



bw | HPC – C5

Batch System – Best Practices

Robert Barthel, SCC, KIT



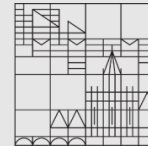
UNIVERSITÄT
HEIDELBERG
ZUKUNFT
SEIT 1386

Hochschule
für Technik
Stuttgart



Hochschule Esslingen
University of Applied Sciences

Universität
Konstanz



UNIVERSITÄT
MANNHEIM



Universität Stuttgart

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN



KIT
Karlsruher Institut für Technologie



ulm university universität
uulm

Funding:



Baden-Württemberg

MINISTERIUM FÜR WISSENSCHAFT, FORSCHUNG UND KUNST

www.bwhpc-c5.de

How to read the following slides

Abbreviation/Colour code	Full meaning
<code>\$ command -opt value</code>	<code>\$</code> = prompt of the interactive shell The full prompt may look like: <code>user@machine:path \$</code> The command has been entered in the interactive shell session
<code><integer></code> <code><string></code>	<code><></code> = Placeholder for integer, string etc
<code>foo, bar</code>	Metasyntactic variables
<code>\${WORKSHOP}</code>	<code>/pfs/data1/software_uc1/bwhpc/kit/workshop/2018-04-11</code>

- Most information given by this talk can be found at <http://bwhpc-c5.de/wiki> under the article:
 - Batch_Jobs

Where to get the slides and exercises?

- http://indico.scc.kit.edu/indico/e/bwhpc_course_2018-04-11 or `uc1:/pfs/data1/software_uc1/bwhpc/kit/workshop/2018-04-11`

- Slides
- Exercises

The screenshot shows a web interface for an event. On the left, there is a navigation menu with the following items: 'Überblick / Overview' (highlighted), 'Agenda', 'Registrierung / Registration', and 'Formular / Form'. On the right, there is a main content area with text in German and English. At the bottom right of the content area, there is a button labeled 'Slides exercises' with a downward arrow icon. Two red arrows originate from the 'Slides' and 'Exercises' items in the list on the left and point to the 'Slides exercises' button. A black arrow points from the URL above to the top of the screenshot.

Job Scripting Basics

Job script: Structure (1)

```
#{WORKSHOP}/exercises/03/01_job.sh
```

```
#!/bin/bash
```

```
#MSUB -l nodes=1:ppn=1
```

```
#MSUB -l walltime=00:01:00
```

```
#MSUB -l pmem=50mb
```

```
#MSUB -q develop
```

```
#MSUB -N serial-test
```

```
#MSUB -m bea
```

```
#MSUB -M my_email_address
```

```
#MSUB -A workshop
```

```
#MSUB -l advres=bwhpc-workshop.64
```

```
printenv
```

Header
(Interpreter)

Declarations
(Resource
requirements,
Notification
options...)

Main section
(Execution part)

Job script: Structure (2)

- Be descriptive! Comment your script code

```
`${WORKSHOP}/exercises/03/02_job.sh
```

```
#!/bin/bash

#MSUB -l nodes=1:ppn=1
#MSUB -l walltime=00:01:00
#MSUB -l pmem=50mb
#MSUB -N serial-test
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop.64

# Printing all environment variable
printenv
```

Job script: Structure (3)

- Print out what your script is doing!

```
`${WORKSHOP}/exercises/03/03_job.sh
```

```
#!/bin/bash

#MSUB -l nodes=1:ppn=1
#MSUB -l walltime=00:01:00
#MSUB -l pmem=50mb
#MSUB -N serial-test
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop.64

# Printing all environment variable
echo "Current available env. vars"
printenv
```

Job script: Structure (4)

■ Use Global Environment Variables!

```
`${WORKSHOP}/exercises/03/04_job.sh
```

```
#!/bin/bash
```

```
#MSUB -l nodes=1:ppn=1
```

```
#MSUB -l walltime=00:01:00
```

```
#MSUB -l pmem=50mb
```

```
#MSUB -N serial-test
```

```
#MSUB -A workshop
```

```
#MSUB -l advres=bwhpc-workshop.64
```

```
# Printing job resources
```

```
echo "Number nodes = `${MOAB_NODECOUNT}`"
```

```
echo "Number cores = `${MOAB_PROCCOUNT}`"
```

```
What is proc?
```


Excursus: Ressource specifications (1)

■ @ MOAB

■ Node

= computer server

■ Proc

= „processors“ → Cores

■ Task

= *atomic collection of resources, such as processors, memory, or local disk, which must be found on the same node*

■ @ OpenMP

■ Thread

= smallest sequence of instructions managed independently by a OS scheduler

→ Normally 1 thread is pinned to 1 core

■ @ MPI

■ Task

= is 1 process excuted by the OS

→ smallest useful unit: 1 MPI task per node

Examples:

1) Addressing 2 cores on 1 node:

```
msub -1 nodes=1:ppn=2
```

2) two cores on nodes 1,2 + four cores on node 3:

```
msub -1 nodes=2:ppn=2+1:ppn=4
```

Excursus: Ressource specifications (2)

■ @ MOAB

■ mem

= Working memory for total job

■ pmem

= Working memory per cores

■ Node access policies

bwUniCluster:

shared (*with other users' jobs*)

how to get whole node? → e.g.:

```
msub -l nodes=1:ppn=16 or
```

```
msub -l naccesspolicy=singlejob
```

bwForClusters:

shared, **singleuser** (*=shared with other own jobs*),

singlejob (*=exclusively for your job*)

ForHLR:

singlejob

→ Addressing not max cores/mem per node is wasting resources!

Example:

Addressing 4GB for 2cores on 1 node:

```
msub -l nodes=1:ppn=2, pmem=2gb
```

or

```
msub -l nodes=1:ppn=2, mem=4gb
```

Job script: Structure (5)

- Use Bash Scripting (assign variables, use conditionals, functions)

```
`${WORKSHOP}/exercises/03/05_job.sh
```

```
#!/bin/bash

#MSUB -l nodes=1:ppn=1
#MSUB -l walltime=00:01:00
#MSUB -l pmem=50mb
#MSUB -N serial-test
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop.64

# Check existance of input
my_input=${HOME}/input
if [ ! -e ${my_input} ] ; then
    echo "ERROR: ${my_input} does not exist"
    exit 1
fi
```

Excursus: Bash Scripting (1)

■ Referenz for intro: <http://tldp.org/HOWTO/Bash-Prog-Intro-HOWTO.html>

■ Script **line** ends with no special character!

■ But multiple statements in one line to be separated by:

;

Semicolon

```
echo hello; echo World; echo bye
```

■ Special characters, excerpt:

\$, {} → addressing contents of variables

```
my_var=hello ; echo ${my_var}
```

→ commenting out lines

' → full quotes, preserves all chars

" → partial quotes, does not preserve e.g. \$

\ → escaping (special) characters

```
echo '${my_var}'  
echo "${my_var}"  
echo "\${my_var}"
```

■ Expansion wildcards (= globbing), excerpt:

* → any multiple character

```
ls ${HOME}/.b*
```

Excursus: Bash Scripting (2)

■ Output & Input Redirection

Syntax	Does?	Examples
<code>exe > log</code>	Standard output (stdout) of application exe is (over)written to file log	<code>\$ date > log; cat log</code>
<code>exe 2> err</code>	Standard output (stderr) of application exe is (over)written to file err	<code>\$ date 2> err; cat err</code>
<code>exe1 exe2</code>	Passes stdout of exe1 to standard input (stdin) of exe2 of next command	<i># Print stdout & stderr to screen and then append both to file</i> <code>\$ date 2>&1 tee -a log</code>
<code>exe < inp</code>	Accept stdin from file inp	<code>\$ wc -l < file</code>

■ (Conditional) Tests:

- Existence of files, directories, success of last command

```
if [ -e file ] ; then
    echo "file exists"
fi
```

```
if [ -d dir ] ; then
    echo "DIR exists"
fi
```

```
date -x
if [ $? -ne 0 ] ; then
    echo "ERROR"
fi
```

Job script: Structure (6)

- Use modulefile statements!

```
`${WORKSHOP}/exercises/03/06_job.sh
```

```
#!/bin/bash

#MSUB -l nodes=1:ppn=1
#MSUB -l walltime=00:01:00
#MSUB -l pmem=50mb
#MSUB -N serial-test
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop.64

# Unload all modules
module purge
# Load gnu compiler
module load compiler/gnu
# Print loaded modules
module list 2>&1
```

Job script: Structure (7)

- Use templates for job scripts
 - Provided by many installed software packages
 - cf. help description of SW

Example:

- How to get? Search in description for example directory, or e.g. Turbomole

```
$ module show chem/turbomole 2>&1 | grep "EXA_DIR"
```

→ shows path:

```
/opt/bwhpc/common/chem/turbomole/7.2.1_tmolex432/bwhpc-examples
```

```
bwHPC_turbomole_single-node_tmpdir_example.sh
```

```
#!/bin/bash
```

```
## Purpose: Turbomole JOB example script for bwHPC, such as bw{For,Uni}Cluster
```

```
##           for   S I N G L E   N O D E   runs   O N L Y
```

```
...
```

```
...
```

Common Issues

Best Practises – Job setup (1)

■ Directory:

- Running your code/application/job in `${HOME}` is not permitted

Valid destinations are:

1. Workspaces (`ws_allocate`)
2. `$TMPDIR` (not suitable for multinode jobs)

■ Issues:

■ Workspaces:

- Does not handle well codes producing Tbyte of scratch files and more then 10000 files. Solution: [Change your application code](#), Apply for Tiger Team Support.

■ `$TMPDIR`:

- Requires job script setup to handle data transfer:

```
#!/bin/bash
#MSUB ...
cp -pr ${MOAB_SUBMITDIR}/<file> ${TMPDIR}
...
cp -pr ${TMPDIR}/<results>* ${MOAB_SUBMITDIR}
```

Best Practises – Job setup (2)

■ Directory: (cont.)

■ Potential problems:

- During job run, binaries/inputfiles etc not found

→ give full path to binaries/inputfiles

→ change DIR in jobscript

```
#!/bin/bash
#MSUB ...
# Change to job submission directory
cd ${MOAB_SUBMITDIR}
```

■ Record resource usage to optimise resource requests

■ Walltime:

```
#!/bin/bash
#MSUB ...

# Record runtime of executed program
time ./program
```

Best Practices: Job Submit

- msub issues:

- msub ***your_script -x argument***

→ msub will interpret `-x` as an own option

- Solution:

A: Submit wrapper script:

```
#!/bin/bash
your_script -x argument
```

- msub options

- Do not use „procs“

```
msub -l procs=2,...
```

- Memory assignment:

→ mem vs. pmem

```
msub -l procs=8,pmem=4gb
msub -l procs=8,mem=4gb
```

Exercise 1

TASK/ToDo: 10 min

- Generate workspace named „workshop“ + subfolder „exercise_1“
- Generate file „ws_exercise.inp“ with content under that subfolder
- Generate job script (1 core, 70mb per core, 10 sec) starting the script from your workspace, checking existence of „ws_exercise.inp“, if exists print content of „ws_exercise.inp“
- Submit script to queueing system + watch progress

Exercise 1

TASK/ToDo: Solution

```
$ ws_allocate workshop 10
$ cd $(ws_find workshop)
$ mkdir exercise_1; cd exercise
$ echo -e "blah\nblah" > ws_exercise.inp
```

```
#!/bin/bash
#MSUB -l nodes=1:ppn=1
#MSUB -l pmem=70mb
#MSUB -l walltime=00:00:10

my_workspace=$(ws_find workshop)
my_file=${my_workspace}/exercise_1/ws_exercise.inp
if [ -e ${my_file} ] ; then
    cat ${my_file}
fi
```

Parallel Jobs

Submitting parallel jobs (OpenMP)

■ Example:

```
`${WORKSHOP}/exercises/03/parallel/omp.sh
```

```
#!/bin/bash
#MSUB -l nodes=1:ppn=2
#MSUB -l walltime=00:01:00
#MSUM -l pmem=50mb
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop.64

# Set executable name to variable
exe=./hello_omp
# Load modules
module load compiler/intel

# Setup OpenMP environment variable
export OMP_NUM_THREADS=${MOAB_PROCCOUNT}

# Printout number of threads
echo "No.threads = ${OMP_NUM_THREADS}"

# Execute program
${exe}
```

■ Shared memory restricts to 1 node.

■ Do not define number of threads explicitly. Use MOAB variables.

Submitting parallel jobs (MPI) (1)

■ Example:

```
${WORKSHOP}/exercises/03/parallel/mpi.sh
```

```
#!/bin/bash
#MSUB -l nodes=1:ppn=2
#MSUB -l walltime=00:01:00
#MSUM -l pmem=50mb
#MSUB -A workshop
#MSUB -l advres=bwhpc-workshop.64

# Set executable name to variable
exe=./hello_mpi
# Load modules
module load compiler/intel mpi/impi

# Printout number of tasks
echo "No.MPI tasks = ${MOAB_PROCCOUNT}"

# Spawn for each core 1 MPI task
mpirun ${exe}
```

■ (For computations on more than 1 node use queue multinode!)

■ The corresponding MPI module has to be loaded on the compute nodes.

■ Use mpirun to execute the binary.

Submitting parallel jobs (MPI) (2)

- Spawning only 1 MPI task per node

- Example:

```
${WORKSHOP}/exercises/03/parallel/mpi_v2.sh
```

```
#!/bin/bash
#MSUB -l nodes=2:ppn=2
#MSUB -l walltime=00:01:00
#MSUM -l pmem=50mb

# Set executable name to variable
exe=./hello_mpi
# Load modules
module load compiler/intel mpi/impi

# Printout number of nodes = MPI tasks
echo "No.MPI tasks = ${MOAB_NODECOUNT}"

# Spawn only one MPI task per node
mpirun -print-rank-map -perhost 1 ${exe}
```

- (For computations on more than 1 node use queue multinode!)

- The corresponding MPI module has to be loaded on the compute nodes.

- Use mpirun to execute the binary.

Parallel Jobs – Common issues

- Manual defining of MPI tasks for mpirun?

- **False: Do not use if your job solely does MPI:**

- `mpirun --machinefile=file binary`

- `mpirun -n <int> binary`

- **Correct way:**

- `mpirun binary` (*because the resource manager tells mpirun what to do*)

Interactive Jobs

Remote Display via Virtual Computing Network

- What for: Do resource intensive graphical apps on cluster, details here

```
uc1:~$ start_vnc_desktop --hw-rendering
```

Hint for TurboVNC Viewer users (command line):

```
vncviewer ExtSSH=1 Via=yc8563@uc1.scc.kit.edu
```

Hint for TurboVNC Viewer users (GUI)

Fill in the following entry field:
VNC server: **vc1n02:5901**

Click "Options" and choose tab "Security".
Fill in the following entry fields:

Gateway (SSH server or UltraVNC repeater)
SSH user: **yc8563**
Host: **uc1.scc.kit.edu**
Click "OK"

Click "Connect"

VNC Authentication:
Password: **AgGQmo8z**

Hint for installing VNC viewer:

```
/usr/bin/start_vnc_desktop --help-client
```

```
local:~$ Download latest turbovnc version
local:~$ sudo apt-get install ./turbovnc_<ver<_amd64.deb
local:~$ /opt/TurboVNC/bin/vncviewer
```

The screenshot shows a web browser window with the address bar containing 'sysinfo:/' and a search bar with 'Google'. The main content area displays system information in a clean, modern layout. The 'Network Status' section shows 'You are online' with a globe icon. The 'OS Information' section lists: OS: Linux 3.10.0-693.2.2.el7.x86_64 x86_64, Current user: yc8563@vc1n02.localdomain, System: Red Hat Enterprise Linux Server release 7.4 (Maipo), and KDE: 4.14.8. The 'CPU Information' section lists: Processor (CPU): Intel(R) Xeon(R) CPU E5-2637 v2 @ 3.50GHz, Speed: 1,769.55 MHz, and Cores: 16. The 'Display Info' section lists: Vendor: NVIDIA Corporation, Model: Quadro K6000/PCIe/SSE2, and Driver: 4.5.0 NVIDIA 384.90. The 'Memory Information' section lists: Total memory (RAM): 251.7 GB, Free memory: 247.1 GB (+ 1.1 GB Caches), and Free swap: 0.0 KB. The browser's status bar at the bottom shows 'Page loaded.' and the system tray includes icons for network, volume, and power, along with the time '10:02 PM'.

Thank you for your attention!