Fondamenti di programmazione

2 | Introduzione al linguaggio C

SOMMARIO

- Linux
- File system
- Shell
- Hello World in C

Linux – caratteristiche principali

Lo sviluppo di **Linux** è strettamente legato al **linguaggio C**.

Il linguaggio C, infatti, venne inizialmente sviluppato (nel 1972 da Brian Kernighan e Dennis Ritchie) per poter essere utilizzato nella stesura del *kernel* del sistema operativo ***Unix***.

Anche se il C nasce come un linguaggio per lo sviluppo di sistema, oggi viene utilizzato per lo sviluppo di software di diversa natura.

In genere, programmi scritti in C sono molto più **efficienti** rispetto ad altri linguaggi.

Linux – caratteristiche principali

Linux è un ***kernel*** (nucleo fondamentale di un sistema operativo) creato nel 1991

Il kernel è responsabile dell'esecuzione dei ***processi***, della gestione della memoria, gestione delle risorse hardware ecc.

Quando si parla di ***distribuzioni Linux***, ci si riferisce a sistemi operativi che hanno il kernel Linux.

Ogni distribuzione viene customizzata con software applicativi differenti, a seconda di quelli che sono gli usi e le caratteristiche che dovrà avere il sistema stesso (basse richieste hardware, alta efficienza…)

File system

Il file system è l'insieme delle funzionalità fornite dal sistema operativo che permettono l'organizzazione *fisica* e *logica* delle informazioni sui dispositivi

Un **file** è un *contenitore* di dati digitali memorizzati su un supporto fisico ed organizzate secondo una determinata struttura

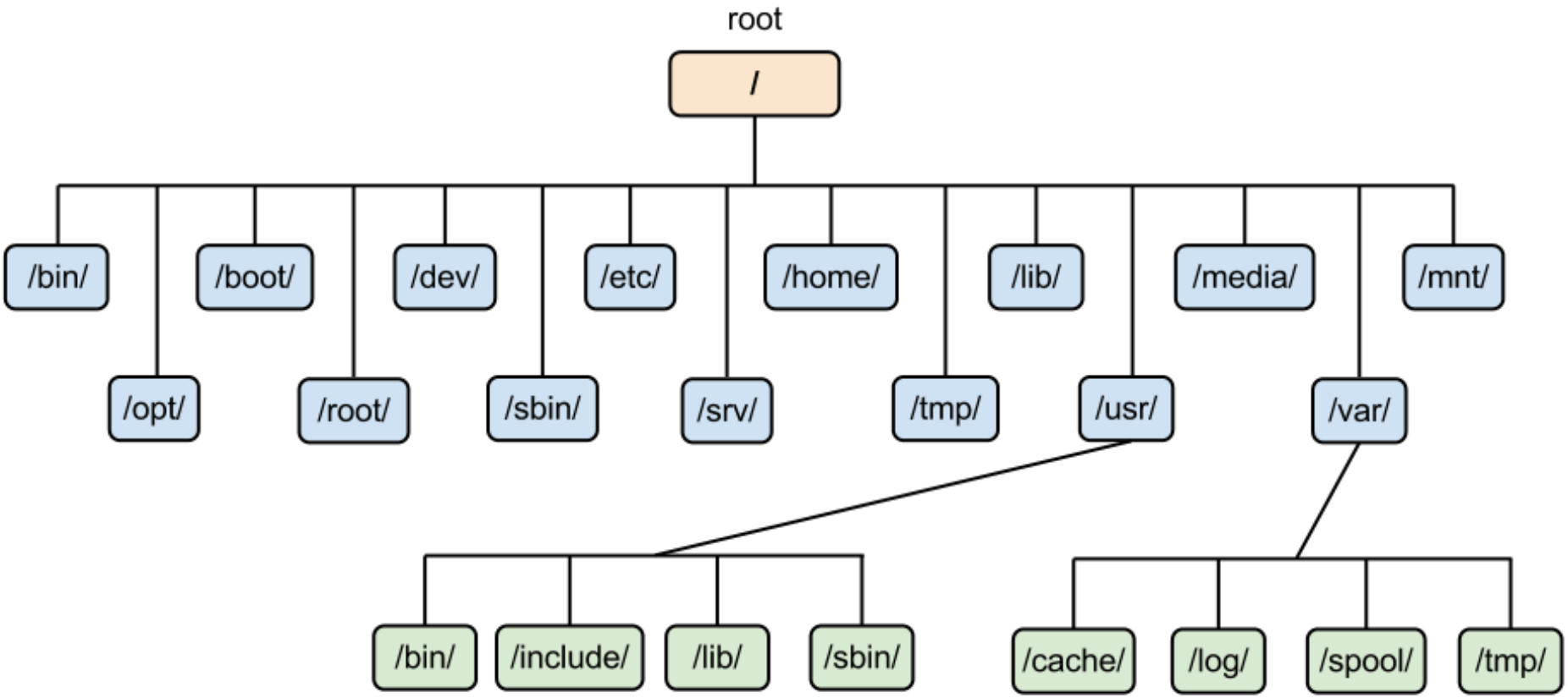
Con **directory** si intende un tipo particolare di file che contiene dei *riferimenti* ad altri file. Può essere visto come un *contenitore di file*

Il file system organizza e gestisce la struttura dei file. In particolare permette:

- Creazione e cancellazione di file
- Creazione e cancellazione di directory
- Modifica di file
- …

File system

Nei sistemi Linux, il file system è organizzato secondo una struttura gerarchica.



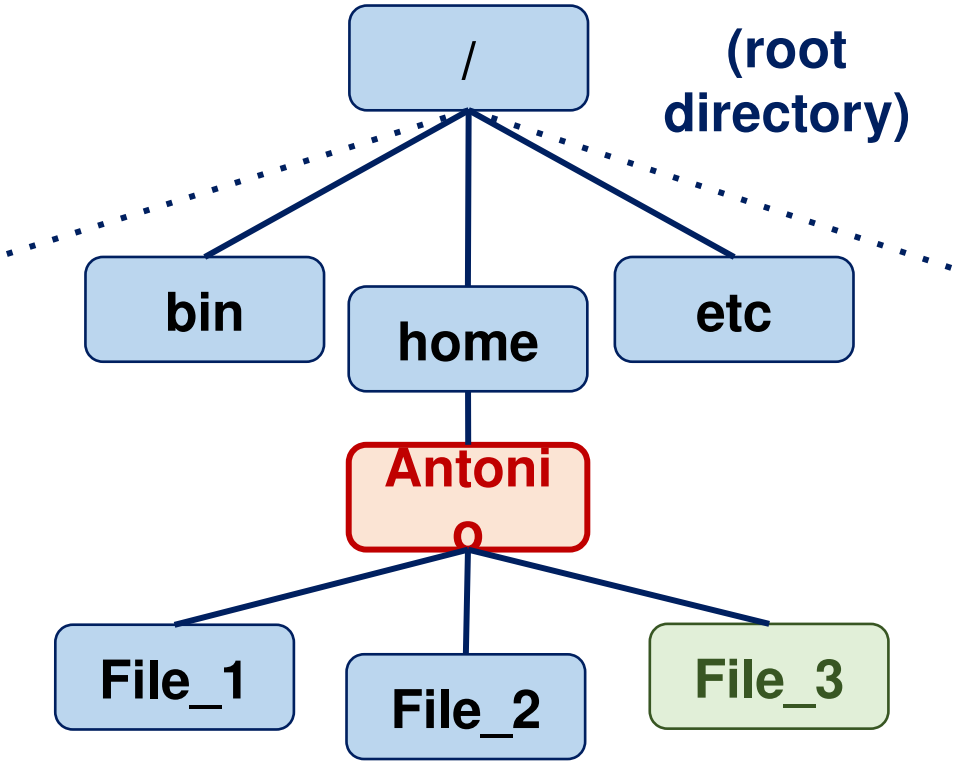
File system Linux

- / **(root directory):** punto iniziale del file system.
- /**bin**: comandi di base. Accessibili sia a *root* che a tutti gli utenti
- /**home**: contiene le *home* di tutte gli utenti.
- /**etc**: contiene file di configurazione di programmi di sistema e script di avvio
- /**dev**: contiene file che rappresentano dispositivi hardware (dischi, lettori…)
- /**mnt**: directory in cui vengono *montati* altri file system
- /**tmp**: directory che il sistema usa per la memorizzazione dei file temporanei
- /**var**: vengono scritte informazioni generate da programmi (log, errori…)
- /**lib**: librerie di sistema

File system Linux

Per individuare un file all'interno del file system, bisogna specificare il **path (percorso)** che può essere **assoluto** o **relativo**.

Supponendo di lavorare nella directory della propria home e di voler individuare il file *File_3*:



Path *assoluto*
/home/Antonio/File_3

Path *relativo*
./File_3

Ogni directory contiene 2 directory speciali:

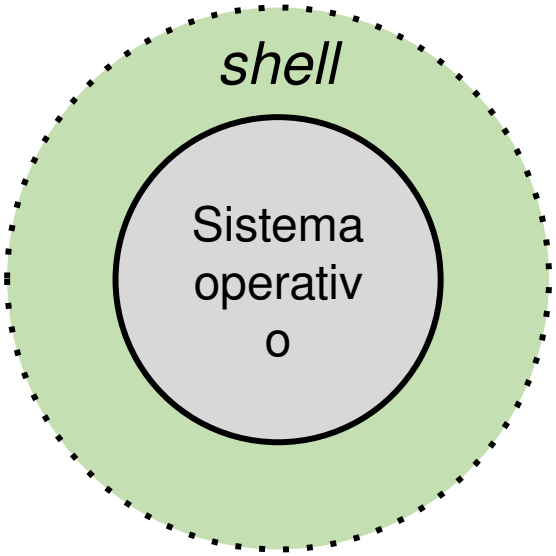
- **. (punto)** : riferimento alla directory corrente
- **.. (punto punto)**: riferimento alla directory *padre.* (in questo caso, **..** individua la directory /home)

Shell

La ***shell*** è la componente del sistema operativo che permette di interagire con il sistema stesso.

E' un interprete che, ricevuti i comandi dall'utente, cerca di interpretarli ed eseguirli

Assieme al Kernel, è una parte fondamentale del sistema operativo e viene considerata l'*involucro*, la parte visibile del sistema ed è dunque definibile come l'interfaccia utente (cioè l'unico modo che ha l'utente di interagire con il sistema operativo)



Shell – principali comandi

I comandi che la shell può *eseguire* sono presenti nel file system come file binari e, almeno i principali, sono eseguibili da tutti gli utenti (molti si trovano in ***/bin***)

I comandi possono ricevere dei *parametri.* La sintassi è la seguente:

comando <opzioni> <argomenti>

- **<opzioni>** sono facoltativi e influenzano il funzionamento del programma. In genere vengono specificati facendoli seguire al carattere ‘’**-**’’
- **<argomenti>** non necessariamente devono essere presenti. Indicano all’opzione precedente qualche informazione in più

Shell – principali comandi

Comando	Descrizione
pwd	Mostra il path assoluto della directory corrente
ls [-options] <dir$_1$>.. <dir$_N$>	Mostra il contenuto della directory corrente
cd [<dir>]	Cambia directory
rm [-options] <dir$_1$> … <dir$_N$>	Rimuovi file/directory
mkdir [-options] <dir$_1$> … <dir$_N$>	Crea Directory
cp [-options] <file> <dir>	Copia file
mv [-options] <file> <dir>	Sposta file
cat [-options] <file$_1$> … <dir$_N$>	Visualizza il contenuto del file

man <*nomeComando*>

Visualizza il manuale relativo a *nomeComando*

Hello world in C

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n");

    return 0;
}
```

Hello world in C

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n");

    return 0;
}
```

- **#include <stdio.h>** E' una ***direttiva di preprocessore*** (verranno trattate più avanti)

  Indica al compilatore di ***includere*** altri file nel processo di **compilazione**.

Con questo meccanismo è possibile utilizzare codice definito altrove (librerie che mette a disposizione il linguaggio, librerie definite dall'utente o librerie esterne)

Hello world in C

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n");

    return 0;
}
```

- **#include <stdio.h>** E' una ***direttiva di preprocessore*** (verranno trattate più avanti)

  Indica al compilatore di ***includere*** altri file nel processo di **compilazione**.

Con questo meccanismo è possibile utilizzare codice definito altrove (librerie che mette a disposizione il linguaggio, librerie definite dall'utente o librerie esterne)

MODULARITA'

Hello world in C

```
1 #include <stdio.h>
2
3 int main(void){
4     printf("Hello world\n");
5
6     return 0;
7 }
```

- **#include <stdio.h>** In questo caso, si sta includendo nel processo di compilazione il file di nome ***stdio.h*** (si vedrà successivamente cosa sono i file **.h**).

  Per il momento può bastare sapere che il file **stdio.h** (Standard Input/Output) *contiene* tutte le funzioni di **input/output,** fra le quali anche la ***printf,*** utilizzata alla linea 4.

Hello world in C

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n");

    return 0;
}
```

- **int main(void)** Il ***main*** è la funzione principale di ogni programma in C.

  E' la funzione che rappresenta l'***entry point*** del programma, cioè il punto dal quale inizia il flusso del programma.

  Il main **deve** essere presente in un programma C e **deve** essere unico (non è possibile definire più funzioni main!)

Hello world in C

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n");

    return 0;
}
```

- **int main(void)** L' **int** indica che la funzione, una volta che viene invocata, ritorna un valore di tipo ***intero.*** Può essere considerato come il valore di **output** della funzione. Verrà approfondito di seguito

  Il **void** indica che, **in questo caso**, non vengono passati *dati in input* (***parametri***) alla funzione main. In altri casi, il main potrebbe ricevere parametri che gli vengono passati da **riga di comando**

Hello world in C

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n");

    return 0;
}
```

- **{ . . . . . }**

Le **parentesi graffe** hanno la funzione di racchiudere un **blocco di istruzioni *(statement)***, che in questo caso rappresenta anche il **corpo della funzione main**.

Le istruzioni contenute in una funzione devono essere sempre racchiuse fra le parentesi graffe.

Hello world in C

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n");

    return 0;
}
```

- **printf(“Hello world\n”);**

La funzione **printf** è definita all’interno del file **stdio.h**

E’ una funzione di **output** che stampa a schermo la ***stringa*** (sequenza di caratteri) specificata.

Il **;** (punto e virgola) finale è necessario e indica la fine di un’**istruzione**.

Hello world in C

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n");

    return 0;
}
```

- **printf(‘’Hello world\n’);** Il caratere \ **(backslash)** viene chiamato **carattere di escape** e modifica il comportamento della printf.

Sequenza di escape	Descrizione
\n	Carattere di *new line.* Inserisce un ritorno a capo
\t	Inserisce un carattere di tabulazione
****	Inserisce il carattere *backslash*
\"	Inserisce i *doppi apici* (evita che vengano intesi come fine stringa

Hello world in C

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n");

    return 0;
}
```

- **return 0;** Valore ritornato dalla funzione main.

  Valori di ritorno e **parametri** verranno spiegati nel dettaglio quando verranno presentate le **funzioni**.

Esercizio 0 - creazione di un file

Prima di iniziare con la realizzazione di un programma completo, è opportuno creare una directory dedicata nella quale si andranno a salvare i codici sorgenti.

Per far ciò, i comandi da eseguire nel terminale sono i seguenti:

> **cd** /home/*nomeUtente*/Scrivania	ci si posiziona nella propria Scrivania (desktop)
> **mkdir** *directoryProgrammi*	crea una directory chiamata directory *Programmi* nella scrivania
> **cd** *directoryProgrammi*	ci si sposta nella directory directory *Programmi* appena creata
> **touch** *helloWorld.c*	crea un file chiamato *primoProgramma.c* L'estensione **.c** è arbitraria ma consigliata
> **ls** -lh	stampa a schermo la lista dei file presenti nella directory corrente. Dovrebbe essere presente il file appena creato

Esercizio 1 - HelloWorld

Se il file (slide precedente) è stato creato correttamente, aprirlo con un editor di testo (***gedit*** nella versione di Ubuntu installata) e scrivere il codice di esempio mostrato di seguito

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n");

    return 0;
}
```

> **gcc** *helloWorld.c* **-o** helloWorld.out ⟶ Compila il file sorgente di nome *helloWorld.c* e genera in output il file *helloWorld.out*
> L'opzione **-o** è opzionale (se non specificato, il file di output sarà ***a.out***)

> **./helloWorld.out** ⟶ Esegue il programma.
> **N.B. ./** indica che si sta *accedendo* alla directory corrente.
> A schermo viene stampato l'output del programma

Esercizio 2 - Wrong HelloWorld

Provare a compilare il codice sorgente seguente. Cosa succede in fase di compilazione?

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n")

    return 0;
}
```

> **gcc** *helloWorld.c* **-o** helloWorld.out

Esercizio 2 - Wrong HelloWorld

Provare a compilare il codice sorgente seguente. Cosa succede in fase di compilazione?

```c
#include <stdio.h>

int main(void){
    printf("Hello world\n")

    return 0;
}
```

> **gcc** *helloWorld.c* **-o** helloWorld.out

Esercizio 2 - Wrong HelloWorld

Provare a compilare il codice sorgente seguente. Cosa succede in fase di compilazione?

```c
#include <stdio.h>

int main(void){
```

```
antonio@mint-antonio:~$ gcc n.c
n.c: In function ‘main’:
n.c:6:2: error: expected ‘;’ before ‘return’
  return 0;
  ^~~~~~
antonio@mint-antonio:~$
```

> **gcc** *helloWorld.c* **-o** helloWorld.out ⟶ **ERRORE!!**

Esercizio 3 - Wrong HelloWorld

Provare a compilare il codice sorgente seguente. Cosa succede in fase di compilazione?

```c
#include <stdio.h>

int my_main(void){
    printf("Hello, world\n");

    return 0;
}
```

> **gcc** *helloWorld.c* **-o** helloWorld.out

Esercizio 3 - Wrong HelloWorld

Provare a compilare il codice sorgente seguente. Cosa succede in fase di compilazione?

```c
1 #include <stdio.h>
2
3 int my_main(void){
```

```
antonio@mint-antonio:~$ gcc n.c
/usr/lib/gcc/x86_64-linux-gnu/7/../../../x86_64-linux-
gnu/Scrt1.o: nella funzione "_start":
(.text+0x20): riferimento non definito a "main"
collect2: error: ld returned 1 exit status
antonio@mint-antonio:~$
```

> **gcc** *helloWorld.c* **-o** helloWorld.out ⟶ **ERRORE!!**