Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2023/2024

Corso di Laurea in Matematica

Bash: primi comandi

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

Per cominciare

Shell e comandi

Shell e comandi

- La shell opera in modalità interattiva:
  - acquisisce in input ogni singolo comando ed i parametri specificati sulla riga di comando
  - manda in esecuzione il comando
  - visualizza l'output sulla medesima finestra di terminale
- È possibile impartire più comandi sulla stessa riga, separandoli l'uno dall'altro con il ";" (punto e virgola)
- È possibile anche spezzare l'inserimento di un comando su due o più righe, terminando ciascuna riga intermedia con il carattere "\" (backslash)

Shell e comandi

- Nella sintassi del linguaggio Bash alcuni caratteri, presenti in una stringa o come argomento di un comando, assumono un significato speciale e svolgono funzioni ben precise:

  \ (backslash) Precede un altro carattere per comporre una sequenza di escape; alla fine di una riga indica che l'istruzione prosegue alla riga successiva

  ; (punto e virgola) Separa un'istruzione dalla successiva, se più istruzioni sono sulla stessa riga

  $ (dollaro) Precede il nome di una variabile

  ' (apici) Delimitano stringhe di caratteri, senza consentire alla shell di interpretare eventuali variabili

  " (doppi apici) Delimitano stringhe di caratteri, consentendo alla shell di interpretare e sostituire nella stringa i valori di eventuali variabili in essa contenute

  ` (backtick) Delimitano un comando consentendo alla shell di sostituire il comando con l'output prodotto

Variabili

- Come in ogni altro linguaggio di programmazione, anche la Bash possiede il concetto di variabile di memoria
- Le variabili identificano aree della memoria entro cui memorizzare temporaneamente un'informazione numerica o alfanumerica (un numero o una stringa di caratteri)
- Per definire una variabile basta assegnarle un valore, come ad esempio:

```
$ a=Buongiorno
```

Variabili

- Dopo aver assegnato un valore ad una variabile, per fare riferimento ad essa in altri comandi della shell, occorre anteporre al nome della variabile il simbolo $
- Ad esempio, per riferirci al valore della variabile a si deve scrivere $a altrimenti non viene visualizzato il valore:

```
$ echo $a
Buongiorno
```

Se non si usa il $ si visualizza invece:

```
$ echo a
a
```

Variabili

- N.B. Nell'assegnazione di un valore ad una variabile mediante l'operatore "=", il simbolo $ non serve
- Cioè scrivere `$a=5` è sbagliato, mentre `a=5` è corretto
- Se si vuole assegnare alla variabile a lo stesso valore della variabile b si deve scrivere `a=$b`, perchè `a=b` produce l'assegnazione del carattere b alla variabile a

Delimitatori

- Nei linguaggi di scripting, come quello per la Bash, **apici**, **virgolette** e **backtick** hanno significato (e uso) speciale
- Gli **apici** (singoli) sono utilizzati per delimitare stringhe di caratteri: l'interprete Bash non controlla il contenuto della stringa, ma si limita ad usare la sequenza di caratteri delimitata dagli apici
- In questo modo, anche i caratteri che hanno un uso speciale possono far parte della stringa
- L'unico carattere che non si può utilizzare all'interno di una stringa delimitata da apici sono gli apici stessi; per definire una stringa che contiene apici, occorre delimitarla con le virgolette

Delimitatori

- Quando le stringhe sono delimitate dalle **virgolette** (doppi apici), l'interprete Bash risolve il valore di eventuali variabili riportate nella stringa stessa
- Ad esempio, se in una stringa delimitata da virgolette è presente un riferimento ad una variabile - come $a - allora al nome della variabile viene sostituito il suo valore
- In una stringa delimitata da virgolette, per stampare caratteri come i doppi apici o il dollaro, che altrimenti verrebbero interpretati ed usati con la loro funzione, bisogna anteporre il carattere backslash \
- N.B. due backslash di seguito permettono di stampare il carattere backslash

Esempi ed esercizi

- **Esempio 1:**

```
$ nome='Anna'
$ echo 'Ciao $nome'
Ciao $nome
```

- **Esempio 2:**

```
$ echo "Ciao $nome"
Ciao Anna
```

- **Esercizio**

  Usando il comando `echo` farsi stampare:

  `Ciao "$nome", anzi ciao Anna.`

Esempi ed esercizi

- **Esempio 1**

```
$ nome='Anna'
$ echo 'Ciao $nome'
Ciao $nome
```

- **Esempio 2**

```
$ echo "Ciao $nome"
Ciao Anna
```

- **Esercizio**

  Usando il comando `echo` farsi stampare:

  `Ciao "$nome", anzi ciao Anna.`

```
$ echo "Ciao \"\$nome\", anzi ciao $nome."
```

Delimitatori

- Il carattere **backtick** ` ha un comportamento tipico dei linguaggi di scripting, assente nei principali linguaggi di programmazione di alto livello
- Il backtick consente di delimitare una stringa che viene interpretata dalla Bash come un comando da eseguire, restituendo come valore l'output del comando stesso prodotto sul canale standard output

Esempio

- Consideriamo il comando `date` che restituisce in output una stringa con la data corrente
- Possiamo assegnare ad una variabile l'output del comando `date` usando il *backtick* e riutilizzare la variabile in seguito
- Ad esempio, mettiamo il risultato del comando `date` nella variabile data specificando il formato (vedere `man date`):
  `$ data=`date +%d/%m/%Y`` oppure `$ data=`date +%D``
- Utilizziamo poi la variabile con `echo`:
  `$ echo "Oggi è il $data."`
- Come risultato verrà prodotto:
  `Oggi è il 02/10/2023.`

Esempio

- Il *backtick* viene interpretato anche se si trova all'interno di una stringa delimitata da doppi apici
- Ad esempio, il seguente comando produce un risultato analogo a quello precedente:

```
$ echo "Oggi è il `date +%d/%m/%Y`."
Oggi è il 02/10/2023.
```

- Il *backtick* può essere sostituito dalla notazione sintattica $(comando) che ha lo stesso comportamento e produce gli stessi risultati
- L'esempio precedente può essere riscritto come segue:

```
$ echo "Oggi è il $(date +%d/%m/%Y)."
Oggi è il 02/10/2023.
```

Per cominciare

I comandi

Comandi

- I comandi che possiamo utilizzare sono distinti in due insiemi: i **comandi interni**, resi disponibili dall'*interprete*, ed i **comandi esterni**, resi disponibili dal *sistema operativo* in cui viene eseguito l'interprete
- Ad esempio il comando `date` è un programma presente nel set di base delle utility di tutti i sistemi operativi UNIX; al contrario, il comando `echo` è un comando interno della Bash
- Sia il programma `date` (comando esterno) che l'istruzione `echo` (comando interno) possono essere utilizzati nello stesso modo (anche in uno shell script), senza che sia necessario invocare il programma esterno con forme sintattiche particolari.

Comandi

- Ogni comando viene indicato come segue
  - `comando [opzioni] [argom. opz.] argom. obbligat.`
  - tutto ciò che è tra parentesi quadre può essere omesso
  - se ci sono parentesi graffe sugli argomenti, allora ci dev'essere *almeno* un argomento (ma ce ne può essere anche più d'uno)
    - esempio: `cp [-r] [-i] [-a] [-u] {filesorgenti} filedestinazione`

Comandi

- Se ci sono le parentesi quadre e i puntini, allora ci possono essere 0, 1 o più argomenti (eventualmente separati dal carattere indicato)
  - esempio: `ps [opzioni] [pid...]`
  - altro esempio: `chmod mode[, mode...] filename`
  - talvolta, se necessario, potrà essere reso esplicito il numero di argomenti: `ps [opzioni] [pid1 ... pid`*n*`]`

Comandi

- Le opzioni sono tipicamente composte da uno o due *dash* (ovvero, il carattere -) seguiti da alcuni caratteri (senza spazi)
  - solitamente, dopo un dash c'è un solo carattere (versione *vecchia*), dopo 2 dash c'è una parola (versione *moderna*)
  - spesso si possono usare 2 opzioni per la stessa cosa: per esempio `-i` e `--interactive` del comando `cp` sono equivalenti
  - le opzioni sono sempre omissibili
  - le opzioni possono avere o no un argomento
    - esempi **senza** argomento: `-r`, `--recursive`:
    - esempi **con** argomento: `-k1`, `-k 1`, `--key=1`
    - le opzioni senza argomento con un trattino solo sono raggruppabili: `-b -r -c` è equivalente a `-brc`
  - gli argomenti sono solitamente (ma non necessariamente) nomi di file e/o directory

Comandi

- Primo esempio (*sinossi*) di comando:
  `man [sezione] comando`
  - dà informazioni complete su un comando
  - per esempio, si può (in un certo senso, ricorsivamente) digitare il comando `man man`
    - come risultato, si apre una pagina che illustra tutte le possibili opzioni che sono accettate dal comando `man`
  - considerando gli altri comandi visti sopra, si può anche eseguire: `man cp`, `man ps`, `man chmod`
  - si vede subito dalla *synopsis* che l'esempio dato sopra è molto semplificato, anche se l'uso tipico è quello

Comandi

- `man [sezione] comando`
  - si può notare che in alto a sinistra c'è scritto `MAN(1)`: vuol dire che la sezione è la `1`, quindi, lo stesso risultato si sarebbe ottenuto scrivendo `man 1 man`
  - si può navigare una pagina di manuale con le frecce cursore e con `PagUp`, `PagDown` (per sistemi in cui manca il programma `less`: si può solo premere la barra spaziatrice...)
  - si può ricercare una parola scrivendo prima lo *slash* (ovvero, il carattere `/`) e poi la parola da cercare (basta poi scrivere solo lo slash per cercarla ancora)
    - non tutto può essere cercato: provare a cercare il singolo carattere `[`
  - per uscire da una pagina di manuale, premere il tasto `q`
  - **Esercizio**: provare ad usare alcune delle opzioni di `man` riportate nella sinossi completa

Comando `man`

- `man [sezione] comando` apre le pagine del manuale (man pages) nel terminale
- Le *man pages* di Linux sono suddivise in 10 tematiche:
  - (1) Comandi utente
  - (2) Collegamenti del sistema
  - (3) Funzioni del linguaggio di programmazione C
  - (4) Formati dei file
  - (5) File di configurazione
  - (6) Giochi
  - (7) Varie
  - (8) Comandi per l'amministrazione del sistema
  - (9) Funzioni del kernel
  - (10) Nuovi comandi
- Ad esempio, sia usando `man clear` che (restringendo la ricerca) usando `man 1 clear`, si apre la pagina del manuale riguardante il comando `clear`

Comandi `whatis` e `apropos`

- `whatis [opzioni] parolachiave` cerca le parole chiave nel manuale (o meglio nel database whatis)
- Se la parola cercata è presente nel manuale, `whatis` ne fornisce una breve descrizione nel terminale
- Sono visualizzate solo le corrispondenze con parole intere
- `apropos stringa` cerca una o più stringhe nel database whatis
- A differenza di whatis, sono visualizzate tutte le corrispondenze
- Ad esempio `apropos keyboard` visualizza le righe del database whatis contenenti la stringa keyboard

Comandi `history` e `clear`

- `history` mostra i comandi eseguiti
- Su Bash vengono memorizzati nella cronologia (*history*) gli ultimi comandi inseriti nella riga di comando (di solito 500)
- Consente di ricercare nella lista dei comandi precedenti con i tasti freccia ed eseguirli di nuovo confermando con il tasto di invio

- `clear` serve a rimuovere il contenuto dello schermo
- Si ottiene un terminale vuoto con aperta solo la finestra della riga di comando
- Gli input immessi precedentemente rimangono comunque memorizzati nello *scrollback buffer*

Comandi `help` e `info`

- `help` mostra una lista dei comandi shell integrati (comandi built-in)
- `help comandoshell` fornisce una descrizione del corrispettivo comando
- Molti comandi accettano anche l'opzione -h (o - -help) che fornisce una breve descrizione sull'utilizzo del comando e delle sue opzioni
- `info comando` fornisce informazioni estese sul comando
- Nella maggior parte dei casi si hanno le informazioni che si possono richiamare tramite man, ma con collegamenti che agevolano la navigazione nel manuale

Utenti, filesystem e file

Il filesystem

Filesystem in Linux

- Un **filesystem** è un'organizzazione di un'area di memoria (tipicamente di massa, come il disco), basata sul concetto di *file* e di *directory*
  - una directory serve a contenere al suo interno altre directory oppure file
  - induce naturalmente una struttura gerarchica, ad albero, dove ogni nodo è una directory o un file
  - solo le directory possono avere figli
  - i file *regolari* contengono sequenze di bit dell'area di memoria sulla quale c'è il filesystem e possono essere testi, dati, programmi sorgente, eseguibili
  - file *speciali* sono directory, device (dispositivi hardware collegati visti come file), pipe (file per lo scambio di dati sincrono tra due processi concorrenti, link (riferimento ad un altro file o directory)

Filesystem in Linux

- Linux ha un solo filesystem principale, che ha come radice la directory / (cioè la directory *root*)
  - tutti i file e le directory sono contenuti, direttamente o indirettamente, in tale directory
  - le foglie dell'albero possono essere directory vuote oppure file
  - all'interno della stessa directory non ci possono essere due file, due directory oppure un file e una directory con lo stesso nome
  - cambiare le maiuscole/minuscole è sufficiente a distinguere tra due files o directory: `nomeFile` è diverso da `nomefile`

Filesystem in Linux

- Ogni file o directory è raggiungibile dalla directory radice attraverso un *path assoluto*
  - una sequenza di directory separate da slash e avente slash come primo carattere
  - (quindi, il carattere slash non può essere usato per dare un nome ad una directory o ad un file)
  - esempio `/home/utente1/dir1/dir3/dir7/file.png`
  - come parziale eccezione, è un path assoluto anche quello che comincia con una tilde $\sim$
  - infatti, come vedremo, la tilde è una scorciatoia per la directory home dell'utente corrente x: `/home/`x