bw|HPC

Tutorial:
Advanced (Batch) Job Scripting

Robert Barthel, SCC, KIT

UNIVERSITÄT HEIDELBERG ZUKUNFT SEIT 1386

Hochschule für Technik Stuttgart

UNI FREIBURG

UNIVERSITÄT HOHENHEIM 1818

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

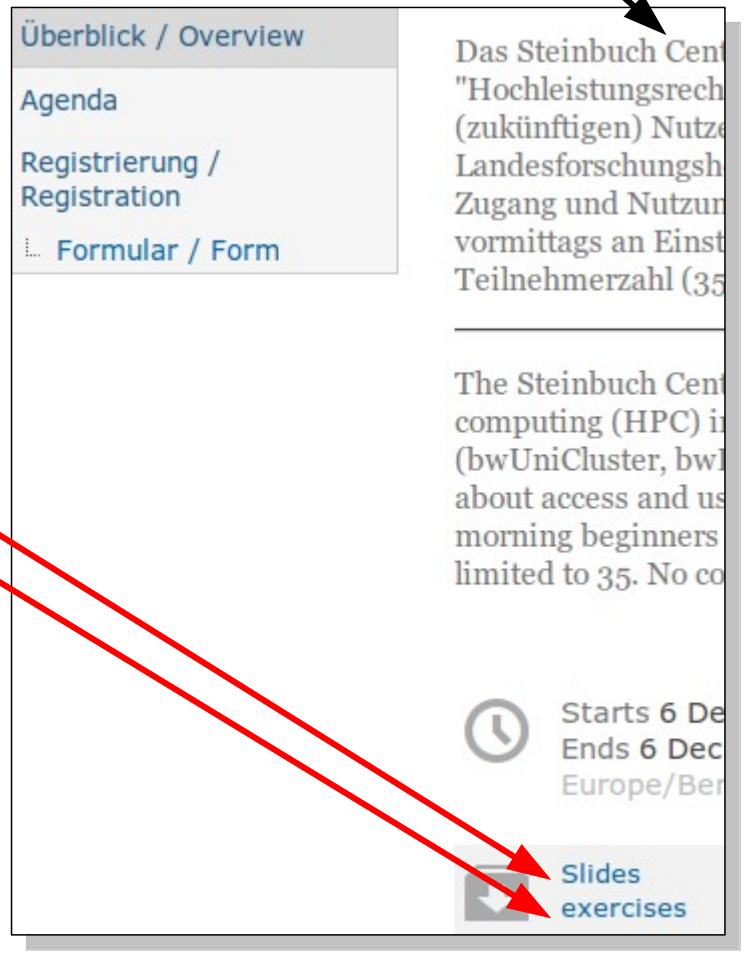
Funding: Baden-Württemberg MINISTERIUM FÜR WISSENSCHAFT, FORSCHUNG UND KUNST

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/2019-10-09`

Where to get the slides/exercises/reservation?

- https://indico.scc.kit.edu/e/bwhpc_course_2019-10-09 or
  bwUniCluster: /pfs/data1/software_uc1/bwhpc/kit/workshop/2019-10-09
  - Slides
  - Exercises



- Workshop reservation:
  single node:

```
msub -A workshop
-l advres=bwhpc-workshop_single.140
```

- multi node:

```
msub -A workshop
-l advres=bwhpc-workshop_single.140
```

How to do excerices?

- Login to cluster & Generate workspace „bwhpc-course“

```
$ ws_allocate bwhpc-course 30
Workspace created. Duration is 720 hours.
Further extensions available: 3
/pfs/work2/workspace/scratch/xy1234-bwhpc-course-0
```

- Copy examples to your workspace

```
$ WORKSHOP=/pfs/data1/software_uc1/bwhpc/kit/workshop/2019-04-10
$ cd $(ws_find bwhpc-course)
$ mkdir -v 2019-10-09; cd 2019-10-09
$ cp -pr ${WORKSHOP}/exercises/02 ./
```

- Submit jobs from your workspace

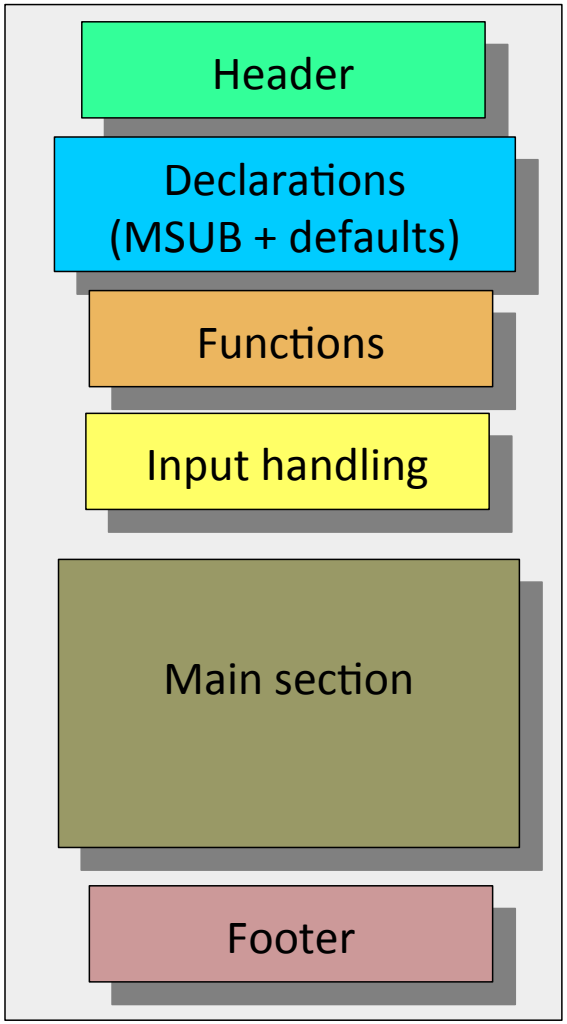
```
$ cd $(ws_find bwhpc-course)/2019-10-09/02
$ msub <jobscript>
```

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/SLURM



Typical Issues & Cases (1)

- Singlenode Issues
  - (Parallel) File System Issues
    - Workflow for job on a different Filesystem (on $TMPDIR, Case 1)
  - *OpenMP Jobs (cf. Afternoon – talk no. 4)*
- Multinode Issues
  - Parallel File System Issues
  - *MPI Jobs (cf. Afternoon – talk no. 4)*
- Walltime Issues
  - Job abortion (Case 2)
- Task Issues
  - Bulk Jobs (Case 3)
  - Array Jobs (Case 4)
  - Chain Jobs (Case 5)

Typical Issues & Cases (2)

- Login Node Issues
  - @bwUniCluster, we have 4 Login nodes but over 2000! users
  - „only want to test fast/interactively“ slows the login nodes

  → Do not Run your code, application, job on login nodes / in `${HOME}`:
    - for interactive jobs use msub -I -V
    - For development use develop queue

MOAB variables

- bwHPC Wiki , excerpt:

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 submit options (excerpt):

```
#!/bin/bash

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

SLURM variables (ForHLR I/II)

- ForHLR Wiki , excerpt:

Environment	Brief explanation
SLURM_JOB_CPUS_PER_NODE	Number of processes per node dedicated to the job
SLURM_JOB_NODELIST	List of nodes dedicated to the job
SLURM_JOB_NUM_NODES	Number of nodes dedicated to the job
SLURM_MEM_PER_NODE	Memory per node dedicated to the job
SLURM_NPROCS	Total number of processes dedicated to the job
SLURM_CLUSTER_NAME	Name of the cluster executing the job
SLURM_CPUS_PER_TASK	Number of CPUs requested per task
SLURM_JOB_ACCOUNT	Account name
SLURM_JOB_ID	Job ID
SLURM_JOB_NAME	Job Name
SLURM_JOB_PARTITION	Partition/queue running the job
SLURM_JOB_UID	User ID of the job's owner
SLURM_SUBMIT_DIR	Job submit folder. The directory from which msub was invoked.
SLURM_JOB_USER	User name of the job's owner

- SLURM Submit options (excerpt):

```
#!/bin/bash

#SBATCH -J test
#SBATCH -N1 -n1 -c1 --mem=50mb
#SBATCH -t 00:05:00
#SBATCH --mail-type=all
#SBATCH --export="All,my_own_variable=arguments"
```

File system issues (1)

- Multinode Job:
  - use *workspaces*
  - Producing Tbyte of scratch files & >10000 File: Change your application code
    Need help for that? Apply for Tiger Team Support.
- Singlenode Job:
  - A lot of I/O?
    → opt out to local file system instead of global one
    - use `${TMPDIR}`: but requires a „workflow“

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/02/01_job_run_under_local_tmpdir.sh
+
${WORKSHOP}/exercises/02/{01_gen_files,01_gen_files.inp}
```

Jobs @ $TMPDIR (2)

- Code snip: `${WORKSHOP}/exercises/02/01_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
```

Jobs @ $TMPDIR (3)

- Decl. + a-c: `${WORKSHOP}/solutions/02/01_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 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
```

Jobs @ $TMPDIR (4)

- Part d-h: `${WORKSHOP}/solutions/02/01_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
```

Walltime Issues (1)

- Revision:
  - Jobs have limited runtime (=walltime)
  - Define walltime by your own, cf. `msub -l walltime=D:HH:MM:SS`
- Issue:
  - Executable needs more time than given walltime

    → queueing system is terminating your jobscript and its child processes
- Solution:
  - `msub -l signal=<sigint>@<seconds>` , e.g. 120 before walltime send sigterm (15)

```
TASK/ToDo:10 min
* combine "msub -l signal" & "trap" to trigger message and "exit 1"
```

  - template: `${WORKSHOP}/exercises/02/04_handle_aborts.sh`

Walltime Issues (2)

Solution!

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

${WORKSHOP}/solutions/02/04_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@10
#MSUB -l advres=bwhpc-workshop.64
#MSUB -A workshop

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) 10 seconds before walltime is reached

Bulk Jobs (1)

- Many (>100) „independent“ jobs with very short runtime
  - Solution:

    → Pack in one multinode/multitask job with long runtime

    HowTo?
    - Assign resources for „parallel“ task processing, aka „workers“
    - Load balance „workers“, i.e., and assign step by step free „workers“ with jobs

Bulk Jobs: MPI based solution

- Parbatch → MPI task based

Example: job script

`${WORKSHOP}/exercises/02/03_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/02/03_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"
```

TASK/ToDo: 5 min

- Prepare joblist with 10 jobs each running max. 15 seconds and submit it with 2 cores

Job Arrays (1)

- Jobs with a „task range“
  - with the same executable and ressource requirements
    → but different input (files)
  - Interactive setup (aka „pure“ bash script setup):
    - Master script:
      - Translating index setup into list, excuting each index value as a job
  - MOAB:
    - Available as submit feature:

```
msub -t <name>.[<indexlist>]%<limit> job.sh
```

    → makes master script obsolete & groups Job IDs (= easier to handle)
    → **`job.sh`** gets index value via **`$MOAB_JOBARRAYINDEX`**
- Issue:
  - Moab job arrays do not work on bwUniClusters

Bash based **Job Array (2)**

- Without array submit features → approach:
  - **a.)** handle each index value as one moab job
  - **b.)** handle as one (=master) moab job

```
export IARR="0-10=2"
${WORKSHOP}/exercises/02/05_master_job_array.sh
```

```
#!/bin/bash

#MSUB -v IARR="2-10=2"   # index setup: min-max=inc

# Define subjob script
subjob="./05_subjob.sh"
# Decipher index setup:
IARR=${IARR:-1-5=1}
if [[ ${IARR/=//} = ${IARR} ]] ; then inc=1 ; else inc=${IARR/*=} ; fi
IARR=${IARR/=*}
if [[ ${IARR/-//} = ${IARR} ]] ; then max=1 ; else max=${IARR/*-} ; fi
min=${IARR/-*}

echo "Generate index list from ${min} to ${max} with increment ${inc}"
while [[ $min -le $max ]] ; do
   echo "  Execute ${subjob} $min"
   #${subjob} $min
   let min+=${inc}
done
```

Job Array (3)

TASK/ToDo: 10min

- Modify `05_master_job_array.sh`
  - To do parallel (use parbatch):
    - subjob.sh write index value to indexed output

`${WORKSHOP}/exercises/02/05_subjob.sh`

```
#!/bin/bash

## Get index value via positional parameter
value="??"

## Define name of output file
outputfile="??"

## Write value to file
??
```

Job Array (4)

Solution!

- Modify `05_master_job_array.sh`
  - To do parallel (use parbatch):
    - subjob.sh write index value to indexed output

`${WORKSHOP}/solutions/02/05_subjob.sh`

```
## Get index value via positional parameter
value="${1:?missing_value}
## Define name of output file
outputfile="array_${value}.out"
## Write value to file
echo ${value} > ${outputfile}
```

`${WORKSHOP}/solutions/02/05_master_job_array.sh`

```
...
module load system/parbatch
...
joblist=joblist_${MOAB_JOBID:-$$}.txt
while [[ $min -le $max ]] ; do
   echo "  Execute ${subjob} $min"
   echo "${subjob} $min" >> ${joblist}
   let min+=${inc}
Done
# Execute parbatch
parbatch ${joblist}
```

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/02/02_chain_job.sh`

Bash script based **Chain Jobs (2)**

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

```
$ export myloop_counter=0
$ ./02_chain_job.sh
```

```
#!/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
```

Chain Jobs (3) → How for MOAB/Slurm?

```
#!/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

Chain Jobs (4) → 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
```

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

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

Chain Jobs (4)

- Moab/Slurm + interactive script =

${WORKSHOP}/solutions/02/02_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
      elif [ ! -z ${SLURM_JOB_NAME} ] ; then
         sbatch -p develop --export="myloop_counter=${myloop_counter}" ./02_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: Optimization (1)

- Problem of 02_generalised_chain_job.sh: **Waiting time**!
  - Solution: two scripts (master + links) + `msub -l depend=afterok:<jobID>`
  - 1. script - links: `${WORKSHOP}/solutions/02/02_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: Optimization (2)

- 2. script - master: `${WORKSHOP}/solutions/02/02_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 occured
   if [[ "${jobID}" =~ "ERROR" ]] ; then
      echo "   -> submission failed!" ; exit 1
   else
      echo "   -> job number = ${jobID}"
   fi
   ## Increase counter
   let counter+=1
done
```

Thank you for your attention!

Questions?