

CORSO DI METODI MOLECOLARI E BIOINFORMATICA

LM Biologia Evoluzionistica, Università di Padova

Dr. Enrico Gaffo, Dr. Andrea Binatti

Prof. Stefania Bortoluzzi

Esercitazione 4

Padova, 10 novembre 2021

Obiettivi dell'esercitazione

- Imparare ad usare ulteriori comandi della shell per maneggiare file di testo
- Conoscere il formato file GTF
- Imparare ad usare le espressioni regolari per ricerca di pattern

Sort: riordinare le righe di un file

Il comando `sort` ci permette di ordinare le righe di un file alfanumericamente

Input	Output
lineB	lineA
lineC	lineB
lineA	lineC

Sort: riordinare le righe di un file

Il comando sort ha alcune utili opzioni:

-k: sceglie la chiave di ordinamento

Input	sort -k 1
tizio 5	caio 42
caio 42	sem 37
sem 37	tizio 5

Input	sort -k 2
tizio 5	sem 37
caio 42	caio 42
sem 37	tizio 5

Sort: riordinare le righe di un file

Il comando sort ha alcune utili opzioni:

-k: sceglie la chiave di ordinamento

Input	sort -k 2
tizio 5	sem 37
caio 42	caio 42
sem 37	tizio 5

Input	sort -k 2n
tizio 5	tizio 5
caio 42	sem 37
sem 37	caio 42

Il formato file GTF per le annotazioni geniche

Il formato GTF (General Transfer Format) è un file di testo usato per rappresentare le annotazioni di una regione genomica.

Consiste di una linea per feature (gene, trascritto, esone, ...), ognuna formata da 9 colonne di dati.

Per una descrizione dettagliata del formato GTF consultare la seguente pagina web di Ensembl:

<http://m.ensembl.org/info/website/upload/gff.html>

Dare un'occhiata al file con il comando less:

```
$ less sample.gtf
```

Sort: esercizio

Ordinare il file GTF per la posizione di inizio gene (colonna 4) prima senza e poi con l'opzione -n.

```
$ sort -k 4 sample.gtf | less
```

```
$ sort -k 4n sample.gtf | less
```

Uniq: restituisci linee univoche

Il comando `uniq` mostra una riga soltanto se è diversa da quella precedente

Input	Output
lineA	lineA
lineB	lineB
lineB	lineC
lineC	
lineC	

Spesso viene usato su input ordinato

Input	Output
lineB	lineB
lineA	lineA
lineB	lineB
lineC	lineC
lineC	

Effetto su input non ordinato

Uniq

Generiamo un file con righe duplicate una dopo l'altra:

```
$ cat in.txt in.txt | sort > duplicated.txt
```

Diamo un'occhiata al file generato:

```
$ less duplicated.txt
```

Usiamo `uniq` per avere una lista univoca delle righe nel file `duplicated.txt`:

```
$ uniq duplicated.txt
```

Confrontare file: diff

Usando il comando `cp` creiamo una copia del file `paradise.txt` e lo salviamo in `jungle.txt`:

```
$ cp paradise.txt jungle.txt
```

Il comando `diff` permette di confrontare il contenuto di due file riga per riga. Usando il comando `diff` possiamo vedere che i due file non hanno differenze:

```
$ diff paradise.txt jungle.txt
```

Confrontare file: diff

Usando un editor di testo (ad es. gedit) modificare il file jungle.txt:

```
$ gedit jungle.txt
```

Welcome to the jungle we got fun and games

We got everything you want honey, we know the names

Take me home (oh won't you please take me home)

Ora usiamo di nuovo diff per vedere se ci sono differenze nei due file:

```
$ diff paradise.txt jungle.txt
```

Comm

Il comando `comm` seleziona o scarta le righe che sono in comune tra due file. Come argomenti vuole due file e come output produce 3 colonne:

- la prima colonna contiene le righe esclusive del primo file
- la seconda colonna contiene le righe esclusive del secondo file
- la terza colonna le righe in comune

```
$ comm a.txt b.txt
```

L'output di questo comando non corrisponde alla descrizione del comportamento caratteristico del comando `comm`, perchè `comm` assume che l'input sia ordinato.

Comm

Se ora ordiniamo i due file con sort:

```
$ sort a.txt > sorted_a.txt
```

```
$ sort b.txt > sorted_b.txt
```

e poi usiamo il comando comm:

```
$ comm sorted_a.txt sorted_b.txt
```

vediamo l'output corretto del comando.

Comm

Per ottenere solo la colonna con le righe in comune tra i due file, si usano le opzioni -1 e -2:

```
$ comm -1 -2 sorted_a.txt sorted_b.txt
```

Espressioni regolari

Un'espressione regolare (in lingua inglese regular expression o, in forma abbreviata, *regexp*, *regex* o *RE*) è una **sequenza di simboli** (quindi una stringa) che identifica un insieme di stringhe.

Programmi come **grep** interpretano le RE e ci permettono di lavorare con esse.

Abbiamo già visto un esempio di utilizzo di espressione regolare in cui cercavamo le righe contenenti il carattere ">":

```
$ grep ">" all.fasta
```

Similmente, potremmo cercare le stringhe contenenti (almeno) una adenina:

```
$ grep "A" all.fasta
```

Oppure, le righe contenenti una adenina seguita da una citosina:

```
$ grep "AC" all.fasta
```

etc.

Meta-caratteri

Vari caratteri hanno significati speciali (**meta-caratteri**).

Grazie ai meta-caratteri e loro combinazioni possiamo generare RE complesse.

Alcuni meta-caratteri fondamentali:

Meta carattere	Significato	Meta carattere	Significato
.	un qualsiasi carattere (wildcard)	*	zero o più
^	inizio stringa	+	almeno uno
\$	fine stringa	?	zero o uno
\<	inizio parola	{m}	ripeti m volte
\>	fine parola	{m,n}	ripeti tra m e n volte
\	escape: modifica l'interpretazione del carattere	[]	insieme di caratteri da trovare
	OR logico: match dell'espressione a sx o a dx	[^]	insieme di caratteri da NON trovare
()	subespressioni	[A-Z]	classi di caratteri (es: lettere maiuscole)

Esempio di utilizzo dei meta-caratteri: wildcard

Meta carattere	Significato
.	un qualsiasi carattere
^	inizio stringa
\$	fine stringa
\<	inizio parola
\>	fine parola
\	escape: modifica l'interpretazione del carattere
	OR logico: match dell'espressione a sx o a dx
()	subespressioni

Considerando solo i caratteri che rappresentano nucleotidi {A,C,G,T}, l'espressione regolare

“C.G”

starà ad indicare ognuna delle seguenti stringhe

- CAG
- CCG
- CGG
- CTG

Esempio di utilizzo dei meta-caratteri: delimitatori

Meta carattere	Significato
.	un qualsiasi carattere
^	inizio stringa
\$	fine stringa
\<	inizio parola
\>	fine parola
\	escape: modifica l'interpretazione del carattere
	OR logico: match dell'espressione a sx o a dx

“^C”

starà ad indicare ad es:

- **C**GCTGCGCAAGCG
- **C**CC
- **C**CAAGCG
- **C**

Per cercare gli header FASTA è più corretto

```
$ grep "^>" all.fasta
```

> deve essere il primo carattere della stringa

Esempio di utilizzo dei meta-caratteri: delimitatori

Meta carattere	Significato
.	un qualsiasi carattere
^	inizio stringa
\$	fine stringa
\<	inizio parola
\>	fine parola
\	escape: modifica l'interpretazione del carattere
	OR logico: match dell'espressione a sx o a dx

“G\$”

starà ad indicare ad es:

- CGCTGCGCAAGC**G**
- C**G**
- CG**G**
- CGCTC**G**

Cerchiamo le righe che terminano con G

```
$ grep "G$" all.fasta
```

G deve essere l'ultimo carattere della stringa

Esempio di utilizzo dei meta-caratteri: delimitatori

Meta carattere	Significato
.	un qualsiasi carattere
^	inizio stringa
\$	fine stringa
\<	inizio parola
\>	fine parola
\	escape: modifica l'interpretazione del carattere
	OR logico: match dell'espressione a sx o a dx

Cerchiamo le parole che iniziano con CA

```
$ echo "CACACTCAAGTCGTA" | grep "\<CA"
```

```
$ echo "CA CACTCAAGTCGTA" | grep "\<CA"
```

Gli spazi, inizio e fine riga delimitano le parole

- \<C → CAC ACCT CCTAGC CC
- C\> → CAC ACCT CCTAGC CC

Cerchiamo le parole che finiscono con TA

```
$ echo "CACACTCAAGTCGTA" | grep "TA\>"
```

```
$ echo "CA CACTCAAGTCGTA" | grep "TA\>"
```

Esempio di utilizzo dei meta-caratteri: BRE, ERE

Meta carattere	Significato
.	un qualsiasi carattere
^	inizio stringa
\$	fine stringa
\<	inizio parola
\>	fine parola
\	escape: modifica l'interpretazione del carattere
	OR logico: match dell'espressione a sx o a dx

Cerchiamo le righe che iniziano con C o che terminano con G:

```
$ grep -E "^C|G$" all.fasta
```

```
$ grep "^C\|G$" all.fasta
```

"^C|G\$"

starà ad indicare ad es:

- GCTGCGCAAGC**G**
- **C**GA
- **C**G**G**
- GCTC**G**

Basic RE (**BRE**): è il default di grep, interpreta "|" come il carattere "barra"

Extended RE (**ERE**): interpreta "|" come un meta-carattere (grep -E)

Cerchiamo le righe che iniziano con C seguito da una barra e da una G:

```
$ echo "C|GCC" | grep "^C|G"
```

Esempio di utilizzo dei meta-caratteri: subespressioni

Meta carattere	Significato
.	un qualsiasi carattere
^	inizio stringa
\$	fine stringa
\<	inizio parola
\>	fine parola
\	escape: modifica l'interpretazione del carattere
	OR logico: match dell'espressione a sx o a dx
(...)	subespressioni: tra parentesi tonde

Le parentesi tonde “(...)” servono a delimitare un pattern (complesso)

$(CG|AA)T \rightarrow CAGCTCAAGT\text{CGTACAATG}$

che è diverso da

$CG|AAT \rightarrow CAGCTCAAGT\text{CGTACAATG}$

CG oppure AA, seguito da una T

```
$ echo "CAGCTCAAGTCGTACAATG" | grep -E "(CG|AA)T" oppure $ grep "\ (CG\|AA\)T"
```

Esempio di utilizzo dei meta-caratteri: quantificatori

Meta carattere	Significato
*	zero o più (del carattere precedente)
+	almeno uno (del carattere precedente)
?	zero o uno (del carattere precedente)
{m}	ripeti esattamente m volte
{m,n}	ripeti tra m e n volte
{m,}	ripeti almeno m volte
{,n}	ripeti al massimo n volte

- $CA^*G \rightarrow \underline{CAGCT} \underline{CAAGT} \underline{CGTA}$
- $CA^+G \rightarrow \underline{CAGCT} \underline{CAAGT} CGTA$
- $CA?G \rightarrow \underline{CAGCT} CAAGT \underline{CGTA}$

```
$ echo "CAGCTCAAGTCGTA" | grep "CA*G"
```

```
$ echo "CAGCTCAAGTCGTA" | grep -E "CA+G" oppure $ grep "CA\+G"
```

```
$ echo "CAGCTCAAGTCGTA" | grep -E "CA?G" oppure $ grep "CA\?G"
```

Esempio di utilizzo dei meta-caratteri: quantificatori

Meta carattere	Significato
*	zero o più (del carattere precedente)
+	almeno uno (del carattere precedente)
?	zero o uno (del carattere precedente)
{m}	ripeti esattamente m volte
{m,n}	ripeti tra m e n volte
{m,}	ripeti almeno m volte
{,n}	ripeti al massimo n volte

CA{2}G → CAGCTCAAGTCGTACAAAG

CA{1,2}G → CAGCTCAAGTCGTACAAAG

CA{2,}G → CAGCTCAAGTCGTACAAAG

CA{,3}G → CAGCTCAAGTCGTACAAAG

```
$ echo "CAGCTCAAGTCGTACAAAG" | grep -E "CA{2}G" oppure $ grep "CA\{2\}G"
```

```
$ echo "CAGCTCAAGTCGTACAAAG" | grep -E "CA{1,2}G" oppure $ grep "CA\{1,2\}G"
```

```
$ echo "CAGCTCAAGTCGTACAAAG" | grep -E "CA{2,}G" oppure $ grep "CA\{2,\}G"
```

```
$ echo "CAGCTCAAGTCGTACAAAG" | grep -E "CA{,3}G" oppure $ grep "CA\{,3\}G"
```


Esempio di utilizzo dei meta-caratteri: gruppi di caratteri

Meta carattere	Significato
*	zero o più
+	almeno uno
?	zero o uno
{m}	ripeti m volte
{m,n}	ripeti tra m e n volte
[]	insieme di caratteri da trovare
[^]	insieme di caratteri da NON trovare
[A-Z]	classi di caratteri (es: lettere maiuscole)

[...] trova un singolo carattere contenuto nelle parentesi

- C[AG]T → ATCATTCTTACGTGCCTA

[^...] trova un singolo carattere NON contenuto nelle parentesi

- C[^AG]T → ATCATTCTTACGTGCCTA

N.B: ^ (inizio stringa) e [^...] hanno significati diversi (il contesto è diverso)

Esempio di utilizzo dei meta-caratteri: range di caratteri

Meta carattere	Significato
*	zero o più
+	almeno uno
?	zero o uno
{m}	ripeti m volte
{m,n}	ripeti tra m e n volte
[]	insieme di caratteri da trovare
[^]	insieme di caratteri da NON trovare
[A-Z]	classi di caratteri (es: lettere maiuscole)

[0-9] trova un singola cifra

- C[0-9]T → ATCATTCTTACGTGCCTAC3TA

[A-Z] trova un singola lettera maiuscola
nel range dalla A alla Z

- C[A-Z]T → ATCATTCITACGTGCTA

[A-C] trova un singola lettera maiuscola
nel range **dalla A alla C**

- C[A-C]T → ATCATTCITACGTGCTA

[a-z] → ATCaTTCTTACGTGCcTAC3TA

[A-Za-z] → ATCaTTCTTACGTGCcTAC3TA

[A-Za-z0-9] → ATCaTTCTTACGTGCcTAC3TA

Combinazioni di regole



Pattern	Matches
[AT][CG]	ATT CAGTCGCATG CTACGT CGTAG CGTA
A[CG]*T	ATT CAGTCGC AT GCT ACGTCGTAGCGTA
A[CG]{2}T	ATTCAGTCGCATGCT ACGT CGTAGCGTA
A[CG]{2,3}T	ATTCAGTCGCATGCT ACGT CGT AGCGTA
A[CG][CG]*T	ATTC AGT CGCATGCT ACGT CGT AGCGTA
\<A[CGT]*\>	AT CAG TC GC ATG CTAC GTC ATGA GTA

Stream editor for filtering and transforming text: sed

- Il programma sed permette di effettuare sostituzioni rimpiazzando un'espressione regolare con altro testo. La sintassi è:

```
sed 's/pattern/sostituzione/' file
```

- Proviamo alcune sostituzioni con sed. Invece che un file diamo l'input dallo stdin stream:

```
$ echo "ciao enrico, ciao!" | sed 's/ciao/addio/'
```

- Per sostituire tutte le occorrenze di “ciao” in una riga usiamo l'opzione g:

```
$ echo "ciao enrico, ciao!" | sed 's/ciao/addio/g'
```

- Per rendere sed case insensitive usiamo l'opzione i (o un'espressione regolare):

```
$ echo "ciao enrico, CIAO!" | sed 's/ciao/addio/gi'
```

Potete provare anche con il file `ciao.txt`

Quanti cromosomi sono presenti nel file sample.gtf?

Sfruttiamo il comando **cut**: “*cut out selected portions of each line of a file*”.

cut ci permette di selezionare soltanto alcune porzioni da ogni riga del file.

Le porzioni (o “colonne”) vengono selezionate con l’opzione **-f** e vengono definite da un carattere delimitatore, che per default è il TAB.

Vediamo l’effetto di **cut** quando selezioniamo il primo campo del GTF, cioè il nome del cromosoma:

```
$ cut -f 1 sample.gtf | less
```

Combinare cut con altri comandi

Ordiniamo i cromosomi e teniamo solo le righe univoche:

```
$ cut -f 1 sample.gtf | sort | uniq
```

Se volessimo contare le righe univoche, cioè sapere quanti cromosomi sono annotati, aggiungiamo il comando per contare le righe

```
$ cut -f 1 sample.gtf | sort | uniq | wc -l
```

Esercizio: quanti cromosomi sono annotati in *C. elegans*?

Link per scaricare le annotazioni geniche di *C. elegans* (repository FTP di Ensembl)

http://ftp.ensembl.org/pub/current_gtf/caenorhabditis_elegans/Caenorhabditis_elegans.WBcel235.104.gtf.gz

Altrimenti, andare su

<https://www.ensembl.org> -> Downloads -> FTP site -> current_gtf/ -> caenorhabditis_elegans/

Decomprimere il file

```
$ gunzip Caenorhabditis_elegans.WBcel235.104.gtf.gz
```

Esercizio: cromosomi annotati in *C. elegans*

Controlliamo il contenuto del GTF

```
$ less -S Caenorhabditis_elegans.WBcel235.104.gtf
```

Sono presenti delle righe di intestazione che cominciano con il carattere #.

Per “rimuovere” l’intestazione del GTF sfruttiamo l’opzione -v di grep, che inverte la selezione del pattern, cioè esclude le righe contenenti il pattern invece di selezionarle

```
$ grep -v "^#" Caenorhabditis_elegans.WBcel235.104.gtf | less -S
```

Ora selezioniamo solo la colonna dei cromosomi:

```
$ grep -v "^#" Caenorhabditis_elegans.WBcel235.104.gtf | cut -f 1 | less
```

Per contare le righe univoche usiamo l’abbinata di comandi sort|uniq|wc -l:

```
$ grep -v "^#" Caenorhabditis_elegans.WBcel235.104.gtf | cut -f 1 | sort |  
uniq | wc -l
```


Esercizi

1. “Quanti geni sono annotati nel cromosoma V di *C. elegans*?”

Suggerimenti:

- selezionare solo le righe che identificano i geni, tralasciando quelle delle altre features (transcritti, CDS, esoni, ..)
- selezionare le righe del cromosoma V

2. Have fun with RE !

- Eseguire lo script `regex_test` per allenarsi con le espressioni regolari:

```
$ ./regex_test
```

3. “Quante adenine ci sono nel cromosoma 13 di *Ciona intestinalis*?”

http://ftp.ensembl.org/pub/current_fasta/ciona_intestinalis/dna/Ciona_intestinalis.KH.dna.chromosome.13.fa.gz

suggerimento: usare l'opzione `-z` di `sed`

4. Caricare su UCSC Genome Browser la track `sample.gtf` (genoma *C. elegans*)

suggerimenti: `sample.gtf` ha nome cromosomi formato Ensembl -> convertire in formato UCSC

Soluzione esercizio 3



Contare quante adenine ci sono nel cromosoma 13 di *Ciona intestinalis*

```
$ grep -v "^>" Ciona_intestinalis.KH.dna.chromosome.13.fa | sed -z 's/[^A]//gi' | wc -m
```

```
560413
```

Soluzione esercizio 4



Prependere ai nomi dei cromosomi il prefisso “chr”

```
$ sed 's^_chr_' sample.gtf > chrSample.gtf
```

Aggiungere la custom track chrSample.gtf nel Genome Browser