# UNIX/Linux Operating System

## Bash script exercises

Stefano Quer, Pietro Laface, and Stefano Scanzio

Dipartimento di Automatica e Informatica

Politecnico di Torino

skenz.it/os          stefano.scanzio@polito.it

# Exercise

❖ Write a bash script that computes the values of a function f(x) for all the triples of integer values stored in a file

➢ $f(x) = 3 \cdot x^2 + 4 \cdot y + 5 \cdot z$

➢ Example

| 1 | 1 | 2 | **17** |
|---|---|---|--------|
| 2 | 1 | 3 | **31** |
| 1 | 3 | 4 | **35** |

Output values

File content

➢ The name of the file must be passed from command line
➢ Write two versions of the script using **while** and **for** statements, respectively

# Solution 1

Using a **for** loop

Reads from file **one value at a time** because the output of the command goes in a list of strings

```bash
#!/bin/bash

flag=1
for val in $(cat $1)
do
    if [ $flag -eq 1 ]
    then
        let f=3*val*val
    elif [ $flag -eq 2 ]
    then
        let f=f+4*val
    elif [ $flag -eq 3 ]
    then
        let f=f+5*val
        flag=0
        echo -n "$f "
    fi
    let flag=flag+1
done

exit 0
```

# Solution 2

Using the **while** loop
(reads a line at a time)

Reads a line and the
string is assigned to
variable **line**

```
#!/bin/bash

while read line
do
    flag=1
    for val in $line
    do
```

Parsing the line

```
        if [ $flag -eq 1 ]
        then
           let f=3*val*val
        elif [ $flag -eq 2 ]
        then
           let f=f+4*val
        elif[ $flag -eq 3 ]
        then
           let f=f+5*val
        fi
        let flag=flag+1
    done
    echo -n "$f "
done < $1
```

Loop on file lines

# Solution 3

Using the while loop
(reads three values at a time)

Values read from
file three at a time!

```
#!/bin/bash

while read x y z
do
   let f=3*x*x+4*y+5*z
   echo -n "$f "
done < $1

exit 0
```

Loop on file lines

# Exercise

❖ Write a bash script that displays the content of

  ➤ All files of the current directory
  ➤ With ".`c`" extension
  ➤ That include string "`POSIX`"

❖ Example of execution

  ➤ `./myScript .`
  ➤ `./myScript /home/bin`

# Solution

```bash
#!/bin/bash
for file in $(ls *.c); do
  grep --quiet "POSIX" $file
  if [ $? -eq 0 ]
  then
    more $file
  fi
done
exit 0

# Alternative (single command):
# more $(grep  POSIX *.c -l)
# Notice the difference !!:
# grep -l POSIX *.c | more
```

grep
1) -q, --quiet, avoids printing the found line
2) If a file is found, returns (echo $?) 0 i.e., the condition is TRUE

grep -l
displays only the filenames
matching the string POSIX

# Exercise

❖ Write a bash script that

  ➢ Takes a filename from command line

  ➢ The file contains two columns of data

  ➢ Example

      7   3

      2  23

      5   0

  ➢ The script must overwrite the file swapping the two columns

  ➢ Note that output and input files are the same

# Solution

Uses a temporary file …

```bash
#!/bin/bash

file="tmp"

while read var1 var2
do
  echo $var2 $var1
done <$1 >$file

mv -f $file $1

exit 0
```

… renamed at the end of the script

# Exercise

❖ Write a bash script that

➢ Takes a filename from command line

➢ Displays the file content

▪ A line at a time, prepending the line number

▪ A string at a time, prepending the string number

# Solution

```
#!/bin/bash
n=1
while read line          # read a line
do
  echo "$n: $line"
  let n=n+1
done < $1                # Redirection !
n=1
for str in `cat $1`      # read a word
do
  echo "$n: $str"
  let n=n+1
done
```

# Exercise

❖ Write a bash script that

➢ Takes a filename from command line

➢ Reads a sequence of integer number from the file

➢ Each number represents a histogram bin value

➢ Displays a horizontal histogram using '*'

➢ Example

| | |
|---|---|
| 1 | * |
| 3 | *** |
| 5 | ***** |
| 4 | **** |
| 2 | ** |

File content

Output

# Solution

```
#!/bin/bash

for n in $(cat $1)
do
  i=1
  while [ $i -le $n ]
  do
    echo -n "*"
    let i=i+1
  done
  echo
done

exit 0
```

Reads a number at a time

Prints without newline

Prints a newline

# Exercise

❖ **Write a bash script that**

➢ Takes a set of strings from command line

➢ The first string is a directory name

➢ The others are filenames

- `$ myScript dir file1 file2 ... filen`

❖ **The script**

➢ Creates the directory if it does not exist

➢ For each file, ask the user if the file should be copied in the destination directory `dir`

➢ Copy only files confirmed by the user

# Solution

```
#!/bin/bash

if [ $# -le 1 ]
then
  echo "Run: $0 dir file1 file2 ..."
  exit 1
fi

if [ ! -d $1 ]
then
  echo "Create directory $1"
  mkdir $1
fi
```

# Solution 1

```
for i in $*
do
  if [ $i != $1 ]
  then
    echo -n "$i in $1 (y/n)? "
    read choice
    if [ $choice = "y" ] ; then
      cp $i $1
      if [ $? ]
      then
        echo "Copy done for $1/$i"
      else
        echo "Error for $1"
      fi
    fi
  fi
done

exit 0
```

P.S.: $* does not include the name of the program

Skip the first parameter

# Solution 2

```
dir=$1
shift
for i in $*
do
  echo -n "$i in $dir (y/n)? "
  read choice
  if ["$choice" = "y" ] ; then
    if cp $i $dir
    then
      echo "Copy done for $dir/$i"
    else
      echo "Error copying $i"
    fi
  fi
done
exit 0
```

The command line arguments are shifted to the left

**Exercise: from exam**

Exam Italian course: 2018/01/30

❖ The **df** file command shows the disk space available on the file system containing files

❖ Example

```
df /data/backup
Fifesystem 1K-blocks Used Avaifable Use% Mounted on
/dev/sda7 41L1A492 5881472 33174616 16% /data
```

➢ The second, third and fourth fields show the total space used and available on the flle system containing /data/backup

➢ Fields are separated with spaces

➢ Suppose no other separating character is used and spaces do not appear anywhere else

Exam Italian course: 2018/01/30             **Exercise: from exam**

❖ Write a script that receives the path of a source file and a destination path, and

  ➢ Check the correct passage of the parameters to the script

  ➢ Make in background a copy of the source file in the destination path

  ➢ Analyze the space occupied on the destination path at regular intervals of one second, displaying on the screen the percentage of progress of the copy operation

The command **sleep n** can be used to block the script for **n** seconds

# Solution

```bash
#!/bin/bash
if [ $# -ne 2 ]; then
  echo "Usage $0 <source> <destination>"
  exit 1
fi
if [ ! -f $1 ]; then
  echo "Source is not a valid file."
  exit 1
fi
if [ ! -d $2 ]; then
  echo "Destination is not a valid directory."
  exit 1
fi
source=$1
destination=$2
```

Check the number of parameters

Check the validity of the parameters

# Solution

```
size=$(ls -l $source | cut -d " " -f 5)
let "size=size/1024"

startUsed = $(df $destination | \
              tail -n 1 | \
              tr -s " " | \
              cut -d " " -f 3)

cp $source $destination &

transferred=0
percentage=0
```

Calculate file size in 1KB blocks

Calculate destination file system size

Copy in background

# Solution

Check the state of the copy in background

```
while [ $transferred -lt $size ]; do
  currentUsed = $(df $destination | \
                tail -n 1 | \
                tr -s " " | \
                cut -d " " -f 3)
  let "transferred=currentUsed-startUsed"
  let "percentage=transferred*100/size"
  echo "Progress: $percentage%"

  sleep 1
done
```

**Exercise: from exam**

Exam Italian course: 2018/02/22

❖ A script receives the following parameters
  ➢ the name of a file (fn) and three integers (n1, n2, and n3)
  ➢ The file (fn) specifies a path on each line
❖ The script must
  ➢ Verify that the 4 parameters are correct, i.e., integer numbers must be positive, and n1≤n2
  ➢ For each row of the file (fn)
    ▪ Check that each string refers to a regular file

**Exercise: from exam**

Exam Italian course: 2018/02/22

➤ If the dimension of the file (fn) is

- Smaller than n1 bytes, delete it
- Between n1 and n2 bytes, ignore it
- Greater than n2 bytes, compress it. Compress a file means
    - Make a copy in a file with the same path but with the additional extension (e.g., .compressed)
    - Modify the contents by copying only one string every n3 strings (ie just copy strings in position 1, 1 * n3, 2 * n3, etc.). Consider strings separated by spaces or by "newline" characters

# Solution

```bash
#!/bin/bash
if [ $# -ne 4 ]; then
  echo "Usage $0 <list> <n1> <n2> <n3>"
  exit 1
fi
if [ ! -f $1 ]; then
  echo "List is not a valid file."
  exit 1
fi
if [ $2 -lt 0 ] || [ $3 -lt 0 ] || [ $4 -lt 0 ]; then
  echo "Values n1, n2 and n3 should be non-negative integers."
  exit 1
fi
if [ $2 -gt $3 ]; then
  echo "Values n1 should be non-greater than n2."
  exit 1
fi
```

Check the number of parameters

Check the validity of the parameters

# Solution

For each path read from file

```
while read file; do

  if [ ! -f "$file" ]; then
    echo "Invalid file: $file"
    continue
  fi

  size=$(cat $file | wc -c)

  if [ $size -lt $2 ]; then
    rm -f $file
```

Skips paths not associated to regular files

Computes the dimension of the file

Removes small files

# Solution

```
    elif [ $size -gt $3 ]; then
      i=1
      for word in $(cat $file); do
        let "i--"
        if [ $i -eq 0 ]; then
          echo $word >> $file".compressed"
          i=$4
        fi
      done
    fi
done < $1
```

Compresses big files

**Exercise**

Exam Italian course: 2013/01/28

❖ Write a bash script that

➢ Takes a filename (of a text file) from command line

➢ **Copy** the file with the same filename, but with extension `xyx`

➢ **Modifies** the original file

  ▪ Adding at the beginning of each line the number of words in the line, and the total number of lines of the file

  ▪ Sorting the lines according to their number of words

# basename command

❖ **Syntax:**

➢ `basename NAME [SUFFIX]`

➢ Prints `NAME` with **any leading directory** components **removed**. If specified, it will also remove a trailing `SUFFIX` (typically a file extension)

```
> name=$(basename /home/user/current/file.txt)
> echo $name
file.txt

> name=$(basename /home/user/current/file.txt ".txt")
> echo $name
file
```

# Solution

```bash
#!/bin/bash
if [ $# -ne 1 ]
then
  echo "usage $0 file.txt"
  exit 1
fi
newfilename=$(basename $1 ".txt")
newfilename=$newfilename".xyz"
cat $1 > $newfilename
nlines=$(cat $1 | wc -l)
rm -f tmp1.txt
while read line
do
  nwords=$(echo $line | wc -w)
  echo $nwords $nlines $line >> tmp1.txt
done < $1
cat tmp1.txt | sort -k 1 -n > $1
rm tmp1.txt
exit 0
```

Filename without extension

".txt"="*.txt"=.txt

Copy file. Also:
`cp $1 $newfilename`

Also:
`nlines=$(wc -l < $1)`

Add information on a temporary file

Sort and overwrite the original file

Clean-up

**Exercise**

Exam Italian course: 2014/02/03

❖ Write a bash script that

➢ Takes 4 arguments (**dir1**, **dir2** e **dir3**, directory names, and **n**, an integer number)

➢ Finds in **dir1** and **dir2** all files that have the same name, extension **txt** and more than **n** lines

➢ Creates in directory **dir3** a version of these files with extension

▪ **eq** save the lines that are equal in both files

▪ **dif** save the lines that differ in the two files

▪ **cat** concatenates the content of the two files

Control the number of parameters.
Create directory **dir3** if it does not exist

# Solution

```bash
#!/bin/bash

if [ $# -ne 4 ]
then
  echo "usage: $0 dir1 dir2 dir3 n"
  exit 1
fi
if [ ! -d $3 ]
then
  mkdir $3
fi

for file in $(ls $1/*.txt); do
  name=$(basename $file ".txt")
  if [ -f "$2/$name.txt" ]; then
    n1=$(cat $file | wc -l)
    n2=$(wc -l < "$2/${name}.txt")
    if [ $n1 -gt $4 -a $n2 -gt $4 ]; then
```

find rather than ls
`find $1 -maxdepth 1 -type f -name "*.txt"`

It was enough to remove the path

For each .txt file in the first directory, generate the corresponding name in the second directory

Counts and controls the number of lines

# Solution

Lines in file1 and file2 go in eq, lines not in file2 go in dif

Control on the result of grep $?=0 (true or find)

Inverse control for lines potentially in dif

File concatenation

```
    while read line; do
      grep -q -e "^$line$" "$2/$name.txt"
      if [ $? -eq 0 ]; then
        echo $line >> "$3/${name}.eq"
      else
        echo $line >> "$3/${name}.dif"
      fi
    done < $file
    while read line; do
      grep -q -e "^$line$" "$3/${name}.eq"
      if [ $? -eq 1 ]; then
        echo $line >> "$3/${name}.dif"
      fi
    done < "$2/$name.txt"
    cat $file "$2/${name}.txt" > "$3/${name}.cat"
  fi
 fi
done
```

# More exercises...

❖ Other examples and small exercises about bash
  ➢ https://www.skenz.it/cs/bash_language