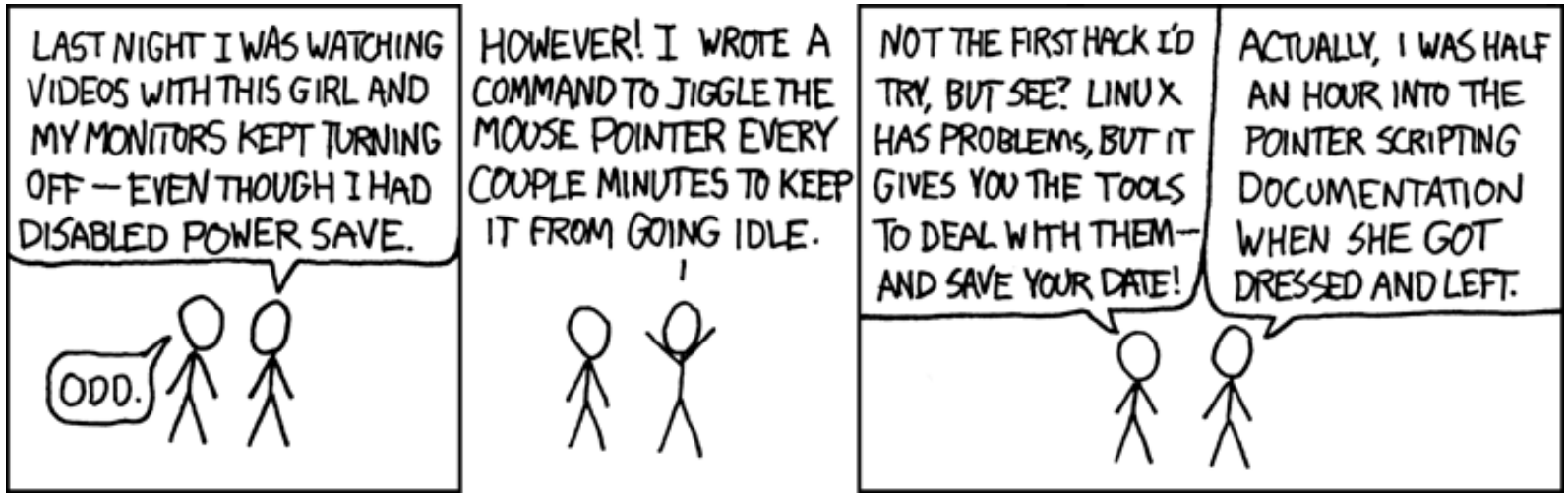
Manipolazione di file in Bash

(built-in commands, coreutils, tools)

P. Oliva

Università di Sassari

oliva@uniss.it



Prendere i dati

```
# mkdir olivaAHO
# cd olivaAHO
# scp  scuola@192.167.10.5:data.tgz .
scuola@192.167.10.5's password: scuola
# tar zxvf data.tgz
# cd Data
# ls
```

Obiettivo

- Conoscere la Bash ed alcune sue utilità
- Incrementare la produttività nella manipolazione di file e del loro contenuto
- Uso della linea di comando
- Anche io uso il Bash scripting!!!

Documentazione

- www.tldp.org (The Linux Documentation Project)
  - BASH Programming - Introduction HOWTO
  - Advanced Bash-Scripting Guide
- www.gnu.org (GNU Project)
  - Manuali

- man *bash*
- man *command*

Cos’è la shell

- La shell è un interprete di comandi.
- Un’interfaccia tra il kernel e lo user.
- Un potente linguaggio di programmazione
  - Script
  - Permette di mettere insieme chiamate di sistema, tool e utilità di sistema e binari compilati

Cos’è la Bash

- Bash è un acronimo per *Bourne-Again shell* dalla classic Bourne shell

- La Bash è diventata uno standard *de facto* per lo shell scripting su molti flavors di UNIX

Alcuni concetti introduttivi

- Redirezione I/O
- Pipe
- Espansione
- Variabili
- cp, mv, ls, mkdir, cd, pwd

Redirezione I/O

- Il classico programma di shell lavora come un filtro tra *stdin* e *stdout*

- **`command < file.in`**

l'input viene preso da *file.in* invece che da *stdin*

- **`command > file.out`**

l'output viene redirezionato da *stdout* a *file.out* (*file.out*, se presente, viene sovrascritto)

- **`command >> file.out`**

l'output viene redirezionato da *stdout* a *file.out* (se *file.out* è esistente, l'output viene aggiunto in fondo al file)

Pipe

- *command1* | *command2*

esegue *command1* e lo *stdout* di *command1* è usato come *stdin* di *command2*

- Molteplici comandi possono essere concatenati

```
cat *.txt | sort | uniq > result-file
# Sorts the output of all the .txt files and deletes duplicate lines
# finally saves results to "result-file".
```

Variabili

Il ***nome*** di una variabile è il contenitore del suo ***valore***, il dato memorizzato. Il riferimento a questo valore è chiamato *sostituzione di variabile*.

```
bash$ variabile=23

bash$ echo variabile
variabile

bash$ echo $variabile
23
```

Alcune variabili particolari:

`$RANDOM`	Contiene un numero pseudo-casuale
In uno script:	
`$0, $1, $2 ...`	Parametri posizionali
`$@`	Tutti i parametri posizionali (tranne **`$0`**)
`$#`	Numero di parametri posizionali (tranne **`$0`**)

Espansione

- L’interprete Bash, dopo l’identificazione delle parole che compongono la linea di comando, effettua le espansioni del caso.

- L’ordine delle espansioni è: ***brace expansion****, tilde expansion,* ***parameter, variable*** *and arithmetic expansion and* ***command substitution*** *(*done in a left-to-right fashion*), word splitting, and* ***pathname expansion***.

- Solo *brace expansion, word splitting, and pathname expansion* possono cambiare il numero di parole dell’espansione; le altre espandono una singola parola in una singola parola

Espansione

Brace expansion

- **`a{d,c,b}e  => ade ace abe`**

- **`{x..z} => x y z`** (caratteri o interi)

- Si possono nidificare
- **`A{b{1,2,4},c,g}FFF`**

`Ab1FFF Ab2FFF Ab4FFF AcFFF AgFFF`

Espansione

Parameter Expansion

- **`${parameter:-word}`**

  Se **`$parameter`** è non assegnato o nullo ritorna l'espansione di **`word`**
- **`${parameter:offset:length}`**

  Ritorna **`lenght`** caratteri di **`parameter`** a partire da **`offset`**
- **`${parameter#word}`**

  `${parameter##word}`

  Toglie il più piccolo (**`#`**) o più grande (**`##`**) *match* di **`word`** all'inizio di **`parameter`**
- **`${parameter%word}`**

  `${parameter%%word}`

  Toglie il più piccolo (`%`) o più grande (`%%`) *match* di **`word`** alla fine di **`parameter`**
- **`${parameter/pattern/string}`**

  `${parameter//pattern/string}`

  Il primo (`/`) o ogni (`//`) *match* di **`pattern`** è sostituito con **`string`**

Espansione

Command Substitution

- **`$(command)`**

L'espansione è effettuata sostituendo *command* con il suo stdout, rimuovendo l caratteri di *newline*

Espansione

Pathname Expansion

`*` Matches any string, including the null string.

`?` Matches any single character.

`[...]` Matches any one of the enclosed characters.

`[adFG23]`

`[a-f]=[abcdef]`

`[[:class:]]` dove **`class`** è **`alnum alpha ascii blank cntrl digit graph lower print punct space upper word xdigit`**

`[[:alpha:]]=[a-zA-Z]`

Alcuni comandi elementari

cp - copy files and directories

```
cp [OPTION]... [-T] SOURCE DEST
cp [OPTION]... SOURCE... DIRECTORY
-f, -i, -r
```

mv - move (rename) files

```
mv [OPTION]... [-T] SOURCE DEST
mv [OPTION]... SOURCE... DIRECTORY
-f, -i
```

ls - list directory contents

```
ls [OPTION]... [FILE]...
-a, -d, -h, -l
```

mkdir - make directories

```
mkdir [OPTION] DIRECTORY...
-p
```

pwd - print name of current/working directory

```
pwd [OPTION]
```

Script vs linea di comando

Script:

file (*genera_file_Data2.sh*)

```
#!/bin/bash
for i in Z X Y W; do
    for s in $(seq -w 0 99); do
            R=$RANDOM
            for j in $(seq 1 100); do
                    echo -ne "$(echo $RANDOM | tr "1234567890" \
                    "AKRFPWMSO")\t" >> "$i"_"$s"_$R.dat;
            echo $RANDOM  >> "$i"_"$s"_$R.dat
            done
    done
done
```

- chmod +x *genera_file_Data2.sh*
- *./genera_file_Data2.sh*

Linea di comando:

```
oliva@oliva-laptop$ for i in Z X Y W; do for s in $(seq -w 0 99); do
R=$RANDOM; for j in $(seq 1 100); do echo -ne "$(echo $RANDOM | tr
"1234567890" "AKRFPWMSO")\t" >> "$i"_"$s"_$R.dat; echo $RANDOM  >>
"$i"_"$s"_$R.dat;done;done;done
```

Shell Builtin Commands

`: [arguments]`

No effect; the command does nothing beyond expanding arguments and performing any specified redirections. A zero exit code is returned.

`echo [-neE] [arg ...]`

Output the args, separated by spaces, followed by a newline. The return status is always 0. If **-n** is specified, the trailing newline is suppressed. If the **-e** option is given, interpretation of backslash-escaped characters is enabled.

`printf format [arguments]`

Write the formatted arguments to the standard output under the control of the format.

Alcuni comandi utili

`cat` - concatenate files and print on the standard output

`cat [OPTION] [FILE]...`

`-n` number all output lines

`less` - opposite of more

Less is a program similar to *more*, but which allows backward movement in the file as well as forward movement.

`head, tail` - output the first (last) part of files

-N print the first (last) N lines instead of the first (last) 10

Alcuni comandi utili

`bc` - An arbitrary precision calculator language

`bc [ -hlwsqv ] [long-options] [ file ... ]`

`-l` Define the standard math library.

Loop

```
for arg in [list]
do
  command(s)...
done
```

- `for planet in Mercury Venus Earth ; do echo $planet ; done`
- `for i in $(seq 1 100); do echo $i; done`

Data 1

Spostare tutti i file che iniziano con A o C in una directory tmp1

Raddoppiare il valore di tutte le entry dei file C tra 90 e 99 e metterli in file con nome analogo ma estensione .DAT in tmp2

Triplicare il valore di tutte le entry dei file D che contengono il 7 o il 3

Data 1

Spostare tutti i file che iniziano con A o C in una directory tmp1

```
mv {A,C}*.dat tmp1/
```

Raddoppiare il valore di tutte le entry dei file C tra 90 e 99 e metterli in file con nome analogo ma estensione .DAT in tmp2

```
for i in C_9?_*; do for j in $( cat $i); do \
echo "$j * 2" |bc -l >> tmp2/${i%.dat}.DAT ; done; done
```

Triplicare il valore di tutte le entry dei file D che contengono il 7 o il 3

(esercizio)

Alcuni comandi utili

`grep` - print lines matching a pattern
grep [OPTIONS] PATTERN [FILE...]

-i Ignore case distinctions in both the PATTERN and the input files.

-v Invert the sense of matching, to select non-matching lines.

-c Suppress normal output; instead print a count of matching lines for each input file. With the -v, match option, count non- matching lines.

-l Suppress normal output; instead print the name of each input file from which output would normally have been printed. The scanning will stop on the first match.

-n Prefix each line of output with the 1-based line number within its input file.

-o Print only the matched (non-empty) parts of a matching line, with each such part on a separate output line.

-H Print the file name for each match. This is the default when there is more than one file to search.

-A NUM Print NUM lines of trailing context after matching lines.

-B NUM Print NUM lines of leading context before matching lines.

-a Process a binary file as if it were text.

-R, -r Read all files under each directory, recursively.

-E Interpret PATTERN as an extended regular expression (ERE).

Alcuni comandi utili

grep – **regular expressions** (see also *man regex*)

A regular expression is a pattern that describes a set of strings. Regular expressions are constructed analogously to arithmetic expressions, by using various operators to combine smaller expressions.

bracket expression: list of characters enclosed by [and]. It matches any single character in that list; if the first character of the list is the caret ^ then it matches any character not in the list.

The period . matches any single character.

The caret ^ and the dollar sign $ are meta-characters that respectively match the empty string at the beginning and end of a line.

The symbols \< and \> respectively match the empty string at the beginning and end of a word.

A regular expression may be followed by one of several repetition operators:

- ? The preceding item is optional and matched at most once.
- * The preceding item will be matched zero or more times.
- + The preceding item will be matched one or more times.
- {n} The preceding item is matched exactly n times.
- {n,} The preceding item is matched n or more times.
- {,m} The preceding item is matched at most m times.
- {n,m} The preceding item is matched at least n times, but not more than m times.

Uso di grep

- 8ofur10.txt: L’Orlando Furioso (L. Ariosto) Etext by The Project Gutenberg
- Diamo un’occhiata al file

Uso di grep

- trovare quante volte è nominata Angelica
- trovare quante volte è nominata Angelica (case insensitive)
- cosa sono le differenze?
- trovare dove è nominata Angelica
- trovare quante volte è nominato Orlando a meno di 5 righe da Angelica
- trovare tutte le volte in cui Orlando è a inizio riga
- trovare tutte le volte in cui Orlando NON è a inizio riga
- quanti sono i nomi propri?

Uso di grep

- trovare quante volte è nominata Angelica
  - `grep  -c Angelica 8ofur10.txt`
  - `grep   Angelica 8ofur10.txt | wc -l`
- trovare quante volte è nominata Angelica (case insensitive)
  - `grep  -ci Angelica 8ofur10.txt`
- cosa sono le differenze?
  - `grep -i Angelica 8ofur10.txt | grep -v Angelica`
- trovare in quali righe è nominata Angelica
  - `grep  -n Angelica 8ofur10.txt`
- trovare quante volte è nominato Orlando a meno di 5 righe da Angelica
  - `grep -A5 -B5 Angelica 8ofur10.txt| grep Orlando |wc -l`
- trovare tutte le volte in cui Orlando è a inizio riga
  - `grep "^Orlando" 8ofur10.txt | wc -l`
- trovare tutte le volte in cui Orlando NON è a inizio riga
  - `grep –c "[^^]Orlando" 8ofur10.txt`
  - `grep –c "Orlando" 8ofur10.txt` (tutti)
- quanti sono i nomi propri?
  - `grep -o "\<[[:upper:]][[:alpha:]]\+" 8ofur10.txt | sort | uniq`
  - `tail -48609 8ofur10.txt | grep -o "[^^]\<[[:upper:]][[:lower:]]\+" \ | tr -d "' " | sort | uniq`

    Esclude le maiuscole a inizio riga (perde qualcosa?) ma NON le maiuscole a inizio di discorso diretto (esercizio)

Alcuni comandi utili

`wc` - print newline, word, and byte counts for each file

wc [OPTION]... [FILE]...

-c print the byte counts
-l print the newline counts
-w print the word counts

`cut` - remove sections from each line of files

cut OPTION... [FILE]...

-dDELIM use DELIM instead of TAB for field delimiter
-fLIST select only these fields;

Each LIST is made up of one range, or many ranges separated by commas.
Each range is one of:

- N N'th byte, character or field, counted from 1
- N- from N'th byte, character or field, to end of line
- N-M from N'th to M'th (included) byte, character or field
- -M from first to M'th (included) byte, character or field

Alcuni comandi utili

`tr` - translate or delete characters

tr [OPTION]... SET1 [SET2]

SETs are specified as strings of characters.

-d delete characters in SET1, do not translate

-s replace each input sequence of a repeated character that is listed in SET1 with a single occurrence of that character

Alcuni comandi utili

`sed` - stream editor for filtering and transforming text

sed [OPTION]... {script-only-if-no-other-script} [input-file]...

`s/regexp/replacement/[g]`

Attempt to match regexp against the pattern space. If successful, replace that portion matched with replacement (without g replace only the first match, whit g replace all)

`y/source/dest/`

Transliterate the characters in the pattern space which appear in source to the corresponding character in dest.

Addresses

Sed commands can be given with no addresses, in which case the command will be executed for all input lines; with one address, in which case the command will only be executed for input lines which match that address. After the address (or address-range), and before the command, a **`!`** may be inserted, which specifies that the command shall only be executed if the address (or address-range) does not match.

`/regexp/`

Match lines matching the regular expression regexp.

Data2

Tutti i file Z che contengono la entry OO

Quante sono le entry che contengono OO (nei file Z)?

Sort dei valori (alfabetico e numerico) (nei file Z)

Sostituire nell'output tutte le entry che contengono OO con oo, solo se c'è un numero dispari associato (nei file Z)

Mostrare per Z, X, Y e W quanti file contengono AA

Cancellare tutti i file Z che contengono AA

Data2

Tutti i file Z che contengono la entry OO

```
grep -l OO Z*
```

Quante sono le entry che contengono OO (nei file Z)?

```
grep  OO Z*| wc -l
```

Sort dei valori (alfabetico e numerico) (nei file Z)

```
grep  -h OO Z*| sort
grep  -h OO Z*| sort -n -k2
```

Sostituire nell'output tutte le entry che contengono OO con oo, solo se c'è un numero dispari associato (nei file Z)

```
grep  -h OO Z* | sed "/[13579]/s/OO/oo/"
```

Mostrare per Z, X, Y e W quanti file contengono AA

```
for i in X Y Z W; do grep AA $i* | wc -l; done
```

Cancellare tutti i file Z che contengono AA
(esercizio)

Alcuni comandi utili

`pr` - convert text files for printing

pr [OPTION]... [FILE]...

- -m print all files in parallel, one in each column,
- -T omit page headers and trailers, eliminate any pagination by form feeds set in input files

Data3

Mo_12deg_28kV_1Be03Mo650Air.dat
Spettro tubo RX, bin di step 0.5 keV fino ad Emax

Quanti bin?

Aggiungiamo l'energia e mettiamo il tutto il fileout.dat

Energia dal file energy.dat

Bin più intensi

Data3

Mo_12deg_28kV_1Be03Mo650Air.dat
Spettro tubo RX, bin di step 0.5 keV fino ad Emax

Quanti bin?

```
cat –n Mo_12deg_28kV_1Be03Mo650Air.dat
wc –l Mo_12deg_28kV_1Be03Mo650Air.dat
```

Aggiungiamo l'energia e mettiamo il tutto il fileout.dat

```
cat -n Mo_12deg_28kV_1Be03Mo650Air.dat| while read e F;  \
   do E=$(echo "$e /2" | bc -l); \
       echo -e "$E\t$F‟ >> fileout.dat; done
```

Energia dal file energy.dat

```
pr -mT energy.dat Mo_12deg_28kV_1Be03Mo650Air.dat| tr -s " \t"
```

Bin più intensi

```
cat fileout.dat | sort –n –k2
```

Data4

Lunghezza del file

Lunghezza dell’header

Estrazione dell’immagine raw

Data4

Lunghezza del file

```
ls -l FR_24k_6_7s_0_Z0000.edf
ls -lh FR_24k_6_7s_0_Z0000.edf
wc -c FR_24k_6_7s_0_Z0000.edf
```

Lunghezza dell'header

```
head -$(cat -n FR_24k_6_7s_0_Z0000.edf | grep -a "}\$"  \
       | head -1| cut -f1| tr -d " ") FR_24k_6_7s_0_Z0000.edf
```

Estrazione dell'immagine raw

Vedi script **`Data4_extract_raw.sh`**

Script

Tutti gli script descritti e quelli usati perr generare Data1, Data2 e Data3

tv_stasera.sh

Prende da televideo I programmi di prima e seconda serata

wget

GRAN PREMIO

TROVARE NELL'ORLANDO FURIOSO IL RIFERIMENTO (**CANTO, STROFA, NUMERI DI LINEA**) ALLA MORIA DELLA MANDRIA IN SEGUITO AL FULMINE