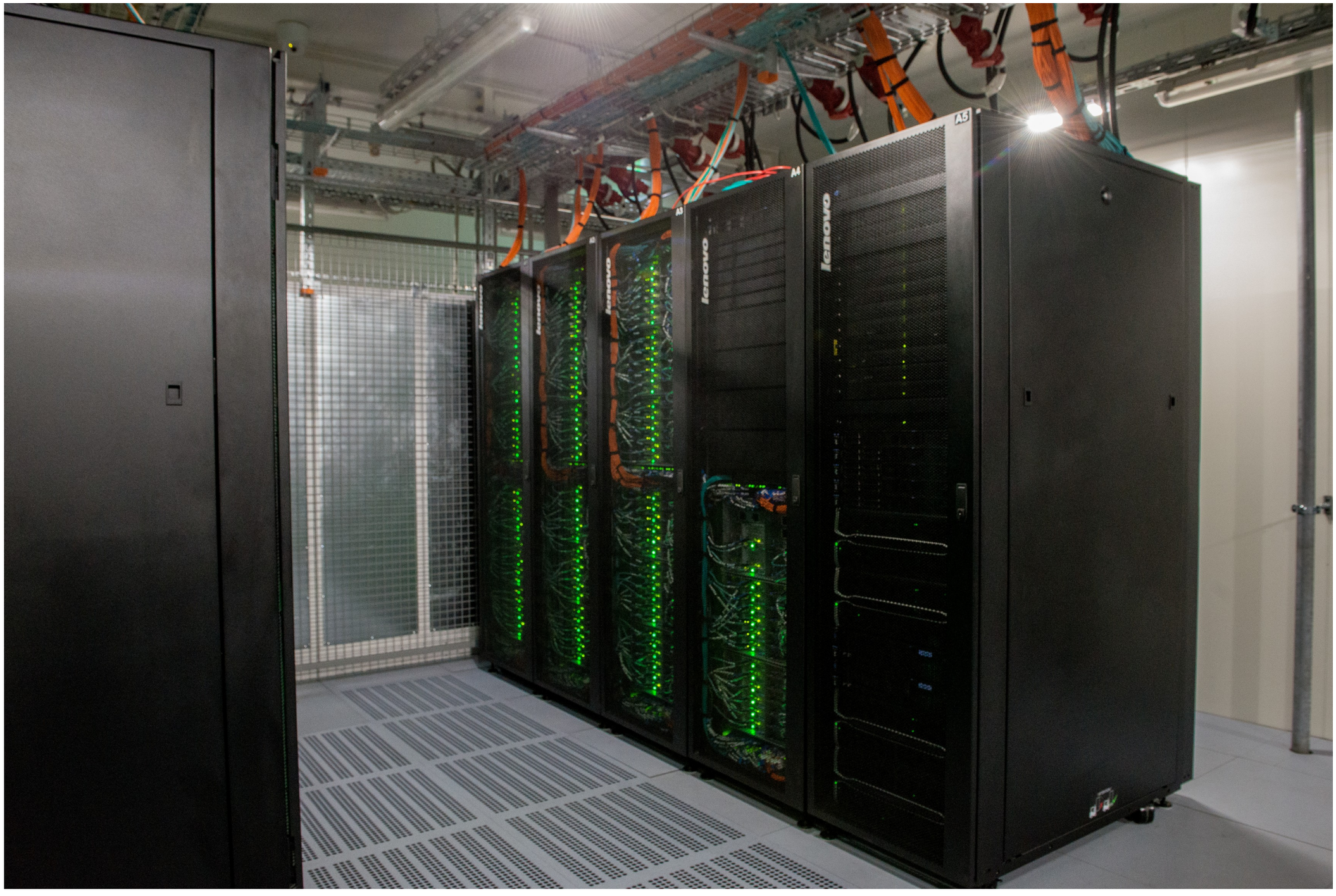
HPC Introduction “Eddy”

M.Sc. Wilke Trei

Forwind Oldenburg

Eddy is here!



Folders and Shares

Login and File System

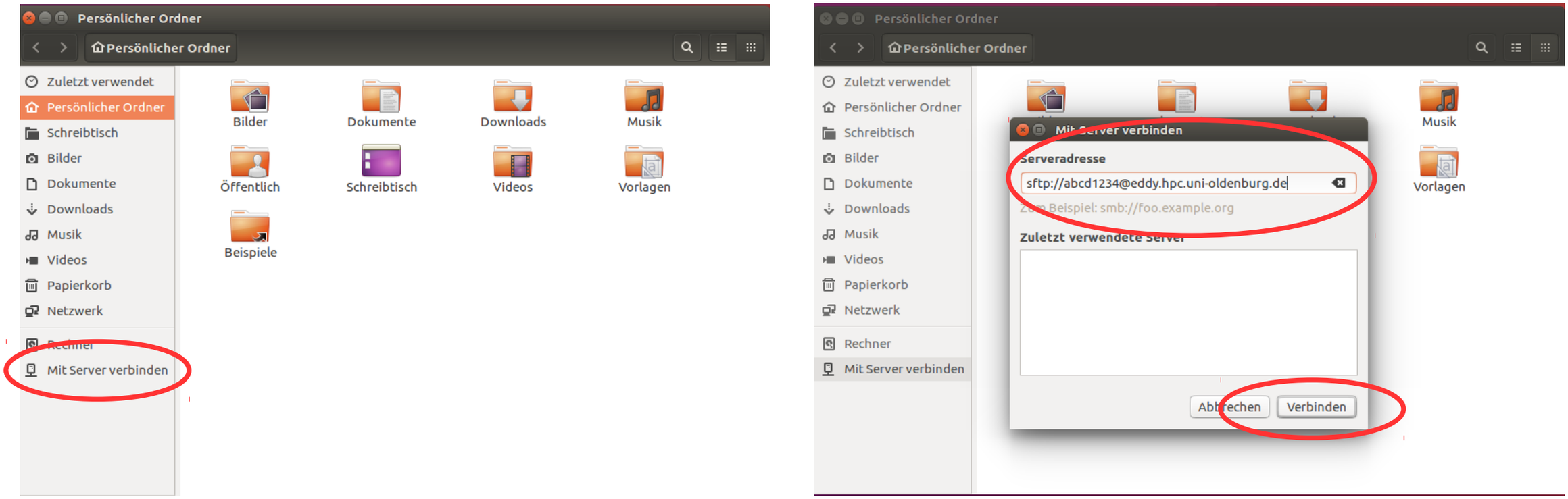
Before (Flow)	Now (Eddy)
ssh -X abcd@flow.hpc.uni-oldenburg.de	ssh -X abcd@eddy.hpc.uni-oldenburg.de

- Your user credentials will remain the same!

Before (Flow)	Now (Eddy)	Quota (Eddy)
/users/fw/abcd1234 /users/iwes/abcd1234	/user/abcd1234	1TB (Fully Backed UP)
/data/work/fw/abcd1234 /data/work/iwes/abcd1234	/gss/data/abcd1234	2TB
/data/work/gpfs/fw/abcd1234 /data/work/gpfs/iwes/abcd1234	/gss/work/abcd1234	2TB (Soft Limit), 20TB (Hard Limit)

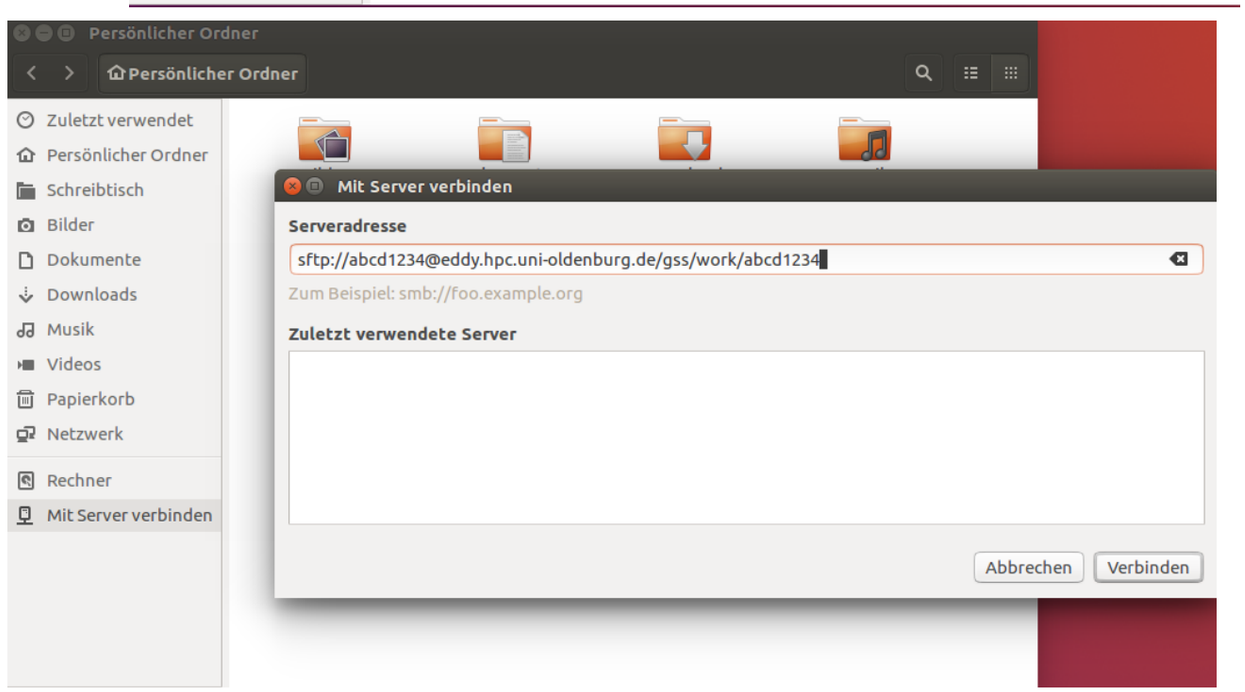
- Quotas can be increased if required by IT services.
- Both – data and work directory – are now located on GPFS file system over Infiniband and all nodes can access this system.

Mounting the File System in Linux - Temporary



Above: Mounting Eddy home folder temporarily

Right Side: Variant for mounting gss



Mounting the Home Folder in Linux

Edit your /etc/fstab (e.g. by calling “sudo gedit /etc/fstab”)

Add a line at the very end:

```
//daten.uni-oldenburg.de/hpchome  <mount_point4home> cifs
workgroup=W2KROOT,username=<user>,file_mode=0600,dir_mode=0700,uid=<l
inux_username>,gid=<linux_group>,noauto,users
```

Now by calling

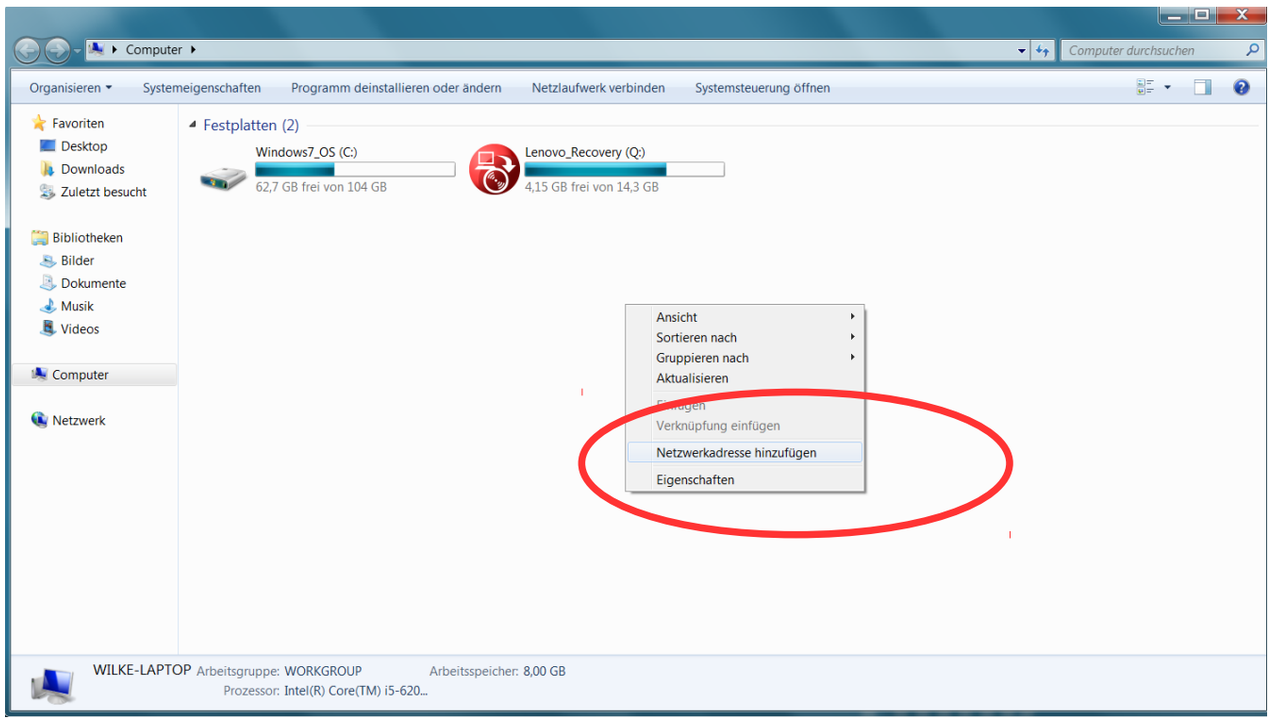
```
mount <mount_point4home>
```

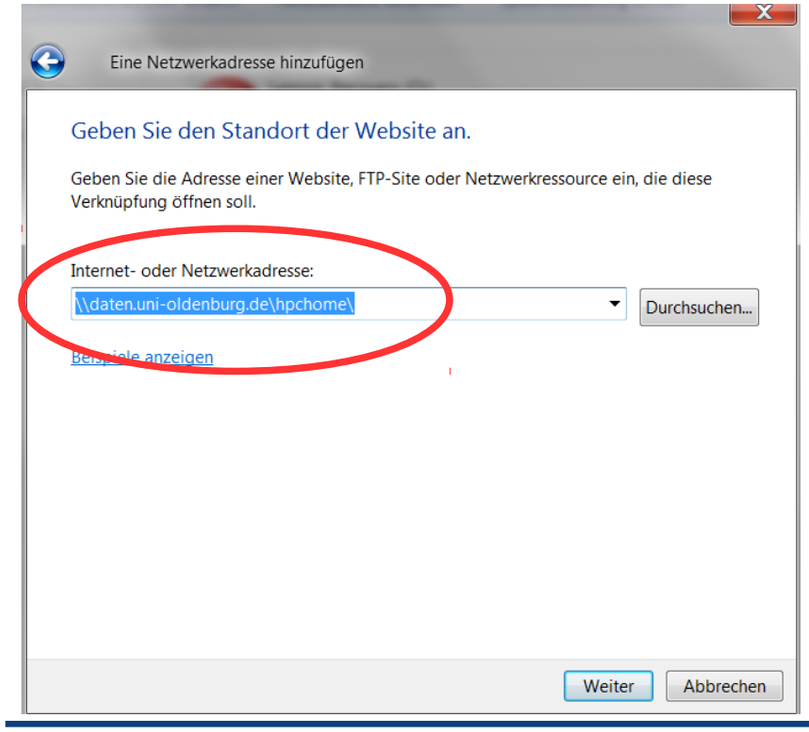
you can mount your folder. This will ask four your Eddy password.

Hints:

- For Ubuntu / Kubuntu => 12.04 you require the package “cifs-utils” in order to mount. This can be installed by calling “sudo apt-get install cifs-utils”
- By adding “,password=” after “username=” and replacing “noauto” by “auto” you can mount the folder on any boot. Please do this only if your computer is secure.
- By replacing “hpchome” by “hpcwork” we will be able to mount the work folder (this is work in progress and will be available at the end of this week)

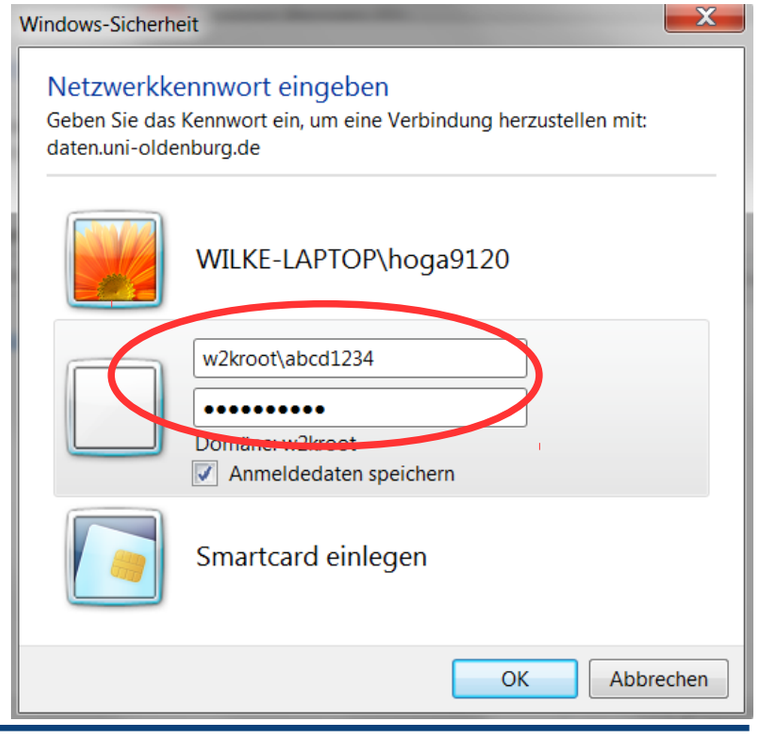
Mounting the File System in Windows





Do not forget using backslashes “\” instead of slashes “/”.

Further when entering your username you have to write it as “w2kroot\abcd1234”, else mounting will not work



Copy your Data from Flow

- Option 1: The old file systems are mounted in Eddy as read only file system during the migration phase.

  Your old shares are linked into your home directory as ~/old_home_abcd1234 and ~/old_data_abcd1234. So you can use the linux command “cp -r” for copying your data.

- Option 2: Use the rsync command for transferring the data. Example:

```
rsync -avz -e ssh $USER@flow.hpc.uni-oldenburg.de:/data/work/gpfs/fw/
$USER/$FOLDER
```

  Advantage: Works with gpfs and in case of a lost connection it resumes correctly

  Disadvantage: For files in your old home and data/work directory the linux copy is quicker.

- Option 3: Use scp … not really an option!

The new Scheduler

The new Scheduler “Slurm” - Commands to Know

Before (Flow)	Now (Eddy)	For which purpose
qfreenodes	sinfo	Get informations about queues and number of free/allocated nodes
qstat -u ‘*’	squeue	Show currently running and queued jobs
qstat -j <jobID>	squeue	Show reason for a job why it is not yet running (On Eddy included to queue list)
qacct -j <jobID>	sacct -j <jobID> -o <format>	Show informations about queued, running or finished jobs. On Eddy we can configure in detail what to show (we will later have a slide for this)
qdel <jobID>	scancel <jobID>	Cancel a running / queued job
qsub <jobScript>	sbatch <jobScript>	Submit a job script. (For Eddy we will later have a slide how these are structured)
	scontrol show partition <partitionName>	Queues are named ‘partition’ in Slrum-slang. This command allows to show detail information about a queue.

The Partitions (Queues) of Eddy

“Partition” is the Slurm name for what we use to call queue. When you specify the partition you want to submit your job to, you have the following options:

Name	Nodes	CPUs per Node	Default Runtime	Maximal Runtime	Default Memory	Maximal Memory (per Node)	Misc
cfdl.p	160	24	2 hours	21 days	2 333 MB	56 000 MB	-
cfdh.p	81	24	2 hours	21 days	5 000 MB	120 000 MB	-
cfdg.p	3	24	2 hours	21 days	10 375 MB	249 000 MB	1x Nvidia Tesla P100 GPU / Node
cfdy.p	10	16	2 hours	2 hours	3 500 MB	56 000 MB	Older Nodes from Flow
eddy.p	241	24	2 hours	21 days	2 333 MB	56 000 MB	-
all_nodes.p	586	24	2 hours	1 day	mixed	mixed	1)

1) The all_nodes.p partition uses all nodes from Eddy and Carl. The nodes for your job will be selected to fit your resources requests. All nodes on Carl have the same properties as the high / low-memory nodes on Eddy except for three special nodes.

The nodes on Carl have 120 000 / 249 000 / 507 000 MB (158 / 128+9 / 30 nodes) of peak memory per node.

Submit a batch job

First of all create a script file for your job. It may include commands just like a normal sh-script, except for the header.

```
#!/bin/bash
#
#SBATCH -J TestJob              # Jobname
#SBATCH -p eddy.p               # Partition: eddy
#SBATCH -t 0-01:00              # time (D-HH:MM)
#SBATCH -o slurm.%j.out         # STDOUT
#SBATCH -e slurm.%j.err         # STDERR
#SBATCH --ntasks 192            # Number of Tasks

module load intel2016b # Some meta-package for Intel
                       # Compiler & Intel MPI

cp somePath/someFile ~/myfile
SomeSerialApplication myfile

mpirun -np 192 someParallelApplication
```

This part will be used immediately to schedule your job and to assign a job id.

When your job starts, a single copy of the sh script will be run on the first granted CPU.

Here we can do the needed serial preparations and finally run the parallel part of the job.

Afterwards, submit your job. We assume its named job.sh.

```
sbatch ./job.sh
```

Thats it!

Notes for sbatch

- The script submitted behaves like any serial sh script that is run on an empty node. It is allowed to do all preparations for your parallel task in there.
- By default the directory your nodes switch to, is the one where the .sh script was at the time sbatch was called.
- When any mpi library is loaded and you do not need any further special treatment of your task (i.e. it is ok that when you want to run n tasks they just use the first n granted CPUs), you just can use mpirun to run your application in parallel. Different to “Flow” you do not need parallel environments any more.
- When you need special treatment, e.g. you want to run an MPI job where each task uses itself a OpenMP parallelism, then load Intel MPI or Open MPI and then use the command srun. If an MPI is loaded, srun (which just replaces mpirun) replaces mpirun but it is able to consider the settings for given at the Top of the batch file, e.g. “--cpus-per-task”.
- When your parallel job ends, it is returned to the single core batch execution. This generally allows to call mpirun multiple times within one batch file or do some post-preparation before the job terminates.

Parameters of “srun” and “sbatch”

The parameters determine the resources one requests for the job:

-J SomeText	Sets the job-name to “SomeText”, this will be displayed in queue.
-p <somePartition>	Selects a partition the job will be submitted to, multiple partitions can be selected by separating multiple partition names by “,”.
-t D-HH:MM -t HH:MM:SS	Sets the time the job expects to need. The job will be canceled as this time expires. Default for CFDH and CFDL is 2 hours.
-o someFilename1 -e someFilename2	Sets the filenames for output (-o) and error-console (-e)
--ntasks <num>, or -n <num>	Sets the number of tasks to be run. Either this or “-N” and “--tasks per node” must be used.
--cpus-per-task <num>	Specifies the number of CPUs for each task. This is useful e.g. for OpenMP parallelism. The default used is 1, the maximum is 24 on cfdl / cfdh.
--mem-per-cpu <num>	Memory each CPU in your tasks may use. The defaults are 2.3 gb (cfdl) and 5 gb (cfdh). If you do not need more, this is optional.
--exclusive	Specifies that your job wants to use a node exclusively. Useful for most parallel cfd-applications e.g. OpenFoam, Palm, WRF. Was default behavior on FLOW.
--gres=gpu:1	Requests a graphics card for your job. This is required for running CUDA / OpenCL jobs on the GPU nodes. Without this no access to the graphic cards drivers will be granted.

Slurm environment variables

When a job is entering the queuing system, there are certain environment variables set by slurm that may help controlling the job.

Variable	Description
$SLURM_JOBID	The id of the job
$SLURM_JOB_NAME	The name of the job
$SLURM_SUBMIT_DIR	The directiory path where the submitted script was placed
$SLURM_JOB_NODELIST	A list of all nodes granted for the job
$SLURM_NODEID	The index relative to $SLURM_JOB_NODELIST where the current task runs on
$PBS_O_VNODENUM	Index to CPU core running on within the current node
$SLURM_PROCID	Index of the current task

Note that these variables are set when the job starts its execution. Therefore this can not be used to change the output filenames in the header of your job script. In order to do so, you can use %j (job-ID), %N (node name of the jobs first node), %u (user Name).

See the example two slides before how to use these symbols.

Submitting Array Jobs

An array job is a job that contains a fixed set of tasks that are almost identical but may use different parameters or data. Also it is required for the tasks to be independent, since it is not guarantied that all tasks run at the same time.

By using sbatch it is rather easy to run an array job by adding

```
#SBATCH --array<someNumbers>
```

to your job script. This will cause the sh script to be scheduled in multiple copies with each of the copies having an array-job-id that is dependent on what was specified in <someNumbers>. Each copy executes the sh script in the first instance granted to the corresponding copy and thus each copy behaves like a normal job.

Within the script and within each program executed, there are two special environment variables available for getting the job-id of the first array task ($SLURM_ARRAY_JOB_ID) and the number of the task that is currently executed (SLURM_ARRAY_TASK_ID).

There are multiple options for <someNumbers>. We give some examples:

```
#SBATCH -array0-31      # $SLURM_ARRAY_TASK_ID will take all numbers from 0 to 31
#SBATCH -array1,2,4,8   # $SLURM_ARRAY_TASK_ID will take numbers 1,2,4 and 8
#SBATCH -array0-8:2     # $SLURM_ARRAY_TASK_ID will take number 0,2,4,6,8
```

Getting Job Execution details

If you call “sacct –helpformat” you will get displayed a set of options which can be attached to sacct to select the information about your running job you like to see.

This are the first lines:

```
sacct --helpformat

AllocCPUS          AllocGRES          AllocNodes         AllocTRES

Account            AssocID            AveCPU             AveCPUFreq ...
```

Now you can run sacct -j <jobID> -o Field1,Field2,… in order to show the information.

One example:

```
sacct -j 21665 -o JobID,User,Partition,Submit,Start,End,TimeLimit
       JobID      User  Partition              Submit               Start                 End  Timelimit
------------ --------- ---------- ------------------- ------------------- ------------------- ----------
21665         erle1100     cfdh.p 2017-02-28T09:54:36 2017-02-28T15:59:26 2017-02-28T16:02:54   01:00:00
```

Using SRUN

There is a second way of submitting jobs to a queue by the command “srun”.

Srun submits a job that just runs one single command. The parameters of srun are identically to those of sbatch, but must be attached to the command directly instead of writing them into a file.

Getting an Interactive Shell

There is one more feature of srun. When run with parameter --pty, it will not return directly, but forward the command line output of the first run task directly to the users shell.

This allows to get an interactive shell on one of the nodes by calling

```
srun -n 1 -p eddy.p --pty bash
```

In case that there is no free node, your request will be scheduled, and the shell will become active when the job starts.

Note that when running this interactive shell, you have to call

```
module load hpc-uniol-env
```

within the interactive shell first in order to use the module system, because it deactivates when switching the shell.

Software on Eddy

Good news – most commands remain the same

```
module avail

---------------------- /cm/local/modulefiles -----------------------
   cluster-tools/7.3    freeipmi/1.5.2     module-git     openldap
   cmd                  gcc/6.1.0          module-info    shared
   dot                  ipmitool/1.8.17    null

------------------ /cm/shared/uniol/modules/core -------------------
   hpc-uniol-env (L)    slurm/current (L)

------------------- /cm/shared/uniol/modules/bio -------------------
   BCFtools/1.3.1                  FastQC/0.11.5
   BEDTools/2.26.0                 IGV/2.3.80
   BLAST/2.2.26                    IGVTools/2.3.75
   BLAST/2.6.0              (D)    Lighter/1.0
   BLAT/3.5                        Lumpy/0.1
   BamTools/2.4.0                  SAMtools/1.3.1
   BioPerl/1.7.0                   Stacks/1.42-goolf-5.2.01
   Biopython/1.68                  Trinity/2.2.0


… (this is a very long list…)
```

Some useful commands for modules

Command...	...and what will happen
module avail	Show a list of all available (visible) modules
module load <moduleName>/<version>	Loads a module in a specific version as well as all of its dependencies.
module load <moduleName>	Loads the most recent version of a module. If there are multiple version the one loaded by default is marked with a (D) when calling module avail.
module list	Lists the currently loaded modules.
module unload <moduleName>	Unloads the specified module. For most modules also all dependencies will be unloaded.
module restore	Resets all loaded modules, such that only the default modules remain loaded.
Module spider <moduleName>	Shows detailed information as a description string and dependencies of a module.

You want to compile something own? Use Toolchains!

Toolchains on Eddy = Compiler + MPI + Math Libraries (FFT, Linear Algebra)

Intel/201xb: Compiler ICC & IFORT (C/C++, Fortran), IntelMPI, IKML Math Library. Versions available: 2016 and 2013

goolf/a.b.c gcc & gfortran (Version a.x), Open MPI (Version b.c), LaPack, FFTW, OpenBlas. Versions of gcc: 4.8, 5.4 and 6.2

CUDA/a gcc 6.2, Nvidia Cuda compiler, OpenCL runtimes, OpenMPI, CUDA-FFT and CUDA-Blas. The libraries are using GPU.

Basic Introduction: WRF

WRF / WPS Versions installed on Eddy

```
WRF/3.6.1       WPS/3.6.1
WRF/3.7.1       WPS/3.7.1
WRF/3.8.0       WPS/3.8.0
WRF/3.8.1 (D) WPS/3.8.1 (D)
```

Good to know:

- There is one more package “WRFGeodata/all”, which sets the environment variable **$WRF_GEO_DATA_DIR** to a directory, where the “geog” data of WRF (landuse, soiltype, topo) in different resolutions is located on the cluster
- The “/data/vault” directory of FLOW can be found in “**/vault**” on Eddy. Reading access is now given for all users of the cluster, so you do not have to ask for an extra permission to use the data (but to modify it).

Basic setup of WRF

- Load your desired WRF and WPS version. Note that the WPS module already loads the fitting WRF version, so “module load WPS/myVersion” is sufficient
- Make a directory where you want to run your simulation and change to it

```
mkdir /gss/work/abcd1234/newDirectioryName
cd /gss/work/abcd1234/newDirectioryName
```

- Run the two setup scripts that will create links to the loaded WRF / WPS folders (just as it was on Flow)

```
sh setup_wrf_dir.sh
sh setup_wps_dir.sh
<If you want the script to setup your namelist, load WRFGeodata before>
```

- Prepare your WRF-run in the folder. Afterwards create a job-script (you can use gedit on the cluster) in the folder. An example is on the next slide.
- Submit your job and enjoy
- Anything not working? Call your admin (me ;))

Example WRF / WPS jobscript

```
#!/bin/bash
#
#SBATCH -J HappyJob           # Jobname
#SBATCH -p eddy.p             # Partition: eddy
#SBATCH -t 1-00:00            # time (D-HH:MM), here one day
#SBATCH -o wrf.%j.out         # STDOUT
#SBATCH -e wrf.%j.err         # STDERR
#SBATCH --ntasks 192          # Number of Tasks (here: 8 Nodes)
#SBATCH --mem-per-cpu 2333    # Maximal Memory for Low-Mem Nodes: 2.333G
#SBATCH --exclusive           # We want the nodes for our own


module load WPS/3.8.1         # Is the same as loading WRF & WPS.
                              # A fitting MPI will be loaded as well.
mpirun -np 48 real.exe        # We can do pre- & post-processing in one Job!

mpirun -np 192 wrf.exe        # Here we go
```

Afterwards, submit your job. We assume its named run_wrf_wps.sh.

```
sbatch ./run_wrf_wps.sh
```

Thats it!

Basic Introduction: PALM

PALM is not installed on Eddy :(But...

… you can get it on your own:

- Load the desired version of PALM via svn to your personal computer (Eddy itself has no svn installed yet)
- Copy the PALM code to /user/abcd1234/palm or /gss/work(or data) /abcd1234/palm or wherever you want
- Create a folder /user/abcd1234/job_queue
- In the HPC Wiki you can find the page

https://wiki.hpcuser.uni-oldenburg.de/index.php?title=Install_PALM_on_Eddy

From there you can download a packed file containing a mrun.config, mrun, mbuild and subjob scripts that are fitted for the use with Eddy. Copy them to your PALM directory (mrun.config) or its /trunk/SCRIPTS directory (the other ones).

- Continue on next slide...

A long setup part 2

- Use “gedit ~/.bashrc” to edit your bashrc file. Add two lines:

```
export PALM_BIN=/path_to_your_palm/current_version/trunk/scripts
export PATH=$PALM_BIN:$PATH
```

- Log in and out again (or source the .bashrc) to make the changes active
- Modify your mrun.config – put the path to your palm directory in lines 6,12, 56 and 57 and your user name in line 55
- Build PALM by calling

```
mbuild -u -h lceddy
mbuild -u lceddy
```

- Prepare your job and run it with mrun, just as you are used it to do :)

Note: When calling mrun the allowed queues (-q) are “cfdh.p”, “cfdl.p”, “eddy.p” and “all_nodes.p”.
Furthermore you must set the option -b (for running a batch job)

Notes regarding PALM

- We currently have a bug in PALM, which prevents us for running jobs with cyclic boundary conditions taking from a pre-run.

  The palm support is currently looking into this, but we are also trying to find a solution. Normal jobs and restart-jobs are not affected by this bug.

- It is currently not possible to do remote calculations (coupling PALM on Eddy with your local installation). We will try to install this feature ones all work on the firewalls are completed.

Basic Introduction:

OpenFoam

OpenFOAM Versions installed on Eddy

```
OpenFOAM/2.1.1          OpenFOAM/4.0
OpenFOAM/2.3.1          OpenFOAM/4.1          (D)
OpenFOAM/2.4.0          OpenFOAM/4.1-se
OpenFOAM/3.0.1          OpenFOAM-Extend/3.2
```

Good to know:

- By loading OpenFOAM also the compiler it is build with as well as a fitting MPI distribution will be loaded automatically
- Currently the foam_bash will not be sourced when loading the module (because the new module system does not like it yet). But there is an environment variable $FOAM_BASH making live easy.
- OF versions < 3.0 are build with gcc 5.4, all later versions with the newest Intel compiler.

  From the first experiments: gcc versions are not slower then on Flow, but Intel compiled versions are much quicker!

Example OpenFOAM jobscript

```
#!/bin/bash
#
#SBATCH -J HappyJob           # Jobname
#SBATCH -p eddy.p             # Partition: eddy
#SBATCH -t 1-00:00            # time (D-HH:MM), here one day
#SBATCH -o foam.%j.out        # STDOUT
#SBATCH -e foam.%j.err        # STDERR
#SBATCH --ntasks 192          # Number of Tasks (here: 8 Nodes)
#SBATCH --mem-per-cpu 2333    # Maximal Memory for Low-Mem Nodes: 2.333G
#SBATCH --exclusive           # We want the nodes for our own

module load Armadillo/7.500.1   # Some of us require this, others not
module load OpenFOAM/4.1      # A fitting MPI will be loaded automatically.
source $FOAM_BASH             # Sourcing is currently required

decomposePar                  # We can do pre- & post-processing in one Job!
mpirun -np $SLURM_NTASKS pimpleFoam -parallel > ./log.pimpleDyMFoamB.dat
                              # here we go
```

Afterwards, submit your job. We assume its named run_wrf_wps.sh.

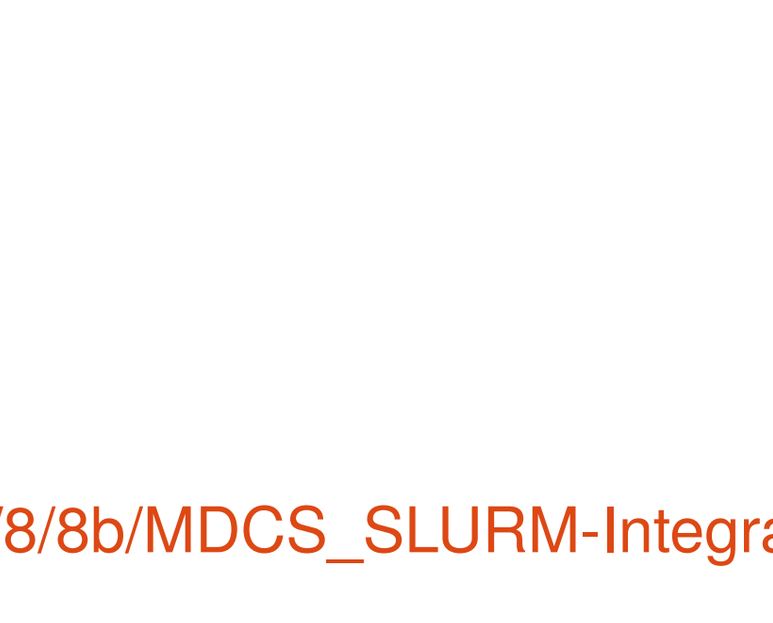
```
sbatch ./run_of.sh
```

Thats it!

Basic Introduction: Matlab

Preparing Matlab – First Steps

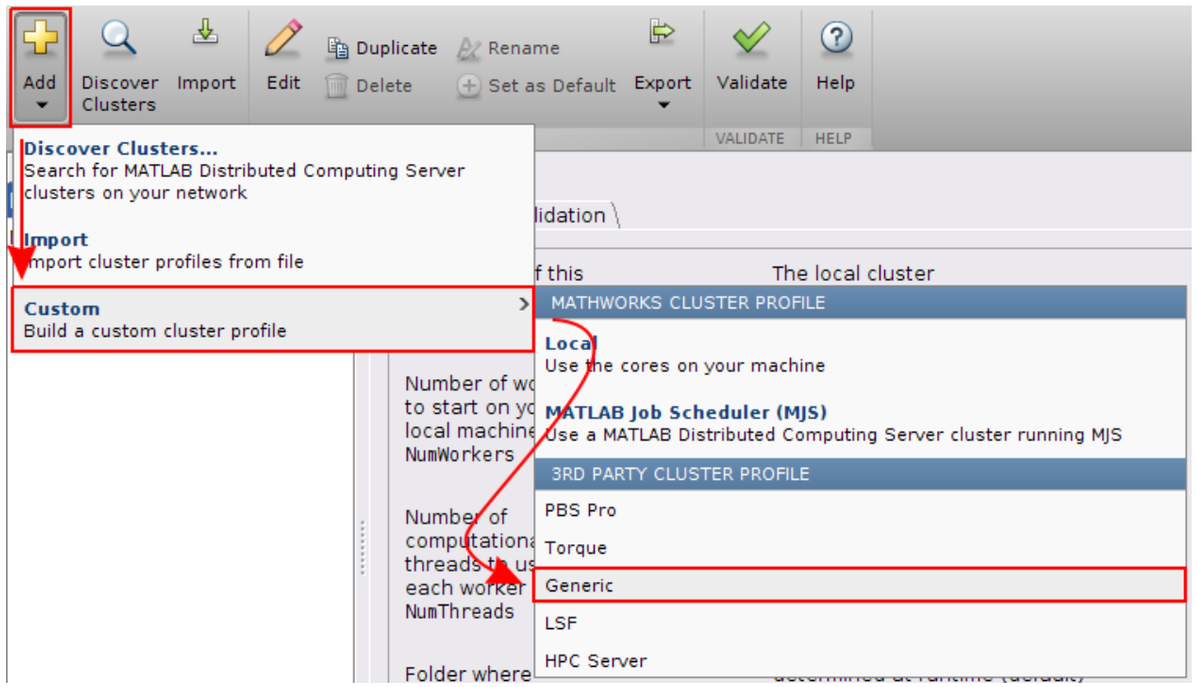
- Download Matlab Configuration Files from https://wiki.hpcuser.uni-oldenburg.de/images/8/8b/MDCS_SLURM-Integration_R2016b.zip
- Unzip the file and copy all contained files to <matlabroot>/toolbox/local/
- Restart Matlab (or start it, if it was not running before)
- Next: Open the Cluster Profiles Manager in your local Matlab Installation.



(This Instruction is also available in the HPC-WIKI under https://wiki.hpcuser.uni-oldenburg.de/index.php?title=Configuration_MDCS_2016)

Preparing Matlab – Next Steps

- In the new window click on Add --> Custom --> Generic. A notification will pop up, confirm with OK.



Now enter the following information (from top to bottom):

- Description: Eddy
- JobStorageLocation:

  /home/USERNAME/MATLAB/R2016b/JobData

  C:\Users\USERNAME\Documents\MATLAB\R2016b\JobData

- In case it does not exist the directory has to be created. Note, that it is strongly recommended to use different directories for different versions of Matlab.
- NumWorkers: 36 (this is the maximum number of workers per job)
- ClusterMatlabRoot: where Matlab is installed on the cluster, enter the following:

  /cm/shared/uniol/software/MATLAB/2016b

Preparing Matlab – Next Steps II

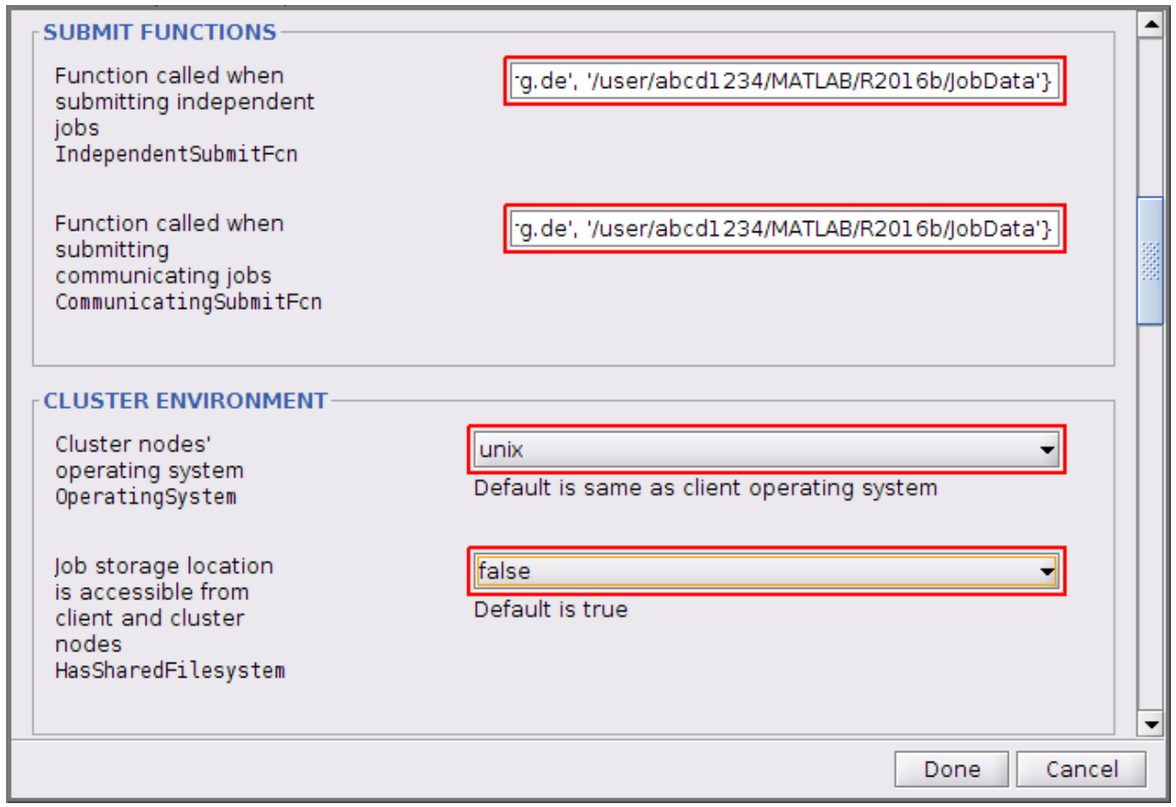
- Skip to Submit Functions and enter for **IndependentSubmitFcn**:

```
{@independentSubmitFcn, 'eddy.hpc.uni-oldenburg.de',
'/user/abcd1234/MATLAB/R2016b/JobData'}
```

and for the **CommunicatingSubmitFcn**:

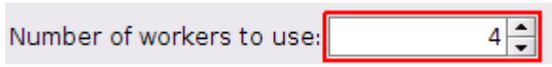
```
{@communicatingSubmitFcn, 'eddy.hpc.uni-oldenburg.de',
'/user/abcd1234/MATLAB/R2016b/JobData'}
```

- **OperatingSystem**: set to 'unix'
- **HasSharedFilesystem**: set to 'false'
- Skip ahead to Jobs and Task Functions and enter for **GetJobStateFcn**: @getJobStateFcn and for **DeleteJobFcn**: @deleteJobFcn
- Click on Done.

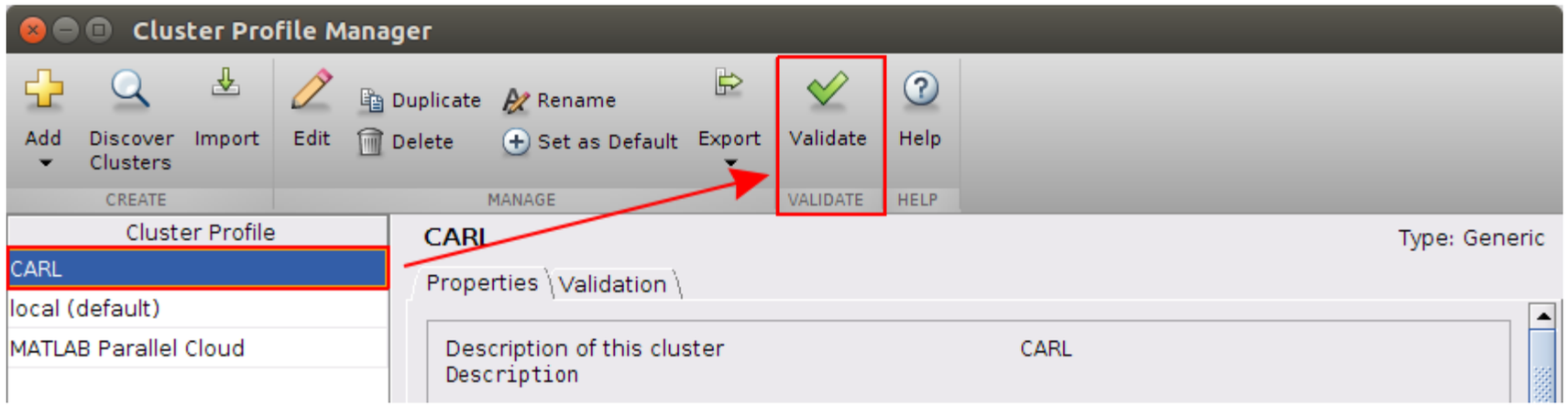


Checking if everything is OK

- In the cluster profile manager the properties of the profile should be displayed. Switch to the 2nd tab “Validation” and specify a number of workers for validation. 4 is good for testing (more causes the test to last longer)



- Click on “Validate” to run the validation



- In a few minutes the test should be completed. The last test “Parpool” may fail if you did not deactivate it on the first tab, but that is no problem for most use cases.

	Stage	Status	Description
☑	Cluster connection test (parcluster)	Passed	
☑	Job test (createJob)	Passed	
☑	SPMD job test (createCommunicatingJob)	Passed	
☑	Pool job test (createCommunicatingJob)	Passed	
☐	Parallel pool test (parpool)	Skipped	Not included in validation.

Lets say “Bye” to Flow and Hero