bw | HPC – C5

Batch System – Best Practices

Robert Barthel, SCC, KIT

Funding: Baden-Württemberg MINISTERIUM FÜR WISSENSCHAFT, FORSCHUNG UND KUNST

www.bwhpc-c5.de

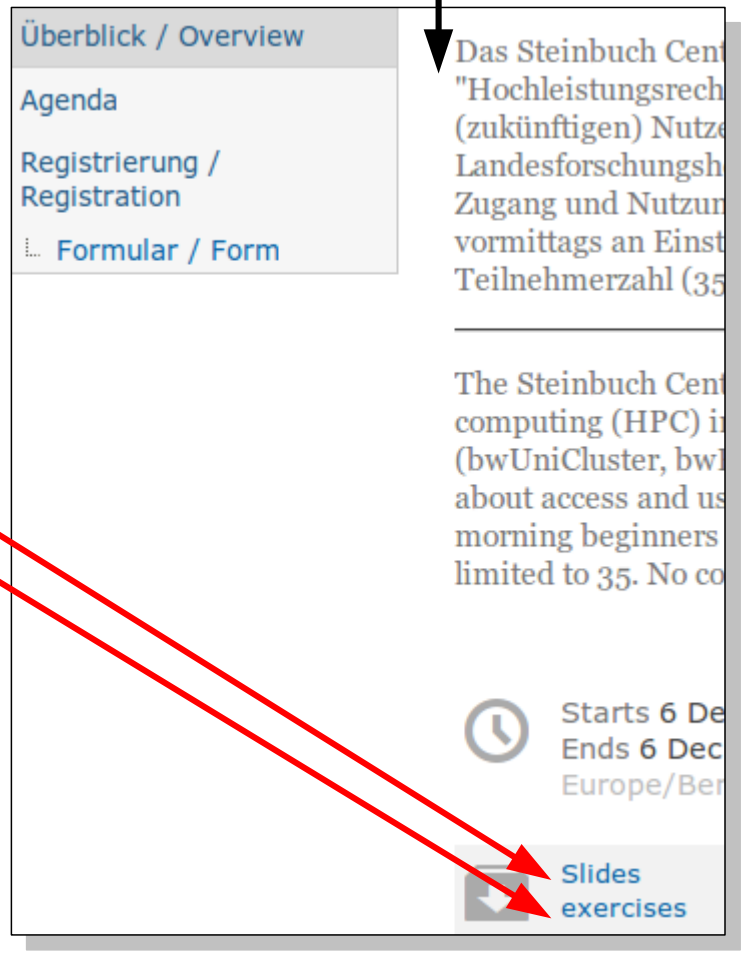
How to read the following slides

Abbreviation/Colour code	Full meaning
`$ command -opt value`	*$* = prompt of the interactive shell The full prompt may look like: `user@machine:path $` The command has been entered in the interactive shell session
`<integer>` `<string>`	<> = Placeholder for integer, string etc
`foo, bar`	Metasyntactic variables
`${WORKSHOP}`	`/pfs/data1/software_uc1/bwhpc/kit/workshop/2017-04-05`

- 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_2017-04-05 or uc1:/pfs/data1/software_uc1/bwhpc/kit/workshop/2017-04-05
  - Slides
  - Exercises



Job Scripting Basics

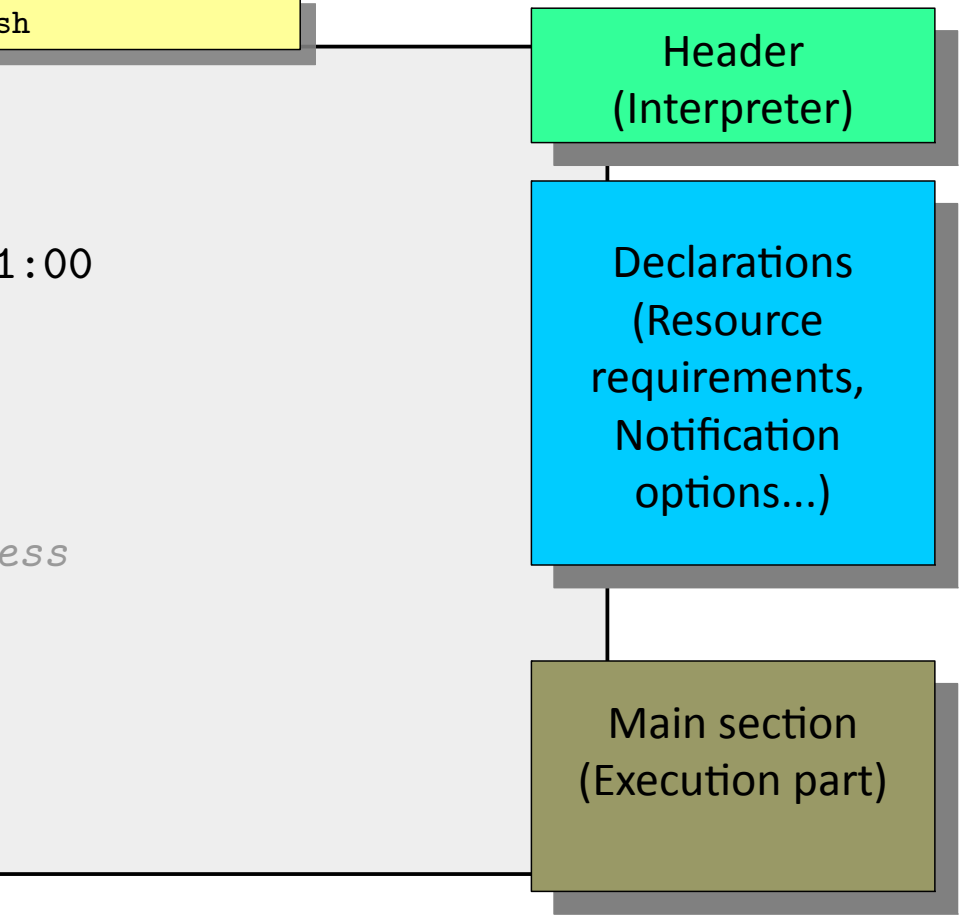
Job script: Structure (1)

${WORKSHOP}/exercises/06/01_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


printenv
```



Job script: Structure (2)

- Be descriptive! Comment your script code

${WORKSHOP}/exercises/06/02_job.sh

```
#!/bin/bash

#MSUB -l nodes=1:ppn=1
#MSUB -l walltime=00:01:00
#MSUB -l pmem=50mb
#MSUB -l advres=workshop.16
#MSUB -N serial-test

# Printing all environment variable
printenv
```

Job script: Structure (3)

- Print out what your script is doing!

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

```
#!/bin/bash

#MSUB -l nodes=1:ppn=1
#MSUB -l walltime=00:01:00
#MSUB -l pmem=50mb
#MSUB -l advres=workshop.16
#MSUB -N serial-test

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

Job script: Structure (4)

- Use Global Environment Variables!

${WORKSHOP}/exercises/06/04_job.sh

```
#!/bin/bash

#MSUB -l nodes=1:ppn=1
#MSUB -l walltime=00:01:00
#MSUB -l pmem=50mb
#MSUB -l advres=workshop.16
#MSUB -N serial-test

# 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 -l nodes=1:ppn=2
```

2) 2 cores on nodes 1,2 + 4 cores on node 3:

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

Excursus: Ressource specifications (2)

- @ MOAB
  - **mem**
    = Working memory for total job
  - **pmem**
    = Working memory per cores

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
```

- 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!

Job script: Structure (5)

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

${WORKSHOP}/exercises/06/05_job.sh

```
#!/bin/bash

#MSUB -l nodes=1:ppn=1
#MSUB -l walltime=00:01:00
#MSUB -l pmem=50mb
#MSUB -l advres=workshop.16
#MSUB -N serial-test

# 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

  # → commenting out lines

  ' → full quotes, perserves all chars

  " → partial quotes, does not perserve e.g. $

  \ → escaping (special) characters

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

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

- (Conditional) Tests:
  - Existance 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/06/06_job.sh

```
#!/bin/bash

#MSUB -l nodes=1:ppn=1
#MSUB -l walltime=00:01:00
#MSUB -l pmem=50mb
#MSUB -l advres=workshop.16
#MSUB -N serial-test


# 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/7.1_tmolex42/bwhpc-examples
```

bwHPC_turbomole_single-node_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 interprete -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 existance 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,walltime=00:00:10
#MSUB -l pmem=70mb
#MSUB -l advres=workshop.16

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/06/parallel/omp.sh`

```
#!/bin/bash
#MSUB -l nodes=1:ppn=2
#MSUB -l walltime=00:01:00
#MSUM -l pmem=50mb
#MSUB -l advres=workshop.16
# 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
${executable}
```

- Shared memory restricts to 1 node.
- Do not define number of threads explicitely. Use MOAB variables.

Submitting parallel jobs (MPI) (1)

- Example: `${WORKSHOP}/exercises/06/parallel/mpi.sh`

```
#!/bin/bash
#MSUB -l nodes=1:ppn=2
#MSUB -l walltime=00:01:00
#MSUM -l pmem=50mb
#MSUB -l advres=workshop.16
# 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}"
# Execute program
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.

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 VNC

- What for: Do resource intensive graphical apps on cluster
- HowTo: Submit interactive job @ bwUniCluster

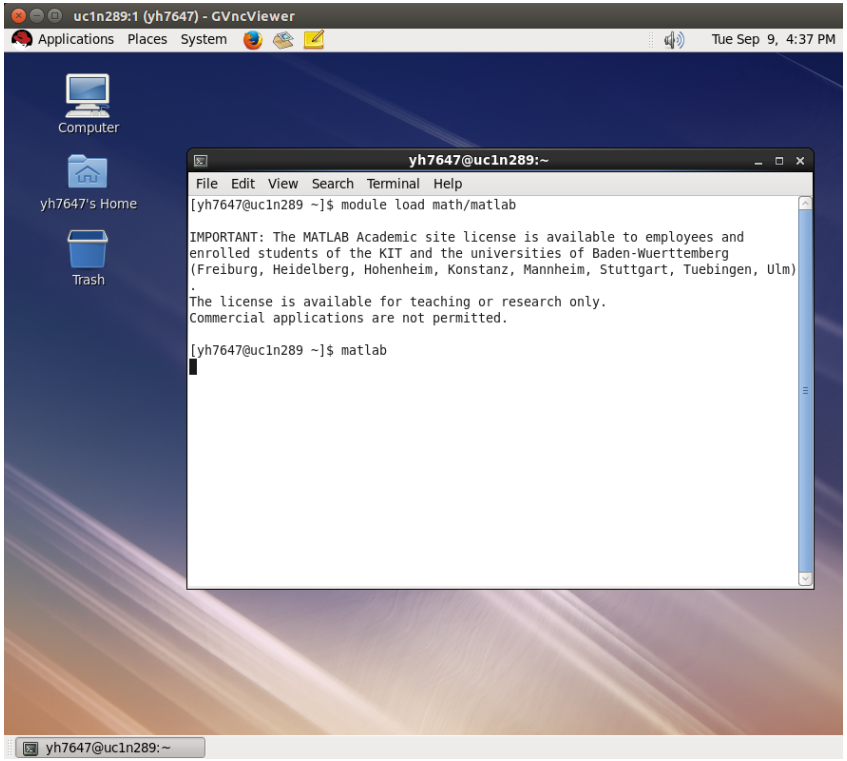
```
$ msub -I -V -l nodes=1:ppn=1,walltime=02:00:00,mem=4000mb
```

- @ compute node: Start VNC server

```
$ module load vis/tigervnc
$ run_vncserver
```

  - Set initial VNC password.
  - Follow displayed instructions.

- @ localhost:
  - Install VNC client
    - sudo apt-get install vncviewer
    - Windows: TightVNC Java Viewer
  - Start VNC client



Thank you for your attention!