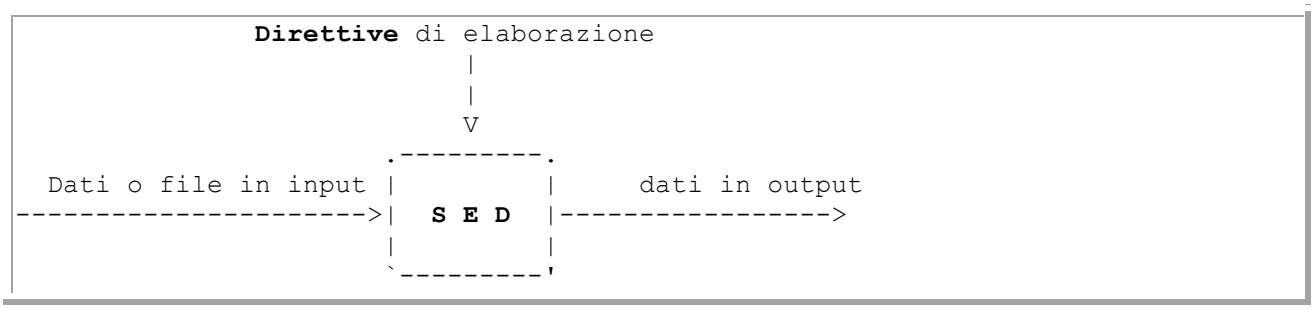
I FILTRI SED, GREP e FIND

SED

SED è un programma in grado di **eseguire delle trasformazioni elementari in un flusso di dati di ingresso,** proveniente indifferentemente da un file o da una pipeline. Questo flusso di dati viene letto sequenzialmente e la sua trasformazione viene restituita attraverso lo standard output.

Il nome è l'abbreviazione di **Stream Editor**. Volendo usare altri termini, lo si potrebbe definire come un programma per la modifica sequenziale di un flusso di dati espressi in forma testuale.

Volendo vedere SED come una scatola nera, lo si può immaginare come un oggetto che ha due ingressi: un flusso di dati in ingresso, composto da uno o più file di testo concatenati assieme; un flusso di istruzioni in ingresso, che compone il programma dell'elaborazione da apportare ai dati; un flusso di dati in uscita che rappresenta il risultato dell'elaborazione.



Il comando **sed**, filtra un file di input (specificato alla fine o lo standard input del comando precedentemente digitato) con un file di filtraggio (espressione regolare) fornito come argomento della riga di comando dopo le opzioni.

La sintassi è la seguente:

sed [***opzioni***] [***Direttive di elaborazione***] [***file di input...***]

Le direttive di elaborazione sono composte da un'**espressione regolare + comando.**
Il testo del file di filtraggio (o direttive di elaborazione) può essere indicato attraverso delle opzioni adatte (–e, -f), oppure, in loro mancanza, può essere indicato come primo degli argomenti che seguono le opzioni.
Alla fine possono essere indicati i file da elaborare, e in loro mancanza si usa lo standard input.

Alcune opzioni

-e '*istruzioni*'
--expression='*istruzioni*'
Questa opzione, che può essere utilizzata anche più volte, permette di specificare delle istruzioni SED che si aggiungono alle altre eventualmente già indicate.

-f '*file_delle_istruzioni*'
--file '*file_delle_istruzioni*'
Questa opzione permette di indicare un file contenente una serie di istruzioni SED. Anche questa opzione può essere usata più volte, aggiungendo ogni volta altre istruzioni al programma globale.

-n | --quiet | --silent

```
In condizioni normali, alla fine di ogni ciclo, SED emette il contenuto di
quello che viene definito come pattern space. In pratica, ogni riga letta
ed elaborata viene emessa attraverso lo standard output senza bisogno di
un comando apposito. Utilizzando questa opzione, si fa in modo di evitare
questo comportamento, così che il programma di elaborazione interpretato
da SED deve ordinare quando emettere ogni riga.
```

Il primo compito di SED è quello di raccogliere tutto ciò che deve andare a comporre il programma di elaborazione (o file di filtraggio) : può trattarsi di direttive fornite singolarmente attraverso l'opzione `-e` e di gruppi di direttive fornite all'interno di file appositi, indicati attraverso l'opzione `-f`. In particolare, SED si prende cura di mantenerne intatto l'ordine. Successivamente, concatena i dati in ingresso secondo la sequenza indicata del/i file posto/i alla fine della riga di comando, oppure utilizza direttamente lo standard input.

Di tutte le operazioni a disposizione di sed, vengono focalizzate, in primo luogo, le tre più comunemente usate. Esse sono **p**rint (visualizza allo stdout), **d**elete (cancella) e **s**ubstitute (sostituisce).

Operatori sed di base

```
Indirizzo = numero di riga
```

Operatore	Nome	Effetto
`[indirizzo]/p`	print	Visualizza [l'indirizzo specificato]
`[indirizzo]/d`	delete	Cancella [l'indirizzo specificato]
`s/esp.reg1/esp.reg2/`	substitute	Sostituisce in ogni riga la prima occorrenza della stringa esp.reg1 con la stringa esp.reg2
`[indirizzo]/s/esp.reg1/esp.reg2/`	substitute	Sostituisce, in tutte le righe specificate in *`indirizzo`*, la prima occorrenza della stringa `esp.reg1` con la stringa `esp.reg2`
`[indirizzo]/y/esp.reg1/esp.reg2/`	transform	sostituisce tutti i caratteri della stringa `esp.reg1` con i corrispondenti caratteri della stringa `esp.reg2`, in tutte le righe specificate da *`indirizzo`*
`g`	global	Agisce su *tutte* le verifiche d'occorrenza di ogni riga di input controllata Se l'operatore `g` (*global*) non è accodato al comando *substitute*, la sostituzione agisce solo sulla prima verifica d'occorrenza di ogni riga.

Le espressioni regolari utilizzabili:

POSIX	**GNU**	Descrizione
\	\	Escape.
^	^	Inizio riga.
$	$	Fine riga.
	\	Alternativa.
	\(\)	Raggruppamento.
\ *n*	\ *n*	Riferimento.

POSIX	**GNU**	Descrizione
x *	*x* *	Zero o più caratteri qualsiasi.
	x \?	Zero o al massimo un carattere qualsiasi.
	x \+	Uno o più caratteri qualsiasi
x \{ *n* \}	*x* \{ *n* \}	Esattamente *n* volte *x*.
x \{ *n* ,\}	*x* \{ *n* ,\}	Almeno *n* volte *x*.
x \{ *n* , *m* \}	*x* \{ *n* , *m* \}	Da *n* a *m* volte *x*.
x \{, *m* \}	*x* \{, *m* \}	Da zero a *m* volte *x*.
[ ]	[ ]	Elenco.
x y ...	*x y* ...	Sequenze.
x - *y*	*x* - *y*	Intervalli.
[. .]		Elementi di collazione.
[= =]		Caratteri equivalenti.
[: :]	[: :]	Classi di caratteri.

Esempi di operatori sed

Notazione	Effetto
`8d`	Cancella l'ottava riga dell'input.
`/^$/d`	Cancella tutte le righe vuote.
`1,/^$/d`	Cancella dall'inizio dell'input fino alla prima riga vuota compresa.
`/Jones/p`	Visualizza solo le righe in cui è presente "Jones" (con l'opzione -n).
`s/Windows/Linux/`	Sostituisce con "Linux" la prima occorrenza di "Windows" trovata in ogni riga dell'input.
`s/BSOD/stabilità/g`	Sostituisce con "stabilità" tutte le occorrenze di "BSOD" trovate in ogni riga dell'input.
`s/ *$//`	Cancella tutti gli spazi che si trovano alla fine di ogni riga.
`s/00*/0/g`	Riduce ogni sequenza consecutiva di zeri ad un unico zero.
`/GUI/d`	Cancella tutte le righe in cui è presente "GUI".
`s/GUI//g`	Cancella tutte le occorrenze di "GUI", lasciando inalterata la parte restante di ciascuna riga.

Ogni direttiva di un programma di elaborazione SED fa riferimento, esplicitamente o implicitamente, a un gruppo di righe, identificate in qualche modo, a cui vengono applicati dei comandi.

`['`*`selezione_righe`*`']`*`/comando/`*

Il modello sintattico mostra l'indicazione di un comando dopo la selezione delle righe; questo comando può essere un raggruppamento di comandi, indicato all'interno di parentesi graffe.

La selezione delle righe per una direttiva SED è il primo elemento importante. La mancanza dell'indicazione di questa selezione rappresenta implicitamente la selezione di tutte le righe.

È importante osservare che le righe possono essere indicate anche attraverso la corrispondenza con un'espressione regolare, che comunque non deve essere confusa con i comandi che a loro volta possono avere a che fare con altre espressioni regolari.

Inoltre, è necessario ricordare che SED numera le righe a partire dalla prima del primo file, continuando fino alla fine dell'ultimo file, senza interrompere la numerazione.

ESEMPI:

```
$ sed "" prova.txt
```

Legge il file `prova.txt` e lo riemette tale e quale, dal momento che non è stato specificata alcuna direttiva per il programma di elaborazione.

```
$ sed -n 'p' prova.txt
```

Si ottiene lo stesso risultato dell'esempio precedente, perché prima viene usata l'opzione `-n` con cui si inibisce la riemissione predefinita delle righe lette, ma poi si specifica una direttiva contenente il comando `p` applicato a tutte le righe del flusso in ingresso.

```
$ sed -e '/^$/d' $nomefile
```

L'opzione -e indica che la stringa successiva deve essere interpretata come un'istruzione di editing. (se a sed viene passata un'unica istruzione, "-e" è facoltativo.)
Il quoting "forte" (') protegge i caratteri speciali delle Esp.Regolari, presenti nell'istruzione, dalla reinterpretazione da parte dello script. (Questo riserva solo a sed l'espansione delle Espressioni Regolari). Agisce sul testo del file $nomefile.
Sed utilizza l'opzione -e per indicare che la stringa che segue è un'istruzione, o una serie di istruzioni. Se la stringa contiene una singola istruzione, allora questa opzione può essere omessa.

```
sed -n '/xzy/p' $nomefile
```
L'opzione -n indica a sed di visualizzare solo quelle righe che verificano il l'espressione regolare. Altrimenti verrebbero visualizzate tutte le righe dell'input. L'opzione -e, in questo caso, non è necessaria perché vi è una sola istruzione di editing.

```
$ sed -n '1,10/p' prova.txt
```
Emette solo le prime 10 righe del file.

```
$ sed '/.\{81,\}/d' prova.txt
```
Elimina le righe più lunghe di 80 caratteri.

```
$ sed '/^$/d' prova.txt
```
Elimina tutte le righe vuote.

```
$ sed '/^---INIZIO---$/,/^---FINE---$/d' prova.txt
```
Elimina tutte le righe comprese negli intervalli delimitati da righe contenenti esclusivamente la stringa `---INIZIO---` e `---FINE---`.

```
$ sed -n '/^---INIZIO---$/,/^---FINE---$/p' prova.txt
```
Emette tutte le righe comprese negli intervalli delimitati da righe contenenti esclusivamente la stringa `---INIZIO---` e `---FINE---`.

Sostituzione del contenuto delle righe

```
$ sed 's/andato/venuto/' prova.txt
```
Sostituisce in ogni riga la prima occorrenza della stringhe «andato» con la stringa «venuto».

```
$ sed 's/andato/venuto/g' prova.txt
```
Sostituisce tutte le occorrenze della stringa «andato» con la stringa «venuto».

```
$ sed 's/^/    /' prova.txt
```
Aggiunge quattro spazi all'inizio di ogni riga del file.

```
$ sed 's/\(.*\):\(.*\):\(.*\):\(.*\):\(.*\):\(.*\):\(.*\)/\1:\3/' /etc/passwd
```
Seleziona solo il primo e il terzo campo del file `/etc/passwd`; in pratica, preleva il nominativo e il numero UID.

Raggruppamenti

```
#!/bin/sed -nf
{
p
w registro
}
```

L'esempio mostra l'unione di due comandi riferiti allo stesso gruppo di righe (tutte). Lo scopo è quello di emettere le righe attraverso lo standard output e di annotarle anche in un file denominato `registro`.

> Si osservi il fatto che la parentesi graffa di chiusura deve essere indicata da sola, come si vede nell'esempio, e di conseguenza può essere opportuno fare altrettanto per quella di apertura.

```
$ sed -n -e 'p' -e 'w registro' prova.txt
```

Questo esempio fa la stessa cosa di quello precedente, con la differenza che i comandi sono stati separati in due direttive riferite allo stesso gruppo di righe, e inoltre si elaborano le righe del file `prova.txt`.

Grep

Il programma **Grep** esegue una **ricerca all'interno dei file** in base a un modello espresso normalmente in forma di espressione regolare.

Storicamente sono esistite tre versioni di questo programma: `grep`, `egrep` e `fgrep`, ognuna specializzata in un tipo di ricerca. Attualmente, Grep GNU, corrispondente a quello che si utilizza normalmente con GNU/Linux, comprende tutte e tre queste funzionalità. In alcuni casi, per mantenere la compatibilità con il passato, possono trovarsi distribuzioni che mettono a disposizione anche i programmi `egrep` e `fgrep` in forma originale.

63.1.1 Avvio di grep

```
grep [opzioni] modello [file...]
grep [opzioni] -e modello [file...]
grep [opzioni] -f file_modello [file...]
```

`grep` esegue una ricerca all'interno dei file indicati come argomento oppure all'interno dello standard input. Il modello di ricerca può essere semplicemente il primo degli argomenti che seguono le opzioni, oppure può essere indicato precisamente come argomento dell'opzione `-e`, oppure ancora può essere contenuto in un file che viene indicato attraverso l'opzione `-f`.

La tabella 63.1 elenca le opzioni principali.

Opzione	Descrizione
-G	Utilizza un'espressione regolare elementare (comportamento predefinito).
-E	Utilizza un'espressione regolare estesa.
-F	Utilizza un modello fatto di stringhe fisse.
-e *modello*	Specifica il modello di ricerca.

Opzione	Descrizione
-f *file*	Specifica il nome di un file contenente il modello di ricerca.
-i	Ignora la differenza tra maiuscole e minuscole.
-n	Aggiunge il numero di riga.
-c	Emette solo il totale delle righe corrispondenti per ogni file.
-h	Elimina l'intestazione normale per le ricerche su più file.
-l	Emette solo i nomi dei file per i quali la ricerca ha avuto successo.
-L	Emette solo i nomi dei file per i quali la ricerca non ha avuto successo.
-v	Inverte il senso della ricerca: valgono le righe che non corrispondono.

Tabella 63.1. Opzioni principali di `grep`.

63.1.2 Espressioni regolari

Un'espressione regolare è un modello che descrive un insieme di stringhe. Le espressioni regolari sono costruite, in maniera analoga alle espressioni matematiche, combinando espressioni più brevi.

Per tradizione esistono due tipi di espressioni regolari: elementari, o BRE, ed estese, o ERE (si vedano in particolare i capitoli dedicati alle espressioni regolari in generale: 193 e 194). Il programma `grep` originale era in grado di interpretare solo quelle elementari, Grep GNU interpreta correttamente anche quelle estese anche se il suo funzionamento predefinito è sempre basato su quelle elementari. Le espressioni regolari elementari sono limitate rispetto a quelle estese; in questo particolare adattamento di `grep` si ottengono le stesse funzionalità, pur se con una sintassi leggermente diversa da quella normale.

In generale, la sintassi delle espressioni regolari non è uguale per tutti i programmi che ne fanno uso, anche se esiste lo standard POSIX, che però è difficile trovare applicato in modo fedele.

Senza costringere il lettore a studiare subito i capitoli 193 e 194, dedicati alle espressioni regolari, viene data una descrizione sommaria delle espressioni regolari estese secondo GNU, e alla fine viene descritto come si comportano quelle elementari.

Il punto di partenza sono le espressioni regolari con le quali si ottiene una corrispondenza con un solo carattere. La maggior parte dei caratteri, includendo tutte le lettere e le cifre numeriche, sono espressioni regolari che corrispondono a loro stessi. Ogni metacarattere con significati speciali può essere utilizzato per il suo valore normale facendolo precedere dalla barra obliqua inversa (`\`).

Una fila di caratteri racchiusa tra parentesi quadre corrisponde a un carattere qualunque tra quelli indicati; se all'inizio di questa fila c'è l'accento circonflesso, si ottiene una corrispondenza con un carattere qualunque diverso da quelli della fila. Per esempio, l'espressione regolare `[0123456789]` corrisponde a una qualunque cifra numerica, mentre `[^0123456789]` corrisponde a un carattere qualunque purché non sia una cifra numerica.

All'interno delle parentesi quadre, invece che indicare un insieme di caratteri, è possibile indicarne un intervallo, mettendo il carattere iniziale e finale separati da un trattino (`-`). I caratteri che vengono rappresentati in questo modo dipendono dalla codifica che ne determina la sequenza. Per

esempio, in base alla codifica ASCII, l'espressione regolare `[9-A]` rappresenta un carattere qualsiasi tra: `9`, `:`, `;`, <, =, >, `?`, `@` e `A`.

All'interno delle parentesi quadre si possono indicare delle classi di caratteri attraverso il loro nome: `[:alnum:]`, `[:alpha:]`, `[:cntrl:]`, `[:digit:]`, `[:graph:]`, `[:lower:]`, `[:print:]`, `[:punct:]`, `[:space:]`, `[:upper:]` e `[:xdigit:]`. Per essere usati, questi nomi di classi possono solo apparire all'interno di un'espressione tra parentesi quadre, di conseguenza, per esprimere la corrispondenza con un qualunque carattere alfanumerico si può utilizzare l'espressione regolare `[[:alnum:]]`. Nello stesso modo, per esprimere la corrispondenza con un qualunque carattere non alfanumerico si può utilizzare l'espressione regolare `[^[:alnum:]]`.

Il punto (`.`) corrisponde a un qualsiasi carattere. Il simbolo `\w` è un sinonimo di `[[:alnum:]]` e `\W` è un sinonimo di `[^[:alnum:]]`.

L'accento circonflesso (`^`) e il dollaro (`$`) sono metacaratteri che corrispondono rispettivamente alla stringa nulla all'inizio e alla fine di una riga. I simboli `\<` e `\>` corrispondono rispettivamente alla stringa vuota all'inizio e alla fine di una parola.

Quando per qualche ragione si hanno difficoltà a indicare dei caratteri che per l'espressione sarebbero comunque caratteri normali, è possibile utilizzare il simbolo `\` anteriormente, purché questo non rappresenti a sua volta un altro metacarattere.

Un'espressione regolare che genera una corrispondenza con un carattere singolo, può essere seguita da uno o più operatori di ripetizione. Questi sono rappresentati attraverso i simboli `?`, `*`, `+` e dai «contenitori» rappresentati da particolari espressioni tra parentesi graffe. La tabella 63.2 mostra l'uso che si può fare di questi operatori.

Codifica	Corrispondenza.
x *	Nessuna o più volte *x*. Equivalente a □*x*□`{0,}`.
x ?	Nessuna o al massimo una volta *x*. Equivalente a □*x*□`{0,1}`.
x +	Una o più volte *x*. Equivalente a □*x*□`{1,}`.
x { *n* }	Esattamente *n* volte *x*.
x { *n* ,}	Almeno *n* volte *x*.
x { *n* , *m* }	Da *n* a *m* volte *x*.

Tabella 63.2. Operatori di ripetizione nelle espressioni regolari estese gestite da Grep GNU.

Due espressioni regolari possono essere concatenate (in sequenza) generando un'espressione regolare corrispondente alla sequenza di due sottostringhe che rispettivamente corrispondono alle due sottoespressioni.

Due espressioni regolari possono essere unite attraverso l'operatore `|`; l'espressione regolare risultante corrisponde a una qualunque stringa per la quale sia valida la corrispondenza di una delle due sottoespressioni.

La ripetizione (attraverso gli operatori di ripetizione) ha la precedenza sul concatenamento che a sua volta ha la precedenza sull'alternanza (quella che si ha utilizzando l'operatore `|`). Una sottoespressione può essere racchiusa tra parentesi tonde per modificare queste regole di precedenza.

La notazione \□*n*□, dove *n* è una singola cifra numerica diversa da zero, rappresenta un riferimento all'indietro a una corrispondenza già avvenuta tra quelle di una sottoespressione precedente racchiusa tra parentesi tonde. La cifra numerica indica l'*n*-esima sottoespressione tra parentesi a partire da sinistra.

Nelle espressioni regolari elementari, i metacaratteri `?`, `+`, `{`, `|`, `(` e `)` perdono il loro significato speciale. Al loro posto si possono utilizzare gli stessi simboli preceduti dalla barra obliqua inversa: `\?`, `\+`, `\{`, `\`, `\(` e `\)`.[(1)]

63.1.3 egrep

Come già accennato, alcune distribuzioni GNU/Linux forniscono un programma `egrep` alternativo, invece di utilizzare il solito collegamento allo stesso `grep`. In tal caso, per quanto riguarda l'interpretazione delle espressioni regolari (estese), la parentesi graffa aperta che inizia la delimitazione di un «contenitore» per rappresentare un operatore di ripetizione, viene indicata come `\{`.

63.1.4 Esempi

```
$ grep -F -e ciao -i -n *
```

Cerca all'interno di tutti i file contenuti nella directory corrente la corrispondenza della parola `ciao` senza considerare la differenza tra le lettere maiuscole e quelle minuscole. Visualizza il numero e il contenuto delle righe che contengono la parola cercata.

```
$ grep -E -e "scal[oa]" elenco
```

Cerca all'interno del file `elenco` le righe contenenti la parola `scalo` o `scala`.

```
$ grep -E -e \`.*\' elenco
```

Questo è un caso di ricerca particolare in cui si vogliono cercare le righe in cui appare qualcosa racchiuso tra apici singoli, nel modo `` `...' ``. Si immagina però di utilizzare la shell Bash con la quale è necessario proteggere gli apici da un altro tipo di interpretazione. In questo caso la shell fornisce a `grep` solo la stringa `` `.*' ``.

```
$ grep -E -e "\`.*\'" elenco
```

Questo esempio deriva dal precedente. Anche in questo caso si suppone di utilizzare la shell Bash, ma questa volta viene fornita a `grep` la stringa `` `.*\' `` che fortunatamente viene interpretata ugualmente da `grep` nel modo corretto.

63.1.5 zgrep

```
zgrep [□opzioni□] □modello□ [□file□...]
```

`zgrep` è un programma aggiuntivo che permette di eseguire delle ricerche all'interno di file compressi (con `compress`, oppure con `gzip`). `zgrep` si occupa semplicemente di decomprimere i file, se necessario, prima di passarli a `grep`. In questo senso, e considerato che le opzioni e il modello sono passati tali e quali a `grep`, la sintassi è la stessa di quel programma.

Ricapitolando, `zgrep` può essere usato indifferentemente con file normali o compressi, senza che l'utente debba preoccuparsi di questo.

63.2 find

Il programma `find` esegue una ricerca, all'interno di uno o più percorsi, per i file che soddisfano delle condizioni determinate, legate alla loro apparenza esterna e non al loro contenuto. Per ogni file o directory trovati, può essere eseguito un comando (programma, script o altro) che a sua volta può svolgere delle operazioni su di essi.

Questa sezione non descrive tutte le funzionalità di `find`. Una volta appresi i rudimenti del suo funzionamento, conviene consultare *find.info* oppure *find*(1).

63.2.1 Avvio di find

La sintassi di `find` è piuttosto insolita, oltre che complessa, anche se dallo schema seguente non sembrerebbe così.

In particolare è indispensabile tenere a mente che molti dei simboli utilizzati negli argomenti di `find` potrebbero essere interpretati e trasformati dalla shell, di conseguenza occorrerà utilizzare le tecniche che la shell stessa offre per evitarlo.

`find [ percorso ...] [ espressione ]`

`find` esegue una ricerca all'interno dei percorsi indicati per i file che soddisfano l'espressione di ricerca. Il primo argomento che inizia con `-`, `(`, `)`, `,` o `!` (trattino, parentesi tonda, virgola, punto esclamativo) viene considerato come l'inizio dell'espressione, mentre gli argomenti precedenti sono interpretati come parte dell'insieme dei percorsi di ricerca.

Se non vengono specificati percorsi di ricerca, si intende la directory corrente; se non viene specificata alcuna espressione, o semplicemente se non viene specificato nulla in contrario, viene emesso l'elenco dei nomi trovati.

63.2.2 Espressioni di find

Il concetto di espressione nella documentazione di `find` è piuttosto ampio e bisogna fare un po' di attenzione. Si può scomporre idealmente in

`[ opzione ...] [ condizioni ]`

e a sua volta le condizioni possono essere di due tipi: test e azioni. Ma, mentre le opzioni devono apparire prima, test e azioni possono essere mescolati tra loro.

Le opzioni rappresentano un modo di configurare il funzionamento del programma, così come di solito accade nei programmi di servizio. Le condizioni sono espressioni che generano un risultato

logico e come tali vanno trattate: per concatenare insieme più condizioni occorre utilizzare gli operatori booleani.

63.2.3 Alcune opzioni

Come già accennato, dopo l'indicazione dei percorsi e prima delle condizioni (test e azioni) vanno indicate le opzioni. Quelle che seguono sono solo le più importanti tra quelle a disposizione.

`-depth`

Elabora prima il contenuto delle directory. In pratica si ottiene una scansione che parte dal livello più profondo fino al più esterno.

`-xdev | -mount`

Non esegue la ricerca nelle directory contenute all'interno di file system differenti da quello di partenza. Tra i due è preferibile usare `-xdev`.

`-noleaf`

Non ottimizza la ricerca. Questa opzione è necessaria quando si effettuano ricerche all'interno di file system che non seguono le convenzioni Unix, come nel caso di CD-ROM senza le estensioni necessario, o partizioni Dos.

Esempi

```
$ find . -xdev -print
```

Elenca tutti i file e le directory a partire dalla posizione corrente restando nell'ambito del file system di partenza.