Nutzung

- Basic Usage (Command Line Interface, Shell Scripts)
- Nutzeranleitung Einsteiger
- Nutzeranleitung Fortgeschrittene
- Queuing-System (SLURM) and Jobfiles
- Using a Visual Node on Windows
- Windows Setup for the Phoenix
- Visual Studio Code

Basic Usage (Command Line Interface, Shell Scripts)

Important info:

Large parts of this script were taken from the documentation of the Hamburg HPC Competence Center (HHCC). Please visit their website for more details on the Use of the Command Line Interface or about Using Shell Scripts.

Description:

- You will learn about the book “The Linux Command Line” by William Shotts which we recommend for learning the command line (basic level)
- You will learn to login remotely (basic level)
- You will learn to use text editors (first steps without the need to consult the book) (basic level)
- You will learn to handle scripting tasks that are often needed in batch scripts: manipulating filenames, temporary files, tracing command execution, error handling, trivial parallelization (basic level)

Use of the Command Line Interface

Users interact with an HPC system through a command line interface (CLI), there are no graphical user interfaces. The command line is used for interactive work and for writing shell scripts that run as batch jobs.

Using a command line interface is a good way to interact very efficiently with a computer by just typing on a keyboard. Fortunately this is especially true for Unix-like operating systems like Linux, because for the typically Linux based cluster systems graphical user interfaces (GUIs) are rarely available. In the Unix or Linux world a program named shell is used as a command language interpreter. The shell executes commands it reads from the keyboard or – more strictly speaking – from the standard input device (e.g. mapped to a file).

The shell can also run programs named shell scripts, e.g. to automate the execution of several commands in a row.

The command line is much more widely used than in HPC. Accordingly, much more has been written about the command line than on HPC. Instead of trying to write yet another text on the

command line we would like to refer the reader to the very nice book The Linux Command Line by William Shotts. The book can be read online. A pdf version (here: 19.01) is also available.

As a motivation to start reading it, we quote a few lines from the Introduction of version 19.01 of the book:

- It's been said that "graphical user interfaces make easy tasks easy, while command line interfaces make difficult tasks possible" and this is still very true today.
- …, there is no shortcut to Linux enlightenment. Learning the command line is challenging and takes real effort. It's not that it's so hard, but rather it's so vast.
- And, unlike many other computer skills, knowledge of the command line is long lasting.
- Another goal is to acquaint you with the Unix way of thinking, which is different from the Windows way of thinking.

The book also introduces the secure shell (ssh). The secure shell is a prerequisite for using HPC systems, because it is needed to log in, and to copy files from and to the system. We also mention the very first steps with text editors.

Remote login

Login nodes

To get access to a cluster system, a terminal emulator program is used to remotely connect to a cluster node (possibly belonging to a group of such nodes) which authenticates the user and grants access to the actual cluster system. Such login nodes are also named gateway nodes or head nodes.

SSH (Secure Shell) and Windows Equivalents

The ssh program provides a secured and encrypted communication for executing commands on a remote machine like building programs and submitting jobs on the cluster system. Windows users can use third-party software like putty or MobaXterm to establish ssh connections to a cluster system. Meanwhile a quite mature OpenSSH port (beta version), i.e. a collection of client/server ssh utilities, is also available for Windows.

Another very useful option is to run Linux (e.g. Ubuntu or openSUSE) on a Windows computer in a virtual machine (VM) using a hypervisor like VMware Workstation Player, VirtualBox, or Hyper-V.

SFTP Clients with Graphical User Interfaces

To make it easier to view files on the server as well as move files between your computer and the server, it possible to use a SFTP Client. SSH File Transfer Protocol (also Secure File Transfer Protocol, or SFTP) is a network protocol that provides file access, file transfer, and file management over any reliable data stream.

There are numerous user-friendly tools providing a GUI, such as WinSCP for Windows or FileZilla for Windows, Linux and Mac OS X (please make sure to decline any possible adware when using the installer).

Other options for more advanced Linux users are SSHFS, a tool that uses SSH to enable mounting of a remote filesystem on a local machine, or the use of the common Gnome Nautilus File Browser. Please visit the links for further information.

Text editors

On an HPC-cluster one typically works in a terminal mode (or text mode in contrast to a graphical mode), i.e. one can use the keyboard but there is neither mouse support nor graphical output. Accordingly, text editors have to be used in text mode as well. For newcomers this is an obstacle. As a workaround, or for editing large portions, one can use the personal computer, where a graphical mode is available, and copy files to the cluster when editing is completed. However, copying files back and forth can be cumbersome if edit cycles are short: for example, if one is testing code on a cluster, where only small changes are necessary, using an editor directly on the cluster is very helpful.

Text user interface

Classic Unix/Linux text editors are vi/vim and GNU Emacs. Both are very powerful but their handling is not intuitive. The least things to know are the key strokes to quit them:

- vi: `<esc>:q!` (quit without saving)
- vi: `<esc>ZZ` (save and quit)
- emacs: `<cntl-x><cntl-c>`

GNU nano is a small text editor that is more intuitive, in particular, because the main control keys are displayed with explanations in two bars at the bottom of the screen. The exit key is:

- nano: `<cntl-x>`.

Graphical user interface

In addition to text user interfaces vim and GNU Emacs have graphical interfaces. On an HPC-cluster it is possible to use graphical interfaces if X11 forwarding is enabled (see -X option of the ssh

command). Two other well know text editors, that you might find on your cluster, are gedit and kate.

However, X11 forwarding can be annoyingly slow if the internet connection is not good enough.

A trick for advanced users is of course again to mount file systems from a cluster with SSHFS. Then one can transparently use one's favourite text editor on the local computer for editing files on the remote cluster.

Using shell scripts

Shell scripts are used to store complicated commands or to automate tasks. A simple script consists of one or a few commands that appear exactly like on the interactive command line. Since the shell is also a programming language, scripts can execute more complicated processes. On HPC systems scripting is needed for creating batch jobs. Here, some scripting tasks are explained that are useful for writing batch scripts.

Manipulating filenames

Handling filenames translates to character string processing. The following table shows some typical examples:

action	command	result
initialization	a=foo	a=foo
	b=bar	b=bar
concatenation	c=$a/$b.c	c=foo/bar.c
	d=${a}_$b.c	d=foo_bar.c
get directory	dir=$(dirname $c)	dir=foo
get filename	file=$(basename $c)	file=bar.c
remove suffix	name=$(basename $c .c)	name=bar
	name=${file%.c}	name=bar
remove prefix	ext=${file##*.}	ext=c

Recommendation: Never use white space in filenames! This is error prone, because quoting becomes necessary, like in: `dir=$(dirname „$c“)`.

Temporary files

There are three issues with temporary files: choice of the directory to write them, unique names and automatic deletion.

Assume that a batch job shall work in a temporary directory. Possibly, the computing center provides such a directory for every batch job and deletes that directory when the jobs ends. Then one can just use that directory. If this is not the case one can proceed like this.

- Choose a file system (or top directory) to work in. The classic directory for this purpose is `/tmp`. However, this is not a good choice on (almost all) HPC clusters, because `/tmp` is probably too small on diskless nodes. You should find out for your system, which file system is well suited. There can be local file systems (on nodes that are equipped with local disks) or global file systems. Let us call that filesystem `/scratch` and set:

```
    top_tmpdir=/scratch
```

- A sub-directory with a unique name can be generated with the `mktemp` command. `mktemp` generates a unique name from a template by replacing a sequence of `X`s by a unique value. It prints the unique name such that it can be stored in a variable. For easy identification of your temporary directories you can use your username (which is contained in the variable `$USER`) in the template, and set:

```
    my_tmpdir=$(mktemp -d "$top_tmpdir/$USER.XXXXXXXXXXX")
```

- The next line in our example handles automatic deletion. Wherever the script exits our temporary directory will be deleted:

```
    trap "rm -rf $my_tmpdir" EXIT
```

- Now we can work in our temporary directory:

```
    cd $my_tmpdir
    ...
```

Tracing command execution

There are two shell settings for tracing command execution. After `set -v` all commands are printed as they appear literally in the script. After `set -x` commands are printed as they are being executed (i.e. with variables expanded). Both settings are also useful for debugging.

Error handling

There are two shell settings that can help to handle errors.

The first setting is `set -e` which makes the script exit immediately if a command exits with an error (non-zero) status. The second setting is `set -u` which makes the script exit if an undefined variable is accessed (this is also useful for debugging).

Exceptions can be handled in the following ways. If `-e` is set an error status can be ignored by using the *or* operator `||` and calling the `true`command which is a no-op command that always

succeeds:

```
command_that_could_go_wrong || true
```

If `-u` is set a null value can be used if variable that is unset is accessed (the braces and the dash after `_set` do the job):

```
if [[ ${variable_that_might_not_be_set-} = test_value ]]
then
    ...
fi
```

Information about the filesystems

Phoenix has two different filesystems: /home and /work.

When you connect to Phoenix, the current dictionary is /home/username.
The /home-folder is a Network File System, a backup is performed every day.
Each user has a quota of 100 GB, to check how much of that is occupied you can use the command:

```
quota -vs
```

Each user also has another folder: /work/username.
The /work-folder is a BeeGFS, a filesystem developed and optimized for high-performance computing.
For all your calculations you should always read from and write to your /work-folder, because in comparison to the /home-folder it doesnot have problems with multiple nodes writing to the filesystem. There are no backups for the /work-folder, so you should copy important information after the calculations to the /home-folder.
The quota of the /work-folder is 5 TB, to check how much is occupied you can use the command:

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

File transfer from your local system to the Phoenix cluster

Files can be transferred from your local system to the Phoenix remote system in a number of ways. There are two common options: `scp` and `rsync`. In general `rsync` is the better option.

One option is to use the `Secure Copy Protocoll (SCP)`. To use this, you have to open a terminal on your local system. The following command copies the file myfile from your local system to a remote directory on the Phoenix:

```
scp path/to/myfile username@phoenix.hlr.rz.tu-bs.de:/path/to/destinationfolder
```

For copying a folder, `-r` has to be included:

```
scp path/to/myfolder -r username@phoenix.hlr.rz.tu-bs.de:/path/to/destinationfolder
```

If the target file does not yet exist, an empty file with the target file name is created and then filled with the source file contents. If copying a source file to a target file which already exists, scp will replace the whole contents of the target file; this will take longer then needed and in this case we recommend the usage of rsync.

`Rsync` is another option. Rsync can synchronize files and directories, it only transfers the differences between the source and the destination. Therefore it is recommended to use rsync for files and folders that already exist on the Phoenix but changed on your local system. It is also recommended when you tried to copy something to the Phoenix but due to a loss of internet connection, it did not finish. In this case only a part of the files are transferred, to copy only the missing files, using `rsync` is the best choice.

```
rsync -avz path/to/file username@phoenix.hlr.rz.tu-bs.de:/path/to/destinationfolder
```

For both ways there is a variety of flags that can be looked up in the corresponding man pages.

File transfer from remote system to the Phoenix cluster

To transfer files from a remote system, it is necessary that you login via ssh on the remote server where the files are located (e.g. a server of the institute). Then start a `screen session` with the command:

```
screen
```

Then it is possible to just use the commands described above. The advantage of launching the copying-process in a screen session is, that you can disconnect from the remote server and the process will still be running. If you don't use a screen session and the SSH session to the remote server terminates (e.g. you disconnect yourself or the internet connection drops), then the copying-process will be terminated as well.

Nutzeranleitung Einsteiger

Nutzeranleitung Einsteiger

English Version see below

Login

Um auf dem Phoenix-Cluster zu arbeiten, sind einige wenige Grundkenntnisse mit Linux-System erforderlich. Zur Arbeit auf dem Cluster stellen Sie eine SSH-Verbindung zu einem Loginknoten unter `username@phoenix.hlr.rz.tu-bs.de` her. Dabei ist `username` Ihr Benutzername, den Sie für alle Dienste des Gauß-IT-Zentrum nutzen; das Passwort ist das dazu gehörige Passwort.

Falls Sie sich noch nicht für die Nutzung des Phoenix registriert haben, können Sie sich an phoenix-support@tu-bs.de wenden.

```
ssh username@phoenix.hlr.rz.tu-bs.de
```

Für die Verwendung graphischer Anwendungen ist eine X-Weiterleitung mittels des Parameters „-Y“ (nicht „-X“) nötig.

```
ssh -Y username@phoenix.hlr.rz.tu-bs.de
```

Um sich abzumelden, geben Sie einfach den Befehl `exit` ein.

Der Fingerprint zum Login Server lautet:

```
phoenix.hlr.rz.tu-bs.de ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDFDm9AGYEiUIZ6wBwXHoO6YmrMD/QKieMZ
phoenix.hlr.rz.tu-bs.de ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABE
phoenix.hlr.rz.tu-bs.de ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAII8Vj77eCWfqTLHNyuW/vvvfALIhd8oTDQX
```

Jobs starten

Wenn Sie mit einem Loginknoten verbunden sind, gibt es zwei Möglichkeiten Jobs zu starten: Direkt per Jobfile (dies ist die übliche Vorgehensweise) oder per Allokierung („Reservierung“) von Ressourcen mithilfe eines Jobfiles, um interaktiv auf einzelnen Knoten zu arbeiten.

Jobfile

Ein Jobfile bzw. Jobscript sieht beispielsweise wie folgt aus:

```
#!/bin/bash -l

#SBATCH --partition=standard
#SBATCH --nodes=2
#SBATCH --time=12:00:00
#SBATCH --job-name=nearest
#SBATCH --ntasks-per-node=20


working_dir=~
cd $working_dir
module load comp/gcc/6.3.0
module load mpi/openmpi/2.1.0/gcc
mpiexec -np 40 ./test/mpinearest
```

`partition` gibt die Partition (auch Queue genannt) an, auf der Sie rechnen wollen.
`time` gibt die maximale Laufzeit Ihres Jobs an, bevor dieser dann abgebrochen wird.
`job-name` gibt einen Namen an, unter dem Sie Ihren Job später in SLURM wieder finden können.

Eine Startzeit kann mit dem Schalter `–begin` vorgegeben werden. Beispielsweise:

```
--begin=16:00
--begin=now+1hour
--begin=now+60 (seconds by default)
--begin=2010-01-20T12:34:00
```

Bei dem Phoenix-Cluster gibt es verschiedene Arten von Knoten, die genutzt werden können. Die in den Knoten verbaute Hardware kann eingesehen werden, wenn auf die jeweilige Knotenart in der Tabelle geklickt wird.
Diese sind zudem in folgende verschiedene Partitionen aufgeteilt:

Partition	Genutze Knoten	RAM	GPU	Maximale Knotenanzahl/Job	Walltime	Shared
standard	node[001-304]	64 GB	-	20	7 d	Ja
shortrun_small	node[001-304]	64 GB	-	50	3 d	Ja
shortrun_large	node[001-304]	64 GB	-	150	3 d	Ja

Partition	Genutze Knoten	RAM	GPU	Maximale Knotenanzahl/Job	Walltime	Shared
longrun	node[001-304]	64 GB	-	4	14 d	Ja
testing	node[001-304]	64 GB	-	150	1 h	Ja
fat	fat[001-008]	256 GB	-	8	7 d	Nein
vis	vis[01-06]	512 GB	-	6	2 d	Nein
gpu01_queue	gpu[01-02]	64 GB	4x NVIDIA Tesla 16GB	2	3 d	Nein
gpu02_queue	gpu[03-08]	64 GB	4x NVIDIA Tesla 16GB	6	7 d	Nein
gpu03_queue	gpu[01-08]	64 GB	4x NVIDIA Tesla 16GB	8	7 d	Nein

Ein Beispiel für ein Jobfile für einen der GPU-Knoten sieht wie folgt aus:

```
#!/bin/bash -l

#SBATCH --partition=gpu02_queue
#SBATCH --nodes=2
#SBATCH --time=2-00:00:00
#SBATCH --job-name=GPUExample
#SBATCH --ntasks-per-node=4
#SBATCH --gres=gpu:4

~/anaconda3/bin/python "/path/to/file/example.py"
```

Ein Jobfile muss dann mittels `sbatch` gestartet werden.

```
username@login01 [~] sbatch jobname.job
```

Weitere Informationen bietet auch `man sbatch`. Sämtliche dort verwendeten Parameter können auch im Jobscript selber mit `#SBATCH` angeben werden.

Interaktive Anwendungen

Wenn Sie sich Knoten reservieren wollen, um interaktiv auf diesen zu arbeiten, ist dies mit einem simplen Jobfile möglich:

```
#!/bin/bash -l

#SBATCH --partition=standard
#SBATCH --nodes=1
#SBATCH --time=7-00:00:00
#SBATCH --job-name=SleepExample
#SBATCH --ntasks-per-node=1

cd ~/data_dir
sleep 168h
```

Bei der Ausführung per `sbatch` wird dann ein Knoten für die angegebene Zeit benutzt. Da ein Job läuft, kann auf diesen vom Besitzer per SSH zugegriffen werden. Um den richtigen Knoten zu finden, verwenden Sie `squeue` und suchen nach dem angegebenen Jobnamen. Danach aktivieren Sie den Sitzungsmanager mit `screen`.

```
username@login01 [~] sbatch jobname.job
[Knoten finden]
username@login01 [~] ssh node265
username@node265 [~] screen
username@node265 [~] ...
```

Nachdem Sie nun eine Anwendung gestartet haben, können Sie die Sitzung mit `Strg + A + D` in den Hintergrund verschieben. Wichtig ist dabei, dass Sie vorher `screen` eingegeben haben, um den Sitzungsmanager zu starten.

Eine in den Hintergrund verschobene Sitzung wird nicht geschlossen, wenn Sie die SSH-Verbindung trennen. Damit laufen Ihre Prozesse in diesen Sitzungen entsprechend weiter, andernfalls würden Sie beendet werden!

Um eine einzelne Sitzung wieder aufzunehmen, verwenden Sie auf dem entsprechenden Knoten `screen -r`. Sollten mehrere Sitzungen im Hintergrund offen sein, wird eine Liste angezeigt, aus der Sie spezifizieren müssen:

```
username@node265 [~] screen -r

    154577.pts-0.node265    (Detached)
    154308.pts-0.node265    (Detached)
    153240.pts-0.node265    (Detached)
Type "screen [-d] -r [pid.]tty.host" to resume one of them.
username@node265 [~] screen -r 153240.pts-0.node265
```

Diese Sitzung lässt sich danach mit Eingabe von `screen -d` wieder in den Hintergrund verschieben.

Grafische Anwendungen via VNC nutzen

Mit Hilfe eines VNC (Virtual Network Computing) kann der Bildschirminhalt des Phoenix auf dem eigenen lokalen Rechner in einem Fenster angezeigt werden. Das Bildschirmfenster kann dann wie ein lokales Fenster bedient werden und Tastatur- und Mausbewegungen des eigenen lokalen

Rechners werden an den Phoenix gesendet.

Dazu müssen Sie sich wie üblich beim Phoenix anmelden und einen interaktiven Job in der vis Partition (Queue) starten, ein entsprechendes Job-File könnte so aussehen:

```
#!/bin/bash -l

#SBATCH --partition=vis
#SBATCH --nodes=1
#SBATCH --time=1:00:00
#SBATCH --job-name=nearest
#SBATCH --ntasks-per-node=20

sleep 1h
```

Das Job-File blockt einen vis-Knoten für eine Stunde, in dieser Zeit kann dann via VNC auf dem Phoenix gearbeitet werden.
Als nächsten Schritt muss sich auf dem zugewiesenen vis-Knoten eingeloggt werden:

```
ssh vis0X wobei X der zugewiesene vis-Knoten ist, diesen kann man mit "squeue -u $USER" herausfi
```

Falls noch nie vorher VNC genutzt wurde, muss ein VNC Passwort festgelegt werden, dies geht mit Hilfe des Befehls:

```
/cluster/share/vnc/setup-vnc.sh
```

Bei vergessenem Passwort kann mit dem selben Befehl auch ganz einfach ein neues Passwort festgelegt werden.
Daraufhin muss ein VNC-Server mit dem folgenden Befehl gestartet werden:

```
vncserver
```

Oder für eine bessere Auflösung kann folgender Befehl genutzt werden:

```
vncserver -geometry 1280x1024
```

In einem weiteren Terminal muss nun auf dem eigenen lokalen Rechner ein VNC-Client geöffnet werden. Wir empfehlen **remmina**, bei Windows muss zusätzlich **Xming** installiert und gestartet werden. Den VNC-Client remmina wird über den Befehl gestartet:

```
remmina
```

Eventuell muss vorher, falls remmina nicht startet, der folgende Befehl ausgeführt werden:

```
export DISPLAY=:0
```

Bei remmina muss mit Hilfe des Knopfes in der oberen linken Ecke ein neues Verbindungsprofil erstellt werden mit folgenden Angaben:

- [**Protocol]:** Remmina VNC Plugin
- unter dem Reiter [**Basic]**:
  - **[Server]**: vis0X:Y, wobei X der zugewiesene vis-Knoten ist und Y die Displaynummer des VNC-Servers, diese ist Teil der Ausgabe nach dem Befehl „vncserver“
  - **[Color depth]**: High color (16 bpp)
  - **[Quality]**: Medium

- unter den Reitern **[Advanced]** und **[Autostart]**:
  - keine Änderungen

- unter dem Reiter **[SSH Tunnel]**:
  - **[Enable SSH Tunnel]** auswählen
  - **[Custom]** auswählen und eingeben: phoenix.hlr.rz.tu-bs.de
  - **[Username]**: eigenen Username eingeben
  - Anmeldung per **[Passwort]** auswählen

Daraufhin das Verbindungsprofil mit dem **[Save]**-Knopf speichern und mit dem **[Connect]**-Knopf verbinden. Es wird dann nach dem eigenen Passwort für den Phoenix gefragt und nach Eingabe des eingenen Passworts nach dem VNC-Passwort, das in setpup-vnc.sh Schritt festgelegt wurde.

In dem Fenster kann dann ein Terminal geöffnet werden. Zur Nutzung einer Software im visuellen und interaktiven Modus muss das entsprechende Modul geladen werden. Wenn die Grafikkarte, also zur Visualisierung genutzt werden soll, muss vor dem entsprechenden Terminalbefehl zur Öffnung der Software noch **vglrun** hinzugefügt werden. Um ParaView im interaktiven Modus zu nutzen, muss also nach dem Laden des Moduls folgender Terminalbefehl ausgeführt werden:

```
vglrun paraview
```

Nachdem Sie fertig mit der Nutzung sind, muss der VNC-Server auf dem Phoenix beendet werden, dies geht mit dem Befehl:

```
vncserver -kill :Y wobei Y für die Displaynummer steht
```

Wird der VNC-Server nicht geschlossen, sollte es beim nächsten Einloggen getan werden, ansonsten verschiebt sich die Displaynummer (:1 wird beispielsweise zu :2).

Jobs überwachen

Sie können sich mit dem Befehlt `squeue` eine Liste aller Jobs in SLURM anzeigen lassen. Mehr Informationen gibt es mit `squeue -l`.

```
username@login01 [~]squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               333  standard MD_BA_GA user0001  R      58:56      1 node265
               336  standard     bash username  R       1:12      2 node267,node268
               334  standard     bash user0002  R      50:16      1 node266
```

```
               331       vis      vis user0003  R    1:25:22      1 vis01
               329  standard    name1 user0003  PD     00:00      1 (Resources)
```

`JOBID` ist die ID der Jobs, um diese in SLURM ansprechen zu können. `NAME` ist der von Ihnen gewählte Name zur leichteren Identifikation des Jobs. `ST` gibt den Status des Jobs an (`R` = Running, `PD` = Wartend). `TIME` beschreibt die Zeit, die der Job bereits läuft. Sollte der Job nicht laufen (`PD`), wird unter `NODELIST` statt der Knoten eine knappe Begründung angegeben, warum der Job (noch) nicht läuft.

Jobs abbrechen

Mit der ID des Jobs lässt sich ein Job mit `scancel` abbrechen:

```
username@login01 [~]scancel 336
salloc: Job allocation 336 has been revoked.
```

Modulsystem

Auf dem Phoenix ist das Environment Modules System (kurz Modules) installiert.

Module verändern/erweitern Pfade und Variablen, um Programme und Bibliotheken von anderen Orten zu laden. Jede auf dem Phoenix installierte Software hat ein Modulfile, welches die beim Laden die nötigen Variablen setzt, die für den Einsatz der Software erforderlich sind.

Verfügbare Module anzeigen

Eine Übersicht aller verfügbaren Module erhält man mittels

```
module avail
```

Die aktuell geladenen Module sieht man mit

```
module list
```

Auf diese Weise erkennen Sie auch, welche Software auf dem Phoenix verfügbar ist und in welchen Versionen diese vorliegen.

Module laden und unloaden

Module können mittels

```
module load MODULNAME
```

geladen werden. Es ist auch möglich, mehrere Module auf einmal zu laden. Diese werden dann getrennt durch Leerzeichen eingegeben.

Entfernt werden die Module dann durch

```
module remove MODULNAME
```

Hier ist es ebenfalls möglich, mehrere Module gleichzeitig zu entfernen.

Alle Module auf einmal entfernen mittels

```
module purge
```

Befehle / Aliase automatisch beim Anmelden ausführen

Die Datei /home/username/.bash_profile verhält sich wie die .bashrc auf den meisten anderen Linux Distributionen. Befehle die dort eingetragen werden, werden automatisch beim Anmelden ausgeführt. Zum Editieren kann zum Beispiel nano benutzt werden: nano .bash_profile

Abaqus Job starten

Um Abaqus mit Slurm und MPI kompatibel zu machen, müssen einige Befehle im Jobfile abgearbeitet werden. Hier ist ein Beispiel Jobfile.

```
#!/bin/bash -l

#SBATCH --partition=standard
#SBATCH --nodes=1
#SBATCH --job-name=dingALE
#SBATCH --ntasks-per-node=20
#SBATCH --time=48:00:00
#SBATCH -o bo-%j.log


module purge
module load software/abaqus/abaqus_2016


input_file=dingALE.inp

working_dir=~/DingALE
cd $working_dir

### Create ABAQUS environment file for current job, you can set/add your own options (Python syn
env_file=custom_v6.env


##############################################################################
```

```
cat << EOF > ${env_file}
mp_file_system = (DETECT,DETECT)
EOF

node_list=$(scontrol show hostname ${SLURM_NODELIST} | sort -u)

mp_host_list="["
for host in ${node_list}; do
    mp_host_list="${mp_host_list}['$host', ${SLURM_CPUS_ON_NODE}],"
done

mp_host_list=$(echo ${mp_host_list} | sed -e "s/,$/]/")

echo "mp_host_list=${mp_host_list}"  >> ${env_file}


### Set input file and job (file prefix) name here
job_name=${SLURM_JOB_NAME}


### ABAQUS parallel execution
abaqus job=${job_name} input=${input_file} cpus=${SLURM_NTASKS} standard_parallel=all mp_mode=mp
```

Falls mehr RAM benötigt wird, kann alternativ auch die *fat*-Partition genutzt werden.

Ansys Job starten

Auf das vorhandene Script (Beispiel.sh) mit dem SBATCH und den auszuführenden Befehlen das Script /cluster/share/make_ansys_job_check_for_license.sh anwenden und dazu Lizenzenanzahl mit Lizenzart angeben. Das Skript hängt an den auszuführenden Skript die Lizenzcheck und zusätzlich benötigte Variablen mit an. Ausgegeben wird das Skript in der Standardausgabe, kann also mit Hilfe der Ein-/Ausgabe Umleitung von Linux in eine Datei mit dem Operator „>“ geschrieben werden.

Beispiel für eine normale Ansys CFX Datei, die noch keinen Lizenzcheck besitzt:

```
#!/bin/bash -l

#SBATCH --partition=standard
#SBATCH --nodes=2
#SBATCH --time=3:00:00
#SBATCH --job-name=TCTM
#SBATCH --ntasks-per-node=20

module load software/ansys/18.0


######################
#### HIER VARIABEL ####
######################
```

```
NUMPROCS=20
module load software/ansys/18.0

cfx5solve -batch -chdir $working_dir -double -verbose -def $working_dir/Heater_pt_Loss.def -star
```

Beispiel für eine normale Ansys Fluent Datei, die noch keinen Lizenzcheck besitzt:

```
#!/bin/bash

#SBATCH --partition=standard
#SBATCH --nodes=3
#SBATCH --time=3:00:00
#SBATCH --job-name=ansys_flu
#SBATCH --ntasks-per-node=20
#SBATCH --exclusive

# load module fluent v19.2
module load software/ansys/19.2

# The Journal file
JOURNALFILE=journalFile.jou

# Total number of Processors. Nodes * 20
NPROCS=60

# MPI key, dont change!
export MPI_IB_PKEY=0x8001

fluent 3ddp -g -t $NPROCS -slurm -i $JOURNALFILE > fluent.out
```

Syntax:

```
/cluster/share/make_ansys_job_check_for_license.sh Script.sh NumberOfLicences1 LicenceType1 Numb
```

Beispiel:

```
/cluster/share/make_ansys_job_check_for_license.sh Beispiel.sh 60 ansys > BeispielMitLizenzCheck
```

Wenn Sie nicht wissen, welcher Lizenztyp genutzt werden soll, wird empfohlen, den Typ **ansys** zu nutzen.

Der Lizenzcheck funktioniert so, dass vorher die angegebene Anzahl an Lizenzen geprüft werden und je nachdem das Programm weiterlaufen kann oder nach einer Stunde der Check nochmal durchgeführt wird, wenn nicht genug Lizenzen vorhanden sein sollten.

Ansys für mechanische Simulationen

```
#!/bin/bash -l
#
### Grosse Knoten mit 256GB RAM (fat) SBATCH --partition=fat
```

```
### Kleinere Knoten mit 64GB RAM (standard) SBATCH --partition=standard
### Die Anzahl der Kerne ist bei beiden Knotentypen (fat und standard) gleich, naemlich 20
### Es gibt insgesamt 8 Knoten (fat) mit jeweils 256GB RAM
### ####### Nehmt bitte grundsaetzlich immer zuerst die Standardknoten !!!!! #######
###
### Variable NUMPROC = (#SBATCH --nodes=3) x (SBATCH --ntasks-per-node=20) = 60

#SBATCH --partition=standard
#SBATCH --nodes=1
#SBATCH --time=10:00:00
#SBATCH --job-name=Tensiletest
#SBATCH --ntasks-per-node=20


######################
#### HIER VARIABEL ####
######################
export working_dir=/beegfs/work/y0090888/cfx
#################
NUMPROCS=20
#################
cd $working_dir

export TMI_CONFIG=/cluster/tools/ansys_2018/ansys_inc/v182/commonfiles/MPI/Intel/5.1.3.223/linx6
export I_MPI_FABRICS=shm:tmi
export I_MPI_FABRICS_LIST=tmi
export I_MPI_FALLBACK=0
export I_MPI_TMI_PROVIDER=psm2
#export I_MPI_DEBUG=5

module load software/ansys/19.2


# Befehle:
# Ausführung der Rechnung im Arbeitsverzeichnis: cfx5solve -batch -chdir $working_dir -single -v
# Def-File: -def "filename.def"
# kein Plan was die alle machen (diverse Clusterbefehle), war copy-paste: -start-method 'Intel M
# Benennung des res-Files: -fullname "filename"
# andere Ergebnisdatei als Initialisierung verwenden: -cont-from-file "filename.res"
# Vorgabe eines ccl-Files für Änderungen im def-File (z.B. andere Randbedingung, Druck, etc.): -
#
### komplettes Bsp:cfx5solve -batch -chdir $working_dir -single -verbose -def V3_4_closed.def -c

ansys192 -B -batch -chdir $working_dir -single -verbose -i tensiletest.dat \ -start-method 'Inte
```

> Der folgende Legacy Code ist veraltet. Er sollte nicht benutzt werden, außer Sie haben sehr gute Gründe dafür.

Alte Methode:

Es müssen die Dateien CFX.sh und nodes2ansys.py angelegt werden. Dann kann ein Ansys Job via sbatch CFX.sh gestartet werden. StaticMixer.def sollte durch die eigene Datei ersetzt werden.

CFX.sh:

```
#!/bin/bash -l
# export PATH=/cluster/tools/ansys_inc/v180/CFX/tools/multiport/mpi/lnamd64/intel/bin:$PATH
# export PATH=/cluster/tools/ansys_inc/v180/commonfiles/MPI/Intel/5.1.3.223/linx64:$PATH

#SBATCH --partition=standard
#SBATCH --nodes=2
#SBATCH --time=3:00:00
#SBATCH --job-name=Ansys
#SBATCH --ntasks-per-node=20
#

#######################
#### HIER VARIABEL ####
#######################
export working_dir=$HOME/ansys
# export ALLMACHINES=$(scontrol show hostname $SLURM_JOB_NODELIST |paste -d, -s )
/usr/bin/python nodes2ansys.py
export ALLMACHINES=`cat $HOME/ansys/cfx_nodefile.dat`
echo $SLURM_JOB_NODELIST>$HOME/ansys/slurmjoblist
echo $ALLMACHINES>$HOME/ansys/machines
#################
NUMPROCS=40
#################
cd $working_dir

export TMI_CONFIG=/cluster/tools/ansys_inc/v180/commonfiles/MPI/Intel/5.1.3.223/linx64/etc/tmi.c
export I_MPI_FABRICS=shm:tmi
export I_MPI_FABRICS_LIST=tmi
export I_MPI_FALLBACK=0
export I_MPI_TMI_PROVIDER=psm2

module load software/ansys/18.0


cfx5solve -batch -chdir $working_dir -double -verbose -def $working_dir/StaticMixer.def -start-m
```

nodes2ansys.py:

```
#!/usr/bin/python

import os

# READ SLURM JOB LIST

#line = example data: fat[1-3]
line = os.environ['SLURM_JOB_NODELIST']
```

```
#line = 'fat[001-003]\n'

line.strip()

file = open( "cfx_nodefile.dat" , "w" )

#name = fat , numbers = 001-003
name,numbers = line.split('[')
numbers = numbers.replace(']', '')

#001-003 -> 001 003
start,end = numbers.split('-')

#001 003 -> 1 3
start_int = int(start)
end_int = int(end)

#from 1 to 3 do:
for number in range(start_int, end_int+1):
    number_string = str(number)

    #add the leading zeroes to node-number 2 -> 002
    while len(number_string) < len(start):
        number_string = "0" + number_string

    #write down the nodename like fat002 20 times... you could do this most likely with ntasks-p
    for i in range (1,21):
        file.write(name + str(number_string) + ',' )

file.close
```

Lehre

Für die Lehre existiert eine Queue: teaching und in Zukunft teaching_gpu. Diese Knoten verfügen nicht über eine funktionierende Infinity Band Verbindung, sondern nutzen klassisches Ethernet für die MPI Kommunikation. Daher muss in der Jobfile angegeben werden, dass Ethernet genutzt werden soll.

```
NUMBER_OF_CPU_CORES_TO_USE=16
mpirun -n $NUMBER_OF_CPU_CORES_TO_USE --mca btl tcp,sm,self --mca btl_tcp_if_include eth0  /home
```

Beispiel Jobfile:

```
#!/bin/bash -l

#SBATCH --partition=visalt
#SBATCH --nodes=2
#SBATCH --time=1:00:00
#SBATCH --job-name=nearest
```

```
#SBATCH --ntasks-per-node=8
#SBATCH --exclude=visalt01

module load mpi/openmpi/2.1.0/gcc comp/gcc/6.3.0
mpirun -n 16  --mca btl tcp,sm,self --mca btl_tcp_if_include eth0  /home/raskrato/alt/visalt/pte
```

Users Guide Beginners (English Version)

Login

For your work with the Phoenix cluster you will need a little experience with Linux systems. At first you need to connect via SSH to `username@phoenix.hlr.rz.tu-bs.de`. Where `username` is your account name for all services of the Gauß-IT-Zentrum with the same password.

In case you aren't registered yet, you can contact phoenix-support@tu-bs.de.

```
ssh username@phoenix.hlr.rz.tu-bs.de
```

If you need to work with graphical interfaces, you need to use trusted X11-forwarding with the parameter „-Y“ (instead of „-X“).

```
ssh -Y username@phoenix.hlr.rz.tu-bs.de
```

To log out you can use the command `exit`.

The Fingerprint to the login server is:

```
phoenix.hlr.rz.tu-bs.de ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDFDm9AGYEiUIZ6wBwXHoO6YmrMD/QKieMZ
phoenix.hlr.rz.tu-bs.de ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABE
phoenix.hlr.rz.tu-bs.de ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAII8Vj77eCWfqTLHNyuW/vvvfALIhd8oTDQX
```

Starting jobs

When you are connected to one of the login nodes, there are two ways to start jobs: With a jobfile (usual approach) or by allocating resources to interactively work on nodes.

Jobfile

For example a jobfile (or jobscript) can look like this:

```
#!/bin/bash -l

#SBATCH --partition=standard
#SBATCH --nodes=2
#SBATCH --time=12:00:00
#SBATCH --job-name=nearest
#SBATCH --ntasks-per-node=20


working_dir=~
cd $working_dir
module load comp/gcc/6.3.0
module load mpi/openmpi/2.1.0/gcc
mpiexec -np 40 ./test/mpinearest
```

`partition` determines the partition (queue) you want to use. `time` is the maximum time your job will run before being cancelled. `job-name` is the name for your job to identify in the queuing system.

The jobfile then has to be started with `sbatch`.

```
username@login01 [~] sbatch jobname.job
```

A starting time can be given with `–begin`.

```
--begin=16:00
--begin=now+1hour
--begin=now+60 (seconds by default)
--begin=2010-01-20T12:34:00
```

There are in total eight GPU-nodes. For these there are three GPU queues with different workloads:

- gpu01_queue:

- 2 GPU-nodes used for this queue
- high priority
- just for jobs, that allocate all four GPUs of a node
- a maximum of one job per user
- walltime: 3 days

- gpu02_queue:

- 6 GPU-nodes used
- high priority
- also for jobs, that just allocate a single GPU of a node
- a maximum of two job per user

- walltime: 7 days

- gpu03_queue:

- 8 GPU-nodes used
- low priority
- also for jobs, that just allocate a single GPU of a node
- number of jubs per user is unlimited
- walltime: 7 days
- jobs can be paused, therefore use checkpoints in your code

In the following you can find an example for a jobfile for one of the GPU-node:

```
#!/bin/bash -l

#SBATCH --partition=gpu02_queue
#SBATCH --nodes=2
#SBATCH --time=2-00:00:00
#SBATCH --job-name=GPUExample
#SBATCH --ntasks-per-node=4
#SBATCH --gres=gpu:4  

~/anaconda3/bin/python "/path/to/file/example.py"
```

You can find further information with `man sbatch`. Any parameters found there can also instead be included in the jobscript with `#SBATCH`.

Interactive jobs

To allocate a node for interactive usage you need a simple jobfile:

```
#!/bin/bash -l

#SBATCH --partition=standard
#SBATCH --nodes=1
#SBATCH --time=7-00:00:00
#SBATCH --job-name=SleepExample
#SBATCH --ntasks-per-node=1

cd ~/data_dir
sleep 168h
```

When executing this file with `sbatch` a node will be used for the given time. Since a job is running the user can connect to the node via SSH. To find the node allocated for you, use `squeue` and look for your given job name. After connecting to the node you will have to activate the session manager with `screen`.

```
username@login01 [~] sbatch jobname.job
[Knoten finden]
```

```
username@login01 [~] ssh node265
username@node265 [~] screen
username@node265 [~] ...
```

After starting an application or processes on the node you can press `Crtl + A + D` to move the session to the background.

A session that was moved to the background will not be closed when you disconnect from the node. Processes and application will continue to run in this session.

The get back into sessions use `screen -r`. If you had multiple sessions moved to the background this will open a list from which you can specify:

```
username@node265 [~] screen -r

    154577.pts-0.node265    (Detached)
    154308.pts-0.node265    (Detached)
    153240.pts-0.node265    (Detached)
Type "screen [-d] -r [pid.]tty.host" to resume one of them.
username@node265 [~] screen -r 153240.pts-0.node265
```

A session can be moved to the background again with `screen -d`.

Use graphical applications via VNC

With the help of a VNC (Virtual Network Computing), the screen content of the Phoenix can be displayed in a window on your own local computer. The window can then be operated like a local window. Keyboard and mouse movements of the local computer are sent to the Phoenix.

Therefore, you have to log on to the Phoenix as usual and start an interactive job in the vis partition (queue). A corresponding job file could look like this:

```
#!/bin/bash -l

#SBATCH --partition=vis
#SBATCH --nodes=1
#SBATCH --time=12:00:00
#SBATCH --job-name=nearest
#SBATCH --ntasks-per-node=20

sleep 1h
```

The job file blocks a vis node for 1 hour, during this time you can then work on the Phoenix via VNC.

The next step is to log in to the assigned vis node:

```
ssh vis0X whereby X is the assigned vis node, you can find this out with "squeue -u $USER".
```

If you have never used VNC before, you must set a VNC password, this can be done with the command:

```
/cluster/share/vnc/setup-vnc.sh
```

If you forget your password, you can easily set a new password with the same command.
A VNC server must then be started with the following command:

```
vncserver
```

Or for a better resolution the following command can be used:

```
vncserver -geometry 1280x1024
```

In another terminal on your own local computer a VNC client must now be opened. We recommend **remmina**, for Windows **Xming** must also be installed and started for X11-forwarding. The VNC client remmina is started with the command:

```
remmina
```

If remmina does not start, the following command may have to be executed beforehand:

```
export DISPLAY=:0
```

With remmina, a new connection profile must be created with the help of the button in the upper left corner with the following information:
- **Protocol**: Remmina VNC Plugin
- in the **Basic** tab:

- *Server*: vis0X:Y, whereby X is the assigned vis node and Y is the display number of the VNC server, this is part of the output after the „vncserver“ command
- *Color depth*: High color (16 bpp)
- *Quality*: Medium

- in the **Advanced** and **Autostart** tabs:

- no changes

- in the **SSH Tunnel** tab:

- Select *Enable SSH Tunnels*
- Select *Custom* and enter: phoenix.hlr.rz.tu-bs.de
- *Username*: Enter your own username
- Select authentication via *Password*

Then save the connection profile with the **Save** button and connect with the **Connect** button. You will then be asked for your own password for the Phoenix and after entering this for the VNC

password that was created in the setup-vnc.sh step.

In the window you can open a terminal. To use software in visual and interactive mode the corresponding module has to be loaded. If the graphic card should be used (this is the case for visualisation), then in front of the terminalcommand to open the software **vglrun** has to be added. E.g. to start ParaView in interactive mode, the following command has to executed after loading the corresponding module:

```
vglrun paraview
```

After you have finished using it, the VNC server on the Phoenix must be terminated, this can be done with the command

```
vncserver -kill :Y where Y is the display number
```

If the VNC server is not closed, it should be done the next time you log in, otherwise the display number will be shifted (e.g. :1 becomes :2).

Monitoring jobs

Using `squeue` you get a list of all jobs currently running on the system. You can get more information with `squeue -l`.

```
username@login01 [~]squeue
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
               333  standard MD_BA_GA user0001  R      58:56      1 node265
               336  standard     bash username  R       1:12      2 node267,node268
               334  standard     bash user0002  R      50:16      1 node266
               331       vis      vis user0003  R    1:25:22      1 vis01
               329  standard    name1 user0003  PD     00:00      1 (Resources)
```

`JOBID` is your jobs ID to specify your job to SLURM. `NAME` is the job name you chose for better identification. `ST` is the status of your job (`R` = running, `PD` = pending). `TIME` gives the time your job is currently running. If your job is pending, `NODELIST` will give a short explanation, why it is not (yet) running.

Cancelling jobs

You can cancel your jobs with the ID from `squeue` using `scancel <ID>`.

```
username@login01 [~]scancel 336
salloc: Job allocation 336 has been revoked.
```

Module system

We use the Environment Modules System (short Modules) on the Phoenix.

Modules change and extend pathes and variables to load applications and libraries from different places. All installed software used on the Phoenix has a modulefile which loads the necessary variables to run the software.

Show available modules

For an overview of available modules you can use

```
module avail
```

The currently loaded modules are shown with

```
module list
```

This way you can see which software and which versions are installed on the Phoenix.

Module load and unload

You can load modules with

```
module load MODULNAME
```

You can remove modules with

```
module remove MODULNAME
```

To load or remove multiple modules at once you just have to seperate them by a space.

Executing commands and aliases on login

/home/username/.bash_profile acts like .bashrc on most Linux distributions. Commands added there will be automatically executed on login. To edit the file you can for example use nano: `nano .bash_profile`

Starting Abaqus jobs

To make Abaqus compatible with Slurm and MPI, some commands have to be executed in the jobfile. Here is an exemplary jobfile:

```
#!/bin/bash -l

#SBATCH --partition=standard
#SBATCH --nodes=1
#SBATCH --job-name=dingALE
#SBATCH --ntasks-per-node=20
#SBATCH --time=48:00:00
#SBATCH -o bo-%j.log


module purge
module load software/abaqus/abaqus_2016


input_file=dingALE.inp

working_dir=~/DingALE
cd $working_dir

### Create ABAQUS environment file for current job, you can set/add your own options (Python syn
env_file=custom_v6.env


##############################################################################


cat << EOF > ${env_file}
mp_file_system = (DETECT,DETECT)
EOF

node_list=$(scontrol show hostname ${SLURM_NODELIST} | sort -u)

mp_host_list="["
for host in ${node_list}; do
    mp_host_list="${mp_host_list}['$host', ${SLURM_CPUS_ON_NODE}],"
done

mp_host_list=$(echo ${mp_host_list} | sed -e "s/,$/]/")

echo "mp_host_list=${mp_host_list}"  >> ${env_file}


### Set input file and job (file prefix) name here
job_name=${SLURM_JOB_NAME}


### ABAQUS parallel execution
abaqus job=${job_name} input=${input_file} cpus=${SLURM_NTASKS} standard_parallel=all mp_mode=mp
```

If more RAM is needed, the *fat*-Partition can also be used.

Starting Ansys jobs

We recommend use our new script, which converts your jobfile automaticly. Just write a normal jobfile and call this script and use > to save the output to a new file.

Here is an example for normal Ansys CFX file without any license checks:

```
#!/bin/bash -l

#SBATCH --partition=standard
#SBATCH --nodes=2
#SBATCH --time=3:00:00
#SBATCH --job-name=TCTM
#SBATCH --ntasks-per-node=20

module load software/ansys/18.0


########################
#### HIER VARIABEL ####
########################
NUMPROCS=40
module load software/ansys/18.0

cfx5solve -batch -chdir $working_dir -double -verbose -def $working_dir/Heater_pt_Loss.def -star
```

Here is an example for normal Ansys Fluent file without any license checks:

```
#!/bin/bash

#SBATCH --partition=standard
#SBATCH --nodes=3
#SBATCH --time=3:00:00
#SBATCH --job-name=ansys_flu
#SBATCH --ntasks-per-node=20
#SBATCH --exclusive

# load module fluent v19.2
module load software/ansys/19.2

# The Journal file
JOURNALFILE=journalFile.jou

# Total number of Processors. Nodes * 20
NPROCS=60

# MPI key, dont change!
export MPI_IB_PKEY=0x8001

fluent 3ddp -g -t $NPROCS -slurm -i $JOURNALFILE > fluent.out
```

Here is the code to add an license check:

```
/cluster/share/make_ansys_job_check_for_license.sh Script.sh NumberOfLicences1 LicenceType1 Numb
```

An example:

```
/cluster/share/make_ansys_job_check_for_license.sh myjob.sh 50 aa_r_hpc > myjob_now_working_on_p
```

There are different license types. If you are not sure which one to choose, just use aa_r_hpc.

Following is the old, legacy way of starting an ansys job.

Don't use the code under this warning, unless you have a very good reason to do otherwise.

The files CFX.sh and nodes2ansys.py must be created. Then an Ansys job can be started via sbatch CFX.sh. StaticMixer.def should be replaced with your own file.

CFX.sh:

```
#!/bin/bash -l
# export PATH=/cluster/tools/ansys_inc/v180/CFX/tools/multiport/mpi/lnamd64/intel/bin:$PATH
# export PATH=/cluster/tools/ansys_inc/v180/commonfiles/MPI/Intel/5.1.3.223/linx64:$PATH

#SBATCH --partition=standard
#SBATCH --nodes=2
#SBATCH --time=3:00:00
#SBATCH --job-name=Ansys
#SBATCH --ntasks-per-node=20
#

######################
### HERE CHANGEABLE ###
######################
export working_dir=$HOME/ansys
# export ALLMACHINES=$(scontrol show hostname $SLURM_JOB_NODELIST |paste -d, -s )
/usr/bin/python nodes2ansys.py
export ALLMACHINES=`cat $HOME/ansys/cfx_nodefile.dat`
echo $SLURM_JOB_NODELIST>$HOME/ansys/slurmjoblist
echo $ALLMACHINES>$HOME/ansys/machines
#################
NUMPROCS=40
#################
cd $working_dir

export TMI_CONFIG=/cluster/tools/ansys_inc/v180/commonfiles/MPI/Intel/5.1.3.223/linx64/etc/tmi.c
export I_MPI_FABRICS=shm:tmi
export I_MPI_FABRICS_LIST=tmi
export I_MPI_FALLBACK=0
export I_MPI_TMI_PROVIDER=psm2

module load software/ansys/18.0
```

```
cfx5solve -batch -chdir $working_dir -double -verbose -def $working_dir/StaticMixer.def -start-m
```

nodes2ansys.py

```
#!/usr/bin/python

import os

# READ SLURM JOB LIST

#line = example data: fat[1-3]
line = os.environ['SLURM_JOB_NODELIST']
#line = 'fat[001-003]\n'

line.strip()

file = open( "cfx_nodefile.dat" , "w" )

#name = fat , numbers = 001-003
name,numbers = line.split('[')
numbers = numbers.replace(']', '')

#001-003 -> 001 003
start,end = numbers.split('-')

#001 003 -> 1 3
start_int = int(start)
end_int = int(end)

#from 1 to 3 do:
for number in range(start_int, end_int+1):
    number_string = str(number)

    #add the leading zeroes to node-number 2 -> 002
    while len(number_string) < len(start):
        number_string = "0" + number_string

    #write down the nodename like fat002 20 times... you could do this most likely with nt
    for i in range (1,21):
        file.write(name + str(number_string) + ',' )

file.close
```

Anaconda

Anaconda is a distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. In this part we will explain, how you can download and install Anaconda on Phoenix and how you can use it to set up custom

environments for different purposes.

Installation

- Go to the Anaconda download page using your web browser: Anaconda Download Page.
- Choose the appropriate distribution for Phoenix „Linux 64-Bit (x86) Installer“ and copy the link address.
- Log in to the Phoenix as usual.
- Use the „wget“ command to download the Anaconda installer using the link you copied (replace <paste_link_here> with the actual link you copied):

```
wget <paste_link_here>
```

- After the download is complete, make the Anaconda installer script executable (replace <version> with the version number you downloaded):

```
chmod +x Anaconda3-<version>-Linux-x86_64.sh
```

- Run the Anaconda installer script. Follow the prompts and accept the license agreement:

```
./Anaconda3-<version>-Linux-x86_64.sh
```

- During the installation process, you will be asked to confirm the installation location. Use the default location, this should be /home/<your_username>/anaconda3. The installation process will take several minutes.

Environment Management

Conda is a powerful package manager and environment manager that you can use in a terminal window. Conda allows you to create separate environments containing files, packages, and their dependencies that will not interact with other environments. When you begin using conda, you already have a default environment named `base`. You don't want to put programs into your base environment, though. Create separate environments to keep your programs isolated from each other. To start the `base` environment, navigate to your `/home`-folder, in there should be a folder named `anaconda3` and use the command:

```
source anaconda3/bin/activate
```

If you installed Anaconda to the default location (your `/home`-folder), you can also use the command:

```
source ~/anaconda3/bin/activate
```

In front of your prompt **(base)** should be displayed.

Nutzeranleitung Fortgeschrittene

Kompilieren

> **Problem:** Das Kompilieren auf dem Phoenix-Cluster klappt nicht, z.B. wenn gegen eine Bibliothek im Modul-System gelinkt wird (OpenCV).

Dies kann daran liegen, dass viele Bibliotheken (Libraries) mit dem optimierten Intel Compiler kompiliert wurden. Denn viele der installierten Bibliotheken sind mit dem neuesten Intel Compiler (icc) mit AVX 2 Optimierungen kompiliert.

In diesem Fall kann es daher helfen, den **Intel Compiler** (icc) zu verwenden.

Das kann im Idealfall hohe Leistungszuwächse - 50% und mehr - z.B. auf neuen Xeon CPUs bringen, und meistens läuft der Code, der mit dem icc kompiliert wird, schneller. Gerade die *auto vectorization* funktioniert gut, da der icc von sich aus Schleifen umwandelt, um die CPU Eigenschaften besser nutzen zu können.

Das kann aber dazu führen, dass der **GNU Compiler** (gcc) Probleme mit Bibliotheken hat, die für den Intel Compiler stark optimierten Code nutzen. Deshalb muss der Kommandozeile/Shell mitgeteilt werden, dass diese den icc (bzw. icpc für C++) statt den gcc nutzen soll, falls dieses Problem auftritt.

Das wird mit folgenden Befehlen getan:

```
module load intel-studio-2019
source /cluster/share/intelenv.sh
```

Anschließend muss, wie üblich, der Befehl

```
cmake
```

ausgeführt werden.

> Mehr Informationen, insbesondere auch für in Fortran geschrieben Code, kann unter https://software.intel.com/en-us/articles/performance-tools-for-software-developers-building-

hdf5-with-intel-compilers gefunden werden.

Wichtiger Hinweis: Beim Fortan Compiler von Intel (ifort) darf unter Standard-Einstellungen Probleme eine Zeile maximal 132 Zeichen umfassen!

GLIBC

Um neuere Versionen von glibc zu verwenden, ist es leider nicht möglich, einfach das Modul zu laden. Stattdessen werden dem Compiler Flags übergeben, mit denen die Pfade für das Finden der Bibliotheken sowie des Loaders der dynamischen Bibliotheken gesetzt werden. Hier ein kurzes Beispiel:

```
g++ -o My_Program -Wl,--rpath=/cluster/lib/glibc/2.29/lib -Wl,--dynamic-linker=/cluster/lib/glib
```

Gaussview

Um Gaussview zu nutzen, muss wie folgt vorgegangen werden. Via ssh -X für eine grafische Oberfläche auf Phoenix einloggen. Dann einen Visualisierungsknoten allokieren. Dann den Namen des allokierten Knotens in Erfahrung bringen und sich auf diesem Knoten einloggen. Dort dann gaussview mit gv starten.

```
ssh -X user@phoenix.hlr.rz.tu-bs.de
sbatch gaussview.job
    Submitted batch job 100990
squeue | grep 100990
    100990       vis gaussvie raskrato  R       0:14      1 vis01
ssh -X vis01
gv
```

Die Jobfile zum Allokieren eines Visualisierungsknoten.

gaussview.job

```
#!/bin/bash -l

#SBATCH --partition=vis
#SBATCH --nodes=1
#SBATCH --time=1:00:00
#SBATCH --job-name=gaussview
#SBATCH --ntasks-per-node=20

module load software/molecules/gaussian
sleep 1h
```

Python Module

Auf dem Cluster sind eine Reihe von Python Module bereits vorinstalliert. Diese können eingesehen werden, wenn Sie mit *python2* oder *python3* deren „Interpreter“ beitreten und dort die folgende Anweisung *help(„modules“)* ausführen.

Sollte ein Modul benötigt werden, welches dort nicht aufgeführt ist, gibt es zwei Möglichkeiten. Sollten Sie nachweisen können, dass eine große Nutzergruppe Bedarf an diesem bestimmten Modul hat, so wenden Sie sich an das HLR-Service-Team. Da es allerdings zu aufwendig wäre und den Betrieb stören würde, das gesamte Cluster ständig mit neuen Python-Packeten zu befüttern, müssen Sie ansonsten auf virtuelle Umgebungen zurückgreifen.

Für Python2 gestaltet sich dies relativ einfach:

```
virtualenv venv //Erstellt im aktuellen Arbeitsverzeichnis einen Ordner mit dem Namen "venv", ir
source venv/bin/activate //Damit wird die Python-Umgebung geladen, dies kann auch in einem Job-F
deactivate //Damit verlässt man die aktuelle Pyton-Umgebung.
```

Solange Sie sich in der Python Umgebung befinden, wird dies durch ein Voranstellen Ihres Names in der Kommandozeile verdeutlicht. Das sieht dann z.B. so aus:

```
(venv)[testuser@login01 ~]
```

In der virtuellen Umgebung können Sie sich nun nach Herzenslust mittels **pip** jedes Modul beschaffen. Sollten Sie keine Verwendung mehr für eine Umgebung haben, lässt sie sich einfach durch das Löschen des Ordners entfernen.

Für Python3 muss aktuell ein kleiner Umweg gemacht werden, bevor virtuelle Umgebungen anlegt werden können:

```
module load python/3.7
python3 -m pip install --user virtualenv //Hiermit wird virtualenv für sie als Nutzer installier
python3 -m virtualenv venv //Damit stellen sie sicher, dass Python3 in der virtuellen Umgebung e
```

Wir hoffen mit dem nächsten Update von Python diesen Schritt obsolet zu machen.

Sie können natürlich fragen, wieso Sie nicht immer *pip install –user* verwenden, um sich virtuelle Umgebungen zu sparen. Und in der Regel haben Sie Recht, aber es gibt immer wieder Konflikte zwischen verschiedenen Modulen und so garantieren Sie, dass Sie für jedes Projekt eine konfliktfreie Umgebung schaffen können.

> Zum Schluss noch eine Anmerkung zu Python-Bibliotheken, die mit C-Code daherkommen: Der Intel-Kompiler wird von pip nicht akzeptiert. Stattdessen laden Sie daher bitte den gcc-Compiler über das entsprechende Modulefile „comp/gcc/8.3.0“ oder welche Version Sie auch immer verwenden möchten.

Queuing-System (SLURM) and Jobfiles

Important info:

Big parts of this script was taken from the documentation of the Hamburg HPC Competence Center (HHCC). Please visit their website for more details on the Use of the Command Line Interface or about Using Shell Scripts.

Description:

- You will learn to how to use Environment Modules, a widely used system for handling different software environments (basic level)
- You will learn to use the workload manager SLURM to allocate HPC resources (e.g. CPUs) and to submit a batch job (basic level)
- You will learn how a simple jobscript looks like and how to submit it (basic to intermediate level)

General Information

Environment Modules are a tool for managing environment variables of the shell. Modules can be loaded and unloaded dynamically and atomically, in an clean fashion. Details can be found on the official website.

The workload manager used on the Phoenix-Cluster is SLURM (Simple Linux Utility for Resource Management). SLURM a widely used open source workload managers for large and small Linux clusters which is controlled via a CLI (Command Line Interface). Details can be found in the official documentation.

Environment Modules

Introduction:

The `module load` command extends variables containing search paths (e.g. `PATH` or `MANPATH`). The `module unload` command is the corresponding inverse operation, it removes entries from search

paths. By extending search paths software is made callable. Effectively software can be provided through Modules. An advantage over defining environment variables directly in the shell is that Modules allow to undo changes of environment variables. The idea of introducing Modules is to be able to define software environments in a modular way. In the context of HPC, Modules make it easy to switch compilers or libraries, or to choose between different versions of an application software package.

Naming:

Names of Modules have the format `program/version`, just `program` or even a slightly more nested path description. Modules can be loaded (and always be unloaded) without specifying a version. If the `version` is not specified the default `version` will be loaded. The default `version` is either explicitly defined (and will be marked in the output of `module avail`) or module will load the `version` that appears to be the latest one. Because defaults can change `version`**s should always be given if reproducibility is required**.

Dependences and conflicts:

Modules can have dependences, i.e. a Module can enforce that other Modules that it depends on must be loaded before the Module itself can be loaded. Module can be conflicting, i.e. these modules must not be loaded at the same time (e.g. two version of a compiler). A conflicting Module must be unloaded before the Module it conflicts with can be loaded.

Caveats:

The name Modules suggest that Modules can be picked and combined in a modular fashion. For Modules providing application packages this is true (up to possible dependences and conflicts described above), i.e. it is possible to chose any combination of application software.

However, today, environments for building software are not modular anymore. In particular, it is no longer guaranteed that a library that was built with one compiler can be used with code generated by a different compiler. Hence, the corresponding Modules cannot be modular either. A popular way to handle this situation is to append compiler information to the version information of library Modules. Firstly, this leads to long names and secondly, to very many Modules that are hard to overlook. A more modern way is to build up toolchains with Modules. For example, in such a toolchain only compiler Modules are available at the beginning. Once a compiler Module is loaded, MPI libraries (the next level of tools) become available and after that all other Modules (that were built with that chain).

Important commands:

Important Module commands are:

list Modules currently loaded	`module list`
list available Modules	`module avail`

load a Module	`module load program[/version]`
unload a Module	`module unload program`
switch a Module (e.g. compiler version)	`module switch program program/version`
add or remove a directory/path to the Module search path (e.g. by an own Module directory)	`module [un]use [–append] path`

Self-documentation:

Modules are self-documented:

show the actions of a Module	`module display program/version`
short description of [one or] all Modules	`module whatis [program/version]`
longer help text on a Module	`module help program/version`
help on module itself	`module help`

Basics of SLURM

Introduction:

There are three key functions of SLURM described on the SLURM website:

“… First, it allocates exclusive and/or non-exclusive access to resources (compute nodes) to users for some duration of time so they can perform work. Second, it provides a framework for starting, executing, and monitoring work (normally a parallel job) on the set of allocated nodes. Finally, it arbitrates contention for resources by managing a queue of pending work. …”

SLURM’s default scheduling is based on a FIFO-queue, which is typically enhanced with the Multifactor Priority Plugin to achieve a very versatile facility for ordering the queue of jobs waiting to be scheduled. In contrast to other workload managers SLURM does not use several job queues. Cluster nodes in a SLURM configuration can be assigned to multiple partitions by the cluster administrators instead. This enables the same functionality.

A compute center will seek to configure SLURM in a way that resource utilization and throughput are maximized, waiting times and turnaround times are minimized, and all users are treated fairly.

The basic functionality of SLURM can be divided into three areas:

- Job submission and cancellation
- Monitoring job and system information
- Retrieving accounting information

Job submission and cancellation:

There are three commands for handling job submissions:

- `sbatch`
  - submits a batch job script to SLURM's job queue for (later) execution. The batch script may be given to sbatch by a file name on the command line or can be read from stdin. Resources needed by the job may be specified via command line options and/or directly in the job script. A job script may contain several job steps to perform several parallel tasks within the same script. Job steps themselves may be run sequentially or in parallel. SLURM regards the script as the first job step.
- `salloc`
  - allocates a set of nodes, typically for interactive use. Resources needed may be specified via command line options.
- `srun`
  - usually runs a command on nodes previously allocated via sbatch or salloc. Each invocation of srun within a job script corresponds to a job step and launches parallel tasks across the allocated resources. A task is represented e.g. by a program, command, or script. If srun is not invoked within an allocation it will via command line options first create a resource allocation in which to run the parallel job.

SLURM assigns a unique *jobid* (integer number) to each job when it is submitted. This *jobid* is returned at submission time or can be obtained from the `squeue` command.

The `scancel` command is used to abort a job or job step that is running or waiting for execution.

The `scontrol` command is mainly used by cluster administrators to view or modify the configuration of the SLURM system but it also offers the users the possibility to control their jobs (e.g. to hold and release a pending job).

The Table below lists basic user activities for job submission and cancellation and the corresponding SLURM commands.

User activities for job submission and cancellation (user supplied information is given in *italics*)

User activity	SLURM command
Submit a job script for (later) execution	`sbatch` *job-script*
Allocate a set of nodes for interactive use	`salloc` –nodes=*N*
Launch a parallel task (e.g. program, command, or script) within allocated resources by `sbatch` (i.e. within a job script) or `salloc`	`srun` *task*
Allocate a set of nodes and launch a parallel task directly	`srun` –nodes=*N task*
Abort a job that is running or waiting for execution	`scancel` *jobid*
Abort all jobs of a user	`scancel` –user=*username* or generally `scancel` –user=$USER

User activity	SLURM command
Put a job on hold (i.e. pause waiting) and Release a job from hold (These related commands are rarely used in standard operation.)	`scontrol` hold *jobid* `scontrol` release *jobid*

The major command line options that are used for `sbatch` and `salloc` are listed in the Table below. These options can also be specified for `srun`, if `srun` is not used in the context of nodes previously allocated via `sbatch` or `salloc`.

Major `sbatch` and `salloc` options

Specification	Option	Comments
Number of nodes requested	–nodes=*N*	
Number of tasks to invoke on each node	–tasks-per-node=*n*	Can be used to specify the number of cores to use per node, e.g. to avoid hyper-threading. (If option is omitted, all cores and hyperthreads are used; Hint: using hyperthreads is not always advantageous.)
Partition	–partition= *partitionname*	
Job time limit	–time=*time-limit*	time-limit may be given as minutes or in hh:mm:ss or d-hh:mm:ss format (d means number of days)
Output file	–output=*out*	Location of stdout redirection

For the `sbatch` command these options may also be specified directly in the job script using a pseudo comment directive starting with `#SBATCH` as a prefix. The directives must precede any executable command in the batch script:

```
#!/bin/bash
#SBATCH --partition=std
#SBATCH --nodes=2
#SBATCH --tasks-per-node=16
#SBATCH --time=00:10:00
...
srun ./helloParallelWorld
```

A complete list of parameters can be retrieved from the `man` pages for `sbatch`, `salloc`, or `srun`, e.g. via

```
man sbatch
```

Monitoring job and system information:

There are four commands for monitoring job and system information:

- `sinfo`
  - shows current information about nodes and partitions for a system managed by SLURM. Command line options can be used to filter, sort, and format the output in a variety of ways. By default it essentially shows for each partition if it is available and how many nodes and which nodes in the partition are allocated or idle (or are possibly in another state like down or drain, i.e. not available for some time). This is useful for the user e.g. to decide in which partition to run a job. The number of allocated and idle nodes indicates the actual utilization of the cluster.
- `squeue`
  - shows current information about jobs in the SLURM scheduling queue. Command line options can be used to filter, sort, and format the output in a variety of ways. By default it lists all pending jobs, sorted descending by their priority, followed by all running jobs, sorted descending by their priority. The major job states are:
    - R for Running
    - PD for Pending
    - CD for Completed
    - F for Failed
    - CA for Cancelled
  - The `TIME` column shows for running jobs their execution time so far (or 0:00 for pending jobs).
  - The `NODELIST (REASON)` column shows either on which nodes a job is running or why the job is pending. A job is pending for two main reasons:
    - it is still waiting for resources to become scheduled, shown as (`Resources`),
    - its priority is still not sufficient for it to become executed, shown as (`Priority`), i.e. there are other jobs with a higher priority pending in the queue.
  - The position of a pending job in the queue indicates how many jobs are executed before and after it. The `squeue` command is the main way to monitor a job and can e.g. also be used to get the information about the expected starting time of a job (see Table below).
- `sstat`
  - is mainly used to display various status information of a running job taken as a snapshot. The information relates to CPU, task, node, Resident Set Size (RSS), and virtual memory (VM), etc.
- `scontrol`
  - is mainly used by cluster administrators to view or modify the configuration of the SLURM system, but it also offers users the possibility to get some information about the cluster configuration (e.g. about partitions, nodes, and jobs).

The Table below lists basic user activities for job and system monitoring and the corresponding SLURM commands.

User activity	SLURM command

View information about currently available nodes and partitions. The state of a partition may be `UP`, `DOWN`, or `INACTIVE`. If the state is `INACTIVE`, no new submissions are allowed to the partition.	`sinfo` [–partition=*partitionname*]
View summary about currently available nodes and partitions. The `NODES`(`A/I/O/T`) column contains corresponding number of nodes being allocated, idle, in some other state and the total of the three numbers.	`sinfo` -s
Check the state of all jobs.	`squeue`
Check the state of all own jobs.	`squeue` –user=$USER
Check the state of a single job.	`squeue` -j *jobid*
Check the expected starting time of a pending job.	`squeue` –start -j *jobid*
Display status information of a running job (e.g. average CPU time, average Virtual Memory (VM) usage – see `sstat` –helpformat and `man sstat` for information on more options).	`sstat` –format=AveCPU, AveVMSize -j *jobid*
View SLURM configuration information for a partition cluster node (e.g. associated nodes).	`scontrol` show partition *partitionname*
View SLURM configuration information for a cluster node.	`scontrol` show node *nodename*
View detailed job information.	`scontrol` show job *jobid*

Retrieving accounting information:

There are two commands for retrieving accounting information:

- `sacct`
  - shows accounting information for jobs and job steps in the SLURM job accounting log or SLURM database. For active jobs the accounting information is accessed via the job accounting log file. For completed jobs it is accessed via the log data saved in the SLURM database. Command line options can be used to filter, sort, and format the output in a variety of ways. Columns for jobid, jobname, partition, account, allocated CPUs, state, and exit code are shown by default for each of the user's jobs eligible after midnight of the current day.
- `sacctmgr`
  - is mainly used by cluster administrators to view or modify the SLURM account information, but it also offers users the possibility to get some information about their account. The account information is maintained within the SLURM database. Command line options can be used to filter, sort, and format the output in a variety of ways.
  - The Table below lists basic user activities for retrieving accounting information and the corresponding SLURM commands.

User Activity	SLURM Command
View job account information for a specific job.	`sacct` -j *jobid*
View all job information from a specific start date (given as yyyy-mm-dd).	`sacct` -S *startdate* -u $USER
View execution time for (completed) job (formatted as days-hh:mm:ss, cumulated over job steps, and without any header).	`sacct` -n -X -P -o Elapsed -j *jobid*

Submitting a batch job:

Below an example script for a SLURM batch job – in the sense of a hello world program – is given. The job is suited to be run in the Phoenix HPC cluster at the Gauß-IT-Zentrum. For other cluster systems some appropriate adjustments will probably be necessary.

```
#!/bin/bash
   # Do not forget to select a proper partition if the default
   # one is no fit for the job! You can do that either in the sbatch
   # command line or here with the other settings.
#SBATCH --partition=standard
   # Number of nodes used:
#SBATCH --nodes=2
   # Wall clock limit:
#SBATCH --time=12:00:00
   # Name of the job:
#SBATCH --job-name=nearest
   # Number of tasks (cores) per node:
#SBATCH --ntasks-per-node=20

   # If needed, set your working environment here.
working_dir=~
cd $working_dir

   # Load environment modules for your application here.
module load comp/gcc/6.3.0
module load mpi/openmpi/2.1.0/gcc

   # Execute the application.
mpiexec -np 40 ./test/mpinearest
```

The job script file above can be stored e.g. in `$HOME/hello_world.sh` (`$HOME` is mapped to the user's home directory).

The job is submitted to SLURM's batch queue using the default value for partition (scontrol show partitions (also see above) can be used to show that information):

```
[exampleusername@node001 14:48:33]~$ sbatch $HOME/hello_world.sh
Submitted batch job 123456
```

The start time can be selected via `–begin`, for example::

```
--begin=16:00
--begin=now+1hour
--begin=now+60 (seconds by default)
--begin=2010-01-20T12:34:00
```

More information can be found via `man sbatch`. All parameters shown there can be included in the jobscript via `#SBATCH`.

The output of `sbatch` will contain the jobid, like 123456 in this example. During execution the output of the job is written to a file, named `slurm-123456.out`. If there had been errors (i.e. any output to the *stderr*stream) a corresponding file named `slurm-123456.err` would have been created.

Cancelling a batch job:

```
scancel <jobid>
```

The required ID can be viewed via the general command `squeue` or the user specific command `squeue -u $USER`.

If you want to delete all jobs of a user:

```
scancel –u <username>
```

How to change a node status (root only):

```
scontrol update nodename=node[005-008] state=drain reason=”RMA”
```

This command will exclude the node from the list of available nodes. This ensures that no more jobs can be submitted to this node, allowing it to be used for testing etc.

```
scontrol update nodename=node[005-008] state=idle
```

This reverses the previous command and returns the node back to the list of available nodes. Executing this command might also be necessary if a node crash caused a ramovel of a node from the batch system.

Interactive jobs (intermediate difficulty):

Method one:

Assume you have submitted a job as follows:

```
sbatch beispiel.job
Submitted batch job 1256
```

Let the corresponding jobfile be the following:

```
beispiel.job

#!/bin/bash -l

#SBATCH --partition=standard
#SBATCH --nodes=1
#SBATCH --time=7-00:00:00
#SBATCH --job-name=towhee
#SBATCH --ntasks-per-node=1

cd ~/data_dir
sleep 168h
```

In this case, the command `squeue -l` will show you which node the job is currently running on. For example:

```
1256 standard towhee raskrato  RUNNING       0:04 7-00:00:00      1 node282
```

You can then log onto that node via `ssh node282` and start a new shell via screen (please follow the link for further information). The program can then be started in this new shell.

Once you are done, you can exit the shell via:

```
strg a d
```

- You can start as many shells as you like. The command `screen -r` will show a list of all shells (if it is only one, you will instead return to said shell).
- You can access a shell running in the background via `screen -r <shellnummer>`.
- You can quit a shell by pressing the key-combination *CTRL+C* and typing in `exit`.

Another way to use the allocated nodes is via the `salloc` command (see method two below).

Method two:

Interactive sessions under control of the batch system can be created via `salloc`. `salloc` differs from `sbatch` by the fact that resources are initially only reserved (i.e. allocated) without executing a job script. Also, the session is running on the node on which `salloc` was invoked (but not on a compute node in contrast to submission with `sbatch`). This is often useful during the interactive development of a parallel program.

A single node is reserved for interactive usage as follows:

```
[exampleusername@node001 14:48:33]~$ salloc
```

When the resources are granted by SLURM, `salloc` will start a new shell on the (login or head) node where `salloc` was executed. This interactive session is terminated by exiting the shell or by

reaching the time limit.

An OpenMP program using *N* threads, for example, can be started on the allocated node as follows:

```
[exampleusername@node001 14:48:33]~$ export OMP_NUM_THREADS=N
[exampleusername@node001 14:48:33]~$ srun my-openmp-binary
```

To start an interactive parallel MPI program *N* nodes can be allocated as follows:

```
[exampleusername@node001 14:48:33]~$ salloc --nodes=N
```

The MPI Program using *n=32* processes, for example, can be started on the allocated nodes as follows:

```
[exampleusername@node001 14:48:33]~$ mpirun -np 32 my-mpi-binary
```

Another way to use the allocated nodes is to use ssh to establish connections to them (see method one above).

Using a Visual Node on Windows

Description:

- You will learn about the installation and use of the program Xming (basic level)
- You will learn how to set up PuTTY and connect to a visual node using Xming (intermediate level)
- You will see an example of how to run a sample GUI application (intermediate level)

Introduction & Prerequirements

While the command-line interface is a powerful tool to remotely access the Phoenix Cluster to manage jobs and access files on the server, some applications require a graphical user interface (GUI) in order to be used fully. In this tutorial, you will therefore learn how to connect to one of Phoenix' visual nodes in order to run such a GUI application.

Please note: This tutorial is for Windows machines only. For a short description of how to use visual nodes on a Linux machine, please visit this link.

In order to perform the steps required here, you will need to have access to the remote network of the TU Braunschweig and have the application PuTTY installed. For a detailled tutorial about the basic usage of the Phoenix Cluster, please visit this link (for Windows users only). Once you have set up the VPN and have PuTTY (and preferably WinSCP) installed, you can continue following this toturial.

Installing and using Xming

Description

Official description: „Xming is the leading X Window System Server for Microsoft Windows®. It is fully featured, lean, fast, simple to install and because it is standalone native Windows, easily made portable (not needing a machine-specific installation or access to the Windows registry).“ Source

This means, that it provides the tools necessary to allow for a display of graphical user interfaces on your local machine, a feature that does not come natively with Windows.

Xming itself does not have a real GUI and generally needs no configuration to run. Once started, it should enable PuTTY to display graphical applications on your machine.

Download and installation

In order to download the installation file, please visit: https://sourceforge.net/projects/xming/

Start the installation file and click through the steps to finish the setup. Then, start the program. You should now see the Xming icon among the active programs on the task bar.

This concludes the setup of Xming, as you do not need to interact with the program itself.

Setup of PuTTY for the use with Xming

Changing the PuTTY Settings

As a first step, you need to enable X11 forwarding. This tells PuTTY that it should allow information regarding graphical user interfaces to be transmitted to your computer. In oder to do so, you need to start PuTTY and take a look at the category table on the left. You need to open the following entries: `Connection` → `SSH` → `X11` (as seen in the screenshot below)

PuTTY Setup - Xming

In this windows with the title `Options controlling SSH X11 forwarding`, you need to do two things:

1. Check the box titled `Enable X11 forwarding`
2. In the textfield labelled `X display location`, type the following: `localhost:0.0`

Now you can go back to the category entry `Sessions`, save the settings under the profile of the Phoenix cluster connect to it as always.

Starting an interactive job and the graphical application

Now that the setup is complete, we can start an interactive job on a visual node and use it to start a graphical application. To do so, we need to reserve the node for a specified amount of time. The following is an example jobfile (for details of how to submit jobfiles, please click here.

```
#!/bin/bash -l

#SBATCH --partition=vis
#SBATCH --nodes=1
```

```
#SBATCH --time=6:00:00
#SBATCH --job-name=VisExample
#SBATCH --ntasks-per-node=1


sleep 6h
```

Once the job is submitted, you can check its status via the command `squeue -u $USER`. Once it has been started, you should see which node you have been assigned on the right side in the column *NODELIST (REASON)*. You need to know the number you have been assigned, which `vis0X`, where `X` is variable. In the following screenshot, you can see that we were assigned node `vis01`.

VIS0X

Knowing this number, you are now able to log into your node with the following command (after replacing the `X` with the real number):

```
ssh -Y vis0X
```

Please note that it is important that you do not forget the `-Y` argument, as this will enable trusted X11 forwarding. This should automatically set the set the `DISPLAY` variable. You can test this by typing

```
echo $DISPLAY
```

This should show something like `localhost:10.0` as an output. (If the variable is not set, you can alternatively do so manually with the command `export DISPLAY=localhost:10.0`, even though this will unlikely be necessary)

In order to test if the setup has worked, we can run a first sample application via the following command:

```
xeyes
```

This should open a Xming window that shows a little image of eyes that follow your mouse cursor which looks as followed:

xeyes

With all of this done, we are now finally able to run our first real graphial application on the visual node. For this test case, we prepared a python GUI appliaction, which we placed in our home folder. We then executed the program with the following command:

```
python3 main.py
```

As you can see in the following, our little game of snake running on the visual node is now being displayed on our local Windows machine (this specific example can be found here).

Vis node snake example

Important note: To run our specific sample program, we previously installed python via Anaconda in our home directory and are using this as our standard python. This step is *not* part of this tutorial, as only few graphical applications are based on python. Please make sure that your specific application is installed correctly with all prerequisites being fulfilled.

If you need information how to load modules, please click here. For information how to install any application not available in the module list, please use your favourite search engine to get help on how to install it on a remote CentOS Linux server. You should be able to do so using PuTTY and WinSCP as detailled here.

Conclusion

This tutorial should give you all the information needed to run a graphical application on a visual node of the Phoenix cluster and display the GUI on your local Windows machine.

This however is only meant to show you the general setup of your PC, it does not include the setup of your specific application. Please find solutions on how to run them on the internet.

Enjoy using the Phoenix clusters visual nodes!

Windows Setup for the Phoenix

Description:

- You will get a quick reminder how to log into the remote network via the Cisco VPN client (basic level)
- You will learn about the installation and use of the program WinSCP (basic level)
- You will learn how to use WinSCP to copy your first jobfile onto the Server (basic level)
- You will learn about the installation and use of the program PuTTY (basic level)
- You will learn how to use PuTTY to start your first sample jobfile (basic level)

Connecting to the remote network via the Cisco VPN client

This tutorial will introduce the tools WinSCP and PuTTY, which are very useful when working with a remote server from a Windows PC.

Please note however: You need to be connected to the remote network via a VPN client. We recommend the use of the Cisco AnyConnect Secure Mobility Client.

You can download the client by logging into your TU Braunschweig user account on the following website: https://vpngate.tu-braunschweig.de/+CSCOE+/logon.html#form_title_text
VPN Download Page

On the next page, you can download the installation file as well as display instructions regarding the installation process.
VPN Installation Page

(Alternatively, you can also manually install the client by downloading the file „anyconnect-win-<Version etc.>.msi“ on the following homepage: https://www.tu-braunschweig.de/it/downloads/software)

Finish the installation and connect to the remote server. When first starting the client, you need to type in the connection details as followed:

- **Connect to:** `vpngate.tu-bs.de` or `vpngate.tu-braunschweig.de`

- **Group:** `Tunnel TU Traffic` (recommended) or `Tunnel all Traffic`
- **Username** and **Password:** The login details of your TU Braunschweig account

For a more detailled tutorial visit: https://books.rz.tu-bs.de/books/vpn/chapter/vpn-einrichten (German only)

Installing and using WinSCP

Description

Official description: „WinSCP is an open source free SFTP client, FTP client, WebDAV client, S3 client and SCP client for Windows. Its main function is file transfer between a local and a remote computer. Beyond this, WinSCP offers scripting and basic file manager functionality.“ Source

This means that WinSCP allows the user to access a remote server (such as the Phoenix-cluster) with the purpose of transfering files between the user's home computer and the server file system. This can be done by navigating through a graphic user interface (GUI) that allows the user to access files just like they are used to on their local machine, for example dragging and dropping files into their file-system on the server.

WinSCP therefore provides an easy alternative for Windows users who are less comfortable navigating the Linux command line interface.

Download and installation

In order to download the installation file, please visit: https://winscp.net/eng/download.php

WinSCP Homepage

Start the installation file and click through the steps to finish the setup. Then, start the program.

Login

Before loging in, please make sure that your VPN is active and you are connected to the remote network of the TU Braunschweig.
In the following window, you will be asked to enter your login data. If the login window does not appear automatically, please click on „New Session“.



Type in the following information:

- **File Protocoll:** `SFTP`
- **Host Name:** `phoenix.hlr.rz.tu-bs.de`
- **Port Number:** `22`
- **Username** and **Password:** The login details of your TU Braunschweig account

Uploading files to the server

You should now see the file system of your local machine on the left and the content of your remote directory on the right.
Let's now create a sample jobfile called *ExampleJob.job* and upload it to our remote file system. The jobfile, which does nothing at all, looks as followed:

```
#!/bin/bash -l

#SBATCH --partition=standard
#SBATCH --nodes=1
#SBATCH --time=01:00:00
#SBATCH --job-name=TestJob
#SBATCH --ntasks-per-node=1

sleep 1h
```

Please note: In order for jobfiles to be recognized and executed by the Linux server, the file needs to be encoded as a Linux file.

One simple way to achieve this, is the use of the handy text editing tool Notepad++. Simply download it via the link provided, open the jobfile, click on *Edit* in the navigation bar and then on *EOL Conversions*. There select the option *Unix (LF)*.

Once you ensure that the file has the right encoding, you can drag-and-drop it onto the right side of the WinSCP interface either from its left side or Windows native File Explorer. You should then see your file appear in your remote folder as you can see in the following screenshot:

WinSCP File System
You have sucessfully placed your jobfile on the server in the directory `/home/username/`. Unless otherwise specified, this is also the folder where the output-file of the submitted job will be saved later.
You are now ready to submit the job via our next application, PuTTY.

Installing and using PuTTY

Description

Official description: „In really simple terms: you run PuTTY on a Windows machine, and tell it to connect to (for example) a Unix machine. PuTTY opens a window. Then, anything you type into that

window is sent straight to the Unix machine, and everything the Unix machine sends back is displayed in the window. So you can work on the Unix machine as if you were sitting at its console, while actually sitting somewhere else.“ Source

This means that PuTTY allows you to execute Linux command line commands on the remote server from your own home PC. This allows you to submit and monitor jobfiles and perform numerous other tasks.

As PuTTY does not provide a GUI, you are required to know about the basic functionality of command line interfaces. For more information on command line interfaces and submitting jobfiles, please read the other pages of this book.

Download and installation

In order to download the installation file, please visit:

https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

Start the installation file and click through the steps to finish the setup. Then, start the program.

Login

Before loging in, please make sure that your VPN is active and you are connected to the remote network of the TU Braunschweig.
In the first window, you will be asked to enter your login data.

PuTTY Login Dialogue

Type in the following information:

- **Host Name (or IP address):** `phoenix.hlr.rz.tu-bs.de`
- **Port Number:** `22`
- **Connection Type:** `SSH`

You can save this session data to make it easier to log in next time. Start the saved session by double clicking it or via the *Open* button.
In the following window, you will be asked to enter your login data. Please type in the login data of your TU Braunschweig account.

PuTTY Main Window

Using the command line interface by submitting our jobfile

We now want to start the jobfile we added to the directory `/home/username/` earlier via WinSCP.
Let's therefore first test if we are currently in the right directory with the following command:

```
pwd
```

It should display the correct folder, namely `/home/username/`. This means we are in the right folder and can start our jobfile named `ExampleJob.job` via the following command:

```
sbatch ExampleJob.job
```

The resulting output should be similar to the following: `Submitted batch job 1234567`.
If you instead see an error that the files contain unknown characters, you might have forgotten to change the encoding of the file to be compatible with the Linux server. In this case, please see the section about WinSCP where the tool *Notepad++* is introduced.
Now that the job is submitted, we can monitor all of our submitted jobs on the server with the following command:

```
squeue -u $USER
```

The output should now contain some information about your job such as the job-ID, your username and the time passed since the job was started.
Please note: The column `ST` shows the status of the job. If the job is started, it will display `R` for Running. If it instead shows `PD` for Pending, it means that all nodes are currently occupied and that you need to wait until a slot opens. For more information about the queuing system SLURM, read the dedicated page in this book.
Since our job is just blocking a node without doing anything, let's cancel it with the following command after filling in the correct job-ID:

```
scancel 1234567
```

If we now once again check our current jobs via the command above, it should no longer be visible.
If you have done everything correctly, your PuTTY console output should look a little like this:

PuTTY - First finished job

To see the output of your job, you can revisit the folder `/home/username/` in WinSCP. It should now contain a file called `slurm-1234567.out` (with the correct job-ID) with approximately the following content:

```
slurmstepd: error: *** JOB 1234567 ON node043 CANCELLED AT 2020-12-09T17:04:17 ***
```

For other applications, this file will usually contain more information.

Conclusion

Congratulations! By following this tutorial, you have now installed all the required software and also uploaded and submitted your first jobfile.
This should teach you the basic skills needed to use the Phoenix server from your local Windows

PC.

You can use your folder system to upload jobfiles, scripts and even programs such as Anaconda to be able to use your own Python version.
For more information about jobfiles, please visit the other pages of our tutorial.

Enjoy working on the Phoenix cluster!

Visual Studio Code

Visual Studio Code

Version

On newer Versions of VS Code problems may come up. Version 1.78 from April 2023 is recommended.