Workshop Agenda – Feb 25th 2015

Time	Presenter	Title
09:30	T. König	Talk – bwHPC Concept & bwHPC-C5 - Federated User Support Activities
09:45	R. Walter	Talk – bwHPC architecture (bwUniCluster, bwForCluster JUSTUS, ForHLR Phase I)
10:00	A. Fuchs	Talk – Cluster: Access, Data Transfer and Storage, GUI
10:30		*Break*
10:45	R. Barthel	Talk – File System, Software System (modulefiles), Batch System
11:10	A. Fuchs	Tutorial – bwUniCluster: Access, Data Transfer, Compiling, Modulefiles, Batch Job Scripting
11:50		*Lunch Break*
13:00	R. Barthel	Talk – Advanced Bash Scripting
13:30	R. Barthel	Tutorial – Advanced (Batch) Job Scripting
14:15		*Break*
14:30	A. Fuchs	Tutorial – Compiling, Makefile, Parallelising
15:15		User Forum – Solving User Cases
16:00		*End*

bw|HPC – C5

Tutorial:
Advanced (Batch) Job Scripting

Robert Barthel

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 -option 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

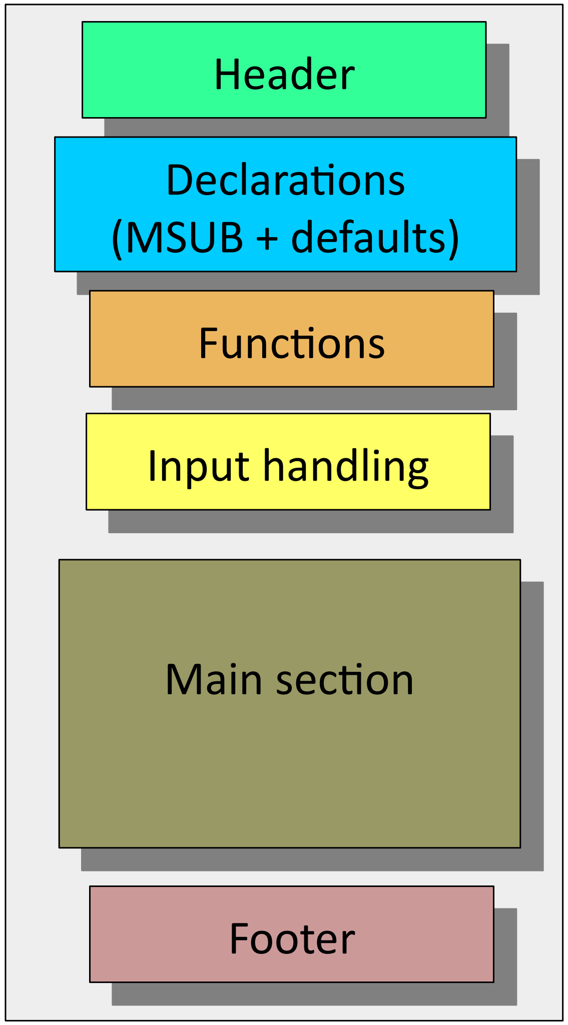
`foo, bar`	Metasyntactic variables
`${WORKSHOP}`	`/pfs/data1/software_uc1/bwhpc/kit/workshop/2015-02-25`

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 modules
    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 in `${HOME}`
- Multinode Job:
  - use `${WORK}`
  - Producing Tbyte of scratch files + <10000 File: Change your application code
- Singlenode Job:
  - use `${TMPDIR}`: HowTo → Case 1
- Chain jobs
  - HowTo → Case 2
- Many sequential tiny jobs:
  - Bundle to one big job: HowTo → Case 3

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?
  - Compare: Examples

```
${WORKSHOP}/exercises/bash/job_run_under_local_tmpdir.sh
+
${WORKSHOP}/exercises/bash/{gen_files,gen_files.inp}
```

Case 1: Jobs @ $TMPDIR (2)

- Code snip: `${WORKSHOP}/exercises/bash/job_run_under_local_tmpdir.sh`

```
#!/bin/bash

#MSUB -N test
#MSUB -l advres=workshop.54,nodes=1:ppn=1,mem=50mb
#MSUB -l walltime=00:05:00
#MSUB -m n

## a) Tutorial ToDo: load modules if necessary

## b) Define your run directory and generate it
mkdir -pv "${TMP}/${USER}.${MOAB_JOBID:-$$}"

## c) Tutorial ToDo: Check existance of run_DIR

## d) copy files from submit_DIR to run_DIR
cd $MOAB_SUBMITDIR
cp -pv gen_files gen_files.inp "${TMP}/${USER}.${MOAB_JOBID:-$$}"

## e) cd to run_DIR and start "binary" together with input file
cd "${TMP}/${USER}.${MOAB_JOBID:-$$}"
./gen_files 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
* Write code for a,c,f,h
* Generalise blue code
  avoiding repetition
* Redirect output of binary
```

Case 1: Jobs @ $TMP (3)

- Header: `${WORKSHOP}/exercises/bash/solutions/generalised_job_run_under_local_tmpdir.sh`

Solution!

```
. . .
## a) Load modules if necessary
module load compiler/intel
module load numlib/mkl

## b) Define full path of your binary
EXE="gen_files"

## c) Define output file
output="$(basename ${EXE})_${MOAB_JOBID:-$$}.log"

## d) Define input files
input="gen_files.inp"

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

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

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

Case 1: Jobs @ $TMP (4)

- Body + Footer: `${WORKSHOP}/exercises/bash/solutions/generalised_job_run_under_local_tmpdir.sh`

Solution!

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

## i) Copy files from submit_DIR to run_DIR
cd "${MOAB_SUBMITDIR}"
for X in ${copy_list} ; do
   cp -pv "${X}" "${run_DIR}"
done

## j) cd to runDIR and start binary
cd "${run_DIR}"
./$EXE ${input} > $output 2>&1

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

## l) Transfer files to submit directory
cd "${run_DIR}"
for X in ${save_list} ; do
   cp -pv "${X}" "${MOAB_SUBMITDIR}"
done

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

Case 2: Chain Jobs (1)

- Idea:
  - Split ***N*** consecutive Jobs into ***N*** MOAB Batch Jobs
- Goal:
  - Do everything in one script
  - Submit only once to MOAB

- „Pre-step“: generate script that runs interactively
  - Result:

```
${WORKSHOP}/exercises/bash/interactive_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
```

```
$ export myloop_counter=0
$ ./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} ./moab_chain_job.sh
   else
      ## Terminate chain
      echo "  ERROR: ${cmd} of chain job no. ${myloop_counter} terminated unexpectedly"
      exit 1
   fi
fi
```

```
$ msub -v myloop_counter=0 ./moab_chain_job.sh
```

Case 2: Chain Jobs (4)

- moab_chain_job.sh + interactive_chain_job.sh =

```
${WORKSHOP}/exercises/bash/solutions/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

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

- Prestep: Schedule N jobs to one slot:

```
TASK: 10min
* Write Loop that executes 10 times para_slave.sh but:

  a) the execution waits until file lock does NOT exist

  b) the execution generates the file lock

  c) the successful termination of para_slave.sh removes
     the lock file
```

```
USE the templates:
${WORKSHOP}/exercises/bash/para_master.sh
${WORKSHOP}/exercises/bash/para_slave.sh
```

Case 3: Pseudo Parallelisation (2)

Solution!

${WORKSHOP}/exercises/bash/solutions/para_master.sh

```
no_loop=10
slave_sh=para_slave.sh
lock_file=lock

## LOOP ##
i=1
while (( $i <= ${no_loop} ))  ; do
   ## If lock does not exists -> execute slave job
   if [ ! -e "${lock_file}" ] ; then
      ## generate lock
      touch "${lock_file}"
      ## generate logfile name
      log_file=${slave_sh%.sh}_${i}.log

      ## execute job + options
      echo "Starting Job $i of ${slave_sh}"
      ./${slave_sh} "${lock_file}" "${log_file}"
      ##
      ## increase counter
      let i+=1
   fi

   ## Do not finish current iteration until lock disappears
   while [ $(ls ${lock_file} 2>/dev/null | wc -l) -ge 1 ] ; do
      sleep 2
   done

done
```

Case 3: Pseudo Parallelisation (3)

Solution!

${WORKSHOP}/exercises/bash/solutions/para_master.sh

```
lock_file=${1}
log_file=${2}
cmd="sleep 2"          ## user command for this tutorial

## Execute command (cmd) and protocol execution time and finishing time to log file
echo "Starting  time: $(date)" > ${log_file}
${cmd} >> ${log_file} 2>&1

## Determine exit status of command
stat_cmd=$?

## Remove lock file
if [ "${stat_cmd}" -eq 0 ] ; then
   echo "Job successfully finished" >> ${log_file}
   rm -f "${lock_file}"
else
   echo "Job finished with error" >> ${log_file}
fi
echo "Finishing time: $(date)" >> ${log_file}
```

Case 3: Pseudo Parallelisation (4)

- Further generalisation of para_master.sh & para_slave.sh
  - inner loop over number of slots = number of core of MOAB job
  - Generate para_slave.sh automatically
  - Allow input your subjobs

  - Example: bwUniCluster_pseudo-parallel.sh
    - run with 2 cores per 2 nodes:

```
$ cp ${WORKSHOP}/exercises/bash/solutions/{bwUniCluster_pseudo-parallel.sh,jobfile.txt} .

$ ./bwUniCluster_pseudo-parallel.sh -h  # read help
$ nano jobfile.txt                      # edit jobs

$ msub bwUniCluster_pseudo-parallel.sh jobfile.txt
```

Batch jobs with input parsing

- Not working:
  - msub [options] *your_script* -x *argument*

    → msub will interprete -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
  - Do not use if your job solely does MPI:

    mpirun –machinefile=file binary

    mpirun -n <int> binary
  - Correct, (because the resource manager tells mpirun what to do):
    - mpirun binary

- 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!