# TCCM lectures – Advanced Computational Techniques

Peter Reinhardt

Laboratoire de Chimie Théorique, Sorbonne Université, 75252 Paris CEDEX 05,

Peter.Reinhardt@upmc.fr

# Monday afternoon — I

- `bash` programming
- use of variables and quotes "...", ’...’, ‘...‘
- loops, conditions
- set, echo, head, tail, wc, grep, awk, sed, pipes (|)
- I/O, “here documents”

# Monday afternoon — I

```
:
A=`date`
B=`hostname`
D=$USER
D=$LOGNAME
OUT='version.h'
if [ "$#" -eq 0 ]
then
  echo
  echo " usage: stamp_unix a b [output_file]"
  echo
  echo " a and b denote two arbitrary strings"
  echo " default output_file is 'version.h' "
  echo
  echo " this file would contain: "
  echo
  echo "      write(6,*) 'a b' " >  $OUT
  echo "      write(6,*) '$A' "    >> $OUT
  echo "      write(6,*) '$D@$B' " >> $OUT
  echo
  echo
exit
fi
```

# Monday afternoon — I

```
C=$1
F=$2
if [ "$#" -ge "3" ]
then
OUT=$3
else
OUT=version.h
fi
echo "      write(6,66001) " > $OUT
echo "      write(6,*) ' $C $F '"  >>  $OUT
echo "      write(6,*) ' $A '"     >> $OUT
echo "      write(6,*) ' $D@$B '"  >> $OUT
echo "      call system(' echo \"  execution host is : \`hostname\`\"') " >> $OUT
echo "      write(6,66001) " >> $OUT
echo "66001 FORMAT(1X,79('=')) "       >> $OUT
exit
~
~
```

- \ hides the next character from interpretation
- simple text file
- execute a script with `sh script`

# bash

- a bash script starts with `:`
- `exit` terminates the execution
- indentation has no influence
- all commands are lowercase
- all text after a `#` in a line is considered as comment
- if a line becomes too long, you may break it with `\`
- `$` has a special meaning
- if a string is a number or text depends on the context
- very powerful tool for multiple execution of commands

# bash

Variables and quotes

- give a value to a variable: `A=13`, `FILE=output`
- use a variable: `B=$A` or `OUT=$FILE$A`
- attention: `$FILE13` will look for a variable of name `FILE13`, use in this case `$FILE\13`
- back quotes ‘...‘ insert the output of a UNIX command
- single quotes delimit a string without substitution of variables
- double quotes expand variables in a string

Exercise: try

```
FILE='out'
echo 'output file is $FILE'
echo "output file is $FILE"
```

# bash

Loops:

```
:

for I in a b c
do
 echo $I
done
```

Replace `a b c` with `*`. What happens?

Another one:

```
:

A=1
while [ $A -le  10 ]
do
 echo $A
 A=`expr $A + 1`
done
```

# bash

Conditions and branchings

- construction `if [ condition ] ... then ... elif ... else ... fi`
- binary operators ==, `-le` etc
- attention to compare numbers with numbers and strings with strings

```
EQ=`echo $R1 $R2 | awk '$1 == $2 {print "0"}; $1 != $2 {print "1"}'`
if [ $EQ -eq '1' ]
then
  ...
fi
```

- several choices: `case $VAR in` ... `esac`

# bash

```
PAGE=1

while [ $# -ge 2 ]
do
A=$1
case $A in
          -p)
            PAGE=$2
            shift
            shift;;
          *)
            break;;
esac
done
```

# bash

Replacements:

- replace `tst.com` with `tst.out` in a variable:
  `FILE='tst.com'`
  `echo $FILE | sed -e 's+.com+.out+'`
  invokes the stream-line editor `sed`.
- `sed -e 's+.com+.out+' $FILE` replaces `.com` by `.out` in every line of the file `$FILE`
- `awk` allows for applying an action on every line matching a pattern or condition
- try `awk 'BEGIN {count=0};{count=count+NF};END {print count}' a1`
- compare to the command `wc a1`

# Useful UNIX commands

`gzip`, `tar`, `grep`, `head`, `tail`

- `gzip` compresses a file, options `-v` (verbose) and `-d` (decompress)
- `tar` generates an archive; options `-cvzf` (create, verbose, zip, file), `-tvzf` (list, verbose, zip, file) or `-xvzf` (extract, verbose, zip, file)
- for example: `tar -xvzf some_files.tar.gz`
- `grep pattern file(s)`, find lines in `file` which match `pattern`. Options
  - `-v` : show lines which do not contain `pattern`
  - `-i` : do not distinguish upper and lower case
  - `-r` : go recursively through sub-directories
  - `-n` : add line numbers
- `head`, `tail` shows the first (last) lines of a file. Option `-n` *n* gives the first (last) *n* lines
- get the 5th line of a file: `head -n 5 file | tail -n 1`

# Input/output

- `>` redirects the output stream to a file; the file is created or overwritten
- `>>` appends the output stream to a file
- `<` provides input to a command from a file
- `command < input > output` or `command > output < input`
- give input data within the script: "here document"

```
    command > output <<!
data1
data2
!
```

```
  FILE=f_x_tau_$C\_$I
  echo $taumin $FILE >> integ_$C\_$I
   integrate_1D << ! | grep 1D >> integ_$C\_$I
$taumin 150.
$FILE
!
```

# Exercises

take the files from the web site

- extract all energies from the given output file
- extract angles for the O−C−O configurations
- ...