

Redirezione, filtri, espressioni regolari

Andrea Passerini
passerini@disi.unitn.it

Informatica

Dispositivi standard di I/O

- I programmi scritti per terminali a carattere (compresi i comandi per shell) usano 3 dispositivi standard di I/O:

`stdin` (input)

`stdout` (output)

`stderr` (error)

- Normalmente `stdin` è collegato alla tastiera, mentre `stdout` e `stderr` sono collegati al terminale video a caratteri
- I dispositivi possono essere “rediretti” su file o in ingresso ad altri comandi tramite gli operatori di redirezione `>`, `>>`, `<`, `|`.

Esempi di redirectione

Redirezionare l'output su file

- Redirezionare l'output su file permette di memorizzare in maniera rapida l'output dei comandi, invece di vederlo e poi perderlo come accade quando va su stdout
- L'operatore `>` redireziona l'output su un file, fornito come argomento dopo l'operatore
- Il file viene creato se non esiste, o sovrascritto se esistente

```
[andrea@praha Data]$ ls
Fasta  Labels  README  Results  tmp2
[andrea@praha Data]$ ls > ls.out
[andrea@praha Data]$ ls
Fasta  Labels  ls.out  README  Results  tmp2
[andrea@praha Data]$ cat ls.out
Fasta
Labels
ls.out
README
Results
tmp2
```

Appendere l'output ad un file

- L'operatore `>>` redireziona l'output su un file, fornito come argomento dopo l'operatore
- Il file viene creato se non esiste, ma se esiste, l'output viene aggiunto in fondo al file, senza sovrascriverne il contenuto

```
[andrea@praha Data]$ ls tmp2/  
tmp  tmp3  
[andrea@praha Data]$ ls tmp2/ >> ls.out  
[andrea@praha Data]$ cat ls.out  
Fasta  
Labels  
ls.out  
README  
Results  
tmp2  
tmp  
tmp3
```

Filtri

- I sistemi basati su UNIX fanno ampio uso di questi operatori di redirezione
- Esiste una vasta gamma di comandi che opera prendendo del testo dallo standard input, processandolo in qualche modo, e mandando il risultato su standard output
- Tali comandi prendono il nome di *filtri* poiché operano come dei filtri su un flusso di dati
- L'operatore < seguito dal nome di un file, redireziona il contenuto del file sullo standard input di un comando
- Applicata ad un filtro, la redirezione dell'input permette di filtrare il contenuto del file

Ordinamento con `sort`

- Il filtro `sort` resituisce in output una versione dell'input con le righe riordinate lessicograficamente

```
[andrea@praha ~]$ cat data
```

```
S'same old string'
```

```
p0
```

```
.(I-1
```

```
F4.5
```

```
aI1
```

```
I1
```

```
s.
```

```
[andrea@praha ~]$ sort < data
```

```
aI1
```

```
F4.5
```

```
I1
```

```
.(I-1
```

```
p0
```

```
s.
```

```
S'same old string'
```

l'operatore |

- L'operatore | (*pipe*) posto tra due comandi, prende l'output del primo e lo redireziona come input al secondo
- Tale operatore è fondamentale in quanto permette di creare *pipelines* di comandi, in cui vari filtri sono applicati in cascata ciascuno sull'output del filtro precedente

```
[andrea@praha ~]$ cat data | sort
aI1
F4.5
I1
.(I-1
p0
s.
S'same old string'
```

Eliminazione di replicati con `unique`

- Il filtro `unique` permette di eliminare le righe replicate di un file, stampando un solo esemplare per ogni riga.
- Poiché le righe replicate devono essere contigue, il filtro è spesso usato in cascata ad un filtro `sort`

```
[andrea@praha Data]$ cat Labels/labels | sort | uniq
```

```
0
```

```
0 1
```

```
0 1 2
```

```
0 1 2 10
```

```
0 1 2 18
```

```
0 1 2 3
```

```
0 1 2 5
```

```
0 1 2 7
```

```
0 1 2 8
```

Eliminazione di replicati con `unique`

- L'opzione `-c` di `unique` permette di anteporre ad ogni riga il numero di duplicati trovati

```
[andrea@praha Data]$ cat Labels/labels | sort | uniq -c
199
  2 0
  4 0 1
  2 0 1 2
  1 0 1 2 10
  1 0 1 2 18
25 0 1 2 3
  2 0 1 2 5
  1 0 1 2 7
  1 0 1 2 8
```

Eliminazione di replicati con `unique`

- L'opzione `-d` di `unique` permette di stampare solo le righe duplicate

```
[andrea@praha Data]$ cat Labels/labels | sort | uniq -d
```

```
0  
0 1  
0 1 2  
0 1 2 3  
0 1 2 5  
0 1 3 4  
0 1 4 5  
0 1 4 6
```

Redirezione su file

- E' sempre possibile, dopo aver applicato una serie di filtri in pipeline, redirezionare l'uscita finale su file
- Se un comando non entra in una singola riga, è possibile spezzarlo su più righe terminando ogni riga con \ (e premendo invio)

```
[andrea@praha Data]$ cat Labels/labels | sort \  
> | uniq -dc > uniq.txt
```

```
[andrea@praha Data]$ cat uniq.txt
```

```
199
```

```
2 0
```

```
4 0 1
```

```
2 0 1 2
```

```
25 0 1 2 3
```

```
2 0 1 2 5
```

```
12 0 1 3 4
```

```
8 0 1 4 5
```

```
2 0 1 4 6
```

Letture in reverse con `tac`

- `tac` esegue la concatenazione di file come `cat`, ma stampa ciascuno di essi dall'ultima alla prima riga
- Entrambi possono prendere dati da input con pipe invece che da file

```
[andrea@praha Data]$ ls | cat
```

```
Fasta
```

```
Labels
```

```
README
```

```
Results
```

```
tmp2
```

```
[andrea@praha Data]$ ls | tac
```

```
tmp2
```

```
Results
```

```
README
```

```
Labels
```

```
Fasta
```

Prime righe con head

- head legge le prime n righe dell'input, dove n viene specificato con l'opzione `- <n>` (o 10 di default)

```
[andrea@praha Data]$ ls | tac | head -3  
uniq.txt  
tmp2  
Results
```

Ultime righe con `tail`

- `tail` legge le ultime n righe dell'input, dove n viene specificato con l'opzione `- <n>` (o 10 di default)

```
[andrea@praha Data]$ ls | tac
```

```
uniq.txt
```

```
tmp2
```

```
Results
```

```
README
```

```
ls.out
```

```
Labels
```

```
Fasta
```

```
[andrea@praha Data]$ ls | tac | tail -2
```

```
Labels
```

```
Fasta
```

Righe “a partire da” con `tail`

- con la sintassi `-n +<x>`, `tail` stampa tutte le righe a partire dalla numero `<x>`

```
[andrea@praha Data]$ ls | tac | tail -n +3
Results
README
ls.out
Labels
Fasta
```

Scelta di colonne con `cut`

- `cut` permette di stampare solo una selezione delle colonne in ingresso
- l'opzione `-f <list>` permette di specificare una lista di colonne da stampare, come:
 - lista di numeri di colonna separati da virgola (e.g. `cut -f 1,4,5,7`)
 - range di numeri di colonna, indicando i limiti (e.g. `cut -f 1-4` `cut -f 5-9`)
 - range di numeri di colonna, indicando un solo limite (e.g. `cut -f 5-`, `cut -f -7`)
 - una qualunque combinazione delle modalità precedenti (e.g. `cut -f -5,7-9,10,11,15-`)

Scelta di colonne con `cut`

- di default, `cut` usa il TAB come delimitatore tra colonne
- E' possibile specificare un delimitatore diverso tramite l'opzione `-d`
- Tipici delimitatori sono lo spazio (e.g. `cut -d ' ' -f 1-5`), la virgola (e.g. `cut -d ',' -f 7-`).

```
[andrea@praha Data]$ ls -l
total 28
drwxr-xr-x 2 andrea andrea 4096 2008-12-22 13:06 Fasta
-r--r--r-- 1 andrea andrea   50 2009-01-14 17:55 README
drwxr-xr-x 4 andrea andrea 4096 2008-12-22 13:57 Results
dr-xr-xr-x 4 andrea andrea 4096 2009-01-14 17:42 tmp2
-rw-r--r-- 1 andrea andrea  797 2009-01-14 19:01 uniq.txt
[andrea@praha Data]$ ls -l | cut -d ' ' -f 3
```

```
andrea
andrea
andrea
andrea
andrea
```

Attenzione

- `cut` considera come colonna ciò che si trova tra due delimitatori successivi
- se compaiono più delimitatori in sequenza (e.g. una sequenza di spazi se si usa `-d ' '`), `cut` stamperà delle colonne vuote

```
[andrea@praha Data]$ ls -l | cut -d ' ' -f 4,5
```

```
andrea 4096  
andrea 4096  
andrea  
andrea  
andrea 4096  
andrea 4096  
andrea
```

Estrazione di statistiche con `wc`

- `wc` (WordCount) permette di estrarre statistiche sul numero di righe, parole e caratteri in ingresso
- Ad esempio per calcolare il numero di righe, parole e caratteri di un file

```
[andrea@praha Data]$ cat data.txt | wc
    55     225     797
```

- Per stampare il numero di parole distinte presenti in un file

```
[andrea@praha Data]$ cat data.txt | tr -s ' ' \
> | tr ' ' '\n' | sort | uniq | wc
    24     23     62
```

(NB: ci sono 24 righe e 23 parole, poiché una delle righe è vuota, quindi non contiene alcuna parola)

Modifica di caratteri con `tr`

- `tr` (TRAnslate) permette di modificare tutte le occorrenze di un carattere
- L'uso più semplice consiste nel convertire un carattere in un altro

```
[andrea@praha Data]$ ls -l | tr a b
totbl 28
drwxr-xr-x 2 bndreb bndreb 4096 2008-12-22 13:06 Fbstb
drwxr-xr-x 2 bndreb bndreb 4096 2009-01-14 17:37 Lbbels
-rw-r--r-- 1 bndreb bndreb   49 2009-01-14 18:30 ls.out
-r--r--r-- 1 bndreb bndreb   50 2009-01-14 17:55 README
drwxr-xr-x 4 bndreb bndreb 4096 2008-12-22 13:57 Results
dr-xr-xr-x 4 bndreb bndreb 4096 2009-01-14 17:42 tmp2
-rw-r--r-- 1 bndreb bndreb  797 2009-01-14 19:01 uniq.txt
```

Modifica di caratteri con `tr`

- E' possibile specificare come carattere anche caratteri speciali. I più utili sono il ritorno a capo (`\n`) e il TAB (`\t`)
- I caratteri speciali devono essere racchiusi tra apici

```
[andrea@praha Examples]$ ls -l | tr ' ' '\n' | head -10
total
32
-rw-r--r--
1
andrea
andrea

5231
2009-01-12
15:35
```

Modifica di caratteri con `tr`

- E' possibile specificare come carattere anche caratteri speciali. I più utili sono il ritorno a capo (`\n`) e il TAB (`\t`)
- I caratteri speciali devono essere racchiusi tra apici

```
[andrea@praha Examples]$ ls -l | tr ' ' '\n' | head -10
total
32
-rw-r--r--
1
andrea
andrea

5231
2009-01-12
15:35
```

Modifica di caratteri con `tr`

- E' possibile specificare set di caratteri invece che caratteri singoli.
- Il modo più semplice è specificando il range, come abbiamo visto per le wildcards (e.g. 0-9, a-z)
- Esempio: convertire minuscole in maiuscole

```
[andrea@praha Examples]$ ls -l | tr a-z A-Z
TOTAL 32
-rw-r--r-- 1 ANDREA ANDREA 5231 2009-01-12 15:35 DATA.DST
-rw-r--r-- 1 ANDREA ANDREA 10549 2009-01-12 15:09 DATA.GENBANK
-rw-r--r-- 1 ANDREA ANDREA 10 2009-01-19 11:38 P
-rw-r--r-- 1 ANDREA ANDREA 865 2009-01-12 14:35 SP.FASTA
```

- Esempio: convertire numeri in lettere

```
[andrea@praha Examples]$ ls -l | tr 0-9 a-z
total dc
-rw-r--r-- b andrea andrea fcdb caaj-ab-bc bf:df data.dst
-rw-r--r-- b andrea andrea bafej caaj-ab-bc bf:aj data.genbank
-rw-r--r-- b andrea andrea ba caaj-ab-bj bb:di p
-rw-r--r-- b andrea andrea igf caaj-ab-bc be:df sp.fasta
```

Cancellazione di caratteri con `tr`

- L'opzione `-d` permette di specificare un set di caratteri da cancellare, invece che sostituirlo con altro

```
[andrea@praha Examples]$ ls -l | tr -d a
total 32
-rw-r--r-- 1 ndre ndre 5231 2009-01-12 15:35 dt.dst
-rw-r--r-- 1 ndre ndre 10549 2009-01-12 15:09 dt.genbnk
-rw-r--r-- 1 ndre ndre 10 2009-01-19 11:38 p
-rw-r--r-- 1 ndre ndre 865 2009-01-12 14:35 sp.fst
-rw-r--r-- 1 ndre ndre 669 2009-01-12 15:14 texput.log
[andrea@praha Examples]$ ls -l | tr -d 0-9
total
-rw-r--r--  andrea andrea  -- : data.dst
-rw-r--r--  andrea andrea  -- : data.genbank
-rw-r--r--  andrea andrea  -- : p
-rw-r--r--  andrea andrea  -- : sp.fasta
-rw-r--r--  andrea andrea  -- : texput.log
```

Eliminazione di duplicati con `tr`

- L'opzione `-s` (*squeeze*) permette di sostituire sequenze di un certo carattere con una singola occorrenza

```
[andrea@praha Examples]$ ls -l | tr -s -
total 32
-rw-r-r- 1 andrea andrea  5231 2009-01-12 15:35 data.dst
-rw-r-r- 1 andrea andrea 10549 2009-01-12 15:09 data.genbank
-rw-r-r- 1 andrea andrea   10 2009-01-19 11:38 p
-rw-r-r- 1 andrea andrea   865 2009-01-12 14:35 sp.fasta
-rw-r-r- 1 andrea andrea   669 2009-01-12 15:14 texput.log
[andrea@praha Examples]$ ls -l | tr -s 0-9
total 32
-rw-r--r-- 1 andrea andrea  5231 209-01-12 15:35 data.dst
-rw-r--r-- 1 andrea andrea 10549 209-01-12 15:09 data.genbank
-rw-r--r-- 1 andrea andrea   10 209-01-19 1:38 p
-rw-r--r-- 1 andrea andrea   865 209-01-12 14:35 sp.fasta
-rw-r--r-- 1 andrea andrea   69 209-01-12 15:14 texput.log
```

Eliminazione di duplicati con `tr`

- Esempio: selezione di proprietario e dimensione dei file con `tr` e `cut`:

```
[andrea@praha Examples]$ ls -l
total 32
-rw-r--r-- 1 andrea andrea  5231 2009-01-12 15:35 data.dst
-rw-r--r-- 1 andrea andrea 10549 2009-01-12 15:09 data.genbank
-rw-r--r-- 1 andrea andrea   10 2009-01-19 11:38 p
-rw-r--r-- 1 andrea andrea  865 2009-01-12 14:35 sp.fasta
-rw-r--r-- 1 andrea andrea  669 2009-01-12 15:14 texput.log
[andrea@praha Examples]$ ls -l | tr -s ' ' | cut -d ' ' -f 3,5

andrea 5231
andrea 10549
andrea 10
andrea 865
andrea 669
```

Selezione di righe tramite `grep`

- `grep` è un potente filtro che permette di selezionare righe che soddisfano determinate espressioni
- Nell'uso più semplice, `grep` seleziona le righe che contengono una determinata sequenza di caratteri

```
[andrea@praha Data]$ less ligands | grep HEM
101m_ H HEM H HEM
1a7vA C HEM C HEM H HEM
1cl6A H HEM H HEM C HEM
1cxyA C CYS H HEM C CYS H HEM
1dt6A E SM E SM H HEM C HEM
1e7pC H HEM H HEM H HEM H HEM
1lzoA H HEM H HEM C HEM
1mj4A H HEM H HEM H HEM
1oj6A H HEM H HEM
1q16C H HEM H HEM H HEM H HEM
1vf5A C HEM H HEM H HEM H HEM H HEM
```

Uso di *espressioni regolari* tramite `grep`

- In generale, `grep` permette di specificare delle *espressioni regolari* che devono essere soddisfatte dalle righe da selezionare (con l'opzione `-E`)
- Le wildcards viste precedentemente sono semplici esempi di espressioni regolari. `grep` supporta espressioni regolari più complesse (ma alcune caratteristiche sono comuni)
- Le espressioni regolari sono uno strumento estremamente utile per eseguire operazioni su sequenze, e sono usate diffusamente in bioinformatica
- Come le espressioni aritmetiche, le espressioni regolari sono costruite combinando espressioni più semplici tramite degli *operatori*

Espressioni regolari di base

- La più semplice espressione regolare è un singolo carattere, che rappresenta sé stesso

```
[andrea@praha Data]$ ls -l | grep R
-r--r--r-- 1 andrea andrea  50 2009-01-14 17:55 README
drwxr-xr-x 4 andrea andrea 4096 2008-12-22 13:57 Results
```

- Il punto `.` è un'espressione che rappresenta un qualsiasi carattere (ed ha senso combinato in espressioni complesse)
- Una *classe di carattere* può essere specificata racchiudendone la rappresentazione tra parentesi quadre
- Come per le wildcards, è possibile rappresentare esplicitamente insiemi di caratteri, o specificare dei range

Principali classi di caratteri

`[CDE]` uno qualsiasi dei caratteri specificati

`[A-Z]` un carattere nel range dei caratteri specificati

`[^0-9]` un carattere che non sia nel range dei caratteri specificati (NOTA: nelle wildcards si usa `!` invece di `^`)

`[[:space:]]` qualsiasi carattere di spaziatura (e.g. `\t\n o spazio`)

Ancore

- Il carattere speciale `^` indica l'inizio della riga

```
[andrea@praha Data]$ ls -l | grep -E ^d
drwxr-xr-x 2 andrea andrea 4096 2008-12-22 13:06 Fasta
drwxr-xr-x 2 andrea andrea 4096 2009-01-14 17:37 Labels
drwxr-xr-x 4 andrea andrea 4096 2008-12-22 13:57 Results
dr-xr-xr-x 4 andrea andrea 4096 2009-01-14 17:42 tmp2
```

- Il carattere speciale `$` indica la fine della riga

```
[andrea@praha Data]$ ls -l | grep -E t$
-rw-r--r-- 1 andrea andrea 49 2009-01-14 18:30 ls.out
-rw-r--r-- 1 andrea andrea 797 2009-01-14 19:01 uniq.txt
```

Ripetizione

- Una certa espressione regolare può essere seguita da un operatore di *ripetizione*
 - ? l'espressione è opzionale, e può essere soddisfatta al massimo una volta
 - * l'espressione può essere soddisfatta da zero ad un qualsiasi numero di volte
 - + l'espressione deve essere soddisfatta almeno una volta, o un qualsiasi numero maggiore di volte
 - {*n*} espressione soddisfatta esattamente *n* volte
 - {*n*,} espressione soddisfatta *n* volte o più
 - {, *m*} espressione soddisfatta al massimo *m* volte
 - {*n*, *m*} espressione soddisfatta tra *n* ed *m* volte

Ripetizione: esempi

```
[andrea@praha Data]$ ls -l | grep -E [5-7]{2}
-r--r--r-- 1 andrea andrea  50 2009-01-14 17:55 README
drwxr-xr-x 4 andrea andrea 4096 2008-12-22 13:57 Results
[andrea@praha Data]$ ls -l | grep -E [A-Z]+
drwxr-xr-x 2 andrea andrea 4096 2008-12-22 13:06 Fasta
drwxr-xr-x 2 andrea andrea 4096 2009-01-14 17:37 Labels
-r--r--r-- 1 andrea andrea  50 2009-01-14 17:55 README
drwxr-xr-x 4 andrea andrea 4096 2008-12-22 13:57 Results
```

Ripetizione: problema dell'espansione di shell

- Abbiamo visto che l'interprete di comandi *espande* le wildcards nella lista dei nomi di file che le soddisfano
- Questo avviene *prima* che il comando venga eseguito
- Nel caso di un comando che prende un'espressione regolare, come `grep`, se l'espressione regolare contiene dei caratteri da espandere, l'interprete li sostituirà con i nomi compatibili, e *dopo* eseguirà il comando.

Problema dell'espansione di shell: esempio

```
[andrea@praha Data]$ ls -l
total 28
drwxr-xr-x 2 andrea andrea 4096 2008-12-22 13:06 Fasta
drwxr-xr-x 2 andrea andrea 4096 2009-01-14 17:37 Labels
-rw-r--r-- 1 andrea andrea  49 2009-01-14 18:30 ls.out
-r--r--r-- 1 andrea andrea  50 2009-01-14 17:55 README
drwxr-xr-x 4 andrea andrea 4096 2008-12-22 13:57 Results
drwxr-xr-x 4 andrea andrea 4096 2009-01-19 14:37 tmp2
-rw-r--r-- 1 andrea andrea  797 2009-01-14 19:01 uniq.txt
[andrea@praha Data]$ ls -l | grep -E R*
[andrea@praha Data]$
```

- La shell espande `R*` con la lista dei file compatibili con tale wildcard (ossia tutti i file che cominciano per `R`). Si ottiene quindi:

```
[andrea@praha Data]$ ls -l | grep -E README Results
[andrea@praha Data]$
```

- Che significa *cerca l'espressione `README` nel file `Results`* (il secondo argomento di `grep` se presente indica il nome del file da usare come input)

Ripetizione: protezione dall'espansione di shell

- Per evitare che la shell espanda le wildcards, si deve racchiudere l'espressione regolare tra apici singoli

```
[andrea@praha Data]$ ls -l | grep 'R*'
total 28
drwxr-xr-x 2 andrea andrea 4096 2008-12-22 13:06 Fasta
drwxr-xr-x 2 andrea andrea 4096 2009-01-14 17:37 Labels
-rw-r--r-- 1 andrea andrea  49 2009-01-14 18:30 ls.out
-r--r--r-- 1 andrea andrea  50 2009-01-14 17:55 README
drwxr-xr-x 4 andrea andrea 4096 2008-12-22 13:57 Results
drwxr-xr-x 4 andrea andrea 4096 2009-01-19 14:37 tmp2
-rw-r--r-- 1 andrea andrea  797 2009-01-14 19:01 uniq.txt
```

Ripetizione: protezione dall'espansione di shell

- Di norma è sempre utile proteggere l'espressione regolare, per evitare risultati inattesi dovuti ad un'espansione non prevista

```
[andrea@praha Data]$ cat Fasta/* | grep -E A{2,3}
grep: A3: No such file or directory
```

- La shell espande l'espressione in:

```
[andrea@praha Data]$ cat Fasta/* | grep -E A2 A3
grep: A3: No such file or directory
```

- L'espressione va quindi protetta

```
[andrea@praha Data]$ cat Fasta/* | grep -E "A{2,3}"
```

```
MADAAVHGHG
AAILGDEYLW
AAVILESIFL
GSAAEVMKKY
AIGPAA SLVV
```

Espressioni regolari vs Wildcards

- Le wildcards permettono di costruire semplici espressioni regolari per l'interprete di comandi
- Le espressioni regolari usate da `grep` sono più complesse
- Nonostante vari aspetti in comune, la sintassi dei due tipi di espressioni differisce

Esempio

- Elenca tutti i file che cominciano per R

```
[andrea@praha Data]$ ls -l R*  
-r--r--r-- 1 andrea andrea 50 2009-01-14 17:55 README  
drwxr-xr-x 4 andrea andrea 4096 2008-12-22 13:57 Results
```

- Seleziona le righe che contengono R zero o più volte

```
[andrea@praha Data]$ ls -l | grep 'R*'  
total 28  
drwxr-xr-x 2 andrea andrea 4096 2008-12-22 13:06 Fasta  
drwxr-xr-x 2 andrea andrea 4096 2009-01-14 17:37 Labels  
-rw-r--r-- 1 andrea andrea 49 2009-01-14 18:30 ls.out  
-r--r--r-- 1 andrea andrea 50 2009-01-14 17:55 README  
drwxr-xr-x 4 andrea andrea 4096 2008-12-22 13:57 Results  
drwxr-xr-x 4 andrea andrea 4096 2009-01-19 14:37 tmp2  
-rw-r--r-- 1 andrea andrea 797 2009-01-14 19:01 uniq.txt
```

Esempio

- Elenca tutti i file che cominciano per 12 oppure 13

```
[andrea@praha TmpDownloads]$ ls 1{2,3}*  
12.1.1.135.3092.pdf 1328439704253438_article.pdf
```

- Seleziona le righe che contengono 1 ripetuto due o tre volte

```
[andrea@praha TmpDownloads]$ ls | grep -E "1{2,3}"  
10.1.1.74.5119.pdf  
1128439704253438_article.pdf
```

Nota

- Gli operatori di ripetizione permettono di espandere la *sottostringa* che deve soddisfare l'espressione
- Il numero massimo di ripetizioni vincola la sottostringa che soddisferà l'espressione, NON la riga intera.
- Se una riga contiene una sottostringa che soddisfa l'espressione, `grep` la restituirà in uscita, anche se tale sottostringa è seguita da un'ulteriore ripetizione oltre al numero consentito dall'espressione
- Vedremo che questo farà differenza quando si *concatenano* espressioni

Esempio

```
[andrea@praha Data]$ cat Fasta/* | grep -E "A?" \  
> | grep -E "A{2}"
```

MADAAVHG**H**G

AAILGDEYLW

AAVILESIFL

GS**A**AEVMKKY

AIGPAASLVV

GTN**A**AMRKAF

Concatenazione

- Le espressioni regolari possono essere *concatenate* per creare espressioni più complesse.
- L'espressione risultante sarà soddisfatta se si riesce a trovare una sequenza di stringhe contigue ciascuna delle quali soddisfa la corrispondente espressione
- Ad esempio, una parola è una semplice concatenazione di espressioni che consistono in singoli caratteri
- Quando una espressione contiene operatori di ripetizione, le sottostringhe compatibili con tali operatori vengono provate l'una dopo l'altra, verificando se ciò che rimane dopo la sottostringa è compatibile con le espressioni successive

Concatenazione: esempi

```
[andrea@praha Data]$ cat Fasta/* | grep -E "A?Q"
```

```
MVLSEGEWQL  
MQKGNFRNQR  
AQCEATIESN  
GERRRSQLDR  
MALTNAQILA  
QTDVIAQRKA  
MHPRFQTAF  
QNEGHECQCQ
```

```
[andrea@praha Data]$ cat Fasta/* | grep -E "TS+K"
```

```
MAKKTSSKGK
```

```
[andrea@praha Data]$ cat Fasta/* | grep -E "S.*E{2,3}V"
```

```
MTGMSREEVE  
MSSSEEVSWI
```

Alternative

- E' possibile specificare più espressioni regolari *alternative*, separandole con |
- La riga sarà soddisfatta se una qualsiasi delle espressioni alternative è soddisfatta
- Alternative e concatenazione di espressioni possono essere combinate a piacere

Alternative: esempi

```
[andrea@praha Data]$ cat Fasta/* \  
> | grep -E "G.*D{2,}|^E.*F"
```

```
ETFEIPESVT
```

```
EEEFQFLRCQ
```

```
GPQTDDPRNK
```

```
[andrea@praha Data]$ cat Fasta/* | \  
> grep -E "EI+.*E$|H[CDE]{2}"
```

```
QNEGHECQCQ
```

```
MELTKEIISE
```

```
LDAFQEIPLE
```

```
GSHMRVYHEC
```

Precedenza

- Quando si combinano più espressioni regolari, il modo con cui si interpreta l'espressione risultante dipende dalle *regole di precedenza*
- Nelle operazioni aritmetiche, la precedenza tra operatori ci dice ad esempio che

$$a + b \times c = a + (b \times c)$$

- Nelle espressioni regolari, l'ordine di precedenza è :
 - 1 ripetizione
 - 2 concatenazione
 - 3 alternativa

Precedenza: esempi

"A?Q" = "(A?)Q"

"TS+K" = "T(S+)K"

"S.*E{2,3}V" = "S(.*) (E{2,3})V"

"G.*D{2,}|^E.*F" = "(G(.*) (D{2,})) | (^E(.*)F)"

"EI+.*E\$|H[CDE]{2}" = "(E(I+) (.*)E\$) | (H([CDE]{2}))"

Precedenza

- Come per le espressioni algebriche, è possibile stabilire una precedenza diversa racchiudendo le espressioni tra parentesi tonde
- Tale possibilità è utile per forzare precedenze diverse da quelle di default, tipo:
 - applicare un operatore di ripetizione ad una espressione complessa
 - far precedere o seguire un'alternativa da altre espressioni

Precedenze: esempi

```
[andrea@praha Data]$ cat Fasta/* | grep -E "E(I+.)*E$"
```

```
MEFDYVICEE
```

```
MELTKEIISE
```

```
MLMREVTKEE
```

```
MDLSALRVEE
```

```
[andrea@praha Data]$ cat Fasta/* \  
> | grep -E "D(K+E|[CDE]{2})*T"
```

```
MDKDCEMKRT
```

```
PKYTIVDKET
```