# Using bash. Some handy features

Presenter: Kevin Stratford EPCC, The University of Edinburgh

kevin@epcc.ed.ac.uk

4 March 2026

archer2

epcc

# Using bash

- Everybody needs bash:
  - Interactive use (the terminal)
  - SLURM batch submissions

# Variables

- Environment variables

```
echo $SHELL          ➔ /bin/bash
echo $BASH           ➔ /bin/bash
echo $BASH_VERSION   ➔ 4.4.23(1)-release
```

- Use braces to delimit variable name

```
echo ${BASH_VERSION}  ➔ 4.4.23(1)-release
echo ${BASH}_VERSION  ➔ /bin/bash_VERSION
```

# Variables: assignment

- Variable assignment:

```
my_var="Hello world"        # Use quotes to preserve whitespace
my_number=7                 # a character/string
```

- Referring to variables

```
echo $my_var                ➔  Hello world
echo "my_var is $my_var"    ➔  my_var is Hello world
```

- Single quotes preserve string literally

```
echo 'my_var is $my_var'    ➔  my_var is $my_var
```

# Capturing output

- Of a string

```
my_number=7
result=$(echo "6 * $my_number")
echo $result                                  ➔  6 * 7
```

- Integer arithmetic:

```
result=$(( 6 * $my_number ))                  # note double parenthesis
echo $result                                  ➔  42
```

# Redirection

- Redirect stdout to file:

```
echo "Hello World!" > hello.txt
```

- Append stdout to file

```
echo "Goodbye World!" >> hello.txt
```

- Pipe results of one command into another:

```
squeue | grep " PD "
```

- View and save stdout to file using *tee:*

```
squeue | tee current_queue.txt
```

# Standard output and standard error

- Redirect standard output

```
module list > modules.txt
```

- Explicitly (descriptor 1 is output):

```
module list 1> modules.txt
```

- Redirect standard error (descriptor 2):

```
module list 2> modules.txt
```

- Redirect output to file and error to same place

```
module list > modules.txt 2>&1
```

- Redirect both to same place

```
module list &> modules.txt
```

# Search and replace

- Search and replace using /../.. (first match):

```
echo ${HOME}                    ➔ /home/z19/z19/kevin
echo ${HOME/home/work}          ➔ /work/z19/z19/kevin
```

- Match a special character using escape \:

```
echo ${HOME/\/${USER}}          ➔ /home/z19/z19
```

- Match all occurrences using //

```
echo ${HOME//z19/n02}           ➔ /home/n02/n02/kevin
```

- Match at end using %; e.g., strip off file extension:

```
file="name.of.file.ext"
echo ${file/%.*}                ➔ name.of.file
```

# Conditionals

- Conditional expressions, e.g.,:

```
if [[ -e "$file" ]]; then echo "File exists"; fi
if [[ -d "$file" ]]; then echo "Exists and is a directory"; fi
```

- Logical not

```
if [[ ! -e "$file" ]]; then "Does not exist"; fi
```

- General form:

```
if [[ condition ]]; then
   #  .. true ..
elif [[ otherwise ]]; then
   # .. zero or more other conditions ...
else
   # ... optional else block ..
fi
```

# Logical equivalence

- For strings:

```
if [[ "$var" == "YES" ]]; then echo "Yes!"; fi
```

- For integers

```
if (( "$var1" == "$var2")); then echo "Equal"; fi
```

- May also see single [ ]:

```
if [ condition ]; then
```

# Iteration

- Use e.g,:

```
list="1 2 3"
for item in $list; do
  echo "$item"
done
```

- C-style, e.g.,

```
for (( item=0; item<=3; item++ )); do
  echo "$item"
done
```

# Arrays

- Initialiser:

```
array=(red white blue)                    # 3 elements
echo ${#array}                            # Print number of elements
```

- Indexing (note braces)

```
echo ${array[0]}                          ➔ red
echo ${array[@]}                          ➔ red green blue
```

- Iteration, e.g.,

```
for hue in ${array[@]}; do echo $hue; done
```

- C-style, e.g.,

```
for (( c=0; c<${#array}; c++ )); do echo ${array[$c]}; done
```

# Real numbers

- No real numbers (only strings):

```
four=4.0
pi=3.14
```

- Require external facility to perform arithmetic, e.g.:

```
result=$(echo "$four * $pi" | bc)          # binary calculator
echo   $(echo "$four * $pi" | bc)          # 12.56
```

- C-style printf-like function is available to format

```
printf "Result %12.5e\n" $result          ➔ Result is 1.25600e+01
```

# A bash script

- Longer, reusable series of commands can be placed in a script :

```
#!/usr/bin/env bash
# The first line is typically the "shebang"

set -e   # Stop on error

# ... content ...
```

- Run the script, e.g.,

```
bash ./my_script.sh
chmod 700 my_script.sh
./my_script.sh
```

# A SLURM submission script

- A bash script with special comments to be interpreted by slurm:

```
#!/bin/bash
#
#SBATCH --export=none
# ...
# Normal bash commands ...
```

- Submit the script, e.g.,

```
sbatch my_script.sh
```

# Summary

- bash is the essential tool you cannot do without on ARCHER2

# Reusing this material

# Partners

epcc


epcc


THE UNIVERSITY of EDINBURGH

**Hewlett Packard** Enterprise