Laboratorio Linux/FOSS

Alberto Cammozzo
anno accademico 2005-2006

Parte Terza

GNU/Linux - 2

indice

filtri: adesso facciamo sul serio

- tr – translate

- *regular expressions* - espressioni regolari

- sed – *stream editor*
- re – *regular expressions*
- awk – *Aho, Weinberger, Kernighan*

tr

traduce i caratteri dell'input da un insieme a un'altro insieme

```
tr set1 set2
```

`tr [:upper:] [:lower:]`	da maiuscole a minuscole
`tr [:blank:] [:space:]`	spaziature in spazi
`tr [:punct:] .`	tutta la punteggiatura in “.”

`tr -s "[:space:][:punct:]" .`

sed

- *stream editor*
- usato soprattutto per *cerca e sostituisci*

  ```
  sed s/,/./g
  ```

  ```
  sed s@/home/mmzz@/home/zzmm@
  ```
- cancella (agisce in modo simile a grep -v)

  ```
  sed -n 1,3d
  ```
- stampa (agisce in modo simile a grep)

  ```
  sed -n 1p
  ```

espressioni regolari

- servono a *catturare* regolarità nelle stringhe di testo
- vengono usate da molti programmi, non solo unix/linux: grep, vi, sed, ...
- sono tremendamente utili
- diventano rapidamente complicate e illeggibili

  nuova patologia → “regexp fatigue”

espressioni regolari

letterali

```
/a/
Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.
```

```
/Mary/
Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.
```

re: classi di caratteri

```
/[Mma]/
Mary had a little lamb.

/./
Mary had a little lamb.

/\s[al]/
Mary had a little lamb.

/\s./
Mary had a little lamb.
```

`[abc]`	qualsiasi carattere abc
`[^abc]`	qualsiasi carattere NON abc
`.`	qualsiasi carattere eccetto fine riga
`\s`	qualsiasi carattere di spaziatura
`\t`	tab
`\n`	fine riga (NEWLINE)

espressioni regolari

```
/^Ma/
Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.
```

```
/Mary$/
Mary had a little lamb.
And everywhere that Mary
went, the lamb was sure
to go.
```

posizionali:

^ inizio riga

$ fine riga

sed con regexp

sed `/`regexp`/s/`regexp`/`rimpiazzo`/`flags

Indirizzo: regexp che *attiva* la sostituzione

regexp che identifica la stringa da cercare

stringa da sostituire alla regexp.

`&` indica la stringa identificata dalla regexp

flags

g: global, rimpiazza oltre la prima riga

i: case insensitive

w *file*: scrivi il pattern space in *file*

n: rimpiazza la *n*-esima istanza nella riga

sed + re

```
$ sed "s/m/***/1i" < mary

***ary had a little lamb.
And everywhere that ***ary
went, the la***b was sure
to go.
```

sed + re (raffreddore)

```
$ sed "s/[mlv]/b/ig" < mary
```

```
bary had a bittbe babb.
And eberywhere that bary
went, the babb was sure
to go.
```

sed + re

```
$ sed  "/Mary/s/./ ./g"  mary
.......................
........................
went, the lamb was sure
to go.
```

sed

```
$ sed -e "s/esa/=&=/g" teresa

La vispa Ter=esa= avea tra lerbetta
A volo sorpr=esa= gentil farfalletta
E tutta giuliva stringendola viva
gridava dist=esa=: "L'ho pr=esa=! L'ho pr=esa=!"
```

sed [-n] *indirizzo comando*

indirizzi

`n`	n-esima riga
`n~m`	n-esima riga modulo m
`$`	fine file
`/re/`	espr. regolare
`/re/I`	*case insensitive*

comandi

`s/``A`**`/`**`B`**`/`***`fl`*** sostituisci A con B

`fl`: **`I`**: case insensitive

`g`: global

n: rimpiazza solo n-esimo

`d`	cancella *pattern space*
`p`	stampa *pattern space*
`{ }`	gruppo di comandi
`;`	concatenazione

awk

- linguaggio di programmazione
  - variabili, assegnazioni, confronti, cicli, ...
  - operazioni aritmetiche
  - funzioni predefinite
- predisposto per dati in forma di tabella
  - sa cos'è un *record* e un *campo*
  - formati numerici interi, float, ...
  - può *formattare* i dati in modo complesso
- sa cosa sono le *regular expressions*

awk: esempio semplice

elencare i file eseguibili nella mia home

```
ls -laR . |  awk '/^-........x/ { print $3,$5,$8 }'
```

```
./asid:
total 9196
-rwxr-xr-x  1 mmzz users    5580 2000-08-21 10:56 asidadduser.pl
-rwxr-xr-x  1 mmzz users     864 2000-08-21 10:56 asidexpfile.pl
-rw-r--r--  1 mmzz users 9379840 2000-03-07 13:22 Sopravvivere.tar

./bin:
total 32
-rwxr-xr-x  1 mmzz users  8148 2000-10-13 12:35 shus
```

```
mmzz 5580 asidadduser.pl
mmzz 9379840 asidexpfile.pl
mmzz 5580 shus
```

awk: invocazione

```
awk [ -F<ch> ] {pgm} | { -f <pgm_file> } [ <vars> ] [ - | <data_file> ]
```

`ch`	separatore di campo
`pgm`	programma
`pgm_file`	file con il programma
`vars`	inizializzazioni variabili di awk
`data_file`	file dati in input

Formato del programma:

```
BEGIN {<initializations>}
<RE>  {<program actions>}
<RE>  {<program actions>}
...
END   {<final actions>}
```

BEGIN: eseguite all'avvio

END: eseguite alla fine

awk: variabili predefinite

- variabili importanti:

  `FS` separatore di campo (space)

  `RS` separatore di record (newline)

- altre variabili

  `OFS` output field separator (blank)

  `ORS` output record separator (newline)

  `NR` numero di record

  `NF` numero di campi nel record corrente

awk: strutture

Strutture di controllo

```
if (condition) statement [ else statement ]
while (condition) statement
do statement while (condition)
for (expr1; expr2; expr3) statement
for (var in array) statement
break
continue
delete array[index]
delete array
exit [ expression ]
{ statements }
```

Funzioni

```
next
print
printf
system

match
split
length
sub
tolower
toupper

...
```

awk, esempio

```
# Operazioni sulle colonne:
# somma i valori della prima colonna,
# alla fine stampa la media

    { s += $1 }
END { print "somma", s, " media", s/NR }
```

`#`	Commento
`+=` /	Operazioni
`print`	stampa
`END`	azioni da compiere alla fine
`$1`	prima colonna

awk, esempio

```
# Operazioni sulle righe:
# sostituisci ogni valore in una riga (record)
# con il suo valore assoluto

{ for (i = 1; i <= NF; i=i+1) if ($i < 0) $i = -$i print}
```

awk, altri esempi

```
# Print first two fields in opposite order:
  awk '{ print $2, $1 }' file

# Print lines longer than 72 characters:
  awk 'length > 72' file

# Print length of string in 2nd column
  awk '{print length($2)}' file

# Print fields in reverse order:
  awk '{ for (i = NF; i > 0; --i) print $i }' file

# Print the last line
      {line = $0}
  END {print line}

# Print the total number of lines that contain the
# word Pat
  /Pat/ {nlines = nlines + 1}
  END {print nlines}

# Print all lines between start/stop pairs:
  awk '/start/, /stop/' file

# Print all lines whose first field is different
# from previous one:
  awk '$1 != prev { print; prev = $1 }' file
```

```
# Print column 3 if column 1 > column 2:
  awk '$1 > $2 {print $3}' file

# Print line if column 3 > column 2:
  awk '$3 > $2' file

# Count number of lines where col 3 > col 1
  awk '$3 > $1 {print i + "1"; i++}' file

# Print sequence number and then column 1 of file:
  awk '{print NR, $1}' file

# Print every line after erasing the 2nd field
  awk '{$2 = ""; print}' file
```

```
# Print hi 28 times
  yes | head -28 | awk '{ print "hi" }'

# Print hi.0010 to hi.0099 (NOTE IRAF USERS!)
  yes | head -90 | awk '{printf("hi00%2.0f \n", NR+9)}'

# Print out 4 random numbers between 0 and 1
yes | head -4 | awk '{print rand()}'

# Print out 40 random integers modulo 5
yes | head -40 | awk '{print int(100*rand()) % 5}'
```

Patrick Hartigan
hartigan@sparky.rice.edu
http://sparky.rice.edu/~hartigan/awk.html

oltre sed, awk

"C'è più di un modo per farlo"

- altri linguaggi:
  - perl, python, C, PHP, emacs lisp, tcl-tk, expect, ...
- strumenti di supporto alla programmazione
  - make, m4, emacs, yacc, lex
- strumenti di produzione di documentazione
  - nroff/troff, TeX, LaTeX, TeXinfo, Docbook

The Art of Unix Programming

Eric Steven Raymond
Thyrsus Enterprises
<esr@thyrsus.com>

http://www.catb.org/~esr/writings/taoup/

Rule of Modularity: Write simple parts connected by clean interfaces.
Rule of Clarity: Clarity is better than cleverness.
Rule of Composition: Design programs to be connected to other programs.
Rule of Separation: Separate policy from mechanism; separate interfaces from engines.
Rule of Simplicity: Design for simplicity; add complexity only where you must.
Rule of Parsimony: Write a big program only when it is clear by demonstration that nothing else will do.
Rule of Transparency: Design for visibility to make inspection and debugging easier.
Rule of Robustness: Robustness is the child of transparency and simplicity.
Rule of Representation: Fold knowledge into data so program logic can be stupid and robust.
Rule of Least Surprise: In interface design, always do the least surprising thing.
Rule of Silence: When a program has nothing surprising to say, it should say nothing.
Rule of Repair: When you must fail, fail noisily and as soon as possible.
Rule of Economy: Programmer time is expensive; conserve it in preference to machine time.
Rule of Generation: Avoid hand-hacking; write programs to write programs when you can.
Rule of Optimization: Prototype before polishing. Get it working before you optimize it.
Rule of Diversity: Distrust all claims for "one true way".
Rule of Extensibility: Design for the future, because it will be here sooner than you think.