

# Ambienti Operativi

SUPSI Dipartimento Tecnologie Innovative

Gianni Grasso

15 ottobre 2024

**Classe:** I1B

**Anno scolastico:** 2024/2025

## Indice

<b>1</b>	<b>Introduzione</b>	<b>3</b>
1.1	Tipi di file . . . . .	3
1.2	SSH . . . . .	3
1.3	Struttura di un filesystem . . . . .	3
<b>2</b>	<b>Bash</b>	<b>4</b>
2.1	Comandi . . . . .	4
2.2	Globbering . . . . .	5
2.3	Redirezione . . . . .	5
2.3.1	Pipe . . . . .	6
2.4	Esecuzione sequenziale e condizionale . . . . .	6
2.5	Gestione dei permessi . . . . .	7
2.5.1	Cambiare i permessi . . . . .	7
2.6	Elaborazione parallela . . . . .	8
2.6.1	Approccio manuale . . . . .	8
2.6.2	GNU Parallel . . . . .	8

# 1 Introduzione

Per riuscire a capire cos'è un ambiente operativo, dobbiamo innanzitutto riuscire a vedere il computer come uno strumento di elaborazione dei dati alla quale vengono passati degli input e date delle istruzioni. L'ambiente operativo è il luogo nella quale vengono date queste istruzioni.

È importante non fare confusione e non confondere il sistema operativo con l'ambiente operativo, il primo ha lo scopo di nascondere i meccanismi di gestione della macchina, rendendo l'utente in grado di poterla utilizzare senza conoscerne il funzionamento a basso livello, l'ambiente operativo invece fa da tramite tra gli utenti ed il sistema operativo e può essere visto come l'interfaccia nella quale si danno istruzioni alla macchina.

## 1.1 Tipi di file

Tutti i dati che si trovano su un computer sono rappresentati da una sequenza binaria, con **file binari** intendiamo che i dati non sono direttamente comprensibili da una persona mentre per con il termine **file personali** si intendono i file che possono essere compresi da una persona sotto forma di testo.

Per codificare i dati testuali si associa ogni carattere a una sequenza binaria, non esiste però un'unica codifica dei caratteri, di seguito sono riportati alcuni esempi:

- ASCII
- Windows code pages
- ISO 8859
- Unicode
- ...

Dobbiamo poi fare distinzione tra documenti testuali semplici e documenti strutturati, i primi sono quei documenti in cui la struttura logica non è facilmente distinguibile mentre nel secondo caso parliamo di documenti di testo in cui c'è una struttura che stabilisce il contenuto del file (ad esempio i file **csv**).

## 1.2 SSH

SSH, ovvero Secure Shell, è un protocollo di rete crittografico che ci consente di utilizzare servizi di rete in modo sicuro su una rete non protetta. Le sue applicazioni più comuni sono il login remoto e l'esecuzione da riga di comando, noi useremo il SSH per connetterci ad un server didattico.

Per connetterci al server da macOS/Linux dobbiamo digitare a terminale il seguente comando:

```
ssh linux1-didattica.supsi.ch -l nome.cognome@supsi.ch
```

## 1.3 Struttura di un filesystem

Il filesystem di un sistema ha una struttura ad albero, la radice è la cartella **root** (**/**), tutte le altre directory sono sottostanti ad essa.

Tra le directory principali che ci interessano ci sono **/home**, cartella che contiene le directory degli utenti locali, e **/tmp** cartella nella quale risiedono i file temporanei.

Un percorso può essere assoluto o relativo, nel primo caso specifichiamo l'intero percorso di una directory o un file, indipendentemente da dove ci troviamo, mentre nel secondo indichiamo il percorso per raggiungere un file a partire dalla posizione corrente.

## 2 Bash

La prima volta che apriremo un terminale potremo notare che il cursore è preceduto da: `utente@host:path$`, dove utente sta per il nome dell'utente connesso alla macchina, host il nome del server e path il percorso corrente.

È importante ricordare che in bash è tutto **case-sensitive**, sia i comandi che i nomi dei file e delle directory.

### 2.1 Comandi

Ecco una lista di comandi utili visti durante il corso:

- **history**

Stampa la cronologia dei comandi

```
history
```

`ctrl + r` per cercare

- **touch**

Aggiorna la data dell'ultima modifica del file, se il file non esiste ne crea uno

```
touch filename
```

- **seq**

Stampa una sequenza di numeri

```
# inizia da 1 e finisce a 100 (inrementa di 1 di default)
```

```
.
```

```
seq 1 100 > file.txt
```

```
cat file.txt
```

```
# 1
```

```
# 2
```

```
# ...
```

```
# 100
```

- **find**

Cerca dei file secondo i criteri imposti. Argomenti comuni

- `-name`: il nome dei file

- `-iname`: il nome dei file (non case sensitive)

- `-user`: il proprietario dei file

- `-size`: la dimensione dei file

- `-exec`: esegue un comando per ogni file trovato. `-exec rm {} \;`

```
# trova i file che finiscono con .jpg e li copia nella  
# cartella /destinazione
```

```
find / -name '*.jpg' -exec copy {} /destination \;
```

- **wc**

Conta le parole passate come input

```
# stampa il numero di linee di file.txt
```

```
cat file.txt | wc -l
```

- **tr**

Sostituisce una parola con un'altra

```
# sostituisce tutte le lettere `a` con `b`
```

```
cat file.txt | tr a b
```

- **cut**

Estrae porzioni di testo, utilizzando delimitatori o posizioni specifiche.

```
# estrae il secondo campo di testo dividendo il file a
ogni ';'
cat file.txt | cut -d ";" -f2
```

- **grep**

Filtra le righe di un file

```
# Visualizza le righe di file.txt che contengono la
parola 'ciao'
grep ciao file.txt
```

## 2.2 Globbing

Il Globbing consiste nell'utilizzare un pattern con uno o più caratteri "wildcard" per trovare o fare azioni sui file.

- \*, corrispondenza con zero o più caratteri qualsiasi
- ?, corrispondenza con esattamente un carattere
- [a-zA-Z9-0], corrispondenza tra un gruppo di caratteri
- [^abc], non-corrispondenza tra un gruppo di caratteri

Ad esempio per trovare tutti i file che iniziano con un numero e hanno un'estensione di tre caratteri:

```
find [0-9]*.???
```

Facciamo un esempio più complicato, il seguente codice trova tutti i file che hanno un nome che inizia con un numero compreso tra 10 e 20.

```
ls 1[0-9][^0-9]* 20[^0-9]*
```

Notiamo innanzitutto che il comando `ls` prende due intervalli, il primo trova i file che iniziano con il carattere 1 e che hanno come secondo carattere un numero da 0 a 9, il terzo carattere però non può essere un altro numero, in questo modo troviamo tutti i file che iniziano con un numero compreso tra 10 e 19. Il secondo intervallo invece trova i file che iniziano con il numero 20 (il terzo carattere) non può essere un numero.

## 2.3 Redirezione

Sui sistemi Unix i programmi hanno accesso a tre stream di input e output.

- **stdin** [0], input dalla tastiera

```
> cat > greetings.txt
```

- **stdout** [1], output su schermo

```
cat file > /dev/null
```

- **stderr** [2], output degli errori su schermo

```
cat nonexistentfile 2> errors.txt
```

**Nota:** Di default `1>` non include gli errori e sovrascrive il file di destinazione. Per fare l'append usare `>>`.

È anche possibile redirezionare `stdout` e `stderr` insieme, tuttavia non è buona pratica farlo in un file come nell'esempio sottostante:

```
cat file nonexistent >& results.txt
```

### 2.3.1 Pipe

La pipe è un buffer che permette di passare lo stdout di un comando come stdin di un altro, da sinistra a destra. Il seguente comando ad esempio prende l'input del comando `cat` e lo passa al comando `grep`:

```
cat A.txt | grep "ABC"
```

È possibile redirezionare lo stderr **nello** stdout, in questo caso possiamo usare `'2>&1'`:

```
ls nonesistente 2>&1 >/dev/null | grep impossibile
```

## 2.4 Esecuzione sequenziale e condizionale

È possibile eseguire sequenzialmente una serie di comandi inserendoli in un'unica linea di comando, ogni istruzione deve essere separata da un `;`. Ad esempio:

```
utente@host:~$ echo "ciao"; echo "mondo"; pwd
ciao
mondo
/home/utente
```

È anche possibile eseguire più comandi inserendoli sulla stessa linea di comando legandoli insieme con un'operazione booleana, **OR** `||` oppure **AND** `&&`. Ad esempio:

```
cat miofile || date || ls
```

L'esecuzione termina quando uno dei comandi (eseguiti da sinistra verso destra) termina correttamente senza errori.

```
cat miofile && date && ls
```

L'esecuzione termina quando uno dei comandi (eseguiti da sinistra verso destra) produce un errore.

## 2.5 Gestione dei permessi

Linux è un sistema operativo **multiutente**. Per autenticarsi c'è bisogno di un nome utente (che è univoco) e una password. La lista di tutti gli utenti è disponibile nel file `/etc/passwd`. L'utente `root` ha tutti i privilegi di sistema.

Gli utenti possono essere organizzati in gruppi, la lista di tutti i gruppi è disponibile nel file `/etc/groups`, di base ogni utente viene anche inserito in un gruppo con il suo nome. Ovviamente un utente può appartenere a più gruppi ma si è attivi in un unico gruppo alla volta.

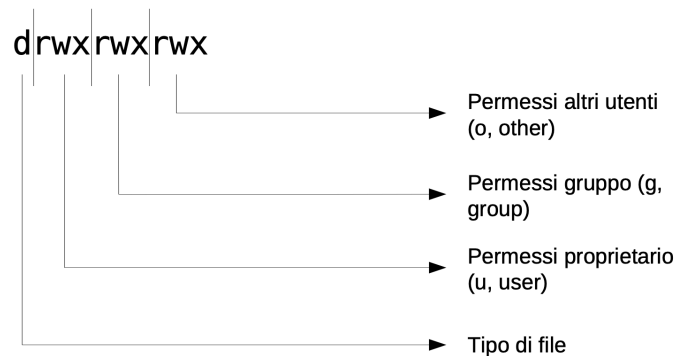


Figura 1: Struttura dei permessi

**Nota:** ci sono diversi tipi di file ma generalmente ci sarà `d` (directory) o `-` (file regolare).

Se l'immagine di prima rappresentava la struttura dei permessi, ora cerchiamo di descrivere i permessi di base e come si comportano su file e directory:

- **Lettura (r)**
  - *File:* permette di leggerne il contenuto
  - *Directory:* permette di elencarne il contenuto (file, sotto-directory)
- **Scrittura (w)**
  - *File:* permette di modificarne il contenuto
  - *Directory:* permette di aggiungere o rimuovere contenuto
    - \* posso cancellare un file solo se ho permessi sulla directory che lo contiene, non basta il permesso di scrittura sul file
- **esecuzione (x)**
  - *File:* permette di eseguirli (su un file `.txt` ad esempio è inutile)
  - *Directory:* permette di attraversarle per accedere a file e sotto-directory, non permette di elencare il contenuto

### 2.5.1 Cambiare i permessi

Ci sono due modi per cambiare i permessi:

- Modalità simbolica, `chmod [-R] [ugo] [+ -=] [rwxst] files`
- Modalità ottale, `chmod [-R] [N] [N] [N] [N] files`

**Nota:** `ugo` indica (user, group, others)

**Nota:** nella modalità ottale i permessi vengono rappresentati con dei bit, in generale `r` vale 4, `w` 2 e `x` 1

È inoltre possibile cambiare il proprietario di un file con `chown` e cambiare il gruppo di un file con `chgrp`

## 2.6 Elaborazione parallela

Per visualizzare ed elencare i processi presenti sul sistema possiamo usare:

- `ps`
- `top`
- `htop`

Le informazioni sulla cpu a disposizione sono presenti nel file `/proc/cpuinfo`.

Per vedere le informazioni relative al tempo di un comando possiamo usare il comando `time` prima dell'effettivo comando.

### 2.6.1 Approccio manuale

Con il comando `split` possiamo dividere un file in  $n$  file in base al numero di linee che esso contiene. Ad esempio:

```
$ wc -l file.tsv
9804311 file.tsv #Il numero di righe del file

$ split -l 4902155 file.tsv #Separa il file ogni 4902155 righe (
  la metà), verranno creati due file
```

### 2.6.2 GNU Parallel

Il comando `parallel` ci permette di eseguire un comando per ogni riga di un file e lo fa in modo parallelo. Funziona in modo simile al comando `exec` con `find`.

Il comando `parallel` riceve lo `stdin` e lo passa al comando successivo, è utile con comandi che funzionano con lo `stdin`, come ad esempio `rm` o `touch`. Facciamo qualche esempio:

```
cat lista | parallel rm {}

cat lista | parallel echo {} "Riga"
```

Si può anche specificare quante righe alla volta usare come parametro, ad esempio:

```
cat lista | parallel -n 2 echo "Ciao {1} mondo {2}"
```

Infine, possiamo usare il comando `--pipe` per passare le righe tramite pipe. Questo è utile solo quando vogliamo eseguire operazioni con più righe alla volta:

```
cat lista | parallel --pipe -n2 wc -l
```