Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2024/2025

Corso di Laurea in Matematica

Bash script e comandi

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

1. Bash script
   - Parentesi
   - Stream e redirezioni
2. Comandi
   - Comandi su testo

Shell Programming

Parentesi

Parentesi graffe

- Qualsiasi comando o sequenza di comandi può essere inserito tra **parentesi graffe** (*group command*)
- Quel comando o sequenza di comandi viene eseguito nella bash corrente, cioè dalla stessa bash in cui viene eseguito lo script
  - I comandi devono essere separati da un punto e virgola
  - Anche l'ultima istruzione della lista deve essere seguita da un punto e virgola
  - Tra le parentesi graffe e la prima e l'ultima istruzione deve essere presente uno spazio: infatti le parentesi graffe sono parole riservate del linguaggio Bash e devono essere separate da uno spazio dalle parole chiave adiacenti

Parentesi graffe

- Se il comando è unico, mettere le parentesi graffe è inutile (anzi, può complicare le cose)
- Utilità delle parentesi graffe:
  - raggruppare più comandi in **esecuzione condizionale** (vedere sotto), facendo sì che l'effetto sia sulla bash corrente
  - raggruppare tutte le **redirezioni** in una volta sola (lo vedremo)
  - raggruppare tutto l'input/output da mandare in **pipelining** in una volta sola (lo vedremo)

Parentesi tonde

- Qualsiasi comando o sequenza di comandi può essere inserito tra **parentesi tonde**
- Il comando composto viene eseguito in una *sottoshell* della shell in cui viene eseguito,
  - se il comando è unico, allora il nuovo processo è costituito da quel comando
  - se è una lista di comandi (separati ad esempio da `;`), allora il nuovo processo è una bash che esegue quei comandi
  - il comando composto eredita tutte le variabili definite nello script, ma lo scope delle assegnazioni e delle definizioni effettuate nell'ambito del comando composto è limitato al comando stesso
- Le parentesi tonde che delimitano il comando composto non sono delle parole riservate del linguaggio (come le parentesi graffe), per cui non è necessario separarle dalle altre istruzioni con degli spazi

Parentesi tonde

- Utilità delle parentesi tonde:
  - raggruppare più comandi in **esecuzione condizionale** (vedere sotto), facendo sì che non ci sia effetto sulla bash corrente
  - raggruppare tutte le **redirezioni** in una volta sola (vedere sotto)
  - raggruppare tutto l'input/output da mandare in **pipelining** in una volta sola (vedremo dopo cosa si intende per pipelining)
- **Esercizio** provare a dare i comandi `(cd ..)` e `{ cd ..; }`

Parentesi graffe e tonde

Esempio parentesi graffe

- Consideriamo lo script:

```
a=1; b=2
echo "Prima del comando composto:  A = $a, B = $b"
{ b=3; echo "Durante il comando composto:  A = $a, B =
$b"; }
echo "Dopo il comando composto:  A = $a, B = $b"
```

  - Vengono definiti e visualizzati i valori delle variabili `a` e `b`
  - La sequenza di istruzioni nelle parentesi graffe viene eseguita come un singolo comando, nella stessa shell dello script
  - Il valore della variabile `b` viene modificato, mentre il valore di `a`, definita fuori dal comando composto, rimane lo stesso

- L'output è:

```
Prima del comando composto:  A = 1, B = 2
Durante il comando composto:  A = 1, B = 3
Dopo il comando composto:  A = 1, B = 3
```

Parentesi graffe e tonde

Esempio parentesi tonde

- Consideriamo lo script:

```
a=1; b=2
echo "Prima del comando composto:  A = $a, B = $b"
(b=3; echo "Durante il comando composto:  A = $a, B =
$b")
echo "Dopo il comando composto:  A = $a, B = $b"
```

  - Vengono definiti e visualizzati i valori delle variabili `a` e `b`
  - Il comando composto può utilizzare le variabili definite in precedenza e può modificarne il valore, ma le modifiche effettuate **non** sono visibili al termine del comando

- L'output è:

```
Prima del comando composto:  A = 1, B = 2
Durante il comando composto:  A = 1, B = 3
Dopo il comando composto:  A = 1, B = 2
```

Stream e redirezioni

Generalità sui comandi

- Ogni comando genera un processo cui vengono subito associati 3 *stream* o canali di input/output:
  - `stdin` o *standard input*, che di default è la tastiera e ha *file descriptor* 0
  - `stdout` o *standard output* che di default è lo schermo e ha *file descriptor* 1
  - `stderr` o *standard error* che di default è la schermo e ha *file descriptor* 2
  - un file descriptor (o identificativo) è un intero non negativo associato o ad uno stream (come sopra) o ad un file vero e proprio
- Ciascuno degli stream dati sopra può essere **rediretto**

Generalità sui comandi

- Per reindirizzare il canale di input in modo che il comando riceva l'input da un file invece che da tastiera, si usa l'operatore `<`, quindi: `comando < file`
- Per reindirizzare il canale di output in modo che il comando invii l'output ad un file invece che sullo schermo si usa l'operatore `>`, quindi: `comando > file`
- I comandi di redirezione dell'input e dell'output possono essere utilizzati simultaneamente, utilizzando una sintassi del tipo:
  `comando < file input > file output`
- Per redirigere l'output su un file già esistente, senza cancellarlo, ma *appendendo* l'output del comando alla fine del contenuto del file, si usa l'operatore `>>` invece di `>`, quindi:
  `comando >> file`

Generalità sui comandi

- Per redirigere su un file ciò che è stato inviato sul canale standard error è necessario utilizzare l'operatore `2>`, con la stessa sintassi con cui si è utilizzato l'operatore `>`
- È possibile anche redirigere i due canali di output (standard output e standard error) su due file differenti con lo stesso comando, utilizzando entrambi gli operatori di redirezione dell'output `>` e `2>`
- È possibile anche redirigere il canale standard error sullo standard output o viceversa: nel primo caso si userà l'operatore di redirezione `2>&1`, mentre nel secondo caso si userà l'operatore `1>&2`
- Infine, con l'operatore di redirezione `&>` si redirigono entrambi i canali standard output e standard error su un file

Generalità sui comandi

- Le **redirezioni** avvengono sempre *prima* che il comando sia eseguito
- Le **redirezioni** possono trovarsi in *qualsiasi punto* di un comando (anche prima del comando stesso)
- Le redirezioni, se applicate ad un group command (comandi tra graffe) o ad una subshell (comandi tra tonde) hanno effetto su tutti i comandi del gruppo
- Ad esempio, anziché scrivere
  `cmd1 > file; cmd2 >> file`
  si può scrivere
  `{ cmd1; cmd2; } > file`

Generalità sui comandi

- **Esercizio** Mettere in un file l'ouput del comando `ls -l`, poi appendere al file l'output del comando history usando `>` e `>>`
- **Esercizio** Mettere in un file l'ouput del comando `ls -l`, poi appendere al file l'output del comando history usando le parentesi graffe
- **Esercizio** Scrivere uno *script* che esegue i due esercizi precedenti usando due file diversi per l'output

Comandi su testo

Comandi `wc`, `uniq`, `cut`

Comando `wc`

- Comando `wc [-c] [-l] [-m] [-w] [-L] [file...]`
  - stampa statistiche su file di testo: numero di righe, parole, byte
  - l'input può essere da file o da tastiera
  - le opzioni servono a controllare cosa stampare:
    - `-c` numero di bytes
    - `-m` numero di caratteri (diverso dal precedente se ci sono caratteri accentati)
    - `-l` numero di righe
    - `-w` numero di parole
    - `-L` numero caratteri della sola riga più lunga

Comando `wc`

- Comando `wc [-c] [-l] [-m] [-w] [-L] [file...]`
  - **Esercizio**
    - Preparare un file testo in cui è sono scritte alcune righe di una poesia o del testo di una canzone
    - Scrivere uno script che usi il file di testo come input e produca in uscita un file con le seguenti informazioni, specificando cosa si visualizza:
      Titolo della poesia/canzone
      numero di righe #
      numero di parole #
      numero di caratteri del verso più lungo # dove al posto di # è visualizzato il valore ottenuto con il comando `wc`

Comando `uniq`

- Comando `uniq [-u] [-d] [-c] [filein [fileout]]`
  - elimina le righe identiche *consecutive*
  - l'input può essere da file o da tastiera
  - se viene dato il file di input, si può specificare un file di output (altrimenti output a schermo)
  - con `-c`, per ogni riga si specifica quante volte è ripetuta
  - con `-d`, non stampa le righe singole (mai ripetute)
  - con `-u`, stampa solo le righe singole (mai ripetute)
  - **Esercizio**
    - *Preparare un file testo in cui è sono scritte alcune righe scritta una poesia o del testo di una canzone*
    - Scrivere uno script che usi il file di testo come input e produca in uscita le seguenti informazioni, mettendole in un file, specificando cosa si visualizza:
      Titolo della poesia/canzone
      specifica quante volte è ripetuta ogni singola riga
      stampa il testo senza ripetizioni
      stampa le righe ripetute

Comando cut

- Comando `cut [-d sep] [-f fields] [file...]`
  - comando utile per l'elaborazione di stringhe e caratteri
  - l'input può essere da file o da tastiera
  - `cut` prende una *fetta* verticale di un file,cioè stampa solo le colonne o i campi specificati
  - le colonne sono selezionate utilizzando il IFS (Input Field Separator) standard o un delimitatore di campi specificato con `-d`
    - per utilizzare lo spazio come delimitatore, bisogna metterlo tra apici singoli e scrivere: -d ' '
  - `-f` si aspetta una sequenza di range (separati da virgola)
  - **Esercizio** Usando il file di testo già considerato negli esercizi precedenti, scrivere uno script che stampa sia su schermo e su file la seconda parola di ogni verso.
    Ripetere facendo stampare dalla seconda alla quarta parola.