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

1. Filesystem e file
   - Il filesystem
   - Comandi per la gestione di file e directory
2. Programmazione Bash
   - Caratteristiche principali
3. Bash script
   - Primi passi su shell scripting

Utenti, filesystem e file

Il filesystem

Filesystem in Linux

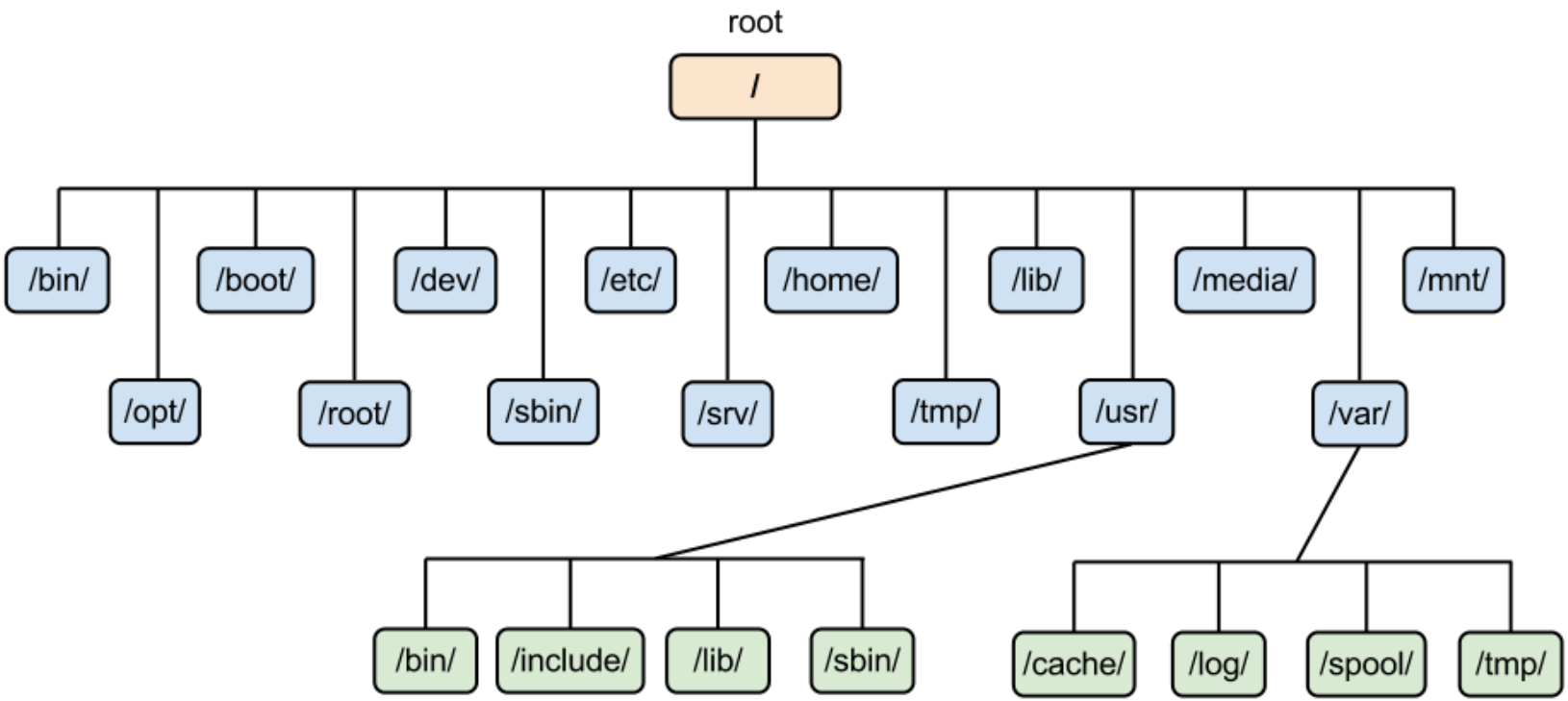
- Un **filesystem** è un'organizzazione di un'area di memoria (tipicamente il disco), basata sul concetto di *file* e di *directory*
  - una directory contiene al suo interno altre directory oppure file
  - induce naturalmente una struttura gerarchica, ad albero, dove ogni nodo è una directory o un file
  - 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 nella directory root
  - le **foglie** dell'albero possono essere directory vuote oppure file
  - all'interno di una directory non ci possono essere due file, due directory oppure un file e una directory con lo stesso nome
  - nomi che differiscono per maiuscole/minuscole sono diversi e possono coesistere: `nomeFile` è diverso da `nomefile`

Filesystem in Linux

- Ogni file o directory è raggiungibile dalla directory radice attraverso un *path assoluto*
  - il **path** è una sequenza di directory separate da slash e avente slash come primo carattere
  - 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


root
/
/bin/
/boot/
/dev/
/etc/
/home/
/lib/
/media/
/mnt/
/opt/
/root/
/sbin/
/srv/
/tmp/
/usr/
/var/
/bin/
/include/
/lib/
/sbin/
/cache/
/log/
/spool/
/tmp/

Directory principali

- Alcune delle principali directory e loro utilizzo su Ubuntu
  - `/bin` Contiene i programmi basilari per la gestione del sistema, cioè buona parte dei comandi base utilizzabili dalla riga di comando da qualsiasi utente senza dover utilizzare i privilegi dell'amministratore
  - `/boot` Contiene le immagini del kernel e i file indispensabili al bootstrap del sistema
  - `/dev` È la directory che individua sotto forma di file le periferiche hardware
  - `/etc` Contiene i file di configurazione del sistema. Ad esempio `/etc/apt` file di configurazione dei repository
  - `/home` Contiene tutte le directory personali degli utenti del sistema
  - `/lib` Contiene tutte le librerie condivise del sistema

Directory principali

- Alcune delle principali directory e loro utilizzo su Ubuntu
  - `/root` Contiene la directory home dell'amministratore del sistema ed è esplorabile solo utilizzando i privilegi da super utente. Il contenuto è analogo a quello delle singole home-utente descritte nel capitolo della directory /home
  - `/tmp` Contiene file temporanei
  - `/usr` È la directory che contiene la maggior parte dei programmi installati sul sistema
    - `/usr/bin` file eseguibili delle applicazioni accessibili a tutti gli utenti, cioè i programmi normalmente avviabili dal menù delle applicazioni.
    - `/usr/sbin` file eseguibili delle applicazioni di sistema accessibili solo all'amministratore
    - `/usr/share` file di vario genere (configurazione, documenti di testo, ecc..) indipendenti dall'architettura del sistema (i386, amd64). Ad esempio, le cartelle backgrounds, icons e themes contengono sfondi, icone e temi del desktop

Directory

- Concetto di *current working directory* (cwd)
  - è la directory mostrata nel prompt
  - per sapere qual è la cwd in dettaglio si usa il comando `pwd`
  - per **cambiare directory**, si usa il comando `cd [path]`
    - se non si specifica il `path`, la nuova directory sarà la home)
  - all'interno di `path` si può usare
    - `..` che porta alla directory *parent*, che contiene quella attuale (se fatto sulla root, ritorna la stessa root),
    - `.` indica la directory stessa
    - `..` e `.` si possono usare anche ripetutamente: `../.././.././` equivale a risalire 3 livelli nella gerarchia delle directory, a partire dalla cwd

- **Esercizio:** posizionarsi nella directory `/lib` (o un'altra a vostra scelta) e controllare come cambia il path nel prompt

Directory

- A partire dalla cwd, si possono usare i *path relativi*
  - sono quelli *non* assoluti, quindi non cominciano con uno slash nè con la tilde
- La **differenza tra un path assoluto ed uno relativo** è che il path assoluto è valido qualsiasi sia la cwd, mentre il path relativo può non essere valido quando si cambia la cwd
  - **esempio**: se la cwd è la home dell'utente `utente1`, allora il file con path relativo `dir1/dir3/dir7/file.png` può essere raggiunto anche con:
    `./dir1/dir3/dir7/file.png` oppure con
    `../utente1/dir1/dir3/dir7/file.png`
  - se si esegue il comando `cd dir1/dir3` (o equivalentemente, `cd /home/utente1/dir1/dir3/`), il file è raggiungibile con il path relativo
    `dir7/file.png` o anche con
    `./dir7/file.png`, o anche con
    `../dir3/dir7/file.png`

Utenti, filesystem e file

Comandi per la gestione di file e directory

Comando `ls`

- Il comando `ls [-la] [-R] [nomedir]` mostra il contenuto della directory `nomedir` (o della cwd, se `nomedir` non è dato)
  - `ls -l` mostra informazioni relative al contenuto della directory
  - `ls -a` mostra i file considerati *nascosti* (*hidden*), cioè i file con nomi che cominciano con il punto
  - `ls -R` mostra tutto il sottoalbero con radice in `nomedir`
  - `ls -s` mostra la dimensione dei file

Comando `mkdir`

- Il comando `mkdir [-p] nomedir` crea la directory `nomedir` (vuota)
  - Se si usa l'opzione `-p` si può creare una sequenza di directory annidate, cioè specificando il path `nomedir` composto da più directory si creano tutte le directory nel path (se non esistono)
  - **Esempio:** supponendo che `dir11` esista già, il comando `mkdir -p dir11/dir13/dir15`, creerà la directory `dir13` dentro `dir11`, e poi `dir15` dentro `dir13`
  - `Esercizio` provare il comando `mkdir` con e senza l'opzione `-p`
- **Esercizio**
  - creare l'intero albero di directory dato sopra (ovvero, `/home/utente1/dir1/dir3/dir7/`), posizionarsi dentro `dir1` e poi in `dir7` sia usando che non usando la directory parent `..`
  - controllare il risultato usare il comando `ls`;
  - controllare come cambia il path riportato nel prompt

Comando `touch`

- Il comando `touch nomefile` crea il file `nomefile` (vuoto)
- Per aprire e scrivere in un file, si può usare un editor di testi
- Provare i comandi `geany nomefile &` o `gedit nomefile &` (se i corrispondenti editor sono installati)
- **Osservazione** il carattere `&` serve per l'esecuzione in background e permette di mantenere la finestra del terminale attiva (altrimenti il processo è bloccato)
- Si possono anche usare editor che si interfacciano direttamente con il terminale (se sono installati):
  - `nano nomefile` oppure `pico nomefile` (**senza** usare il carattere &)
  - oppure `vi nomefile` che è un editor un po' più complicato:
    - per uscire da `vi`, digitate i caratteri `:q` seguiti da invio

Comando `cat`

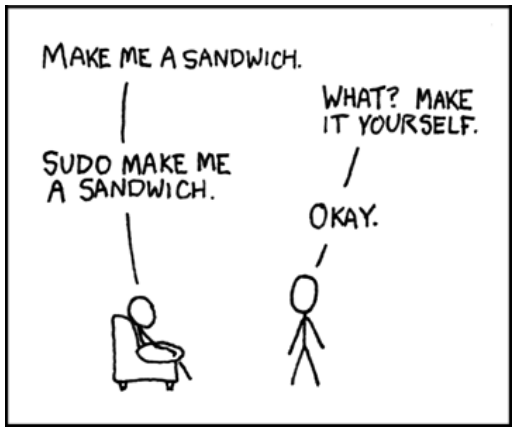
- Il comando `cat [nomefile]`: scrive a schermo il contenuto di `nomefile`
  - funziona bene solo se il file è di testo (e se usa la codifica riconosciuta da `cat`), altrimenti scrive caratteri incomprensibili
  - si può usare `cat` per leggere più di un file alla volta scrivendo `cat file1 file2... filen`:
    - in questo caso il comando `cat` stamperà prima il contenuto del primo file, poi quello del secondo, e così via
    - l'output sarà quindi la concatenazione del contenuto dei file specificati
  - senza argomenti, resta in attesa: se si scrive qualcosa e poi si preme invio, ripete quanto scritto, finché non si preme CTRL+d, che è il carattere EOF (*end-of-file*)

Creare altri file (non di testo)

- Per creare file non di testo, occorre usare opportuni altri programmi (a seconda di cosa serve)
  - ad esempio, i vari applicativi di LibreOffice possono essere usati per creare file contenenti documenti formattati (DOC, DOCX, ODT, etc), fogli di calcolo (XLS, XLSX, ODS, etc)
  - basta usare il comando `libreoffice nomefile`
    - se `libreoffice` non è installato, va installato ma servono i diritti da amministratore (cioè occorre usare il comando `sudo apt-get install libreoffice`)
    - attenzione: `nomefile` deve già esistere; altrimenti si può eseguire `libreoffice` senza argomenti e poi usare l'interfaccia grafica per creare un nuovo documento del tipo desiderato

Amministratore, utente e installazione di pacchetti

- Nel caso di Ubuntu, l'utente creato a tempo di installazione è un **sudoer**, cioè un *utente con privilegi di amministratore di sistema*, e appartiene al gruppo predefinito `sudo`
  - può eseguire comandi da *superutente* semplicemente preponendo il comando `sudo`, derivato da **s***uper* **u***ser* **do**: per esempio, può installare nuovi pacchetti
  - `sudo comando` è un comando particolare, che prende come argomento un altro comando, che può avere svariati argomenti



Comando `tree`

- Si può visualizzare un intero albero di directory con il comando `tree [-a] [-L maxdepth] [-d] [-x] [nomedir]`
  - potrebbe non essere installato e per installarlo si può scrivere: `sudo apt-get install tree`
  - usare l'opzione `-L` per limitare la profondità dell'albero mostrato
- In generale: se si dà un comando sbagliato, e l'output sembra non finire mai, provare a premere CTRL+c

Comando `xdg-open`

- Il comando `xdg-open nomefile|url` apre un file o una url con l'applicazione selezionata come *preferita*
  - se l'argomento è un file sarà scelta l'applicazione preferita per il tipo di file
  - se l'argomento è una *url* sarà scelto il browser preferito
- Le opzioni sono:
  - `--help` mostra la sinossi del comando
  - `--manual` mostra la pagina del manuale relativa al comando
  - `--version` mostra le informazioni sulla versione di `xdg-utils`
- `xdg-open` fa parte del pacchetto `xdg-utils` che è un insieme di *tools* (o *utilities*) che permettono di integrare varie applicazioni con la distribuzione Linux utilizzata

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
- Un po' di storia delle (principali) shell:
  - `sh`, detta *Bourne Shell*, dal nome del ricercatore che la ideò, nel 1977 ai Bell Labs
    - Le shell ispirate a `sh` hanno il prompt che termina in `$`
  - `bash`, da *Bourne Again Shell*, è la `sh` reimplementata, e migliorata, per GNU (Fox, 1989)
    - Come la `sh`, ma con le caratteristiche interattive, come ad esempio `history`

Uso interattivo della Bash

- 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)
- Semplificando possiamo dire che la shell è un programma che esegue iterativamente sempre la stessa operazione:
  - attende che gli venga fornito in input un comando da eseguire, lo valuta per verificare che il comando sia sintatticamente corretto e lo esegue
  - quindi torna ad attendere che sia fornito il comando successivo

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

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

Cominciare

- Tutto quello che si scrive sulla bash interattiva può essere messo in uno script; per separare i comandi, si può usare l'andata a capo al posto del ;
- Il viceversa non è vero, perché le andate a capo possono essere solo negli script, infatti, nella shell interattiva, premere invio vuol dire *esegui il comando*

Esempi

- Provare:

```
# Primo script:  hello.sh
echo Hello World # questo è un commento
```

  - si può eseguire con bash e il nome del file
  - ovviamente si ottiene lo stesso risultato senza usare lo script, con il comando `echo Hello World`

- **Esercizio** Creare uno script che:

  1) crea la directory `dir1`
  2) crea la directory `dir2/21`
  3) crea un file nella directory `dir1`, uno nella directory `dir2` e una nella directory `dir21`