Linux e terminali

Samuele Musiani, Alice Benatti

Università di Bologna, corso di Laurea in Informatica

12 ottobre 2023

Shell e terminali

Cos'è una shell

Una `shell` è un programma che permette di parlare con il sistema operativo attraverso dei comandi da tastiera.

Cos'è una shell

Una `shell` è un programma che permette di parlare con il sistema operativo attraverso dei comandi da tastiera.

```
[samu@leibniz ~]$
```

Figura: Esempio di un *shell prompt*

Perché tornare all'età della pietra

I primi computer utilizzavano i terminali, perché dovremmo farlo anche noi quando abbiamo un'ottima interfaccia grafica?

Perché tornare all'età della pietra

I primi computer utilizzavano i terminali, perché dovremmo farlo anche noi quando abbiamo un'ottima interfaccia grafica?

"Graphical user interface make easy taks easy, while command line iterfaces make difficult tasks possible"

Vantaggi di un terminale

I primi computer utilizzavano i terminali, perché dovremmo farlo anche noi quando abbiamo un'ottima interfaccia grafica?

- È decisamente più veloce di una GUI

Vantaggi di un terminale

I primi computer utilizzavano i terminali, perché dovremmo farlo anche noi quando abbiamo un'ottima interfaccia grafica?

- È decisamente più veloce di una GUI
- Avete il completo controllo di quello che state facendo

Vantaggi di un terminale

I primi computer utilizzavano i terminali, perché dovremmo farlo anche noi quando abbiamo un'ottima interfaccia grafica?

- È decisamente più veloce di una GUI
- Avete il completo controllo di quello che state facendo
- Moltissime cose non si possono fare con una GUI

Vantaggi di un terminale

I primi computer utilizzavano i terminali, perché dovremmo farlo anche noi quando abbiamo un'ottima interfaccia grafica?

- È decisamente più veloce di una GUI
- Avete il completo controllo di quello che state facendo
- Moltissime cose non si possono fare con una GUI
- Il vero potere della `shell` sono le *pipe*

Vantaggi di un terminale

I primi computer utilizzavano i terminali, perché dovremmo farlo anche noi quando abbiamo un'ottima interfaccia grafica?

- È decisamente più veloce di una GUI
- Avete il completo controllo di quello che state facendo
- Moltissime cose non si possono fare con una GUI
- Il vero potere della `shell` sono le *pipe*
- Accedere a server e configurare servizi

I primi passi in un terminale

Aprire un terminale

Per utilizzare una `shell` è necessario disporre di un emulatore di terminale.

Un emulatore di terminale è un'applicazione che permette di interagire con la `shell`.

Digitare e leggere

Proviamo a digitare delle lettere a caso nella nostra `shell`:

```
[samu@leibniz ~]$ fasdjfivb
bash: fasdjfivb: command not found
[samu@leibniz ~]$
```

Figura: Esempio di lettere digitate a caso in una *shell*

Digitare e leggere

Proviamo a digitare delle lettere a caso nella nostra `shell`:

```
[samu@leibniz ~]$ fasdjfivb
bash: fasdjfivb: command not found
[samu@leibniz ~]$
```

Figura: Esempio di lettere digitate a caso in una *shell*

La `shell` non ha riconosciuto il comando e ce lo ha scritto a schermo.

Digitare e leggere

Proviamo ora a digitare un comando esistente:

```
[samu@leibniz ~]$ date
Thu Set 7 12:15:58 PM CEST 2023
[samu@leibniz ~]$
```

Figura: Esempio di lettere digitate a caso in una *shell*

Digitare e leggere

Proviamo ora a digitare un comando esistente:

```
[samu@leibniz ~]$ date
Thu Set 7 12:15:58 PM CEST 2023
[samu@leibniz ~]$
```

Figura: Esempio di lettere digitate a caso in una *shell*

In questo caso la `shell` ha riconosciuto il comando e chi ha stampato la data, l'ora e qualche altra informazione come il fuso orario.

Quali comandi esistono

Come abbiamo appena visto alcuni comandi esistono e altri no.

Non vi è una lista dei comandi esistenti perché sarebbe troppo lunga.

Oggi vi mostreremo alcuni comandi di base per iniziare a familiarizzare con la `shell`.

Muoversi tra i file e le cartelle

Elencare i file

Le operazioni più importanti sono legate alla gestione dei file. Proviamo a digitare il comando `ls` nella shell.

```
[samu@leibniz ~]$ ls
Desktop Documents Downloads Music Pictures
Public  Templates Video
[samu@leibniz ~]$
```

Apriamo ora il programma per gestire i file a livello grafico e vediamo che solo gli stessi.

Entrare in una cartella

Dall'interfaccia grafica possiamo entrare nelle cartelle. Con la `shell` possiamo farlo attraverso il comando `cd`.

Per farlo però è necessario indicare in quale cartella ci vogliamo muovere visto che potrebbe essercene più di una.

Per farlo è quindi necessario fornire al comando `cd` un **argomento**.

```
[samu@leibniz ~]$ cd Documents
[samu@leibniz ~/Documents]$ cd Documents
```

Entrare in una cartella

```
[samu@leibniz ~]$ cd Documents
[samu@leibniz ~/Documents]$ cd Documents
```

L'argomento è successivo al comando. Un modo per pensare a questo ordine è chiedersi:

- "Cosa voglio fare?" -> Entrare di una cartella -> `cd`
- "In quale cartella voglio entrare?" -> `cd` *nomecartella*

Uscire da una cartella

Per uscire da una cartella il comando è `cd ..`

```
[samu@leibniz ~/Documents]$ cd ..
[samu@leibniz ~]$
```

L'argomento `..` indica sempre la cartella precedente a quella attuale.

Current working directory

In Linux le cartelle si chiamano *directory*. Esiste un comando per stampare il *path* della cartella corrente: `pwd`.

Il path assume la seguente forma: `/home/samu`.

Il carattere / viene utilizzato come separatore.

Current working directory

Il path assume la seguente forma: `/home/samu`.

Notiamo che il *path* inizia con uno /

In Linux la radice del file-system è proprio /

Linux file system

Dove tutto ha inizio

Per fare riferimento a uno file è necessario identificarlo con un *path*.

I *path* sono unici, non possono esserci quindi file diversi con lo stesso *path*.

Ne consegue che se due file hanno lo stesso *path* sono lo stesso file.

Ogni *path* in linux inizia sempre dalla radice del file-system, ovvero /

Tutte le cartelle presenti in / sono riservate al sistema e al suo corretto funzionamento

Cartelle di sistema

```
[samu@leibniz /]$ ls
bin   dev   home   lib64        mnt   proc   run    tmp   var
boot  etc   lib    lost+found   opt   root   sbin   sys   usr
[samu@leibniz /]$
```

Figura: Lista dei file presenti in /

- `/bin`: contiene programmi necessari al sistema per funzionare
- `/boot`: contiene il kernel e altri file necessari al sistema per partire.
- `/etc`: contiene tutti i file di configurazione del sistema.
- `/home`: contiene le cartelle riservate agli utenti
- `/tmp`: contiene file temporanei che vengono cancellati ad ogni spegnimento del sistema
- `/usr`: contiene programmi e file usati dagli utenti

Path relativi e assoluti

Per identificare un file è possibile usar due tipi di *path*: assoluto e relativo:

1. Un *path* assoluto è un percorso ad un file che inizia da / e termina con il nome di quel file. prende quindi le seguenti forme:
   - `/home/samu/slides.tex`
   - `/usr/bin/firefox`
   - `/tmp`
2. Un *path* relativo è il percorso necessario per raggiungere un file rispetto alla **cartella corrente**. prende quindi le seguenti forme:
   - `slides.tex`
   - `../`
   - `immagini/gattini.png`

Creare directory e file

Creare directory

Per creare una directory esiste il comando `mkdir`.

```
[samu@leibniz prova]$ ls
castoro  ligma  piante
[samu@leibniz prova]$ mkdir ippo
[samu@leibniz prova]$ ls
castoro  ippo  ligma  piante
[samu@leibniz prova]$ cd ippo
[samu@leibniz ippo]$
```

Figura: esempi del comando mkdir

Sintassi: `mkdir path/to/directory`

Creare file

Per creare un file esistono molti modi, ma il più semplice è il comando `touch`.

```
[samu@leibniz prova]$ ls
castoro  ligma  piante
[samu@leibniz prova]$ touch sedia
[samu@leibniz prova]$ ls
castoro  ligma  piante  sedia
[samu@leibniz prova]$
```

Figura: esempi del comando mkdir

Sintassi: `touch path/to/file`

Chiarimenti su touch

Il comando `touch` in realtà server per cambiare la data di modifica di un file.

Se il file non esiste allora viene creato.

Esistono altri modi di creare file: tipo attraverso `vim` che vi permette di scrivere anche dentro il file.

Interagire con file

A chi serve nautilus?

Per sapere di che tipo è un file possiamo usare il comando `file`:

```
[samu@leibniz ~]$ file Documents/
Documents/: directory
[samu@leibniz ~]$ file screen.png
screen.png: PNG image data, 3840x1080, 8-bit/color RGB
[samu@leibniz ~]$ file lista
lista: Unicode text, UTF-8 text
[samu@leibniz ~]$ file /bin/firefox
/bin/firefox: POSIX shell script, ASCII text executable
[samu@leibniz ~]$
```

Figura: esempi del comando file

In linux i file non hanno bisogno di un'estensione, è quindi molto utile questo comando per determinare il tipo di un file

Copiare file

Il comando per copiare dei file è cp. il suo utilizzo è principalmente: `cp file/da/copiare destinazione`

```
[samu@leibniz prova]$ ls
castoro  ligma  piante
[samu@leibniz prova]$ cp castoro criceto
[samu@leibniz prova]$ ls
castoro  criceto  ligma  piante
[samu@leibniz ~]$
```

Figura: esempi del comando cp

Sono ammessi sia *path* assoluti sia relativi

Spostare file

Il comando per spostare (tagliare) dei file è `mv`. il suo utilizzo è pressoché identico al comando di copia: `mv file/da/muovere destinazione`

```
[samu@leibniz prova]$ ls
castoro  criceto  ligma  piante
[samu@leibniz prova]$ cd piante/
[samu@leibniz prova]$ ls
cubo
[samu@leibniz piante]$ mv ../castoro .
[samu@leibniz piante]$ ls
castoro  cubo
```

Figura: esempi del comando mv

La destinazione `.` indica la directory corrente.

Rinominare file

Il comando `mv` permette anche di rinominare i file:

```
[samu@leibniz piante]$ ls
castoro  cubo
[samu@leibniz piante]$ mv cubo triangolo
[samu@leibniz piante]$ ls
castoro  triangolo
```

Figura: esempi del comando mv

Now I am become death, the destroyer of files

Il comando più pericoloso in linux è indubbiamente `rm`.

`rm nome/file` elimina il file passato come argomento.

Non si può tornare indietro, una volta eliminato un file è perso per sempre!

La *shell* si aspetta che voi sappiate esattamente quello che state facendo e non si preoccupa se questo può distruggere il sistema.

Copiare una cartella

Per copiare le directory il comando `cp` deve funzionare in modalità *ricorsiva*, per permettere la copia di tutti gli elementi all'intero della directory.

Per copiare una cartella è quindi necessario aggiungere la *flag* `-r`

```
[samu@leibniz prova]$ ls
criceto  ligma  piante
[samu@leibniz prova]$ cp piante roveri
cp: -r not not specified; omitting directory 'piante'
[samu@leibniz prova]$ cp -r piante roveri
[samu@leibniz prova]$ ls
criceto  ligma  piante  roveri
```

Figura: copia di una cartella

Flags

Le *flag* sono un modo per estendere le funzionalità di un comando.

Vengono specificate dopo il comando e sono precedute da un trattino `-`

Per comodità sono di una sola lettera, ma in certi casi possono essere anche più verbose es. `--recursive`.

Si possono combinare più *flag* concatenando le lettere dopo il trattino: `-r -t` è equivalente a `-rt`

Man

Esiste un comando per leggere il *manuale* di un comando: `man`.

Sintassi: `man comando`

Aprirà un lettore di testo integrato nel terminale chiamato `less`.
Per navigare sono usati i seguenti comandi:

- `j`: Muoversi verso il basso
- `k`: Muoversi verso l'alto
- `g`: Inizio del file
- `G`: Fine del file
- `/name`: Cerca la stringa `name` in tutto il testo
- `n`: Selezionata una stringa va all'occorrenza successiva
- `q`: Esce e torna al promp

Less

Esiste un lettore di testo integrato nella *shell* chiamato `less`.

`less` è in realtà un comando che permette di leggere file di testo.

Sintassi: `less nome/file`

`less` è molto veloce a leggere file di testo di grandi dimensioni.

Operazioni testuali

Contare caratteri, linee, ecc. - wc

Esiste un comando per contare i caratteri, le linee e altre informazioni all'interno di un file di testo: `wc`

Sintassi: `wc nome/file`

Senza nessuna *flag* stampa:

- Il numero di righe
- Il numero di parole
- Il numero di bytes

Per stampare il numero di caratteri: `wc -c nome/file`

Ricerca di una stringa - grep

Per cercare una stringa in un file esiste `grep`

Sintassi: `grep "string" file`

Esiste la *flag* `-i` per la ricerca case-insensitive.

Uno dei comandi più potenti per la ricerca di pattern.

```
[samu@leibniz castorini]$ grep "13" LICENSE
 13. Use with the GNU Affreo General Pubic License.
Section 13, concerning interaction through a network
[samu@leibniz castorini]$
```

Figura: Ricerca di una stringa

Concatenazione o lettura? - cat

Il comando `cat` è nato per concatenare più file.

Per renderlo però completamente funzionante abbiamo bisogno dell'operatore di ridirezione che verrà spiegato più avanti.

Possiamo però usarlo per leggere file generalmente corti. Spesso è più rapido da usare di `less`.

A differenza di `less` stampa il file completo sul terminale.

Testa e coda - head, tail

Il comando `cat` prende uno o più file in input e li stampa sul terminale.

Esiste anche `head` che stampa solo le prime 10 righe di un file (si possono modificare con la *flag* `-n numero`).

Da notare che la *flag* precedente ha preso un parametro.

Esiste anche `tail` che fa esattamente la stessa cosa di `head`, ma partendo dalla fine del file.

Complementi di comandi base

Complementi di comandi base

Per pulire il terminale esiste il comando `clear`

Per resettare il terminale esiste il comando `reset`. Sarà molto utile quando lavorerete al progetto di programmazione e romperete tutto con la libreria grafica.

Per vedere i vecchi comandi eseguiti esite il comando `history`

Per riprendere un comando eseguito di recente basta utilizzare la freccetta verso l'alto.

Per editare un comando si possono usare le freccette verso destra e verso sinistra.

Copia, incolla e interruzione

Nel terminale non funziona il classico copia e incolla da tastiera eseguito con `ctrl + c` e `ctrl + v`. Queste combinazioni di tasti hanno il loro scopo e non sono fatti per copiare.

Per copiare e incollare dovete usare `shift + ctrl + c` e `shift + ctrl + v`.

`ctrl + c` serve per interrompere un processo in esecuzione.

Pipe e ridirezione: il potere della shell

Il potere della shell

Finora abbiamo visto soltanto comandi base e usati singolarmente.

Nonostante molti di essi siano utili anche da soli, sono sicuramente più utili usati in combinazione con altri comandi.

Come si fa però a combinare più comandi?

Il potere della shell

Finora abbiamo visto soltanto comandi base e usati singolarmente.

Nonostante molti di essi siano utili anche da soli, sono sicuramente più utili usati in combinazione con altri comandi.

Come si fa però a combinare più comandi? Ricordiamoci che:

- Un comando restituisce sempre qualcosa sullo standard output

Il potere della shell

Finora abbiamo visto soltanto comandi base e usati singolarmente.

Nonostante molti di essi siano utili anche da soli, sono sicuramente più utili usati in combinazione con altri comandi.

Come si fa però a combinare più comandi? Ricordiamoci che:

- Un comando restituisce sempre qualcosa sullo standard output
- Lo standard output è considerato come un file dal sistema

Il potere della shell

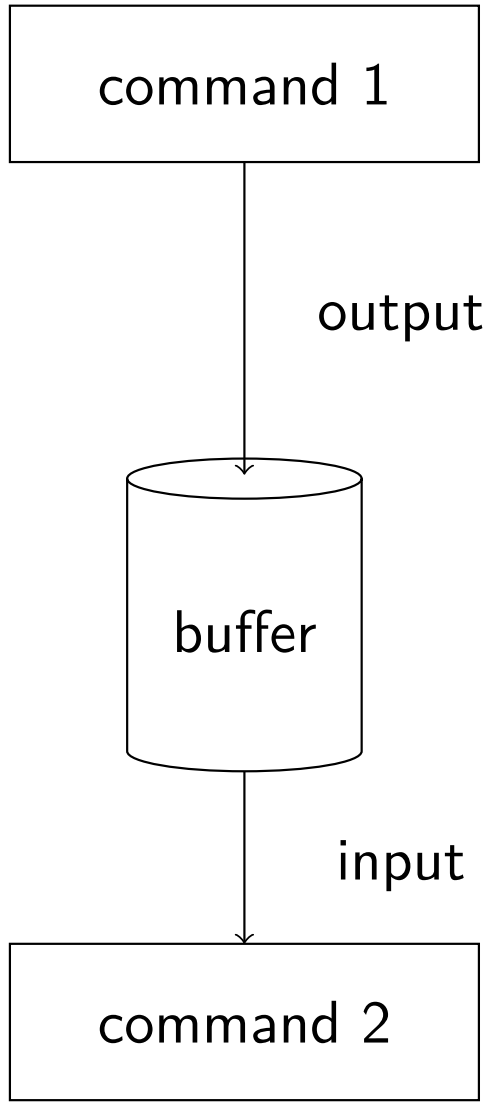
Finora abbiamo visto soltanto comandi base e usati singolarmente.

Nonostante molti di essi siano utili anche da soli, sono sicuramente più utili usati in combinazione con altri comandi.

Come si fa però a combinare più comandi? Ricordiamoci che:

- Un comando restituisce sempre qualcosa sullo standard output
- Lo standard output è considerato come un file dal sistema
- La maggior parte dei comandi visti fino ad adesso hanno la possibilità di prendere in input lo standard output invece che un file classico

Il potere della shell



Pipe

Per prendere l'output di un comando e riderizionarlo in input verso un altro comando si usa la **pipe** |

Per esempio se vogliamo vedere tutti i file presenti in `/bin` il nostro terminale si riempie di scritte.

Possiamo visualizzare il lungo output con il comando `less`: `ls /bin | less`

Pipe - grep

Come abbiamo visto la lista di file presenti in `/bin` è molto lunga.
Se volessimo trovarne uno specifico?

Pipe - grep

Come abbiamo visto la lista di file presenti in `/bin` è molto lunga. Se volessimo trovarne uno specifico?

Nonostante esiste un comando apposito per cercare file, possiamo fare: `ls /bin | grep "firefox"` dove al posto di *firefox* può andarci una qualsiasi stringa.

Pipe - esempi

Di seguito una lista di esempi di utilizzo della `pipe`:

- `cat file1 file2 | grep "word"` cerca una stringa in più file
- `ls /bin | wc -l` conta quanti programmi sono presenti in `/bin`
- `ls /bin | grep "zip" | wc -l` conta quanti programmi hanno la stringa "*zip*" al loro interno nella cartella `/bin`
- `grep "castoro" animali | wc -l` conta le occorrenze di `castoro` trovate nel file `animali`
- `grep "the" book | less` mostra le occorrenze di `the` trovate in `book` attraverso il lettore `less`

Ridirezione su file

Come abbiamo visto è possibile mandare l'output di un comando nell'input di un altro comando.

Se volessimo salvare l'output di un comando su un file?

Ridirezione su file

Come abbiamo visto è possibile mandare l'output di un comando nell'input di un altro comando.

Se volessimo salvare l'output di un comando su un file?
Esiste l'**operatore di ridirezione** >

Al posto di indirizzare l'output in un comando, scrive direttamente su un file.

Sintassi: `comando > file`

ATTENZIONE: Alla *shell* non interessa se il file esiste già, quindi se esiste lo SOVRASCRIVE COMPLETAMENTE.

Ridirezione su file non distruttiva

Esiste anche un operatore per indirizzare su file l'output di un comando senza sovrascrivere il contenuto del file, ma "appendendo" alla fine del file il contenuto scritto.

Sintassi: `comando >> file`

Si usa nello stesso modo dell'operatore classico

Wildcard

Wildcard

Nella *shell* l'asterisco `*` fa da "segnaposto" per una qualsiasi altra sequenza di caratteri.

Esempio: `torr*` si espande in: `torr`, `torra`, `torrb`, $\cdots$, `torraa`, `torrab`, `torrac`, $\cdots$

Valgono ovviamente anche i numeri e altri caratteri oltre alle lettere.

Questo permette di indicare più file con parti comuni nel nome.

Si possono combinare anche più asterischi: `c*a*` fa match con tutte le parole che iniziano per `c` e hanno almeno una `a` nel nome.

`*pila*` fa match con tutte le parole che hanno `pila` nel nome.

Root e permessi

Sistema multi-utente

Linux è un sistema *multi-utente*. Più persone possono usare lo stesso computer simultaneamente.

Questo utilizzo simultaneo non è da intendere con più schermi, tastiere e mouse attaccati allo stesso dispositivo fisico.

Si può utilizzare un computer connesso alla rete tramite il comando `ssh`.

`ssh` stabilisce una connessione cifrata tra l'utente e il computer e fornisce una *shell* su cui poter lavorare.

È quindi importante familiarizzare con la *shell* in quanto è l'unico modo per amministrare macchine e server remoti.

Sistema di permessi

La possibilità per Linux di gestire più utenti non è un aggiunta recedente. Il sistema è stato scritto proprio per supportare questa funzionalità.

Essendoci quindi più utenti è necessario avere un sistema di permessi adeguato.

Il sistema di permessi utilizzato da Linux non è semplice, e di seguito sarà data soltanto un'introduzione.

Root

In tutti i sistemi Linux esiste un unico utente che ha i permessi per seguire qualsiasi operazione: **root**

È l'amministratore del sistema.

NON deve essere MAI usato come utente se non per le operazioni strettamente necessarie.

Per aprire una *shell* come utente `root` è possibile digitare il comando: `su`

Il comando chiederà quindi la password di `root` (che generalmente è impostata durante l'installazione del sistema) e se corretta aprirà una *shell* con i privilegi di amministratore.

Permessi su un file

Dentro la cartella `/etc` esiste un file chiamato `shadow`. Se proviamo a leggerlo con `less` otteniamo: **/etc/shadow: Permission denied**

Questo significa che il nostro utente non ha i permessi per leggere il file. Ma come potevamo saperlo a priori senza tentare di leggerlo?

Permessi su un file

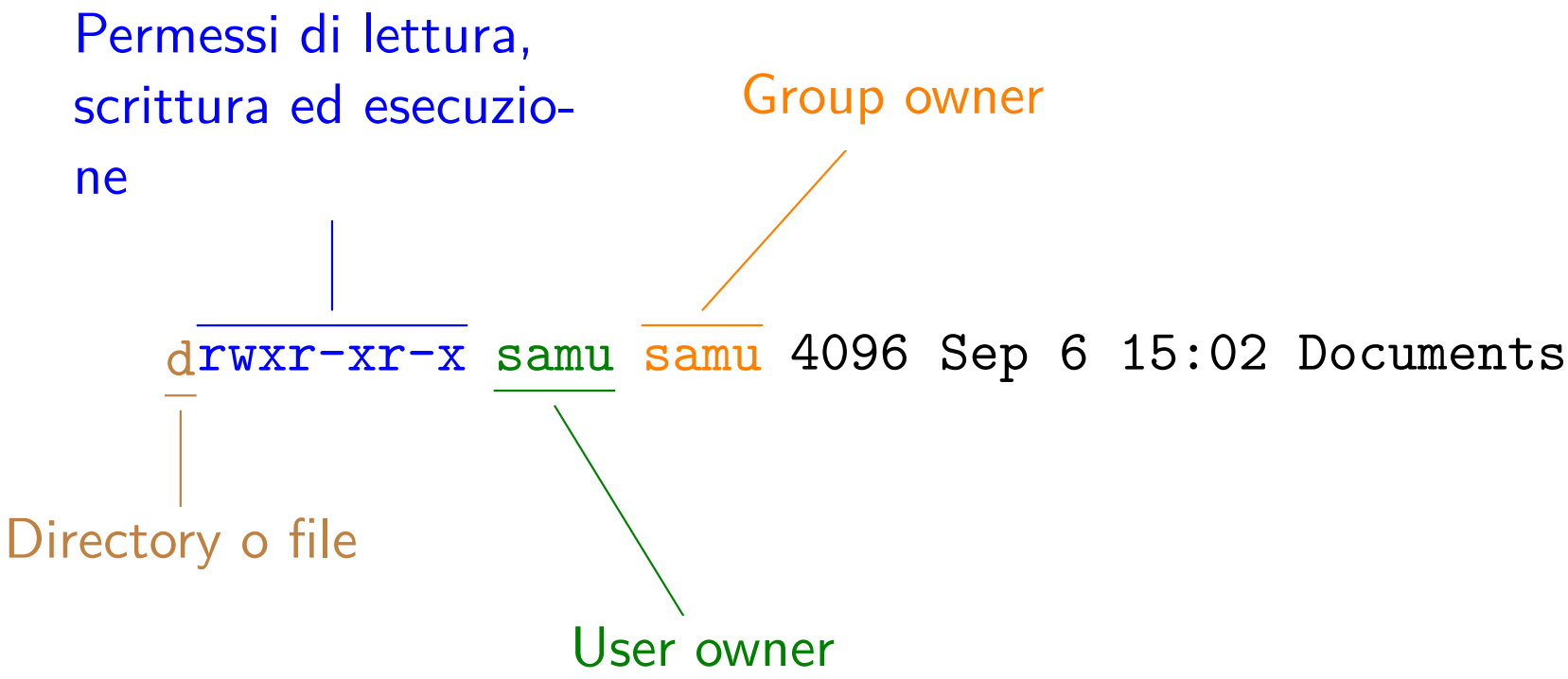
Dentro la cartella `/etc` esiste un file chiamato `shadow`. Se proviamo a leggerlo con `less` otteniamo: **/etc/shadow: Permission denied**

Questo significa che il nostro utente non ha i permessi per leggere il file. Ma come potevamo saperlo a priori senza tentare di leggerlo?

Una *flag* molto usata per il comando `ls` è `-l`.

`ls -l` permette di vedere molte più informazioni sui file presenti in una directory.

Esempio di ls -al



Read, write, execute

Ogni file ha un stringa formata da **9 bit** che determina quali permessi specifici ha quel file rispetto all'*utente*, il *gruppo* e gli *altri*.

I 9 bit sono suddivisi in **gruppi di 3**: il primo è specifico per l'*utente*, il secondo è specifico per il *gruppo* e i rimanenti sono per tutti gli *altri*.

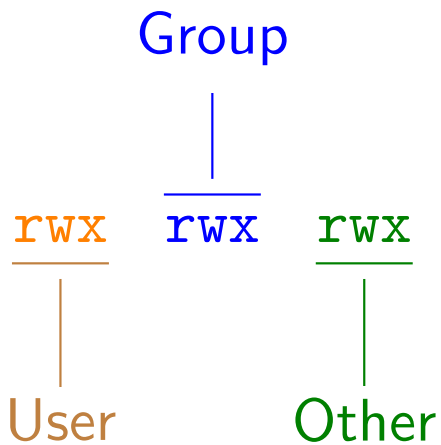


Tabella con i permessi

	File	Directory
r	Permette la lettura di un file	Permette di vedere il contenuto se anche `x` è segnato
w	Permette di scrivere sul file	Permette di creare ed eliminare file dentro la directory se `x` è segnato
x	Permette di eseguire un file	Permette di entrare nella directory

Tabella: Significato dei permessi per file e directory

Cambiare owner e group

Per cambiare l'owner di un file e il gruppo possiamo usare il comando `chown`.

Sintassi: `chown user:group file`

Per eseguire il comando sono necessari i privilegi di root.

Cambiare permessi wrx

Per cambiare i permessi lettura, scrittura ed esecuzione si utilizza il comando `chmod`.

Sintassi: `chmod [PART][ACTION][PERMISSION] file`

Al posto di `[PART]` è necessario specificare la parte che si vuole modificare:

- user: si utilizza `u`
- group: si utilizza `g`
- others: si utilizza `o`
- all: si utilizza `a`

Per eseguire il comando sono necessari i privilegi di root.

Cambiare permessi wrx

Per cambiare i permessi lettura, scrittura ed esecuzione si utilizza il comando `chmod`.

Sintassi: `chmod [PART][ACTION][PERMISSION] file`

Al posto di `[ACTION]` è necessario specificare la l'azione da compiere

- +: Aggiunge il permesso
- −: Rimuove il permesso
- =: Assegna esattamente quel permesso

Per eseguire il comando sono necessari i privilegi di root.

Cambiare permessi wrx

Per cambiare i permessi lettura, scrittura ed esecuzione si utilizza il comando `chmod`.

Sintassi: `chmod [PART][ACTION][PERMISSION] file`

Al posto di `[PERMISSION]` è necessario specificare la il permesso o i permessi da moficare:

- `r`: Read
- `w`: Write
- `x`: Execute

Per eseguire il comando sono necessari i privilegi di root.

Esempi `chmod`

- `chmod u+x pippo` Rende pippo un file eseguibile per l'utente
- `chmod o-w pippo` Rimuove la possibilità a tutti gli utenti diversi dall'owner del file e non appartenenti al gruppo del file di scrivere su pippo.
- `chmod g+r pippo` Rende pippo leggibile al gruppo
- `chmod g+x pippo` Rende pippo eseguibile dal gruppo
- `chmod u=rwx,g=,o= pippo` Rende pippo leggibile, scrivibile ed eseguibile per l'utente. Inoltre rimuove tutti i permessi dal gruppo e altri.

More work to do

More work to do

I comandi presentati sono soltanto una piccolissima parte dell'infinità di comandi presenti in una sistema Linux.

Non possiamo ovviamente includerli tutti, ma di seguito lasceremo alcuni comandi che potete approfondire:

- `ssh`
- `sudo`
- `apt`, `yum` e `pacman` (Dipende dalla distribuzione Linux)
- `top` e `htop`
- `kill`
- `touch`, `locate` e `find`
- `nano` e `vim`