UNIVERSITÄT BERN

How UBELIX Works

Contents

UBELIX consists of a multitude of computers which are connected to each other on a network. They are very similar to standard PCs but have a special shape and are primarily focused on computing power.

All computers are connected to a private network which cannot easily be accessed from the outside. Special servers allow the user to access that network. Within UBELIX, private names and addresses are used. However, connections to the outside are possible from every host. The default domain name within UBELIX is ubelix.unibe.ch, a computer might for instance be called *cnode01.ubelix.unibe.ch.*

There are two classes of computers: The actual computations are done on the cluster nodes. Jobs are automatically distributed to the nodes via a queuing system, the user does not have to worry about which nodes are currently available and where the job is supposed to be run. The nodes are subdivided into different classes, within each class all the nodes are equipped with identical hardware. All nodes are 64-bit multi-CPU systems.

The servers are responsible for the coordination between the nodes. They offer basic services and are not directly relevant to the user, except for the submit host which allows the user to access the cluster.

Submit Host

The user can connect to the submit host using SSH. Unix-systems (Linux, Mac OS X) usually ship with an SSH-client pre-installed. For Windows, there are several free clients, e.g. PuTTY. The official address of the submit host is submit.unibe.ch. Access is only possible within the network of the university. If you are outside you can use the university VPN client to get access to the network.

On the submit host, you can manage your jobs on the queuing system (for details, see Submitting Jobs). There are numerous compilers available to compile your own software. Also, you can try your programs on the submit host before you actually submit them. But be aware, that this computer is shared with all the other cluster users. You should not start computationally intensive and time consuming programs on the submit host.

Cluster Nodes

The following table contains the hardware details for the different nodes.

Class	Number of nodes	Architecture	CPU Cores	RAM	Disk
cnodes	51	AMD Opteron 2354 2.2GHz	8	16GB	200GB
dnodes	20	Intel Xeon E5450 3.0GHz	8	16GB	100GB
enodes	36	Intel Xeon X5550 2.67GHz / Intel Xeon E5620 2.4GHz	8	32GB	100GB
fnodes	28	Intel Xeon E5649 2.53GHz	12	48GB	250GB
gnodes	20	AMD Opteron 8356 2.3GHz	12	32GB	350GB
hnodes	1-42	Intel Xeon E5-2665 2.40GHz	16	64GB	250GB
hnodes	43-49	Intel Xeon E5-2695v2 2.40GHz	24	96GB	500GB
jnodes	21	Intel Xeon E5-2665 2.40GHz	16	256GB	500GB

As the jobs are distributed by a queuing system, you normally won't have to log in directly on the nodes. In case you need an interactive session, that can also be provided by the queuing system (see Interactive Jobs).
Jobs should never be started by hand directly on the nodes. This would interfere with the queuing system.

Storage

UBELIX provides a few storage locations that are summarized in the table below.

	Path	Connection	Availability	Backup	Quota
Home	/home/ubelix/group/user	Network	global	no	yes*
Safe storage	/home/storage/group/user	Network	submit	yes	yes (50GB)
Job scratch	$TMP	Harddisk	nodes	no	yes**

* 2TB per user, 15TB for the group, we may grant you more on an individual basis if needed, please e-mail us.

** you can request this quota yourself when submitting a job, see here.

To check current quota usage, issue *quota* on a submit host.

Your usual home directory is in /home/ubelix/group/user. It is located on a fast, network attached storage (General Parallel Files System, GPFS) and is intended to be used for your general use such as custom installed software, configuration files and your cluster jobs and results, i.e. as a global scratch. Have a look at the characteristics of this file system. Please be aware that the GPFS filesystem is not backed up and we do not give any guarantees in the case of data loss or corruption. In the case of a major failure, data on the GPFS Filesystem might be lost permanently. To back up important files and results, you must manually copy them to /home/storage/group/user where they will be backed up to an external backuper on a regular basis. Please note that the backup storage is not available from the compute nodes.

There are also local scratch directories on all cluster nodes. As write operations on the global storage are slower than on local storage, you might want to consider saving intermediate results on a storage local to the compute node rather than directly on the network storage. Within a job, the local scratch's path is stored in the environment variable $TMP. Data can be copied to $TMP at the beginning of the job. During the computations, the data is then accessed locally. At the end of the job, data can be copied back on the network storage and the rest is removed. All data left on $TMP will automatically be removed after the job has finished. Local scratch is only available to a job if it has been reserved as a resource. Please see Submitting Jobs on how to do that.

Software

UBELIX runs RedHat Enterprise Linux and comes with a plethora of software pre-installed. THere is an open list of tools that you can find on the Cluster: GCC, Intel Compiler, PGI Compiler, Intel MKL, AMD ACML, Altas/Lapack, Openmpi, Mvapich, Java, Matlab, NAMD2, Octave, R, Gnuplot, Vim, Emacs, TeXLive, Subversion, CVS, Ghostscript, Perl, Python and Ruby.

The RedHat setup supports Environment Modules to select software.

In co-operation with the Vital-IT Group of the SIB Swiss Institute of Bioinformatics, a large set of bioinformatics software tools and databases is available to the life science community.

If you think a particular software you use should be installed on the cluster, send us a message. Most of the time however, it is easier and quicker for you to install the software for yourself only. See Installing Custom Software for more information on how to do this.

Universität Bern | Informatikdienste | Gesellschaftsstrasse 6 | 3012 Bern | helpdesk@id.unibe.ch | Tel +41 (0)31 631 49 99

Further information

- GPFS File System Hints (http://www.id.unibe.ch/content/services/ubelix/how_ubelix_works/gpfs_file_system_hints/)

Support

For questions and problems

- grid-support@id.unibe.ch (mailto:grid-support@id.unibe.ch)

Quick Links

- job monitoring (http://ubelix.unibe.ch/xmlqstat/jobs.html)
- grid-users mailing list (http://listserv.unibe.ch/mailman/listinfo/grid-users)

Quck Links

- Swiss National Grid Association (http://www.swing-grid.ch/)
- Swiss Multi-Science Computing Grid (http://smscg.ch)
- The Swiss HPC Service Provider Community (http://www.hpc-ch.org/)

UNIVERSITÄT BERN

Submitting Jobs

Contents

Basics

A cluster manages several resources such as CPUs, memory, storage etc. In order to have these used as efficiently as possible by various jobs submitted by various users, a queuing system is usually used. In theory, a cluster could also be operated without such a system, but that would complicate operation and lead to conflicts between users.

UBELIX runs on Sun/Oracle Grid Engine (SGE). This chapter is supposed to give an introduction in SGE's basic functionality. For further information, see the official documentation available at the address mentioned above, as well as the man pages on UBELIX (*man sge_intro*).

Submitting Jobs

Jobs usually consist of a shell script which calls the actual programs with the required options. The command to submit a job is qsub and can only be executed on the submit host. The following table shows an overview of qsub's most important options. The following example submits a job which executes the shell script script.sh (here is a further example):

user@submit ~ $ qsub -M user@group.unibe.ch script.sh

Option	Description (default)
-M user@group.unibe.ch	Mailaddress for notifications
-m b,e,a,s,n	Events for notifications
-cwd	Execute the job from the current working directory. Also puts the output files there.
-b yes / no	Job script is a binary (default: no)
-r yes / no	Job is rerun if it was aborted
-l res1=val,res2=val	request resources (defaults: see Resources and queues)
-p prio	Job priority (default:~0)
-q queue	Queue name (default: all.q, alternative: short.q, mpi.q)
-pe smp / mpi	Parallel environment (only use this parameter for parallel jobs)

You should always indicate a valid e-mail address with -M *<user>*@*<group>*.unibe.ch. By default,

mail is only sent when a job is aborted. With *-m b,e,a,s,n,...* you can specify additional events which trigger an e-mail notification. For example, -m be sends an e-mail at the beginning and end of a job, -m bea also sends an e-mail when a job is aborted or rescheduled. For details, see the man page.

-cwd sets the job's current working directory. By default, this is also where the files for STDOUT (output) and STERR (errors) are created. If you don't specify this explicitly, they will be called *<job-name>.o<job-id>* for STDOUT and *<job-name>.e<job-id>* for STERR. The job-id is a unique integer for each job. It is continuously incremented whenever a new job is submitted.

In case you'd like to directly execute a binary file instead of a script you have to set *-b yes.* The option *-r yes* causes the job to automatically rerun with the same options after a crash. In this case, the job-id is not changed.

Using *-l* you can tell SGE to allocate a list of resources for a job. You'll find more information on this in the next chapter. The option *-p prio* offers the possibility the adjust the job's priority within your user to change the order in which your jobs are run. *prio* has to be an integer between -1023 and 0, the higher the number, the higher the priority.

short.q offers the possibility to run jobs which have to be executed urgently. Here, CPU time is usually limited to an hour. **Important:** Only use *short.q* if your jobs need to complete within a short time.

mpi.q offers the possibility to run parallel jobs which have to be executed over multiple processor cores or even hosts. This queue supports the smp and mpi parallel environments **Important:** The *mpi.q* only accepts jobs with a valid parallel environment.

Resources and Queues

SGE expects that you to allocate resources using qsub's *-l <res1>=<val1>,<res2>=<val2>,...* option, at least for *h_cpu* and *h_vmem*. The following table shows an overview of the most frequently used resources on UBELIX:

Name	Description	Possible values
h_cpu	CPU time, this parameter is *mandatory*	*hh:mm:ss* or *ss*
h_rt	Run time	*hh:mm:ss* or *ss*
h_vmem	Memory, this parameter is *mandatory*	*x, xK, xM, xG, xk, xm, xg*
scratch	Request local scratch	*1*
scratch_size	Local scratch, free space	*x, xK, xM, xG, xk, xm, xg*
scratch_files	Local scratch, number of files	*x, xK, xM, xG, xk, xm, xg*
use_highmem	If set, the highmem.q is used for the job.	1

Using *h_cpu* you can specify the CPU time your job requires. This only takes the actually used CPU cycles into account, idle cycles or cycles used by system processes are not included. *h_vmem* specifies the amount of memory required. Both resources have an upper limit which can not be exceeded and which depends on the queue and on the host. Be sure to specify a valid format, including the unit, eg. *-l h_cpu=50:15:00 -l h_vmem=500M*. If your jobs just remains pending, then the chance is high that you didn't specify the right format.

h_rt specifies the job's runtime independently of the used CPU-time and can be ignored in most cases.

If a job needs local scratch, it needs to be requested with the following arguments: *scratch=1*, further *scratch_size=<size>*, which specifies the desired amount of space in kilo-, mega- or giga-bytes (using upper case letters) or in kilo-, mega- or giga-bits (using lower case letters) and *scratch_files=<#files>*, which specifies the number of files, accepting the same suffixes as scratch_size. Make sure that you add all three arguments to your *qsub* command.

Resource limits do not only depend on the resource request you make in your jobs, but also they have a upper system limit which you cannot exceed:

Resource	Queue/Host	Limit
h_cpu	all.q	96:00:00
	mpi.q	72:00:00
	short.q	01:00:00
h_rt	all.q	97:00:00

	mpi.q	73:00:00
	short.q	01:10:00
h_vmem	cnode, dnode	16G (2G per job)
	enode	32G (4G per job)
	fnode	48G (4G per job)
	gnode	32G (~3G per job)
	hnode	64G (4G per job)
	jnode	256G (16G per job)
scratch_size	cnode	200G
	dnode, enode	100G
	fnode, hnode	250G
	gnode	350G
	jnode	500G
scratch_files	cnode	200M
	dnode, enode	100M
	fnode, hnode	250M
	gnode	350M
	jnode	500M

If a job exceeds the queue limits or the user-specified limits it is aborted. It's possible to have the job warned before it's canceled by setting the soft limits *s_cpu, s_rt, s_vmem*. In this case, a signal is sent to the process which can be captured and allows the job to react accordingly (for details, see *man queue_conf*). For example this allows your job to cleanly close any open files before it's terminated. **Important:** You need to set *h_cpu* and *h_vmem*. If not job submission fails with an error message or your jobs will be queued/in state pending indefinitely.

The default limits may change. *qstat -F* and *qhost -F* inform you about available resources for each queue and each host. As these commands produce a lot of output, it's recommended to search for specific resources, eg. by using *qhost -F scratch_size* or by piping the output to *grep*.

You should always specify your resources as low as possible since unused resources can not be assigned to a different job and are thus wasted. Also, a job which requires fewer resources may be preferred because if there are not enough resources available, a job can't be started. On the other hand, if you set your resources to low your job might be aborted prematurely. Thus you should learn to know your programs' resource requirements and find reasonable values.

A side note on memory usage:
On alls queue except the highmem.q the memory is **not** consumable. This means your job should not use more memory than what is available on a given host per core, i.e. 4G for fnode (48G/12Cores). If your jobs need 10G and 12 of your jobs are simultaneously running on a given fnode, the memory consumption clearly overloads the compute node. To sum up, if your jobs require a lot of RAM (>~5G) you can use the highmem.q (add use_highmem=1 to your resource request) where memory is consumable and where it is safe to use large portions of memory. This setting might happen to change in the near future though, which will make this paragraph obsolete.

Interactive Jobs

Besides regular batch jobs, it's also possible to run interactive applications. In order to start an interactive job you should use *qlogin*, the commands qsh and qrsh are currently disabled. The command waits until a node becomes available and then logs in on that node. There you can execute your programs like on the submit host. Remember that the resource limits also apply for interactive jobs, and you can specify the same resource options to *qlogin* as to *qsub*. However, using interactive jobs you have to wait until a node becomes available which can take a while. If the all.q is full, you can use the short.q for short running interactive jobs, since it's more likely that a slot is free in this queue.

qlogin's options mostly correspond to those of *qsub*. The man page informs you about possible differences. Also consider that interactive jobs can only be submitted to queues that support interactive jobs. *qhost -q* will show all queue instances an what type of jobs can be run on them.

Monitoring Jobs

There are several possibilities to monitor a job's status. On the submit host, you can run *qstat* which, if called without any further arguments, shows all users' waiting and running jobs. The *-u <username>* option can be used to only show jobs of a specific user. *-s p$|$r$|$z* filters the output by the job's state (*p: pending, r: running, z: finished*). If called with *-j <job-id>*, qstat shows details for a specific job, e.g. the reason why a waiting job has not been dispatched yet. You can find further options in the man page.

In case you don't have a shell at hand, you can also monitor the jobs on a web interface.

Deleting Jobs

Running or waiting jobs can be deleted with qdel. Using qdel, the option *-j <job-id>* allows you to delete individual jobs or job lists. Further options allow you for instance to delete all jobs from a specific user. The corresponding man page gives an overview.

Parallel Jobs

Using parallel jobs, you can have your job spread over several concurrent processes. Solving a problem using sequential jobs (jobs running within a single process) is usually much simpler to program and more flexible. However, there are problems which are parallel by nature and are hard to split into independent sequential jobs. Also, it can be more efficient if highly coupled processes work with the same data instead of working independently.

Currently, UBELIX offers two parallel environments which allows you to run jobs distributed over several CPUs on a single node (using SMP) or distributed over multiple nodes (using MPI). On our current hardware, up to 8 CPU cores can be used for an SMP job and up to 200 CPU cores for MPI.

reserve slots as they become free

In order to submit an SMP job you basically need two further options for *qsub: -pe smp <slots>* and *-R y*. The former specifies that the job should run in the *smp* parallel environment and the number of slots (CPU core's) to be used. Reasonable values for *<slots>* are *2, 3, 4, 5, 6, 7, 8*. It's also possible to specify a range of slots, like *2-8*. In this case, the job starts with the number of slots presently available. If the cluster is busy, the job will most likely start off using 2 slots. The second option specifies that in case there is currently no node with enough free slots, slots are to be reserved. Otherwise, a parallel job might have to wait forever because single free slots could constantly be reassigned to sequential jobs. If you use parallel jobs it can take a while until the job starts - first, a node with enough free slots has to become available.

Parallel jobs with MPI are similar, but a bit more complicated. You need to supply the same 2 additional options to *qsub: -pe mpi <slots>* and *-R y*. With this you're specifying the *mpi* parallel environment and again the number of slots (CPU cores) to be used. Reasonable values for *<slots>* are between *8 and 128*, you may set higher values but this might lead to the job not getting scheduled, ever. Furthermore you need to compile your software for MPI usage. You can do this using the tools from openmpi: *mpicc, mpic++, mpiCC, mpicxx, mpif77 and mpif90*.

Finally the software needs to be started with either *mpiexec or mpirun*, furthermore it is important **not** to set the *-np, -machinefile or -hostfile* parameters, as this information will be provided automatically by the grid-engine. Setting the *-np* or the other parameters manually will most probably crash the job. If you need to know which machines are going to be used for your application, the machines is provided by the grid-engine under *$TMP/machines*. If you need to know the number of slots you got allocated, it can be read out from the *$NSLOTS* environment variable.

For parallel jobs, resources are always allocated per slot. E.g. a parallel slot which reserves a gigabyte of memory using *h_vmem=1G* and runs on 3 slots gets a total of 3 gigabytes assigned. The smp environment leaves it up to the user to start the individual processes. A parallel job's processes can also be very heterogeneous, the user himself is responsible to make sure that his processes don't exceed the allocated resources. Otherwise the job will, just like a sequential job, be aborted by the system.

As there are multiple parallel environments on multiple queue's it's best to not manually request a queue, as the requested environment might not be able in that queue. The Grid Engine will figure out which queue to use on its own.

631 49 99

Support

For questions and problems

- grid-support@id.unibe.ch (mailto:grid-support@id.unibe.ch)

Quick Links

- job monitoring (http://ubelix.unibe.ch/xmlqstat/jobs.html)
- grid-users mailing list (http://listserv.unibe.ch/mailman/listinfo/grid-users)

Quck Links

- Swiss National Grid Association (http://www.swing-grid.ch/)
- Swiss Multi-Science Computing Grid (http://smscg.ch)
- The Swiss HPC Service Provider Community (http://www.hpc-ch.org/)

UNIVERSITÄT BERN

Advanced Topics

Contents

Screen

Frequently, people want to run programs on the submit host independently from an SSH session. *screen* is a text-mode window-manager which allows you to have a running shell even after you disconnect.

The command *screen -S nossh* starts a screen with the name "nossh" and *attaches* its input and output to the current session. You can now work just like in a normal shell. Additionally, you have the possibility to *detach* from the screen using *CTRL-A D*. Now you're back in the original shell, screen continues to run in the background even after you terminate your SSH session. Using *screen -ls*, you can have all your running screens displayed, *screen -r <screen-name>* reattaches to a specific screen. Further options can be found in the corresponding man page.

Installing Custom Software

UBELIX already comes with a plethora of software pre-installed. It is however, well possible that you will need custom software for your particular work on the cluster. If you think your software may be of general interest for the cluster write us an e-mail otherwise you will have to compile and install the software for yourself.

Most unix software can be installed as follows:

```
tar xzvf some-software-0.1.tar.gz
cd some-software-0.1
./configure --prefix=$HOME
make
make install
```

This would install the software in your home directory, placing binaries for example in *$HOME/bin*. If this is not what you want, you might want to use a different prefix such as *$HOME/opt/some-software*. For you to be able run the software like any other installed binary, you will need to add the new bin directory to your path. Add the following to your *~/.bash_profile*:

```
export PATH="$HOME/bin:$PATH"
```

Make sure to read your software's *INSTALL or README* file to find out about special requirements or custom install procedures that it might need.

Selecting what Software Version to use

Some software - such as the Intel compiler, PGI, Matlab and others - is available as different versions on the cluster.

RedHat uses Environment Modules to provide multiple versions of a specific software. You can see what versions are available by issuing

module avail

To select a specific module, load it via *module load*:

module load intel/12.1
module load pgi/12.5
module load mkl/10.3
module load matlab/R2012a

To see what modules you are currently using, use *module list*, which should look similar to this:

module list
Currently Loaded Modulefiles:
1) intel/12.1 4) matlab/R2012a
2) mkl/10.3 5) namd2/2.9-ibverbs
3) openmpi/1.4.3-intel 6) auto/intel

Please note that some software conflicts with each other. For example it is not possible to load multiple compilers, or multiple versions of the same software at the same time, and trying to do so results in an error.

To load an initial set of software, the auto modules have been specified. they are grouped by compiler. For example to use the intel compiler as default compiler you could use:

module load auto/intel

Please make sure to load the modules inside of your jobs as well, otherwise the jobs will fail as they won't be able find the software on the nodes.

Gaussian License

1. Gaussian, Inc. permits Universität Bern to install, use and modify solely at Universität Bern's places of business specified separately and on computers owned, operated by, and under the supervision of Universität Bern at such address, the Gaussian 09 System of Computer programs, and updates provided by GAUSSIAN, in its sole discretion, the media on which the programs are delivered, and any related documentation, are referred to collectively as "the New Software".

2. Subject to Universität Bern's compliance with the provision of paragraph I hereof, GAUSSIAN grants to Universität Bern a non-exclusive license to install, use and modify the Software, solely at Universität Bern's INSTALLATION ADDRESSES, all in accordance with and subject to the terms and conditions of the Universität Bern, and the term "Software", when and as used in the LICENSE, is amended to include the new software.

3. Universität Bern may not provide access to the Software to third parties. Notwithstanding the above, Universität Bern may allow access to the Software by its faculty, staff and students whose confidentiality obligations include the Software, such as access being subject to all of the restrictions set forth in the LICENSE and this agreement.

4. If the Software is used to obtain a result, and that result is published in the public literature, then Universität Bern agrees to acknowledge its use of the Software in an appropriate citation. The citation should include

a. the name of the product (Gaussian 09)
b. the source (Gaussian, Inc., 340 Quinnipiac Street, Building 40, Wallingford, CT 04692)
c. the authorship as designated by GAUSSIAN; and
d. an appropriate copyright notice as designated by GAUSSIAN

Alternatively, the citation may be made in a form of a reference to a published scientific journal

article as designated by GAUSSIAN. If the Software as modified by Universität Bern is used to obtain a result, and that result is published in the public literature, then Universität Bern will include an appropriate citation as defined above in this paragraph, and Universität Bern will also state in the citation that the Software used was a version of the specified GAUSSIAN Software which was modified by Universität Bern.

5. Universität Bern hereby acknowledges that the Software is to be used for education and academic research purposes and may not be used for commercial purposes. Commercial purposes include, but are not limited to, product development, consulting, or research in ollaboration with commercial entities.

6. Universität Bern acknowledges that the Software is licensed to Universität Bern for use at Universität Bern's INSTALLATION ADDRESSES. The export or re-export of the Software is subject to the requirements of the United States Department of Commerce and/or the Office of Foreign Assets Control of the United States Treasury Department. Any export or re-export of the Software by Universität Bern in violation of the laws of the United States is strictly prohibited. Any user of the Software, as a condition of that use, agrees to comply with all of the provisions hereof.

7. Once executed, this ADDENDUM shall automatically become part of the LICENSE, which constitutes the complete and exclusive statement of the agreement between the parties and supersedes all proposals, oral or written, and all other prior or contemporaneous communications between the parties relating to the subject matter of this Agreement. Only the terms and conditions of this Agreement shall apply to the license of the Software. All terms and conditions of any purchase order or other document issued by Licensee in connection with the Software, shall not apply. This agreement may be amended only by a written agreement executed by Universität Bern and GAUSSIAN. This Addendum is binding upon and will inure to the benefit of the parties hereto and to the extent modified hereby, the LICENSE will remain in full legal force and effect.

Universität Bern | Informatikdienste | Gesellschaftsstrasse 6 | 3012 Bern | helpdesk@id.unibe.ch | Tel +41 (0)31 631 49 99

Further information

- GPFS File System Hints (http://www.id.unibe.ch/content/services/ubelix/how_ubelix_works/gpfs_file_system_hints/)

Support

For questions and problems

- grid-support@id.unibe.ch (mailto:grid-support@id.unibe.ch)

Quick Links

- job monitoring (http://ubelix.unibe.ch/xmlqstat/jobs.html)
- grid-users mailing list (http://listserv.unibe.ch/mailman/listinfo/grid-users)

Quck Links

- Swiss National Grid Association (http://www.swing-grid.ch/)
- Swiss Multi-Science Computing Grid (http://smscg.ch)
- The Swiss HPC Service Provider Community (http://www.hpc-ch.org/)

UNIVERSITÄT BERN

UBELIX FAQ

Contents

General

Software

Job Submission

General

What is UBELIX?

UBELIX is the *University of Bern Linux Cluster*. The cluster is available for tasks related to the work at the University of Bern, e.g. within the scope of a thesis or for scientific research. The cluster must not be used for private purposes. Currently, the cluster is used by various institutes and research groups withing chemistry, biology, physics, astronomy, computer science, geography and others.

How do I use UBELIX?

To log in to UBELIX, you need a Campus Account for the University of Bern. If you work at the University, you Campus Account is most likely the same account you use to check your e-mails. See Getting Started for more information.

How do I log in?

Enter the following in a terminal, where *username* is you username:

ssh username@submit.unibe.ch

I can not log in!

Make sure that you have e-mailed us to activate your account. If you still can not log in, tell us and we will look into it. If you are trying to log in from home, see "I can not log in from home!" below.

I can not log in from home!

The submit host submit.unibe.ch is only available from within the university. To log in from home, you will need to connect to the university first using the university's VPN client (site in German).

How do I get an ssh client for my operating system

If you have a Linux or Mac OS X based system, you most likely already have an ssh client installed. Simply open up a terminal and enter s*sh username@submit.unibe.ch* to log into the cluster.

If you are using Windows, you will want to download the free tool PuTTY and use that to log in to UBELIX.

Do I need to know Linux to use UBELIX?

You definitely should have some basic knowledge how to use a command line interface under Linux. To submit jobs on the cluster you will need to enter shell commands and you might also want to edit your files and scripts directly on the cluster. The good news is, it's not hard to learn. Take a look at this tutorial and make an appointment with us if you are still struggling.

UBELIX? Is that some sort of analogy to the comic Asterix?

Uhmm... uhhh.. of course not!

Software

Do I need to buy licenses to use the software available on UBELIX?

It depends on the software you want to use. Some software is free (ACML, molden, ...) while others are paid software. Intel and PGI compiler suites are paied centrally by the IT Services Department and therefore free to use for you. **Only for the following two applications every user have to have a valid license: Matlab and IDL.** That means that you are not allowed to start the softwarre without a vallid license, which your software coordinator has to buy in the Unibe-Softwareshop!. Please note that usage accounting is done within the UBELIX environment

Job Submission

How do I write a job script?

For a start you can download an example job script. Be sure you have read the this user guide, especially the chapter about Submitting Jobs.

Why does job submission fail with "Unable to run job: error: no suitable queues."?

If the user limit has not been reached, you probably made a mistake with resource allocation. For instance, it's not possible to reqest *-q short.q and h_cpu=10:00:00* at the same time if short.q only offers 1h of CPU time. This is often the case because you make a hard request for one of the queues. Usually it's best to let the system figure out which queue to use.

Please also make sure you made a valid request for the mandatory ressources *h_cpu* and *h_vmem*.

Furthermore this might also be, because the queues are temporarilly suspended or disabled. You can check this with *qstat -f* An **S** behind a queue means suspended, a **d** means disabled.

Also see Resources and Queues.

Why has my job been "pending" for ages?

The queuing information is currently disabled on UBELIX. Please ask us if you are having trouble with your jobs.

Using qstat *-j <job-id>*, SGE tells you why a specific job has not been dispatched to a node yet. If a queue instance is busy, ... *dropped because it is full* will be displayed. If UBELIX is working to full capacity it can take up to several days until a freshly submitted job is executed.

There is a maximum number of jobs per user which can be run simultaneously. The message *job dropped because of user limitations* indicates that the maximum number of jobs possible is already running. Usually, this limit is at about 200 jobs per user.

If there are unused queue instances left and the user limit has not been reached, you probably made a mistake with resource allocation. If all listed queue instances show messages similar to *cannot run in queue instance ... because it offers only ...*, no queue is able to offer all the requested resources. For instance, it's not possible to reqest *-q short.q and h_cpu=10:00:00* at the same time if short.q only offers 1h of CPU time.

Also see Resources and Queues.

Such mistakes with resource allocation can sometimes be corrected using qalter *-j <job-id> ....*

Why does my job have the state "S"?

This indicates that your job was "suspended", meaning it's inactive for the moment but was not terminated and will continue at a later time. This mostly happens if there is a problem with the cluster and job activity must be paused for some time.

Why does my job have the state "Eqw"?

If a job crashed for some reason and had the option "rerunnable" set, it is resubmitted with the error-flag set. If the reason for the error has been found, you can clear the flag using qmod and the job is re-executed.

Why is the software I need not installed?

See Installing Custom Software for information how you can install custom software for yourself.

If you believe a particular software you use would be useful for other users as well, write us an e-mail and let us know.

I get the error message "Disk quota exceeded".

Disk space is limited for all users by a filesystem quota. If you exceed this quota you must either remove some old data or ask for more quota before you can write again new data to the disk. See

Storage for more details.

My application runs fine when started by hand, but why does it crash as soon as I start it within a job?

Be sure you have requested enough resources for the job. See the chapter about Resources and Queues for more information. If you run multithreaded applications you may need to lower the default stack size like *ulimit -s 8192* (8 Megabytes in this example) before you start your application.

How can I access the local scratch?

First you have to request it with *qsub [...] -l scratch=1,scratch_size=200M,scratch_files=100*. Then you can access it within your job script through the environmental variable *$TMP*. See the chapter about Resources and Queues for more information.

My namd jobs run only on a single host even though I submit them via mpirun/mpiexec

The namd installation is compiled directly against the infinband interconnect libraries and does not support mpirun/mpiexec. Instead submit them using charmrun:

charmrun +p${NSLOTS} ++mpiexec /opt/NAMD_2.9_Linux-x86_64-ibverbs/namd2 my_namd_calculation.inp

The *++mpiexec* command is only there to interface with the Grid Engine, but it still needs to be loaded additionally to namd with

module load openmpi/1.4.3-gcc

How can I select what Compiler of which Version of a Software to use?

Please see Selecting what Software Version to use.

I get the message "bash: module: line 1: syntax error: unexpected end of file
bash: error importing function definition for `module'".

This is a cosmetic problem caused by the Grid Engine and can be safely ignored.

Why are my file operations on the global file system so slow?

We use the GPFS file system as global network file system. It has some special characteristics which are described here: GPFS File System Hints.

My job did fail with the message 'kill'. Who did kill my job?

This can have several reasons:

- You or the administrator did delete the job with *qdel*
- You're resource reservations were exceeded (e.g. *h_cpu* is expired) and so the job was killed by the batch system
- The node your job was running on had a problem and needed to restart.

How do I use qlogin on the short.q?

qlogin -l qname=short.q,h_cpu=01:00:00,h_vmem=2G

Universität Bern | Informatikdienste | Gesellschaftsstrasse 6 | 3012 Bern | helpdesk@id.unibe.ch | Tel +41 (0)31 631 49 99

Further information

- GPFS File System Hints (http://www.id.unibe.ch/content/services/ubelix/how_ubelix_works/gpfs_file_system_hints/)

Support

For questions and problems

- grid-support@id.unibe.ch (mailto:grid-support@id.unibe.ch)

Quick Links

- job monitoring (http://ubelix.unibe.ch/xmlqstat/jobs.html)
- grid-users mailing list (http://listserv.unibe.ch/mailman/listinfo/grid-users)

Quck Links

- Swiss National Grid Association (http://www.swing-grid.ch/)
- Swiss Multi-Science Computing Grid (http://smscg.ch)
- The Swiss HPC Service Provider Community (http://www.hpc-ch.org/)

UNIVERSITÄT BERN

GPFS File System Hints

GPFS File System Hints

GPFS http://www-03.ibm.com/systems/software/gpfs/ is a distributed network file system. Its main purpose is to provide a cluster file system with good scalability and a high I/O bandwidth. Unfortunately this goes in costs of access time. As you may have already noticed, especially with a lot of small files in one directry, the user experience with GPFS is quite slow. This affects file meta-data retrieval as well as file creation and access time. Therefore if you want to fully take advantage of this file system, you should respect some basic principles:

- Whenever possible use the local scratch to run your jobs.
- Don't store too many files in one directory. This especially speeds up the 'ls' command.
- Try to avoid extensive use of meta-data operations. Operations like 'find' or 'grep' can take a long time on GPFS.
- Large files are better than small files. If you run your operations on the GPFS file system try to avoid file operations on many small files. Better copy the required files to the local scratch, modify them there and copy them back again after the job run.
- If you have really large input files (>1GB) which are used in several jobs, make copies of it and distribute the read operation over these files. Reason: Every file is stored on one single storage element. There is no file striping between storage elements. If you have to read the file from several jobs in parallel this storage element becomes a bottle neck. If you duplicate the file, the second copy is likely stored on a different storage element and so you already double the available bandwidth for this file.

Example

Here a small comparison how much time you can win when first copying the required files to the local scratch:

Job 1:

Copies an archive with the kernel sources (~45'000 files) to the local scratch, extracts them and compiles the kernel. After compilation the files are packed again and the archive finally moved back to the /home directory:

CPU = 00:09:58
User Time = 00:09:06
System Time = 00:00:52
Wallclock Time = 00:10:59

Job 2:

Extracts the archive in a subfolder of the /home directory, compiles it and packs it again:

CPU = 00:10:45
User Time = 00:9:27
System Time = 00:01:17
Wallclock Time = 00:18:17

When we extract and compile the archive directly on the GPFS file system the job time (i.e. Wallclock Time) is almost double that of job 1. We also see that the CPU, User and System Time are only slightly higher than those of job 1, so the overhead comes indeed from reading and writing to disk.

Job Script

Here an example of a job script using the local scratch. It's used in job 1. Submitted with:

qsub -cwd -l h_cpu=01:00:00,h_vmem=1.5G,scratch=1,

```
scratch_size=2G,scratch_files=60000 kernel-compile.sh
```

```
#!/bin/bash
# kernel-compile.sh - Compile kernel source on local scratch
WORKDIR="linux-2.6.39"
ARCHIVE="${WORKDIR}.tar.bz2"

echo "`date +%c`: starting job 'kernel-compile.sh' ${JOB_ID}"
cp ${ARCHIVE} ${TMP}
cd ${TMP}
echo "`date +%c`: ${JOB_ID} - unpacking kernel source"
tar -xjf ${ARCHIVE}

cd ${WORKDIR}
zcat /proc/config.gz > .config
echo "`date +%c`: ${JOB_ID} - compiling kernel"
make
cd ..

echo "`date +%c`: ${JOB_ID} - packing kernel source"
tar -cjf ${JOB_ID}-${ARCHIVE} ${WORKDIR}
cd ..
mv ${TMP}/${JOB_ID}-${ARCHIVE} ${HOME}/kernel-compile/

echo "`date +%c`: job ${JOB_ID} finished"
exit 0
```

» Download this example as txt file (txt, 925Bytes) (/unibe/verwaltungsdirektion/informatikdienste/content/e5911/e6197/e15895/e15967/e15978/example_skript.txt)

For more infos about the local scratch see: FAQ: How can I access the local scratch?

Universität Bern | Informatikdienste | Gesellschaftsstrasse 6 | 3012 Bern | helpdesk@id.unibe.ch | Tel +41 (0)31 631 49 99

Back

- back to the informations about UBELIX (http://www.id.unibe.ch/content/services/ubelix/how_ubelix_works/)

Support

For questions and problems

- grid-support@id.unibe.ch (mailto:grid-support@id.unibe.ch)

Quick Links

- job monitoring (http://ubelix.unibe.ch/xmlqstat/jobs.html)
- grid-users mailing list (http://listserv.unibe.ch/mailman/listinfo/grid-users)

Quck Links

- Swiss National Grid Association (http://www.swing-grid.ch/)
- Swiss Multi-Science Computing Grid (http://smscg.ch)
- The Swiss HPC Service Provider Community (http://www.hpc-ch.org/)