

Espressioni regolari, sed e awk

Laboratorio Sistemi Operativi

Antonino Staiano
Email: antonino.staiano@uniparthenope.it

Introduzione

- Una *espressione regolare* (ER) è uno schema che descrive un insieme di stringhe
 - È una stringa di caratteri e metacaratteri
 - I metacaratteri sono caratteri speciali che vengono interpretati in modo “non letterale” dal meccanismo delle espressioni regolari
- Le espressioni regolari sono utilizzate per ricerche e manipolazioni di stringhe
 - Utilizzate da molti comandi quali grep, sed, awk, etc.
- **Match** (definizione):
 - Una ER fa un match con una particolare stringa se è possibile generare la stringa a partire dalla ER

Espressioni regolari

- La concatenazione è l'operazione alla base di ogni espressione regolare
 - **ABE**: ogni carattere letterale è un'espressione regolare che corrisponde solo a quel carattere
 - Questa espressione descrive una “**A** seguita da una **B** seguita poi da una **E**”, o semplicemente la stringa ABE
 - Il termine stringa significa ogni carattere concatenato a quello che lo precede
- Le espressioni regolari sono case-sensitive: “A” non fa match con “a”

Espressioni regolari

- Una ER non è limitata ai soli caratteri letterali
 - Esistono i **metacaratteri**
 - Es: il punto (.) può essere usato come *wild card* (qualsiasi carattere singolo eccetto il newline)
 - Es.: "A.E" fa il match con "ACE", "ABE", "ALE", ecc.
 - Il metacarattere asterisco, *, è usato per fare il match con zero o più occorrenze dell'espressione regolare che lo precede (tipicamente un carattere singolo)
 - Oss.: abbiamo familiarità con * come metacarattere della shell in cui il suo significato è "zero o più caratteri"
 - Il significato è differente nell'ambito delle espressioni regolari in cui di per sé l'* non fa il match con nulla
 - Piuttosto esso modifica ciò che lo precede
 - Es: l'espressione regolare .* fa il match con un qualsiasi numero di caratteri (ovvero con qualsiasi stringa), laddove nella shell è * che ha quel significato
 - L'espressione regolare "A.*E" fa il match con qualsiasi stringa che fa il match con "A.E" ma anche con un qualsiasi numero di caratteri tra A ed E: "AIRPLANE", "A FINE", "AFFABLE" o "A LONG WAY HOME"

{ 5 }

CdL in Informatica - Laboratori di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

Espressioni Regolari: ingredienti

- **insiemi di caratteri**: pattern elementari che specificano la presenza di un carattere appartenente ad un certo insieme
- **ancore**: legano il pattern a comparire in una posizione specifica della stringa (es. inizio, fine)
- **gruppi**: "racchiudono" l'espressione regolare che può quindi essere riferita come una singola entità
- **modificatori**: specificano ripetizioni dell'espressione che precede il modificatore stesso

{ 6 }

CdL in Informatica - Laboratori di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

Espressioni regolari: metacaratteri (I)

- L'asterisco * fa match con qualsiasi numero (incluso zero) di ripetizioni del carattere che lo precede
 - Es: "11*33" fa match con 133, 1133, 11133, 111133, ecc.
- Il punto . Fa match con qualsiasi carattere escluso il newline
 - Es.: "13.3" fa match con 13a3, 1303, ma non con 13\n3
- Il carattere ^ fa match con l'inizio di una linea (ha anche significati aggiuntivi)
 - Es: "^Oggetto:.*" fa match con una linea di "Oggetto:" di posta elettronica
- Il dollaro \$ fa match con la fine di una riga
 - Es.: "^\$" fa match con una linea vuota

{ 7 }

CdL in Informatica - Laboratori di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

Espressioni regolari metacaratteri(II)

- Le parentesi quadre [...] sono utilizzate per fare il match di un sottoinsieme (o di un intervallo) di caratteri
 - "[xyz]" fa match con i caratteri x, y, z
 - "[c-n]" fa match con qualsiasi carattere fra c ed n
 - "[a-zA-Z0-9]" fa match con qualsiasi carattere alfanumerico
 - "[0-9]*" fa match con qualsiasi stringa di numeri decimali
 - "[^0-9]" fa match con qualsiasi carattere non numerico
 - In questo caso ^ indica not
 - "[Yy][Ee][Ss]" fa match con yes, Yes, YES, yEs, ...
- Il backslash \ è usato come escape per i metacaratteri ed il carattere viene interpretato letteralmente: ad esempio, "\\$"
- I metacaratteri perdono il loro significato speciale nelle [...]

{ 8 }

CdL in Informatica - Laboratori di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

Espressioni regolari metacaratteri(III)

- Le espressioni regolari hanno un concetto di “parola”:
 - Una parola è un pattern contenente solo lettere, numeri e underscore `_`
- E' possibile fare matching
 - Con l'inizio di una parola: `\<`
 - Es.: `\<Fo` fa match con tutte le parole che iniziano con Fo
 - Con la fine di una parola: `\>`
 - Es.: `ox\>` fa match con tutte le parole che finiscono con ox
 - Con parole complete:
 - Es.: `\<Fox\>`
 - Esempio: come cercare la parola **The** o **the**?
 - `[tT]he` → errato: troverebbe anche other
 - `[]the[]` → ([] è lo spazio) errato: non troverebbe the all'inizio o fine di linea
 - `\<[tT]he\>` → OK: utilizza la sintassi `\<pattern\>`

Espressioni regolari estese

- Il carattere `?` fa match con 0 o 1 ripetizioni dell'ER precedente
 - Es.: `"cort?o"` fa match con `coro` e `corto`
- Il carattere `+` fa match con 1 o più ripetizioni della ER precedente, ma non 0
 - Es.: `"[0-9]+"` fa match con numeri non vuoti
- I caratteri `{ }` indicano il numero di ripetizioni della ER precedente
 - Es.: `"[0-9]{5}"` fa il match con tutti i numeri a 5 cifre
- I caratteri `()` servono a raggruppare espressioni regolari
 - Es.: `"(re)*"` fa match con `"", re, rere, rerere, ecc.`
- Il carattere `|` indica un'alternativa (or)
 - Es.: `"(bio|psico)logia"` fa match con `biologia` e `psicologia`
- Nelle ER di base, i metacaratteri `?`, `+`, `{ }`, `|`, `()` perdono il loro significato speciale
 - È necessario usare le versioni con backslash: `\?`, `\+`, `\{ }`, `\|`, `\()` e `\<`

Esempi ER di base ed estese

- `xy*z` un'occorrenza di 'x', seguita da zero o più 'y', seguita da una 'z'
- `xy{3,5}z` una 'x', seguita da 3 a 5 'y' e poi 'z' (**estesa**)
- `xy\{3,5\}z` una 'x', seguita da 3 a 5 'y' e poi 'z'
- `[A-Z][A-Z]*` una o più lettere maiuscole (stesso di `[A-Z]+`)
- `[A-Z]*` zero o più lettere maiuscole
- `a|the` la stringa 'a' o la stringa 'the'
- `cat` la stringa 'cat'
- `.at` qualsiasi char seguito da 'at', es. 'cat', 'rat', 'mat', 'bat', 'fat', 'hat'
- `^cat` 'cat' all'inizio della riga
- `cat$` 'cat' alla fine della riga
- `[cC]at` 'cat' o 'Cat'

Sommario ER (I)

Operatore

`.`
`?`

`*`
`+`
`\{N\}`
`\{N,\}`
`\{N,M\}`

`-`

`^`

`$`
`\<`
`\>`

Effetto

Corrisponde ad un qualsiasi carattere singolo
L'item precedente è opzionale ed il match avviene, al più, una volta

L'item precedente sarà eguagliato zero o più volte
L'item precedente sarà eguagliato una o più volte
L'item precedente sarà eguagliato esattamente N volte
L'item precedente sarà eguagliato N o più volte
L'item precedente sarà eguagliato almeno N volte, massimo M volte

Rappresenta un range se non è il primo o l'ultimo di una lista o il punto finale di un range in una lista
Corrisponde alla stringa vuota all'inizio di una riga; rappresenta anche i caratteri che non sono nel range di una lista
Corrisponde alla stringa vuota alla fine di una riga
Ancora di inizio di una parola
Ancora di fine di una parola

Sommario ER (II)

Pattern	Match
<code>^A</code>	Una A all'inizio di una riga
<code>A\$</code>	Una A alla fine di una riga
<code>A^</code>	Una A^ in qualsiasi punto di una riga
<code>\$A</code>	Un \$A in qualsiasi punto di una riga
<code>^\^</code>	Un ^ all'inizio di una riga
<code>^^</code>	Stesso di <code>^\^</code>
<code>\\$\$</code>	Un \$ alla fine di una riga
<code>\$\$</code>	Stesso di <code>\\$\$</code>
<code>^.\$</code>	Una riga con un qualsiasi carattere singolo

- E' da osservare che il carattere ^ è un'**ancora** solo se è il primo carattere in una ER, mentre \$ è un'**ancora** solo se è l'ultimo carattere in un'ER

[13]

Sommario ER(III)

Pattern	Match
<code>[0-9]</code>	Qualsiasi numero
<code>[^0-9]</code>	Qualsiasi carattere eccetto i numeri
<code>[-0-9]</code>	Qualsiasi numero o un -
<code>[0-9-]</code>	Qualsiasi numero o un -
<code>[^0-9-]</code>	Qualsiasi carattere eccetto i numeri o un -
<code>[]0-9]</code>	Qualsiasi numero o una]
<code>[0-9]</code>	Qualsiasi numero seguito da una]
<code>[0-9m-z]</code>	Qualsiasi numero o carattere tra m e z
<code>[]0-9-]</code>	Qualsiasi numero, un -, o una]
<code>[0-9]\$</code>	Qualsiasi numero che termina una riga

- E' da osservare che] e - non hanno un significato speciale se seguono o sono seguiti direttamente da una [

[14]

Sommario ER (IV)

Pattern	Match
<code>*</code>	Qualsiasi riga con un *
<code>*</code>	Qualsiasi riga con un *
<code>\\</code>	Qualsiasi riga con un \
<code>^*</code>	Qualsiasi riga che inizia con un *
<code>^A*</code>	Qualsiasi riga che inizia con A *
<code>^AA*</code>	Qualsiasi riga che inizia con una o più A
<code>^AA*B</code>	Qualsiasi riga che inizia con una o più A seguita da una B
<code>^A\{4,8\}B</code>	Qualsiasi riga che inizia con quattro, cinque, sei, sette o otto A seguite da una B
<code>^A\{4,\}B</code>	Qualsiasi riga con quattro o più A seguite da una B
<code>^A\{4\}B</code>	Qualsiasi riga che inizia con AAAAB

[15]

Classi di caratteri aggiuntive POSIX

- Lo standard POSIX formalizza il significato dei caratteri delle espressioni regolari e degli operatori
- Lo standard definisce due classi di espressioni regolari
 - Espressioni regolari di base, usate da grep e sed
 - Espressioni regolari estese, usate da egrep e awk
- Lo standard POSIX estende la rappresentatività delle classi di caratteri per fare il match con caratteri che non sono nell'alfabeto inglese
 - Esempio: la nostra "è" non sarebbe rappresentata con la tipica classe dei caratteri [a-z] (`[=e=]`)

[16]

Classi di caratteri POSIX

<code>[[:alnum:]]</code>	Caratteri stampabili
<code>[[:alpha:]]</code>	Caratteri alfabetici
<code>[[:blank:]]</code>	Caratteri di spazio e tab
<code>[[:digit:]]</code>	Caratteri numerici
<code>[[:lower:]]</code>	Caratteri alfabetici minuscoli
<code>[[:upper:]]</code>	Caratteri alfabetici maiuscoli

- **Esempio:**

- `[^[:alpha:]]` qualsiasi carattere non alfabetico (equivalente a `[^a-zA-Z]`)
- N.B.: le classi di caratteri (es., `[[:alpha:]]` oppure `[[:digit:]]`) si usano come fossero dei caratteri e quindi vanno indicate così: `[[:alpha:]]`, all'interno di altre parentesi quadre

(17)

Grep

(18)

Comando grep

- Il comando **grep** è utile per ricercare linee in un file che corrispondano ad un pattern specifico
- Il nome deriva da un command editor di UNIX chiamato **g/re/p = globally search for regular expression and print the matching lines**

(19)

Comando grep

- Esistono quattro versioni
 - `grep [options] pattern [file(s)]`
 - Pattern è un'espressione regolare
 - `fgrep [options] pattern [file(s)]`
 - Pattern è una stringa fissa
 - Versione più veloce
 - `egrep [options] pattern [files(s)]`
 - Pattern è un'espressione regolare estesa
 - `rgrep [options] pattern [file(s)]`
 - Cerca in ogni file in ogni directory, ricorsivamente

(20)

Comando grep

- Se non vi sono file specificati, la ricerca è nello standard input
- Il pattern è espresso come espressione regolare
- Tutte le linee che contengono il pattern vengono stampate su stdout
- In caso di file multipli, le linee sono precedute dal pathname del file che le contiene (a meno dell'opzione -h)
- Opzione -v
 - Fa in modo che l'output contenga tutte le linee che non contengono il pattern
- Opzione -c
 - Conta le occorrenze, invece di stamparle
- Opzione -i
 - Ignora distinzioni tra caratteri minuscoli e maiuscoli
- Opzione -q
 - Modo silente, non scrive nulla su stdout
- Opzione -n
 - Numera le righe del pattern cercato
- Opzione -w
 - Il pattern è considerato come una parola

[21]

Comando GREP: esempi

- Si consideri il file "elenco.txt" e si proceda alla ricerca di un pattern:

```
$grep "Scrivania" elenco.txt
```

- Scrivania da ufficio

```
$grep "Scri*" elenco.txt
```

- Scrivania da ufficio

```
$fgrep "Scri*" elenco.txt
```

- ---vuoto---

```
$grep -i "scrivania" elenco.txt
```

(ignora maiusc/minusc.)

```
$grep -v "Scrivania" elenco.txt (mostra quelle  
che non contengono il pattern)
```

```
Stampante di marca  
proprietaria  
Scrivania da ufficio  
Sedia da ufficio  
Materiale vario
```

[22]

Comando GREP: esempi

- E' possibile combinare più espressioni regolari assieme utilizzando il carattere di pipe "|":

```
$grep "Scrivania|Sedia" elenco.txt
```

- Scrivania da ufficio

- Sedia da ufficio

```
$grep -n "Scrivania" elenco.txt
```

- 2:Scrivania da ufficio

```
$grep -c "ufficio" elenco.txt
```

- 2

[23]

Comando grep

- Esempio: cercare tutti i file .html che contengono la parola "casa"

- **grep "casa" *.html**

- Grazie alla sintassi sopra espressa verrà cercata la stringa "casa" tra tutti i file .html (l'asterisco, infatti, sta ad indicare il metacarattere della shell ed è utilizzato, nel nostro esempio, per cercare ogni file con estensione .html)

- Da notare che nel nostro esempio abbiamo cercato non la parola "casa", ma la stringa! Ne consegue che nei risultati potremo trovare anche file che contengono, ad esempio, la parola "casale" o "casata"

- Se avessimo voluto cercare solo la parola "casa" avremmo dovuto usare l'opzione "-w" come di seguito:

- **grep -w "casa" *.html**

- Avremmo anche potuto scrivere **grep "\<casa\>" *.html**

[24]

Sed

Uno stream editor

- Sed è un editor orientato allo stream non interattivo
 - Orientato allo stream perché l'input fluisce attraverso il programma ed è diretto verso lo standard output
 - L'input su cui operare proviene da un file
 - L'output, di default, è diretto verso il video, ma può essere rediretto in un file
- sed esibisce funzionalità che sono estensione naturale del text editing interattivo
 - Offre, ad esempio, facility search-and-replace che possono essere applicate globalmente ad un file singolo o ad un gruppo di file
- Negli script di shell, **sed** è, di solito, uno delle molteplici componenti di una pipeline

Come funziona sed

- Sed elabora un file una riga alla volta ed invia l'output a video
- Sed memorizza la riga in un buffer temporaneo chiamato **pattern space**
 - Quando termina di elaborare una riga nel pattern space, la riga nel pattern space è stampata a video
 - Una volta elaborata la linea, viene rimossa dal pattern space e si procede con la riga successiva
 - Sed termina quando è elaborata l'ultima riga del pattern space
- Il file originale non è mai alterato o cancellato

sed

- Ci sono due modi per invocare sed
 - Specificando le istruzioni di editing da riga di comando
 - **sed -e '<edit commands>' <filename>**
 - l'opzione -e è necessaria solo quando si fornisce più di un'istruzione su riga di comando
 - Scrivendo le istruzioni in un file e fornendo il nome del file
 - **sed -f scriptfile <filename>**

Un file di esempio: lista di nomi ed indirizzi

```
$ cat list
```

```
John Dagget, 341 King Road, Plymouth MA
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
Sal Carpenter, 73 6th Street, Boston MA
```

(29)

Panoramica

- Vogliamo sostituire “MA” con “Massachusetts”

```
$ sed 's/MA/Massachusetts/' list
```

John Dagget, 341 King Road, Plymouth Massachusetts

Alice Ford, 22 East Broadway, Richmond VA

Orville Thomas, 11345 Oak Bridge Road, Tulsa OK

Terry Kalkas, 402 Lans Road, Beaver Falls PA

Eric Adams, 20 Post Road, Sudbury Massachusetts

Hubert Sims, 328A Brook Road, Roanoke VA

Amy Wilde, 334 Bayshore Pkwy, Mountain View CA

Sal Carpenter, 73 6th Street, Boston Massachusetts

(30)

Panoramica (II)

- Osservazione
 - Non è obbligatorio racchiudere l'istruzione tra apici ma è preferibile farlo
 - Si impedisce alla shell di interpretare caratteri speciali o spazi presenti nelle istruzioni

```
$ sed 's/ MA/, Massachusetts/' list
```

John Dagget, 341 King Road, Plymouth, Massachusetts

Alice Ford, 22 East Broadway, Richmond VA

Orville Thomas, 11345 Oak Bridge Road, Tulsa OK

Terry Kalkas, 402 Lans Road, Beaver Falls PA

Eric Adams, 20 Post Road, Sudbury, Massachusetts

Hubert Sims, 328A Brook Road, Roanoke VA

Amy Wilde, 334 Bayshore Pkwy, Mountain View CA

Sal Carpenter, 73 6th Street, Boston, Massachusetts

(31)

Istruzioni multiple

- Esistono tre possibilità per specificare istruzioni multiple sulla riga di comando

- Separare le istruzioni con ;

```
sed 's/ MA/, Massachusetts/; s/ PA/, Pennsylvania/' list
```

- Precedere ogni istruzione con -e

```
sed -e 's/ MA/, Massachusetts/' -e 's/ PA/, Pennsylvania/' list
```

- Impiegare la funzionalità multilinea della Bash

```
$ sed `
> s/ MA/, Massachusetts/
> s/ PA/, Pennsylvania/
> s/ CA/, California/' list
```

(32)

Segnalazione errori

- E' facile commettere errori di editing o omettere un elemento richiesto durante la stesura di un'istruzione

```
sed -e 's/MA/Massachusetts' list
sed: command garbled: s/MA/Massachusetts
```
- Sed solitamente visualizza ciascuna linea che non è in grado di eseguire
 - Ma non dice quale sia il problema con il comando
 - In questo caso manca lo slash che marca la porzione dell'istruzione di ricerca e sostituzione alla fine del comando di sostituzione

[33]

Colloquio in Informatica - Laboratorio di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

File di script

- Quando le istruzioni di editing sono numerose è preferibile inserirle in uno script
 - Lo script è una lista di comandi sed che sono eseguiti nell'ordine in cui compaiono
 - Questa forma usa l'opzione -f e richiede il nome del file relativo allo script

```
sed -f scriptfile file
```

- Es.: supponendo di inserire i comandi in uno script *sedscript*

```
$ cat sedscript
s/ MA/, Massachusetts/
s/ PA/, Pennsylvania/
s/ CA/, California/
s/ VA/, Virginia/
s/ OK/, Oklahoma/
```

[34]

Colloquio in Informatica - Laboratorio di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

File di script

```
$ sed -f sedscript list
```

```
John Dagget, 341 King Road, Plymouth, Massachusetts
Alice Ford, 22 East Broadway, Richmond, Virginia
Orville Thomas, 11345 Oak Bridge Road, Tulsa, Oklahoma
Terry Kalkas, 402 Lans Road, Beaver Falls, Pennsylvania
Eric Adams, 20 Post Road, Sudbury, Massachusetts
Hubert Sims, 328A Brook Road, Roanoke, Virginia
Amy Wilde, 334 Bayshore Pkwy, Mountain View, California
Sal Carpenter, 73 6th Street, Boston, Massachusetts
```

- Le modifiche sono mostrate a video ma il file list non è modificato!

[35]

Colloquio in Informatica - Laboratorio di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

Display automatico delle linee di input

- Di default, sed visualizza ogni linea di input
- L'opzione -n sopprime l'output automatico
 - Quando si specifica tale opzione, ogni istruzione a cui si intende far produrre un output deve contenere il comando **p**

```
$ sed -n '/MA/p' list
```

```
John Dagget, 341 King Road, Plymouth MA
```

```
Eric Adams, 20 Post Road, Sudbury MA
```

```
Sal Carpenter, 73 6th Street, Boston MA
```

[36]

Colloquio in Informatica - Laboratorio di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

Indirizzamento

- In sed ogni istruzione è composta di due parti: un pattern ed una procedura
 - Il pattern è un'espressione regolare delimitata con gli slash (/)
 - Una procedura specifica una o più azioni da eseguire

Espressione regolare	Descrizione
/./	Restituirà tutte le linee che contengono un solo carattere
/../	Restituirà tutte le linee che contengono due soli caratteri
/^#/	Restituirà tutte le linee che iniziano con #
/^\$/	Restituirà tutte le linee vuote
/}\$	Restituirà tutte le linee che terminano con }
/} *\$/	Restituirà tutte le linee che terminano con } seguito da 0 o più spazi
/[abc]/	Restituirà tutte le linee che contengono a, b o c
/^[abc]/	Restituirà tutte le linee che cominciano con a, b o c

[37]

CdL in Informatica - Laboratori di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

Indirizzamento

- Ad ogni comando è possibile associare opzionalmente una linea di indirizzo
 - Si usa per decidere quale linea editare
- L'indirizzo può essere in forma di numeri, espressioni regolari o combinazione dei due
 - Se non è specificato alcun indirizzo il comando è applicato ad ogni linea
 - Se è specificato un solo indirizzo, il comando è applicato ad ogni linea che fa il match con l'indirizzo
 - Se sono specificati due indirizzi separati da , il comando è eseguito sulla prima linea che fa il match con il primo indirizzo e tutte le linee successive fino alla linea (inclusa) che fa il match con il secondo indirizzo
 - Se un indirizzo è seguito da !, il comando è applicato a tutte le linee che non fanno il match con l'indirizzo

[38]

CdL in Informatica - Laboratori di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

Indirizzamento (cont.)

- Indirizzi
 - **n** linea numero n
 - **n,m** le linee dal numero n ad m
 - **/RegExp/** le linee che corrispondono a RegExp
 - **/RegExp1/,/RegExp2/** tutte le linee tra ogni occorrenza di RegExp1 e la successiva occorrenza di RegExp2

[39]

CdL in Informatica - Laboratori di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

Esempio

```
$ sed -n '/VA/,/CA/p' list
Alice Ford, 22 East Broadway, Richmond VA
Orville Thomas, 11345 Oak Bridge Road, Tulsa OK
Terry Kalkas, 402 Lans Road, Beaver Falls PA
Eric Adams, 20 Post Road, Sudbury MA
Hubert Sims, 328A Brook Road, Roanoke VA
Amy Wilde, 334 Bayshore Pkwy, Mountain View CA
```

[40]

CdL in Informatica - Laboratori di ISO - A.A. 2015/2016 - Prof. Antonino Staiano

Editing

- Alcuni comandi di Editing
 - p** stampa la linea corrente
 - d** cancella la linea corrente
 - s/RegExp/Repl/** sostituisce la prima occorrenza (per ogni riga) di RegExp con Repl
 - s/RegExp/Repl/g** sostituisce ogni occorrenza di RegExp con Repl

[41]

Esempi di editing


- Rimuove tutte le linee che contengono la stringa Richmond
 - sed '/Richmond/d' list**
- Stampa solo le prime dieci linee di un file (simile a head)
 - sed -n '1,5p' list**
- Aggiunge due spazi all' inizio di ogni riga
 - sed 's/^/ /' list**
- Mostra il file delle password, fornendo solo le linee da root a nobody
 - sed -n '/^root/,/^nobody/p' /etc/passwd**
- Elimina l'indentazione in un file, ovvero elimina tutti gli spazi ad inizio riga
 - sed 's/^ *//' filename**

[42]

Il comando: substitute

- Si consideri il caso in cui **sed** venga utilizzato per effettuare la sostituzione (s=substitute):

- sed "s/cane/gatto/g" animali.txt**
 - **s**= sostituire
 - la parola **cane**
 - con la parola **gatto**
 - **g**= global , ossia per tutte le sue ripetizioni



```
cane
gatto volpe
elefante
CANE
234,33
5
6
7
8
```

- Substitute va a ricercare e sostituire le occorrenze del pattern A con il pattern B in ogni linea e, se seguito dal range "g", non si fermerà alla prima occorrenza ma effettuerà la sostituzione su tutte le occorrenze presenti su ciascuna linea del file considerato

[43]

Il comando sed: substitute

- E' possibile anche riferirsi ad una sostituzione che non sia case sensitive (distingua tra maiuscole e minuscole) aggiungendo l'opzione "I" :
 - sed "s/BEGIN/END/Ig" prova.txt**
- Oppure è possibile cancellare tutte le cifre in un file contenente i prezzi delle bevande, ad esempio, sostituendole con il simbolo "#":
 - sed 's/[[:digit:]]/#/g' bevande.txt**

[44]

Il comando SED: delete

- Con **sed** è possibile anche effettuare l'operazione di "d=delete" necessaria per cancellare
- In tal caso, indicando il numero di linea è possibile cancellare una singola linea di testo o un insieme di linee consecutive:
 - **sed '1d' animali.txt**
 - **sed '1~2d' animali.txt** (riga iniziale e passo)
 - **sed '1,3d' animali.txt** (dalla riga 1 alla riga 3)
- Se si vogliono cancellare tutte le righe vuote occorre considerare il comando:
 - **sed '/^\$/d' animali.txt**
- **sed '/windows/d' file**
 - cancella tutte le righe dell'input contenenti l'occorrenza "windows"

(45)

Il comando SED: print

- Esempio:
 - **sed -n '/cane/p' animali.txt**
- Stampa le linee contenenti **cane**
 - **sed = animali.txt** (numera le righe del file)

(46)

Altre applicazioni di SED : append

- Supponiamo di voler aggiungere del testo dopo un pattern trovato (ad esempio, "cane")
- Si consideri:
 - **sed '/cane/a\:pastore' animali.txt**
- Il risultato è
cane
:pastore

(47)

Altre applicazioni di SED : insert

- Supponiamo di voler inserire del testo prima di un pattern trovato (ad esempio, "cane")
- Si consideri:
 - **sed '/cane/i\Tipologia di :\' animali.txt**

(48)

Altre applicazioni di SED : change

- Con **sed** è possibile anche sostituire un range di righe con un testo predefinito
- Esempio:
 - sed '/cane/,/gatto/c\ RIGA MANCANTE ' animali.txt**
 - Il risultato è:

```
RIGA MANCANTE
elefante
CANE
234,33
5
6
7
8
```

```
cane
gatto volpe
elefante
CANE
234,33
5
6
7
8
```

(49)

Comando SED:schema

- Esempi di espressioni regolari:

Notazione	Effetto
<code>ed</code>	Cancella l'ottava riga dell'input.
<code>/^\$/d</code>	Cancella tutte le righe vuote.
<code>1,/^\$/d</code>	Cancella dall'inizio dell'input fino alla prima riga vuota compresa.
<code>/Jones/p</code>	Visualizza solo le righe in cui è presente "Jones" (con l'opzione -n).
<code>s/Windows/Linux/</code>	Sostituisce con "Linux" la prima occorrenza di "Windows" trovata in ogni riga dell'input.
<code>s/BSOD/stabilità/g</code>	Sostituisce con "stabilità" tutte le occorrenze di "BSOD" trovate in ogni riga dell'input.
<code>s/ *\$ /</code>	Cancella tutti gli spazi che si trovano alla fine di ogni riga.
<code>s/00*/0/g</code>	Riduce ogni sequenza consecutiva di zeri ad un unico zero.
<code>/GUI/d</code>	Cancella tutte le righe in cui è presente "GUI".
<code>s/GUI//g</code>	Cancella tutte le occorrenze di "GUI", lasciando inalterata la parte restante di ciascuna riga.

(50)

AWK

(51)

Manipolare testi con awk

- awk** è un utility per la manipolazione di testi e la generazione di report, il cui nome deriva dalle iniziali degli autori (Aho, Weinberger, Kernighan)
 - La sintassi, nelle forme più semplici, è la seguente:
 - awk '<istruzioni>' <file>**
 - awk -f <progfile> <file>**
- scorre un file (o più) ed esegue alcune azioni sulle linee che soddisfano una data condizione in accordo con quanto specificato dalle istruzioni
- La sintassi dei programmi è simile a quella del linguaggio C
 - Si tratta di una utility estremamente flessibile e potente. Qui ci limiteremo a vedere solo alcuni esempi

(52)

Il comando AWK

- Il comando **awk** combina le funzionalità di **sed** e **grep**, divenendo uno dei comandi più importanti di GNU/Linux
- E' un linguaggio di selezione ed elaborazione di testi e viene chiamato anche "filtro programmabile" o "pattern matcher"
- È un linguaggio molto usato per costruire semplici script, sebbene possieda un'ampia serie di funzionalità e di operatori, di cui analizzeremo solo quelle basilari per lo scripting di shell
- **awk** suddivide ogni riga dell'input in **campi**, ossia una stringa di caratteri consecutivi separati da spazi (anche se esistono opzioni per modificare il delimitatore)
- **awk**, quindi, analizza e agisce su ciascun singolo campo. Questo lo rende ideale per trattare file di testo strutturati e dati organizzati in righe e colonne

[53]

Il comando AWK

- Un programma **awk** è composto da una o più specifiche ciascuna con la seguente struttura:
 - **awk criterio_di_selezione(pattern) {azione da compiere}**
dove il "criterio di selezione" identifica i record del file su cui agire e l'azione indica quali trasformazioni devono subire i record selezionati
- Nel caso più semplice (che verrà qui trattato) i record sono le righe di un file di testo
- Osserviamo che in tal caso ogni parola della riga è assegnata ad una variabile numerata in sequenza progressiva : \$1,\$2,\$3.... ; mentre la variabile \$0 contiene l'intera riga
- Un esempio di criterio di selezione potrebbe essere specificare una stringa che le righe selezionate devono contenere. L'azione potrebbe essere semplicemente la stampa di una parte o dell'intera riga selezionata

[54]

Il comando AWK

- Un comando **awk** può contenere l'indicazione esplicita del solo criterio di selezione, o della sola azione da compiere
 - Il motivo è che esistono azioni o criteri di selezione predefiniti (come vedremo nei primi esempi)
- L'azione di una regola **awk** è molto simile a un programma C, o Perl, dove il record selezionato viene passato attraverso i **campi**
- In pratica, l'azione di una regola **awk** è un programma a sé stante, che viene eseguito ogni volta che il criterio di selezione della regola si verifica

[55]

Il comando AWK

- Variabili predefinite in AWK

Variabile	Descrizione
CONVFMT	Formato di conversione da numero a stringa.
FILENAME	Nome del file attuale in ingresso, oppure -.
FNR	Numero del record attuale nel file attuale.
FS	Separatore dei campi in lettura.
NF	Numero totale dei campi nel record attuale.
NR	Numero totale dei record letti fino a questo punto.
OFMT	Formato di emissione dei numeri (di solito si tratta di % .6g).
OFS	Separatore dei campi per print .
ORS	Separatore dei record per print .
RS	Separatore dei record in lettura.
RSTART	Utilizzata da match() per annotare l'inizio di una corrispondenza.
RLENGTH	Utilizzata da match() per annotare la lunghezza di una corrispondenza.

[56]

Il comando awk: esempio 1

- Si consideri il contenuto delle directory

```
$ls -l
total 48
-rw-r--r-- 1 knoppix knoppix 1014 May 20 17:40 CD-ROM [cdrom1]
-rw-r--r-- 1 knoppix knoppix 1000 May 20 17:40 CD-ROM [cdrom]
-rw-r--r-- 1 knoppix knoppix 686 Mar 31 16:22 Floppy
-rw-r--r-- 2 knoppix knoppix 189 May 20 16:54 KNOPPIX.desktop
-rw-r--r-- 1 knoppix knoppix 3792 May 20 17:39 demosed4
-rw-r--r-- 1 knoppix knoppix 3792 May 20 17:37 demosed4~
-rw-r--r-- 1 knoppix knoppix 463 May 20 17:40 hda1
-rw-r--r-- 1 knoppix knoppix 463 May 20 17:40 hdc1
-rw-r--r-- 1 knoppix knoppix 463 May 20 17:40 hdc5
-rw-r--r-- 1 knoppix knoppix 4801 May 20 16:55 trash.desktop
-rw-r--r-- 1 knoppix knoppix 479 May 20 17:40 uba1
```

(57)

Il comando awk: esempio 1

- Listiamo il contenuto delle directory senza il totale

```
$ls -l | tail -n +2
-rw-r--r-- 1 knoppix knoppix 1014 May 20 17:40 CD-ROM [cdrom1]
-rw-r--r-- 1 knoppix knoppix 1000 May 20 17:40 CD-ROM [cdrom]
-rw-r--r-- 1 knoppix knoppix 686 Mar 31 16:22 Floppy
-rw-r--r-- 2 knoppix knoppix 189 May 20 16:54 KNOPPIX.desktop
-rw-r--r-- 1 knoppix knoppix 3792 May 20 17:39 demosed4
-rw-r--r-- 1 knoppix knoppix 3792 May 20 17:37 demosed4~
-rw-r--r-- 1 knoppix knoppix 463 May 20 17:40 hda1
-rw-r--r-- 1 knoppix knoppix 463 May 20 17:40 hdc1
-rw-r--r-- 1 knoppix knoppix 463 May 20 17:40 hdc5
-rw-r--r-- 1 knoppix knoppix 4801 May 20 16:55 trash.desktop
-rw-r--r-- 1 knoppix knoppix 479 May 20 17:40 uba1
```

(58)

Il comando awk: esempio 1

- estraiamo soltanto il campo numero 1 e il numero 9 mettendo il risultato in un file chiamato "out"

```
$ls -l | tail +2 > out
```

- estraiamo soltanto il campo numero 1 e il numero 9

```
$awk '{print $1,$9}' out
```

```
-rw-r--r-- CD-ROM
-rw-r--r-- CD-ROM
-rw-r--r-- Floppy
-rw-r--r-- KNOPPIX.desktop
-rw-r--r-- demosed4
-rw-r--r-- demosed4~
-rw-r--r-- hda1
-rw-r--r-- hdc1
-rw-r--r-- hdc5
-rw-r--r-- trash.desktop
-rw-r--r-- uba1
```

(59)

Il comando awk: esempio 1

awk '{print \$1,\$8}' out ... estraiamo i campi 1 e 8

```
-rw-r--r-- 17:40
-rw-r--r-- 17:40
-rw-r--r-- 16:22
-rw-r--r-- 16:54
-rw-r--r-- 17:39
-rw-r--r-- 17:37
-rw-r--r-- 17:40
-rw-r--r-- 17:40
-rw-r--r-- 17:40
-rw-r--r-- 16:55
-rw-r--r-- 17:40
```

(60)

Il comando awk: esempio 1

`$awk '{print $2,$3}' out` ... estraiamo i campi 2 e 3

```
1 knoppix
1 knoppix
1 knoppix
2 knoppix
1 knoppix
1 knoppix
1 knoppix
1 knoppix
1 knoppix
1 knoppix
1 knoppix
```

(61)

Il comando awk: esempio 1

- Estraiamo i campi 1,8,9

`$awk '{print $1,$8,$9}' out`

```
-rw-r--r-- 17:40 CD-ROM
-rw-r--r-- 17:40 CD-ROM
-rw-r--r-- 2003 Floppy
-rw-r--r-- 16:54 KNOPPIX.desktop
-rw-r--r-- 17:39 demosed4
-rw-r--r-- 17:37 demosed4~
-rw-r--r-- 17:40 hda1
-rw-r--r-- 17:40 hdc1
-rw-r--r-- 17:40 hdc5
-rw-r--r-- 16:55 trash.desktop
-rw-r--r-- 17:40 uba1
```

(62)

Il comando awk: esempio 2

- si consideri un output formattato mediante printf

`$awk '{printf ("%s %d\n", $9,$5)}' out` (campo 9 formattato come stringa e il 5 come intero)

```
CD-ROM 1014
CD-ROM 1000
Floppy 686
KNOPPIX.desktop 189
demosed4 3792
demosed4~ 3792
hda1 463
hdc1 463
hdc5 463
trash.desktop 4801
uba1 479
```

(63)

Il comando awk: esempio 2

- si consideri un output formattato e ordinato mediante SORT
`$awk '{printf ("%s %6d\n", $9,$5)}' out | sort` (campo 9 come stringa e il 5 come decimale a 6 cifre)

```
CD-ROM 1000
CD-ROM 1014
demosed4 3792
demosed4~ 3792
Floppy 686
hda1 463
hdc1 463
hdc5 463
KNOPPIX.desktop 189
trash.desktop 4801
uba1 479
```

(64)

Il comando AWK: esempi

- Con **awk** è anche possibile compiere calcoli; si consideri un file di dati in cui sono contenuti gli stipendi degli ultimi tre mesi e si calcoli la media:

Antonio	200	400	500
Maria	100	150	20
Luca	300	400	120

- awk '{print "La media del sign.", \$1, (\$2+\$3+\$4)/3}' numeri.txt**
- Risultato:
 - La media del sign. Antonio 333.333
 - La media del sign. Maria 150
 - La media del sign. Luca 433.333

(65)

Il comando AWK: esempi

- Si consideri l'uso di **awk** con selezione del pattern. Ad es. si vuole operare la somma degli stipendi mensili solo relativamente ad un dipendente. In riferimento al file precedente si ha:

```
awk '/^Ant/ {print "La somma di", $1, "è", ($2+$3+$4)}' numeri.txt
```

Pattern selezionato: tutti i nomi che iniziano per "Ant"

Antonio	200	400	500
Maria	100	150	20
Luca	300	400	120

- ottenendo:
 - La somma di Antonio è 1100

(66)

Il comando AWK: esempi

- Creazione di un file e scambio degli elementi interni:

```
$ cat dati1
```

1 Dic 1990

12 Dic 1990

5 Feb 1991

(67)

Il comando awk: esempio 3

- Scambiamo le componenti del file creato

```
$ awk '{printf ("%4d %s\n", NR, $0)}' dati1
```

1 1 Dic 1990

2 12 dic 1990

3 5 Febb 1991

```
$ awk '{printf ("%4d%s\n", NR, $0)}' dati1 | sort -r
```

3 5 Febb 1991

2 12 dic 1990

1 1 Dic 1990

```
$ awk '{printf ("%4d%s\n", NR, $0)}' dati1 | sort -r | cut -c5-
```

5 Febb 1991

12 dic 1990

1 Dic 1990

\$echo scambio delle righe nel file originario dati1

(68)

Comando AWK: script di esempio 3

- Il file originale

1 Dic 1990

12 dic 1990

5 febb 1991

riformatto il file stampando i numeri decimali su 4 cifre e le stringhe con il d'accapo

```
awk '{printf ("%4d %s\n",NR,$0)}' datil.txt
```

echo "inserisco un numero progressivo avanti"

riordino al contrario quanto ottenuto

```
awk '{printf ("%4d %s\n",NR,$0)}' datil.txt |sort -r
```

echo " riordino al contrario"

elimino i numeri progressivi che mi sono stati utili per il riordino

```
awk '{printf ("%4d %s\n",NR,$0)}' datil.txt| sort -r  
|cut -c5-
```

echo "elimino i numeri usati per il riordino e stampo tutto a video"

echo " FINE"

(69)

Esempi

- Consideriamo il file datafile

northwest	NW	Charles Main	3.0	.98	3	34
western	WE	Sharon Gray	5.3	.97	5	23
southwest	SW	Lewis Dalsass	2.7	.8	2	18
southern	SO	Suan Chin	5.1	.95	4	15
southeast	SE	Patricia Liu	4.0	.7	4	17
eastern	EA	TB Savage	4.4	.84	5	20
northeast	NE	AM Main Jr.	5.1	.94	3	13
north	NO	Margot Weber	4.5	.89	5	9
central	CT	Ann Stephens	5.7	.94	5	13

(70)

Stampa delle righe

- sed '/north/p' datafile
- Stampa tutte le righe sullo standard output per default
- Se è trovato il pattern north, sed stamperà quella riga in aggiunta a tutte le altre righe

(71)

Stampa delle righe (cont.)

- sed -n '/north/p' datafile
- L'opzione -n sopprime il comportamento di default di sed
- Quando è usato in combinazione con il comando p, la riga nel buffer dei pattern è stampata solo una volta

(72)

Cancellazione righe

- `sed '3d' datafile`

Cancella la terza riga. Tutte le altre righe sono stampate a video di default

- `sed '3,$d' datafile`

Le righe tra la terza e l'ultima sono cancellate

- Il `$` rappresenta l'ultima riga del file
- `sed '$d' datafile`

Cancella l'ultima riga

- Il default è stampare tutte le righe eccetto quella coinvolta nel comando `d`
- `sed '/north/d' datafile`

Tutte le righe contenente il pattern `north` sono cancellate

(73)

Sostituzione

- `sed 's/west/north/g' datafile`

Se sono individuate occorrenze multiple del pattern `west`, tutte saranno sostituite con `north`

- `sed -n 's/^west/north/p' datafile`

L'opzione `-n` con il flag `p` informa `sed` di stampare solo le righe in cui è avvenuta la sostituzione

- `sed 's/[0-9][0-9]$/&.5/' datafile`

`&` rappresenta esattamente cosa è stato trovato nella stringa di ricerca. Ogni riga che finisce con due cifre sarà sostituita con se stessa e con `.5` concatenato

- `sed -n 's/Liu/Jones/gp' datafile`

Tutte le occorrenze di `Liu` sono sostituite con `Jones` e solo le righe modificate sono stampate

(74)

Append

```
sed '/^north /a\  
--->THE NORTH SALES DISTRICT HA MOVED< ---' datafile
```

La stringa è aggiunta dopo la riga che inizia con il pattern `north`, quando `north` è seguito da uno spazio

(75)

Insert

```
sed '/eastern/i\  
NEW ENGLAND REGION\  
-----'  
' datafile
```

- Se è trovata la corrispondenza con il pattern `eastern`, il testo che segue il backslash è inserito sopra la riga che contiene `eastern`
- E' richiesto un backslash dopo ciascuna riga da inserire, eccetto l'ultima riga

(76)

Il comando sed: esempio 1

```
$cat sedprova
ffffffffff
ffff
abc
cde

fgh
$sed '/^$/d' sedprova #Elimina eventuali righe
vuote...
ffffffffff
ffff
abc
cde
fgh
$
Eliminiamo la prima riga:
$sed '1d' sedprova
ffff
abc
cde
fgh
```

(77)

Il comando sed: esempio 1

Eliminiamo la riga 1 e 2

```
$sed '1,2d' sedprova
abc
cde
fgh
```

Aggiungiamo elementi a sedprova

```
$cat sedprova
#commento
fff
rrr
tttttt
gggggggggggg
Eliminiamo le righe che iniziano per #
$sed '/^#/d' sedprova
fff
rrr
tttttt
gggggggggggg
```

(78)

Il comando sed: esempio 2

```
$cat sedprova
#commento
fff
rrr
tttttt
gggggggggggg
Stampa solo le linee che contengono "c" oppure "o" e ...tutte le
altre
$sed '/[co]/p' sedprova
#commento
#commento
fff
rrr
tttttt
gggggggggggg
Stampa linee che contengono "f" e ..tutte le altre
$sed '/[f]/p' sedprova
#commento
fff
fff
rrr
tttttt
gggggggggggg
```

(79)

Il comando sed: esempio 2

Elimina linee che contengono la lettera "f"

```
$sed '/[f]/d' sedprova
#commento
rrr
tttttt
gggggggggggg
Elimina linee che contengono "c" oppure "o"
$sed '/[co]/d' sedprova
fff
rrr
tttttt
gggggggggggg
ora con l'opzione '-n' non stampa tutte le righe se non è stato
esplicitamente dichiarato
$sed -n '/rrr/p' sedprova
rrr
```

(80)

Il comando sed: esempio 3

Si consideri il file

```
$cat sedprova
saluti a tutti
oggi è una bella gironata
...errato ....volevo dire giornata!
BEGIN
prova prova 1
prova2
END tutto ok
prova
ciao
```

Stampiamo a video tutto quanto contenuto tra BEGIN ed END

```
$sed -n '/BEGIN/,/END/p' sedprova
BEGIN
prova prova 1
prova2
END tutto ok
```

(81)

Il comando sed: esempio 4

Comando di sostituzione

Sostituiamo BEGIN con COMINCIA

```
$sed 's/BEGIN/COMINCIA/' sedprova
saluti a tutti
oggi è una bella gironata
...errato ....volevo dire giornata!
COMINCIA
prova prova 1
prova2
END tutto ok
prova
ciao
```

(82)

Il comando sed: esempio 4

sperimentiamo lo stesso comando con il seguente file

```
$cat sedprova2
saluti
BEGIN
prova
prova1
prova2
END
saluti
BEGIN
prova4
END
$sed 's/BEGIN/COMINCIA/' sedprova2
saluti
COMINCIA
prova
prova1
prova2
END
saluti
COMINCIA
prova4
END
```

(83)

Il comando SED: esempio 4

Si consideri ora il seguente file sedprova3

```
saluti
BEGIN BEGIN
prova 1
prova2
END
```

Si ripeta la sostituzione di prima

```
$sed 's/BEGIN/COMINCIA/' sedprova3
saluti
COMINCIA BEGIN
prova 1
prova2
END
```

(84)

Il comando SED: esempio 4

Notiamo che ha cambiato soltanto la prima occorrenza del BEGIN

occorre variare il comando per sostituire tutte le occorrenze

```
$sed 's/BEGIN/COMINCIA/g' sedprova3
```

```
saluti
COMINCIA  COMINCIA
prova 1
prova2
END
```

(85)

CdL in Informatica - Laboratori di ISO - AA. 2015/2016 - Prof. Antonino Staiano

Il comando sed: esempio 5

supponiamo di voler inserire avanti a ciascuna linea la stringa OK

```
$sed 's/^/ OK:/' sedprova3
```

```
OK:saluti
OK:BEGIN  BEGIN
OK:prova 1
OK:prova2
OK:END
```

si provi il comando

(86)

CdL in Informatica - Laboratori di ISO - AA. 2015/2016 - Prof. Antonino Staiano

Il comando sed: esempio 5

si provi il comando

```
$sed -n 's/;/p' sedprova3
```

```
1
saluti
2
BEGIN  BEGIN
3
prova 1
4
prova2
5
END
```

con il “;” siamo riusciti ad eseguire più di un comando per volta

(87)

CdL in Informatica - Laboratori di ISO - AA. 2015/2016 - Prof. Antonino Staiano

Il comando sed: esempio 6

```
$ sed 'i\proviamolo' sedprova3 (aggiunge avanti ad ogni riga la parola “proviamolo”)
```

```
proviamolo
saluti
proviamolo
BEGIN BEGIN
proviamolo
prova 1
proviamolo
prova2
proviamolo
END
```

(88)

CdL in Informatica - Laboratori di ISO - AA. 2015/2016 - Prof. Antonino Staiano

Il comando sed: esempio 6

inseriamo dopo ogni linea una parola

```
$ sed 'a\proviavamoolo' sedprova3
```

saluti

proviavamoolo

BEGIN BEGIN

proviavamoolo

prova 1

proviavamoolo

prova2

proviavamoolo

END

proviavamoolo