



bw|HPC – C5

# Tutorial: Advanced (Batch) Job Scripting

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**

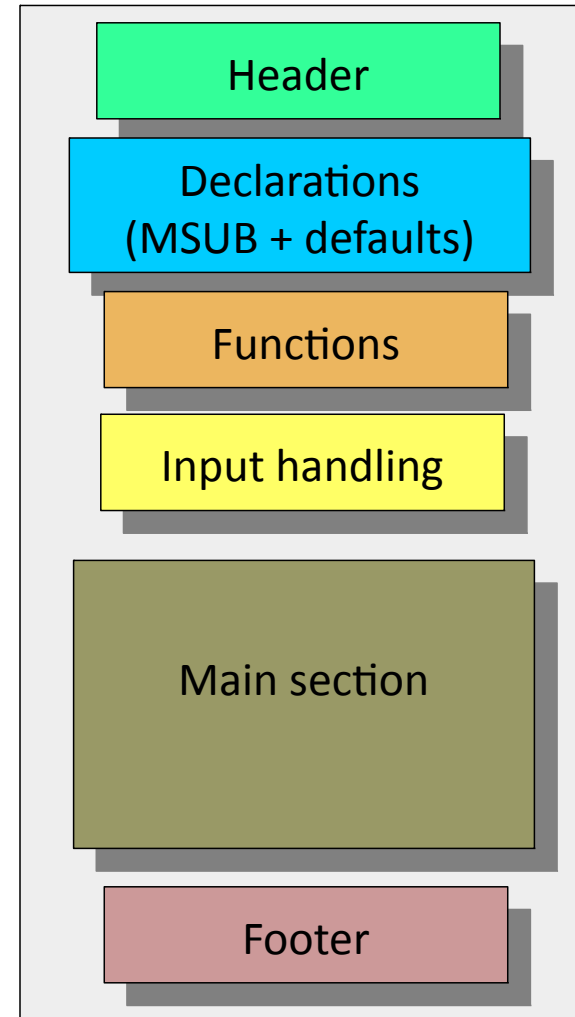


# How to read the following slides

Abbreviation/Colour code	Full meaning
<code>\$ command -opt value</code>	<code>\$</code> = <b>prompt</b> 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>&lt;integer&gt;</code> <code>&lt;string&gt;</code>	<code>&lt;&gt;</code> = Placeholder for integer, string etc
<code>foo, bar</code>	Metasyntactic variables
<code>\${WORKSHOP}</code>	<code>/pfs/data1/software_uc1/bwhpc/kit/workshop/2017-04-06</code>

# Goal

- Be descriptive!
  - Comment your code
    - e.g. via headers sections of script and functions.
  - Decipherable names for variables and functions
- Organise and structure!
  - Break complex scripts into simpler blocks e.g. use functions
  - Use exit codes
  - Use standardized parameter flags for script invocation.
- Write job script that runs **interactively**
  - Then add part for MOAB



# Best Practises – Common Problems (1)

Do not Run your code, application, job on login nodes / in  $\${HOME}$ :

- for interactive jobs use [msub -I -V](#)

## Multinode Job:

- use [workspaces](#)
- Producing Tbyte of scratch files & >10000 File: [Change your application code](#)  
Need help? Apply for Tiger Team Support.

## Singlenode Job:

- use  $\${TMPDIR}$ : [HowTo](#) → [Case 1](#)

## Chain jobs: [HowTo](#) → [Case 2](#)

## Many sequential tiny jobs:

- Bundle to one big job: [HowTo](#) → [Case 3](#)

## Handling walltime based job aborts: [HowTo](#) → [Case 4](#)

## Use of MPI/OpenMP Parallelisation in jobs:

- <https://indico.scc.kit.edu/indico/event/278/material/slides/14.pdf>

# Rev: MOAB variables

■ [http://www.bwhpc-c5.de/wiki/index.php/Batch\\_Jobs#Moab\\_Environment\\_Variables](http://www.bwhpc-c5.de/wiki/index.php/Batch_Jobs#Moab_Environment_Variables)

MOAB variables	
Environment variables	Description
MOAB_CLASS	Class name
MOAB_GROUP	Group name
MOAB_JOBID	Job ID
MOAB_JOBNAME	Job name
MOAB_NODECOUNT	Number of nodes allocated to job
MOAB_PARTITION	Partition name the job is running in
MOAB_PROCCOUNT	Number of processors allocated to job
MOAB_SUBMITDIR	Directory of job submission
MOAB_USER	User name

■ MSUB variables:

```
#!/bin/bash

#MSUB -N test
#MSUB -l advres=workshop.17,nodes=1:ppn=1,mem=50mb
#MSUB -l walltime=00:05:00
#MSUB -m n
#MSUB -v my_own_variables="arguments"
```

# Case 1: Jobs @ \$TMPDIR (1)

- If temporary files of job > Gbyte → Run your job at \${TMPDIR}
  - but ONLY if single node jobs
- What to do:
  - Generate subdirectory under \${TMPDIR} => \${run\_DIR}
  - Copy to \${run\_DIR}
  - Change to \${run\_DIR} & program execution
  - Copy results to start DIR
- How?
  - Start with templates:

```
${WORKSHOP}/exercises/04/01_job_run_under_local_tmpdir.sh  
+  
${WORKSHOP}/exercises/04/{01_gen_files,01_gen_files.inp}
```

# Case 1: Jobs @ \$TMPDIR (2)

Code snip: `${WORKSHOP}/exercises/04/01_job_run_under_local_tmpdir.sh`

```
#!/bin/bash
...
## a) Tutorial ToDo: load modules INTEL+MKL
    if not loaded

## b) Define your run directory under tmpdir
##     incorporating username and JobID/PID
mkdir -pv "${TMPDIR}/${USER}.${MOAB_JOBID:-$$}"

## c) Tutorial ToDo: Check existence of run directory

## d) Copy files from submit directory
##     to run directory
cd $MOAB_SUBMITDIR
cp -pv gen_files.x "${TMPDIR}/${USER}.${MOAB_JOBID:-$$}"
##     Check if copy succeeded
cp -pv gen_files.inp "${TMPDIR}/${USER}.${MOAB_JOBID:-$$}"

## e) Change to run directory (check if succeeded) and start binary + input file
cd "${TMP}/${USER}.${MOAB_JOBID}"
./01_gen_files.x 01_gen_files.inp

## f) Tutorial ToDo: check run status

## g) transfer files to submit directory
cp -pv files_*.out "${MOAB_SUBMITDIR}"

## h) Tutorial ToDo: cleanup run_DIR
```

TASK/ToDo: 10min  
\* Generalise blue code  
 avoiding repetition  
\* Write code for a-h  
\* Redirect output of binary

## Case 1: Jobs @ \$TMP (3)

Decl. + a-c:

```
${WORKSHOP}/solutions/04/01_generalised_job_run_under_local_tmpdir.sh
```

Solution!

```
## 1) Define full path of your binary
EXE="${MOAB_SUBMITDIR:-${PWD}}/01_gen_files.x"

## 2) Define output file
##     = Name of executable + JOBID or PID
output="$(basename ${EXE})_${MOAB_JOBID:-$$}.log"

## 3) Define full path input files
Input="${MOAB_SUBMITDIR:-${PWD}}/01_gen_files.inp"

## 4) Define input files to be copied
copy_list="${EXE} ${input}"

## 5) Define files to be copied back after run, i.e. output file
save_list="${output} files_*.out"

## a) Load modules INTEL+MKL if not loaded
for mod in compiler/intel numlib/mkl ; do
    module list 2>&1 | grep "${mod}" >/dev/null || module load "${mod}"
done

## b) Define your run directory and generate via mkdir
run_DIR="${TMPDIR}/${USER}.${MOAB_JOBID:-$$}"
mkdir -pv "${run_DIR}"

## c) Check existence of run directory
if [ ! -d "${run_DIR}" ] ; then
    echo "ERROR: Run DIR = ${run_DIR} does not exist"; exit 1
fi
```





## Case 1: Jobs @ \$TMP (4)

Part d-h:

```
${WORKSHOP}/solutions/04/01_generalised_job_run_under_local_tmpdir.sh
```

Solution!

```
## d) Change to Submit Dir or PWD / Copy files from submit_DIR to run_DIR
cd "${MOAB_SUBMITDIR:-${PWD}}"
for X in ${copy_list} ; do
    cp -pv "${X}" "${run_DIR}"
    if [ $? -ne 0 ] ; then echo "ERROR: Copy of ${X} failed"; exit 1; fi
done

## e) Change to runDIR and start binary
cd "${run_DIR}"
if [ $? -ne 0 ] ; then echo "ERROR: Entering ${run_DIR} failed"; exit 1; fi
./$EXE ${input} > $output 2>&1

## f) Check run status
if [ $? -ne 0 ] ; then
    echo "WARNING: ${EXE} did not run properly!"
fi

## g) Transfer output files to submit directory
cd "${run_DIR}"
for X in ${save_list} ; do
    cp -pv "${X}" "${MOAB_SUBMITDIR}"
    if [ $? -ne 0 ] ; then echo "WARNING: Copy of ${X} failed"; fi
done

## h) Cleanup run directory
rm -f ${run_DIR}/*; rmdir ${run_DIR}; exit 0
```



## Case 2: Chain Jobs (1)

- Idea:
  - Do  $N$  consecutive Jobs via  $N$  MOAB Batch Jobs
- Goal:
  - Do everything in one script
  - Submit only at the beginning
- „Pre-step“: generate script that runs interactively
  - Result: `${WORKSHOP}/exercises/04/02_chain_job.sh`

## Case 2: Chain Jobs (2)

```
#!/bin/bash
## Defaults
loop_max=10
cmd='sleep 2'

## Check if counter environment variable is set
if [ -z "${myloop_counter}" ] ; then
    echo "  ERROR: myloop_counter is undefined, stop chain job"; exit 1
fi
## Only continue if below loop_max
if [ ${myloop_counter} -lt ${loop_max} ] ; then
    ## Increase counter
    let myloop_counter+=1
    ## Print current Job number
    echo "  Chain job iteration = ${myloop_counter}"
    ## Execute your command
    echo "  -> executing ${cmd}"
    ${cmd}

    if [ $? -eq 0 ] ; then
        ## Continue only if last command was successful
        export myloop_counter=${myloop_counter}
        ./${0}
    else
        ## Terminate chain
        echo "  ERROR: ${cmd} of chain job no. ${myloop_counter} terminated unexpectedly"
        exit 1
    fi
fi
fi
```

```
${WORKSHOP}/exercises/04/02_chain_job.sh
```

```
$ export myloop_counter=0
$ ./02_interactive_chain_job
```

## Case 2: Chain Jobs (2) → How for MOAB?

```
#!/bin/bash
#MSUB ...
## Defaults
loop_max=10
cmd='sleep 2'

## Check if counter environment variable is set
if [ -z "${myloop_counter}" ] ; then
    echo " ERROR: myloop_counter is undefined, stop chain job"; exit 1
fi
## only continue if below loop_max
if [ ${myloop_counter} -lt ${loop_max} ] ; then
    ## increase counter
    let myloop_counter+=1
    ## print current Job number
    echo " Chain job iteration = ${myloop_counter}"
    ## Execute your command
    echo " -> executing ${cmd}"
    ${cmd}

    if [ $? -eq 0 ] ; then
        ## continue only if last command was successful
        export myloop_counter=${myloop_counter}
        ./${0}
    else
        ## Terminate chain
        echo " ERROR: ${cmd} of chain job no. ${myloop_counter} terminated unexpectedly"
        exit 1
    fi
fi
```

TASK/ToDo:5 min  
\* add the parts for MOAB



## Case 2: Chain Jobs (3) → Solution! for Moab

```
#!/bin/bash
#MSUB -l nodes=1:ppn=1,walltime=00:00:05,pmem=50mb
## Defaults
loop_max=10
cmd='sleep 2'

## Check if counter environment variable is set
if [ -z "${myloop_counter}" ] ; then
    echo " ERROR: myloop_counter is undefined, stop chain job"; exit 1
fi
## only continue if below loop_max
if [ ${myloop_counter} -lt ${loop_max} ] ; then
    ## increase counter
    let myloop_counter+=1
    ## print current Job number
    echo " Chain job iteration = ${myloop_counter}"
    ## Execute your command
    echo " -> executing ${cmd}"
    ${cmd}

    if [ $? -eq 0 ] ; then
        ## continue only if last command was successful
        msub -v myloop_counter=${myloop_counter} ./02_chain_job.sh
    else
        ## Terminate chain
        echo " ERROR: ${cmd} of chain job no. ${myloop_counter} terminated unexpectedly"
        exit 1
    fi
fi
```

Solution!

`${WORKSHOP}/solutions/04/02_chain_job.sh`

`$ msub -v myloop_counter=0 ./02_chain_job.sh`

## Case 2: Chain Jobs (4)

■ moab\_chain\_job.sh + interactive\_chain\_job.sh =

```
${WORKSHOP}/solutions/04/02_generalised_chain_job.sh
```

```
...
...
if [ $? -eq 0 ] ; then
  ## continue only if last command was successful
  if [ ! -z ${MOAB_JOBNAME} ] ; then
    ## If MOAB_JOBNAME environment variable is defined
    ## -> this script is under MOAB "control"
    msub -v myloop_counter=${myloop_counter} ./generalised_chain_job.sh
  else
    export myloop_counter=${myloop_counter}
    ./${0}
  fi
else
  ## Terminate chain
  echo " ERROR: ${cmd} of chain job no. ${myloop_counter} terminated unexpectedly"
  exit 1
fi
...
...
```

→ USE bash programming to **generalise** and **unify** your batch job scripts

# Chain Jobs – Alternative (1)

## ■ Problem of `moab_chain_job.sh`: **Waiting time!**

■ Solution: two scripts + `msub -l depend=afterok:<jobID>`

■ 1. script: `${WORKSHOP}/solutions/04/02_chain_link_job.sh`

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

## Define your command
cmd='sleep 30'

## Execute your command
echo "  -> executing ${cmd}"
${cmd}

## Do you check if correctly terminated
if [ $? -ne 0 ] ; then
  ## Terminate chain
  echo "  ERROR: ${cmd} of chain job no. ${myloop_counter:-1} terminated unexpectedly"
  exit 1
fi
```

## Chain Jobs – Alternative (2)

- 2. script: `${WORKSHOP}/solutions/04/02_moab_submitter_f_chain_job.sh`

```
#!/bin/bash
max_nojob=${1:-5}
chain_link_job=${PWD}/02_chain_link_job.sh
dep_type="${2:-afterok}"

counter=1
while [ ${counter} -le ${max_nojob} ] ; do
    ## Differ msub_opt depending on chain link number
    if [ ${counter} -eq 1 ] ; then
        msub_opt=""
    else
        msub_opt="-l depend=${dep_type}:${jobID}"
    fi

    echo "Chain job iteration = ${counter}"
    echo "    msub -v myloop_counter=${counter} ${msub_opt} ${chain_link_job}"
    ## Store job ID for next iteration by storing output of msub command with empty lines
    jobID=$(msub -v myloop_counter=${counter} ${msub_opt} ${chain_link_job} 2>&1 | sed '/^$/d')

    ## Check if ERROR occurred
    if [[ "${jobID}" =~ "ERROR" ]] ; then
        echo "    -> submission failed!" ; exit 1
    else
        echo "    -> job number = ${jobID}"
    fi

    ## Increase counter
    let counter+=1
done
```



## Case 3: Pseudo Parallelisation (1)

- If you have many (>100) tiny jobs (subjobs)
  - Pack in one job (masterjob) doing:
    - Define number of Cores got by queueing system
    - Queue subjobs and assign step by step to free Cores of masterjob

## Case 3: Pseudo Parallelisation - Alternative

- Parbatch → MPI task based

Example: job script

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

```
#!/bin/bash

#MSUB -l nodes=1:ppn=4
#MSUB -l mem=150mb
#MSUB -l walltime=00:03:00

module load system/parbatch

parbatch joblist.txt
```

+ joblist.txt

```
`${WORKSHOP}/exercises/04/03_joblist.txt
```

```
hostname ; sleep 2; echo "Hello 1-a"
hostname ; sleep 2; echo "Hello 2-b"
hostname ; sleep 2; echo "Hello 3-c"
hostname ; sleep 2; echo "Hello 4-d"
hostname ; sleep 2; echo "Hello 5-e"
hostname ; sleep 2; echo "Hello 6-f"
hostname ; sleep 2; echo "Hello 7-g"
hostname ; sleep 2; echo "Hello 8-h"
```

## Case 4: Handling walltime based job aborts

- Use: „msub -l signal“ and „trap“ to abort job on own terms

```
`${WORKSHOP}/exercises/04/04_handle_aborts.sh
```

```
#!/bin/bash
## Pre-termination via MOAB
## sending signal with defined offset

#MSUB -l nodes=1:ppn=1,walltime=00:01:00,mem=100mb
#MSUB -l signal=15@120
#MSUB -l advres=workshop.17

cleanup(){
    echo "Cleanup before walltime reached"
    exit 0
}

trap cleanup 15

echo "Repeating \"sleep 10\" loop until SIGTERM"
while true ; do
    sleep 10
done
```

MOAB sends SIGTERM (kill -15)  
120 seconds before walltime  
is reached

# Best Practises – Batch jobs with input parsing

## ■ Not working:

- msub ***your\_script -x argument***  
→ msub will interpret -x as an own option

## ■ Solution:

(A) Submit wrapper script:

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

(B) Export your script options and arguments to environment variable; read in that variable during runtime of script, cf. [wiki](#)

```
if [ -n "${SCRIPT_FLAGS}" ] ; then
  if [ -z "${*}" ] ; then
    set -- ${SCRIPT_FLAGS}
  fi
fi
```

(C) Use msub wrapper via:

```
$ module load system/msub_addon/1.0
$ msub <options> job.sh
```

## Best Practises – Common problems (2)

- 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*)

- If you want to know about job allocated hosts in your script to:

- (A) Use msub wrapper via:

```
$ module load system/msub_addon/1.0  
$ msub <options> job.sh
```

- (B) Write loop into your batch job script → returns hostname of each task:

```
for tasks in $(srun hostname) ; do  
    echo $tasks  
done
```

Thank you for your attention!