Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2023-2024

Corso di Laurea in Matematica

Sort, tr e sed. Costrutti if e for.

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

1. Comandi su file di testo
   - Comandi su testo
2. Altri comandi
   - Comandi per elaborare contenuti di testo
   - Esempi di costrutti

Altri comandi

Comandi su testo

Comando `sort`

- Comando `sort [-r] [-f] [-n] [-u] [-t sep] [-k POS1[,POS2]] [file...]`
  - ordinamento per righe dei file in input (o da tastiera)
  - ordinamento lessicografico: le cifre vengono prima delle minuscole, che vengono prima delle maiuscole (ma con `-f` non fa differenza tra maiuscole e minuscole); una parola p che è prefisso di una parola q è minore di q
  - se le righe contengono numeri e si vuole che `sort` li interpreti come tali, allora occorre usare l'opzione `-n`
  - ordinamento dal più piccolo al più grande; con `-r` dal più grande al più piccolo
  - con `-u`, stampa una sola volta le righe uguali

Comando `sort`

- Comando `sort [-r] [-f] [-n] [-u] [-t sep] [-k POS1[,POS2]] [file...]`
  - con `-k`, tokenizza ogni riga secondo gli spazi o secondo il separatore specificato da `-t`), e poi ordina rispetto ai contenuti che vanno dal campo numero `POS1` al campo numero `POS2` (se `POS2` non è dato, allora va fino alla fine della riga)
  - **Esercizio** Usando il file di testo già considerato negli esercizi precedenti, scrivere uno script che:
    - utilizza `sort` per ordinare tutti i versi e manda il risultato nel file `righe-ordinate`
    - utilizza `sort` ordina le parole di ogni riga e manda il risultato nel file `parole-ordinate`

Altri comandi

Comandi per elaborare contenuti di testo

Comando `tr`

- Comando `tr [-d] [-c] [-t] stringa1 [stringa2]`
  - si usa per tradurre, eliminare e sostituire insiemi di caratteri
  - legge sempre da tastiera e scrive sempre su schermo, ma si possono usare le redirezioni
  - `stringa1` e `stringa2` sono successioni di caratteri,
  - in generale, `tr` rimpiazza ogni occorrenza del carattere *i*-esimo di `stringa1` con l'*i*-esimo di `stringa2`
  - quindi, `stringa1` e `stringa2` devono essere lunghe uguali
  - con `-d` cancella tutti i caratteri in `stringa1` e `stringa2` non va data
  - con `-c`, si prendono i caratteri *non* presenti in `stringa1`; usata soprattutto in combinazione con `-d`

Comando `tr`

- Comando `tr [-d] [-c] [-t] stringa1 [stringa2]`
  - Preparare un **file** con una canzone (esem. Tanti auguri a te) o una poesia
  - **Esercizi**
    - Provare `tr abc ABC` su input da tastiera
    - Provare `tr [a-z] [A-Z]` su input da tastiera
    - Provare `tr [a-z] [A-Z]` su **file** usando echo o cat e poi | oppure <
    - Provare `tr [:lower:] [:upper:]` con input da tastiera o da file
    - Provare `tr [:space:] "\t"`
    - Provare `tr -cd [:digit:]` usando un input con numeri
    - Provare `tr -s '\n' ' '` sul **file**
  - **Esercizio** Scrivere uno script che usa il testo scritto in **file** e applica `tr` per sostituire caratteri in `stringa1` con caratteri in `stringa2` e manda il risultato su un file

Comando sed

- Comando `sed [-e script] [-f file_script] [-r] [-s] [files...]`
  - `sed` sta per Stream EDitor ed è un editor di testo non interattivo
  - Riceve un testo come input, o dallo stdin o da un file, esegue alcune operazioni sulle righe specificate, una alla volta, e invia il risultato allo stdout o in un file
  - `sed` procede sull'input riga per riga: ogni riga del file viene letta individualmente, elaborata e poi riprodotta
  - La funzione più importante di `sed` è quella di **cercare determinate stringhe di caratteri** nel file e poi **sostituirle con altri caratteri**

Comando sed

- Comando `sed [-e script] [-f file_script] [-r] [-s] [files...]`
  - L'opzione `-e` indica che la stringa che segue è un'istruzione, o una serie di istruzioni (se viene passata un'unica istruzione `-e` può essere omesso)
  - L'opzione `-f` indica se il comando debba fare riferimento a un file aggiuntivo
  - L'opzione `-n` serve a visualizzare solo le righe interessate dal comando ed è molto importante se si applica su file molto grandi
  - Si può specificare una linea (qui detta indirizzo) o un intervallo di (indirizzi di) linee su cui eseguire l'azione, altrimenti l'azione viene applicata a tutte le linee in input, cioè:
    - n indica la riga n-esima
    - n, m indica le righe da n a m (incluse)
    - $ vale vero all'ultima riga

Comando sed

Alcune azioni da usare con il comando `sed`

- `a` - *append*: aggiunge una o più righe alle righe selezionate
- `c` - *change*: sostituisce le righe selezionate con un nuovo contenuto
- `d` - *delete*: elimina le righe selezionate
- `i` - *insert*: aggiunge una o più righe prima della riga indicata
- `p` - *print*: mostra le righe selezionate
- `q` - *quit*: termina il comando SED.
- `s` - *substitute*: sostituisce una parola con un'altra
- `y` - *yank*: sostituisce un carattere predefinito con un altro (anche più caratteri)
- `w` - *write*: scrive righe in un file di testo
- `!` - *negation*: applica il comando sulle righe che non corrispondono all'inserimento

Comando sed

Sintassi di sed

- ([indirizzo])/p (print) Visualizza [l'indirizzo specificato]
- ([indirizzo])/d (delete) Cancella [l'indirizzo specificato]
- s/stringa1/stringa2/ (substitute) Sostituisce in ogni riga la prima occorrenza della stringa stringa1 con la stringa stringa2
- ([indirizzo])/s/stringa1/stringa2/ Sostituisce, in tutte le righe specificate in indirizzo, la **prima** occorrenza di stringa1 con stringa2
- g *global* Agisce su tutte le occorrenze di ogni riga di input
- NB Se l'operatore g *global* non è accodato al comando substitute, la sostituzione agisce solo sulla prima verifica d'occorrenza di ogni riga.
- ([indirizzo])/y/stringa1/stringa2/ (transform) sostituisce i singoli caratteri in stringa1 con i singoli caratteri in stringa2, nelle righe specificate da indirizzo (equivalente di `tr`)

Comando sed

Esempi

- `sed '4,$d' filename`
  - stampa a video soltanto le prime 3 righe del file `filename`, `d` è il comando di cancellazione che elimina dall'output tutte le righe a partire dalla quarta ($ si usa per l'ultima riga del file)
- `sed 3q filename`
  - stesso effetto del precedente comando: in questo caso sed esce dopo aver elaborato la terza riga (3q)
- `sed 22,29d filename`
  - stampa tutte le righe eccetto le righe da 22 a 29 usando l'opzione `d`
- `sed -n 22,29p filename`
  - stampa le righe da 22 a 29 usando l'opzione `p`

Comando sed

Esempi

- `sed /stringa/y/char1char2/new1new2/ filename`
  - sostituisce in tutte le righe che contengono la stringa `stringa` il carattere char1 con il carattere new1 ed il carattere char2 con il carattere new2 (`y` sta per transform)
- `sed '/stringa/!y/char1char2/new1new2/' filename`
  - invece sostituisce in tutte le stringhe che non contengono `stringa` ma deve usare ' per impedire di interpretare !
- `sed s/stringa1/stringa2/` Sostituisce in ogni riga la prima occorrenza della stringa `stringa1` con la stringa `stringa2`
- `sed s/stringa1/stringa2/k` Sostituisce in ogni riga la *k*-ima occorrenza di `stringa1` con `stringa2`
  `sed s/stringa1/stringa2/g` Agisce su tutte le occorrenze di ogni riga di input controllata, `/g` sta per *globally*
- `sed 'k s/stringa1/stringa2'` Sostituisce a partire da riga *k*

Comando sed

- **Esercizio** Provare i seguenti comandi su un file in input che contenga aaa e ao

```
sed s/a/A/g
sed s/aaa/A/g
sed s/'aaa'/A/g
sed s/ao/A/g
sed s/ao/AO/g
```

- **Esercizio** Scrivere uno script che applichi le diverse opzioni di `sed` su righe opportunamente selezionatee di un file in input

Esempi di costrutti

IF e FOR

Struttura di controllo condizionale if

- Una struttura di controllo condizionale consente di valutare un'espressione logica e, sulla base del suo valore (vero o falso), eseguire determinate istruzioni
- La forma più semplice di una struttura di controllo è:

  se condizione **allora**
  istruzioni
  fine-condizione

- Nel linguaggio Bash questa struttura viene implementata dalle istruzioni `if` ... `then` ... `fi` secondo la seguente sintassi:

  if lista comandi **then**
  lista comandi
  fi

Struttura di controllo condizionale if

- Esempio di **if**

```
if [[ -e originale.txt ]] ; then
  cp originale.txt copia.txt
  echo "Copia riuscita."
fi
```

  - la condizione -e risulta vera se il file che la segue esiste

- Esempio di **if-then-else**

```
if [[ -e originale.txt ]] ; then
  cp originale.txt copia.txt
  echo "Copia riuscita."
else
  echo "Il file non esiste."
fi
```

  - notare che gli if si possono annidare oppure si può usare `eli`

Struttura di controllo iterative

- Per eseguire più operazioni con uno script di poche righe, si usano strutture di controllo iterative
- I costrutti iterativi permettono di ripetere un certo blocco di comandi più volte, fino a quando non risulta verificata l'espressione logica usata come istruzione di controllo
- Bash mette a disposizione tre istruzioni principali per la realizzazione di iterazioni (cicli): le istruzioni **for**, **while** e **until**
- Per tutte e tre, all'inizio del blocco di istruzioni da ripetere, viene valutata una condizione logica

Struttura iterativa for

- L'istruzione `for` consente di implementare una struttura di controllo usando una variabile di controllo in una lista di valori o di un intervallo numerico e ha la seguente struttura:

  for variabile **in** lista-di-valori
  do
  istruzioni
  done

- L'istruzione `for` si può anche usando contatore e in tal caso ha una struttura simile al C in cui si hanno tre parametri tra doppie parentesi tonde separati da un punto e virgola del tipo:

  for ((espr1 ; espr2 ; espr3))
  do
  istruzioni
  done

Esempi

- Esempio di **for**

```
for i in 1 2 3 4 5
do
  echo "Numero $i"
done
```

  - La variabile *i* assume ciascuno dei valori indicati dopo la parola chiave `in` nello stesso ordine
  - Per scorrere una lista di numeri interi è possibile anche usare la scorciatoia 1..5

- Esempio di **for**

```
for i in 1 2 topolino pippo 3 pluto
do
  echo "i ha valore $i"
done
```

  - i valori assunti dalla variabile i possono essere di qualsiasi tipo

Esempi

- Esempio di **for**

```
for (( i = 0 ; i <= 20 ; i += 2 )) ; do
  echo "i ha valore $i"
done
```

  - per definire il ciclo `for` si possono usare le espressioni aritmetiche, separate da ;

- Esempio di **for**

```
for (( i=1 , j=1 ; i<=34 ; i+=j, j+=i )) ; do
  echo -n "$i" "$j "
done
echo
```

  - è possibile definire cicli su sequenze di numeri arbitrarie: nell'esempio la sequenza di valori stampata corrisponde ai primi 10 elementi della successione di Fibonacci