Introduzione ad UNIX e la Shell

Laboratorio di Fondamenti di Programmazione

Chiara Bodei, *Damiano Di Francesco Maesa, Roberta Gori*

CdL in Matematica, Università di Pisa

a.a. 2021/2022

Chi siamo

Docenti

- Chiara Bodei
- Damiano Di Francesco Maesa
- Roberta Gori

Assistenti

- Filippo Lari
- Niccolò Piazzesi

Replit

Le esercitazioni di laboratorio saranno svolte live dagli studenti usando *Replit* (`https://replit.com/`).

Live Coding: Replit permette ai docenti di vedere e modificare live i progetti su cui state lavorando durante le ore di laboratorio o ricevimento

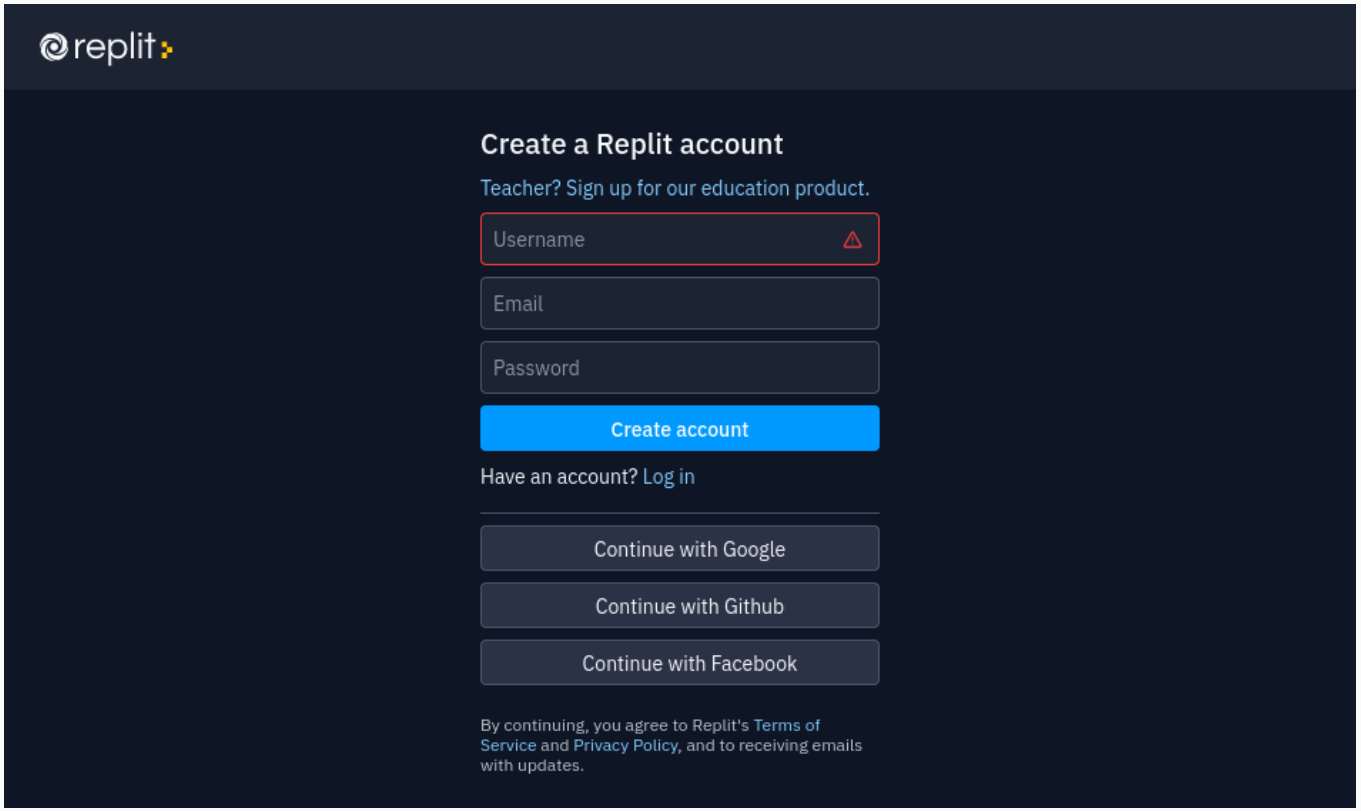
Domande: Replit offre una funzione di chat per interagire con il docente, ma tale strumento non è sempre monitorato. Lo studente dovrà quindi prima alzare la mano su teams da cui il docente sarà poi rediretto al corretto progetto Replit

Testing: Replit include dei test di autovalutazione ma NON sarà la piattaforma ufficiale di autovalutazione (che sarà invece Dijkstra come spiegato in seguito)

Replit - Fisrt Login

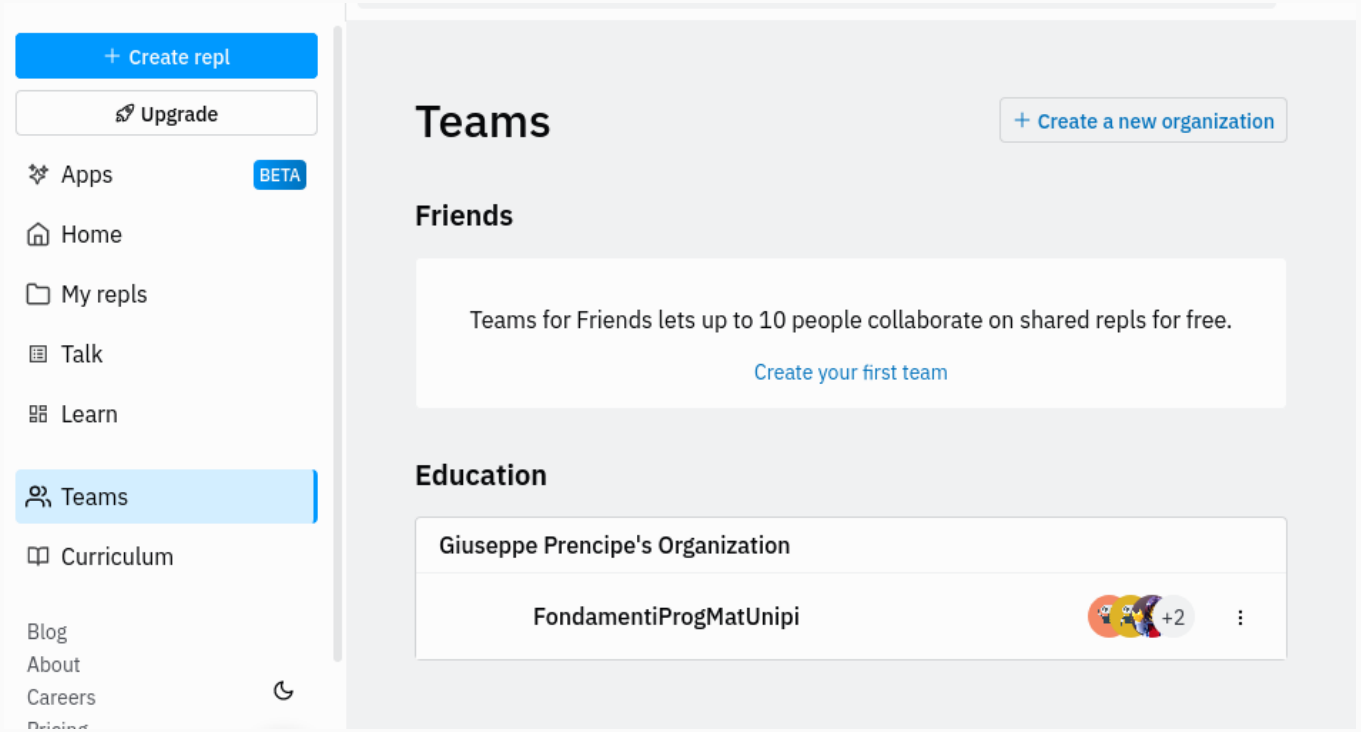
Per usare Replit dovrete prima creare un account `https://replit.com/signup?from=landing`.

Siete pregati di registrarvi con la vostra mail ufficiale di studenti UNIPI.



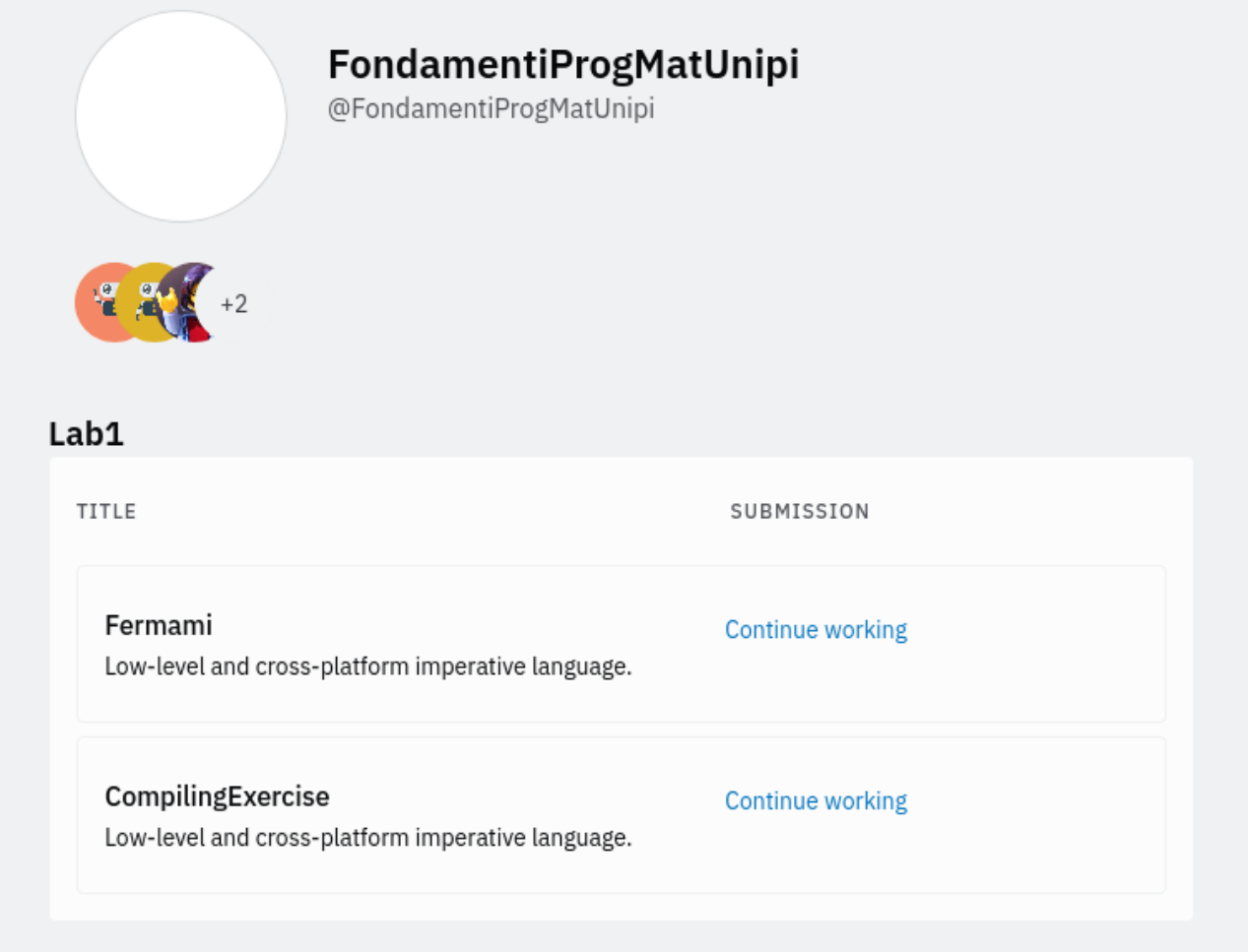
Replit - Join a Team

Una volta creato il vostro account con la mail studenti dovete unirvi al gruppo FondamentiProgMatUnipi usando il link `https://replit.com/teams/join/rrlvfvbhtgwworxwnocqljyzkgjvmebm-FondamentiProgMatUnipi`.



Replit - Start a Lesson

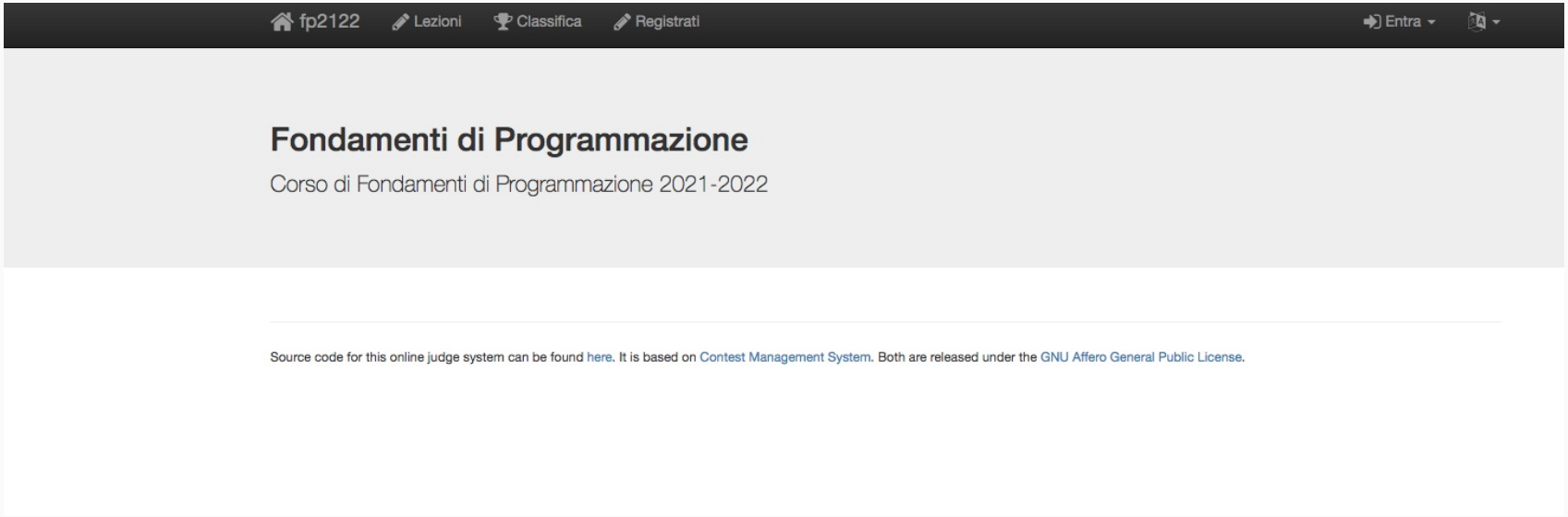
Dal nostro team potete accedere a tutte le esercitazioni di laboratorio pubblicate. Ogni esercitazione conterrà un certo numero di esercizi da completare.



Replit - Live Coding

Per ogni esercizio potete modificare i file, eseguirli e accedere direttamente alla shell.

Piattaforma di autovalutazione



`http://fp2122.dijkstra.di.unipi.it/#/overview`

Registratevi (con mail di studenti UNIPI) e memorizzate bene la password! :)

Unix e Shell

Storia di Unix (1)

Il primo sistema *Unix* fu sviluppato nei laboratori Bell AT&T alla fine degli anni sessanta (1 Gennaio 1970). Unix fu progettato con le seguenti caratteristiche:

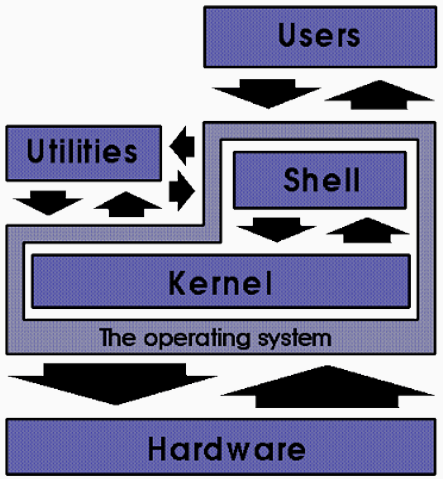
- ambiente di programmazione;
- semplice interfaccia utente;
- semplici utility che possono essere combinate per realizzare potenti funzioni;
- file system gerarchico (ad albero);
- semplice interfacciamento con i dispositivi;
- sistema *multi-utente* e *multi-processo*: più utenti possono collegarsi al sistema ed eseguire processi (istanze di programmi) contemporaneamente;
- architettura indipendente e trasparente all'utente.

Storia di Unix (2)

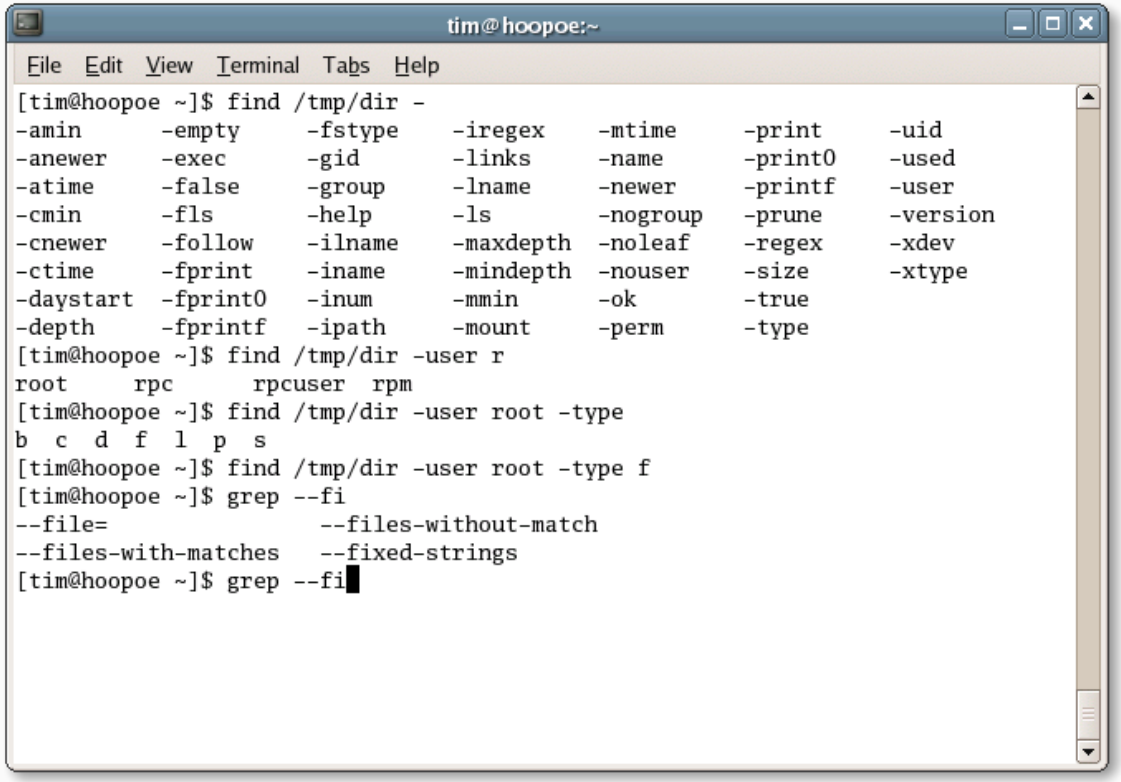
- Nel 1973 Unix è riscritto prevalentemente in **C**, un linguaggio di programmazione ad alto livello sviluppato da **Dennis Ritchie**.
- Dal 1974 Unix si diffonde prevalentemente in campo accademico grazie ad una licenza stipulata con le università per scopi educativi.
- **Come arriviamo a Linux? Richard Stallman** nel 1980 circa, iniziò a scrivere un sistema operativo chiamato GNU (GNU's Not Unix). Nel 1991 lo studente finlandese **Linus Torvalds** creò un kernel *unix-like* (conforme alla Single Unix Specification) e lo chiamò **Linux**. Il kernel Linux venne inserito dentro GNU dando vita così al sistema operativo libero GNU/Linux, più conosciuto come Linux.

Unix in generale

- Le funzionalità di Unix sono organizzate logicamente a *strati*;
- Il *Kernel* consente ai programmi utente l'accesso alle risorse fisiche (memoria, CPU, I/O);
- Il file system è una organizzazione gerarchica di file e directory (cartelle) il cui livello più alto è la **root** (o radice);
- I programmi utente interagiscono con il kernel attraverso un insieme di *system call* standard.



Cosa è la shell



- Programma che fornisce un'interfaccia testuale alle funzionalità del sistema;
- Legge i comandi digitati dall'utente e li esegue (ad es. navigare sul file system, creare file e directory, eseguire programmi).

Tante shell

I sistemi Unix offrono diverse shell, le seguenti sono offerte anche dalla console di Replit:

- **sh:** Bourne shell. La shell presente sui primi sistemi Unix.
- **bash:** shell di default per gli utenti Linux. Sarà la shell di riferimento in questo corso.
- **dash:** Una shell molto compatta usata in Debian e Ubuntu

Esistono anche **csh** e **tcsh**.

Il file /etc/shells contiene l'elenco delle shell installate.
`more /etc/shells` – per vedere le shells disponibili
`echo $SHELL` – per vedere la shell che state usando

Perche' usare una shell testuale?

- **Potenza e semplicità:** i comandi UNIX sono progettati per risolvere problemi specifici. Sono semplici (senza menù e opzioni nascoste) e proprio per questo potenti (ad es. `grep [<parola>] [<filename>]`).
- **Velocità e flessibilità:** è più veloce scrivere pochi caratteri da tastiera piuttosto che cercare un programma opportuno e usare le operazioni che fornisce sulla base delle proprie specifiche esigenze.
- **Accessibilità:** permette di accedere efficientemente a un sistema in remoto.

Sintassi dei comandi Unix

La sintassi tipica dei comandi UNIX è la seguente:

```
comando <opzioni> <argomenti>
```

- `<opzioni>` sono facoltative e influiscono sul funzionamento del comando. Generalmente consistono nel simbolo del "-" seguito da una sola lettera;
- `<argomenti>` si possono avere più argomenti o anche nessuno in base al comando.

Ogni comando richiede al kernel l'esecuzione di un'azione, ad es.

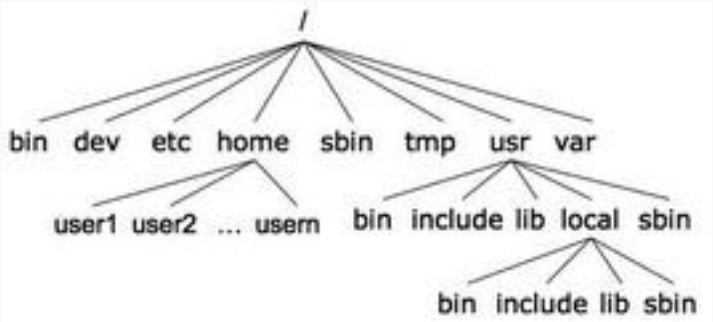
```
date
```

permette di stampare la data corrente.

Il file system (1)

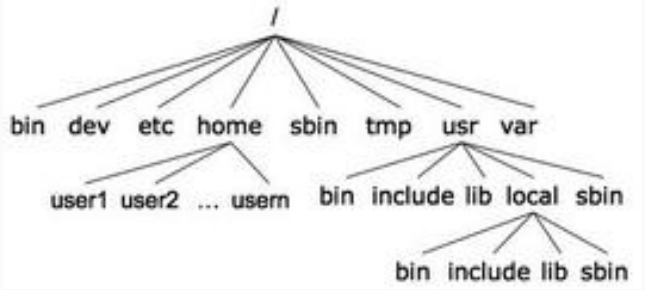
Un **file system** è il meccanismo fornito dal sistema operativo che regola l'organizzazione fisica e logica delle informazioni sui dispositivi (disco, cd-rom, dvd, ecc.).

In Unix, il file system può essere visto come un *albero*:



Omogeneità: tutto è un file (documenti, sorgenti di programmi, applicazioni, immagini,...). Tre categorie di file: *ordinari*, *directory* e *dispositivi*.

Il file system (2)



Directory di sistema che si ritrovano in tutti i sistemi unix-like:

- **bin:** file eseguibili tipicamente da tutti gli utenti
- **dev:** file speciali associati ai dispositivi (*device*)
- **etc:** file di configurazione
- **home:** directory che contiene le home directory degli utenti
- **sbin:** file eseguibili tipicamente dall'amministratore di sistema
- **var:** utilizzata per il logging e lo spooling

Il file system (3)

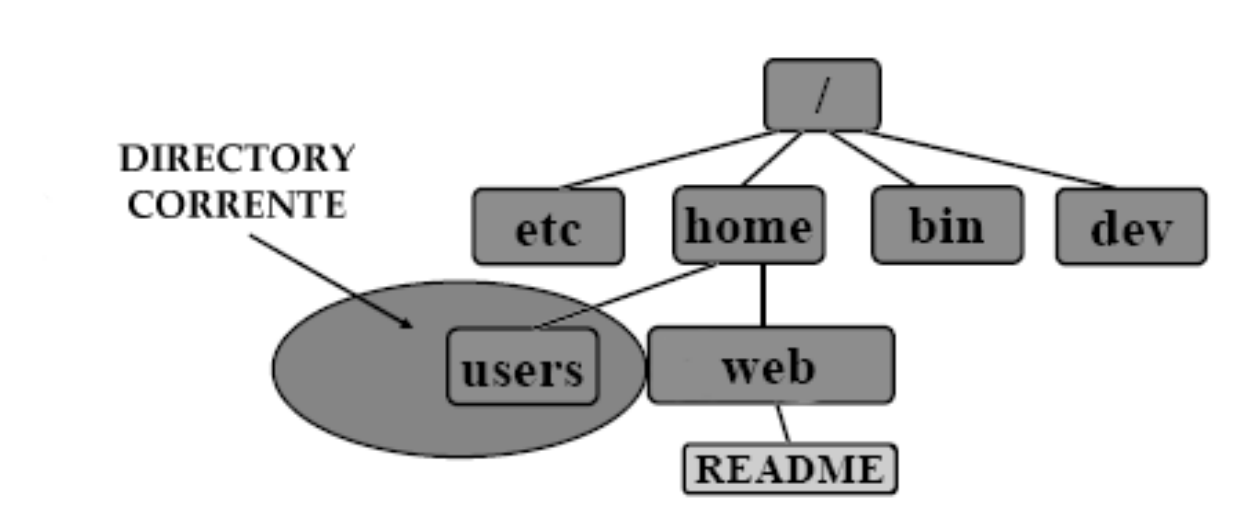
Ogni nodo dell'albero è o un file o una directory di file, dove quest'ultima può contenere altri file e directory.

- Un **file** è una sequenza non strutturata di bytes (unità logica di memorizzazione);
- Una **directory** è un file che indicizza altri file.

Un file, identificato da un **path name**, ha i seguenti attributi: tipo, permessi (diritti di accesso), nome utente proprietario, nome gruppo proprietario, dimensione, data di creazione, ultima modifica, ultimo accesso.

Il path name di un file o di una directory può essere **assoluto**, riferito alla radice della gerarchia (/), oppure **relativo**, riferito alla posizione dell'utente nel file system.

Path assoluti/relativi



PATH ASSOLUTO: `/home/web/README`

PATH RELATIVO: `../web/README`

Ulteriori informazioni sulla shell

- funzione di autocompletamento (tasto TAB);
- history (freccia SU/GIU).

Attenzione
I file system dei sistemi unix-like sono **case-sensitive**: maiuscole e minuscole sono importanti.

Esempio
file1, **File1**, **FILE1**, **FiLe1** sono tutti nomi di file diversi.

Comandi Unix

Navigare nel filesystem

`cd [<dir>]` serve per muoversi attraverso le directory.

Il parametro `<dir>` è opzionale — se non viene indicato, il comando porta nella home directory.

Esempio

- Supponiamo di voler accedere ai nostri documenti personali in `/home/user/documenti`
- supponiamo che la directory corrente sia la nostra home: `/home/user`
- per portarsi nella directory dei documenti eseguire: `cd documenti`
- per la navigazione risultano utili le directory: "`.`" (working directory), "`..`" (directory padre) e "`~`" (directory home).

Creare una directory

```
mkdir [-p] <dir1> ...  <dirN>
```

I parametri `dir` indicano i nomi (path assoluti o relativi) delle directory da creare.
Opzioni:
`-p` crea eventuali directory intermedie esplicitate nei parametri `dir`.

Esempio

- `mkdir temp` — crea directory `temp` nella directory corrente.
- `mkdir -p documenti/personali` — crea le directory `personali` dentro la directory `documenti` (se `documenti` non esiste viene creata).

Creare un file

```
touch nome_file.estensione
```

Vi appare subito sulla sinistra.

Esempio

- `touch prova.txt` — crea il file prova.txt nella cartella corrente

Visualizzare il contenuto di una directory

`ls [-alsFR] [<dir1> ... <dirN>]`

Se non viene specifica alcuna directory, si riferisce alla directory corrente.

Alcune opzioni:
`-a` visualizza anche i file nascosti (il loro nome inizia per ".")
`-l` visualizza informazioni estese sui file (ad es. permessi, dimensione, owner, group)
`-s` visualizza la dimensione in bytes
`-F` aggiunge un carattere finale al nome del file che ne denota il tipo (ad es. "nome/" indica una directory)
`-R` visualizza ricorsivamente le sottodirectory (esegue ls ricorsivamente sulle subdir)

Word Count

`wc [-wci] <file>`

Esempio

- `wc -w prova.txt` conta le parole in `prova.txt`
- `wc -c prova.txt` conta i caratteri in `prova.txt`
- `wc -l prova.txt` conta le linee in `prova.txt`

Visualizzare il contenuto di un file

```
cat [-nve] <file1> ...  <fileN>
```

Opzioni:
`-n` precede ogni linea con un numero
`-v` visualizza i caratteri non stampabili eccetto newline, tab e form-feed
`-e` visualizza $ alla fine di ogni linea (quando usato insieme con l'opzione `-v`)

`cat file1 file2 file3` concatena il contenuto dei file seguendo lo stesso ordine di immissione e ne mostra il contenuto

altro comando: `more <file>`

Copiare file

```
cp [-if] <file1> <file2>
```

copia `file1` in `file2` — se `file2` esiste viene sovrascritto!

```
cp [-if] <file1> ...  <fileN> <dir>
```

copia i `file` nella directory `dir` — se un `file` esiste in `dir` viene sovrascritto!

Opzioni:
`-i` chiede conferma prima di sovrascrivere
`-f` non chiede conferma prima di sovrascrivere

Spostare file

```
mv [-if] <file1> <file2>
```

sposta `file1` in `file2` — se `file2` esiste viene sovrascritto!

```
mv [-if] <file1> ...  <fileN> <dir>
```

sposta i `file` nella directory `dir` — se un `file` esiste in `dir` viene sovrascritto!

Opzioni:
`-i` chiede conferma prima di sovrascrivere
`-f` non chiede conferma prima di sovrascrivere

Eliminazione di file

```
rm [-rif] <file1> ...  <fileN>
```

Opzioni:

`-r <dir>` cancella la directory con il suo contenuto

`-i` prima di cancella il file chiede conferma all'utente

`-f` cancella senza chiedere conferma

Eliminare una directory (vuota)

```
rmdir [-p] <dir1> ...  <dirN>
```

I parametri `dir` indicano i nomi (pathname assoluti o relativi) delle directory da eliminare.
Opzioni:
`-p` elimina eventuali directory intermedie esplicitate nei pathname dei parametri `dir`.

Esempio

- `rmdir temp` — elimina la directory `temp` se è vuota
- `rmdir -p documenti/personali` — elimina le directory `personali` e `documenti`, se entrambe vuote

I metacaratteri (wildcards)

La shell Unix riconosce alcuni caratteri speciali, chiamati **metacaratteri**, che possono comparire nei comandi.
I più comuni sono:

`?`	`qualunque carattere`
`*`	`qualunque sequenza di caratteri`

Esempio
Supponiamo di voler copiare tutti i file `.html` di una directory nella sotto-directory `html-src`. Usando la wildcard `*` (asterisco) si può scrivere semplicemente:

```
cp *.html html-src
```

Nomi di file e convenzioni

- Esistono precise regole che stabiliscono i nomi con cui possono venire chiamati file e directory;
- Nomi con caratteri come /, *, & e % devono essere evitati per evitare possibili errori di sistema;
- Anche utilizzare nomi composti da parole divise da spazi non è una buona abitudine;
- Nominare file o directory usando solo caratteri alfanumerici, lettere e numeri, uniti insieme da _ (underscore) e . (punti).

Il comando echo

Il comando `echo` stampa sullo schermo la stringa passata come parametro al comando.

Esempi

```
$ echo Ciao!
Ciao!

$ ls
data-new data1 data2 inittab esempio1.txt

$ echo data*
data-new data1 data2

$ echo data?
data1 data2
```

Redirezione

Di default i comandi Unix prendono l'input da tastiera (**standard input - stdin**) e mandano l'output ed eventuali messaggi di errore su video (**standard output - stdout**, **standard error - stderr**). L'input/output in Unix può essere rediretto da/verso file, utilizzando opportuni metacaratteri:

Metacarattere	**Significato**
>	ridirezione dell'output
>>	ridirezione dell'output (append)
<	ridirezione dell'input
<<	ridirezione dell'input dalla linea di comando

Redirezione — Esempi

Esempi

```
$ echo Topolino > file.txt
$ cat file.txt
Topolino
$ echo Pippo >> file.txt
$ cat file.txt
Topolino
Pippo

$ sort < file.txt
Pippo
Topolino
```

I Processi

Un processo è un programma in esecuzione.

La **shell esegue ripetutamente i seguenti passi**:

- stampa il prompt e attende l'input dell'utente;
- legge la linea di comando e espande le eventuali alias e wildcard;
- lancia un processo per eseguire il comando mettendosi in attesa;
- quando l'esecuzione del comando termina, riprende il controllo.

Poiché Unix è un sistema multitasking, la shell permette di lanciare più processi in parallelo (con &, ma noi non lo vedremo...).

Comandi per la gestione dei processi

- `ps`: elenca i processi (con rispettivi pid) della shell corrente; con opzione `- aux` elenca tutti i processi in esecuzione
- `CTRL-z`: combinazione di tasti che sospende il comando in esecuzione
- `CTRL-c`: combinazione di tasti che termina il comando in esecuzione
- `kill -signal_name <p>`: invia il segnale `signal_name` al processo con pid `<p>`; 9 è il segnale che termina un processo

Altri comandi utili (1)

- `comando --help` — mostra il significato del comando e tutte le opzioni
- `pwd` (**print working directory**) — visualizza il percorso assoluto della directory corrente
- `head` — visualizza le prime linee di un file di testo
  ad es. `head -10 esempio.txt` visualizza le prime 10 righe di `esempio.txt`
- `tail` — visualizza le ultime linee di un file di testo
  ad es. `tail -10 esempio.txt` — visualizza le ultime 10 righe di `esempio.txt`
- `sort` — ordine le linee di un file di testo lessicograficamente
  ad es. `sort esempio.txt` — ordina le righe di `esempio.txt`

Altri comandi utili (2)

- `gzip/gunzip` — compressione/decompressione di file
  ad es. `gzip esempio.txt` — ottengo il file compresso `esempio.txt.gz`
- `bzip2/bunzip2` — compressione/decompressione di file
- `tar` — creazione/estrazione da archivi
- `zip/unzip` e `rar/unrar` — creazione e estrazione di archivi compressi
- `file <nome>` — visualizza il tipo del file `<nome>`, ad es. `file lezione1.pdf` — stampa
  `lezione1.pdf: PDF document, version X.X`

Esercitazione

Esercizio 1

- Cliccate sulla `unit fermami di Lab1`;
- Posizionatevi sulla shell ed eseguite le seguenti operazioni usando solo la shell (ad es. senza modificare file attraverso l'editor o file manager di Replit);
- Creare una directory `temp`;
- Entrare nella directory appena creata;
- Creare due sotto-directory `sorgente` e, come sotto livello, `destinazione` (`destinazione` sarà una sottodirectory di `sorgente`);
- Creare nella directory sorgente un file di nome `esempio.txt`;
- Editare il file e scrivere all'interno del file la riga "contenuto";
- Controllare da shell il percorso assoluto della directory corrente (`sorgente`) e scriverlo (append) nel file `esempio.txt`.

Esercizio 2

- Posizionatevi (se non ci siete già) all'interno della directory `sorgente`;
- Cancellate il file `esempio.txt` creato durante l'esercizio precedente (attenzione il file non è vuoto);
- Create un nuovo file di testo `lista1.txt` ed inserite all'interno 5 nomi di amici;
- Create un nuovo file di test `lista2.txt` ed inserite all'interno 5 nomi di amici;
- Muovi il file `lista1.txt` dalla directory `sorgente` alla directory `destinazione`;
- Copia il file `lista2.txt` dalla directory `sorgente` alla directory `destinazione`.

Esercizio 3

- Posizionatevi all'interno della directory `destinazione`;
- Visualizzate tutti i file contenuti nella directory corrente;
- Concatenare i due file nel file di destinazione `lista3.txt` e visualizzare il risultato.

Esercizio 4

- Ordinate il file `lista3.txt` dell'esercizio precedente salvando il suo contenuto nel file `lista3Ord.txt`;
- Visualizzate le prime 5 linee del file appena creato.

Esercizio 5

- Eseguire sulla shell il comando cd (che vi riporta alla home directory)
- Eseguire sulla shell il comando `./fermami`
- Interrompere l'esecuzione del comando in modo da riottenere il prompt (`CTRL-C`).

Compilazione

Dal sorgente all'eseguibile

Prima di poter essere eseguito, un programma deve essere

1. pre-processato (pre-processing)
2. compilato (compiling)
3. collegato (linking)
4. caricato in memoria (loading)

I file testuali .c sono parte del codice sorgente di un programma

- Il compilatore trasforma il sorgente .c in codice oggetto .o (binario)
- Il linking collega i file oggetto in un eseguibile

In questa fase ci concentriamo su pre-processore e compilatore

Pre-processing

- Fase preliminare alla compilazione
- Comporta la sostituzione di informazioni simboliche (testo) nel codice sorgente con un contenuto che viene specificato dal programmatore mediante direttive per il pre-processore
- Le direttive per il pre-processore vanno scritte in testa ai sorgenti C e sono precedute dal simbolo #
  - Inclusione file: `#include`
  - Macro: `#define`

Il compilatore

Trasforma il file preprocessato (senza più `#include` o `#define`) in un file eseguibile che contiene il codice assembler eseguibile dal processore target

Il compilatore C esegue una serie di controlli di correttezza

- Sintassi dei comandi: ad es. terminazione con ";", parentesi bilanciate, ...
- Coerenza dei tipi di dato: ad es. operatori algebrici usati solo con variabili numeriche, parametri delle funzioni,...
- Ogni variabile o funzione utilizzata deve essere stata dichiarata in precedenza

GNU Compiler Collection (GCC)

- Sviluppato per sistemi Linux permette generare eseguibili per diverse piattaforme target
- Non è solo un compilatore ma è anche pre-processore e linker

Sintassi:

```
gcc [OPT] file.c
```

- `[OPT]` - Insieme di opzioni che controllano le funzionalità di GCC (le vediamo dopo)
- `file.c` - Nome del file sorgente
- il file eseguibile sarà memorizzato nel file `a.out`
- per vedere il risultato dell'esecuzione dobbiamo eseguire `a.out` con il comando `./a.out`

Opzioni importanti di GCC

- `-o fileExec` - Imposta il nome del file eseguibile a `fileExec` (`a.out` di default)
- `-Wall -pedantic` opzioni che aumentano il numero di controlli e di messaggi di avvertimento (warnings) visualizzati

Esercizio 6

- Compilare il file `main.c` che si trova nella unit CompilingExercise memorizzando il codice oggetto in `main`;
- Eseguire il programma `main`