Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2024/2025

Corso di Laurea in Matematica

Bash: primi comandi

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

1. Filesystem e file
   - Comandi su file: cp, mv e rm
2. Programmazione Bash
   - Caratteristiche principali
   - Primi passi su shell scripting

Utenti, filesystem e file

I comandi cp, mv e rm

Il comando cp

- Comando `cp [-r] [-i] [-a] [-u] {filesorgenti} filedestinazione`: permette di copiare file e directory
  - 2 modalità basilari:
    - con 2 argomenti, `filedestinazione` può essere una directory od un file (in questo caso, ovviamente, il sorgente dev'essere a sua volta un file...)
    - con 2 argomenti, la destinazione può non esistere, e verrà creata (un file se la sorgente è un file, una directory se la sorgente è una directory)
    - con più di 2 argomenti, `filedestinazione` dev'essere una directory (esistente)
    - se si vuole copiare un file che si trova in un'altra directory nella cwd (mantenendone il nome), il secondo argomento sarà `.`

Il comando cp

- Comando `cp [-r] [-i] [-a] [-u] {filesorgenti} filedestinazione`
  - Tra i `filesorgenti` ci possono essere file e/o directory
  - Ovviamente, se la destinazione è una directory esistente, i sorgenti vengono copiati *dentro* la directory destinazione (anche se il sorgente è una directory)
  - Se ci sono directory tra i sorgenti, allora occorre dare l'opzione `-r`, altrimenti quelle directory non verranno copiate (verranno copiati solo i file)
  - Se la copia avviene su file esistenti, verranno sovrascritti; con l'opzione `-i`, prima di sovrascrivere viene chiesta conferma

Il comando cp

- **Esercizio** verificare il funzionamento del comando cp con diverse tipologie di argomenti:
  1. 2 file (tutti e 4 i casi tra esistenti e non)
  2. 2 directory (tutti e 4 i casi tra esistenti e non)
  3. file o directory in subdirectory
  4. 3 file (l'ultimo sia esistente che non)
  5. 3 directory (idem)
  6. 2 file e 2 directory
  7. ...

Il comando cp

- Comando `cp [-r] [-i] [-a] [-u] {filesorgenti} filedestinazione`
  - Con l'opzione `-u`, la sovrascrittura avviene solo se l'mtime del sorgente è più recente di quello della destinazione (o quello di destinazione non esiste)
  - I permessi del file sorgente potrebbero non venire preservati:
    - sono soggetti alle regole del comando `umask` che vedremo
    - per forzare a mantenere i permessi, c'è l'opzione `-a` (che serve anche per altri motivi)

Il comando cp

- Comando `cp [-r] [-i] [-a] [-u] {filesorgenti} filedestinazione`
  - **Esercizio** verificare il funzionamento di ciascuna delle opzioni mostrate:
    - per l'opzione `-i`, provare a copiare un file sorgente in un file destinazione esistente e vedere che effettivamente viene chiesta conferma
    - l'opzione `-u`
    - per l'opzione `-a`, è sufficiente creare un file ed aggiungere qualche permesso, poi fare la copia con e senza `-a`

Il comando `mv`

- Comando: `mv [-i] [-u] [-f] {filesorgenti} filedestinazione`
  - Come cp, ma serve a *spostare* anziché *copiare*: quindi, i sorgenti risulteranno *cancellati* dopo l'esecuzione del comando, ed esisteranno solo nella destinazione
  - Con 2 argomenti omologhi (2 file o 2 directory) effettua in pratica una *ridenominazione*
  - Le opzioni `-i` e `-u` hanno lo stesso significato di cp
  - `-f` è il contrario di `-i` (ed è l'opzione di default)
  - **Esercizio** rifare entrambi gli esercizi visti per cp

Il comando `rm`

- Comando `rm [-f] [-i] [-r] {file}`: per cancellare
  - Cancella definitivamente i file e le directory indicati (non completamente vero nel caso di hard link, che vedremo)
  - Le opzioni `-f` e `-i` sono come quelle della `mv` (ma stavolta il default è `-i`)
  - L'opzione `-r` è come quella della `cp`
  - **Esercizio** Provare a cancellare file, dicrectory, sottodirectory e file in contenuti sottodirectory

Programmazione Bash

Caratteristiche principali

La shell

- La **shell** è un interprete di comandi che permette all'utente di comunicare col sistema operativo
- Tramite la **shell** è possibile impartire comandi e richiedere l'avvio di altri programmi
- Finora, abbiamo visto un uso molto limitato della Bash:
  - Solo **comandi singoli** e poi invio (in *foreground* o in *background* usando &)
  - **Input** da tastiera oppure da file
  - **Output** su schermo (vedremo che si può specificare un file dove mandare l'output)

Bash script

Primi passi su shell scripting

Cominciare

- Prendere un comando qualsiasi (ad esempio, `ls -l`), e scriverlo su un file di testo a salvarlo con un nome a vostra scelta, per esempio `filename`
- È possibile eseguire tale file in diversi modi:
  1. `source filename`
  2. `. filename`
  3. `bash filename`
  4. `chmod u+x filename; ./filename` (ma ci vorrebbe una prima riga, che comincia con `#!` - detta *shabang* - che specifica quale interprete utilizzare per eseguire il file stesso, ad esempio: `#!/bin/bash`)

Cominciare

- Il file `filename` contenente un insieme di comandi è un *bash script* (spesso, si usa l'estensione `.sh` o `.script`)
- Eseguirlo nei secondi 2 modi equivale a lanciare una sottoshell (sempre, anche se c'è dentro un solo comando) che esegue uno alla volta i comandi contenuti nello script
- Invece, nei primi 2 modi *non* si lancia una sottoshell, e l'esecuzione avviene nel contesto della shell corrente
- La bash permette di avere un vero e proprio linguaggio di programmazione i cui comandi base sono i comandi della shell (non solo quelli visti finora) più gli assegnamenti e i controllori di flusso

Script bash di prova

- **Esercizio** Creare uno script che contenga i seguenti comandi
  - comando ls -l
  - comandi per creare una directory con dentro una sottodirectory
  - comandi per creare diverse directory (almeno due)
  - comandi per creare una directory con dentro una sottodirectory
  - comandi per creare uno o più file nelle diverse directory o sottodirectory file o directory in subdirectory
  - comando tree
  - ...

Cominciare

- Quindi, è possibile prendere decisioni (ad esempio, con l'`if`) ed eseguire cicli (ad esempio, con il `for` e il `while`)
- Di conseguenza, alcuni comandi potrebbero non essere eseguiti o eseguiti più volte
- Se ci sono errori di sintassi:
  - la parte che precede l'errore viene sempre eseguita
  - la parte che segue l'errore potrebbe essere eseguita o no, a seconda della gravità dell'errore