

Sed & Awk

Sed e Awk

sed: editor non interattivo di file di testo (1974 nei Bell Labs come evoluzione di grep, Lee E. McMahon)

awk: linguaggio per l'elaborazione di modelli orientato ai campi (1977, Bell Labs Aho, Weinberger, Keringhan)

Condividono una sintassi d'invocazione simile:

- fanno uso delle [Espressioni Regolari](#)
- leggono l'input, in modo predefinito, dallo stdin
- inviano i risultati allo stdout
- le loro capacità combinate danno agli script di shell parte della potenza di Perl

Stream EDitor (sed)

- sed: editor di linea che non richiede l'interazione con l'utente
- sed può filtrare l'input che riceve da un file o una pipe
- La sintassi di sed NON definisce un output:
 - L'output viene inviato allo standard output e può essere rediretto
- sed NON modifica l'input

sed

Sed significa stream editor. Consente di effettuare in modo non interattivo le seguenti operazioni:

- ▶ sostituzioni
- ▶ cancellare linee
- ▶ aggiungere linee
- ▶ rimpiazzare linee

Usando anche le funzionalità meno conosciute, ... anche giocare ad arkanoid.

<http://aurelio.net/bin/sed/arkanoid/arkanoid.sed>

Stream EDiTOr (sed)

SYNOPSIS

sed [-an] command [file ...]

[-an] [-e command] [-f command_file] [file ...]

- sed legge i file specificati, oppure lo standard input se non specificati file;
- modifica l'input come specificato da una lista di comandi;
- L'input è quindi scritto sullo standard output.

-n di default ogni linea e' replicata sullo standard output dopo l'applicazione dei comandi. Opzione -n elimina questo.

Comandi sed

Alcuni comandi:

- a\ “Append” di testo al di sotto della riga corrente
- c\ Modifica il testo della riga corrente
- d Cancella testo
- i\ Inserisci testo al di sopra della riga corrente
- p Stampa testo
- r Legge un file
- s Cerca e modifica testo

Comandi sed

- # comment
- q <code> exit returning <code>
- d delete pattern-space
- p print pattern-space
- {...} raggruppa comandi
- s/regex/repl/flag rimpiazza il pattern-space
- y/from/to/ traslittera da from a to
- a\ stampa il testo che segue alla fine del ciclo
- i\ stampa il testo che segue subito
- c\ cancella il pattern-space e lo rimpiazza con il testo che segue
- = stampa il numero di linea corrente
- r <file> legge file e lo stampa alla fine del ciclo
- w <file> salva in file il pattern-space

Comandi sed

- D** cancella fino a `\n` il pattern-space (e se non vuoto riparte)
- h** rimpiazza l'hold-space con il pattern-space
- H** aggiungi in coda il pattern-space all'hold-space
- g** inverso di **h**
- G** inverso di **H**
- x** scambia il pattern-space con l'hold-space
- :** `<label>` definisce una etichetta
- b** `<label>` salta all'etichetta
- t** `<label>` salta all'etichetta solo se almeno un comando **s** ha avuto successo

Comandi Sed

La forma del comando sed e' la seguente:

[address[,address]]function[arguments]

Sed è una macchina a registri:

1. copia ciclicamente una linea di input in un pattern space,
2. applica tutti i comandi con address selezionati dal pattern space,
3. copia il pattern space nell standard output, aggiungendo newline,
4. quindi cancella il pattern space.

Indirizzo: [address[,address]]

L'**indirizzo** non e' richiesto, ma se specificato deve essere:

1. un numero (che conta linee di input nei file di input),
 2. un carattere ``\$' per l'ultima line di input
 3. oppure un address di contesto (espressione regolare preceduta o seguita da un delimitatore).
- Una linea di comando senza **indirizzo** seleziona ogni pattern space.
 - Una linea di comando con un **indirizzo** seleziona ogni pattern space dato dall'address.
 - Una linea di comando con due **indirizzo** seleziona il range inclusivo del primo pattern space tra i due primo indirizzi.

Indirizzo

Un indirizzo può essere

`n` n-esima riga

`$` ultima riga

`/regexp/` espressione regolare fa match

`n~m` a partire dalla riga n-esima, ogni m righe (GNU sed)

Un range è dato da una coppia di indirizzi `begin` e `cont` separati da `,` e inizia dalla prima linea che fa match con `begin` (inclusa) e si estende fino a che `cont` fa match. Estensione GNU `start,+n` per dire da `start` a n linee dopo.

Sed

- Se non si specificano azioni, sed stampa sullo standard output le linee in input, lasciandole inalterate
- Se non viene specificato un **indirizzo** o un intervallo di indirizzi di linea su cui eseguire l'azione, quest'ultima viene applicata a tutte le linee in input.
- Gli indirizzi di linea si possono specificare come **numeri** o **espressioni regolari**.
- Se vi è più di un'azione (**comandi multipli**), esse possono essere specificate sulla riga di comando precedendo ognuna con l'opzione -e, oppure possono essere lette da un file esterno specificato sulla linea di comando con l'opzione -f.

sed: Esempi

lso:~> sed 'd' /etc/services

Non visualizza nulla, ma cancella linea per linea il contenuto del file

lso:~> sed '1d' /etc/services | more

Cancella la prima riga il resto in stdio

lso:~> sed '1,10d' /etc/services | more

Righe tra 1 e 10 in stdio

lso:~> sed '/^#/d' /etc/services | more

Espressione regolare

Esempio

Iso:~>cat esempio

- 1 Questo e' un esempio.
- 2 Questa riga contiene un errore
- 3 Un altro errore in questa riga
- 4 Questa riga e' corretta
- 5 Questa riga contiene un altro errore.

Iso:~>grep errore esempio

- 2 Questa riga contiene un errore
- 3 Un altro errore in questa riga
- 5 Questa riga contiene un altro errore.

Comando stampa

Iso:~>sed '/errore/p' esempio

- 1 Questo e' un esempio.
- 2 Questa riga contiene un errore
- 2 Questa riga contiene un errore
- 3 Un altro errore in questa riga
- 3 Un altro errore in questa riga
- 4 Questa riga e' corretta
- 5 Questa riga contiene un altro errore.
- 5 Questa riga contiene un altro errore.

sed '/espressione/p' file

Stampa tutte le linee, quelle che contengono la stringa si ripetono

Iso:~>sed -n '/errore/p' esempio

- 2 Questa riga contiene un errore
- 3 Un altro errore in questa riga
- 5 Questa riga contiene un altro errore.

Per stampare solo le linee che contengono la stringa si usa l'opzione -n

Comando Stampa

```
lso:~> sed -n -e '/BEGIN/,/END/p' /my/test/file | more
```

```
lso:~> sed -n -e '/main[:space:]]*(/,/^}/p' sourcefile.c
```


Comando Cancelli

Iso:~>sed '/erore/d' esempio sed '/espressione/d' file

1 Questo e' un esempio.

4 Questa riga e' corretta

Il comando **d** porta
ad escludere linee
dalla visualizzazione

Iso:~>sed -n '/^Questo.*esempio.\$/d' example

2 Questa riga contiene un errore

3 Un altro errore in questa riga

4 Questa riga e' corretta

5 Questa riga contiene un altro errore.

Escluse le linee che
iniziano con una stringa
e terminano con un'altra

Comando Cancella

Iso:~> sed /^\$/d inputFileNames

Iso:~>sed '/^ *\$/d' inputFileNames

Iso:~> sed '1,/^\$/d' inputFileNames

Iso:~> sed 8d inputFileNames

Selezione di un range di righe

`sed '[add1,[add2]]com'`

Iso:~>sed '2,4d' esempio

(Cancella righe tra
2,4)

1 Questo e' un esempio.

5 Questa riga contiene un altro errore.

Iso:~>sed '3,\$d' esempio

(Cancella righe
tra 3 e ultima \$)

1 Questo e' un esempio.

2 Questa riga contiene un errore

Ricerca e Sostituzione

Iso:~>sed 's/erore/errore/g' esempio

- 1 Questo e' un esempio.
- 2 Questa riga contiene un **errore**
- 3 Un altro **errore** in questa riga
- 4 Questa riga e' corretta
- 5 Questa riga contiene un altro **errore**.

Iso:~>sed 's/^/> /g' esempio

- > 1 Questo e' un esempio.
- > 2 Questa riga contiene un errore
- > 3 Un altro errore in questa riga
- > 4 Questa riga e' corretta
- > 5 Questa riga contiene un altro errore

Sostituzione

's/{old value}/{new value}/'

Sostituzione globale

's/{old value}/{new value}/g'

Ricerca e Sostituzione

**Iso:~>sed -e 's/erore/errore/g'
-e 's/^/> /g' esempio**

Multiple Changes

- > 1 Questo e' un esempio.**
- > 2 Questa riga contiene un errore**
- > 3 Un altro errore in questa riga**
- > 4 Questa riga e' corretta**
- > 5 Questa riga contiene un altro errore.**

Ricerca e Sostituzione

Iso:~> sed 's/yourword//g' yourfile

Cancella tutte le parole che contengono yourword

Iso:~> sed -e 's/firstword//g' -e 's/secondword//g' yourfile

Cancella le parole firstword e secondword

Iso:~> sed 's/ *\$//' yourfile

Iso:~> sed 's/00*/0/g' yourfile

Iso:~> sed 's/^/ /' file > file.indent

Ricerca e sostituzione

sed '/^\$/,/^END/s/hills/mountains/g' myfile3.txt

Sostituzione da riga nulla a riga che inizia con END

sed -e 's:/usr/local:/usr:g' mylist.txt

Separatore : al posto di /

sed -e 's/<.*>//g' myfile.html

Cancella la parola più lunga che fa il match:

This is what I meant.

sed -e 's/<[^>]*>//g' myfile.html

Ricerca e Sostituzione

sed -e 's/.*/lui dice: &/' origmsg.txt

Tutte le righe precedute da “lui dice:”. & ripete
l'ultimo match

Ricerca e Sostituzione

foo bar oni
aaa bbb ccc
bla curly bho

Sostituire “eny me min” con “Sig aaa-bbb Da ccc

Serve un'espressione per 3 stringhe separate `'.* .* .'`

Si usa `'\(.*\) \(.*\) \(.*)'`

`sed -e 's/\(.*\) \(.*\) \(.*)/Sig \1-\2 Da \3/' myfile.txt`

Centrare righe di un file

```
#!/usr/bin/sed -f
# Put 80 spaces in the buffer
1 { x
    s/^$/ /
    s/^.*$/&&&&&&&&&/
    x }
# del leading and trailing spaces
s/^[[[:blank:]]]*//
s/[[[:blank:]]]*$///
# add a newline and 80 spaces to end of line
G
# keep first 81 chars (80 + a newline)
s/^\(.\{81\}\).*$/\1/
# \2 matches half of the spaces
s/^\(.*\)\\n\(.*\)\\2/\\2\\1/
```

AWK - Aho, Kernighan and Weinberger

AWK è un linguaggio di scripting

- ▶ awk è un linguaggio standard POSIX
- ▶ gawk è una sua implementazione ben documentata
- ▶ strumento ideato per processare file di testo strutturati in “record” (definibili dall'utente), farne report.
- ▶ molto usato per scriptini “one liner”
- ▶ sintassi simil-C
- ▶ Riferimenti:
<http://www.gnu.org/software/gawk/manual/>

Elementi di Awk

- Funzione awk cerca su file linee o altre unità di testo che contengono pattern;
- Quando una linea corrisponde ad un pattern, azioni speciali vengono eseguiti sulla linea.
- In awk i programmi sono "data-driven": descrivi cosa cerchi, poi esegui;
- Il programma e' definito da un insieme di regole;
- Ogni regola e': azione da fare trovato il pattern.

Elementi di awk

- awk suddivide ogni linea in una sequenza di campi delimitati da separatori
 - *I separatori di default sono uno (o piu') spazi o caratteri di tabulazione.*
- Le variabili $\$1$, $\$2$, ..., $\$n$ identificano i primi n campi riconosciuti da awk
- La variabile $\$0$ contiene la riga in esame
- Sintassi:
 - ***awk '<programma>' <input-file1> <input-file2>...***
 - ***awk -f <file-script> <input-file> <input-file2>...***

Struttura di un programma awk

Un programma awk è costituito da una sequenza di regole

```
pattern { action }
```

Dove pattern è uno tra

BEGIN prima di processare l'input

END dopo aver processato l'input

boolexpr fa match se è vera (es. `$1 == "ciao"`)

/regexp/ fa match se la regex fa match (es. `/^July/`)

bpat,epat dal record che fa match con bpat a quello che fa match con epat

Default action (`print $0`) e default pattern (`true`):

```
awk 'length($0) > 80' data
```

```
awk '{ print $2 }' data
```

Esempio

Iso:~>df display free disk space

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/hda5	16682168	9932136	5902608	63%	/
none	256840	0	256840	0%	/dev/shm

Iso:~>df | awk '{ print \$1,\$2, \$3}'

Filesystem	1K-blocks	Used
/dev/hda5	16682168	9932556
none	256840	0

Le variabili \$1, \$2, \$3, ..., \$N contengono il primo, secondo, terzo ... ultimo campo di una linea di input. La variabile \$0 (zero) contiene la linea intera.

Selezione di non-matching lines

Iso:~>df | grep -v Filesystem | awk '{ print "La partizione :"\$1 "\t e' usata al "\$5}'

La partizione :/dev/hda5 e' usata al 63%

La partizione :none e' usata al 0%

awk ed Espressioni Regolari

- E' possibile combinare espressioni regolari e programmi awk utilizzando la seguente sintassi:

– awk '<espressione>{<programma>}' <file>

- Esempio:

Iso:~>df | awk '/devVhd/ { print "La partizione :"\$1 "\t e usata al "\$5}'

La partizione :/dev/hda5 e usata al 63%

- Il programma viene eseguito solo sulle righe che corrispondono al pattern dell'espressione regolare

awk: BEGIN ed END

Gli statement BEGIN ed END consentono eseguire operazioni prima e dopo il corpo del comando

```
Iso:~>df | awk 'BEGIN {print "Elenco partizioni"} /devVhd/ { print  
"La partizione :"$1 "\t e' usata al "$5} END {print "Fine  
Report\n"}'
```

Elenco partizioni

La partizione :/dev/hda5 e' usata al 63%

Fine Report

Esempi awk

Calcolo della riga più lunga di un file

```
awk '{ if (length($0) > max) max = length($0) }  
  END { print max }' data
```

Implementazione di du -sh

```
ls -l | awk '{ x += $5 }  
  END { print "total K-bytes: " (x + 1023)/1024 }'
```

Conta i processi appartenenti all'utente tassi

```
ps aux | awk '/^tassi/ { tot++ }  
  END { print "total processes: " tot }'
```

Implementazione di wc

```
awk '{ tot += NF } END { print tot }' data
```

Stampa delle potenze del 2 fino a 9

```
awk 'BEGIN { for (i=1; i<10; i++) print (2**i) }'
```

awk: Operatori e Predicati

Operatori aritmetici (sia per interi che floating point)

- + somma
- sottrazione
- * prodotto
- ** esponente
- / divisione
- % resto

Gli operatori sono anche disponibili in versione `<op>=` (es. `x **= 2`). Gli operatori (stile C) di pre/post incremento sono disponibili (es. `x++`).

I seguenti predicati sono disponibili sia per numeri che stringhe

- `<` `<=`
- `>` `>=` ordinamento (es. `3<2`, `"ciao" <= "delta"`)
- `==` uguaglianza (es. `3.4 == "3.4"`)
- `!=` diversità (es. `"x" != "ciao"`)
- `~` match con regex (es. `$3 ~ /foo/`)
- `!~` match negato (es. `$2 !~ /ugly/`)
- `in` chiave in array (es.
`if (2 in vect) print vect[2]`)

Operatori booleani

- `&&` congiunzione
- `||` disgiunzione
- `!` negazione

awk scripts

- E' possibile definire script awk
 - L'esempio precedente

```
Iso:~>df | awk 'BEGIN {print "Elenco partizioni"} /devVhd/ { print "La  
partizione :"$1 "\t e usata al "$5} END {print "Fine Report\n"}
```

- Diventa:

```
BEGIN {print "Elenco partizioni"}  
/devVhd/ { print "La partizione :"$1 "\t e usata al "$5}  
END {print "Fine Report\n"}
```

- Se il nome dello script e' report.awk

```
Iso:~>df |awk -f report.awk
```

Elenco partizioni

La partizione :/dev/hda5 e usata al 63%

Fine Report

awk: le variabili

awk usa molte variabili, alcune editabili, altre read-only.

- **La variabile FS** (Field Separator) identifica il separatore di input
 - Default spazi o tab
- **La variabile OFS** (Output Field Separator) identifica il separatore di output
 - Default spazio
- **La variabile ORS** (Output Record Separator) identifica il separatore di “record”
 - Default \n
- E' possibile modificare il valore di queste variabili
 - **BEGIN { FS=";" ; OFS="---"; ORS=">\n<-" }**

awk: le variabili

- La variabile **NR** contiene il numero di record processati
 - Viene incrementata automaticamente
- Ogni riferimento ad una variabile non definita comporta la creazione della stessa e la sua inizializzazione a ""
 - I riferimenti successivi utilizzeranno il valore corrente della stessa

awk: le variabili

FS Separatore di campi (anche con `-F` da command line). es. `BEGIN { FS = "(:|,)" }`

RS Separatore di record (default `\n`). es. `\n\n+`

NF Il numero di campi nel record corrente. es.
`awk 'NF > 2' data`

\$n Campo n-esimo del record corrente. `$0` è l'intero record. es.

`awk -F ':' '{ print $1, $3, $NF }' data`

OFS Separatore di campi in output, cambia il risultato di `print $0`. es. `BEGIN { OFS = ":" }`

ORS Separatore di record in output, default `\n`. es.
`\n---\n`

Nota: `$n` può essere modificato così come `NF`, modificando di conseguenza il valore di `$0`

Esempio

Iso:~>cat somma.awk

```
BEGIN {  
    FS=":";  
    print "Calcolo Subtotali e Totale"  
}  
{  
    subtotale=$1*$2;  
    print "Subtotale per " $3 "="subtotale;  
    totale = totale+subtotale;  
}  
END {  
    print "Totale ="totale  
}
```

Iso:~>cat dati.txt

```
100:2:Cliente 1  
200:8:Cliente 2  
500:2:Cliente 3
```


Esempio

Iso:~>cat dati.txt

100:2:Cliente 1

200:8:Cliente 2

500:2:Cliente 3

Iso:~>awk -f somma.awk dati.txt

Calcolo Subtotali e Totale introiti

Subtotale per Cliente 1=200

Subtotale per Cliente 2=1600

Subtotale per Cliente 3=1000

Totale =2800

Output formattato

- awk consente di formattare l'output utilizzando la funzione printf (invece della funzione print)
- Sintassi: **printf formato, item1, item2,...**

- Esempio:

```
Iso:~>awk 'BEGIN {w=5; p=3; s="abc"; printf "%d %4.3f %s\n",w,p,s}'
```

```
5 3.000 abc
```

- Il “formato” include %d, %f, %c, %o, %x...
- Nota: Lo statement BEGIN consente di eseguire programmi awk SENZA specificare un input (file o redirectione)

Redirezione in awk

- E' possibile utilizzare gli operatori di redirezione in script awk.
- **print items > nomefile**
 - **awk '{ print \$2 > "phone-list"; print \$1 > "name-list" }' nomefile**
- **print items >> nomefile**
- E' possibile utilizzare anche l'operatore "<"

Redirezione in awk

- Esecuzione di comandi

```
lso:~>awk '{
    print $1 > "names.unsorted";
    command = "sort -r > names.sorted";
    print $1 | command
}' file
```

awk: array

Gli array sono associativi (le chiavi possono essere anche stringhe) e anche multi-dimensionali. Esempi:
esistenza di una chiave

```
if (2 in array) print array[2]
```

assegnamento alla chiave 2 e "mario"

```
array[2] = "pippo"; array["mario"] = 30
```

visita di tutte le coppie chiave, valore

```
for (i in array) print i, array[i]
```

cancellazione di un elemento

```
delete array[2]
```

calcolo della dimensione

```
length(array)
```

Riferimenti

- Capitolo 4,5 e 6 di [Bash Guide for Beginners]
- Effective Awk Programming