUIS cluster. Detailed information about SLURM can be found on the official SLURM [website](#).

Here are some of the most important commands to interact with SLURM:

- **sbatch** - submit a batch script
- **salloc** - allocate compute resources
- **srun** - allocate compute resources and launch job-steps
- **squeue** - check the status of running and/or pending jobs
- **scancel** - delete jobs from the queue
- **sinfo** - view information about cluster nodes and partitions
- **scontrol** - show detailed information on active and/or recently completed jobs, nodes and partitions
- **sacct** - provide the accounting information on running and completed jobs
- **slurmtop** - text-based view of cluster nodes' free and in-use resources and status of jobs

Some usage examples for these commands are provided below. As always, you can find out more using the manual pages on a terminal/console on the system (like `man squeue`) or on the SLURM manuals' [website](#).

## Partitions

Compute nodes with similar hardware attributes (like e.g. the same cpu) in the cluster are usually grouped in partitions. Each partition can be regarded as somewhat independent from others. A batch job can be submitted in such a way that it can run on one of several partitions, and a compute node may also belong to several partitions simultaneously to facilitate selection. Jobs are allocated resources like cpu cores, memory and time within a single partition for executing tasks on the cluster.

A concept called “job steps” is used to execute several tasks simultaneously or sequentially within a job using the `srun` command.

The table below lists the currently defined partitions and their parameters/constraints:

Part of cluster	Max Job Runtime	Max Nodes Per Job	Max CPUs per User	Default Runtime	Default Memory per CPU	Shared Node Usage
amo, dumbo, haku, lena, taurus, ... (generic)	200 hours		800	24 hours	4000 MB	yes
gpu nodes	48 hours	1		1 hour	1600 MB	yes

To keep things fair, control job workload and keep SLURM responsive, we enforce some additional restrictions:

SLURM limits	Max number of jobs running	Max number of jobs submitted
per user	64	500
cluster-wide	10000	20000

Based on available resources and when still able to maintain a fair balance between all users' needs, we may sometimes also consider requests for a higher priority for a short time, which may be submitted to [cluster-help@luis.uni-hannover.de](mailto:cluster-help@luis.uni-hannover.de). You should include an explanation for what period of time you need which kind of priority, and of course why we should consider your request regarding the fact that usually all other users want priority, too.

To list job limits relevant for you, use the `sacctmgr` command:

```
sacctmgr -s show user
sacctmgr -s show user format=user,account,maxjobs,maxsubmit,maxwall,qos
```

Up-to-date information on ALL available nodes:

```
sinfo -Nl
scontrol show nodes
```

Information on partitions and their configuration:

```
sinfo -s
scontrol show partitions
```

The `clusterinfo` command (Python script) retrieves real-time information about node and partition configurations, resource (CPU/GPU) usage, and user access rights to resources through native SLURM commands and displays the data in a structured format for easier interpretation. It shows which nodes are accessible by all users and which are reserved for specific research groups with exclusive access during configured times (see [Forschungscluster-Housing](#)). By executing `clusterinfo -l`, your configured SLURM limits (such as the maximum number of running and pending jobs, maximum wall clock time, etc.) will also be displayed. For a list of available options and their descriptions, run `clusterinfo -h`.

## Interactive jobs

**Please note:** when you have a *non-interactive* (standard) reservation/running job on a node or a set of nodes, you may *also* directly open additional shell(s) to that node(s) coming from a login node, e.g. for watching/debugging/changing what happens. But beware: you will get kicked out as soon as your job finishes.

Batch submission is the most common and most efficient way to use the computing cluster. Interactive jobs are also possible; they may be useful for things like:

- working with an interactive terminal or GUI applications like R, iPython, ANSYS, MATLAB, etc.
- software development, debugging, or compiling

You can start an interactive session on a compute node using the SLURM `salloc` command. The following example submits an interactive job that requests 12 tasks (this corresponds to 12 MPI ranks) on two compute nodes and 4 GB memory per CPU core for an hour:

```
[user@login02 ~]$ salloc --time=1:00:00 --nodes=2 --ntasks=12 --mem-per-cpu=4G --x11
salloc: slurm_job_submit: set partition of submitted job to amo,tnt,gih
salloc: Pending job allocation 27477
salloc: job 27477 queued and waiting for resources
salloc: job 27477 has been allocated resources
salloc: Granted job allocation 27477
salloc: Waiting for resource configuration
salloc: Nodes amo-n[001-002] are ready for job
[user@amo-n001 ~]$
```

The option `--x11` sets up X11 forwarding on the first(master) compute node enabling the use of graphical applications.

**Note:** Unless you specify a cluster partition explicitly, all partitions that you have access to will be available for your job.

**Note:** If you do not explicitly specify memory and time parameters for your job, the corresponding default values for the cluster partition to which the job will be assigned will be used. To find out the default time and memory settings for a partition, e.g. `amo`, look at the `DefaultTime` and `DefMemPerCPU` values in the `scontrol show partitions amo` command output.

**Note:** In case you get an error message like `srun: Warning: can't honor --ntasks-per-node set to X which doesn't match the requested tasks YY with the number of requested nodes ZZ`. Ignoring, check (using `set | grep SLURM_N` within the job shell, for example) that your request has been honored despite the message, and then ignore the message.



Once the job starts, you will get an interactive shell on the first compute node (`amo-n001` in the example above) that has been assigned to the job, where you can interactively spawn your applications. The following example compiles and executes the MPI Hello World program (save the source code to the file `hello_mpi.c`):



## hello\_mpi.c

```
#include "mpi.h"
#include <stdio.h>

int main (int argc, char** argv) {
    int ntasks, taskid, len;
    char hostname[MPI_MAX_PROCESSOR_NAME];

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD,&ntasks);
    MPI_Comm_rank(MPI_COMM_WORLD,&taskid);
    MPI_Get_processor_name(hostname, &len);

    printf ("Hello from task %d of %d on %s\n", taskid, ntasks,
hostname);

    MPI_Finalize();
}
```

```
[user@amo-n001 ~]$ module load GCC/9.3.0 OpenMPI/4.0.3
[user@amo-n001 ~]$ mpicc hello_mpi.c -o hello_mpi
[user@amo-n001 ~]$ srun --ntasks=6 --distribution=block hello_mpi
Hello from task 0 of 6 on amo-n001
Hello from task 1 of 6 on amo-n001
Hello from task 2 of 6 on amo-n001
Hello from task 3 of 6 on amo-n001
Hello from task 4 of 6 on amo-n001
Hello from task 5 of 6 on amo-n002
```

**Note:** If you want to run a parallel application using Intel MPI Library (e.g by loading the module impi/2020a) then provide the srun command with an additional option `--mpi=pmi2`

**Note:** Environment variables set on the login node from which the job was submitted are not passed to the job.

The interactive session is terminated by typing `exit` on the shell:

```
[user@amo-n001 ~]$ exit
logout
salloc: Relinquishing job allocation 27477
```

Alternatively you can use the `srun --pty $SHELL -l` command to interactively allocate compute resources, e.g.

```
[user@login02 ~]$ srun --time=1:00:00 --nodes=2 --ntasks=12 --mem-per-cpu=4G
--x11 --pty $SHELL -l
srun: slurm_job_submit: set partition of submitted job to amo,tnt,gih
[user@amo-n004 ~]$
```

At this point, we would like to note that SLURM differentiates between `--ntasks`, which may roughly be translated into the number of (independent) MPI-ranks or instances of a job, and `--cores-per-task`, which may translate into the number of (OpenMP) threads. MPI-jobs usually request `--ntasks` larger than one, while OpenMP-jobs may request `--ntasks=1` and `--cores-per-task` higher than one.

If you want to run your jobs on nodes with a specific CPU type, you can request them using the SLURM option `--constraint=CPU_ARCH:<cpu_arch>`, where `<cpu_arch>` can currently have the following values: `sse`, `avx`, `avx2`, and `avx512`.

To check the available CPU architectures for different SLURM partitions and nodes, you can use the command `clusterinfo -n -i`.

If your job can run on nodes with any of multiple CPU types, you can specify them using the following syntax: `--constraint=[CPU_ARCH:<cpu_arch1>,CPU_ARCH:<cpu_arch1>,...]`.

## Submitting a batch script

A SLURM job submission file for your job (a “batch script”) is a shell script with a set of additional directives that are only interpreted by the batch system (Slurm) at the beginning of the file. These directives are marked by starting the line with the string `#SBATCH`, so the batch system knows that the following parameters and commands are not just a comment (which the `#` character otherwise would imply). The shell (the command line interpreter of Unix) usually ignores everything that follows a `#` character. But at the beginning of your file, the Slurm commands used to submit a batch script will also check whether the `#` character is immediately followed by `SBATCH`. If that is the case, the batch system will interpret the following characters as directives. Processing of these directives stops once the first non-comment non-whitespace line has been reached in the script. The very first line of your script usually should read `#!/bin/bash` - ask Wikipedia for the meaning of “Shebang (Unix)” in case you want to understand what this is for.

Valid directives can be found using the command `man sbatch`. In principle, you may write almost any option that you could feed to `sbatch` at the command line as a `#SBATCH`-line in your script.

A suitable batch script is usually submitted to the batch system using the `sbatch` command.

## An example of a serial job

The following is an example of a simple serial job script (save the lines to the file `test_serial.sh`).

**Note:** change the `#SBATCH` directives to your use case where applicable.

[example\\_serial\\_slurm.sh](#)

```
#!/bin/bash -l
#SBATCH --job-name=test_serial
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=2G
#SBATCH --time=00:20:00
#SBATCH --constraint=[CPU_ARCH:avx512|CPU_ARCH:avx2]
```

```
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --output test_serial-job_%j.out
#SBATCH --error test_serial-job_%j.err

# Change to my work dir
# SLURM_SUBMIT_DIR is an environment variable that automatically gets
# assigned the directory from which you did submit the job. A batch job
# is like a new login, so you'll initially be in your HOME directory.
# So it's usually a good idea to first change into the directory you
# did
# submit your job from.
cd $SLURM_SUBMIT_DIR

# Load the modules you need, see corresponding page in the cluster
# documentation
module load my_modules

# Start my serial app
# srun is needed here only to create an entry in the accounting system,
# but you could also start your app without it here, since it's only
# serial.
srun ./my_serial_app
```

To submit the batch job, use

```
sbatch example_serial_slurm.sh
```

**Note:** as soon as compute nodes are allocated to your job, you can establish an ssh connection from the login machines to these nodes.

**Note:** if your job oversteps the resource limits that you have defined in your #SBATCH directives, the job will automatically be killed by the SLURM server. This is particularly the case when you try to use more memory than you allocated, which results in an OOM (out-of-memory) -event.

The table below shows frequently used sbatch options that can either be specified in your job script with the #SBATCH directive or on the command line. Command line options override options in the script. The commands srun and salloc accept the same set of options. Both long and short options are listed.

Options	Default Value	Description
--nodes=<N> or -N <N>	1	Number of compute nodes
--tasks=<N> or -n <N>	1	Number of tasks to run
--cpus-per-task=<N> or -c <N>	1	Number of CPU cores per task
--ntasks-per-node=<N>	1	Number of tasks per node
--ntasks-per-core=<N>	1	Number of tasks per CPU core
--mem-per-cpu=<mem>	partition dependent	memory per CPU core in MB

Options	Default Value	Description
<code>--mem=&lt;mem&gt;</code>	partition dependent	memory per node in MB
<code>--gres=gpu:&lt;type&gt;:&lt;N&gt;</code>	-	Request nodes with GPUs; <type> may be omitted (thus: <code>--gres=gpu:&lt;N&gt;</code> )
<code>--time=&lt;time&gt;</code> or <code>-t &lt;time&gt;</code>	partition dependent	Walltime limit for the job
<code>--partition=&lt;name&gt;</code> or <code>-p &lt;name&gt;</code>	none	Partition to run the job
<code>--constraint=&lt;list&gt;</code> or <code>-C &lt;list&gt;</code>	none	Node-features to request; to find out the features assigned to a specific node, use e.g. <code>scontrol show nodes &lt;nodename&gt;</code>
<code>--job-name=&lt;name&gt;</code> or <code>-J &lt;name&gt;</code>	job script's name	Name of the job
<code>--output=&lt;path&gt;</code> or <code>-o &lt;path&gt;</code>	<code>slurm-%j.out</code>	Standard output file
<code>--error=&lt;path&gt;</code> or <code>-e &lt;path&gt;</code>	<code>slurm-%j.err</code>	Standard error file
<code>--mail-user=&lt;mail&gt;</code>	your account mail	User's email address
<code>--mail-type=&lt;mode&gt;</code>	-	Event types for notifications
<code>--exclusive</code>	nodes are shared	Exclusive access to node

To obtain a complete list of parameters, refer to the sbatch man page: `man sbatch`

**Note:** if you submit a job with `--mem=0`, it gets access to the complete memory configured in SLURM for each node allocated.

By default, the stdout and stderr file descriptors of batch jobs are directed to `slurm-%j.out` and `slurm-%j.err` files, where %j is set to the SLURM batch job ID number of your job. Both files will be found in the directory in which you launched the job. You can use the options `--output` and `--error` to specify a different name or location. The output files are created as soon as your job starts, and the output is redirected as the job runs so that you can monitor your job's progress. However, due to SLURM performing file buffering, the output of your job will not appear in the output files immediately. To override this behaviour (**this is not recommended in general, especially when the job output is large**), you may use `-u` or `--unbuffered` either as an `#SBATCH` directive or directly on the sbatch command line.

If the option `--error` is not specified, both stdout and stderr will be directed to the file specified by `--output`.

## Example of an OpenMP job

For OpenMP jobs, you will need to set `--cpus-per-task` to a value larger than one and explicitly define the `OMP_NUM_THREADS` variable. The example script launches eight threads, **each** with 2 GiB of memory and a maximum run time of 30 minutes.

[example\\_openmp\\_slurm.sh](#)

```
#!/bin/bash -l
#SBATCH --job-name=test_openmp
#SBATCH --ntasks=1
#SBATCH --cpus-per-task=8
```

```
#SBATCH --mem-per-cpu=2G
#SBATCH --time=00:30:00
#SBATCH --constraint=[CPU_ARCH:avx512|CPU_ARCH:avx2]
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --output test_openmp-job_%j.out
#SBATCH --error test_openmp-job_%j.err

# Change to my work dir
cd $SLURM_SUBMIT_DIR

# Bind your OpenMP threads
export OMP_NUM_THREADS=$SLURM_CPUS_PER_TASK
# Intel compiler specific environment variables
export KMP_AFFINITY=verbose,granularity=core,compact,1
export KMP_STACKSIZE=64m

## Load modules
module load my_module

# Start my application
srun ./my_openmp_app
```

The `srun` command in the script above sets up a parallel runtime environment to launch an application on multiple CPU cores, but on **one** node. For MPI jobs, you may want to use multiple CPU cores on **multiple** nodes. To achieve this, have a look at the following example of an MPI job:

**Note:** `srun` should be used in place of the “traditional” MPI launchers like `mpirun` or `mpiexec`.

## Example of an MPI job

This example requests 10 compute nodes on the lena cluster with 16 cores each and 320 GiB of memory **in total** for a maximum duration of 2 hours.

[example\\_mpi\\_slurm.sh](#)

```
#!/bin/bash -l
#SBATCH --job-name=test_mpi
#SBATCH --partition=lena
#SBATCH --nodes=10
#SBATCH --ntasks-per-node=16
#SBATCH --mem-per-cpu=2G
#SBATCH --time=02:00:00
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --output test_mpi-job_%j.out
#SBATCH --error test_mpi-job_%j.err

# Change to my work dir
```

```
cd $SLURM_SUBMIT_DIR

# Load modules
module load foss/2018b

# Start my MPI application
#
# Note: if you use Intel MPI Library provided by modules up to
intel/2020a, execute srun as
#
# srun --mpi=pmi2 ./my_mpi_app
#
# For all Intel MPI libraries set the environment variable
I_MPI_PMI_LIBRARY=/usr/lib64/libpmi.so before executing srun

srun --cpu_bind=cores --distribution=block:cyclic ./my_mpi_app
```

As mentioned above, you should use the `srun` command instead of `mpirun` or `mpiexec` in order to launch your parallel application.

Within the same MPI job, you can use `srun` to start several parallel applications, each utilizing only a subset of the allocated resources. However, the preferred way is to use a Job Array (see section ). The following example script will run 3 MPI applications simultaneously, each using 64 tasks (4 nodes with 16 cores each), thus totalling to 192 tasks:

#### [example\\_mpi\\_multi\\_srun\\_slurm.sh](#)

```
#!/bin/bash -l
#SBATCH --job-name=test_mpi
#SBATCH --partition=lena
#SBATCH --nodes=12
#SBATCH --ntasks-per-node=16
#SBATCH --mem-per-cpu=2G
#SBATCH --time=00:02:00
#SBATCH --constraint=[CPU_ARCH:avx512|CPU_ARCH:avx2]
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --output test_mpi-job_%j.out
#SBATCH --error test_mpi-job_%j.err

# Change to my work dir
cd $SLURM_SUBMIT_DIR

# Load modules
module load foss/2018b

# Start my MPI application
srun --cpu_bind=cores --distribution=block:cyclic -N 4 --ntasks-per-
node=16 --exclusive ./my_mpi_app_1 &
srun --cpu_bind=cores --distribution=block:cyclic -N 4 --ntasks-per-
```

```
node=16 --exclusive ./my_mpi_app_1 &
srun --cpu_bind=cores --distribution=block:cyclic -N 4 --ntasks-per-
node=16 --exclusive ./my_mpi_app_2 &
wait
```

Note the `wait` command in the script; it results in the script waiting for all previously commands that were started with `&&` (execution in the background) to finish before the job can complete. We kindly ask to take care that the time necessary to complete each subjob is not too different in order not to waste too much valuable cpu time

## Job arrays

Job arrays can be used to submit a number of jobs with the same resource requirements. However, some of these requirements are subject to changes after the job has been submitted. To create a job array, you need to specify the directive `#SBATCH --array` in your job script or use the option `--array` or `-a` on the `sbatch` command line. For example, the following script will create 12 jobs with array indices from 1 to 10, 15 and 18:

### [example\\_jobarray\\_slurm.sh](#)

```
#!/bin/bash -l
#SBATCH --job-name=test_job_array
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=2G
#SBATCH --time=00:20:00
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --array=1-10,15,18
#SBATCH --output test_array-job_%A_%a.out
#SBATCH --error test_array-job_%A_%a.err

# Change to my work dir
cd $SLURM_SUBMIT_DIR

# Load modules
module load my_module

# Start my app
srun ./my_app $SLURM_ARRAY_TASK_ID
```

Within a job script like in the example above, the job array indices can be accessed by the variable `$SLURM_ARRAY_TASK_ID`, whereas the variable `$SLURM_ARRAY_JOB_ID` refers to the job array's master job ID. If you need to limit (e.g. due to heavy I/O on the BIGWORK file system) the maximum number of simultaneously running jobs in a job array, use a `%` separator. For example, the directive `#SBATCH --array 1-50%5` will create 50 jobs, with only 5 jobs active at any given time.

**Note:** the maximum number of jobs in a job array is limited to 300. The index number must be

smaller than 1 million.

## SLURM environment variables

SLURM sets many variables in the environment of the running job on the allocated compute nodes. Table 7.4 shows commonly used environment variables that might be useful in your job scripts. For a complete list, see the “OUTPUT ENVIRONMENT VARIABLES” section in the sbatch man page.

SLURM environment variables

<code>\$SLURM_JOB_ID</code>	Job id
<code>\$SLURM_JOB_NUM_NODE</code>	Number of nodes assigned to the job
<code>\$SLURM_JOB_NODELIST</code>	List of nodes assigned to the job
<code>\$SLURM_NTASKS</code>	Number of tasks in the job
<code>\$SLURM_NTASKS_PER_CORE</code>	Number of tasks per allocated CPU
<code>\$SLURM_NTASKS_PER_NODE</code>	Number of tasks per assigned node
<code>\$SLURM_CPUS_PER_TASK</code>	Number of CPUs per task
<code>\$SLURM_CPUS_ON_NODE</code>	Number of CPUs per assigned node
<code>\$SLURM_SUBMIT_DIR</code>	Directory the job was submitted from
<code>\$SLURM_ARRAY_JOB_ID</code>	Job id for the array
<code>\$SLURM_ARRAY_TASK_ID</code>	Job array index value
<code>\$SLURM_ARRAY_TASK_COUNT</code>	Number of jobs in a job array
<code>\$SLURM_GPUS</code>	Number of GPUs requested

## GPU jobs on the cluster

The LUIS cluster has a number of nodes that are equipped with NVIDIA Tesla GPU Cards.

Currently, 4 nodes containing 2 NVIDIA Tesla V100 and 3 nodes containing 4 A100 cards (each) are available for general use in the partition `gpu` (named `euklid-n00x`). So the resources regularly available to all users are still relatively scarce, and respecting the tip in this page's introduction to start small to first test your jobs really is important — or you will possibly just wait for a long time, only to see your job instantly crashing. So the more pressure you have, the more thorough you should test your job first.

There's also some additional resources available that you might have a fair chance to use. Several institutes have entrusted us with running their nodes in the so-called FCH service (“Forschungscluster-Housing”, consult the LUIS-website for details). These nodes are usually reserved during daytime Mo-Fr 08:00-20:00 for the respective institute, but during the night and on weekends, they participate in the common queue. Whether you can run a job there will mainly depend on the respective institute's own usage, and of course your job has to request a walltime shorter than 12 hours during the week to squeeze in. To try your luck on FCH nodes, omit the `--partition=gpu` request.

Use the following command to display the current status of all nodes in the `gpu` partition and the computing resources they provide, including type and number of GPUs:



```

sinfo --partition gpu -Node --
Format="nodelist:15,memory:8,disk:10,cpusstate:15,gres:30,gresused:40"
NODELIST      MEMORY  TMP_DISK  CPUS(A/I/O/T)  GRES
GRES_USED
euklid-n001    125000    291840    2/38/0/40      gpu:v100:2(S:0-1)
gpu:v100:2(IDX:0-1)
euklid-n002    125000    291840    2/38/0/40      gpu:v100:2(S:0-1)
gpu:v100:2(IDX:0-1)
euklid-n003    125000    291840    2/38/0/40      gpu:v100:2(S:0-1)
gpu:v100:2(IDX:0-1)
euklid-n004    125000    291840    2/38/0/40      gpu:v100:2(S:0-1)
gpu:v100:2(IDX:0-1)
euklid-n005    1025000   3600000   4/44/0/48      gpu:a100m40:4(S:0-1)
gpu:a100m40:4(IDX:0-3)
euklid-n006    1025000   3600000   4/44/0/48      gpu:a100m40:4(S:0-1)
gpu:a100m40:4(IDX:0-3)
euklid-n007    1025000   3600000   4/44/0/48      gpu:a100m40:4(S:0-1)
gpu:a100m40:4(IDX:0-3)

```

To inquire about *all* nodes that have at least one gpu, including those reserved during daytime for FCH, use

```

sinfo --Node --
Format="nodelist:15,memory:8,disk:10,cpusstate:15,gres:30,gresused:30" |
grep -v null

```

To ask for GPU resources, you need to add the directive `#SBATCH --gres=gpu:<type>:n` to your job script, or on the command line, respectively, “n” being the number of GPUs requested. The type of GPU can be omitted. Thus, `#SBATCH --gres=gpu:n` will give you a wider selection of potential nodes to run the job. The following job script requests 2 Tesla V100 GPUs, 8 CPUs in the gpu partition and 30 minutes of wall time:

#### [example\\_gpu\\_slurm.sh](#)

```

#!/bin/bash -l
#SBATCH --job-name=test_gpu
#SBATCH --partition=gpu
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=8
#SBATCH --gres=gpu:v100:2
#SBATCH --mem-per-cpu=2G
#SBATCH --time=00:30:00
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --output test_gpu-job_%j.out
#SBATCH --error test_gpu-job_%j.err

# Change to my work dir
cd $SLURM_SUBMIT_DIR

```

```
# Load modules
module load fosscuda/2018b

# Run GPU application
srun ./my_gpu_app
```

When submitting a job to the gpu partition, you **must** specify the number of GPUs. Otherwise, your job will be rejected at the submission time.

**Note:** on the Tesla V100 nodes, you may currently only request up to 20 CPU cores for each requested GPU.

**Note:** the maximum runtime for jobs in the gpu partition is limited to 48 hours.

## Job status and control commands

This section provides an overview of commonly used SLURM commands that allow you to monitor and manage the status of your batch jobs.

### Query commands

The status of your jobs in the queue can be queried using

```
$ squeue
```

or – if you have array jobs and want to display one job array element per line –

```
$ squeue -a
```

Note that the symbol \$ in the above commands and all other commands below represents the shell prompt. The \$ is NOT part of the specified command, do NOT type it yourself.

The squeue output should look more or less like the following:

```
$ squeue
JOBID PARTITION   NAME     USER ST       TIME  NODES NODELIST(REASON)
  412      gpu     test username PD        0:00      1 (Resources)
  420      gpu     test username PD        0:00      1 (Priority)
  422      gpu     test username R       17:45      1 euklid-n001
  431      gpu     test username R       11:45      1 euklid-n004
  433      gpu     test username R       12:45      1 euklid-n003
  434      gpu     test username R        1:08      1 euklid-n002
  436      gpu     test username R       16:45      1 euklid-n002
```

ST shows the status of your job. JOBID is the number the system uses to keep track of your job. NODELIST shows the nodes allocated to the job, NODES the number of nodes requested and – for jobs in the pending state (PD) – a REASON. TIME shows the time used by the job. Typical job states are

PENDING(PD), RUNNING(R), COMPLETING(CG), CANCELLED(CA), FAILED(F) and SUSPENDED(S). For a complete list, see the "JOB STATE CODES" section in the `squeue` man page.

You can change the default output format and display other job specifications using the option `--format` or `-o`. For example, if you want to additionally view the number of CPUs and the walltime requested:

```
$ squeue --format="%.7i %.9P %.5D %.5C %.2t %.19S %.8M %.10l %R"
JOBID PARTITION NODES  CPUS TRES_PER_NODE ST MIN_MEMORY  TIME
TIME_LIMIT NODELIST(REASON)
  489      gpu      1   32      gpu:2 PD          2G    0:00
20:00 (Resources)
  488      gpu      1    8      gpu:1 PD          2G    0:00
20:00 (Priority)
  484      gpu      1   40      gpu:2 R           1G   16:45
20:00 euklid-n001
  487      gpu      1   32      gpu:2 R           2G   11:09
20:00 euklid-n004
  486      gpu      1   32      gpu:2 R           2G   12:01
20:00 euklid-n003
  485      gpu      1   16      gpu:2 R           1G   16:06
20:00 euklid-n002
```

Note that you can make the `squeue` output format permanent by assigning the format string to the environment variable `SQUEUE_FORMAT` in your `$HOME/.bashrc` file:

```
$ echo 'export SQUEUE_FORMAT="%.7i %.9P %.5D %.5C %.13b %.2t %.19S %.8M
%.10l %R"'>> ~/.bashrc
```

The option `%.13b` in the variable assignment for `SQUEUE_FORMAT` above displays the column `TRES_PER_NODE` in the `squeue` output, which provides the number of GPUs requested by each job.

The following command displays all job steps (processes started using `srun`):

```
squeue -s
```

To display estimated start times and compute nodes to be allocated for your pending jobs, type

```
$ squeue --start
JOBID PARTITION NAME      USER ST          START_TIME  NODES SCHEDNODES
NODELIST(REASON)
  489      gpu test username PD 2020-03-20T11:50:09      1 euklid-n001
(Resources)
  488      gpu test username PD 2020-03-20T11:50:48      1 euklid-n002
(Priority)
```

A job may be waiting for execution in the pending state for a number of reasons. If there are multiple reasons for the job to remain pending, only one is displayed.

- **Priority** - the job has not yet gained a high enough priority in the queue
- **Resources** - the job has sufficient priority in the queue, but is waiting for resources to become

available

- **JobHeldUser** - job held by user
- **Dependency** - job is waiting for another job to complete
- **PartitionDown** - the queue is currently closed for new jobs

For the complete list, refer to the `squeue` man page the section "JOB REASON CODES".

If you want to view more detailed information about each job, use

```
$ scontrol -d show job
```

If you are interested in the detailed status of one specific job, use

```
$ scontrol -d show job <job-id>
```

Replace `<job-id>` by the ID of your job.

Note that the command `scontrol show job` will display the status of jobs for up to 5 minutes after their completion. For batch jobs that finished more than 5 minutes ago, you need to use the `sacct` command to retrieve their status information from the SLURM database (see section ).

The `sstat` command provides real-time status information (e.g. CPU time, Virtual Memory (VM) usage, Resident Set Size (RSS), Disk I/O, etc.) for running jobs:

```
# show all status fields
sstat -j <job-id>

# show selected status fields
sstat --format=AveCPU,AvePages,AveRSS,AveVMSize,JobID -j <job-id>
```

**Note:** the above commands only display your own jobs in the SLURM job queue.

## Job control commands

The following command cancels a job with ID number `<job-id>`:

```
$ scancel <job-id>
```

Remove all of your jobs from the queue at once using

```
$ scancel -u $USER
```

If you want to cancel only array ID `<array_id>` of job array `<job_id>`:

```
$ scancel <job_id>_<array_id>
```

If only job array ID is specified in the above command, then all job array elements will be canceled.

The commands above first send a `SIGTERM` signal, then wait 30 seconds, and if processes from the job continue to run, issue a `SIGKILL` signal.

The `-s` option allows you to issue any signal to a running job which means you can directly communicate with the job from the command line, provided that it has been prepared for this:

```
$ scancel -s <signal> <job-id>
```

A job in the pending state can be held (prevented from being scheduled) using

```
$ scontrol hold <job-id>
```

To release a previously held job, type

```
$ scontrol release <job-id>
```

After submitting a batch job and while the job is still in the pending state, many of its specifications can be changed. Typical fields that can be modified include job size (amount of memory, number of nodes, cores, tasks and GPUs), partition, dependencies and wall clock limit. Here are a few examples:

```
# modify time limit
scontrol update JobId=279 TimeLimit=12:0:0

# change number of tasks
scontrol update jobid=279 NumTasks=80

# change node number
scontrol update JobId=279 NumNodes=2

# change the number of GPUs per node
scontrol update JobId=279 Gres=gpus:2

# change memory per allocated CPU
scontrol update Jobid=279 MinMemoryCPU=4G

# change the number of simultaneously running jobs of array job 280
scontrol update ArrayTaskThrottle=8 JobId=280
```

For a complete list of job specifications that can be modified, see section “SPECIFICATIONS FOR UPDATE COMMAND, JOBS” in the `scontrol` man page.

## Job accounting commands

The `sacct` command is primarily designed to display job data from the SLURM accounting database, specifically for jobs that have exited the queue (e.g., completed, failed, or canceled). If a job is still running, tools like `sstat` or `squeue` might provide more current metrics. Here are a few usage examples:

```
# list IDs of all your jobs since January 2019
sacct -S 2019-01-01 -o jobid

# show brief accounting data of the job with <job-id>
```

```
sacct -j <job-id>

# display all job accounting fields
sacct -j <job-id> -o ALL
```

The complete list of job accounting fields can be found in section “Job Accounting Fields” in the `sacct` man page. You could also use the command `sacct --helpformat`

## Analyzing Job Efficiency

Monitoring job efficiency helps reduce queue waiting times by identifying resource allocation mismatches (e.g., over-requesting CPUs or memory). Efficient resource utilization not only ensures faster job completion but also increases overall system throughput, enabling more users to benefit from the cluster.

### Seff

`seff` is a command-line tool used to display resource utilization efficiency for completed jobs.

**Note:** The job must be completed; `seff` does not work for running jobs.

Syntax

```
seff <job_id>
```

Sample Output

```
Job ID: 12345
Cluster: luis
User/Group: user/group
State: COMPLETED
Nodes: 1
Cores: 8
CPU Utilized: 02:00:00
CPU Efficiency: 25% of 08:00:00 core-walltime
Memory Utilized: 4 GB
Memory Efficiency: 50% of 8 GB requested
```

- **CPU Efficiency:** Calculated as the ratio of CPU time used to the total CPU time allocated (cores × walltime). Low efficiency indicates under-utilized cores.
- **Memory Efficiency:** Indicates how much of the requested memory was actually consumed. Over-requesting memory can lead to wasted resources.

### Reportseff

The `reportseff` command is a tool available on the cluster to help users analyze the efficiency of their Slurm jobs. While `seff` focuses on a single job, allowing you to evaluate resource utilization per

job ID, reportseff provides broader capabilities, including:

- Analyzing jobs over a specified time period (e.g., --since and --until options)
- Filtering jobs based on partition, job state, or multiple job IDs simultaneously
- Generating efficiency details for a single or all array job elements

The tool reads accounting data via sacct and is particularly helpful for identifying how effectively resources are being used, enabling users to adjust job submissions for optimal performance.

This example generates a report for jobs completed in the last 3 days, including additional fields for the requested time limit (timelimit), CPUs (reqcpus), and memory (reqmem)

```
reportseff --since now-3days --until now --state COMPLETED --format
+timelimit,reqcpus,reqmem
```

JobID	State	Elapsed	TimeEff	CPUEff	MemEff
Timelimit	ReqCPUS	ReqMem			
4043949	COMPLETED	6-01:43:06	97.1%	23.0%	15.3%
6-06:00:00	64	512G			
4048121	COMPLETED	5-00:21:51	60.2%	20.8%	100.0%
8-08:00:00	48	128G			
4056804	COMPLETED	4-21:07:01	61.0%	87.7%	15.0%
8-00:00:00	12	48G			
4059203	COMPLETED	5-05:39:14	65.4%	13.6%	9.2%
8-00:00:00	8	128G			
4059230	COMPLETED	4-19:06:34	60.0%	8.9%	58.3%
8-00:00:00	8	128G			
4059298	COMPLETED	5-07:27:45	66.4%	13.2%	25.3%
8-00:00:00	8	128G			
4059303	COMPLETED	4-16:25:54	58.6%	13.0%	23.2%
8-00:00:00	8	128G			
4059317	COMPLETED	4-23:42:34	62.3%	12.9%	20.9%
8-00:00:00	8	128G			
4066049	COMPLETED	3-10:06:40	85.5%	18.4%	3.0%
4-00:00:00	16	64G			
4067800	COMPLETED	3-05:45:30	46.3%	99.5%	12.0%
7-00:00:00	36	108G			
...					

The tool is pre-installed and ready for use on the cluster. For further information, refer to the [reportseff GitHub page](#).

## How the scheduler works

The scheduler has to consider many constraints, rules, priorities and limits until a particular job gets scheduled. The definitive guide can be found on [SLURM's website](#). Some of the factors to consider when you want to ask “why does my job not run?” are:

→ First, there's priorities. Compare your job's priority against that of others (e.g. with a command like `queue -states=pending,configuring -sort=-p,-i -priority -Format="JobID:11`

,UserName:9 ,StateCompact:3 ,NumNodes:.3 ,NumCPUs:.4 ,MinCPUs:.5 ,MinMemory:.5 ,TimeLimit:.11 ,SubmitTime:.20 ,StartTime:.20 ,PriorityLong:8 ,TRES-per-node:20 ,Partition:15"). If other jobs have higher priority, they will get considered first. Only if jobs with higher priority have resource requests that currently can not be fulfilled, the next jobs in order will get considered. Jobs that do not yet have priority, but could instantly run on currently free resources and finish before any prioritized job could use them, may get run instantly via a mechanism called "backfill". The important thing is that they will ONLY run if they block no priority job in ANY of their respective dimensions, like walltime, cpu count, memory, ...

→ Next, check whether your job may have resource requests that are difficult to fulfill. Compare the resources listed in our [Available Hardware Table](#) and ask on which partitions your job could run at all. If you want to configure your jobs to use a particular partition, e.g. because you know your software can use cpu-specific features like AVX-512, try the commands `scontrol show partition <partitionname>` and `scontrol show node <nodename-n001>` to show which resources every node in the cluster exactly provides.

→ If you are in doubt whether any of your jobs would run, you could try to submit a very small and short (!) job (ex. 1 cpu core, 2 GB mem, 10 minutes, like `salloc -nodes=1 -ntasks=1 -mem=2GB -time=10:00`). Even with such a job, the "problem" could just be that the resources of the cluster are not infinitely large. A future reservation for somebody else may prevent your job from starting immediately. Or, in case some nodes have not been used for a while, they will have been powered-down automatically to save energy. That means that if you just see a message "queued and waiting for resources", it may also mean that in the background, a node has begun to boot just for your job, and in about 10 minutes time, your job and any of the same kind you would care to submit afterwards will start almost instantly.

→ We take only moderate influence on how many jobs our users may submit. The scheduler will try to at least run at least one job of every user before turning to the next job of the same user, provided the resources for other jobs are available. In case the cluster is really empty, up to 64 jobs of one user will run at the same time, and if they all request 200 hours of wall time, that may in extreme cases mean that others will wait for a correspondingly long time until new resources become free.

Generally speaking, the cluster is a shared tool for hundreds of users. We often get requests ("the resources are free, why does my job not start?") that show a lack of awareness that at any instant, there possibly are many users on the system waiting for many jobs' execution.

So the conclusion is: just because something matching the requirements of your job currently appears to be free, this does NOT automatically mean that the scheduler will pick or even consider exactly *your* job as the next to run. Scheduling is a little bit like a more complicated version of stock trading, in case the comparison helps someone...

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/350\\_slurm\\_usage\\_guide](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/350_slurm_usage_guide)

Last update: **2025/07/24 09:44**





# Modules & Application Software

---

The number of software packages that are installed together with the operating system on cluster nodes is kept light on purpose. Additional packages and applications are provided by a module system, which enables you to easily customise your working environment on the cluster. This module system is called Lmod<sup>2)</sup>. Furthermore, we can provide different versions of the software which you can use on demand. Loading a module, software specific settings are applied, e.g. changing environment variables like PATH, LD\_LIBRARY\_PATH and MANPATH.

Alternatively, you can manage software packages on the cluster yourself by building software from source, by means of EasyBuild or by using Singularity containers. Python packages can also be installed using the Conda manager. The first three possibilities, in addition to the module system, are described in the current section, whereas Conda usage in the cluster is explained in [this section](#).

We have adopted a systematic software naming and versioning convention in conjunction with the software installation system EasyBuild<sup>3)</sup>.

Software installation on the cluster utilizes a hierarchical software module naming scheme. This means that the command `module avail` does not display all installed software modules right away. Instead, only the modules that are immediately available for loading are displayed. More modules become available after their prerequisite modules are loaded. Specifically, loading a compiler module or MPI implementation module will make available all the software built with those applications. This way, we hope the prerequisites for certain software become apparent.

At the top level of the module hierarchy, there are modules for compilers, toolchains and software applications that come as a binary and thus do not depend on compilers. Toolchain modules organize compilers, MPI implementations and numerical libraries. Currently the following toolchain modules are available:

- Compiler only toolchains
  - GCC: GCC updated binutils
  - iccifort: Intel compilers, GCC
- Compiler & MPI toolchains
  - gomp: GCC, OpenMPI
  - iimpi: iccifort, Intel MPI
  - iompi: iccifort, OpenMPI
- Compiler & MPI & numerical libraries toolchains
  - foss: gomp, OpenBLAS, FFTW, ScaLAPACK
  - intel: iimpi, Intel MKL
  - iomkl: iompi, Intel MKL

Note that Intel compilers newer than 2020.x are provided by the toolchain module `intel-compilers`. It is strongly recommended to use this module as after 2023 the intel compiler modules `iccifort` will be removed.

## Working with modules

This section explains how to use software modules.

List the entire list of possible modules

```
module spider
```

The same in a more compact list

```
module -t spider
```

Search for specific modules that have “string” in their name

```
module spider string
```

Detailed information about a particular version of a module (including instructions on how to load the module)

```
module spider name/version
```

Searches for all module names and descriptions that contain the specified string

```
module key string
```

List modules immediately available to load

```
module avail
```

Some software modules are hidden from the `avail` and `spider` commands. These are mostly modules for system library packages that other user applications depend on. To list hidden modules, you may provide the `--show-hidden` option to the `avail` and `spider` commands:

```
module --show-hidden avail  
module --show-hidden spider
```

A hidden module has a dot (.) in front of its version numbers (eg. `zlib/.1.2.8`).

List currently loaded modules

```
module list
```

Load a specific version of a module

```
module load name/version
```

If only a name is given, the command will load the default version which is marked with a (D) in the `module avail` listing (usually the latest version). Loading a module may automatically load other modules it depends on.

It is not possible to load two versions of the same module at the same time.

To switch between different modules

```
module swap old new
```

To unload the specified module from the current environment

```
module unload name
```

To clean your environment of all loaded modules

```
module purge
```

Show what environment variables the module will set

```
module show name/version
```

Save the current list of modules to “name” collection for later use

```
module save name
```

Restore modules from collection “name”

```
module restore name
```

List of saved collections

```
module savelist
```

To get the complete list of options provided by Lmod through the command `module` type the following

```
module help
```

## Exercise: Working with modules

As an example, we show how to load the `gnuplot` module.

List loaded modules

```
module list
```

```
No modules loaded
```

Find available `gnuplot` versions

```
module -t spider gnuplot
```

```
gnuplot/4.6.0
gnuplot/5.0.3
```

Determine how to load the selected gnuplot/5.0.3 module

```
module spider gnuplot/5.0.3

-----
----
gnuplot: gnuplot/5.0.3
-----
----
Description:
  Portable interactive, function plotting utility - Homepage:
http://gnuplot.sourceforge.net/

  This module can only be loaded through the following modules:

  GCC/4.9.3-2.25  OpenMPI/1.10.2

Help:
  Portable interactive, function plotting utility - Homepage:
http://gnuplot.sourceforge.net/
```

Load required modules

```
module load GCC/4.9.3-2.25  OpenMPI/1.10.2

Module for GCCcore, version .4.9.3 loaded
Module for binutils, version .2.25 loaded
Module for GCC, version 4.9.3-2.25 loaded
Module for numactl, version .2.0.11 loaded
Module for hwloc, version .1.11.2 loaded
Module for OpenMPI, version 1.10.2 loaded
```

And finally load the selected gnuplot module

```
module load gnuplot/5.0.3

Module for OpenBLAS, version 0.2.15-LAPACK-3.6.0 loaded
Module for FFTW, version 3.3.4 loaded
Module for ScaLAPACK, version 2.0.2-OpenBLAS-0.2.15-LAPACK-3.6.0 loaded
Module for bzip2, version .1.0.6 loaded
Module for zlib, version .1.2.8 loaded
.....
.....
```

In order to simplify the procedure of loading the gnuplot module, the current list of loaded modules can be saved in a “mygnuplot” collection (the name string “mygnuplot” is, of course, arbitrary) and then loaded again when needed as follows:

Save loaded modules to “mygnuplot”

```
module save mygnuplot
```

```
Saved current collection of modules to: mygnuplot
```

If “mygnuplot” not is specified, the name “default” will be used.

Remove all loaded modules (or open a new shell)

```
module purge
```

```
Module for gnuplot, version 5.0.3 unloaded
```

```
Module for Qt, version 4.8.7 unloaded
```

```
Module for libXt, version .1.1.5 unloaded
```

```
.....
```

```
.....
```

List currently loaded modules. This selection is empty now.

```
module list
```

```
No modules loaded
```

List saved collections

```
module savelist
```

```
Named collection list:
```

```
1) mygnuplot
```

Load gnuplot module again

```
module restore mygnuplot
```

```
Restoring modules to user's mygnuplot
```

```
Module for GCCcore, version .4.9.3 loaded
```

```
Module for binutils, version .2.25 loaded
```

```
Module for GCC, version 4.9.3-2.25 loaded
```

```
Module for numactl, version .2.0.11 loaded
```

```
.....
```

```
.....
```

## List of available software

---

In this section, you will find user guides for some of the software packages installed in the cluster. The guides provided can, of course, not replace documentation that comes with the application. Please

read that as well.

A wide variety of application software is available in the cluster system. These applications are located on a central storage system that is accessible by the module system Lmod via an NFS export. Issue the command `module spider` on the cluster system or visit the [page](#) for a comprehensive list of available software. If you really need a different version of an already installed application, or one that is currently not installed, please get in touch. The main prerequisite for use of a software within the cluster system is its availability for Linux. Furthermore, if the application requires a license, we need to clarify additional questions.

Some select Windows applications can also be executed on the cluster system with the help of Wine or Singularity containers. For information on Singularity, see [Singularity Containers](#).

## A current list of available software

## Usage instructions

- [Abaqus](#)
- [ANSYS / CFX](#)
- [COMSOL](#)
- [Conda](#)
- [Conda](#)
- [CPMD](#)
- [FEKO](#)
- [Jupyter in the cluster](#)
- [MATLAB](#)
- [Moose](#)
- [mpiFileUtils](#)
- [NFFT](#)

## Build software from source code

**Note:** We recommend using [EasyBuild](#) (see next section) if you want to make your software's build process reproducible and accessible through a module environment that EasyBuild automatically creates. EasyBuild comes with pre-configured recipes for installing thousands of scientific applications.

Sub-clusters of the cluster system have different CPU architectures that provide different instruction set capabilities/extensions. The command `lcpuarchs` (available on the login nodes) lists all available CPU types.

```
login03:~$ lcpuarchs -v
CPU arch names      Cluster partitions
-----
CPU arch names      Cluster partitions
-----
```

```

sse                LUIS[smp,helena]
                   FCH[]

avx                LUIS[dumbo]
                   FCH[iazd,isu,itp]

avx2               LUIS[haku,lena,smp]
                   FCH[ai,gih,isd,iqo,iwes,pci,fi,imuk]

avx512             LUIS[gpu,gpu.test,amo,taurus,vis]
                   FCH[tnt,isd,stahl,enos,pcikoe,pcikoe,isu,phd,phdgpu,muk,fi,itp]

CPU of this machine: avx2

For more verbose output type: lcpuarchs -vv

```

The technical sequence of these architectures is (oldest) -sse-avx-avx2-avx512- (newest). Cpus capable of executing instructions from newer instruction sets usually are able to execute commands from older extensions, so e.g. avx512 includes sse.

Software compiled to use a newer cpu instruction set will usually not typically abort with an “**Illegal instruction**” error when run on an older cpu. The important message here is that compilers may automatically set flags for the platform you are currently working on. If you compile your program on a node providing avx512 instructions (e. g. the amo sub-cluster) using the gcc compiler option -march=native, the program will usually not run on older nodes that are only equipped with cpus providing, say, avx instructions. To check which instruction set extensions a cpu architecture provides, you can run the command “lscpu”, which lists them in the “flags” section.

This section explains how to build a software on the cluster system to avoid the aforementioned issue if you want to be able to submit jobs to all compute nodes without specifying the CPU type. Beware, though, that compatibility usually comes at a price and allowing a compiler to use the newer instruction sets will usually boost performance. Depending on your workload, the effects/speedup on newer cpu architectures may even be called “drastic”. But there's usually no better way to tell than testing.

In the example below we want to compile a sample software my-soft in version 3.1.

In your [HOME](#) (or, perhaps better, in your [\\$SOFTWARE](#) directory, if all members of your project want to use the software) directory, create build/install directories for each available CPU architecture listed by the command `lcpuarchs -s`, as well as a directory source to storing the installation sources

**Note:** you can usually refer to a variable using `$variable_name` and all will be well. In the following examples, however, we demonstrate the use of curly brackets around the variable's designator, which will ensure proper separation of variables even in case of ambiguities (which in theory could occur in long paths composed out of several variables). For all normal purposes, `$HOME` and `${HOME}` or `$LUIS_CPU_ARCH` and `${LUIS_CPU_ARCH}` will be equivalent. If, however, you use spaces in your directories (like `dir="my directory"`), this will not be sufficient, you'll then also need to put double quotation marks “ around the variable (like in `cd "${dir}"`).

```

login03:~$ mkdir -p ${HOME}/sw/source
login03:~$ eval "mkdir -p ${HOME}/sw/${$(lcpuarchs -ss)}/my-

```

```
soft/3.1/{build,install}"
```

Copy software installation archive to the source directory

```
login03:~$ mv my-soft-3.1.tar.gz ${HOME}/sw/source
```

Build my-soft for each available CPU architecture by submitting an interactive job to each compute node type requesting the proper CPU type. For example, to compile my-soft for avx512 nodes, first submit an interactive job requesting the avx512 feature:

```
login03:~$ salloc --nodes=1 --constraint=CPU_ARCH:avx512 --cpus-per-task=4  
--time=6:00:00 --mem=16G
```

Then unpack and build the software. Note the environment variable `${LUI5_CPU_ARCH}` that contains the cpu instruction set of the compute node reserved.

```
taurus-n034:~$ tar -zxvf ${HOME}/sw/source/my-soft-3.1.tgz -C  
${HOME}/sw/${LUI5_CPU_ARCH}/my-soft/3.1/build  
taurus-n034:~$ cd ${HOME}/sw/${LUI5_CPU_ARCH}/my-soft/3.1/build  
taurus-n034:~$ ./configure --prefix=${HOME}/sw/${LUI5_CPU_ARCH}/my-  
soft/3.1/install && make && make install
```

Finally, use the environment variable `${LUI5_CPU_ARCH}` in your job scripts to access the correct installation path of my-soft executable for the current compute node. Note that you may need to set/update the `${LD_LIBRARY_PATH}` environment variable to point to the location of your software's shared libraries.

### my-soft-job.sh

```
#!/bin/bash -l  
#SBATCH --job-name=my-soft  
#SBATCH --nodes=1  
#SBATCH --ntasks-per-node=16  
#SBATCH --mem=60G  
#SBATCH --time=12:00:00  
#SBATCH --constraint="[CPU_ARCH:avx512|CPU_ARCH:avx2]"  
#SBATCH --output my-soft-job_%j.out  
#SBATCH --error my-soft-job_%j.err  
#SBATCH --mail-user=myemail@...uni-hannover.de  
#SBATCH --mail-type=BEGIN,END,FAIL  
  
# Change to work dir  
cd ${SLURM_SUBMIT_DIR}  
  
# Load modules  
module load my_necessary_modules  
  
# run my_soft  
export LD_LIBRARY_PATH=${HOME}/sw/${LUI5_CPU_ARCH}/my-  
soft/3.1/install/lib:${LD_LIBRARY_PATH}
```



```
srun $HOME/sw/${LUIS_CPU_ARCH}/my-soft/3.1/install/bin/my-soft.exe --
input file.input
```

You can certainly consider combining the software build and execution steps into a single batch job script. However, it is recommended that you first perform the build steps **interactively** before adding them to a job script to ensure the software compiles without errors. For example, such a job script might look like this:

#### my-soft-job.sh

```
#!/bin/bash -l
#SBATCH --job-name=my-soft
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=32
#SBATCH --mem=120G
#SBATCH --time=12:00:00
#SBATCH --constraint=CPU_ARCH:avx512
#SBATCH --output my-soft-job_%j.out
#SBATCH --error my-soft-job_%j.err
#SBATCH --mail-user=myemail@...uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL

# Change to work dir
cd ${SLURM_SUBMIT_DIR}

# Load modules
module load my_necessary_modules

# install software if the executable does not exist
[ -e "${HOME}/sw/${LUIS_CPU_ARCH}/my-soft/3.1/install/bin/my-soft.exe" ] || {
    mkdir -p ${HOME}/sw/${LUIS_CPU_ARCH}/mysoft/3.1/{build,install}
    tar -zxvf ${HOME}/sw/source/my-soft-3.1.tgz -C
    ${HOME}/sw/${LUIS_CPU_ARCH}/my-soft/3.1/build
    cd $HOME/sw/${LUIS_CPU_ARCH}/my-soft/3.1/build
    ./configure --prefix=${HOME}/sw/${LUIS_CPU_ARCH}/my-soft/3.1/install
    make
    make install
}

# run my_soft
export LD_LIBRARY_PATH=${HOME}/sw/${LUIS_CPU_ARCH}/my-
soft/3.1/install/lib:${LD_LIBRARY_PATH}
srun ${HOME}/sw/${LUIS_CPU_ARCH}/my-soft/3.1/install/bin/my-soft.exe --
input file.input
```

# EasyBuild

**Note:** If you want to manually build the software from source code, please refer to the [section](#) above.

EasyBuild is a software build and installation framework that allows you to manage (scientific) software on High Performance Computing (HPC) systems in an efficient way.

## EasyBuild framework

The EasyBuild framework is available in the cluster through the module EasyBuild-custom. This module defines the location of the EasyBuild configuration files, recipes and installation directories. You can load the module using the command:

```
module load EasyBuild-custom
```

EasyBuild software and modules will be installed by default under the following directory:

```
$HOME/my.soft/software/${LUIS_CPU_ARCH}
$HOME/my.soft/modules/${LUIS_CPU_ARCH}
```

Here, the variable ARCH, which stores the CPU type of the machine on which the above module load command was executed, will currently be either haswell, sandybridge or skylake. The command `lcpuarchs` executed on the cluster login nodes lists all currently available values of ARCH. You can override the default software and module installation directory, and the location of your EasyBuild configuration files (MY\_EASYBUILD\_REPOSITORY) by exporting the following environment variables before loading the EasyBuild module:

```
export EASYBUILD_INSTALLPATH=/your/preferred/installation/dir
export MY_EASYBUILD_REPOSITORY=/your/easybuild/repository/dir
module load EasyBuild-custom
```

If other project members should also have access to the software, the recommended location is a subdirectory in [\\$SOFTWARE](#).

## How to build your software

After you load the EasyBuild environment as explained in the section above, you will have the command `eb` available to build your code using EasyBuild. If you want to build the code using a given configuration `<filename>.eb` and resolving dependencies, you will use the flag `-r` as in the example below:

```
eb <filename>.eb -r
```

The build command just needs the configuration file name with the extension `.eb` and not the full path, provided that the configuration file is in your search path: the command `eb --show-config` will print the variable `robot-paths` that holds the search path. More options are available - please have a look at the short help message typing `eb -h`. For instance, using the search flag `-S`, you can

check if any EasyBuild configuration file already exists for a given program name:

```
eb -S <program_name>
```

You will be able to load the modules created by EasyBuild in the directory defined by the EASYBUILD\_INSTALLPATH variable using the following commands:

```
module use $EASYBUILD_INSTALLPATH/modules/${LUIS_CPU_ARCH}/all
module load <modulename>/version
```

The command `module use` will prepend the selected directory to your `MODULEPATH` environment variable, therefore the command `module avail` will show modules of your software as well.

If you want the software module to be automatically available when opening a new shell in the cluster, modify your `~/ .bashrc` file as follows:

```
echo 'export EASYBUILD_INSTALLPATH=/your/preferred/installation/dir' >>
~/.bashrc
echo 'module use $EASYBUILD_INSTALLPATH/modules/${LUIS_CPU_ARCH}/all' >>
~/.bashrc
```

Note that to preserve the dollar sign in the second line above, the string must be enclosed in single quotes.

## Further Reading

- [EasyBuild documentation](#)
- [Easyconfigs repository](#)

## Apptainer Containers (replaces Singularity)



Apptainer will replace Singularity on the the LUH-Clusters. Currently you can use both commands `apptainer` and `singularity` because the last one is a symlink to `apptainer`. This may change in the future.

**Please note:** This instruction has been written for Apptainer 1.3.3-\*

**Please note:** If you would like to fully manage your apptainer container images directly on the cluster, including build and/or modify actions, please contact us and ask for the permission “apptainer fakeroot” to be added to your account (because you will need it).

## Apptainer containers on the cluster

Apptainer enables users to execute containers on High-Performance Computing (HPC) cluster like they are native programs or scripts on a host computer. For example, if the cluster system is running

CentOS Linux, but your application runs in Ubuntu, you can create an Ubuntu container image, install your application into that image, copy the image to an approved location on the cluster and run your application using Apptainer in its native Ubuntu environment.

The main advantage of Apptainer is that containers are executed as an unprivileged user on the cluster system and, besides the local storage TMPDIR, they can access the network storage systems like HOME, BIGWORK and PROJECT, as well as GPUs that the host machine is equipped with.

Additionally, Apptainer properly integrates with the Message Passing Interface (MPI), and utilizes communication fabrics such as InfiniBand and Intel Omni-Path.

If you want to create a container and set up an environment for your jobs, we recommend that you start by reading [the Apptainer documentation](#). The basic steps to get started are described below.

## Building Apptainer container using a recipe file

If you already have a pre-build container ready for use, you can simply upload the container image to the cluster and execute it. See the [section](#) below about running container images.

Below we will describe how to build a new or modify an existing container directly on the cluster. A container image can be created from scratch using a recipe file, or fetched from some remote container repository. In this sub-section, we will illustrate a recipe file method. In the next one, we will take a glance at remote container repositories.

Using a Apptainer recipe file is the recommended way to create containers if you want to build reproducible container images. This example recipe file builds a RockyLinux 9 container:

[rocky9.def](#)

```
BootStrap: yum
OSVersion: 9
MirrorURL: https://ftp.uni-
hannover.de/rocky/{OSVERSION}/BaseOS/$basearch/os
Include: yum

%setup
    echo "This section runs on the host outside the container during
bootstrap"

%post
    echo "This section runs inside the container during bootstrap"

    # install packages in the container
    yum -y groupinstall "Development Tools"
    yum -y install wget vim python3 epel-release
    yum -y install python3-pip

    # install tensorflow
    pip3 install --upgrade tensorflow
```

```
# enable access to BIGWORK and PROJECT storage on the cluster system
mkdir -p /bigwork /project

%runscript
echo "This is what happens when you run the container"

echo "Arguments received: $*"
exec /usr/bin/python3 "$@"

%test
echo "This test will be run at the very end of the bootstrapping
process"

/usr/bin/python3 --version
```

This recipe file uses the yum bootstrap module to bootstrap the core operation system, RockyLinux 9, within the container. For other bootstrap modules (e.g.. docker) and details on apptainer recipe files, refer to [the online documentation](#).

The next step is to build a container image on one of the cluster login servers.

**Note:** your account must be authorized to use the --fakeroot option. Please contact us at [cluster-help@luis.uni-hannover.de](mailto:cluster-help@luis.uni-hannover.de).

**Note:** Currently, the --fakeroot option is enabled only on the cluster login nodes.

```
username@login01$ apptainer build --fakeroot rocky9.sif rocky9.def
```

This creates an image file named `rocky9.sif`. By default, apptainer containers are built as read-only SIF (Apptainer Image Format) image files. Having a container in the form of a file makes it easier to transfer it to other locations both within the cluster and outside of it. Additionally, a SIF file can be signed and verified.

Note that a container as the SIF file can be built on any storage of the cluster you have a write access to. However, it is recommended to build containers either in your `$BIGWORK` or in some directory under `/tmp` (or use the variable `$MY_APPTAINER`) on the login nodes.

**Note:** Containers located only under the paths `$BIGWORK`, `$SOFTWARE` and `/tmp` are allowed to be executed using `shell`, `run` or `exec` commands, see the [section](#) below,

The latest version of the `apptainer` command can be used directly on any cluster node without prior activation.

## Downloading containers from external repositories

Another easy way to obtain and use a Apptainer container is to retrieve pre-build images directly from external repositories. Popular repositories are [Docker Hub](#) or [Apptainer Library](#). You can go there and search if they have a container that meets your needs. For docker images, use the [search form at Docker Hub](#) instead.

In the following example we will pull the latest python container from Docker Hub and save it in a file named `python_latest.sif`:

```
username@login01$ apptainer pull docker://python:latest
```

The build sub-command can also be used to download images, where you can additionally specify your preferred container file name:

```
username@login01$ apptainer build my-ubuntu22.04.sif  
library://library/default/ubuntu:22.04
```

## How to modify existing Apptainer images

First you should check if you really need to modify the container image. For example, if you are using Python in an image and simply need to add new packages via `pip` you can do that without modifying the image by running `pip` in the container with the `--user` option.

To modify an existing SIF container file, you need to first convert it to a writable sandbox format.

**Please note:** Since the `--fakeroot` option of the `shell` and `build` sub-commands does not work with container sandbox when the container is located on a shared storage such as `BIGWORK`, `PROJECT` or `HOME`, the container sandbox must be stored locally on the login nodes. We recommend using the `/tmp` directory (or variable `$MY_APPTAINER`) which has sufficient capacity.

```
username@login01$ cd $MY_APPTAINER  
username@login01$ apptainer build --sandbox rocky9-sandbox rocky9.sif
```

The build command above creates a sandbox directory called `rocky9-sandbox` which you can then shell into in writable mode and modify the container as desired:

```
username@login01$ apptainer shell --writable --fakeroot rocky9-sandbox  
Apptainer> yum install -qy python3-matplotlib
```

After making all desired changes, you exit the container and convert the sandbox back to the SIF file using:

```
Apptainer> exit  
username@login01$ apptainer build -F --fakeroot rocky9.sif rocky9-sandbox
```

**Note:** you can try to remove the sandbox directory `rocky9-sandbox` afterward but there might be a few files you can not delete due to the namespace mappings that happens. The daily `/tmp` cleaner job will eventually clean it up.

## Running container images

**Please note:** In order to run a Apptainer container, the container SIF file or sandbox directory must be located either in your `$BIGWORK`, in your group's `$SOFTWARE` or in the `/tmp` directory.

There are four ways to run a container under Apptainer.

If you simply call the container image as an executable or use the Apptainer run sub-command it will carry out instructions in the %runscript section of the container recipe file:

How to call the container SIF file:

```
username@login01:~$ ./rocky9.sif --version
This is what happens when you run the container
Arguments received: --version
Python 3.8.6
```

Use the run sub-command:

```
username@login01:~$ apptainer run rocky9.sif --version
This is what happens when you run the container
Arguments received: --version
Python 3.8.6
```

The Apptainer exec sub-command lets you execute an arbitrary command within your container instead of just the %runscript. For example, to get the content of file /etc/os-release inside the container:

```
username@login01:~$ apptainer exec rocky9.sif cat /etc/os-release
NAME="Rocky Linux"
VERSION="8.4 (Green Obsidian)"
....
```

The Apptainer shell sub-command invokes an interactive shell within a container. Note the Apptainer> prompt within the shell in the example below:

```
username@login01:$ apptainer shell rocky9.sif
Apptainer>
```

Note that all three sub-commands shell, exec and run let you execute a container directly from remote repository without first downloading it on the cluster. For example, to run an one-liner "Hello World" ruby program:

```
username@login01:$ apptainer exec library://sylabs/examples/ruby ruby -e
'puts "Hello World!'"
Hello World!
```

**Please note:** You can access (read & write mode) your HOME, BIGWORK and PROJECT (only login nodes) storage from inside your container. In addition, the /tmp (or TMPDIR on compute nodes) directory of a host machine is automatically mounted in a container. Additional mounts can be specified using the --bind option of the exec, run and shell sub-commands, see `apptainer run --help`.

## Apptainer & parallel MPI applications

In order to containerize your parallel MPI application and run it properly on the cluster system you have to provide MPI library stack inside your container. In addition, the userspace driver for Mellanox InfiniBand HCAs should be installed in the container to utilize cluster InfiniBand fabric as a MPI transport layer.

This example Apptainer recipe file `ubuntu-openmpi.def` retrieves an Ubuntu container from Docker Hub, and installs required MPI and InfiniBand packages:

## Ubuntu 20.04

[ubuntu-openmpi.def](#)

```
BootStrap: docker
From: ubuntu:focal

%post
# install openmpi & infiniband
apt-get update
apt-get -y install openmpi-bin openmpi-common libibverbs1 libmlx4-1

# enable access to BIGWORK storage on the cluster
mkdir -p /bigwork /project

# enable access to /scratch dir. required by mpi jobs
mkdir -p /scratch
```

## Ubuntu 22.x - 24.x

[ubuntu-openmpi.def](#)

```
BootStrap: docker
From: ubuntu:latest

%post
# install openmpi & infiniband
apt-get update
apt-get -y install openmpi-bin openmpi-common ibverbs-providers

# enable access to BIGWORK storage on the cluster
mkdir -p /bigwork /project

# enable access to /scratch dir. required by mpi jobs
```



```
mkdir -p /scratch
```

Once you have built the image file `ubuntu-openmpi.sif` as explained in the previous sections, your MPI application can be run as follows (assuming you have already reserved a number of cluster compute nodes):

```
module load GCC/10.2.0 OpenMPI/4.0.5
mpirun apptainer exec ubuntu-openmpi.sif /path/to/your/parallel-mpi-app
```

The above lines can be entered at the command line of an interactive session, or can also be inserted into a batch job script.

## Further Reading

- [Apptainer home page](#)
- [Apptainer Library](#)
- [Docker Hub](#)

2)

[https://lmod.readthedocs.io/en/latest/010\\_user.html](https://lmod.readthedocs.io/en/latest/010_user.html)

3)

<https://easybuild.readthedocs.io/en/latest/>

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/400\\_modules\\_and\\_application\\_software](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/400_modules_and_application_software)

Last update: **2025/03/24 13:36**



# Abaqus

---

Abaqus is a comprehensive Finite Element program system for solving complex linear and non-linear tasks in structural analysis, dynamics, heat conduction and acoustics with large geometry non-linearities and the possibilities of substructure technology. Abaqus is commercial software suite developed by [Dassault Systèmes Simulia Corp.](#).

There are various Abaqus software products accessible on the cluster: *Abaqus/Standard*, *Abaqus/Explicit*, *Abaqus/CAE*, *Abaqus/Viewer*.

For a complete list of Abaqus products, after loading the appropriate module, type `abaqus help`.

## Abaqus licensing

The use of Abaqus on the cluster system is strictly limited to teaching and academic research for non-industrially funded projects only.

Abaqus analysis or interactive application running on the cluster must contact the license server (provided by LUIS) at the beginning of the execution and periodically during the execution, i.e. Abaqus must have uninterrupted communication with the license server. A single CPU job from *Abaqus/Standard* or *Abaqus/Explicit* requires 5 so-called Analysis Tokens. For each additional CPU per job, an additional analysis token is required. Currently 720 Abaqus license-tokens can be totally utilized by cluster jobs. To display the actual status of the license usage, after loading the Abaqus module(see below), type:

```
abaqus licensing lmstat -a
```

## Usage on the cluster

You can list all available Abqaus versions by calling `module avail abaqus`. To load a particular software version, use `module load ABAQUS/<version>`. For example, to activate Abaqus version 2019, type

```
module load ABAQUS/2019
```

Abaqus contains a large number of example problems which can be used to become familiar with Abaqus on the system. These example problems are described in the Abaqus documentation and can be obtained using the Abaqus `fetch` command. For example, the following will extract the input file `s4d.inp` for the test problem **s4d**:

```
abaqus fetch job=s4d
```

## Abaqus GUI

The pre- and post-processor *Abaqus/CAE* or the post-processor *Abaqus/Viewer* can be used with a

graphical user interface. The Abaqus/CAE GUI can be launched by the command:

```
abaqus cae -mesa
```

Whereas the Abaqus/Viewer GUI can be started using:

```
abaqus viewer -mesa
```

## Abaqus batch usage

if your model makes use of a **user defined subroutine**, in order to run Abaqus the option `user=<your_subroutine>` has to be provided when calling Abaqus in all batch scripts.

Below is the example batch script `abaqus-serial.sh` for a **serial** (single CPU core) run:

### [abaqus-serial.sh](#)

```
#!/bin/bash -l
#SBATCH --job-name=abaqus_smp
#SBATCH --nodes=1
#SBATCH --ntasks-per-node=1
#SBATCH --mem=4G
#SBATCH --time=00:30:00
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=END

# Load modules
module load ABAQUS/2020
unset SLURM_GTIDS

# Change to work dir:
cd $SLURM_SUBMIT_DIR

# Run Abaqus
abaqus job=my_job input=<my_input_file.inp> scratch=$TMPDIR interactive
```

Submit the file `abaqus-serial.sh` to SLURM using the command: `sbatch abaqus-serial.sh`.

The keyword `interactive` in the script is required to tell Abaqus not to return until the simulation has completed. It is assumed that the input file `<my_input_file.inp>` is located in the job submit directory.

For very large Finite Element models (over 100,000 degrees of freedom), computing in parallel mode using several processors at the same time is often better suited to obtain the result of the analysis more quickly. In the ideal case, the wall-clock time is shortened proportionally to the number of processors involved.

The following is a sample batch script to run Abaqus using **thread**-based parallelization (multiple CPU cores on a single compute node):

## abaqus-parallel-smp.sh

```
#!/bin/bash -l
#SBATCH --job-name=abaqus_parallel_smp
#SBATCH --nodes=1
#SBATCH --cpus-per-task=20
#SBATCH --mem=60G
#SBATCH --time=00:30:00
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=END

# Load modules
module load ABAQUS/2020
unset SLURM_GTIDS

# Change to work dir:
cd $SLURM_SUBMIT_DIR

# Set Abaqus environment variables based on SLURM job
expand-slurm-nodelist --abaqus

# Run Abaqus
abaqus job=my_job input=<my_input_file.inp> scratch=$TMPDIR interactive
```

In this mode, the script `expand-slurm-nodelist --abaqus` sets the Abaqus variable `mp_mode=threads` and defines the variable `cpus` within the `abaqus_v6.env` file in the working directory. The file `abaqus_v6.env` is created or overwritten based on the SLURM job parameters, such as node allocation and CPU cores. This ensures Abaqus uses the correct settings for SMP (threading) execution on a single node.

Submit this script using:

```
sbatch abaqus-parallel-smp.sh
```

The following script runs Abaqus using **MPI** parallelization (multiple CPU cores across multiple compute nodes):

## abaqus-parallel-mpi.sh

```
#!/bin/bash -l
#SBATCH --job-name=abaqus_parallel_mpi
#SBATCH --nodes=2
#SBATCH --cpus-per-task=20
#SBATCH --mem=120G
#SBATCH --time=00:30:00
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=END

# Load modules
module load ABAQUS/2020
```

```
unset SLURM_GTIDS

# Change to work dir:
cd $SLURM_SUBMIT_DIR

# Set Abaqus environment variables based on SLURM job
expand-slurm-nodelist --abaqus

# Run Abaqus
abaqus job=my_job input=<my_input_file.inp> scratch=$TMPDIR interactive
```

For MPI runs, the script `expand-slurm-nodelist --abaqus` sets `mp_mode=mpi` and defines the `mp_host_list` and `cpus` variables in the file `abaqus_v6.env`. This file is created or overwritten in the working directory based on your SLURM job settings (compute nodes and CPU cores allocated).

Submit this script using:

```
sbatch abaqus-parallel-mpi.sh
```

If you want to override the Abaqus environment parameters `mp_mode`, `mp_host_list` or `cpus` for a particular run, you must provide them explicitly on the Abaqus command line. Command-line arguments take precedence over values in `abaqus_v6.env`. Other Abaqus parameters can be set either in the `abaqus_v6.env` file or on the Abaqus command line, depending on your needs.

**Performance tipp:** try to fill one node before requesting more than one - inter-node communication (between nodes) is almost always slower than intra-node communication (within a node). So try to stay on one node and only grow if you need to. If your job can use all cores on one node, request them, and remember to also request *almost* all memory when you fill a node (if you only use half the cores, of course, you should also request less than half of the node's memory). Rule of thumb: leave about 2 GB free for the Linux kernel and system buffers. Have a look in the table of [Hardware specifications of cluster compute nodes](#) to find a partition that suits your needs, and test which setup gives you the best performance.

In case you get a `LookupError: unknown encoding: ISO-8859-1`, use `export LANG=en_US.utf8`. Abaqus does not seem to like system other languages.

From:  
<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:  
[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_abaqus](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_abaqus)

Last update: **2025/06/01 21:40**



# ANSYS / CFX

## ANSYS Workbench

ANSYS workbench can be started with the following command.

```
runwb2
```

## ANSYS Mechanical APDL

ANSYS Mechanical APDL can be started with the following command (replace the number in the binary name `ansys231` with the appropriate version you use; this example is done after a `module load ANSYS/2023.1`):

```
ansys231
```

Likewise an interactive session in graphics mode can be started with the following command.

```
ansys231 -g
```

Starting Ansys on one node (shared memory) from a job script:

```
..  
#SBATCH --cpus-per-task=12  
..  
export ANSWAIT=1  
ansys221 -b -np $SLURM_CPUS_PER_TASK -i test.dat -o test.out
```

To use `cfx5solve`:

```
cfx5solve -batch -def mytest.def -par-dist $nodes -start-method "Open MPI  
Local Parallel"
```

Starting Ansys on multiple nodes (distributed memory) from a job script; Attention: fill up complete nodes before you start using multiple nodes. Communication between nodes takes much longer than intra-node, so try to stay on one node as long as you can.

```
..  
#SBATCH --nodes=2  
#SBATCH --ntasks-per-node=32  
..  
export ANSWAIT=1  
ansys221 -b -dis -np $SLURM_NTASKS -machines $(expand-slurm-nodelist -m) -i  
test.dat -o test.out
```

Submit an interactive `cfx5solve` job to multiple nodes (distributed parallel); again, try to stay on one node as long as you fit onto one machine, only use multiple nodes when resources needed are too

large:

```
salloc --nodes=2 --ntasks-per-node=32 --mem-per-cpu=3G --time=6:0:0
```

As soon as the nodes have been allocated:

```
module load ANSYS/2021.2
nodes=$(expand-slurm-nodelist --cfx)
cfx5solve -batch -def mytest.def -par-dist $nodes -start-method "Open MPI
Distributed Parallel"
```

## ANSYS Tips

### Problems

In case you get a message “Unexpected error: The following required addins could not be loaded: Ans.SceneGraphChart.SceneGraphAddin. The software will exit.”, try `module load foss/2021b Mesa/.21.1.7` first. The problem seems to be specific to Ansys/2025.1, which seems to have incompatibilities with the default OpenGL environment. For X2GO, a workaround (loading module “StdEnv”, which is only available on the login nodes and which basically just loads the hidden module Mesa/.21.1.7 that seems to work with Ansys/2025.1) is already in place. In the web portal (“OOD”, OpenOnDemand, the thing you log into by pointing your browser to <https://login.cluster.uni-hannover.de>), one needs to work around manually loading Mesa/.21.1.7, a prerequisite of which is e.g. the above mentioned foss/2021b module (or the specific GCC version loaded by this version, to be more precise). Check for the module using `module --show-hidden spider Mesa` in case you are interested.

### Memory usage

Many errors are due to jobs not being configured properly, in particular requesting not enough memory. The system checks the memory request you made at several instances and from time to time, and in case you overstep what you requested, your job will/may get killed (it may, at times, make it through, however, giving rise to questions like “but it ran without problems up to now” - which is not true, but you just did not yet see the problems and the system did not yet kill your jobs).

So please try to adapt your job to your requirements. To find out how the nodes are configured, see our [table of computing hardware](#). Try to use about the same fraction of memory as the number of cpu cores on a node to facilitate matching those hardware components - our computers internally usually consist of several so-called NUMA-nodes, which means that each cpu-socket also has RAM that is directly attached to it, and this usually is the fraction of memory that corresponds to the fraction of cpu cores the socket contains. Use slightly less than the maximum amount of memory in that fraction to leave room for the operating system (Linux) and some buffers so your job doesn't have to be squeezed. 4 GB should be enough, 8 GB may see some improvements, and your exact mileage may vary.

## Node usage

If you can, you should try to stay on one node instead of requesting fractions of several nodes. Inter-node communication usually takes much longer than intra-node communication, so you may benefit from filling up nodes first and only expanding to other nodes when the job gets too big to fit on one node. We see, of course, that one may fit in faster by requesting only fractions of nodes, but that may not deliver the best overall performance. And if you occupy only parts of nodes, you'll also make it more difficult for others to get full nodes.

## Enos partition equipped with OmniPath instead of Infiniband

You might come across an error like this when running ANSYS on enos nodes.

```
ansysdis201: Rank 0:8: MPI_Init_thread: multiple pkey found in partition key table, please choose one via MPI_IB_PKEY
```

Enos nodes of the cluster system do not have an InfiniBand interconnect but use Omni-Path instead. If you would like to run ANSYS on enos nodes, chose the correct partition key (pkey) by adding the following line to your job script before calling the ANSYS application.

```
[[ $HOSTNAME =~ ^enos-.* ]] && export MPI_IB_PKEY=0x8001
```

However, sometimes ANSYS, or the underlying mpi-implementation used, does not seem to honour the exported variable causing the error to persist. Unfortunately the ANSYS documentation on their mpi implementations is scarce. In this case please contact ANSYS support or exclude enos nodes from your job in order to circumvent the error. You can write a PARTITION line in your resource specification with all partitions you would like to use except enos. There is no option to exclude enos or any other partition.

## Debugging

To see what ansys does when it runs, set export `ANS_SEE_RUN_COMMAND=1`

From:  
<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:  
[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_ansys](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_ansys)

Last update: **2025/07/21 16:53**





# COMSOL

---

**COMSOL Multiphysics** is a finite element analysis, solver and simulation software/FEA software package for various physics and engineering applications, especially for coupled phenomena, and multiphysics. In addition to conventional physics-based user interfaces, COMSOL Multiphysics also allows entering coupled systems of partial differential equations (PDEs)

## Prerequisite for use on the cluster

In order to use COMSOL on the cluster system, you need a valid license, which usually means that your home institute permits you to use theirs. Institutes buy licenses primarily for local use on their own workstations and you cannot use them by default on the cluster. That means if you want to use your institute's licenses on the cluster system, you will need to contact us beforehand, because the corresponding user names need to be added to the unix group `comsol` on the cluster system before they are allowed to use COMSOL here.

A license token can either be used only locally, or additionally on the cluster system. As long as it is in use, regardless of where that may be, it can not be used elsewhere. Please be aware that licenses available on the cluster system to the "comsol" unix group are available to everyone in that group. This includes multiple institutes. I.e. if you decide to make your license available for use on the cluster, it will not be exclusive to you or your institute. Usually, though, this does not pose a problem, since normally not all license tokens are used at the exact same times.

Please contact our colleagues from the license management department in order to purchase licenses<sup>5)</sup>.

## Using COMSOL on the cluster

COMSOL can run either in graphical/interactive (GUI) mode or in batch mode.

To start the graphical user interface in an interactive batch job, you could use:

```
salloc --job-name=myjob --nodes=1 --ntasks=8 --time=2:00:00 --mem=30G --x11
```

After the system tells you your batch job is ready, load the corresponding module:

```
module load COMSOL
```

The following command lists all available versions:

```
module avail comsol
```

Then start the COMSOL GUI by typing

```
comsol -np $SLURM_NTASKS -3drend sw
```

Please avoid starting the program on login nodes so you do not steal resources from other users – your jobs would be killed here after using 30 minutes cpu time, anyway. Either use an interactive job or use “Interactive Apps” in the OnDemand web portal. Both methods are described in detail in this handbook.

In order to use COMSOL in batch mode, you will need a batch job script.

**Attention:** COMSOL is a rather memory intensive application. If you do not explicitly request memory, you will get a default value, which may be somewhere between 1600 MB and 4 GB for each cpu core requested, depending on the node you land on. So better make this clear by requesting what you need. Have a look in our available resources table that describes the various types of nodes we run in the cluster.

Here is a sample script to run a COMSOL job on one node:

[comsol\\_serial.sh](#)

```
#!/bin/bash -login
#SBATCH --job-name COMSOL
#SBATCH --mail-user=youremail@...uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --time=1:00:00
#SBATCH --nodes=1
#SBATCH --ntasks=8
#SBATCH --mem=30G

# load module
module load COMSOL/5.4

# go to work dir
cd $SLURM_SUBMIT_DIR

# start comsol
comsol batch -np $SLURM_NTASKS -inputfile infile.mph -outputfile
outfile.mph -tmpdir $TMPDIR
```

Another sample script starting an MPI job on two nodes:

[comsol\\_parallel.sh](#)

```
#!/bin/bash -login
#SBATCH --job-name COMSOL
#SBATCH --mail-user=youremail@...uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --time=1:00:00
#SBATCH --nodes=2
#SBATCH --ntasks=16
#SBATCH --mem=60G

# load module
```

```

module load COMSOL/5.4

# go to work dir
cd $SLURM_SUBMIT_DIR

echo "Running on $SLURM_JOB_NODELIST"
/sw-eb/system/bin/expand-slurm-nodelist >hostfile
# start comsol (all options must be on a single line)
comsol -nn $SLURM_NNODES -np $SLURM_NPROCS batch -f hostfile -mpirsh
ssh -inputfile input_cluster.mph -outputfile output_cluster.mph -tmpdir
$TMPDIR

```

In case you want to run parallel COMSOL jobs, we strongly recommend to first do a scaling study.

We do not provide a central COMSOL server, but of course you may start one yourself from within the job.

**WARNING:** COMSOL can produce BIG recovery files that by default will be placed in your home directory. This in turn may be a reason to unexpectedly exceed your home quota (the maximum amount of space you may occupy in \$HOME). You will not be able to login graphically if you exceed your home quota, so avoid this by either moving the recovery directory to \$BIGWORK using the option `-recoverydir $BIGWORK/comsol`, or by disabling recovery files altogether using the option `-autosave off`.

If you work with the graphical interface, you should go to “Options → Preferences → Files” and set all directories to a subdirectory of your \$BIGWORK directory, see [figure 1](#)

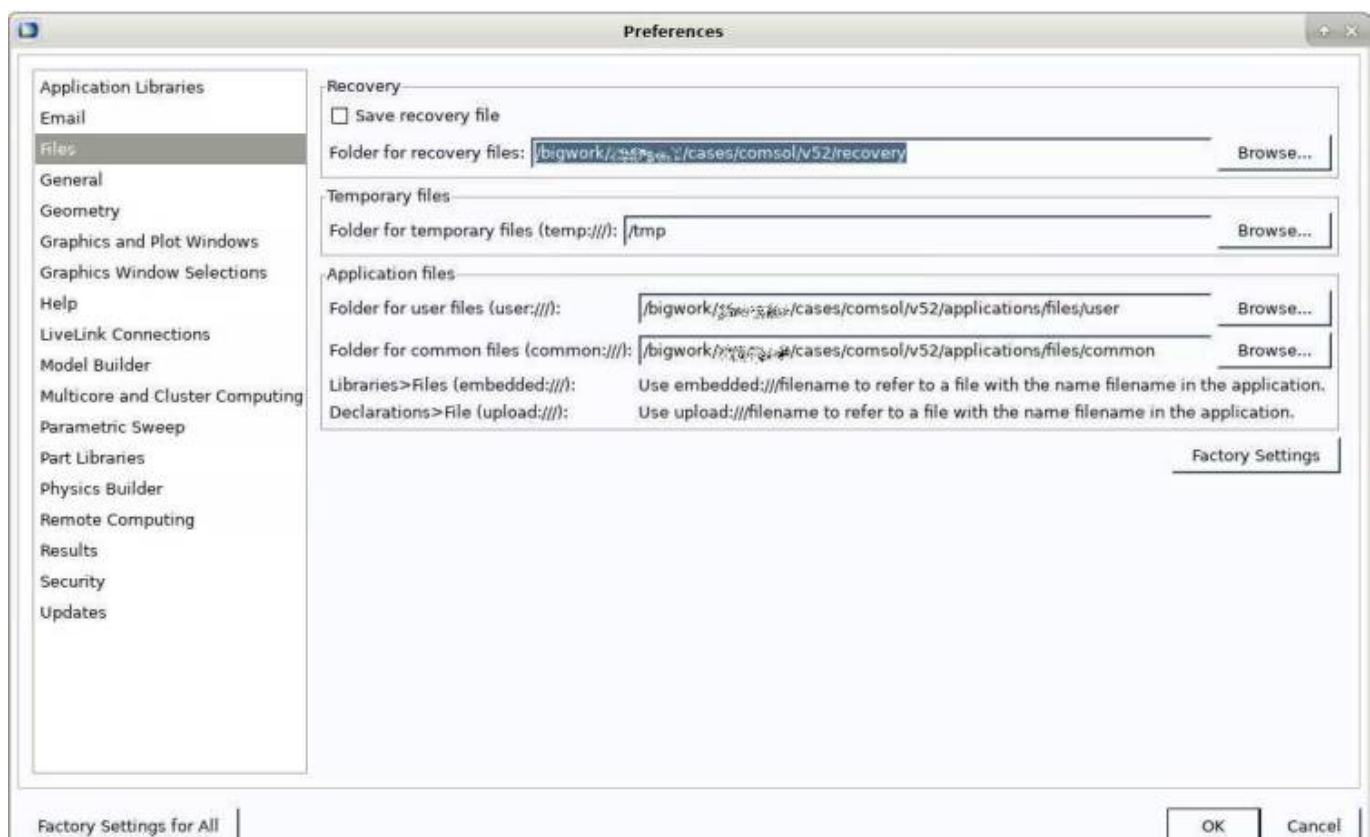


Fig. 1: Comsol GUI

5)

<https://www.luis.uni-hannover.de/software.html>

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_comsol](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_comsol)

Last update: **2022/08/03 08:09**



# CPMD

---

The CPMD code is a parallelized plane wave/pseudopotential implementation of Density Functional Theory, particularly designed for ab-initio molecular dynamics.

## Prerequisites to use CPMD on the cluster system

In order to use CPMD on the cluster system, you need a valid license, which is usually granted free of charge to members of academic institutions for non-profit, non-transferable personal usage after an application on their [web site](#).

After you receive your license, please let us know ([cluster-help@luis.uni-hannover.de](mailto:cluster-help@luis.uni-hannover.de), please also include your user account name). We will then ask the CPMD consortium to validate your license and thereafter add you to the unix group `cpmd` that has access to the software.

## Using CPMD on the cluster

To list all installed CPMD versions, run `module spider cpmd`.

If you want to access CPMD version 4.3, you have to load all modules recommended by the command `module spider CPMD/4.3`:

```
module load GCC/8.3.0 OpenMPI/3.1.4 CPMD/4.3
```

In a sample job-script below you will need to set an input file and provide the location to [pseudo-potential](#) that are available for download from CPMD website after authentication.

# SLURM script

[cpmd-job.sh](#)

```
#!/bin/bash -l
#SBATCH --job-name=cpmd_job
#SBATCH --nodes=2
#SBATCH --ntasks-per-node=20
#SBATCH --mem=120G
#SBATCH --time=1:00:00
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=END

# Compute node the job ran on
echo "Job ran on:" $HOSTNAME

# Load modules
```

```
module load GCC/8.3.0 OpenMPI/3.1.4 CPMD/4.3

# Change to work dir
cd $SLURM_SUBMIT_DIR

CPMD_INPUT=<your input file>
CPMD_OUTPUT=cpmd-job-`${SLURM_JOB_ID}`.log
PP_PATH=<path to the location of pseudo-potentials>

srun cpmd.x `${CPMD_INPUT}` `${PP_PATH}` > `${CPMD_OUTPUT}`
```

## Further Reading

- CPMD [home page](#)
- CPMD [manual](#)
- CPMD [pseudo-potentials](#)

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_cpmd](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_cpmd)

Last update: **2022/12/13 15:46**



# FEKO

Under SLURM, you may experience very bad parallel performance. When looking at a running job, this may show in the single processes using only a small part of each cpu core, and this increases with the number of cpu cores used. In this case, you should disable the cpu binding that is set by SLURM using the following line before starting your job:

```
export SLURM_CPU_BIND=none
```

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_feko](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_feko)

Last update: **2022/04/13 10:10**



# Jupyter in the cluster

---

“Project Jupyter exists to develop open-source software, open-standards, and services for interactive computing across dozens of programming languages.” <sup>6)</sup>

“The [Jupyter notebook](#) extends the console-based approach to interactive computing in a qualitatively new direction, providing a web-based application suitable for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results.” <sup>7)</sup>

“[JupyterLab](#) is a web-based interactive development environment for Jupyter notebooks, code, and data.” <sup>8)</sup>

Detailed information about various components of Project Jupyter can be accessed [here](#).

In this section we explain Jupyter usage in the cluster.

## How to start Jupyter in the cluster

On the cluster, Jupyter Notebook and JupyterLab are available from within the cluster web portal as OOD interactive application, which starts Jupyter sessions on a compute node by means of a SLURM job. To access the Jupyter app, login onto the portal and in the *Interactive Apps* menu select the *Jupyter* server. This will open a web dialog page which allows you to start a Jupyter session in a SLURM job using the standard cluster-wide Conda and Module environments you can select from the drop-down *Standard environment* list. On this page, you can also specify the job parameters such as time limit, number of CPU cores, amount of allocated memory, number of GPU units and the cluster partition you want to start your Jupyter session on. Note that OOD Jupyter applications currently launch Jupyter sessions on a **single** compute node.



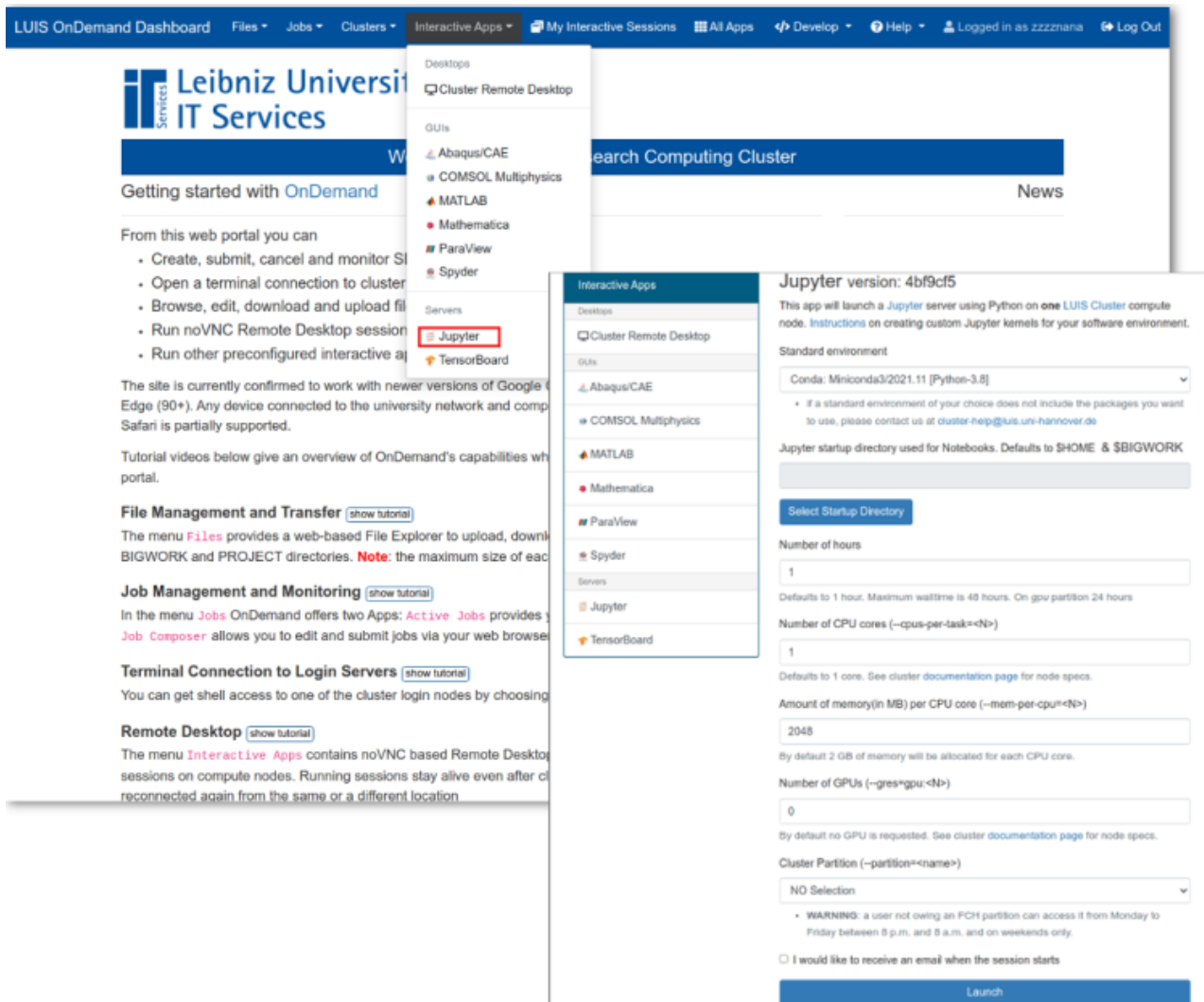


Fig. 1: Jupyter on the cluster

By selecting the standard Jupyter environment in addition to the IPython kernel providing many popular Python packages for Data Science, Machine Learning and Scientific Computing (Tensorflow, PyTorch, Pandas, NumPy, SciPy, Matplotlib, etc) you can also run the Jupyter kernels for R and MATLAB. If a standard environment of your choice does not contain packages you want to use or you would like more kernels to be included, please get in touch with the cluster group at [cluster-help@luis.uni-hannover.de](mailto:cluster-help@luis.uni-hannover.de). You can also make the packages you need available in your JupyterLab session via a custom Jupyter kernel, see next subsection below. Check out the list of available [Jupyter kernels](#).

## Creating custom Jupyter kernels

If the standard JupyterLab environments do not contain packages you require for your project, you need a different python version or other programming languages you may provide them by creating JupyterLab kernels. In this section we explain how to install a custom python kernel for your Python virtualenv or Conda environments and how to switch between environments without resubmitting JupyterLab session SLURM job.

The installation of a new kernel is done in two steps:

1. Create an environment (Conda or Python virtualenv) and install required Python libraries and

packages including `ipykernel`

2. Install a custom Python kernel for JupyterLab. The kernel will be located in a sub-directory of `$HOME/.local/share/jupyter/kernels`

**Please note:** since the installation of Python packages requires an access to internet, the configuration of a kernel must be done on a login node.

## Conda environment

To make your conda environment available in a JupyterLab session, follow the instructions below.

For details about creating conda environments, see [the conda usage](#) in the cluster.

1. Create a conda environment (skip this step if the environment already exists)

```
[username@login01 ~]$ module load Miniforge3  
[username@login01 ~]$ conda create -n my_env
```

2. Activate the environment, install the `ipykernel` library and create a Python kernel for JupyterLab:

```
[username@login01 ~]$ conda activate my_env  
(my_env)[username@login01 ~]$ conda install ipykernel  
(my_env)[username@login01 ~]$ make-ipykernel --name my_conda_env --display-name "My Software Env"
```

The configuration file, `kernel.json`, of the `my_conda_env` kernel will be created in the directory `$HOME/.local/share/jupyter/kernels/my_conda_env`.

**Note:** The `--name` option must uniquely identify the kernel.

3. Install additional packages you need:

```
(my_env)[username@login01 ~]$ conda install numpy matplotlib sympy
```

## Python virtualenv

1. Select your preferred Python version by loading the appropriate module:

```
[username@login01 ~]$ module load GCC/10.2.0 Python/3.8.6
```

2. Create a Python virtual environment named `myvenv` at the specified location (skip this step if the environment already exists):

```
[username@login01 ~]$ virtualenv $HOME/myvenv
```

3. Activate the environment, install the `ipykernel` library and create a Python kernel for JupyterLab:

```
[username@login01 ~]$ source $HOME/myenv/bin/activate
(myenv)[username@login01 ~]$ pip install ipykernel
(myenv)[username@login01 ~]$ make-ipykernel --name my_venv --display-name
"My Software Env"
```

The configuration file, `kernel.json`, of the `my_venv` kernel will be created in the directory `$HOME/.local/share/jupyter/kernels/my_venv`.

**Note:** The `--name` option must uniquely identify the kernel.

#### 4. Install other Python packages

```
(myenv)[username@login01 ~]$ pip install numpy sympy
```

A few seconds after creating the kernel, your software environment will appear in your active JupyterLab session listed as `My Software Env` on the launcher page. To make the kernel immediately available in JupyterLab, just refresh the browser window.

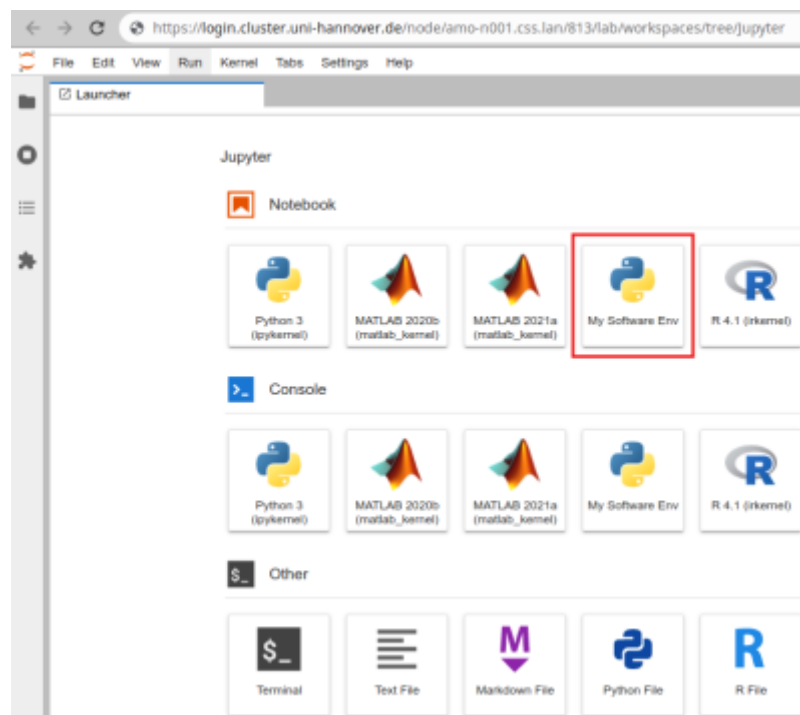


Fig. 2: Python Jupyter kernel

You can switch kernels using the menu `Kernel → Change Kernel...` The kernel `Python3 (ipykernel)` corresponds to the standard cluster-wide environment you selected when submitting your JupyterLab session.

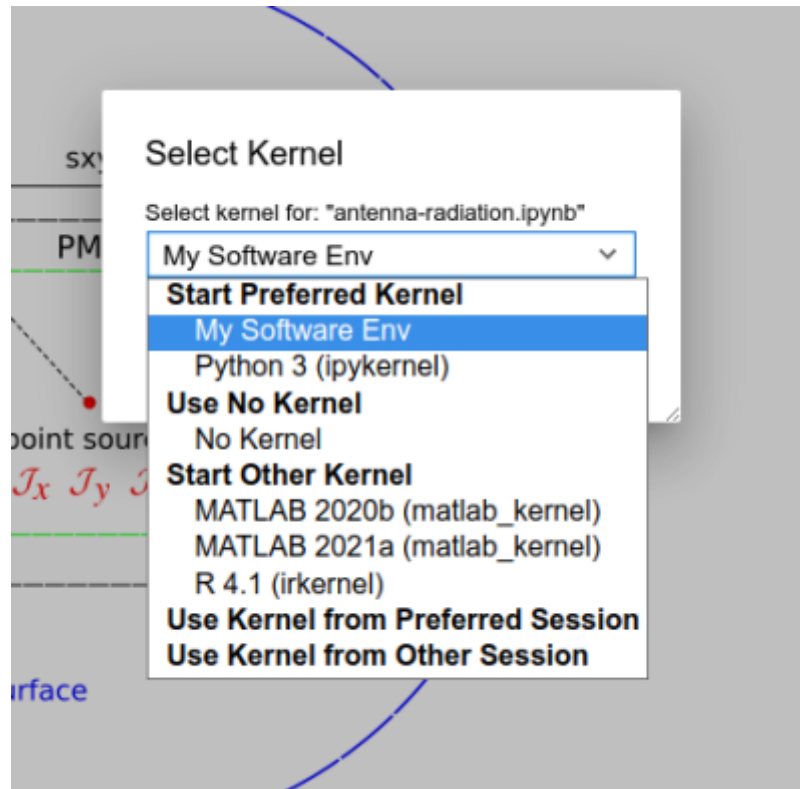


Fig. 3: Change Kernel

**Classic Jupyter notebook:** Your kernel is listed in the menu *New*:

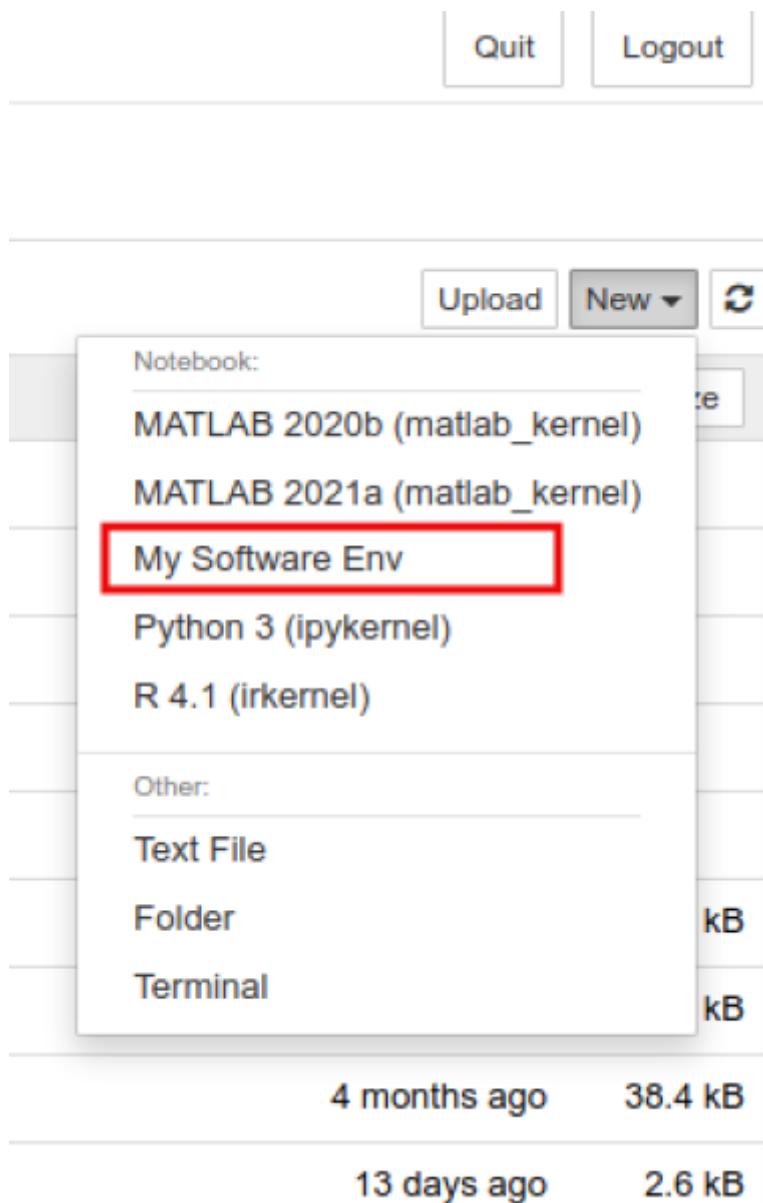
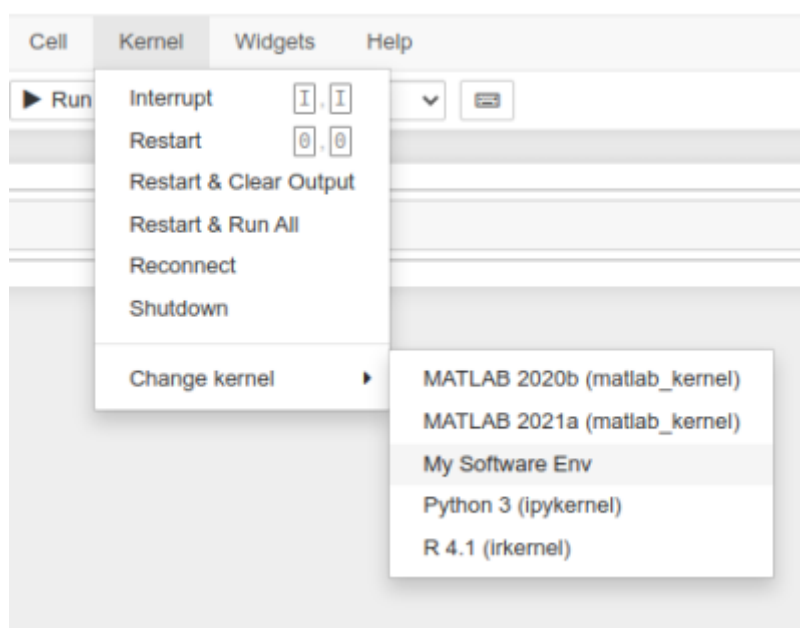


Fig. 4: Python Jupyter kernel

**Classic Jupyter notebook:** To switch kernels visit the menu *Kernel*:



## Fig. 5: Switch kernels

<sup>6)</sup>

[Project Jupyter webpage](#)

<sup>7)</sup>

[Jupyter Notebook documentation](#)

<sup>8)</sup>

[JupyterLab documentation](#)

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_jupyterlab](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_jupyterlab)

Last update: **2024/10/25 07:35**



# MATLAB

---

[MATLAB](#) is a technical computing environment for high performance numeric computation and visualization. MATLAB integrates numerical analysis, matrix computation, signal processing and graphics. MATLAB toolboxes are collections of algorithms that enhance MATLAB's functionality in domains such as signal and image processing, data analysis and statistics, mathematical modeling, etc.

## Versions and Availability

You can use the command `module spider matlab` to get a list of available versions. Feel free to contact cluster team if you need other versions. MATLAB is proprietary software and can be used on the cluster for non-commercial, academic purposes only. As a free alternative to MATLAB you might consider GNU Octave, which is mostly compatible with MATLAB and provides most of the features as well.

## Running MATLAB

As MATLAB by default utilizes all available CPU cores, excessive use of MATLAB on the login nodes would prevent other logged in users from using resources. The recommended way to work with MATLAB is to submit a job (interactive or batch) and start MATLAB from within the dedicated compute node assigned to you by the batch system.

To submit an interactive job, you can use the command (you may adjust the numbers per your need).

```
login03~$ salloc --nodes=1 --ntasks=12 --mem-per-cpu=2G --time=02:00:00 --x11
```

The `--x11` flag enables X11 forwarding on the compute node if you need to use the graphical MATLAB user interface.

Once you are assigned a compute node, you can run MATLAB interactively by loading the MATLAB module. To load the default version of MATLAB module, use `module load MATLAB`. To select a particular MATLAB version, use `module load MATLAB/version`. For example, to load MATLAB version 2017a, use `module load MATLAB/2017a`.

To start MATLAB interactively with graphical user interface (GUI), after having loaded MATLAB module, type the command.

```
smp-n010~$ matlab
```

This requires the `--x11` flag for the `salloc` command as well as X11 forwarding enabled for your SSH client or using [X2GO](#) for logging in onto the cluster system. MATLAB GUI can also be launched from a web browser via [Remote Desktop Session](#). The following command will start an interactive, non-GUI version of MATLAB.

```
smp-n010~$ matlab -nodisplay -nosplash
```

Type `matlab -h` for a complete list of command line options.

In order to run MATLAB non-interactively via the batch system, you will require a batch submission script. Below is an example batch script (`matlab-job-serial.sh`) for a serial run that will execute MATLAB script (`hello.m`) in a single thread.

#### `matlab-job-serial.sh`

```
#!/bin/bash -l
#SBATCH --job-name=matlab_serial
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --mem-per-cpu=4G
#SBATCH --time=00:20:00
#SBATCH --output matlab_serial-job_%j.out

# Compute node the job ran on
echo "Job ran on:" $HOSTNAME

# Load modules
module load MATLAB/2017a

# Change to work dir:
cd $SLURM_SUBMIT_DIR

# Log file name
LOGFILE=$(echo $SLURM_JOB_ID | cut -d"." -f1).log

# The program to run
matlab -nodesktop -nosplash < hello.m > $LOGFILE 2>&1
```

Example MATLAB script, `hello.m`:

#### `hello.m`

```
% Example MATLAB script for Hello World
disp 'Hello World'
exit
% end of example file
```

To run `hello.m` via the batch system, submit the `matlab-job-serial.sh` file with the following command:

```
login03~$ sbatch matlab-job-serial.sh
```



## Submitted batch job 736583

Output from the running of the MATLAB script will be saved in the \$LOGFILE file, which in this case expands to matlab\_serial-job\_736583.out.

### Parallel Processing in MATLAB

**Please note:** MATLAB parallel computing across multiple compute nodes is not supported by the cluster system at the moment.

MATLAB supports both implicit multithreading as well as explicit parallelism provided by the Parallel Computing Toolbox (PCT) which requires specific commands in your MATLAB code in order to create threads.

Implicit multithreading allows some functions in MATLAB, particularly linear algebra and numerical routines such as `fft`, `eig`, `svd`, etc, to distribute the workload between cores of the node that your job is running on and thus run faster than on a single core. By default, all of the current versions of MATLAB available on the cluster have multithreading enabled. **A single MATLAB session will run as many threads as there are cores on a compute node reserved for your job by the batch system.** For example, if you request `--ntasks=4`, your MATLAB session will spawn four threads.

[matlab-job-multithread.sh](#)

```
#!/bin/bash -l
#SBATCH --job-name=matlab_multithread
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --nodes=1
#SBATCH --ntasks=4
#SBATCH --mem-per-cpu=4G
#SBATCH --time=00:20:00
#SBATCH --output matlab_multithread-job_%j.out

# Compute node the job ran on
echo "Job ran on:" $HOSTNAME

# Load modules
module load MATLAB/2017a

# Change to work dir:
cd $SLURM_SUBMIT_DIR

# Log file name
LOGFILE=$(echo $SLURM_JOB_ID | cut -d"." -f1).log

# The program to run
matlab -nodesktop -nosplash < hello.m > $LOGFILE 2>&1
```

However, if you want to disable multithreading you may either request `--ntasks=1` (see an example

job script above) or add the option `-singleCompThread` when running MATLAB.

## Using the Parallel Computing Toolbox

MATLAB Parallel Computing Toolbox (PCT) (parallel for-loops, special array types, etc.) lets you make explicit use of multicore processors, GPUs and clusters by executing applications on workers (MATLAB computational engines) that run locally. At the moment, the cluster system does not support parallel processing across multiple nodes (MATLAB Distributed Computing Server). As such, parallel MATLAB jobs are limited to a single compute node with the “local” pool through use of the MATLAB PCT.

Specific care must be taken when running multiple MATLAB PCT jobs. When you submit multiple jobs that are all using MATLAB PCT for parallelization, all of the jobs will attempt to use the same default location for storing information about the MATLAB pools that are in use, thereby creating a race condition where one job modifies the files that were put in place by another. The solution is to have each of your jobs that will use the PCT set a unique location for storing job information. An example batch script `matlab-job-pct.sh` is shown below.

### [matlab-job-pct.sh](#)

```
#!/bin/bash -l
#SBATCH --job-name=matlab_multithread_pct
#SBATCH --mail-user=user@uni-hannover.de
#SBATCH --mail-type=BEGIN,END,FAIL
#SBATCH --nodes=1
#SBATCH --ntasks=12
#SBATCH --mem-per-cpu=3G
#SBATCH --time=00:40:00
#SBATCH --output matlab_multithread-job_%j.out

# Compute node the job ran on
echo "Job ran on:" $HOSTNAME

# Load modules
module load MATLAB/2017a

# Change to work dir:
cd $SLURM_SUBMIT_DIR

# Log file name
LOGFILE=$(echo $SLURM_JOB_ID | cut -d"." -f1).log

# The program to run
matlab -nodesktop -nosplash < pi_parallel.m > $LOGFILE 2>&1
```

And the corresponding MATLAB script `pi_parallel.m`, which in addition starts the correct number of parallel MATLAB workers depending on the requested cores.

### [pi\\_parallel.m](#)

```

% create a local cluster object
pc = parcluster('local')

% explicitly set the JobStorageLocation to
% the temp directory that is unique to each cluster job
pc.JobStorageLocation = getenv('TMPDIR')

% start the matlabpool with maximum available workers
parpool (pc, str2num(getenv('SLURM_CPUS_ON_NODE'))))

R = 1
darts = 1e7
count = 0
tic
parfor i = 1:darts
    x = R*rand(1)
    y = R*rand(1)
    if x^2 + y^2 <= R^2
        count = count + 1
    end
end
myPI = 4*count/darts
T = toc
fprintf('The computed value of pi is %8.7f\n',myPI)
fprintf('The parallel Monte-Carlo method is executed in %8.2f\n', T)
delete(gcf)
exit

```

For further details on MATLAB PCT refer to the online [documentation](#).

## Build MEX File

See e.g. [online documentation](#) for details on how to build mex files. This section is a straight transcript of that website adapted to fit the cluster system's module system. First, get hold of the example file `timestwo.c` which comes with MATLAB. This can be done from within MATLAB.

```

copyfile(fullfile(matlabroot,'extern','examples','refbook','timestwo.c'),'.' , 'f')

```

Each MATLAB version needs a specific version of GCC in order to build mex files. This is the crucial part. See MATLAB documentation for details.

```

$ module load MATLAB/2018a
$ module load GCC/6.3.0-2.27
$ ls

```

```
timestwo.c
$ mex timestwo.c
Building with 'gcc'.
MEX completed successfully.
$ ls
timestwo.c timestwo.mexa64
```

Notice the file `timestwo.mexa64` was created. You can now use it like a function.

```
$ matlab -nodesktop -nosplash>> timestwo(6)

ans =

    12
```

## Toolboxes and Features

On the cluster system you can use the university's MATLAB campus licence. Please see [for details and a list of available toolboxes and features](#).

## Further Reading

- MATLAB [online documentation](#)
- MATLAB [Parallel Computing Toolbox](#)

From:  
<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:  
[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_matlab](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_matlab)

Last update: **2022/10/11 14:00**



# Conda



**Important Notice:** We ask you to avoid using the Miniconda3 modules, as the defaults channel appears to be no longer freely available under Anaconda's updated licensing terms. Please switch to the Miniforge3 module, which provides access to free and open-source channels like conda-forge. Please update your environments accordingly to ensure unrestricted access to packages. Additionally, please check your ~/.condarc file and remove the defaults channel if it is still in use. You can also do this by running: `conda config --env --remove channels defaults`

Here is the Page documenting the use of Miniforge3: [Miniforge3 Usage](#)

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_miniconda3](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_miniconda3)

Last update: **2024/11/04 11:48**



# Conda



**Important Notice:** We ask you to avoid using the Miniconda3 modules, as the defaults channel appears to be no longer freely available under Anaconda's updated licensing terms. Please switch to the Miniforge3 module, which provides access to free and open-source channels like conda-forge. Please update your environments accordingly to ensure unrestricted access to packages. Additionally, please check your `~/.condarc` file and remove the defaults channel if it is still in use. You can also do this by running: `conda config --env --remove channels defaults`

Conda is a package management system which was initially created for Python, but currently also supports several other languages such as R or lua. Conda can be used to quickly find, install and update packages and their dependencies using community-maintained remote repositories (also called channels). Software packages and scientific libraries are installed in "environments" to provide the ability to maintain different, often incompatible, sets of software. Environments are also managed by Conda.

**Please note:** Compared to the software modules that we provide on the cluster, there are much more and newer libraries available via Conda that you can manage yourself, but they may not be as well optimized for the processor architectures on the cluster. In addition, by default, Conda installs packages in a user's home directory (`$HOME`), which has a quota for the size and number of files. You should take care to re-set the Conda installation directory to a subdirectory of your group's `$SOFTWARE` directory. Otherwise, you'll probably quite quickly run into your quota in `$HOME`, with inconvenient consequences (like not being able to use graphical logins any more before you remove the extraneous files, possibly erroneously also deleting files that should remain).

**Please note:** It is not recommended to mix cluster software modules and Conda-managed software in the same work environment.

**Please note:** Pre-configured conda environments are available on the cluster, providing a range of packages commonly used across scientific fields. Please refer to the [Pre-Configured Conda Environments](#) section for details on accessing and using these resources.

In this section we explain how to use Conda in the cluster.

## Conda usage on the cluster

On the cluster, Conda is available through the [Miniforge](#) installation, which provides default access to the free conda-forge channel. This installation includes only the Conda package manager, Python, and a minimal set of additional packages. The Miniforge installer also includes the *mamba* library, which offers fast package dependency resolution and speeds up package downloads by using parallel processing through multi-threading. For more details, see below.

## Creating a Conda environment

In order to create and use Conda environments on the cluster, you first need to load the Miniforge3 module:

```
module load Miniforge3
```

**Please note:** Since loading the Miniforge module will also initialize conda for shell interaction, you do not need to additionally run the `conda init <shell>` command modifying your shell init file, e.g. `$HOME/.bashrc`. In fact, you should avoid doing so, as it may cause issues with your interactive or batch jobs. If you ran the command in the past, you may have conda entries in your shell init file. Please edit the file and remove the lines between `# >>> conda initialize >>>` and `# <<< conda initialize <<<`.

If you use Conda-installed packages, the Miniforge3 module should be the only module you load in your work environment or in your job.

The following will create a conda environment named `myenv` and install the packages `python` and `numpy` into it:

```
conda create -n myenv python numpy
```

Note that if you have loaded the `Miniforge3/23.11.0-0` module, Python 3.10.x will be installed, unless you explicitly specify the python version above. For example, to create a Python 3.8 environment, use the following command:

```
conda create -n myenvpy38 python=3.8
```

Your conda environments by default are located in your home directory under the `$HOME/.conda/envs` path. We recommend to change this location. Either use the `--prefix` flag specifying the path to the directory you want (probably `$SOFTWARE/<nameofmycondadir>`), or assign the path to the environment variable `$CONDA_ENVS_PATH`. Another way is to edit your `$HOME/.condarc` file and define your conda environment locations using the key `envs_dirs`:

Packages that can be installed using conda are provided in channels. Popular channels are [Conda Forge](#) and [Bioconda](#), which are set by default on the cluster. If you want to install packages from a channel which is not among the default channels, you can specify it using the `--channel <your-channel>` (or shortly `-n <your-channel>`) flag during package installation. Alternatively, you can add it permanently to your `$HOME/.condarc` file under the `channels` key. This can also be done by running the command: `conda config --add channels <your-channel>`.

The default conda settings are defined in the cluster-wide configuration file `$EBROOTMINIFORGE3/.condarc`

```
auto_activate_base: false
```

```
envs_dirs:
```

```
- $HOME/.conda/envs
```

```
pkgs_dirs:
```

```
- $HOME/.conda/pkg
```

channels:

- conda-forge
- bioconda

The file also defines the default location of conda's package cache directory (key `pkgs_dirs:`). The `$CONDA_PKGS_DIRS` environment variable overwrites the `pkgs_dirs:` setting. The command `conda config --show-sources` displays all identified configuration sources.

If you want to remove packages that are not used in any environment, run:

```
conda clean --all
```

Execute `conda info` to see your current conda settings, including the default channel URLs and the location of your conda environments.

The command lists all your environments:

```
conda info --envs
```

The active conda environment is marked with an asterisk (\*).

To search for a package in configured channels, use the command:

```
conda search <package-name>
```

If you want to install additional packages in an existing environment, e.g. in `myenv`, it must first be activated:

```
conda activate myenv
```

Once a conda environment is activated, which may be recognized by the presence of the environment name (`myenv`)\$ on the command prompt, you can install additional software using either Conda or Pip. For example, the following installs `pandas`, `matplotlib` and a specific version of `scipy`:

```
(myenv)$ conda install scipy=1.6.3 pandas matplotlib
```

**Note** that it is strongly advised not to use conda for package installations after having installed packages via pip, as this may cause inconsistencies in the environment. For best practices, install packages using conda first, and reserve pip for any final installations.

**Note** that package installation may be done on both the cluster login servers as well as on the compute nodes, at least when using the default conda channels.

Should you need to see all packages installed in the environment `myenv` then run:

```
conda list -n myenv
```

Without the `-n` option, packages in the current active environment are listed.



## A faster solver for Conda

The default package dependency solver of Conda is known to be slow or even fail to resolve some environments. If this is your case, you may try an alternative, faster solver for Conda using the `--solver` flag as follows:

```
conda create -n myconda --solver=libmamba package1 package2 ...
```

The option `--solver` is also available for `conda install|remove|update` commands. If you want to enable the `libmamba` solver permanently, either add `solver:libmamba` to your `$HOME/.condarc` file or run the command:

```
conda config --set solver libmamba
```

To revert back to the default classic solver:

```
conda config --remove-key solver
```

## Using your Conda environments

To use applications from your conda environments interactively or in a job script, you first need to load the `Miniforge3` module and then activate the environment containing the applications:

```
module load Miniforge3
conda activate myenv
```

**Note** that we do **not** recommend putting the above lines in your shell init file, e.g. `~/.bashrc`. This may cause issues with your interactive or batch jobs.

Here is a sample job script to run an application from the conda environment:

[conda-app-job.sh](#)

```
#!/bin/bash -l
#SBATCH --job-name=my-conda-application
#SBATCH --nodes=1
#SBATCH --cpus-per-task=20
#SBATCH --mem=60G
#SBATCH --time=00:30:00
#SBATCH --mail-user=user@yourinstitute.uni-hannover.de
#SBATCH --mail-type=END

# Activate your conda environment
module load Miniforge3
conda activate <your_conda_env_name>

# Run app
```

```
<run your application>
```

As already mentioned, if you use conda managed software, you should not mix it with the cluster software modules, thus loading only the Miniforge3 module in your interactive shell or in a job script.

## Pre-Configured Conda Environments

To streamline workflows and minimize redundant installations, several pre-configured conda environments are available to all users on the cluster. These shared environments include a broad selection of packages commonly used across various scientific disciplines, such as data analysis, machine learning, bioinformatics, and more.

Using these environments helps conserve resources and ensure consistent setups, allowing you to quickly get started without needing to install the same packages individually.

Below is a list of environments and some primary packages they contain:

Path to Conda Environment	Key Packages
/sw/system/home/ood/conda/envs/miniforge3-2024.08_python-3.12	numpy, scipy, pandas, scikit-learn, scikit-bio, biopython, sqlalchemy, pymol, matplotlib, seaborn, pytorch, tensorflow, keras
/sw/system/home/ood/conda/envs/miniforge3-2023.08_python-3.10	numpy, scipy, pandas, scikit-learn, matplotlib, seaborn, pytorch, tensorflow, keras, qiime2

For a complete list of packages within each environment, use the command: `conda list -p <path to conda environment>`. If you believe that a shared environment is missing packages that would be beneficial for a reasonably broad user community in the cluster, feel free to reach out to the cluster support team.

For each of the environments listed above, you can also access the corresponding GPU version by simply appending `_gpu` to the environment path.

To activate a shared environment, use the following command:

```
module load Miniforge3
conda activate <path to conda environment>
```

To switch to the GPU-optimized version, use:

```
module load Miniforge3
```

```
conda activate <path to conda environment>_gpu
```

In addition to the command line, these environments are also available through the JupyterLab service provided by the cluster web portal. This allows users to run Jupyter notebooks with the desired conda environments directly from the web interface. For more details on how to access and use the JupyterLab service, please refer to [the cluster web portal documentation](#).

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_miniforge3](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_miniforge3)

Last update: **2025/03/28 17:09**



# Moose

[https://mooseframework.inl.gov/getting\\_started/new\\_users.html](https://mooseframework.inl.gov/getting_started/new_users.html)

## Installation via conda

```
$ module load Miniconda3/4.10.3
Module for Miniconda3, version 4.10.3 loaded

$ conda info --envs
# conda environments:
#
base                                /sw-eb/apps/software/noarch/Miniconda3/4.10.3

$ conda create --prefix=/bigwork/zzzzsaal/mamba

$ conda info --envs
# conda environments:
#
                                /bigwork/zzzzsaal/mamba
base                                /sw-eb/apps/software/noarch/Miniconda3/4.10.3

$ conda activate /bigwork/zzzzsaal/mamba
$ conda info --envs
# conda environments:
#
                                * /bigwork/zzzzsaal/mamba
base                                /sw-eb/apps/software/noarch/Miniconda3/4.10.3

$ conda install -c conda-forge mamba

$ conda config --add channels https://conda.software.inl.gov/public

$ mamba create --prefix /bigwork/zzzzsaal/moose moose-dev

$ conda activate /bigwork/zzzzsaal/moose

$ conda info --envs
# conda environments:
#
                                /bigwork/zzzzsaal/mamba
                                * /bigwork/zzzzsaal/moose
base                                /sw-eb/apps/software/noarch/Miniconda3/4.10.3
```

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_moose](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_moose)

Last update: **2023/07/14 10:16**



# mpiFileUtils

---

## Handling large datasets within the cluster

---

While working on the cluster, you may need to perform operations such as *copy*, *delete*, *sync*, *find*, etc. on a very large number of files or on files that are very large in size. While we do recommend to first have a look whether you can reduce the number of files you need to handle e.g. by packing them (because both the file system Lustre that is driving BIGWORK and the controllers of the disk drives have limits how many files they can handle at once. That means that if you need more IOPs, the overall performance you get may deteriorate significantly), we also realize that this frequently is more a long-term project. For this reason, we provide some parallel tools provided by the MPI-based package [mpiFileUtils](#). The standard Unix commands like `cp`, `rm` or `find` are often comparatively slow, as they are implemented as single process applications.

As a typical example, consider copying directories containing a large number of files from your \$BIGWORK to the local \$TMPDIR storage on the compute nodes you allocated for further processing by your job, or/and transferring computation results back to your \$BIGWORK.

Another example would be a quick freeing up of space in your \$BIGWORK by first copying files to your \$PROJECT storage and then deleting them from \$BIGWORK.

Also, you could utilize the command `dsync`, if you use your \$PROJECT storage for backing up directories on your \$BIGWORK.

Below we will look at some of the `mpiFileUtils` tools in these and other practical examples.

In order to speed up the recursive transfer of the **contents** of the directory \$BIGWORK/source to \$TMPDIR/dest on a compute node, put the lines below in your job script or enter them at the command prompt of your interactive job - we assume that you've requested 8 cores for your batch job:

```
module load GCC/8.3.0 OpenMPI/3.1.4 mpifileutils/0.11
mpirun -np 8 dcp $BIGWORK/source/ $TMPDIR/dest
```

Please note a trailing slash (/) on the source path, which means “**copy the contents of the source directory**”. If the \$TMPDIR/dest directory does not exist before copying, it will be created. The command `mpirun` launches `dcp` (distributed copy) in parallel with 8 MPI processes.

The directory \$BIGWORK/dir and its contents can be removed quickly using the `drm` (distributed remove) command:

```
mpirun -np 8 drm $BIGWORK/dir
```

**Note:** here and below we assume that the module `mpifileutils/0.11` is already loaded.

The command `drm` supports the option `--match` allowing to delete files selectively. See `man drm` for more information.

The next useful command is `dfind` - a parallel version of the unix command `find`. In this example we find all files on `$BIGWORK` larger than 1GB and write them to a file:

```
mpirun -np 8 dfind -v --output files_1GB.txt --size +1GB $BIGWORK
```

To learn more about other `dfind` options, type `man dfind`.

If you want to synchronize the directory `$BIGWORK/source` to `$PROJECT/dest` such that the directory `$PROJECT/dest` has content, ownership, timestamps and permissions of `$BIGWORK/source`, execute:

```
mpirun -np 8 dsync -D $BIGWORK/source $PROJECT/dest
```

Note that for this example the `dsync` command has to be launched on the login node where both `$BIGWORK` and `$PROJECT` are available.

The last `mpiFileUtils` tools we consider in this section are `dtar` and `dbz2`. The following creates a compressed archive `mydir.tar.dbz2` of the directory `mydir`:

```
mpirun -np 8 dtar -c -f mydir.tar mydir
mpirun -np 8 dbz2 -z mydir.tar
```

**Please note:** If the directory to be archived is located on your `$HOME`, the archive file itself should be placed on `$BIGWORK`.

**Please note:** Transferring a large number of files from the cluster to an external server or vice versa as a single (compressed) tar archive is much more efficient than copying files individually.

Some other useful commands are:

- `dstripe` - restripe(lustre) files in paths
- `dwalk` - list, sort, summarize files
- `ddup` - find duplicate files

A complete list of `mpiFileUtils` utilities and their description can be found at <http://mpifileutils.readthedocs.io/>.

You might also consider alternative tools like [GNU parallel](#), [pigz](#) or [pcp](#). They are all available as modules on the cluster.

From:  
<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:  
[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_mpiutils](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_mpiutils)

Last update: **2021/10/22 09:07**



# NFFT

**Please note:** These instructions were written for [NFFT 3.4.1](#).

NFFT is available as a module on the cluster system. However, there may be situations where you need to compile your own version. For example if you need the MATLAB interface as mex file. When compiling from source, take into account that the cluster system's CPU architecture is heterogeneous and you best compile a version for every architecture you will be using in order to avoid problems, see section [8.1](#).

Execute the following commands to load prerequisites.

```
$ module load foss/2016a FFTW/3.3.4 gomp/2016a
```

Download the NFFT sources and change to the directory containing nfft-3.4.1 sources. Execute the following commands in order to build NFFT with MATLAB 2018a interface. Substitute zzzzsaal with your own user name.

```
$ ./configure --prefix=/bigwork/zzzzsaal/nfft/nfft-3.4.1-compiled  
--with-matlab=/sw-eb/apps/software/haswell/Core/MATLAB/2018a/  
$ make  
$ make install
```

You end up with libnfft.mexa64 located in the lib directory. Use it as detailed [here](#). In order to test your build, start the MATLAB version you built for.

```
$ module load MATLAB/2018a  
$ matlab -nodisplay -nosplash
```

Inside MATLAB issue the following commands. Change zzzzsaal to your user name.

```
>> addpath(genpath('/bigwork/zzzzsaal/nfft/nfft-3.4.1-compiled/lib'))>> cd  
/bigwork/zzzzsaal/nfft/nfft-3.4.1-compiled/share/nfft/matlab/nfft/>>  
simple_test
```

From:  
<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:  
[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/401\\_nfft](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/401_nfft)

Last update: **2021/04/18 13:44**





# Remote visualization

---

While the cluster is mainly designed to run batch jobs without user interaction, we realize that in some cases it is necessary to also interactively work on a node - for example when setting up a 3D-model that is too large to fit into your workstation's memory in a simulation software, or if you need to postprocess/filter large amounts of data that has been generated by a job in your BIGWORK directory. To facilitate that, we have installed a special node that has specifically been set up to support interactive/graphical use. We will watch its usage and possibly add further nodes in the future, depending on user demand. CURRENTLY, THIS IS JUST A TEST.

**Please note:** in this initial phase and due to the very limited resources, only one visualization session (job) lasting no longer than 3 hours is allowed per user at the same time to give everyone a chance to test the service. When the session expires, it will be terminated automatically and without warning (!). A job can request a maximum of 8 CPU cores and 32 GB of main memory, and a maximum of three simultaneous sessions can run on the visualization node at any time to limit the influence of other users on the experience of others.

**Please note:** If you do not require an intensive 3D hardware acceleration, please use a regular remote desktop session instead to interactively work with your GUI applications (accessed via the menu **Interactive Apps > Cluster Remote Desktop** on the web portal).

## Concept of a visualization server

The traditional approach to remote data visualization relies on tunneling the X11 protocol through an SSH connection to run graphical applications. That means adding up latencies between client and server, resulting in slow reactions and a “sluggish” handling of large 3D-geometries in particular. The approach of a visualization server is to perform all rendering on the server's graphics hardware, transferring only the resulting 2D-images to the remote workstation for display. This permits to operate 3D applications efficiently over standard network connections. In addition, the local workstation neither requires a special dedicated graphics card nor large memory or an installation of 3D processing tools.

Since the cluster visualization servers are managed by SLURM via the partition `vis`, you need to submit a job to this partition to get access to the server's resources. Currently, the `vis` partition consist of only one OpenGL-capable server with a single NVIDIA Quadro P4000 GPU, 20 CPU cores and 92 GB of main memory. The node runs a 3D X server and has all the required software tools installed, including TurboVNC (Virtual Network Computing) and VirtualGL (an open source software that intercepts 3D remote-rendering commands). The server has a fast connection to both the BIGWORK and PROJECT file-systems (where your datasets are supposed to be placed).

## Starting a visualization session

To use the remote visualization service of the LUIS cluster, you need to authenticate to the cluster's [web portal](#) and on the dashboard activate the **Cluster Visualization Desktop** item in the **Interactive Apps** menu, cf. [figure 1](#). On the service page that opens, leave the checkbox `Enable`

OpenGL desktop session deactivated for the time being (see below the section [Optimization](#)). Set the remaining session(job) parameters according to your requirements and click the **Launch** button at the bottom of the service page.



Fig. 1: Remote visualization settings page

You will be redirected to the session information page, where you can see the status of your job, see the top window in [figure 2](#). Wait for the job to get into running state, then click **Launch Cluster Visualization Desktop**, see the bottom window in [figure 2](#), to connect to the remote VNC session that is allocated on the visualization server.

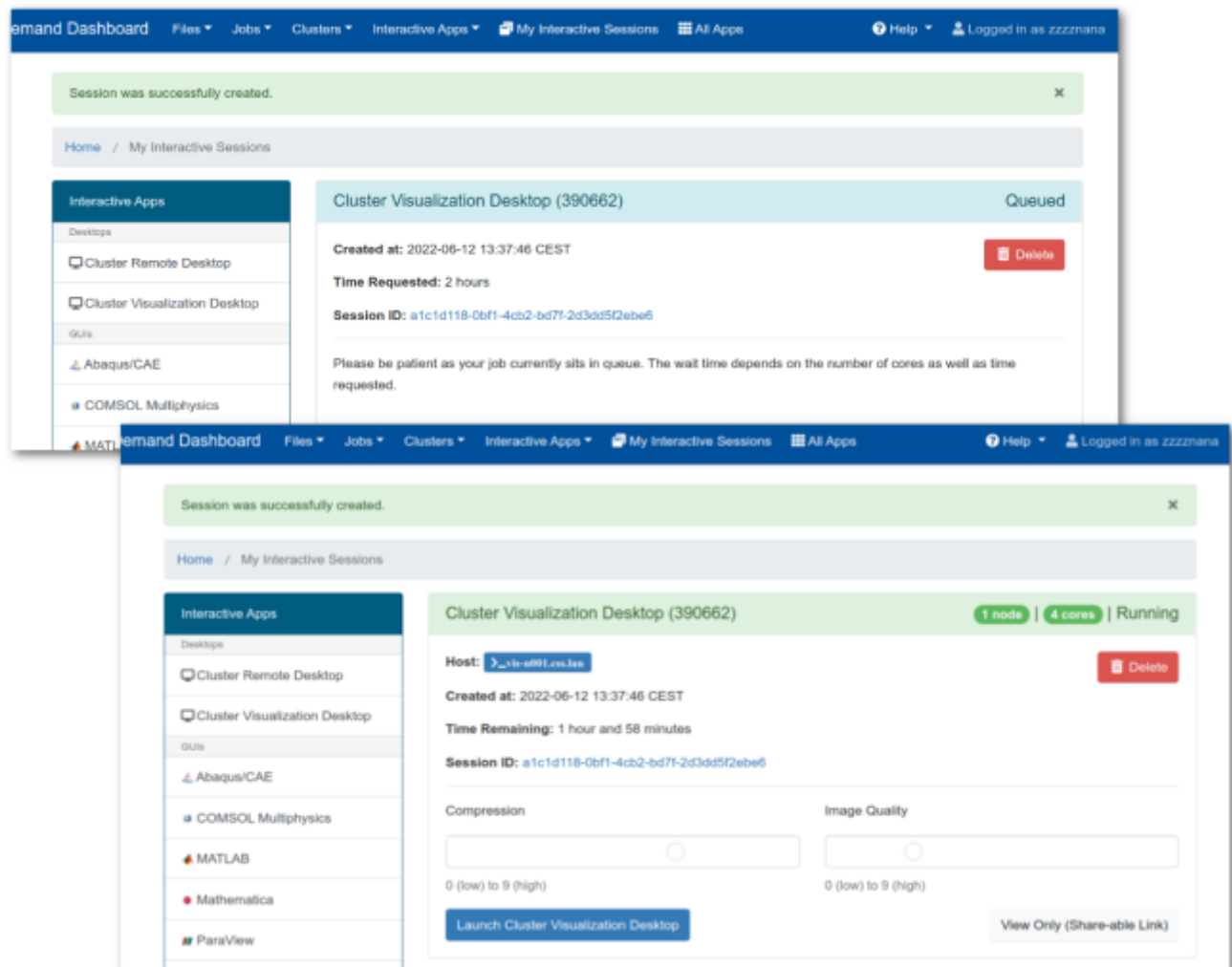


Fig. 2: Remote visualization session job

Once in the desktop session, invoke a terminal (Terminal Emulator or similar, e.g. Terminator from the sub-menu **Applications > System**) from the menu **Applications** at the top-left of the screen as shown in [figure 3](#).

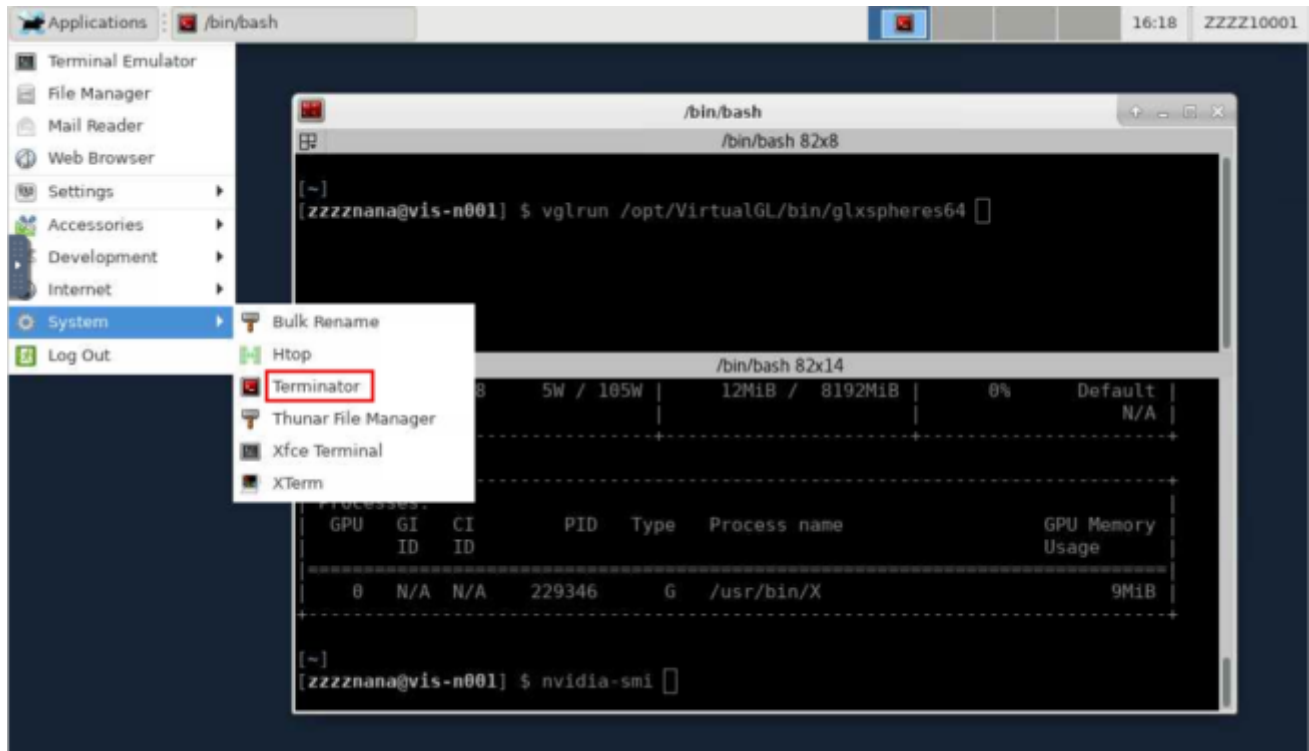


Fig. 3: Remote visualization desktop

To terminate your visualization session either click the item **Applications > Log Out** (figure 3) or cancel the session job using the **Delete** button on the **My Interactive Sessions** page (figure 2).

## First test

As a first test, we'll run the OpenGL program `glxspheres64` supplied with the VirtualGL package (cf. figure 4):

```
[user@vis-n001 ~]$ glxspheres64
```

You should see a new window containing some colourful animated spheres. Notice the frame rate. Exit the program (ESC) and start it anew, this time using:

```
[user@vis-n001 ~]$ vglrun glxspheres64
```

You should see a noticeable difference in framerate and measured performance. Exit the tool. You do not need to do this every time, this was just a demonstration to give you a feel for the difference to expect.

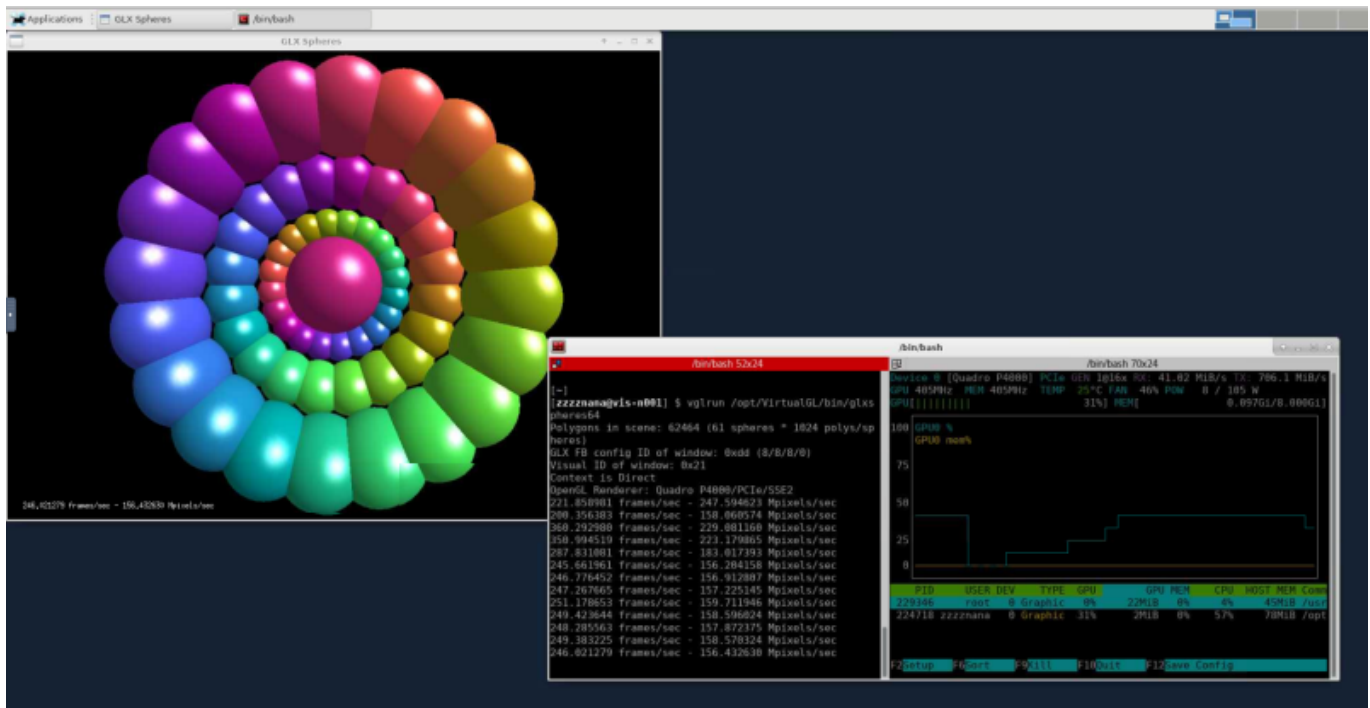


Fig. 4: Running glxspheres64 on the cluster

You can now activate the application you want to use (by loading the appropriate modules, activating your conda environment, etc.) and run it using the `vglrun` prefix:

```
[user@vis-n001 ~]$ vglrun <your-application> <your-application-options>
```

To check whether the graphics cards of the server are actually being used by your application, open a second terminal window and run the command `nvttop`. `nvttop` (Neat Videocard TOP) is a (h)top-like tool that provides real-time information about the usage of NVIDIA and AMD GPUs as well as the processes executing on the GPUs (see the right tab of the terminal window in [figure 4](#)). The `nvidia-smi` utility may also be useful.

## Optimization

Here are some further optimizations you could try:

- If you activate the checkbox near `Enable OpenGL desktop session` in the session setup, the X server itself will be started using `vglrun`. This should have the effect that software started from within a terminal opened on that desktop should automatically start with OpenGL acceleration enabled (without explicitly using `vglrun`). In some cases, this may not work as expected — then just leave the option deactivated and start your software using `vglrun`.
- In case your internet connection bandwidth is small, you can tell `vglrun` to use a smaller frame rate. Compare: `vglrun -fps 60 glxspheres64`, `vglrun -fps 20 glxspheres64` and `vglrun -fps 10 glxspheres64`.
- You may also find other `vglrun`-parameters to tune interactivity by just typing the command by itself. Namely the options `-c`, `-np`, `-q` could be of use over a limited connection.

# Application examples

Here are instructions on how to run some applications to enable hardware accelerated OpenGL rendering.

## ABAQUS

In a terminal window, load the ABAQUS module and type:

```
vglrun abaqus cae
```

## ANSYS Fluent

Load the ANSYS module and run:

```
vglrun fluent -driver opengl
```

If ANSYS Fluent is launched from ANSYS Workbench, the following variables must be set, otherwise ANSYS Fluent will not use hardware rendering.

```
export FLUENT_WB_OPTIONAL_ARGS="-driver opengl"  
export CORTEX_PRE=/opt/VirtualGL/bin/vglrun.
```

## COMSOL

Load the COMSOL module and execute:

```
vglrun comsol -3drend ogl
```

## GaussView

Load the GaussView module and execute:

```
vglrun gview
```

## MATLAB

Load the MATLAB module and execute:

```
vglrun matlab -nosoftwareopengl
```

## ParaView

```
module load GCC/11.2.0 OpenMPI/4.1.1 ParaView/5.9.1-mpi  
vglrun paraview
```

## VMD

```
module load GCC/10.2.0 OpenMPI/4.0.5 VMD/1.9.4a51  
vglrun vmd
```

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/451\\_remote\\_visualization](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/451_remote_visualization)

Last update: **2022/06/13 10:01**



# When your work is done

When you are done with your work on the cluster system and possibly leave the LUH, it would be nice if you did a few things before you go, to help others get a lightweight system that does not lug around old data:

1. clean up your directories:
  1. \$BIGWORK — where most of the stuff should reside, to make room for the next generation
  2. \$HOME — to allow the backup of others to run quicker
  3. your subdirectories in \$PROJECT — in case you have no successor who could use the data, to reduce the quota occupied for your colleagues
  4. possibly \$SOFTWARE — in case you installed stuff here that your colleagues don't need any more
2. have your project management (at your institute) delete your user account

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/500\\_when\\_your\\_work\\_is\\_done](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/500_when_your_work_is_done)

Last update: **2025/08/19 11:25**





# Contact & Help

---

Need help running your job on the cluster? Need a question answered about a particular tool or cluster resources?

- Please contact the cluster team at [cluster-help@luis.uni-hannover.de](mailto:cluster-help@luis.uni-hannover.de) **using the email address associated with your cluster account.**
- You can also call us on the phone +49 511 762 791000 (be aware, though, that some problems are complex and we sometimes need to look through logs and think about what we see, too).
- **Sie können uns natürlich auch auf Deutsch kontaktieren** (we get a lot of inquiries in English, even from obviously German-speaking users; so we just want to mention that both German and English are equally suitable to contact our support. We maintain the documentation in English since many users come from an international background and almost anything on a cluster system is in English, anyway).

**However, please read this documentation first and check whether your question is already answered in the [FAQ](#)** or possibly even a page specifically written for the issue at hand. If you find the documentation lacking, let us know what's missing so we can update and improve it. We will also be happy to integrate useful workflows users develop e.g. using software installed on the cluster, since we are no experts in these products.

Whenever you have a question, please try to help us help you by providing the following information:

- **Do NOT use “reply” to an old mail to create a new inquiry.** Create a new one, choose a suitable subject line, and do not address it to an individual person, but to the common cluster-help-Adress mentioned above. Otherwise, the topic will be both obsolete and completely different to what you really want, and on top, it will be attached to an old case. You will **not** get a timely response if we think you are just responding to an introductory course invitation, or if you personally address someone who is currently not at their desk. And your case will get sorted to the bottom of the queue in at least some of the to-do lists we have due to its old age — and thus easily get overlooked.
- **State your username** (nh...), **Job ID(s)** and **locations on the system** (directories, hostnames etc.)
- **NEVER send us your password.** We do not need it. **Sending your password will lead to your account being locked.**
- **One problem per case:** Nobody at the LUIS knows everything. If you pack several different topics into one case, fewer persons are able to answer them all, even if three out of your five questions would be easy to reply to. So the most difficult question in an inquiry will usually influence response time. One question at a time will usually get you answers much quicker. *If* things could depend on each other, then by all means describe them in one case. Some things are complex.
- **Details known about the problem, e.g.**
  - JobID(s) of the job(s) that have problems
  - what command(s) in what sequence lead to the problem? What did you do? In which directory?
  - **Have you checked your quota (use the command `checkquota`)?** (really!)
  - the time something happened; be precise if you can: log files are big and sometimes there are 1000 entries every minute

- the batch script (as an attachment or - even better - its location on the cluster)
- output from the program, if available, like myjob.o12345 and myjob.e12345 (as an attachment or - even better - its location on the cluster)
- what did you already try to solve the problem yourself?
- for batch jobs:
  - did you request sufficient memory for your job?
  - in case of problems running a multi-node job: did you already try a single-node job? What was the outcome? Is it even reasonable to run your job over several nodes when it would fit on a single node (intra-node communication between processes may be a lot faster than inter-node communication)?
  - is the combination of cores, nodes and memory reasonable or even possible on the system? Did you check the [list of nodes](#) available to find your setup, or did you just copy a very old setup for a partition that does not even exist any more from a former colleague of a former colleague?
  - make sure you do not change/delete the files we need to trace your problem while we are at the case. The best method is to provide a minimal setup in a separate directory that we can use. The simpler and clearer your bug report, the easier it is to isolate the problem. A very good case shows “this runs ok” in comparison to “this does not” (e.g. a smaller job setup, less single/multi nodes, memory request, different parameter, different environment variable...)
- for questions related to file transfers or graphical logins: **how do you access the cluster?** Weblogin? PuTTY? X2Go? SSH? What is your workstation's OS?
- what is unusual or unexpected in the output files you already have?
- bus errors and segmentation violations usually are not a system problem. They occur when a program tries to access memory locations or hardware that does not exist, which may happen e.g. when the software has problems with proper use of pointers or when it does not deal properly with unexpected input data.
- Screenshots help, but consider doing copy&paste in plain text for textual information
- **After you got your response, do not just reuse the case to ask for something completely different.** The person who replied to your first case may not know the answer to your new problem, but the case remains assigned to the same agent. One case per problem.
- In case you solve your problem yourself in the time it takes us to answer your case, please tell us. Ideally, you also tell us *how* you solved it: another user may have the same problem one day, and we do not know everything.
- **We will NOT do your Homework for you.** That means we expect you to at least read your own log files and check with the documentation and the FAQ on this site. We also expect you to do your own thinking. It is *not* a problem if you have read your stuff and just do not understand what it means. We know this is complex business and we will try to help. But please try to avoid creating a case that just says “my jobs fail” and clearly shows that you neither read any documentation nor used your own brains.

## Examples for typical cases that make it almost impossible for us to help without asking further questions

- **“The cluster is running very slow”** *What are you trying to do? On what node(s)? How do you log in (describe the method, ssh, X2Go, OpenOnDemand)? How do you quantify “slow”? What part of the cluster is “slow”, a file system?*
- **“I cannot log in”** *What is your account name? Did you succeed to log in previously, or is this the first time? How do you try to log in (ex. ssh, X2Go)? Did you check that you are within the LUH network, either in an institute, or are you using the VPN? What is the operating system of the machine you come from? When exactly (hh:mm) did you try it? Do you even get a password*

*prompt or similar? Which messages, if any, did you get? Did you try an alternative method, like e.g. ssh instead of the web site? Did you check your disk quota?*

- **“My job does not start”** *What do you mean by “does not start” - 2 minutes or 3 weeks? Where is the job script or the job ID, respectively? Which account? Are you using a resource that is scarce in the cluster, like large-memory nodes or GPU nodes?*
- **“My jobs crashed”** *Which job IDs, what account? Did you check that you are not over quota (checkquota)? Did the same jobs run previously without errors, and if so, what changed?*
- **“Some part of a complex software suite like Ansys, Abaqus, Comsol, Matlab etc. does not work as I expect”** *What exactly did you do to create the problem? Please also include a description of how you started the software, which version/modules you loaded, what commands you entered, where you clicked to start what. We are in no way experts in every software that's available on the cluster and can only provide basic support for individual applications on a best-effort basis.*

From:

<https://docs.cluster.uni-hannover.de/> - **Cluster Docs**

Permanent link:

[https://docs.cluster.uni-hannover.de/doku.php/guide/to\\_pdf/550\\_how\\_to\\_get\\_support](https://docs.cluster.uni-hannover.de/doku.php/guide/to_pdf/550_how_to_get_support)

Last update: **2025/07/22 07:52**

