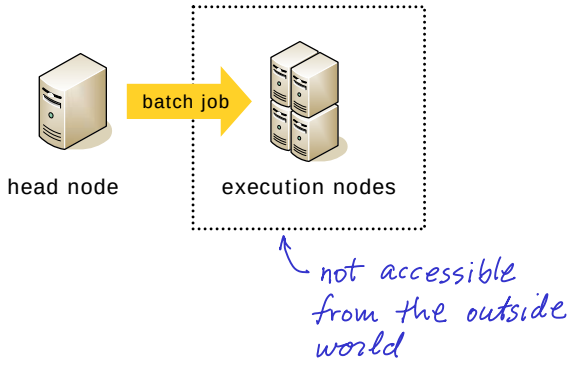
Shell Scripting

With Applications to HPC

Edmund Sumbar
research.support@ualberta.ca

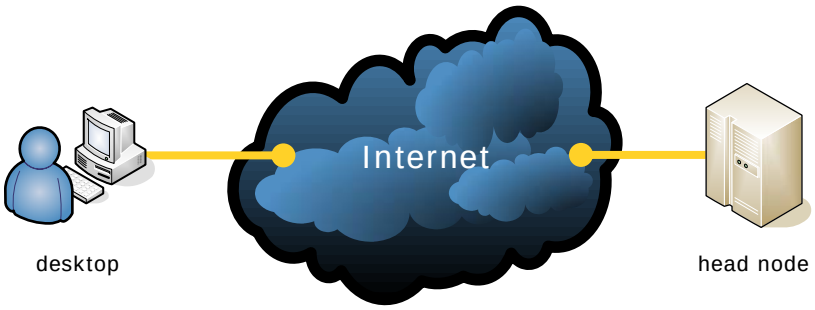
- High performance computing environment (1)



create delete

- High performance computing environment (2)
  - Portable Batch System (PBS)
  - Batch job described by a PBS script
  - PBS script is a shell script
  - Therefore, a batch job is a shell script
  - Also, shell scripts for utility purposes

- What is a shell (1)



- What is a shell (2)
  - ssh client (on desktop) connects to ssh server (on head node)
  - User authentication (login)
  - A “shell” program is launched on head node (login shell)
  - Local ssh client displays the results from the shell
  - Allows you to execute commands interactively on the head node

- Two major shell program variants
  - Bourne-like shells
    - **`sh`**
    - **`ksh`**
    - **`bash`**
  - C-like shells
    - **`csh`**
    - **`tcsh`**
- Login shell specified for each user account
  - Look for your **`CCID`** in **`/etc/passwd`**
  - **`echo $SHELL`**

- Very little difference for basic interactive use
- Non-interactive use
  - Execute a sequence of commands contained in a file (script)
  - More advanced usage
  - Differences in shell syntax become important
- Focus on the bash shell
- Reference documentation
  - `man bash`
  - `info bash`

- Example commands (* internal command)
  - `cd*`
  - `cp`
  - `mv`
  - `rm`
  - `mkdir`
  - `ls`
  - `cat`
  - `diff`
  - `find`
  - `grep`
  - `read*`
  - `chmod`
  - `awk`
  - `ps`
  - `kill*`
  - `sleep`
  - `echo*`
  - `exit*`
  - `qsub`
  - `qstat`

- Types of commands
  - Simple
    - `ls ~/src > filelist`
    - `test -e mhd.c`
  - Pipeline (sequence of one or more simple commands)
    - `cat mhd.c | grep MPI | grep -v Bcast`
  - List (sequence of one or more pipelines)
    - `cd ~/src ; test -e mhd.c && cp mhd.c mhd2.c &`
  - Compound commands (act on lists)
    - `if ! grep MPI mhd.c ; then echo serial code ; fi`
    - `for f in * ; do test -L $f && rm $f ; done`

a newline character can be used to replace a semi colon

- Three ways to execute a script
  - If script syntax is appropriate for current shell, set executable bit on the file (**`chmod`**), and run script file directly
  - Run shell program with script file as argument (**`bash scriptfile`**)
  - Specify the path to the shell program on the first line, set the executable bit, and execute the script directly (**`#!/bin/bash`** same effect as **`/bin/bash scriptfile`**)

PBS runs a job by executing the job script on a node

- Workshop exercises (1)
  - Execute the following sequence of simple commands in a shell script using each of the three techniques.

```
date
uname –n
id
```


- Exit status
  - Return value of...
    - simple command
    - last command in pipeline
    - last command executed in list
    - last command executed by compound command
    - last command executed by a script
  - **`echo $?`**
    - **`0`** success
    - **`1..128`** failure
    - **`128+n`** termination due to signal

- Workshop exercises (2)
  - What's the difference between a command that crashes and a command that fails?
  - Compare the exit status of these commands

```
ls -l .
ls -l zzz
/home/esumbar/crasher
```


- Execution modes
  - Synchronous
    - Command terminated by semicolon (**;**) or newline
    - Shell waits for command to terminate
    - Shell returns with command's exit status
  - Asynchronous
    - Command terminated by ampersand (**&**)
    - Shell does not wait for command to finish
    - Command runs in the background
    - Shell returns with an exit status of zero (not the exit status of background command)

- Special characters and words
  - Metacharacters (delimit words) **`| & ; ( ) < > space tab`**
  - File name pattern matching characters **`* ? [`**
  - Parameter expansion character **`$`**
  - Quoting characters **`‘ “ \`**
  - Tilde character **`~`**
  - Reserved words that have a special meaning
    `! case do done elif else esac fi for function if in select then until while { } time [[ ]]`

- Parameters and parameter expansion
  - Named parameters (variables)
    - **`remotehost=num.srv.ualberta.ca`**
      `ping -c 1 ${remotehost} > /dev/null`
  - Positional parameters (arguments passed to script)
    - Assigned automatically
    - **`echo ${1} ${15}`**
  - Special parameters and shell variables
    - Assigned automatically
    - **`echo ${?}`**
    - **`cd ${HOME}`**

no spaces around "=" otherwise shell would try to execute a command called "remotehost" with "=" and "num.srv" as arguments

alternatively use "declare" command

braces are optional they are required to resolve ambiguities

Shell variables (environment variables) (1)

- All uppercase names (by convention)
- Used to define attributes
  - **PATH** search path for commands
  - **HOME** home directory
  - **PWD** current working directory
- Inherited by subcommands
- Add variable to environment
  - `EMAIL_ADDRESS=research.support@ualberta.ca`
    `export EMAIL_ADDRESS`
- Display with **printenv** or **env**

Shell variables (environment variables) (2)

- PBS augments a script's environment
- **PBS_O_WORKDIR** original working directory
  - `cd $PBS_O_WORKDIR`
- **PBS_NODEFILE** name of file containing the list of exec nodes assigned to the job
  - `mpirun -machinefile $PBS_NODEFILE -np $NP ./a.out`
- **PBS_JOBID** useful for uniquely naming output files
  - `./a.out > output.$PBS_JOBID`
- Pass additional environment variables using **-v** or **-V** options
  - `qsub –v BASE=linear,TARGET=3600 scriptfile`

- Command substitution
  - Standard output of command used as a string
  - Old style (still valid)
    - `echo "current date and time is `date`"`
  - New style
    - ```
      for file in $(ls $HOME/core*)
      do rm –f $file
      done
      ```


- Workshop exercises (3)
  - Assign the output from a command to a parameter
    - `nfiles=$(ls –a | wc -l)`
  - Recall the value of the parameter later
    - `echo "number of files is $nfiles"`
  - Try to recall the value of **`nfiles`** from a script
      ```

- Shell arithmetic
  - **x=45**
    - Internal command **let "x = x * 5"**
    - Compound command **((x = x * 5))**
    - Expansion **x=$((x * 5))**
  - **echo $x**
    225

- Comments used in scripts
  - Introduced by **#**
  - Everything to the end of the line is ignored
  - PBS directives (**#PBS nodes=2**) are shell script comments
    - **qsub** scans the script for PBS directives up to the first shell command that it encounters to determine job specs
    - Any command line args to **qsub** override PBS script directives

Quoting

- Full (using single quotes)
  all special characters and words lose their special meaning
  - `prompt=greetings`
    `echo '$prompt: $(date)'`
- Partial (using double quotes)
  only **$** and \ retain their special meaning
  - `args="file1 file2"`
    `test "$args" = "file1 file2" && rm -f $args`
- Escaping (using backslash)
  remove special meaning of single character
  - `echo "value of \$prompt: $prompt"`

I/O redirection (1)

- Three open files/file descriptors
  - Standard input (default stdin from keyboard) **0**
  - Stardard output (default stdout to terminal) **1**
  - Stardard error output (default stderr to terminal) **2**
- Examples...
  - `0 < inputfile`
  - `1 > outputfile`
  - `>> outputfile`
  - `2 > /dev/null`
  - `&> outputfile`
  - `>outputfile 2>&1`

can appear before the command too

equivalent

- I/O redirection (2)
  - PBS redirects stdout and stderr from the job script to files located on the execution node
  - At the end of the job, by default, PBS copies these files from the execution node to the original working directory as **`<job name>.o<job id>`** and **`<job name>.e<job id>`**
  - Use the **`-o`** and/or **`-e`** qsub options to specify alternate files

- Workshop exercises (4)
  - Explain the output from this shell command
    - **`echo 2 * 3 > 5 is a valid inequality`**

Functions

- Group command (a type of compound command)
  -
```
test -e $resultsfile && {
    mkdir $resultsdir
    mv $resultsfile $resultsdir
    echo moved $resultsfile to $resultsdir
    }
```
- A function is a named group command
  -
```
function store {
    mkdir $resultsdir
    mv $resultsfile $resultsdir
    echo moved $resultsfile to $resultsdir
    }
  test –e $resultsfile && store
```


Branching (1)

- **if**-command branches on the basis of exit status
  -
```
if [ $denominator -ne 0 ]
then
    value=$(( $numerator/$denominator ))
else
    echo “divide-by-zero condition”
fi
```
- `[ $denominator –ne 0 ]`
- `test $denominator -ne 0`

these are not brackets

equivalent

Branching (2)

- **case**-command branches on the basis of pattern matches
  - ```
    case $(uname -p) in
        mips    ) echo "SGI machine" ;;
        powerpc ) echo "IBM machine" ;;
        *86     ) echo "Intel machine" ;;
        *       ) echo "unknown processor" ;;
    esac
    ```


Looping (1)

- **for**-command loops over items in a "word" or "token" (a sequence of characters considered as a single unit)
- Environment variable IFS defines the characters used to split the word into items (by default **<space> <tab> <newline>**)
  - ```
    for file in $(ls $PBS_O_WORKDIR/temp)
    do
        rm -f $file
    done
    ```

Looping (2)

- **while**-command loops on the basis of exit status
  - **while :**

```
do
    ./a.out
    test -e output && break
done
```

a command that does nothing but return zero exit status — see also 'true'

Workshop exercises (5)

- Say that you've parameterized your PBS script to accept three environment variables. You plan to use the **-v** option to specify the values of these variables when you submit your jobs, and you'd like to take the parameter values from a file.

  Create a utility bash-shell script that formats an appropriate **qsub** command line for each line of the parameter file and saves all the command lines in a file that can be later executed as a script to submit the jobs.

- Workshop exercises (5)
  - PBS script

```
#!/bin/bash
#PBS –l walltime=02:00:00

cd $PBS_O_WORKDIR

./a.out $varx $vary $varz > output.$PBS_JOBNAME
```

  - Parameter file

```
blue A 45
blue A 89
green K 35
yellow Q 51
```

  - Result file

```
qsub –v varx=blue,vary=A,varz=45 script.pbs
qsub –v varx=blue,vary=A,varz=89 script.pbs
...
```


- Workshop exercises (5)

```
#!/bin/bash

src=$1 #---parameter file
pbs=$2 #---pbs script file
res=$3 #---result file

exec 0< $src

while read x y z; do
    echo “qsub –v varx=$x,vary=$y,varz=$z $pbs” >> $res
done
```

  - `./format.sh param.file pbs.script result.file`

- Workshop exercises (5)

add a test for sufficient number of command line arguments

```
#!/bin/bash

src=$1 #---parameter file
pbs=$2 #---pbs script file
res=$3 #---result file

test -e x$src || exit
test -e x$res && rm –f $res

exec 0< $src

while read x y z; do
    test -n “$x” || continue
    test -n “${x##\#*}” || continue
    echo “qsub –N $x$y$z –v varx=$x,vary=$y,varz=$z $pbs” >> $res
done
```

in case $src is empty

skip blank lines

skip comment lines

add "chmod u+x $res"