Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2023/2024

Corso di Laurea in Matematica

Bash: comandi e script

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

1. Generalità sui comandi
   - Stream e redirezioni
   - Pipelining

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

Pipelining

Generalità sui comandi

- Finora, abbiamo visto che si possono dare comandi in sequenza nei seguenti modi:
  - con il ; o l'andata a capo (equivalenti)
  - con && e il ||
  - mettendoli dentro parentesi tonde o graffe
- In tutti questi esempi, quello che succede è :
  - che prima si esegue un comando, e poi (eventualmente) un altro
  - input ed output sono scollegati, a meno di redirezioni comuni nel caso di sottoshell o di group command

Generalità sui comandi

- Si possono concatenare tra loro due o più comandi inviando ciò che viene prodotto sul canale di standard output di un comando sul canale di standard input di un altro comando
- Questa operazione di concatenazione di programmi si chiama **pipelining** e si realizza usando il simbolo |
- Quindi il **pipelining** serve a collegare gli `stdout` agli `stdin` per le sequenze non condizionali

Generalità sui comandi

Esempio

- Si può inviare l'output del comando `ls -1` (verificare cosa produce l'opzione `-1`) in input al comando `wc -l` scrivendo:

```
ls -1 | wc -l
```

- Così lo `stdout` del comando a sinistra viene rediretto nello `stdin` del comando a destra `wc` che, con l'opzione `-l`, esegue il conteggio delle righe in input e lo stampa in output

- Con `|&` anzichè `|`, anche lo `stderr` viene rediretto nello `stdin`

- Sulla stessa riga possono essere utilizzati più operatori di pipe per costruire sequenze di operazioni che, prima di produrre l'output finale, passano i dati le une alle altre

Generalità sui comandi

- L'operatore **pipe** può essere applicato ad un *group command*, cioè comandi tra parentesi graffe, o ad una *subshell*, cioè comandi tra parentesi tonde
- Così ha effetto sull'input/output combinato di tutti i comandi del gruppo
- Ad esempio, anziché scrivere

  ```
  { cmd1; cmd2; } > out; cmd3 < out; rm out
  ```

  si può scrivere

  ```
  { cmd1; cmd2; } | cmd3
  ```

Informazioni sui comandi

- Ricapitolando, la shell Bash è in grado di elaborare i seguenti tipi di comandi, sia che essi siano forniti dall'utente sulla linea di comando in modalità interattiva, sia che siano inseriti in uno script:
  1. **comandi semplici**: singoli comandi interni o esterni
  2. **liste di comandi**: sequenze di comandi, concatenati con i connettori ;, &, &&, ||
  3. **comandi in pipeline**: sequenze di comandi concatenati dall'operatore di pipe, cioè "|", che consente di trasformare l'output di un comando nell'input per il comando successivo nella sequenza della pipeline

Informazioni sui comandi

- Inoltre, la shell Bash è in grado di elaborare i seguenti tipi di comandi:
  1. **comandi composti**: comandi più complessi, composti da più istruzioni e da strutture di controllo che formano una struttura logico-algoritmica non solo sequenziale
  2. **funzioni**: sotto-programmi, identificati da un nome, che possono essere richiamati nella shell o in uno script