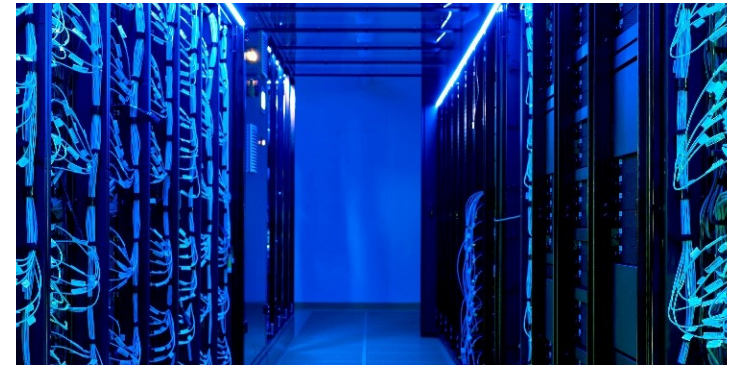
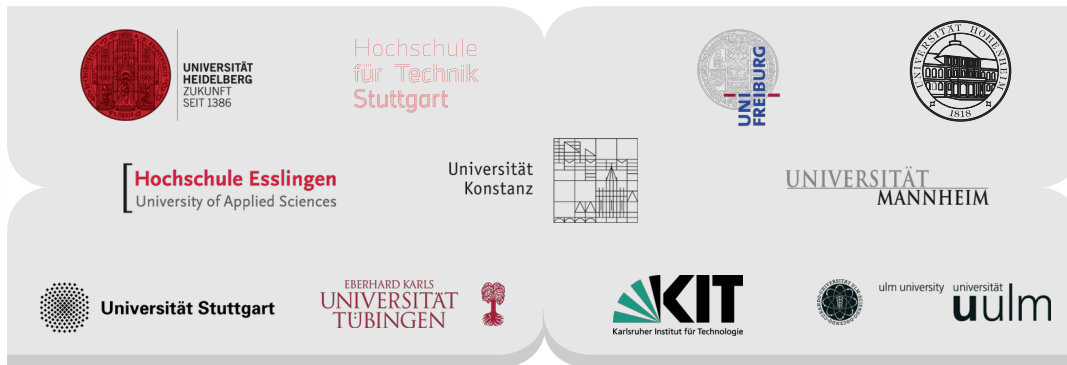


Advanced (Batch) Job Scripting

Robert Barthel, SCC, KIT



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>/opt/bwhpc/common/workshops/2021-10-27</code>

Where to get the slides/exercises/reservation?

■ https://indico.scc.kit.edu/e/bwhpc_course_2021-10-27 or

bwUniCluster: /opt/bwhpc/common/workshops/2021-10-27

horeka:/software/all/workshops/2021-10-27

■ exercises

■ slides

■ Workshop reservation:
single node:

```
SBATCH -p ws ...
```

■ multi node:

```
SBATCH -p wspar ...
```

Overview

Agenda

Registration

Contact

✉ courses@bwhpc.de

Das Steinbuch Centre for High-Performance Computing (HPC) (Hochleistungsrechnen) (Performance Computing) (Veranstaltung richtet sich an die Anwender (User) (bwForCluster) und die Administratoren (Admin) (Speichersysteme. Die Teilnahme an der Veranstaltung fallen keine Teilnahmegebühren an.)

The Steinbuch Centre Performance Computing course is aimed at (f) information about advanced and tutorials, and advanced

Starts 21 Oct 2021
Ends 21 Oct 2021
Europe/Berlin

exercises

slides

How to do exercises?

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

```
$ ws_allocate bwhpc-course 30
Info: creating workspace
/pfs/work7/workspace/scratch/ab1234-bwhpc-course-0
remaining extensions : 3
remaining time in days: 30
```

- Copy examples to your workspace

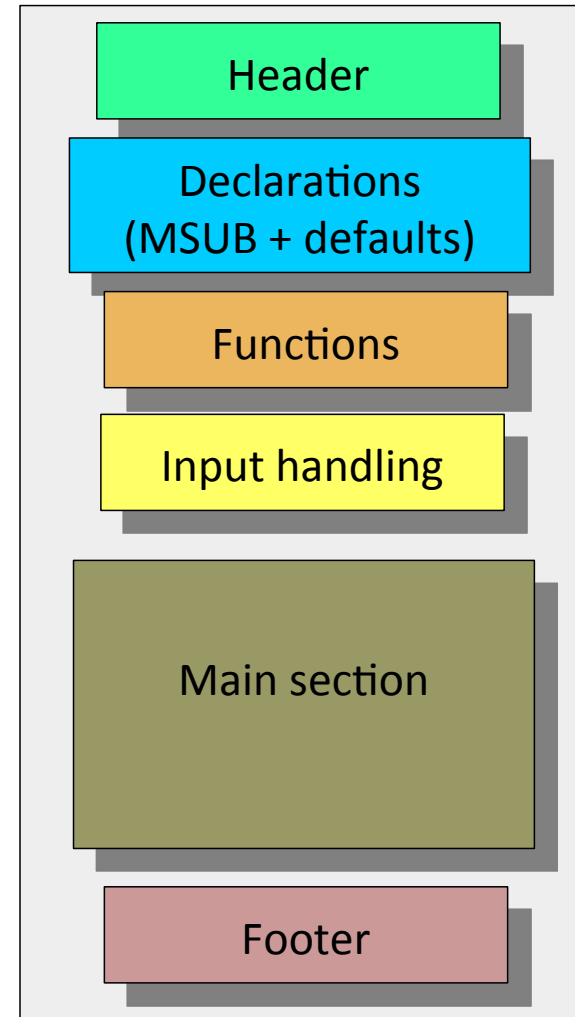
```
$ WORKSHOP=/opt/bwhpc/common/workshops/2021-10-27
$ cd $(ws_find bwhpc-course)
$ mkdir -v 2021-10-27; cd 2021-10-27
$ cp -vpr ${WORKSHOP}/exercises/02 ./
```

- Submit jobs from your workspace

```
$ cd $(ws_find bwhpc-course)/2021-10-27/02
$ sbatch <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 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 [sacct](#)
- For development use [develop queue](#)

SLURM variables (bwUniCluster 2.0, HoreKa)

■ bwHPC 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=50
#SBATCH -t 00:05:00
#SBATCH --mail-type=all
#SBATCH --export="All,my_own_variable=arguments"
```


MOAB variables (bwForC. NEMO & BinAC)

■ [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"
```

File system issues (1)

■ Multinode Job:

- For most cases

→ use *workspaces*

But: Producing Tbyte of scratch files & >10000 File?

Change your application code

Need help for that? Apply for [Tiger Team Support](#).

- A lot of I/O over all nodes

→ **opt out to local parallel file system** instead of global one

use BeeOND: but requires a „workflow“

■ 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 back to DIR where job was submitted
- 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}.${SLURM_JOB_ID:-$$}"

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

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

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

## f) Tutorial ToDo: check run status

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

## 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_job_run_under_local_tmpdir_solv.sh
```

Solution!

```
## 1) Define your binary
submitdir=${SLURM_SUBMIT_DIR:-${PWD}}
EXE="01_gen_files.x"

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

## 3) Define full path input files
Input="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}.${SLURM_JOB_ID:-$$}"
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_job_run_under_local_tmpdir_solv.sh
```

Solution!

```
## d) Change to Submit Dir or PWD / Copy files from submit_DIR to run_DIR
cd "${submitdir}"
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}" "${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. `sbatch -t D-HH:MM:SS`

■ Issue:

- Executable needs more time than given walltime
→ queueing system is terminating your jobscript and its child processes

■ Solution:

- `sbatch --signal=B:<sigint>@<seconds>` , e.g. 120 before walltime send `sigterm` (15)

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

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

Walltime Issues (2)

Solution!

- Use: „`sbatch --signal`“ and „`trap`“ to abort job on own terms

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

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

#SBATCH -n 1 -t 00:01:00
#SBATCH --mem=100
#SBATCH --signal=B:15@10
#SBATCH -p ws

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

trap cleanup 15

echo "Repeating \"sleep 1\" loop until SIGTERM"
i=1
while true ; do
    sleep 1; echo $i; let i+=1
done
```

Slurm 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_parbatch.sh
```

```
#!/bin/bash

#SBATCH -n 4 -N 1
#SBATCH --mem=150
#SBATCH -t 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 resource requirements
→ but different input (files)

- Interactive setup (aka „pure“ bash script setup):

- Master script:

- Translating index setup into list, executing each index value as a job

- Slurm:

- Available as submit feature:

```
sbatch --array [<indexlist>]:<delim> job.sh
```

→ makes master script obsolete & groups Job IDs (= easier to handle)

→ **job.sh** gets index value via `$SLURM_ARRAY_TASK_ID`

Job Arrays (2)

- Simple slurm array example: `${WORKSHOP}/exercises/02/05_slurm_array.sh`

```
#!/bin/bash

#SBATCH -J ws_array
#SBATCH -N1 -n2
#SBATCH -t 00:01:00
#SBATCH --array=0-10:2
#SBATCH --output=array_%A-%2a.out

printenv | grep "SLURM_ARRAY"

## Generate real number, e.g. via exp fct.
echo ${SLURM_ARRAY_TASK_ID} | awk '{printf "%.4f\n",exp($X)}'
sleep 2
```

List from 0 to 10
with increment of 2

Default is: `slurm-%A_%a.out`,
but generates here:
`slurm-12346_0.out`
...
`slurm-123456_10.out`

`%2a` = pads one-digit
numbers with zeros

current array task ID

Bash based Job Array (3)

■ Without array submit features → approach:

a.) handle each index value as one job/task

b.) handle as one (=master) sbatch job

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

```
export IARR="0-10:2"
```

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

```
#SBATCH --export="All,IARR=2-10:2" # index setup: min-max:inc
```

```
# Define subjob script
```

```
subjob="./05_array_task.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/-*}
```

```
ndm="${#max}" ## Number of digits of max
```

Automatic padding
digits with zeros

```
echo "Generate index list from ${min} to ${max} with increment ${inc}"
```

```
while [[ $min -le $max ]] ; do
```

```
    printf " Execute ${subjob} %0${ndm}d\n" "${min}"
```

```
    #${subjob} $(printf "%0${ndm}d" "${min}")
```

```
    let min+=${inc}
```

```
done
```

Job Array (4)

TASK/ToDo: 5min

- Modify 05_master_job_array.sh
 - To do parallel (use parbatch):
- 05_array_task.sh writes index value to indexed output

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

```
#!/bin/bash

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

## Define name of output file, array_<jobid>-<task-id>.out
outputfile="?"

## Write value to file
??
```

Job Array (4)

Solution!

- Modify 05_master_job_array.sh
 - To do parallel (use parbatch):
- 05_array_task.sh writes index value to indexed output

```
#{WORKSHOP}/solutions/02/05_array_task_solv.sh
```

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

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

```
#{WORKSHOP}/solutions/02/05_master_job_array_solv.sh
```

```
module load system/parbatch
...
joblist=joblist_${SLURM_JOB_ID:-$$}.txt
while [[ $min -le $max ]] ; do
    printf "  Execute ${subjob} %0${ndm}d\n" "${min}"
    echo "${subjob}" $(printf "%0${ndm}d" "${min}") >> ${joblist}
    let min+=${inc}
done
# Execute parbatch
parbatch ${joblist}
```

Chain Jobs (1)

- Idea:
 - Do N consecutive Jobs via N 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)

```
#!/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
```

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

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

Chain Jobs (3) → How for Slurm?

```
#!/bin/bash
#SBATCH ...
## 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 Slurm

Chain Jobs (4) → Solution! for Slurm

```
#!/bin/bash
#SBATCH -N1 -n1 -t 00:00:05 -mem=250 -p ws
## 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 [ $? -eq 0 ] ; then
        ## continue only if last command was successful
        sbatch --export=myloop_counter=${myloop_counter} ./02_chain_job_solv.sh
    else
        ## Terminate chain
        echo " ERROR: ${cmd} of chain job no. ${myloop_counter} terminated unexpectedly"
        exit 1
    fi
fi
fi
```

```
${WORKSHOP}/solutions/02/02_chain_job_solv.sh
```

```
$ sbatch --export=myloop_counter=0 ./02_chain_job_solv.sh
```

Chain Jobs (4)

■ Slurm + interactive script =

```
{WORKSHOP}/solutions/02/02_gen_chain_job_solv.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} ./02_gen_chain_job_solv.sh
  elif [ ! -z ${SLURM_JOB_NAME} ] ; then
    sbatch --export="myloop_counter=${myloop_counter}" ./02_gen_chain_job_solv.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) + `sbatch -d afterok:<jobID>`

- 1. script - links: `${WORKSHOP}/solutions/02/02_chain_link_job_solv.sh`

```
#!/bin/bash
#SBATCH ...

## 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_submitter_f_chain_job_solv.sh`

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

myloop_counter=1
while [ ${myloop_counter} -le ${max_nojob} ] ; do
  if [ ${myloop_counter} -eq 1 ] ; then
    slurm_opt=""
  else
    slurm_opt="-d ${dep_type}:${jobID}"
  fi

  echo "Chain job iteration = ${myloop_counter}"
  echo "  sbatch --export=myloop_counter=${myloop_counter} ${slurm_opt} ${chain_link_job}"
  ## Store job ID for next iteration by storing output of msub command with empty lines
  jobID=$(sbatch --export=myloop_counter=${myloop_counter} ${slurm_opt} ${chain_link_job}
    2>&1 | sed 's/[S,a-z]* //g')

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

Thank you for your attention!
Questions?