Sistemi Operativi

AAF - Secondo anno - 3CFU

A.A. 2024/2025

Corso di Laurea in Matematica

UNIX - Linux - Bash e primi comandi

Annalisa Massini

Dipartimento di Informatica
Sapienza Università di Roma

Argomenti trattati

Il sistema operativo UNIX

Nascita di UNIX

- Il sistema operativo UNIX viene progettato tra gli '60 e gli anni '70 nei laboratori Bell di AT&T
- Il progetto venne portato avanti soprattutto grazie agli sforzi di Ken Thompson e Dennis Ritchie
- Nel 1973 Thompson e Ritchie riscrissero il kernel di Unix nel linguaggio ad alto livello C, usando per la prima volta un linguaggio diverso dall'assembly
- Anche oggi (quasi) tutte le implementazioni di UNIX sono scritte in C

Nascita di UNIX

- L'interesse per il nuovo sistema operativo iniziò a diffondersi, in particolare dopo la presentazione al Symposium for Operating System Principles
- La distribuzione di UNIX sotto forma di codice sorgente permetteva agli utilizzatori di personalizzare il sistema operativo in base alle proprie esigenze
- La possibilità di personalizzare il codice e il basso costo fece sì che diverse università iniziassero a produrre modifiche di UNIX per utilizzarlo su computer diversi
- Alla fine degli anni '70, si affermarono due diversi standard: **Unix System V** della AT&T e **BSD (Berkeley Software Distribution)** dell'Università di Berkeley California

Caratteristiche di UNIX

Caratteristiche principali di un sistema operativo Unix

- **Ambiente multiutente** - Permette a più utenti di interagire contemporaneamente con il sistema da terminali diversi, evitando interferenze tra le attività degli utenti
  - Ogni utente è identificato univocamente da un nome logico (username) e suddiviso in gruppi, ciascuno identificato da un nome (groupname)
  - Il sistema definisce anche un utente root, che rappresenta l'amministratore di sistema e non ha limitazioni nell'accesso alle risorse
- **Ambiente multiprogrammato** (multitasking) - Il kernel supporta l'esecuzione contemporanea di più processi gestiti tramite divisione di tempo (*timesharing*).

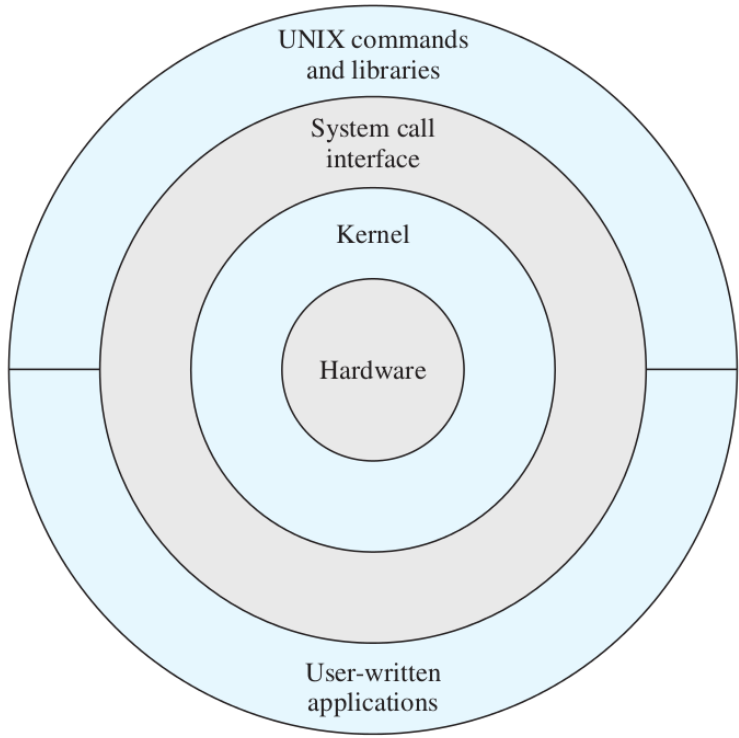
Caratteristiche di UNIX

Caratteristiche principali di un sistema operativo Unix

- **Gestione della memoria virtuale** - Unix utilizza i meccanismi paginazione e segmentazione per consentire a ogni processo di indirizzare un'area di memoria superiore alla memoria centrale disponibile
- **Portabilità** - Grazie all'uso del linguaggio C, Unix è altamente portabile e disponibile su una vasta gamma di architetture hardware
  - Offre ambiente di sviluppo per programmi in C, con un ricco insieme di strumenti per lo sviluppo di applicazioni in C, tra cui il compilatore cc
- **Ambiente aperto** - Unix implementa molti dei servizi e protocolli di comunicazione più diffusi su Internet, facilitando l'integrazione dei sistemi Unix in una rete

Architettura di UNIX

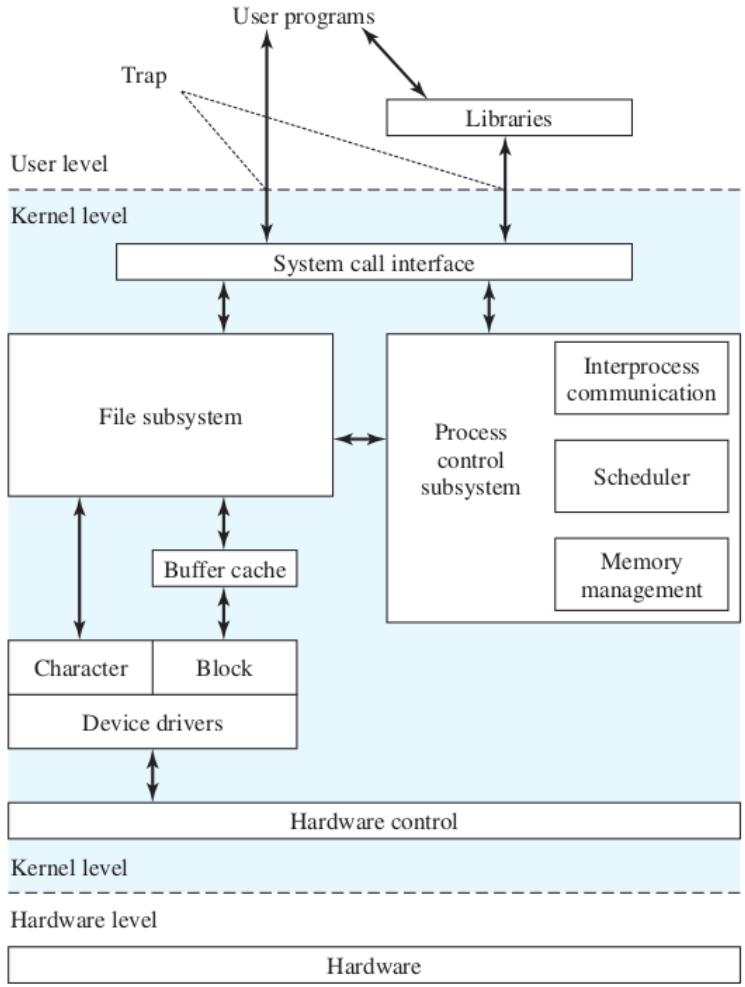
- In figura è mostrata lo schema classico dell'architettura di UNIX
- L'hardware è completamente circondato dal software del SO
- Il SO viene chiamato **kernel** per evidenziare che è isolato da utente e applicazioni
- UNIX comprende anche servizi e interfacce considerate parte del SO:
  - shell
  - interfaccia software
  - componenti del compilatore C (loader, assembler, compiler)
  - strato più esterno con applicazioni utente e interfaccia



UNIX tradizionale

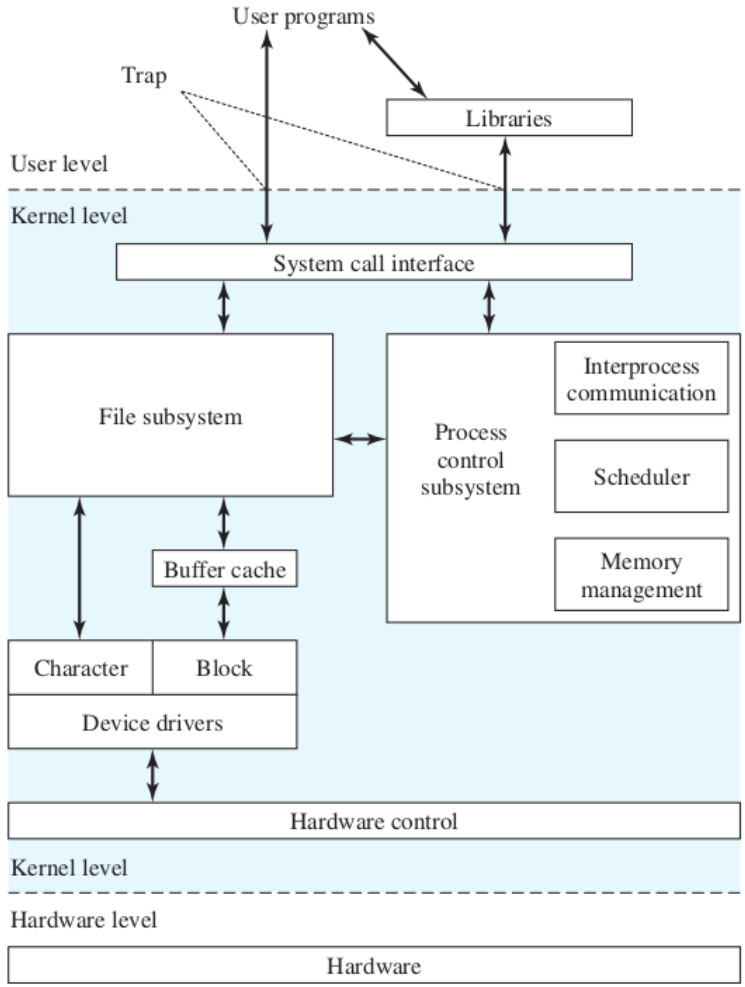
Schema del kernel

- I programmi utente invocano i servizi del SO direttamente o tramite programmi di libreria
- L'interfaccia di chiamate al sistema rappresenta il confine con l'utente e permette di accedere a funzioni specifiche del kernel tramite sw di livello più alto
- Dall'altro lato, il SO contiene routine che interagiscono direttamente con l'HW



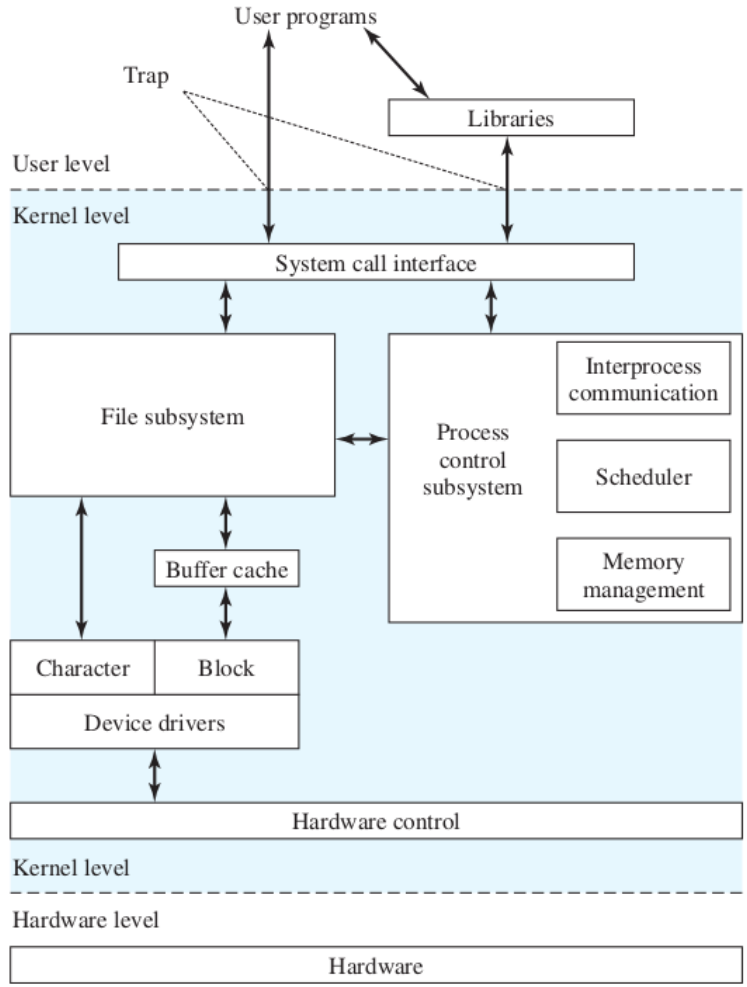
UNIX tradizionale

- Lo strato tra le due interfacce è diviso in due parti:
  - **parte per il controllo di sistema**: responsabile per la gestione della memoria, l'assegnazione e la sincronizzazione dei processi, la comunicazione tra processi
  - **parte per la gestione dei file e dell'I/O**: il file-system scambia dati tra memoria e dispositivi esterni



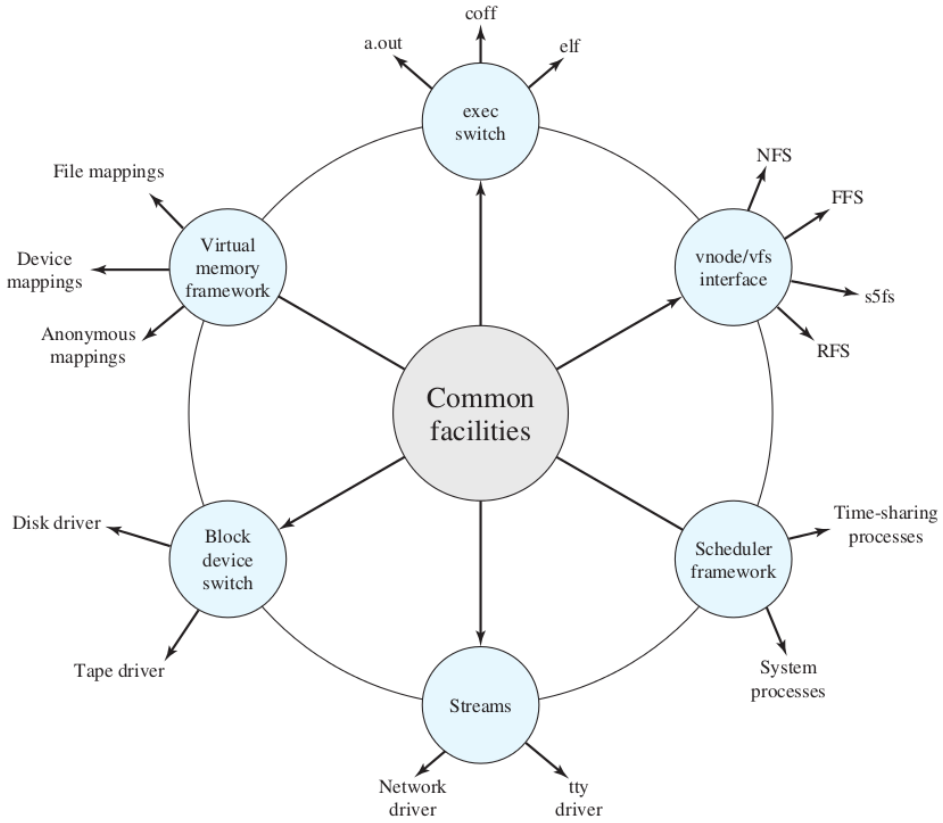
UNIX Tradizionale

- Il sistema UNIX tradizionale:
  - è stato progettato per un singolo processore e non ha la capacità di proteggere le strutture dati da accessi concorrenti di più processori
  - il kernel non è versatile (supporta un solo tipo di file system, di politica di scheduling dei processi e formato di file eseguibile)
- L'aggiunta di nuove capacità alla versione tradizionale di UNIX ha prodotto un kernel sproporzionato



UNIX moderno

- Il sistema UNIX si è evoluto producendo innumerevoli implementazioni
- Si era creata la necessità di:
  - unificare le molte innovazioni introdotte
  - aggiungere caratteristiche più attuali
  - produrre un'architettura più modulare
- Nel tipico Unix moderno c'è un piccolo **core** di funzioni e serviz scritti in maniera modulare necessari ai processi del SO



Kernel monolitico

- Molti kernel UNIX sono **monolitici**:
  - includono tutte le funzionalità del SO in un unico blocco di codice che gira come un singolo processo con un singolo spazio di memoria
  - il kernel è tutto in memoria dal boot allo spegnimento
  - tutte le componenti funzionali del kernel hanno accesso a tutte le strutture dati interne e a tutte le routine
  - se viene effettuato un cambiamento ad una porzione, tutti i moduli e le routine devono essere rilinkate e reinstallate e il sistema deve essere riavviato

Microkernel

- Nei SO con microkernel:
  - solo una minima parte del kernel è in memoria, il resto viene caricato quando serve
  - sempre in memoria: scheduler, sincronizzazione
  - solo a richiesta: gestore memoria, filesystem, driver
- Un kernel monolitico è più efficiente come velocità, ma occupa più memoria e rende difficile la modularità
- Molti sistemi operativi moderni sono a kernel monolitico
  - eccezione degna di nota: Mac OS X

Unix

Caratteristiche di Unix

Caratteristiche principali

Multiutente: più utenti contemporaneamente possono usare lo stesso computer grazie a Unix

Multiprocesso: lo stesso utente può lanciare contemporaneamente più di un processo

File system gerarchico: il file system è organizzato come un albero, dove ogni nodo interno è una directory e ogni foglia è un file o una directory

- è possibile aggiungere file system aggiuntivi (ad esempio, una chiave USB o un CD): viene "montato" su una qualche directory

Caratteristiche principali

Kernel: gestisce memoria (principale e secondaria), processi, I/O, risorse hardware in generale

System calls: funzioni C che possono essere chiamate se ci si vuole interfacciare con il kernel (ad es., per creare un file...)

Shell: programma interattivo che accetta comandi da "girare" al kernel (del tipo: mostra il contenuto di una directory)

Caratteristiche principali

Altro:

- *Ambienti di programmazione:* per permettere di scrivere programmi, tipicamente in C
  - compilatore, debugger, text editor
  - i programmi scritti in linguaggi interpretati (ad es., Python o Java) non vengono eseguiti direttamente, ma appunto tramite l'interprete
  - quindi è come se ci fosse un ulteriore *intermediario*, che con il C è rimosso
  - pertanto, il linguaggio principe per *dialogare* direttamente con il kernel è il C

Caratteristiche principali

Altro:

- *Utilities:* altri programmi, per fare qualsiasi cosa che sia computabile
  - suite Office (OpenOffice, LibreOffice)
  - lettore PDF (Acrobat Reader, evince, ...)
  - browser (Firefox, Chrome, ...)
  - messaggistica (Skype, ...)
  - riproduttore audio/video (VLC, ...)
  - editor di immagini (xfig, gimp, ...)
  - giochi (semplici!)
  - ...
- *Modularità:* programmi installabili a pacchetti, moduli del sistema operativo attivabili e disattivabili

Linux

Linux

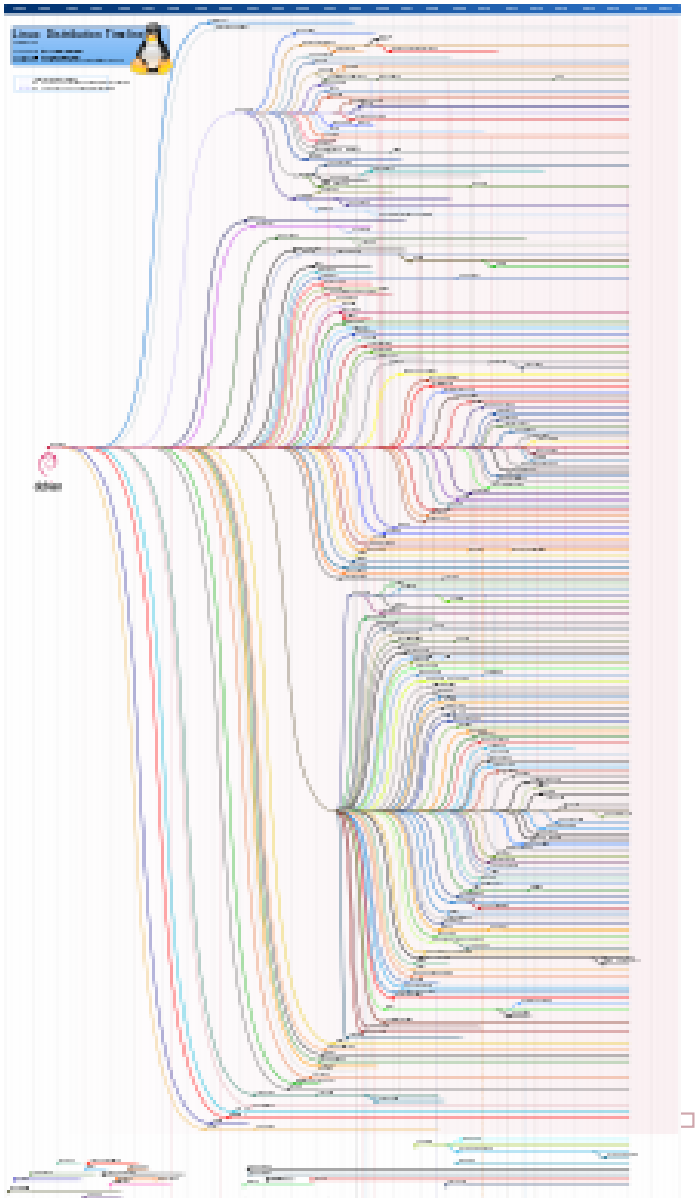
Nascita di Linux

- Nel 1991 Linus Torvalds, allora studente dell'Università di Helsinki, iniziò la scrittura di una estensione del sistema operativo Minix, realizzato a scopo didattico dal prof. Tanenbaum
- L'idea era quella di scrivere una versione di Unix per PC, ispirata a Minix, e sulla quale tutti potessero apportare modifiche al codice sorgente
- Nel 1994 venne rilasciata la prima versione definitiva (versione 1.0) del kernel di Linux e nacquero RedHat, Debian e SUSE, che sono ancora oggi tra le distribuzioni più diffuse

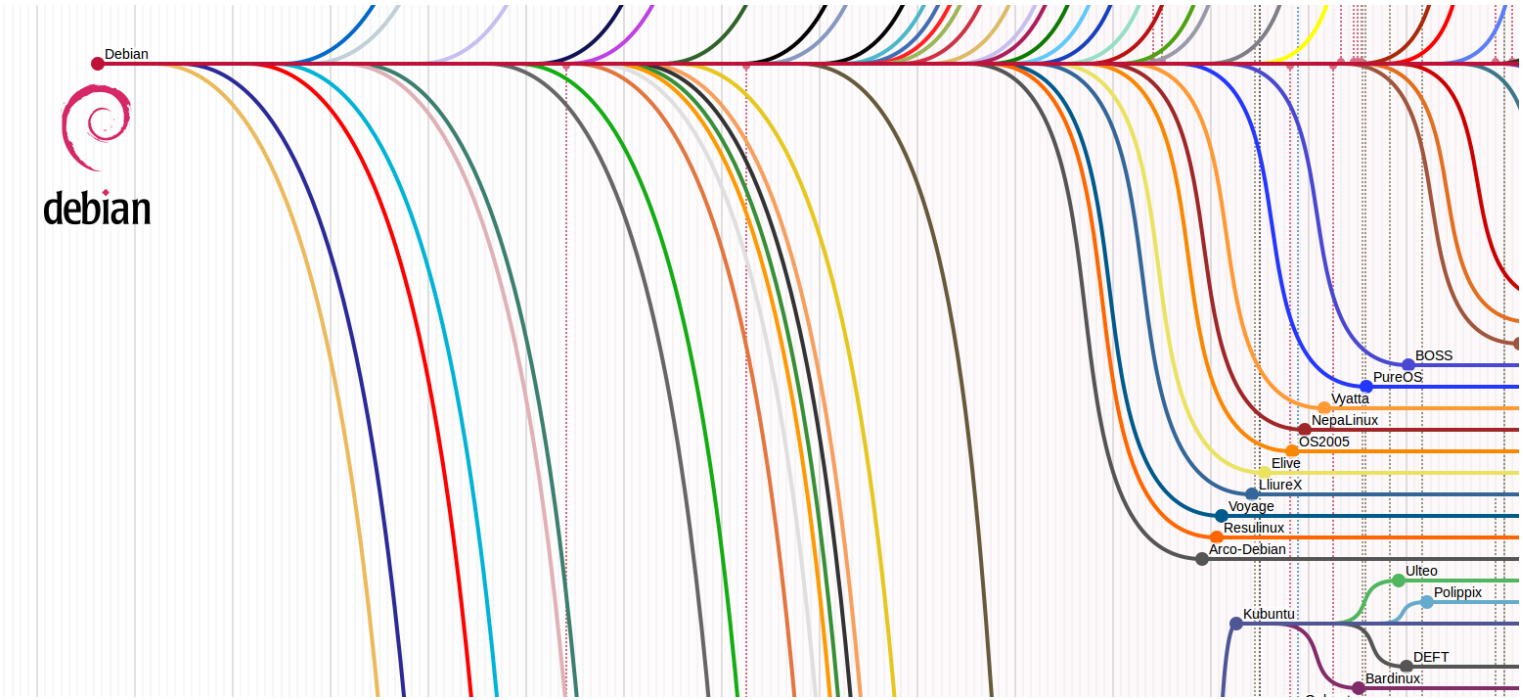
Nascita di Linux

- Linux viene distribuito come software open source, nel rispetto della General Public License (GPL) del movimento GNU Open Source
- L'idea di OpenSource fu lanciata nel 1984 da Richard Stallman, ricercatore del MIT AI Lab, fondatore della FSF (Free Software Foundation) e del progetto GNU (acronimo ricorsivo di "GNU's Not Unix")
- Dal primo rilascio di Linux, molte persone hanno contribuito al suo sviluppo, collaborando su Internet

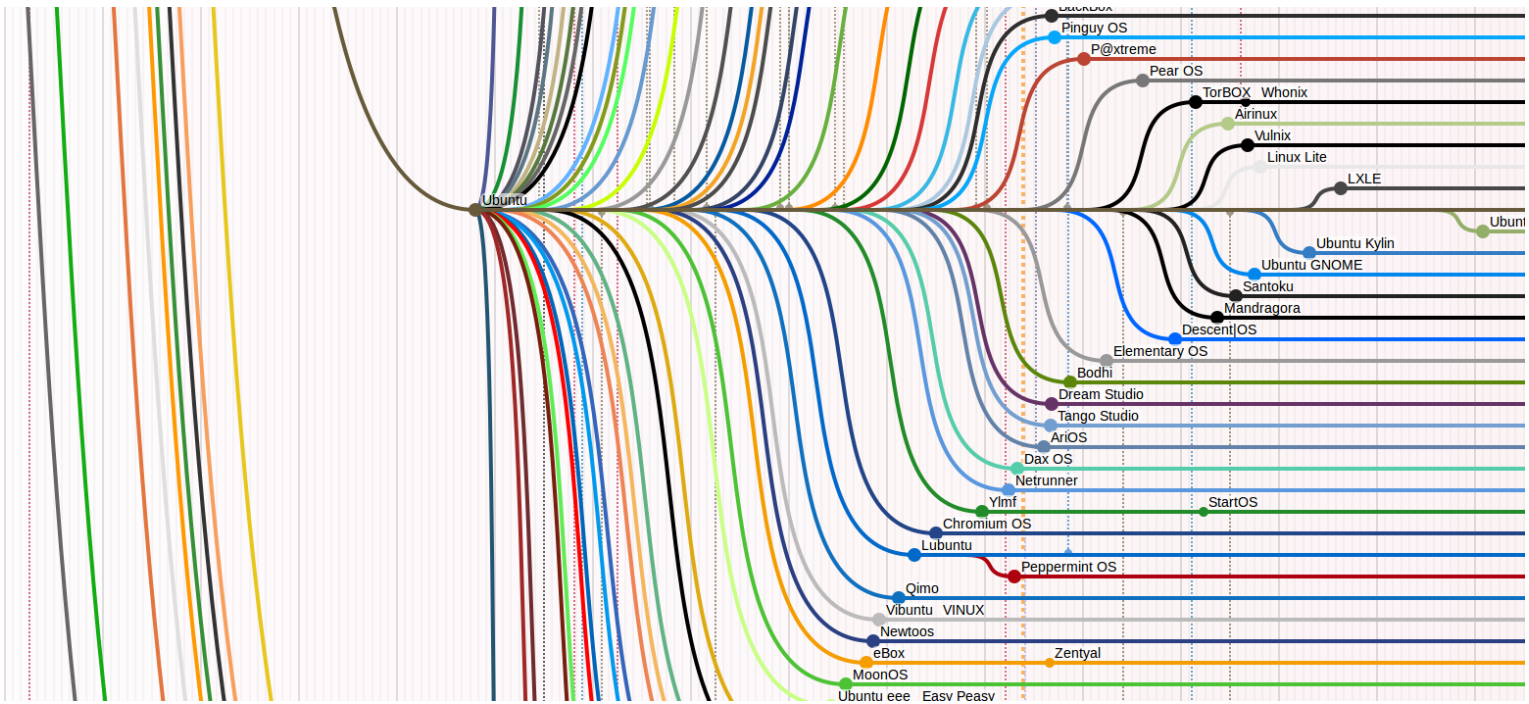
- Ci sono moltissime versioni
- Timeline delle distribuzioni di Linux, da Wikipedia



- Timeline delle distribuzioni di Linux, da Wikipedia (particolare)



- Timeline delle distribuzioni di Linux, da Wikipedia (particolare)



Kernel moderno di Linux

- Linux è principalmente monolitico, ma ha i **moduli**, quindi vantaggi simili ai microkernel
  - è strutturato come una collezione di moduli
  - alcuni moduli particolari possono essere aggiunti e tolti a richiesta dall'immagine in memoria del kernel - **loadable modules**
  - essenzialmente, i diversi file system, i driver per determinati dispositivi di I/O, l'implementazione delle funzionalità di rete
  - un modulo non viene eseguito tramite un suo processo o thread, ma è eseguito in *kernel mode* da parte del processo corrente

- Si può installare Linux su una *macchina virtuale*, come VirtualBox
  - per *macchina virtuale* si intende un applicativo che replica in tutto e per tutto un computer, a partire dal tasto di accensione
  - è possibile far sì che la rete sia condivisa con il sistema operativo *ospitante*
    - per *ospitante* si intende il sistema operativo sul quale è in esecuzione VirtualBox
    - per *ospitato*, si intende il sistema operativo che viene eseguito all'interno di VirtualBox
- Se si usa Windows si può usare WLS, sottosistema Windows per Linux
  - WSL è una funzionalità del sistema operativo Windows
  - consente di eseguire un file system Linux, insieme agli strumenti da riga di comando e alle app dell'interfaccia utente grafica di Linux, direttamente in Windows, insieme alle tradizionali app e desktop di Windows.

Introduzione alla Shell

Informazioni generali e comandi

Terminale e shell

- Per interagire con il sistema operativo si possono dare comandi dalla riga di comando usando il **terminale**
- In questo modo viene utilizzata la **shell**, che funziona su un terminale
- Quando si chiama il terminale viene subito avviata la **shell standard**
- Tra le shell principali vanno menzionate:
  - `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`

Bash shell

- La *shell* è un programma che svolge la funzione di interfaccia tra sistema e utente:
  - comprende un interprete della riga di comando,
  - accoglie gli input utente tramite tastiera (cioè nella riga di comando) e li analizza,
  - avvia i programmi e restituisce all'utente il risultato sotto forma di testo

Bash shell

- Ogni shell possiede un proprio linguaggio di programmazione che consente di scrivere *script shell*
- Per sapere quale shell è in uso si possono usare due comandi:
  - `echo $0` oppure
  - `ps -p "$$" -ocmd -h`
- 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 shell

- Si possono replicare comandi già dati usando i tasti cursore *freccia giù* e *freccia su*
  - una volta trovato il comando cercato, lo si può modificare: utile ad esempio se si vuole rilanciare un programma già dato in precedenza con piccole modifiche
- Si possono anche ricercare parti di comandi dati in precedenza con CTRL+r (trovato il comando, lo si può modificare)

Bash shell

- La bash scrive un *prompt* ed attende che l'utente scriva un `comando`
  - *prompt* sta per *pronto*, e infatti la presenza del prompt indica, solitamente, che la shell è pronta ad accettare un nuovo comando
  - il prompt tipico è così costituito: `nomeutente@nomemacchina:~cammino$`, dove `cammino` è il path dalla directory home alla directory attuale
  - Se si è nella directory principale dell'utente, detta home, c'è solo la tilde ~
  - Se la directory corrente non si trova nel sottoalbero radicato nella home, allora `cammino` è il path assoluto (la tilde non c'è)

Standard input, output ed error

- In Linux si usano i concetti di standard input, output ed error, cioè `stdin, stdout e stederr`
- Quando viene eseguito un comando vengono generati i tre *flussi* (o stream) standard, dove per *flusso* si intende un meccanismo usato per trasferire dati, in questo caso testo
- `stdin` è lo standard input stream e accetta testo come input
- `stdout` è lo standard output stream ed è il testo prodotto in output dal comando eseguito, di solito scritto sullo schermo
- `stderr` è lo standard error stream ed è visualizzato sullo schermo tramite messaggi di errore
- Come per i file, agli standard stream vengono associati degli identificatori: 0 per `stdin`, 1 per `stdout` e 2 per `stederr`

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

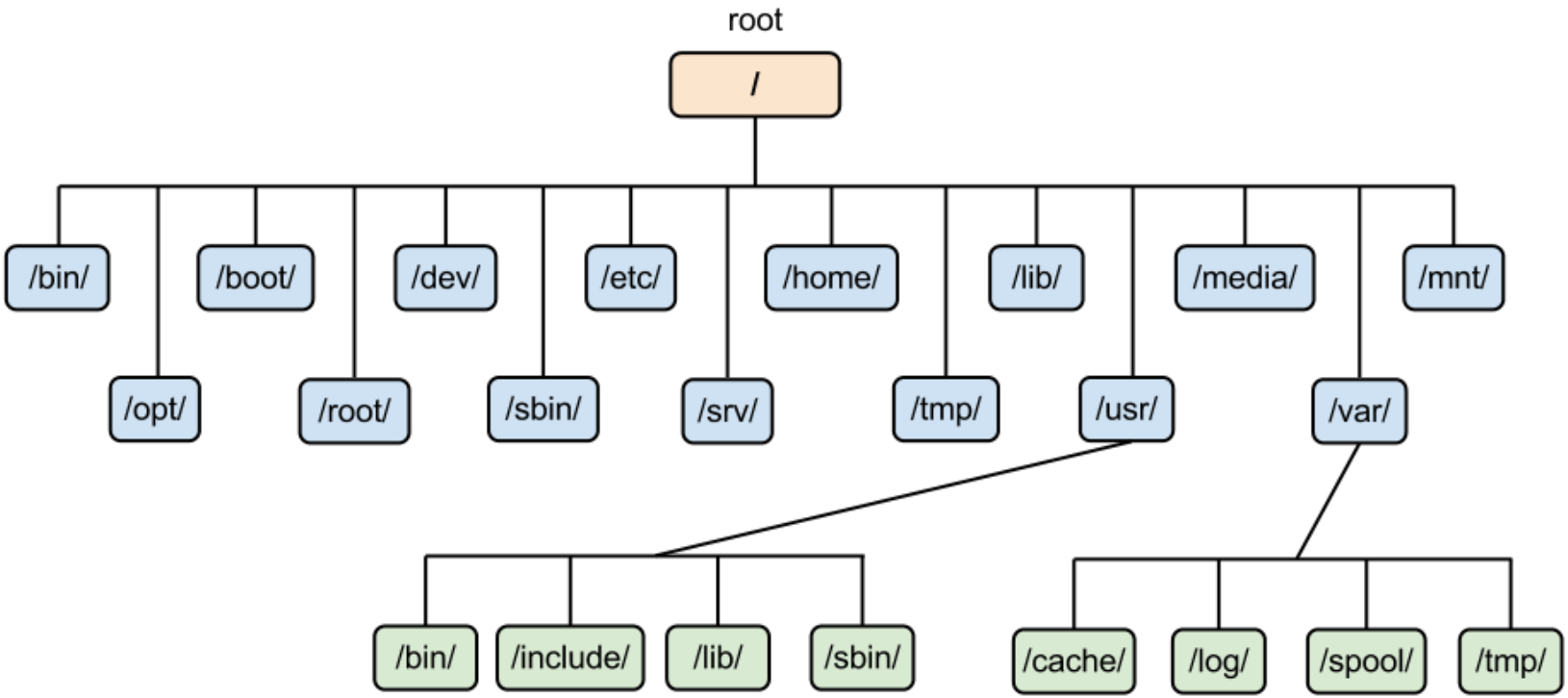
- Un **filesystem** è un'organizzazione di un'area di memoria (tipicamente di massa, come il disco), basata sul concetto di *file* e di *directory*
  - una directory serve a contenere al suo interno altre directory oppure file
  - induce naturalmente una struttura gerarchica, ad albero, dove ogni nodo 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


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`